Learn How MySQL 5.6 Makes Query Optimization Easier

Jaime Crespo
Percona Technical Webinars
March 15, 2013

About This Presentation

- 5.6 from a pure technical view
  - Only query tuning, not other optimization or profiling features
  - Practical, simple examples but with real data (not “best cases”)
- **NOT benchmarks**
  - Only response time/query strategy, not extra CPU/memory/locking
  - No comparisons with Percona Server or MariaDB
- 5.6 optimizer has a lot of room for improvement, specially for query plan decision
  - Some features are poorly documented

Agenda

- MySQL 5.6 Optimizer New Features
  - Filesort with short LIMIT
  - ICP
  - MRR
  - BKA
  - Index merge
  - Other Subquery and JOIN Optimizations
- MySQL 5.6 Query Plan Information
  - EXPLAIN for DML
  - Structured EXPLAIN and Query Plan Profiler
  - Persistent Optimizer Statistics
  - Duplicate Key Check

Test Platform for Queries

- We will compare official Oracle's MySQL 5.5.30 vs 5.6.10 binary tarballs
- Base platform:
  - CentOS 6.3
  - 4 Intel(R) Core(TM) i7-3770 CPU @ 3.40GHz
  - 32 GB RAM
  - Software RAID1 of SATA 6 GB/s HDD 7200 rpm
- Standard options (not default) except otherwise noted
  - E.g. bigger transaction log and buffer pool, disabled P_S
- Queries are executed several times between reboots
- We will be using FORCE INDEX quite liberally
  - Not a good general practice

Original Test Schema

```
CREATE TABLE `cast_info` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `person_id` int(11) NOT NULL,
  `movie_id` int(11) NOT NULL,
  `person_role_id` int(11) DEFAULT NULL,
  `note` text DEFAULT NULL,
  `nr_order` int(11) DEFAULT NULL,
  `role_id` int(11) NOT NULL,
  PRIMARY KEY (`id`),
) ENGINE=InnoDB AUTO_INCREMENT=22187769
DEFAULT CHARSET=utf8;

CREATE TABLE `char_name` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `name` text NOT NULL,
  `imdb_index` varchar(12) DEFAULT NULL,
  `imdb_id` int(11) DEFAULT NULL,
  `name_pcode_nf` varchar(5) DEFAULT NULL,
  `surname_pcode` varchar(5) DEFAULT NULL,
  PRIMARY KEY (`id`),
) ENGINE=InnoDB AUTO_INCREMENT=2406562
DEFAULT CHARSET=utf8;
```

```
CREATE TABLE `movie_info` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `movie_id` int(11) NOT NULL,
  `info_type_id` int(11) NOT NULL,
  `info` text NOT NULL,
  `note` text,
  PRIMARY KEY (`id`),
) ENGINE=InnoDB AUTO_INCREMENT=9748371 DEFAULT
CHARSET=utf8;

CREATE TABLE `title` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `title` text NOT NULL,
  `imdb_index` varchar(12) DEFAULT NULL,
  `kind_id` int(11) NOT NULL,
  `production_year` int(11) DEFAULT NULL,
  `imdb_id` int(11) DEFAULT NULL,
  `phonetic_code` varchar(5) DEFAULT NULL,
  `episode_of_id` int(11) DEFAULT NULL,
  `season_nr` int(11) DEFAULT NULL,
  `episode_nr` int(11) DEFAULT NULL,
  `series_years` varchar(49) DEFAULT NULL,
  `title_crc32` int(10) unsigned DEFAULT NULL,
  PRIMARY KEY (`id`),
) ENGINE=InnoDB AUTO_INCREMENT=1543721 DEFAULT
CHARSET=utf8;
```

Learn How MySQL 5.6 Makes Query Optimization Easier

Optimizer New Features

Filesort with Short LIMIT

- Queries that require a filesort but only the first records are selected can benefit from this optimization:

```
mysql> EXPLAIN select * from movie_info
ORDER BY info LIMIT 100\G
******************** 1. row ******************
           id: 1
  select_type: SIMPLE
        table: movie_info
         type: ALL
possible_keys: NULL
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 6927988
        Extra: Using filesort
1 row in set (0.00 sec)
```

Note: Both EXPLAIN and the STATUS Handlers show the same outcome

Filesort with Short LIMIT (cont.)

