When and How to Take Advantage of New Optimizer Features in MySQL 5.6

Øystein Grøvlen
Senior Principal Software Engineer, MySQL
Oracle
Program Agenda

- Improvements for disk-bound queries
- Subquery improvements
- Index condition pushdown
- Misc. optimizer improvements
MySQL 5.6: Improvements for Disk-Bound Queries

- Main idea: Sort keys retrieved from index before accessing table
- Benefits:
  - Read more rows from a page while it is still in buffer pool
  - Increased benefits from prefetching pages into the buffer pool
  - Sequential instead of random disk access?
- Range scan:
  - Disk Sweep Multi-Range Read (DS-MRR)
- Index lookup (Ref access):
  - Batched Key Access
MySQL 5.5: Data Access without DS-MRR

Index scan

Random access
MySQL 5.6: Data Access with DS-MRR

InnoDB Example

Index scan → Collect PKs in buffer → Sort PKs in index order → PKs in PK order → Sweep-read rows → Table
MySQL 5.5 vs MySQL 5.6: DBT-3 Queries using DS-MRR

DBT-3, Scale 10 (23 GB)

innodb_buffer_pool_size= 1 GB (disk-bound)

read_rnd_buffer_size = 4 MB

Query Execution Time Relative to MySQL 5.5

<table>
<thead>
<tr>
<th>Q3</th>
<th>Q4</th>
<th>Q10</th>
<th>Q14</th>
<th>Q15</th>
</tr>
</thead>
<tbody>
<tr>
<td>MySQL 5.5</td>
<td>MySQL 5.6</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

0 % 20 % 40 % 60 % 80 % 100 %
DS-MRR

Usage

- Default: Cost-based choice for tables larger than `innodb_buffer_pool_size` (Otherwise: off)
- Force MRR on:
  ```sql
  set optimizer_switch = 'mrr_cost_based=off';
  ```
- Force MRR off:
  ```sql
  set optimizer_switch = 'mrr=off';
  ```
- Configurable size for buffer used to sort keys:
  ```sql
  read_rnd_buffer_size (Default: 256 kB)```
mysql> explain select l_suppkey, sum(l_extendedprice * (1 - l_discount))
    from lineitem where l_shipdate >= '1996-07-01' and l_shipdate <
    date_add('1996-07-01', interval '90' day) group by l_suppkey;

*************************** 1. row ***************************
  id: 1
  select_type: SIMPLE
  table: lineitem
  type: range
  possible_keys: i_l_shipdate
      key: i_l_shipdate
      key_len: 4
      ref: NULL
      rows: 4354466
Extra: Using index condition; Using MRR; Using temporary; Using
    filesort
1 row in set (0.00 sec)
DS-MRR: Sort Buffer Size Matters

DBT-3, Query 15
Scale 10 (23 GB)

innodb_buffer_pool_size = 1 GB (disk-bound)

Varying read_rnd_buffer_size

optimizer_switch settings:
MRR Off:
  mrr=off
MRR Cost-based:
  mrr=on,mrr_cost_based=on
MRR Always on:
  mrr=on,mrr_cost_based=off
MySQL 5.6: Batched Key Access (BKA)

DS-MRR Applied to Join Buffering

Table1

Join buffer

Index

Collect PKs in buffer

Table2

Sweep-read rows

Sort

PKs in join buffer order

PKs in PK order

Table1

Join buffer

Index

Collect PKs in buffer

Table2

Sweep-read rows

Sort

PKs in join buffer order

PKs in PK order
MySQL 5.5 vs MySQL 5.6: Queries using BKA

DBT-3, Scale 10 (23 GB)

innodb_buffer_pool_size= 1 GB (disk-bound)

join_buffer_size = 4 MB

optimizer_switch = 'batched_key_access=on, mrr_cost_based=off'

Query Execution Time Relative to MySQL 5.5

MySQL 5.5 vs MySQL 5.6
Batched Key Access

Usage

- Default: Off
- Force BKA on:
  ```sql
  set optimizer_switch = 'batched_key_access=on,mrr_cost_based=off';
  ```
- Configurable size for buffering keys to sort:
  ```sql
  join_buffer_size (Default: 256 kB)
  ```
Batched Key Access

EXPLAIN

mysql> explain select sum(l_extendedprice* (1 - l_discount)) as revenue from lineitem, part where p_partkey = l_partkey and p_brand = 'Brand#22' and l_quantity >= 6 and p_size between 1 and 5;
<p>|</p>
<table>
<thead>
<tr>
<th>id</th>
<th>select_type</th>
<th>table</th>
<th>type</th>
<th>possible_keys</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>part</td>
<td>ALL</td>
<td>PRIMARY</td>
</tr>
<tr>
<td></td>
<td></td>
<td>NULL</td>
<td>NULL</td>
<td>200000</td>
</tr>
<tr>
<td></td>
<td></td>
<td>NULL</td>
<td>NULL</td>
<td>Using where</td>
</tr>
<tr>
<td>1</td>
<td>SIMPLE</td>
<td>lineitem</td>
<td>ref</td>
<td>i_l_suppkey_partkey,i_l_partkey</td>
</tr>
<tr>
<td></td>
<td></td>
<td>NULL</td>
<td>NULL</td>
<td>Using where;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>NULL</td>
<td>NULL</td>
<td>Using join buffer (Batched Key Access)</td>
</tr>
</tbody>
</table>

2 rows in set (0.00 sec)
DBT-3, Query 2
Scale 10 (23 GB)

