Advanced query optimization techniques on large queries

Peter Boros
4/25/13
Santa Clara, CA
Agenda

• Showing the title slide
• Going through this agenda
• Prerequisites
• Case study #1
• Case study #2
• Case study #3
• Case study #4
This talk is not about

- SQL basics
- Reading explain
- Indexing basics
Case Study #1
#1: Query

- Query runs in the 10 seconds range
- According to explain output it can use some indexes
- Mostly single-column indexes were defined

```
SELECT t.id, ts.meta1, lt.meta2
FROM things t
INNER JOIN things_stuff ts
  ON t.fk_things_stuff_id = ts.id
INNER JOIN yet_an_other_table yat
  ON t.fk_yat_id = yat.id
WHERE t.active = 1
AND t.type != 'guest'
AND t.title like '%the%'
AND LOWER(t.username) like '%e%'
AND t.grants & 1 != 0
AND t.attribute1=3;
```
#1: Query

- Rows examined in the 100k range
- Rows returned is single digit
- This means something is selective, but most likely not indexed

```sql
SELECT t.id, ts.meta1, lt.meta2
FROM things t
INNER JOIN things_stuff ts
  ON t.fk_things_stuff_id = ts.id
INNER JOIN yet_an_other_table yat
  ON t.fk_yat_id = yat.id
WHERE t.active = 1
AND t.type != 'guest'
AND t.title like '%the%'
AND LOWER(t.username) like '%e%'
AND t.grants & 1 != 0
AND t.attribute1=3;
```
#1: issue identification

<table>
<thead>
<tr>
<th>Row</th>
<th>id</th>
<th>select_type</th>
<th>table</th>
<th>type</th>
<th>key</th>
<th>key_len</th>
<th>rows</th>
<th>Extra</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>SIMPLE</td>
<td>t</td>
<td>const</td>
<td>idx_active</td>
<td>4</td>
<td>332873</td>
<td>Using where</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>SIMPLE</td>
<td>ts</td>
<td>eq_ref</td>
<td>PRIMARY</td>
<td>4</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>eq_ref</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

This is the issue, the rest of the tables are just some metadata fetching (type is eq_ref)
#1: finding the filtering clause

- In the original query, filtering in the WHERE clause are all for table t (things)
- Find out if any of them are selective alone (it's possible that they are not, but some combination of them will be selective)

SELECT COUNT(t.id) FROM things t WHERE t.active=1;

<table>
<thead>
<tr>
<th>COUNT(1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>168520</td>
</tr>
</tbody>
</table>

1 row in set (10.15 sec)
#1: finding the filtering clause

```sql
SELECT COUNT(t.id) FROM things t WHERE t.status!='guest';
+----------+
| COUNT(1) |
+----------+
|   284582 |
+----------+
1 row in set (10.13 sec)

SELECT COUNT(t.id) FROM things t WHERE t.title LIKE '%the%';
+----------+
| COUNT(1) |
+----------+
|   173895 |
+----------+
1 row in set (10.12 sec)

SELECT COUNT(t.id) FROM things t WHERE lower(t.username) like '%e%';
+----------+
| COUNT(1) |
+----------+
|   190345 |
+----------+
1 row in set (10.15 sec)
```
### #1: finding the filtering clause

```sql
SELECT COUNT(t.id) FROM things t WHERE t.grants & 1 != 0;
```

<table>
<thead>
<tr>
<th>COUNT(1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
</tr>
</tbody>
</table>

1 row in set (10.13 sec)

- This is selective
- In the original table definition the grants column has an index on it, but it's not used (the index on the not too selective active column is used instead)
- Data type of grants in TINYINT
#1: reason for slowness

SELECT COUNT(t.id) FROM things t WHERE t.grants & 1 != 0;

<table>
<thead>
<tr>
<th>COUNT(1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
</tr>
</tbody>
</table>

1 row in set (10.13 sec)

- The & is bitwise AND operator
  - On the left hand side of comparison it's not the column itself, which is indexed, but some kind of function of the column is compared, which is not indexed.
#1: TINYINT column used as bitmap

