MySQL™

ORACLE®

MySQL Performance & Reporting

MySQL Performance Tuning for Embedded Systems

- Query tuning is the most critical part of MySQL performance tuning
- Can give 100% to 1000% (or more!) performance improvement
- Critical for reporting and embedded systems
  - Reporting queries are usually complex
  - Embedded systems runs “as is”, sometimes without a DBA

Agenda

- Understanding Query Performance
- Indexes and Temporary Tables in MySQL
- GROUP BY / ORDER BY Optimizations
- Subqueries
- Reporting Queries and Summary Tables
- Full Text Search Queries
- What's New in MySQL 5.6 Beta

How to Deal with Slow Performance

- Find slow queries
- Profile/Explain
- Fix queries
  - Add indexes
  - Re-write queries
  - Add summary tables
  - Cache queries
- = Better performance!

Main Query Performance Problems

- Full table scans (no index)
- Temporary tables
- Filesort

Using EXPLAIN to Analyze Queries

- Find out how the query optimizer would improve a **SELECT** query

```
mysql> EXPLAIN select * from City where Name = 'London'\G
           id: 1
  select_type: SIMPLE
        table: City
         type: ALL
possible_keys: NULL
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 4079
        Extra: Using where
```

Query Profiling

```
mysql> set profiling =1;
mysql> show profiles\G

******************** 1. row ***************************
Query_ID: 4
Duration: 0.00069300
   Query: select * from sbtest where k >0 order by
pad desc limit 10
******************** 2. row ***************************
Query_ID: 5
Duration: 0.00044800
   Query: explain select * from sbtest where k >0
order by pad desc limit 10
```

Using Query Profiling

```
mysql> set profiling =1;
mysql> select pad, c, count(*) from sbtest where k = 0
  group  by pad, c limit 10;
... (22.90 sec)
mysql> show profile;
+----------------------+-----------+
| Status               | Duration  |
+----------------------+-----------+
| starting             |  0.000017 |
| init                 |  0.000086 |
| checking permissions |  0.000016 |
| Opening tables       |  0.000037 |
| System lock          |  0.000021 |
| init                 |  0.000034 |
| optimizing           |  0.000014 |
| statistics           |  0.000126 |
| preparing            |  0.000027 |
| Creating tmp table   |  0.000420 |
| executing            |  0.000005 |
| Copying to tmp table | 22.904328 |
| Sorting result       |  0.000053 |
| Sending data         |  0.000028 |
| end                  |  0.000005 |
| removing tmp table   |  0.000210 |
| end                  |  0.000008 |
| query end            |  0.000008 |
| closing tables       |  0.000018 |
| freeing items        |  0.000030 |
| logging slow query   |  0.000005 |
| cleaning up          |  0.000015 |
+----------------------+-----------+
```