- `SELECT * FROM movie_info ORDER BY info LIMIT 100;`
  - MySQL 5.5: 20.06 sec
  - MySQL 5.6 (P_S on): 9.14 sec
  - MySQL 5.6 (P_S off): 8.51 sec
- Over 2x faster.
  - Exact speed-up may depend on the original sort buffer size

Index Condition Pushdown

- Let's prepare a use case:

```
UPDATE cast_info SET note = left(note,
250);
ALTER TABLE cast_info MODIFY note
varchar(250), ADD INDEX role_id_note
(role_id, note);
```

- We want to execute:

```
SELECT * FROM cast_info
WHERE role_id = 1
and note like '%Jaime%';
```

Without ICP (5.5)

```
mysql> EXPLAIN SELECT *
FROM cast_info
WHERE role_id = 1
and note like '%Jaime%'\G
********** 1. row **********
           id: 1
  select_type: SIMPLE
        table: cast_info
         type: ref
possible_keys: role_id_note
          key: role_id_note
      key_len: 4
          ref: const
         rows: 11553718
        Extra: Using where
1 row in set (0.01 sec)
```

```
mysql> SHOW STATUS like 'Hand%';
+----------------------------+---------+
| Variable_name              | Value   |
+----------------------------+---------+
| Handler_commit             | 1       |
| Handler_delete             | 0       |
| Handler_discover           | 0       |
| Handler_prepare            | 0       |
| Handler_read_first         | 0       |
| Handler_read_key           | 1       |
| Handler_read_last          | 0       |
| Handler_read_next          | 8346769 |
| Handler_read_prev          | 0       |
| Handler_read_rnd           | 0       |
| Handler_read_rnd_next      | 0       |
| Handler_rollback           | 0       |
| Handler_savepoint          | 0       |
| Handler_savepoint_rollback | 0       |
| Handler_update             | 0       |
| Handler_write              | 0       |
+----------------------------+---------+
16 rows in set (0.00 sec)
```

With ICP (5.6)

```
mysql> EXPLAIN SELECT *
FROM cast_info
WHERE role_id = 1 and note like
'%Jaime%'\G
************ 1. row ************
           id: 1
  select_type: SIMPLE
        table: cast_info
         type: ref
possible_keys: role_id_note
          key: role_id_note
      key_len: 4
          ref: const
         rows: 10259274
        Extra: Using index
condition
1 row in set (0.00 sec)
```

```
mysql> SHOW STATUS like 'Hand%';
+----------------------------+-------+
| Variable_name              | Value |
+----------------------------+-------+
| Handler_commit             | 1     |
| Handler_delete             | 0     |
| Handler_discover           | 0     |
| Handler_external_lock      | 2     |
| Handler_mrr_init           | 0     |
| Handler_prepare            | 0     |
| Handler_read_first         | 0     |
| Handler_read_key           | 1     |
| Handler_read_last          | 0     |
| Handler_read_next          | 266   |
| Handler_read_prev          | 0     |
| Handler_read_rnd           | 0     |
| Handler_read_rnd_next      | 0     |
| Handler_rollback           | 0     |
| Handler_savepoint          | 0     |
| Handler_savepoint_rollback | 0     |
| Handler_update             | 0     |
| Handler_write              | 0     |
+----------------------------+-------+
18 rows in set (0.00 sec)
```

Comparison of ICP Execution

- Execution time for this example:
  - MySQL 5.5: 5.76 sec
  - MySQL 5.6: 1.09 sec
- Over 5x faster
- In this example, it would not work with a prefix index (e.g. TEXT/BLOB), as it must search the whole field
- Fun fact: if covering technique is tried, it actually runs slower than with ICP for this case/hardware (#68554)