```
innodb_buffer_pool_size= 1 GB  
(disk-bound)
```

Varying join_buffer_size

```
optimizer_switch =  
'batched_key_access=on,  
mrr_cost_based=off'
```

Batched Key Access: Buffer Size Matters

![Graph showing the impact of join buffer size on query time with Batched Key Access (BKA) on and off. The graph indicates a significant reduction in query time as the join buffer size increases.]
MySQL 5.6: Subquery Improvements

Optimize IN subqueries

```sql
select o_orderdate, o_totalprice
from orders
where o_orderkey in (select l_orderkey
from lineitem
where l_quantity > 49);
```

- New optimizations in MySQL 5.6:
  - Subquery Materialization
  - Semi-join
Subquery Materialization

1. Execute subquery and store result in a temporary table with unique index (For quick look-up and duplicate removal.)
2. Execute outer query and check for matches in temporary table.

```sql
select o_orderdate, o_totalprice
from orders
where o_orderkey in (select l_orderkey
from lineitem
group by l_orderkey
having sum(l_quantity) > 313);
```

Materialize
MySQL 5.5 vs MySQL 5.6: Subquery Materialization

DBT-3, Scale 10 (23 GB)

innodb_buffer_pool_size= 24 GB (CPU-bound)

For Q20:
optimizer_switch = 'semijoin=off;
subquery_materialization_cost_based=off'

MySQL 5.5 vs MySQL 5.6:
Subquery Materialization

Q18:
MySQL 5.5: ~37 years?
MySQL 5.6: 69 seconds
MySQL 5.6: Semi-join

- Convert subquery to inner join, BUT
  - Need some way to remove duplicates
- Different strategies for duplicate removal:
  - FirstMatch (equivalent to traditional subquery execution)
  - LooseScan (index scan, skip duplicates)
  - Materialization: MatLookup (like subquery materialization), MatScan (materialized table is first in join order)
  - Duplicate WeedOut (insert result rows of semi-join query into temporary table with unique index; duplicate rows will be rejected. Any join order.)
- If duplicate removal is not necessary:
  - Table pull-out
Semi-join, cont.

- Main advantage:
  - Opens up for more optimal "join orders".
  - Example:
    ```sql
    select o_orderdate, o_totalprice
    from orders
    where o_orderkey in (select l_orderkey
                          from lineitem
                          where l_shipDate='1996-09-30');
    ```
    Will process less rows if starting with `lineitem` instead of `orders`

- Restriction:
  - Cannot use semi-join if subquery contains union or aggregation
MySQL 5.6: Semi-join: Example 1

```
select o_totalprice
from orders
where o_orderkey in
(select l_orderkey
 from lineitem
where l_shipdate = '1996-09-30');
```

DBT-3, Scale 10 (23 GB)
inoddb_buffer_pool_size= 24 GB (CPU-bound)
select 
  sum(l_quantity * l_extendedprice) 
from lineitem 
where l_orderkey in 
  (select o_orderkey 
   from orders 
   where o_orderdate = '1996-09-30');

DBT-3, Scale 10 (23 GB)
inodb_buffer_pool_size= 24 GB (CPU-bound)
MySQL 5.6: Semi-join: Example 3