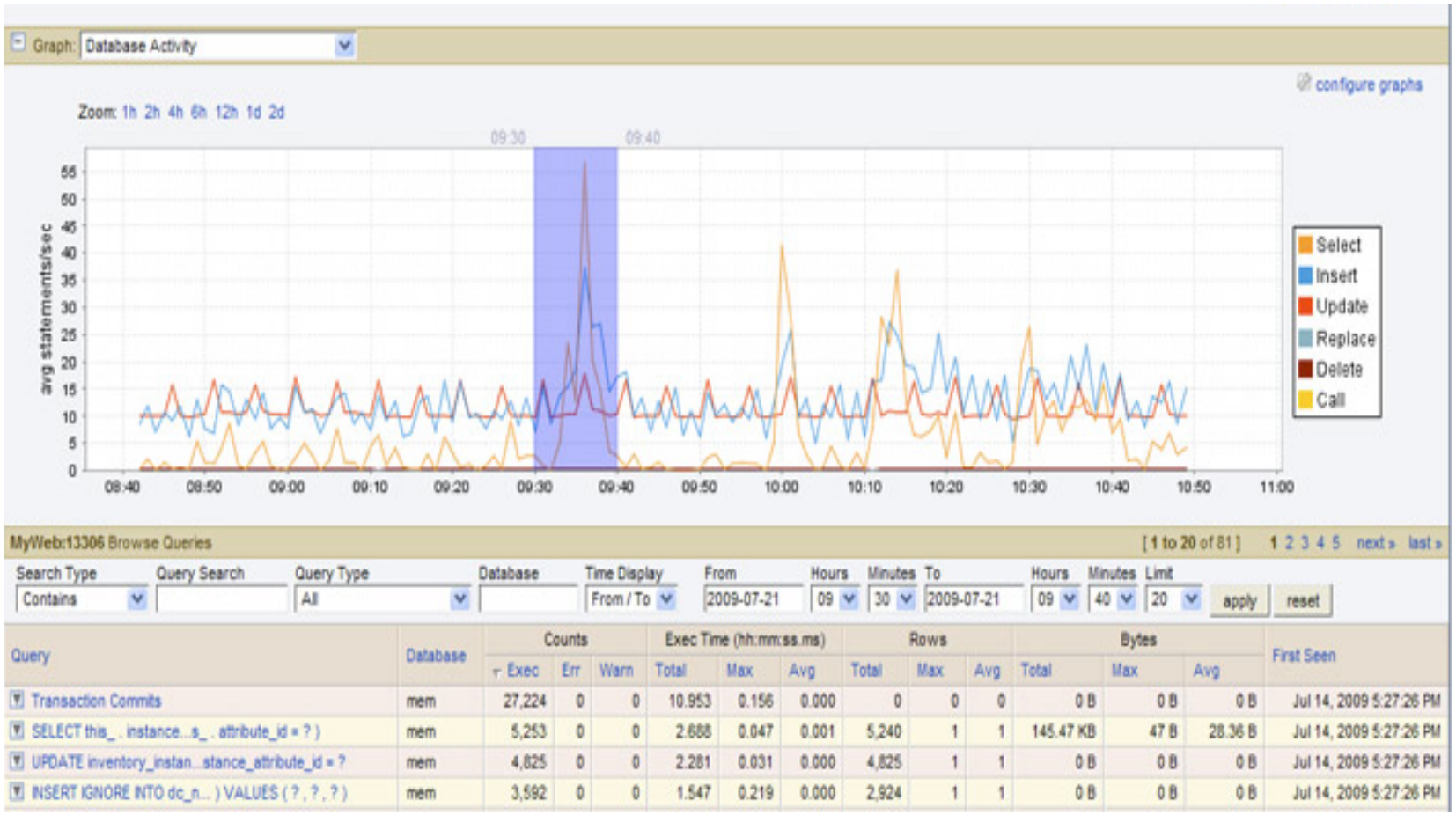
Temporary table

Query Profiling: CPU

```
mysql> select * from sbtest where k = 0 order  by pad desc
  limit 10;
... (22.90 sec)
mysql> show profile cpu;
+----------------------+----------+----------+------------+
| Status               | Duration | CPU_user | CPU_system |
+----------------------+----------+----------+------------+
| starting             | 0.000019 | 0.000000 |   0.000000 |
| init                 | 0.000079 | 0.000000 |   0.000000 |
| checking permissions | 0.000016 | 0.000000 |   0.000000 |
| Opening tables       | 0.000039 | 0.000000 |   0.000000 |
| System lock          | 0.000021 | 0.000000 |   0.000000 |
| init                 | 0.000036 | 0.000000 |   0.000000 |
| optimizing           | 0.000015 | 0.000000 |   0.000000 |
| statistics           | 0.000117 | 0.000000 |   0.000000 |
| preparing            | 0.000028 | 0.000000 |   0.000000 |
| executing            | 0.000007 | 0.000000 |   0.000000 |
| Sorting result       | 3.431279 | 3.430478 |   0.000000 |
| Sending data         | 0.000130 | 0.001000 |   0.000000 |
| end                  | 0.000009 | 0.000000 |   0.000000 |
| query end            | 0.000008 | 0.000000 |   0.000000 |
| closing tables       | 0.000020 | 0.000000 |   0.000000 |
| freeing items        | 0.000413 | 0.000000 |   0.000000 |
| cleaning up          | 0.000015 | 0.000000 |   0.000000 |
+----------------------+----------+----------+------------+
```