ICP and Indexing

- ICP will change the way we index our tables

```
SELECT * FROM cast_info
FORCE INDEX(person_role_id_role_id)
WHERE person_role_id > 0
and person_role_id < 150000
and role_id > 1 and role_id < 7;
```

- For example, a multiple column index can be, in some cases, efficiently used for a range condition on several columns:
  - Effectiveness is highly dependent on how selective is the second part of the index (an “index scan” is still done at engine level)

MySQL Option Change

- For the next test cases, we will change the Buffer Pool size to:
  `innodb_buffer_pool_size = 100M`
  as the next optimizations depend on data being accessed mainly on disk

Multi-Range Read (MRR)

- Reorders access to table data when using a secondary index on disk for sequential I/O
  - Like ICP, its efficiency is highly dependent on data distribution and memory contents
- It also depends on hardware sequential access (e.g., InnoDB's read ahead)
  - Not useful on SSDs
- Not compatible with Using index
- Only for range access and equi-joins

MRR Example

- We want to execute:

```
SELECT * FROM cast_info
WHERE person_role_id > 0
and person_role_id < 150000;
SELECT * FROM cast_info
WHERE role_id = 3
and person_role_id > 0
and person_role_id < 500000;
```

- We will use these indexes, respectively:

```
ALTER TABLE cast_info
ADD INDEX person_role_id (person_role_id);
ALTER TABLE cast_info
ADD INDEX person_role_id_role_id (person_role_id,
role_id);
```

Without MRR (5.5)

```
mysql> EXPLAIN SELECT * FROM
cast_info FORCE
INDEX(person_role_id)  WHERE
person_role_id > 0 and
person_role_id < 150000\G
*********** 1. row ***********
           id: 1
  select_type: SIMPLE
        table: cast_info
         type: range
possible_keys: person_role_id
          key: person_role_id
      key_len: 5
          ref: NULL
         rows: 8966490
        Extra: Using where
1 row in set (0.00 sec)
```

```
mysql> SHOW STATUS like 'Hand%';
+----------------------------+---------+
| Variable_name              | Value   |
+----------------------------+---------+
| Handler_commit             | 1       |
| Handler_delete             | 0       |
| Handler_discover           | 0       |
| Handler_prepare            | 0       |
| Handler_read_first         | 0       |
| Handler_read_key           | 1       |
| Handler_read_last          | 0       |
| Handler_read_next          | 4654312 |
| Handler_read_prev          | 0       |
| Handler_read_rnd           | 0       |
| Handler_read_rnd_next      | 0       |
| Handler_rollback           | 0       |
| Handler_savepoint          | 0       |
| Handler_savepoint_rollback | 0       |
| Handler_update             | 0       |
| Handler_write              | 0       |
+----------------------------+---------+
16 rows in set (0.00 sec)
```

With MRR (5.6)

```
mysql> EXPLAIN SELECT * FROM
cast_info FORCE
INDEX(person_role_id)  WHERE
person_role_id > 0 and
person_role_id < 150000\G
*********** 1. row ***********
           id: 1
  select_type: SIMPLE
        table: cast_info
         type: range
possible_keys: person_role_id
          key: person_role_id
      key_len: 5
          ref: NULL
         rows: 8966490
        Extra: Using index
condition; Using MRR
1 row in set (0.00 sec)
```

```
mysql> SHOW STATUS like 'Hand%';
+----------------------------+---------+
| Variable_name              | Value   |
+----------------------------+---------+
| Handler_commit             | 1       |
| Handler_delete             | 0       |
| Handler_discover           | 0       |
| Handler_external_lock      | 4       |
| Handler_mrr_init           | 0       |
| Handler_prepare            | 0       |
| Handler_read_first         | 0       |
| Handler_read_key           | 4654313 |
| Handler_read_last          | 0       |
| Handler_read_next          | 4654312 |
| Handler_read_prev          | 0       |
| Handler_read_rnd           | 4654312 |
| Handler_read_rnd_next      | 0       |
| Handler_rollback           | 0       |
| Handler_savepoint          | 0       |
| Handler_savepoint_rollback | 0       |
| Handler_update             | 0       |
| Handler_write              | 0       |
+----------------------------+---------+
18 rows in set (0.00 sec)
```

Comparison of MRR Execution

- Execution time for this example:
  - MySQL 5.5: 4654312 rows in set (1 min 4.79 sec)
  - MySQL 5.6 (w/MRR, wo/ICP): 4654312 rows in set (48.64 sec)
- Consistent 33% execution improvement for this test case
  - Difficult to see for smaller resultsets
- What has changed?
  - 15% more “read_ahead”s, resulting in a 40% less data reads
  - Reordering has an overhead, can be tuned with read_rnd_buffer_size

What's the Point for the 'FORCE INDEX'?