- The grants in this case used as 1 raw byte (8 bits), 1 is a mask we compare to

- A common case for this is storing privileges

```
<table>
<thead>
<tr>
<th></th>
<th>Privilege “A”</th>
<th>Privilege “H”</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
```

Does some document / user / page has privilege “H”?

```
1 0 1 0 0 0 0 1
&
0 0 0 0 0 0 0 1
```

The result is non-0 when the appropriate privilege bit is 1.
#1: solution

- The solution itself is very space efficient, but the data is not accessible.
- If only privilege “H” is checked frequently, just create a privilege “H” column, which is indexed and store data redundantly.
- If other checks are frequent, use separate columns for each privilege.
- Other antipatterns in the query, which are not making it slower.
Case Study #2
#2: Query

- Query runs in the 10 seconds range
- Rows examined is much higher than rows returned

```sql
SELECT ta.*,
    tb.attr1,
    tb.attr2,
    tb.attr3,
    tb.attr4,
    tc.attr1
FROM table_a ta,
    table_b tb,
    table_c tc
WHERE tb.ID = ta.tb_id
    AND tc.attr1 = ta.attr1
    AND tc.attr2 = 1
    AND ta.attr3 = 'non_selective'
    AND ta.attr4 = 715988
    AND ta.attr5 BETWEEN 1 AND 10
ORDER BY ta.attr6 ASC
```
#2: Explain

<table>
<thead>
<tr>
<th>id: 1</th>
<th>table: ta</th>
</tr>
</thead>
<tbody>
<tr>
<td>select_type: SIMPLE</td>
<td></td>
</tr>
<tr>
<td>type: ref</td>
<td></td>
</tr>
<tr>
<td>possible_keys: idx_attr4,...,idx_attr3,...</td>
<td></td>
</tr>
<tr>
<td>key: idx_attr3</td>
<td></td>
</tr>
<tr>
<td>key_len: 202</td>
<td></td>
</tr>
<tr>
<td>ref: const</td>
<td></td>
</tr>
<tr>
<td>rows: 100280</td>
<td></td>
</tr>
<tr>
<td>Extra: Using where; Using filesort</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>id: 1</th>
<th>table: tb</th>
</tr>
</thead>
<tbody>
<tr>
<td>type: eq_ref</td>
<td></td>
</tr>
<tr>
<td>key: PRIMARY</td>
<td></td>
</tr>
<tr>
<td>rows: 1</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>id: 1</th>
<th>table: tc</th>
</tr>
</thead>
<tbody>
<tr>
<td>type: ref</td>
<td></td>
</tr>
<tr>
<td>key: idx_attr1</td>
<td></td>
</tr>
<tr>
<td>rows: 2</td>
<td></td>
</tr>
</tbody>
</table>
#2: Show indexes and data types

```sql
mysql> SHOW INDEX FROM table_a WHERE Key_name = "attr4" OR Key_name = "attr3"
```

```
*************************** 1. row ***************************
Table: table_a
Key_name: attr4
Seq_in_index: 1
Column_name: attr4
Cardinality: 523608

*************************** 2. row ***************************
Table: table_a
Key_name: attr3
Seq_in_index: 1
Column_name: attr4
Cardinality: 21
```

```
`attr3` varchar(200) DEFAULT NULL,
`attr4` varchar(200) DEFAULT NULL
```
#2: solution

```sql
SELECT ta.*,
tb.attr1,
tb.attr2,
tn.attr3,
tb.attr4,
tc[attr1]
FROM table_a ta,
table_b tb,
table_c tc
WHERE tb.ID = ta.tb_id
AND tc.attr1 = ta.attr1
AND tc.attr2 = 1
AND ta.attr3 = 'non_selective'
AND ta.attr4 = 715988
AND ta.attr5 BETWEEN 1 AND 10
ORDER BY a.date_recorded ASC
```

```sql
SELECT ta.*,
tb.attr1,
tb.attr2,
tn.attr3,
tb.attr4,
tc[attr1]
FROM table_a ta,
table_b tb,
table_c tc
WHERE tb.ID = ta.tb_id
AND tc.attr1 = ta.attr1
AND tc.attr2 = 1
AND ta.attr3 = 'non_selective'
AND ta.attr4 = 715988
AND ta.attr5 BETWEEN 1 AND 10
ORDER BY a.date_recorded ASC
```