Sort is CPU-intensive

MySQL Enterprise Monitor: Query Analyzer, I



Query	Database	Counts			Exec Time (hh:mm:ss.ms)			Rows			Bytes			First Seen
		Exec	Err	Warn	Total	Max	Avg	Total	Max	Avg	Total	Max	Avg	
Transaction Commits	mem	27,224	0	0	10.953	0.156	0.000	0	0	0	0 B	0 B	0 B	Jul 14, 2009 5:27:26 PM
SELECT this_ . instance...s_ . attribute_id = ?)	mem	5,253	0	0	2.688	0.047	0.001	5,240	1	1	145.47 KB	47 B	28.36 B	Jul 14, 2009 5:27:26 PM
UPDATE inventory_instan...stance_attribute_id = ?	mem	4,825	0	0	2.281	0.031	0.000	4,825	1	1	0 B	0 B	0 B	Jul 14, 2009 5:27:26 PM
INSERT IGNORE INTO dc_n...) VALUES (? , ? , ?)	mem	3,592	0	0	1.547	0.219	0.000	2,924	1	1	0 B	0 B	0 B	Jul 14, 2009 5:27:26 PM

http://www.mysql.com/products/enterprise/monitor.html

MySQL Enterprise Monitor: Query Analyzer, II

Canonical Query | Example Query | Explain Query | 908 B | 165 B | 1

Overview of information collected and aggregated for queries of this form.

Alias
None specified.

Canonical Form
truncated | full | formatted

```
SELECT
     COUNT( message_id ) AS cnt,
     DATE_FORMAT(FROM_UNIXTIME(datetime), ?) AS
      fmtdate
     , process_mode, process_type
FROM
     logs_amavis
WHERE
     datetime >= ? AND datetime <= ?
GROUP BY
     process_mode ASC, process_type ASC, fmtdate
     ASC
```

Execution Time Statistics

Max Time	Min Time	Avg Time	Total Time	Standard Deviation
11.123	0.274	2.140	12.838	

Row Statistics

Max Rows	Min Rows	Avg Rows	Total Rows	Standard Deviation	Total Size	Max Size
408	37	129	776	82	24.21 KB	12.8 KB

Number of Executions
6

Time Span
From Oct 27, 2008 10:16:07 AM to Oct 27, 2008 10:46:07 AM.

hide

expand »

Aggregated Form

Canonical Query

Execution and Row Statistics

MySQL Enterprise Monitor: Query Analyzer, III

Canonical Query | Example Query | Explain Query | 908 B | 165 B | 134.

The query with the longest execution time during the Time Span (usually the slowest but not always).

Sampled Query

truncated | full | formatted

```
SELECT
     COUNT( message_id ) AS cnt,
     date_format(from_unixtime(datetime), "%Y-%m-%d")
     AS fmtdate
     , process_mode, process_type
FROM
     logs_amavis
WHERE
     datetime >= 1224498843 and datetime <= 1225080000
GROUP BY
     process_mode ASC, process_type ASC, fmtdate ASC
```

Execution Time
11,122 ms

Date
Oct 27, 2008 10:35:01 AM

User
root

Thread ID
298107

From Host
192.168.0.2:22717

To Host
127.0.0.1:3306

Comments

hide

expand »

Example Query

Execution Time for this Example

MySQL Enterprise Monitor: Query Analyzer, IV

MySQL Explain Plan

Canonical Query | Example Query | Explain Query | 908 B | 165 B | 134.67 B | 10:37:01 AM | alias

Explain of a query that occurred during the Time Span (usually the slowest but not always).