- For demonstration purposes (full table scan is better here)
- Some features are not properly detected yet:

```
SELECT *
FROM cast_info FORCE INDEX(person_role_id_role_id)
WHERE role_id = 3
and person_role_id > 0
and person_role_id < 500000;
```

- Full table scan (any version): Empty set (8.08 sec)
- MySQL 5.5: Empty set (1 min 16.88 sec)
- MySQL 5.6 ICP+MRR*: Empty set (0.46 sec)

*ICP is responsible for the dramatic change in execution time, not MRR

Batched Key Access (BKA)

- It retrieves keys in batches and allows MRR usage for JOINs, as an alternative to standard Nested Loop Join execution
- Not enabled by default

```
SET optimizer_switch=
'mrr=on,mrr_cost_based=off,batched_key_access=on';
```

Without BKA (5.5) - EXPLAIN

```
mysql> EXPLAIN
SELECT cast_info.note, char_name.name
FROM cast_info FORCE index(person_role_id)
JOIN char_name
ON cast_info.person_role_id = char_name.id
WHERE cast_info.person_role_id > 0 and
cast_info.person_role_id < 150000
************* 1. row *************
           id: 1
  select_type: SIMPLE
        table: char_name
         type: range
possible_keys: PRIMARY
          key: PRIMARY
      key_len: 4
          ref: NULL
         rows: 313782
        Extra: Using where
...
```

```
...
************* 2. row *************
           id: 1
  select_type: SIMPLE
        table: cast_info
         type: ref
possible_keys: person_role_id
          key: person_role_id
      key_len: 5
          ref: imdb.char_name.id
         rows: 4
        Extra: NULL
2 rows in set (0.00 sec)
```

Without BKA (5.5) - Handlers

```
mysql> SHOW STATUS like 'Hand%';
+----------------------------+---------+
| Variable_name              | Value   |
+----------------------------+---------+
| Handler_commit             | 1       |
| Handler_delete             | 0       |
| Handler_discover           | 0       |
| Handler_prepare            | 0       |
| Handler_read_first         | 0       |
| Handler_read_key           | 150000  |
| Handler_read_last          | 0       |
| Handler_read_next          | 4804311 |
| Handler_read_prev          | 0       |
| Handler_read_rnd           | 0       |
| Handler_read_rnd_next      | 0       |
| Handler_rollback           | 0       |
| Handler_savepoint          | 0       |
| Handler_savepoint_rollback | 0       |
| Handler_update             | 0       |
| Handler_write              | 0       |
+----------------------------+---------+
18 rows in set (0.00 sec)
```

With BKA (5.6) - EXPLAIN

```
mysql> EXPLAIN
SELECT cast_info.note, char_name.name
FROM cast_info FORCE index(person_role_id)
JOIN char_name ON cast_info.person_role_id
= char_name.id
WHERE cast_info.person_role_id > 0 and
cast_info.person_role_id < 150000\G
************ 1. row ************
           id: 1
  select_type: SIMPLE
        table: char_name
         type: range
possible_keys: PRIMARY
          key: PRIMARY
      key_len: 4
          ref: NULL
         rows: 313782
        Extra: Using where
...
```

```
...
************ 2. row ************
           id: 1
  select_type: SIMPLE
        table: cast_info
         type: ref
possible_keys: person_role_id
          key: person_role_id
      key_len: 5
          ref: imdb.char_name.id
         rows: 4
        Extra: Using join buffer
(Batched Key Access)
2 rows in set (0.00 sec)
```