After this change, the right index is used (changing column type could be better)
#2: conclusion

- Always quote strings, never quote numbers
- Similar thing can happen when there is a data type mismatch between joined tables
  - For example an id is in a INT column is one table, VARCHAR(x) in the other, join won't use indexes on them
#3: Query

```
SELECT * FROM things WHERE ((thing_id=87459234 AND thing_flag1 = 1)
OR (thing_other_thing_id=87459234 AND thing_flag2 = 1)) AND
(thing_id=87459234 OR other_thing_id=87459234) AND (thing_id <>
3846571 AND other_thing_id <> 3846571) AND (thing_prop1 IS NOT NULL)
AND (thing_rangedate >= '2012-12-01') OR (thing_rangedate_min >
'2012-12-03' AND thing_rangedate_max < '2012-12-04')
AND (NOT EXISTS (SELECT 1 FROM thing_stuffs WHERE fk_things_id =
things.id AND important_num BETWEEN 1 and 4 AND other_range_value
between 1 and 3));
```

If a query is big enough, always start with formatting the query.
SELECT * 
FROM things 
WHERE 
((thing_id=87459234 AND thing_flag1 = 1) 
OR (thing_other_thing_id=87459234 AND thing_flag2 = 1)) 
AND (thing_id=87459234 OR other_thing_id=87459234) 
AND (thing_id <> 3846571 AND other_thing_id <> 3846571) 
AND (thing_prop1 IS NOT NULL) 
AND (thing_rangedate >= '2012-12-01') 
OR (thing_rangedate_min > '2012-12-03' 
AND thing_rangedate_max < '2012-12-04') 
AND (NOT EXISTS 
(SELECT 1 FROM thing_stuffs 
WHERE fk_things_id = things.id 
AND important_num BETWEEN 1 and 4 
AND other_range_value BETWEEN 1 and 3));
id: 1
select_type: PRIMARY
table: things
type: ref
possible_keys: idx_thing_id,idx_other_thing_id,idx_prop1
  key: idx_thing_id
key_len: 4
  rows: 23917933
Extra: Using where

id: 2
select_type: DEPENDENT SUBQUERY
table: thing_stuffs
type: subquery
possible_keys: PRIMARY,fk_things_id
  key: PRIMARY
key_len: 4
  rows: 1
Extra: Using where
2 rows in set (0.01 sec)
SELECT *  
FROM things  
WHERE  
AND (NOT EXISTS  
(SELECT 1 FROM thing_stuffs  
    WHERE fk_things_id = things.id  
    AND important_num BETWEEN 1 and 4  
    AND other_range_value BETWEEN 1 and 3));

SELECT t.*  
FROM things t  
LEFT JOIN thing_stuffs ts ON ts.fk_things_id = things.id  
WHERE ts.fk_things_id IS NULL  
AND ts.important_num BETWEEN 1 AND 4  
AND ts.other_range_value BETWEEN 1 AND 3
#3: Explain after rewrite

*************** 1. row ***************

id: 1
select_type: SIMPLE
table: ts
type: ref
possible_keys: idx_important_num_other_range
key: idx_important_num_other_range
key_len: 4
ref: const
rows: 93854
Extra: Using where

Composite index on important_num and other_range_value

*************** 2. row ***************

id: 1
select_type: SIMPLE
table: t
type: ref
possible_keys: PRIMARY
key: PRIMARY
key_len: 4
ref: ts.fk_things_id
rows: 5
Extra: Using where

2 rows in set (0.01 sec)
#3: Examining further

Still more rows examined than returned.

```sql
possible_keys: idx_important_num_other_range
    key: idx_important_num_other_range
key_len: 4
rows: 93854
```

The index is a composite index, however only important_num is used

```sql
mysql> SELECT COUNT(1) FROM thing_stuffs WHERE important_num BETWEEN 1 AND 4;
+----------+
| count(1) |
+----------+
|     87520 |
+----------+
1 row in set (0.15 sec)
```

```sql
mysql> SELECT COUNT(1) FROM thing_stuffs WHERE other_range_value BETWEEN 1 AND 3;
+----------+
| count(1) |
+----------+
|   134526 |
+----------+
1 row in set (0.17 sec)
```
#3: Examining further

```
mysql> SELECT COUNT(1) FROM thing_stuff WHERE important_num BETWEEN 1 AND 4
     AND other_range_value BETWEEN 1 AND 3;
+----------+
| count(1) |
+----------+
|   125    |
+----------+
1 row in set (0.15 sec)
```