Explain

id	select_type	table	type	possible_keys	key	key_len	ref	rows	extra
1	SIMPLE	logs_amavis	index	null	mdatetimepmpt	47	null	276087	Using where; Using index; Using temporary; Using filesort

hide

expand »

MySQL Query Analyzer Documentation:

http://dev.mysql.com/doc/mysql-monitor/2.0/en/mem-query-analyzer-queries.html

Composite Indexes in MySQL

Composite Indexes

- MySQL chooses 1 (best) index per table
  - With some exceptions...
- The more unique values the better
  - Do not index status, gender, etc
- Order of fields inside index matters
  - (in most cases)
- **"Where region = 'US' and date_added>'2010-05-01' "**
  - Index on (region, date_added) preferred

Composite Indexes

- **“Where region = ‘US’ and date_added>’2010-05-01’ “**
- Index (region, date_added):
  1. MySQL will “jump” to index leaf where Region=‘US’
  2. Scan date_added range starting with the leaf
- Constant + range: Put constant first, range second

GROUP BY Queries

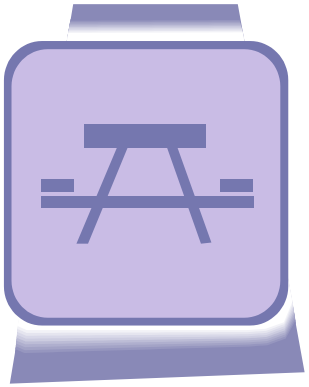
GROUP BY and Temporary Tables

- How many cities in each country?

```
mysql> explain select CountryCode, count(*) from City
   group by CountryCode\G
*************************** 1. row
           id: 1
  select_type: SIMPLE
        table: City
         type: ALL
possible_keys: NULL
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 4079
        Extra: Using temporary; Using filesort
1 row in set (0.00 sec)
```

Temporary tables are slow!

Temporary Tables: Theory



Temporary Tables

- Main performance issues
- MySQL can create temporary tables when query uses:
  - GROUP BY
  - Range + ORDER BY
  - Some other expressions
- 2 types of temporary tables
  - MEMORY
  - On-disk

Temporary Tables

- First, MySQL tries to create temporary table in memory
  - **tmp_table_size**
    - maximum size for in Memory temporary tables
  - **max_heap_table_size**
    - Sets the maximum size for **MEMORY** tables
- If (tmp_table > tmp_table_size OR

  tmp_table > max_heap_table_size)
- { convert to MyISAM temporary table on disk }

Temporary Tables

- MEMORY engine does not support BLOB/TEXT
- `select blob_field from table group by field1`
- `select concat(...string>512 chars) group by field1`
  - Create on-disk temporary table right away

Temporary Tables: Profiling

- Watch:
  - **Created_tmp_tables –** number of temporary table MySQL created in both *RAM and DISK*
  - **Created_tmp_disk_tables -** number of temporary table MySQL created on *DISK*

```
mysql> show session status like 'created%';
+-------------------------+-------+
| Variable_name           | Value |
+-------------------------+-------+
| Created_tmp_disk_tables | 1     |
| Created_tmp_files       | 0     |
| Created_tmp_tables      | 10    |
+-------------------------+-------+
3 rows in set (0.00 sec)
```

Temporary Tables: Practice

Air Traffic Statistics Table for Testing

5M rows, ~2G in size

```
CREATE TABLE `ontime_2010` (
  `YearD` int(11) DEFAULT NULL,
  `MonthD` tinyint(4) DEFAULT NULL,
  `DayofMonth` tinyint(4) DEFAULT NULL,
  `DayOfWeek` tinyint(4) DEFAULT NULL,
  `Carrier` char(2) DEFAULT NULL,
  `Origin` char(5) DEFAULT NULL,
  `DepDelayMinutes` int(11) DEFAULT NULL,
...
) ENGINE=InnoDB DEFAULT CHARSET=latin1
```

http://www.transtats.bts.gov/DL_SelectFields.asp?Table_ID=236&DB_Short_Name=On-Time

GROUP BY Query Example

- Find maximum delay for flights on Sunday
- Group by airline

```
select max(DepDelayMinutes),
     carrier, dayofweek
from ontime_2010
where dayofweek = 7
group by Carrier,  dayofweek
```

GROUP BY Query Example

```
select max(DepDelayMinutes),  carrier, dayofweek
from ontime_2010
where dayofweek = 7
group by Carrier,  dayofweek
```

```
        type: ALL
possible_keys: NULL
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 4833086
        Extra: Using where; Using temporary; Using
filesort
```

Full table scan!