With BKA (5.6) - Handlers

```
mysql> SHOW STATUS like 'Hand%';
+----------------------------+---------+
| Variable_name              | Value   |
+----------------------------+---------+
| Handler_commit             | 1       |
| Handler_delete             | 0       |
| Handler_discover           | 0       |
| Handler_external_lock      | 6       |
| Handler_mrr_init           | 1       |
| Handler_prepare            | 0       |
| Handler_read_first         | 0       |
| Handler_read_key           | 4804312 |
| Handler_read_last          | 0       |
| Handler_read_next          | 4804311 |
| Handler_read_prev          | 0       |
| Handler_read_rnd           | 4654312 |
| Handler_read_rnd_next      | 0       |
| Handler_rollback           | 0       |
| Handler_savepoint          | 0       |
| Handler_savepoint_rollback | 0       |
| Handler_update             | 0       |
| Handler_write              | 0       |
+----------------------------+---------+
18 rows in set (0.00 sec)
```

Comparison of BKA Execution

- Execution time for this example:
  - MySQL 5.5: 4654312 rows in set (1 min 6.78 sec)
  - MySQL 5.6 (w/MRR, w/BKA): 4654312 rows in set (1 min 0.47 sec)
- The results are consistent between executions, but the gain is not too big. But if we change the join_buffer_size…
  - MySQL 5.6 (w/MRR, w/BKA, join_buffer_size = 50M): 4654312 rows in set (19.54 sec)
    (join_buffer_size does not affect execution time in the 5.5 version)

Better Index Merge

- In this example, it avoids a full table scan:

```
mysql>  EXPLAIN SELECT COUNT(*) FROM title WHERE (title =
'Pilot' OR production_year > 2010) AND kind_id < 4\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: title
         type: index_merge
possible_keys: title,production_year
          key: title,production_year
      key_len: 77,5
          ref: NULL
         rows: 4434
        Extra: Using sort_union(title,production_year);
Using where
1 row in set (0.00 sec)
```

(almost all titles have kind_id < 4)

- MySQL 5.5: 0.79s – MySQL 5.6: 0.01s

Extended Secondary Keys

- Implicit primary keys inside secondary keys can be used for filtering (ref, range, etc), not only for covering index or sorting.
- Requires `use_index_extensions=on` (default)

```
ALTER TABLE title add index
(title(25));
SELECT COUNT(*) FROM title
WHERE title = 'Pilot'
AND id BETWEEN 1000 AND 1000000;
```

Extended Secondary Keys

```
mysql> EXPLAIN SELECT COUNT(*) FROM title WHERE
title = 'Pilot' AND id BETWEEN 1000 AND
1000000\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: title
         type: range
possible_keys: PRIMARY,title
          key: title
      key_len: 81
          ref: NULL
         rows: 531
        Extra: Using index condition; Using
where
1 row in set (0.00 sec)
```

JOINs and Subqueries

- Lazy Subquery Materialization
  - Useful for FROM subqueries if further filtering is done
- Better detection of IN + non-dependent subqueries
  - Do not need to be converted to JOIN anymore for better execution
- Join Order with Many Tables
  - Table order algorithm has been optimized, which leads to better query plans

Learn How MySQL 5.6 Makes Query Optimization Easier

Optimizer Profiling and Tools

EXPLAIN on DML

- EXPLAIN is now available also for INSERT, UPDATE and DELETE

```
mysql> EXPLAIN DELETE FROM title WHERE title = 'Pilot'\G
*********************** 1. row ***********************
           id: 1
  select_type: SIMPLE
        table: title
         type: range
possible_keys: PRIMARY,title
          key: title
      key_len: 77
          ref: NULL
         rows: 1380
        Extra: Using where
1 row in set (0.00 sec)
```