- Together they are selective
- Both of them are range conditions, so they can't be used together in a composite index
- Unless we rewrite
#3: Second rewrite

```
SELECT COUNT(1) from thing_stuff WHERE important_num IN (1,2,3,4) AND other_range_value IN (1,2,3);
*************************** 1. row ***************************
    id: 1
  select_type: SIMPLE
    table: ts
    type: ref
      key: idx_important_num_other_range
    key_len: 8
      ref: const
    rows: 125
Extra: Using where
*************************** 2. row ***************************
    id: 1
  select_type: SIMPLE
    table: t
    type: ref
      key: PRIMARY
    key_len: 4
      ref: ts.fk_things_id
    rows: 5
Extra: Using where
2 rows in set (0.01 sec)
```
The query is still slow.

A lot more rows examined than returned.

Checking explain the relevant part is that the query uses an index merge.

Extra: Using union(idx_thing_id,idx_other_thing_id); Using where

```sql
SELECT t.*
FROM things t
LEFT JOIN thing_stuffs ts ON ts.fk_things_id = t.id
WHERE (
  ((t.thing_id=87459234 AND t.thing_flag1 = 1)
  OR (t.other_thing_id=87459234 AND t.thing_flag1 = 1))
  AND (t.thing_id=87459234 OR t.other_thing_id=87459234)
  AND (t.thing_id <> 3846571
  AND t.other_thing_id <> 3846571)
  AND (t.thing_prop1 IS NOT NULL)
  AND (t.thing_rangedate >= '2012-12-01')
  OR (t.thing_rangedate_min > '2012-12-03'
  AND t.thing_rangedate_max < '2012-12-04')
  AND ts.important_num IN (1,2,3,4)
  AND ts.other_range_value IN (1,2,3)
  AND ts.fk_things_id IS NULL;
```
SELECT * FROM things
WHERE (thing_id=87459234 OR other_thing_id=87459234)
AND thing_id <> 3846571 AND other_thing_id <> 3846571;

*************************** 1. row ***************************
    id: 1
  select_type: SIMPLE
    table: things
      type: index_merge
possible_keys: idx_thing_id,idx_other_thing_id
    key: idx_thing_id,idx_other_thing_id
key_len: 4,4
    ref: NULL
  rows: 59834
 Extra: Using union(idx_thing_id,idx_other_thing_id); Using where
1 row in set (0.00 sec)
#3: Rewritten query

- Index merge is an expensive operation
  - Especially if indexes are large
- UNION always create an on-disk temp table
- Just turning off index merge could be enough for a significant performance advantage

```
SELECT *
FROM things
WHERE thing_id=87459234
AND other_thing_id<>3846571
UNION
SELECT *
FROM things
WHERE other_thing_id=87459234
AND thing_id<>3846571;
```

mysql> select @@optimizer_switch
********** 1. row **********
@@optimizer_switch:
index_merge=on,index_merge_union=on,index_merge_sort_union=on,index_merge_intersection=on,engine_condition_pushdown=on
1 row in set (0.00 sec)
#3: Putting it together

```sql
SELECT t.* FROM things t
LEFT JOIN things_stuff ts
ON ts.fk_things_id = t.id
WHERE t.thing_id=87459234
AND t.other_thing_id<>3846571
AND t.thing_flag1=1
AND t.thing_prop IS NOT NULL
AND (t.thing_rangedate_min >= '2012-12-01')
AND ((t.thing_rangedate_min <= '2012-12-03'
AND t.thing_rangedate_max > '2012-12-03')
OR (t.thing_rangedate_min < '2012-12-04'
AND t.thing_rangedate_max >= '2012-12-04'))
AND ts.fk_things_id IS NULL
AND ts.important_num IN (1,2,3,4)
AND ts.other_range_value IN (1,2,3)
UNION

SELECT t.* FROM things t
LEFT JOIN things_stuff ts
ON ts.fk_things_id = t.id
WHERE t.thing_id=87459234
AND t.other_thing_id<>3846571
AND t.thing_flag2=1
AND t.thing_prop1 IS NOT NULL
AND (t.thing_rangedate_min >= '2012-12-01')
AND ((t.thing_rangedate_min <= '2012-12-03'
AND t.thing_rangedate_max > '2012-12-03')
OR (t.thing_rangedate_min < '2012-12-04'
AND t.thing_rangedate_max >= '2012-12-04'))
AND ts.fk_things_id IS NULL
AND ts.important_num IN (1,2,3,4)
AND ts.other_range_value IN (1,2,3);
```
Case Study #4
#4: Summary