Temporary table!

Fixing Full Table Scan

```
mysql> alter table ontime_2010 add key (dayofweek);

mysql> explain select max(DepDelayMinutes), Carrier,
   dayofweek from ontime_2010
where dayofweek =7 group by Carrier, dayofweek\G
```

```
          type: ref
possible_keys: DayOfWeek
           key: DayOfWeek
       key_len: 2
           ref: const
          rows: 817258
 Extra: Using where; Using temporary; Using filesort
```

Many rows scanned!

Temporary table!

GROUP BY: Adding Covered Index

```
mysql> alter table ontime_2010 add key covered
   (dayofweek, Carrier, DepDelayMinutes);

mysql> explain select max(DepDelayMinutes), Carrier,
   dayofweek from ontime_2010
where dayofweek =7 group by Carrier, dayofweek\G

...
possible_keys: DayOfWeek,covered
          key: covered
      key_len: 2
          ref: const
         rows: 905138
        Extra: Using where; Using index
```

No temporary table!

When Covered Indexes Aren't Good …

```
mysql> explain select max(DepDelayMinutes), Carrier,
  dayofweek from ontime_2010
where dayofweek > 3
group by Carrier,
dayofweek\G
...

          type: range
possible_keys: covered
           key: covered
       key_len: 2
           ref: NULL
          rows: 2441781
         Extra: Using where; Using index; Using
  temporary; Using filesort
```

Range scan

ORDER BY and filesort

ORDER BY and filesort

- Find 10 cities in the US with the largest population

```
mysql> explain select district, name, population from
   City where CountryCode = 'USA' order by population
   desc limit 10\G

           table: City
            type: ALL
   possible_keys: NULL
             key: NULL
         key_len: NULL
             ref: NULL
            rows: 4079
           Extra: Using where; Using filesort
```

Fixing Filesort: Adding Index

```
mysql> alter table City add key my_sort2 (CountryCode,
   population);
```

```
mysql> explain select district, name, population from
   City where CountryCode = 'USA' order by population
   desc limit 10\G
```

```
       table: City
        type: ref
         key: my_sort2
     key_len: 3
         ref: const
        rows: 207
       Extra: Using where
```

No filesort

Sorting and Limit

```
mysql> alter table ontime_2010 add key (DepDelayMinutes);
Query OK, 0 rows affected (38.68 sec)
```

```
mysql> explain select * from ontime_2010
where dayofweek in (6,7) order by DepDelayMinutes desc
limit 10\G
```

```
         type: index
possible_keys: DayOfWeek,covered
          key: DepDelayMinutes
      key_len: 5
          ref: NULL
         rows: 24
        Extra: Using where

10 rows in set (0.00 sec)
```

1. Index is sorted
2. Scan whole table in the order of the index
3. Filter results
4. Stop after finding 10 rows matching the "where" condition

Subqueries and Joins Optimizations

Subqueries, I

- Subquery inside select

```
SELECT (SELECT s1 FROM t2) FROM t1;
```

- Subquery inside where

```
SELECT * FROM t1 WHERE
column1 in (SELECT column2 FROM t2);
```

- Subquery in FROM and joins

```
SELECT sb1,sb2,sb3 FROM (SELECT s1 AS sb1, s2
  AS sb2, s3*2 AS sb3 FROM t1) AS sb
WHERE sb1 > 1;
```

Subqueries, II

- Subquery inside select

```
SELECT (SELECT max(s1) FROM t2), ... FROM t1
where mydate>now();
10000 rows in set
```

- Will execute subquery SELECT max(s1) FROM t2 10,000 times
- Can rewrite it:

```
SELECT max(s1) into @m FROM t2
SELECT @m, ... FROM t1 where ...
```