Structured EXPLAIN

```
mysql> EXPLAIN FORMAT=JSON SELECT COUNT(*) FROM title
WHERE (title = 'Pilot' OR production_year > 2010)
AND kind_id < 4\G
*************************** 1. row ***************************
EXPLAIN: {
  "query_block": {
    "select_id": 1,
    "table": {
      "table_name": "title",
      "access_type": "index_merge",
      "possible_keys": [
        "title",
        "production_year"
      ],
      "key": "sort_union(title,production_year)",
      "key_length": "77,5",
      "rows": 4434,
      "filtered": 100,
      "attached_condition": "((`imdb`.`title`.`title` = 'Pilot') or
(`imdb`.`title`.`production_year` > 2010)) and
(`imdb`.`title`.`kind_id` < 4))"
    }
  }
}
```

Optimizer Trace

- Allows profiling of MySQL query planner
- It shows not only information about the final query plan (EXPLAIN), but also about other discarded strategies, and its “execution cost”
- It can be accessed via the INFORMATION_SCHEMA database
- It is off by default

Checking the Optimizer Trace

- `mysql> SET optimizer_trace="enabled=on";`
- `mysql> SELECT COUNT(*) FROM title WHERE (title = 'Pilot' OR production_year > 2010) AND kind_id < 4;`

```
+-----------+
| COUNT(*) |
+-----------+
|     3050 |
+-----------+
1 row in set (0.01 sec)
```

- `mysql> SELECT trace FROM information_schema.optimizer_trace;`

Checking the Optimizer Trace (cont.)


```
        {
          "range_scan_alternatives": [
            {
              "index": "production_year",
              "ranges": [
                "2010 < production_year"
              ],
              "index_dives_for_eq_ranges": true,
              "rowid_ordered": false,
              "using_mrr": false,
              "index_only": true,
              "rows": 3054,
              "cost": 615.16,
              "chosen": true
            }
          ],
          "index_to_merge": "production_year",
          "cumulated_cost": 905.69
        }
      ],
      "cost_of_reading_ranges": 905.69,
      "cost_sort_rowid_and_read_disk": 3672.8,
      "cost_duplicate_removal": 9467.6,
      "total_cost": 14046
    }
  ],
  "chosen_range_access_summary": {
    "range_access_plan": {
      "type": "index_merge",
      "index_merge_of": [
```

Persistent Optimizer Statistics

- InnoDB index statistics are no longer discarded on server shutdown and recomputed the next time a table is accessed
- Controlled by variable:
  `innodb_stats_persistent = ON` (default)
- Remember that SHOW commands/accessing to I_S do not automatically regenerate statistics by default

Duplicated Key Check

```
mysql> alter table title add index (production_year);
Query OK, 0 rows affected (10.08 sec)
Records: 0  Duplicates: 0  Warnings: 0
mysql> alter table title add index (production_year);
Query OK, 0 rows affected, 1 warning (5.11 sec)
Records: 0  Duplicates: 0  Warnings: 1

mysql> show warnings\G
*************************** 1. row ***************************
  Level: Note
   Code: 1831
Message: Duplicate index 'production_year_2' defined on
the table 'imdb.title'. This is deprecated and will be
disallowed in a future release.
1 row in set (0.00 sec)
```

Learn How MySQL 5.6 Makes Query Optimization Easier

CONCLUSION

5.6 is a Great Release

- Many optimizer changes that can potentially improve query execution time
- Some of them are transparent, some others require tuning and understanding
- Some old tricks and indexing strategies become obsolete in 5.6
- pt-upgrade, from Percona Toolkit, and Percona Playback can be great tools to analyze improvements and regressions

Where to Learn More ?

- More query and server optimization techniques in our training courses (http://www.percona.com/training):
  - Chicago, begins Monday, April 8, 2013
  - London, begins Monday, April 8, 2013
  - 15% discount for April training with coupon code **W15**
- Stay tuned for our new MySQL 5.6 course in May
- http://www.mysqlperformanceblog.com

Percona Live Conference and Expo

Learn from Leading MySQL Experts
Santa Clara, CA, April 22nd – 25th, 2013

- **Use Discount Code: "PerWebinar1" to receive 15% discount**
- **For more information and to register:**
  Visit: http://www.percona.com/live/mysql-conference-2013/

Thank You!

Jaime Crespo
jaime.crespo@percona.com