```sql
select s_name, s_address
from supplier
where s_suppkey in
(select ps_suppkey
from partsupp, part
where ps_partkey=p_partkey
and p_name like 'grey%'
and ps_availqty > 9990);
```

DBT-3, Scale 10 (23 GB)

innodb_buffer_pool_size= 24 GB (CPU-bound)
Semi-join

Usage

- Default: All IN sub-queries that do not contain aggregation or union are converted to semi-join

- Disable semi-join conversion:
  ```
  set optimizer_switch = 'semijoin=off';
  ```

- Disable individual semi-join strategies:
  ```
  set optimizer_switch = 'firstmatch=off';
  set optimizer_switch = 'loosescan=off';
  set optimizer_switch = 'materialization=off';
  ```

- Force traditional IN-to-EXIST evaluation:
  ```
  set optimizer_switch = 'semijoin=off,materialization=off';
  ```
MySQL 5.6: Index Condition Pushdown (ICP)

DBT3 Query 6: Forecasting Revenue Change Query

```sql
select sum(l_extendedprice * l_discount) as revenue
from lineitem
  force index(j_l_shipdate_discount_quantity)
where l_shipdate >= '1994-01-01'
  and l_shipdate < date_add('1994-01-01', interval '1' year)
  and l_discount between 0.09 - 0.01 and 0.09 + 0.01
  and l_quantity < 24;
```

Need force index to get ICP for this query

Index range scan criteria

Conditions evaluated during index scan
MySQL 5.6: Index Condition Pushdown

DBT-3, Query 6
Scale 10 (23 GB)

innodb_buffer_pool_size = 24 GB (CPU-bound)

optimizer_switch settings:
index_condition_pushdown = on/off

![Query Time (seconds) vs. Scan Type](chart)

- Table Scan
- Index Scan (ICP off)
- Index Scan (ICP on)
**Index Condition Pushdown**

**EXPLAIN**

```sql
mysql> explain select sum(l_extendedprice * l_discount) as revenue from lineitem force index (i_l_shipdate_discount_quantity) where l_shipdate >= '1994-01-01' and l_shipdate < date_add( '1994-01-01' , interval '1' year) and l_discount between 0.09 - 0.01 and 0.09 + 0.01 and l_quantity < 2

*************************** 1. row ***************************
id: 1
select_type: SIMPLE
table: lineitem
type: range
possible_keys: i_l_shipdate_discount_quantity
    key: i_l_shipdate_discount_quantity
    key_len: 16
    ref: NULL
    rows: 18940908
    Extra: Using index condition
1 row in set (0.00 sec)
```
MySQL> explain FORMAT=JSON select sum(l_extendedprice * l_discount) as revenue from lineitem force index (i_l_shipdate_discount_quantity) where l_shipdate >= '1994-01-01' and l_shipdate < date_add('1994-01-01', interval '1' year) and l_discount between 0.09 - 0.01 and 0.09 + 0.01 and l_quantity < 24;

| {
|   "query_block": {
|     "select_id": 1,
|     "table": {
|       "table_name": "lineitem",
|       "access_type": "range",
|       ...
|       "filtered": 100,
|       "index_condition": "((`dbt3`.`lineitem`.`l_shipDATE` >= '1994-01-01')
|                               and (`dbt3`.`lineitem`.`l_shipDATE` < ('1994-01-01' + interval '1' year))
|                               and (`dbt3`.`lineitem`.`l_discount` between (0.09 - 0.01) and (0.09 + 0.01))
|                               and (`dbt3`.`lineitem`.`l_quantity` < 24))"
|   }
| }
MySQL 5.6: More Optimizer Improvements

- ORDER BY with LIMIT optimization
- Delayed Materialization of Derived Tables
- Extended secondary keys (InnoDB)
- Reduced optimization time for large IN-lists
- Reduced optimization time for many-table joins
- Reduced space usage for large temporary tables with VARCHAR
- Speed-up of information schema queries
- EXPLAIN for INSERT, UPDATE, DELETE
- Structured EXPLAIN (JSON format)
- Optimizer trace
More information

- My blog: 
  - http://oysteing.blogspot.com/
- Optimizer team blog: 
  - http://mysqloptimizerteam.blogspot.com/
- What’s new in MySQL 5.6: 
The preceding 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.