Subqueries, III

- Subquery inside where

```
SELECT * FROM t1 WHERE
column1 in (SELECT column2 FROM t2);
```

- Will not use index on **column1**
  - http://bugs.mysql.com/bug.php?id=8139
- Can rewrite query as join

Subqueries, IV

- Subquery in FROM and joins

```
SELECT sb1,sb2,sb3 FROM (SELECT s1 AS sb1, s2
  AS sb2, s3*2 AS sb3 FROM t1) AS sb
WHERE sb1 > 1;
```

- MySQL will create a temporary table for **`(SELECT s1 AS sb1, s2 AS sb2, s3*2 AS sb3 FROM t1)`** with no indexes

- Rewrite as join

Joins, I

- Use INT to join tables rather than VARCHAR

```
select * from city ct
join country c on ct.CountryCode = c.Code;
```

```
CREATE TABLE `country` (
   `Code` varchar(50) NOT NULL DEFAULT '',
 ...
  PRIMARY KEY (`Code`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1

CREATE TABLE `city` (
  `ID` int(11) NOT NULL AUTO_INCREMENT,
  ...
  `Name` char(35) NOT NULL DEFAULT '',
  `CountryCode` varchar(50) NOT NULL DEFAULT '',
  PRIMARY KEY (`ID`),
  KEY `CountryCode` (`CountryCode`),
  CONSTRAINT `city_ibfk_1` FOREIGN KEY (`CountryCode`) REFERENCES `country` (`Code`)
```

Use code_id INT instead

Joins, II

- Use INT to join tables rather than VARCHAR

```
select * from city ct
join country c on ct.CountryCode = c.Code;
```

```
CREATE TABLE `country` (
   `Code_ID` INT NOT NULL AUTO_INCREMENT DEFAULT '',
 ...
  PRIMARY KEY (`Code_ID`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1

CREATE TABLE `city` (
  `ID` int(11) NOT NULL AUTO_INCREMENT,
  ...
  `Name` char(35) NOT NULL DEFAULT '',
  `CountryCode_ID` INT NOT NULL DEFAULT '',
  PRIMARY KEY (`ID`),
  KEY `CountryCode` (`CountryCode`),
  CONSTRAINT `city_ibfk_1` FOREIGN KEY (`CountryCode`) REFERENCES `country` (`Code`)
```

Use code_id INT instead

Reporting Queries

Reporting Example: Table

```
mysql> CREATE TABLE `lineitem` (
  `l_shipdate` date NOT NULL,
  `l_orderkey` int(11) NOT NULL,
  `l_partkey` int(11) NOT NULL,
  `l_suppkey` int(11) NOT NULL,
  `l_linenumber` int(11) NOT NULL,
  `l_quantity` decimal(15,2) NOT NULL,
  `l_extendedprice` decimal(15,2) NOT NULL,
  `l_discount` decimal(15,2) NOT NULL,
  `l_tax` decimal(15,2) NOT NULL,
  `l_returnflag` char(1) NOT NULL,
  `l_linestatus` char(1) NOT NULL,
  KEY `lineitem_fk2` (`l_suppkey`),
  KEY `lineitem_fk3` (`l_partkey`,`l_suppkey`),
  KEY `li_shp_dt_idx` (`l_shipdate`),
  KEY `li_com_dt_idx` (`l_commitdate`),
  KEY `li_rcpt_dt_idx` (`l_receiptdate`),
);
```

Reporting Example: Query

- Group by year(date)

```
mysql> explain
select sum(l_extendedprice), year(l_shipdate) as yr
from lineitem group by yr limit 10\G
*************************** 1. row
           id: 1
  select_type: SIMPLE
        table: lineitem
         type: ALL
possible_keys: NULL
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 116771866
        Extra: Using temporary; Using filesort
```

year(field) = calculated

MySQL can't use index

Adding Year/Month

```
mysql> alter table lineitem add yr year, add key(yr);
mysql> update lineitem set yr = year(l_shipdate);

mysql> explain  select sum(l_extendedprice),  yr
from lineitem  group by yr desc limit 10\G
*************************** 1. row ******************
           id: 1
  select_type: SIMPLE
        table: lineitem
         type: index
possible_keys: NULL
          key: yr
      key_len: 2
          ref: NULL
         rows: 116771866
        Extra:
1 row in set (0.00 sec)
```

No temporary, no filesort

Add covered index for better performance

Dimension Table

```
CREATE TABLE dates (
      date_id           INT UNSIGNED NOT NULL PRIMARY KEY,
      date              DATE NOT NULL,
      day_of_week       INT NOT NULL,
      month             VARCHAR(10) NOT NULL,
      month_day         INT NOT NULL,
      year              INT NOT NULL,
     UNIQUE KEY `date` (`date`)
);
```

Flexible for different queries

```
mysql> alter table lineitem
add date_id INT UNSIGNED NOT NULL,
add key (date_id);
```

Dimension Table: Test

```
mysql> explain select d.year, d.month_day, sum(l_extendedprice)
   from dates d, lineitem1 l where l.date_id = d.date_id group
   by d.year, d.month_day\G
          table: l
           type: ALL
possible_keys: date_id
            key: NULL
        key_len: NULL
            ref: NULL
           rows: 116771866
          Extra: Using temporary; Using filesort
*************************** 2. row ******************
          table: d
           type: eq_ref
possible_keys: PRIMARY,date_id
            key: PRIMARY
        key_len: 4
            ref: tpch.l.date_id
           rows: 1
          Extra:
```

Flexible, but slow!

No index, temporary table

Summary Tables, I

```
mysql> create table lineitem_summary as
select year(l_shipdate) as yr,
month(l_shipdate) as mon,
sum(l_extendedprice) as revenue,
count(*) as num_orders
from lineitem group by yr, mon;

Query OK, 365 rows affected
 Records: 365  Duplicates: 0
```

Data is already aggregated in summary table

Summary Tables, II

- Aggregate by Year, based on summary table

```
mysql> select yr,
sum(l_extendedprice) as revenue,
count(*) as num_orders
from lineitem_summary group by yr;
```

Only 356 rows for 1 year or 3560 for 10 years

Data already aggregated
Small number of records
= Queries are much faster!

Summary Tables, III

- Advantages
  - Significantly faster for queries
  - Smaller number of rows
- Disadvantages
  - Needs to be updated: cron or manually
  - More data to store

Make sense for reporting

Use MySQL slave server

MySQL Full Text Index

- Available in MyISAM tables before MySQL 5.6
  - InnoDB supports full text index in MySQL 5.6
- Natural language search and Boolean search
- 4 char per word by default
- Stop word list by default
- Frequency-based ranking
  - Distance between words is not counted

MySQL Full Text: Creating Full Text Indexes

```
mysql> CREATE TABLE articles (
-> id INT UNSIGNED AUTO_INCREMENT
 NOT NULL PRIMARY KEY,
-> title VARCHAR(200),
-> body TEXT,
-> FULLTEXT (title,body)
-> ) engine=MyISAM;
```

MySQL Full Text: Natural Language Mode

```
mysql> SELECT * FROM articles
-> WHERE MATCH (title,body)
-> AGAINST ('database' IN NATURAL
 LANGUAGE MODE);
+-------------------+----------------------------------------+
| title | body                                                |
+-------------------+----------------------------------------+
| MySQL vs. YourSQL | In the following database comparison ... |
| MySQL Tutorial | DBMS stands for DataBase ...              |
```

In natural language mode:
Default sorting by relevance!

MySQL Full Text: Boolean Mode

```
mysql> SELECT * FROM articles
-> WHERE MATCH (title,body)
-> AGAINST ('cat AND dog' IN
 BOOLEAN MODE);
```

No default sorting in Boolean mode!

New in MySQL 5.6: Features, Performance and Monitoring

ORACLE

The World's Most Popular Open Source Database

Monitoring: Performance Schema, I

```
| Tables_in_performance_schema                                |
+-------------------------------------------------------------+
| events_statements_current                                   |
| events_statements_history                                   |
| events_statements_history_long                              |
| events_statements_summary_by_host_by_event_name             |
| events_statements_summary_by_thread_by_event_name           |
| events_statements_summary_by_user_by_event_name             |
| events_statements_summary_by_user_host_by_event_name        |
| events_statements_summary_global_by_event_name              |
```

Statement list for all queries

Used for query profiling

Monitoring: Performance Schema, II

- How to find queries creating disk temporary tables

```
mysql> select * from events_statements_history_long
   where  CREATED_TMP_DISK_TABLES > 0 limit 10\G
              ...
                SQL_TEXT: SELECT DISTINCT c from sbtest where id
   between 847399 and 847499 order by c
               ROWS_SENT: 1
           ROWS_EXAMINED: 103
CREATED_TMP_DISK_TABLES: 1
      CREATED_TMP_TABLES: 1
        SELECT_FULL_JOIN: 0
 SELECT_FULL_RANGE_JOIN: 0
            SELECT_RANGE: 1
      SELECT_RANGE_CHECK: 0
             SELECT_SCAN: 0
       SORT_MERGE_PASSES: 0
              SORT_RANGE: 0
               SORT_ROWS: 1
               SORT_SCAN: 1
           NO_INDEX_USED: 0
      NO_GOOD_INDEX_USED: 0
```

MySQL 5.6 “labs release”

from labs.mysql.com/

Monitoring: Performance Schema, III

- List of queries creating disk temporary tables

```
mysql> select sql_text, count(*) as cnt  from
   events_statements_history_long
where  CREATED_TMP_DISK_TABLES > 0
group by sql_text order by cnt desc  limit 10;
+-----------------------------------------------------------------------------+-----+
| sql_text                                                                    | cnt |
+-----------------------------------------------------------------------------+-----+
| SELECT DISTINCT c from sbtest where id between 242012 and 242112 order by c |   2 |
| SELECT DISTINCT c from sbtest where id between 797388 and 797488 order by c |   2 |
| SELECT DISTINCT c from sbtest where id between 973150 and 973250 order by c |   1 |
| SELECT DISTINCT c from sbtest where id between 478783 and 478883 order by c |   1 |
| SELECT DISTINCT c from sbtest where id between 967035 and 967135 order by c |   1 |
| SELECT DISTINCT c from sbtest where id between 602102 and 602202 order by c |   1 |
| SELECT DISTINCT c from sbtest where id between 123827 and 123927 order by c |   1 |
| SELECT DISTINCT c from sbtest where id between 980527 and 980627 order by c |   1 |
| SELECT DISTINCT c from sbtest where id between 450354 and 450454 order by c |   1 |
| SELECT DISTINCT c from sbtest where id between 674804 and 674904 order by c |   1 |
+-----------------------------------------------------------------------------+-----+
10 rows in set (0.04 sec)
```

Optimizer Improvements, I

- Index Condition Pushdown
  - Moves more of the processing for WHERE clauses to the storage engine
  - = less I/O overhead
  - = less internal communication overhead
  - InnoDB, MyISAM, and NDBCLUSTER
- Multi-Range Read (MRR)
  - Faster to read data sequentially than to do random accesses
  - MRR-
    - Scans one or more index ranges used in query
    - Sorts the associated disk blocks for the row data
    - Reads those disk blocks using larger sequential I/O requests.
  - Works for all storage engines

Optimizer Improvements, II

- File Sort Optimization
  - ORDER BY *non_indexed_column* LIMIT n
  - speeds up the sort when the contents of n rows can fit into the sort buffer
  - Works for all storage engines

MySQL 5.6 InnoDB Improvements

- Full Text Search Indexes for InnoDB tables
- Split Kernel Mutex
  - Remove bottlenecks in busy systems
- Multi-Threaded Purge and Separate Flush Thread
  - Increases performance and provides better scalability
- Persistent Optimizer Stats
  - Provides improved accuracy of InnoDB index statistics and consistency across MySQL restarts
- New InnoDB monitoring tables in INFORMATION_SCHEMA
- Much more!

The presentation 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.