- Very large, 20 table join
- Explain looks ok
- Query runs for almost a hour
- Handler status variables are way off
#4: Query

```sql
SELECT tmain.id, tmain.date,
CONCAT_WS(" ", COALESCE(m1.col1, 'n/a'), COALESCE(m2.col2, 'n/a'))
/* a lot more columns selected */)
FROM main_table as tmain ah
LEFT JOIN (metadata_table1 m1 JOIN metadata_table2 m2
ON m1.attr1 = m2.attr1 AND m1.const = 'some text')
on tmain.id = m1.tmain_id AND tmain.important_flag=1
AND ((m1.attr11='a') OR (m2.attr12 IS NOT NULL) AND 3>=m1.attr13 AND 5<m2.attr14)
OR m1.attr15 IS NOT NULL AND m2.attr16 BETWEEN 1 AND 10
LEFT JOIN (metadata_table3 m3 JOIN metadata_table4 m4
ON m3.attr1 = m4.attr1 AND m1.const = 'some text')
on tmain.id = m4.tmain_id AND tmain.important_flag=1
AND ((m3.attr11='a') OR (m4.attr12 IS NOT NULL) AND 3>=m3.attr13 AND 5<m4.attr14)
OR m3.attr15 IS NOT NULL AND m4.attr16 BETWEEN 11 AND 20
LEFT JOIN (metadata_table5 m5 JOIN metadata_table2 m6
ON m5.attr1 = m6.attr1 AND m1.const = 'some text')
on tmain.id = m5.tmain_id AND tmain.important_flag=1
AND ((m5.attr11='a') OR (m6.attr12 IS NOT NULL) AND 3>=m5.attr13 AND 5<m6.attr14)
OR m1.attr15 IS NOT NULL AND m2.attr16 BETWEEN 21 AND 30
LEFT JOIN (metadata_table1 m7 JOIN metadata_table2 m8
ON m7.attr1 = m8.attr1 AND m7.const = 'some text')
on tmain.id = m7.tmain_id AND tmain.important_flag=1
AND ((m7.attr11='a') OR (m8.attr12 IS NOT NULL) AND 3>=m7.attr13 AND 5<m8.attr14)
OR m7.attr15 IS NOT NULL AND m8.attr16 BETWEEN 31 AND 40
/* 6 more joins exactly like the previous ones... */
WHERE tmain.important_flag = 1;
```
<table>
<thead>
<tr>
<th>id</th>
<th>select_type</th>
<th>table</th>
<th>type</th>
<th>rows</th>
<th>Extra</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>SIMPLE</td>
<td>tmain</td>
<td>ref</td>
<td>1786</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>SIMPLE</td>
<td>m1</td>
<td>ref</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>SIMPLE</td>
<td>m2</td>
<td>eq_ref</td>
<td>1</td>
<td>Using index</td>
</tr>
<tr>
<td>1</td>
<td>SIMPLE</td>
<td>m3</td>
<td>ref</td>
<td>4</td>
<td>Using index</td>
</tr>
<tr>
<td>1</td>
<td>SIMPLE</td>
<td>m4</td>
<td>eq_ref</td>
<td>1</td>
<td>Using index</td>
</tr>
<tr>
<td>1</td>
<td>SIMPLE</td>
<td>m19</td>
<td>ref</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>SIMPLE</td>
<td>m20</td>
<td>eq_ref</td>
<td>1</td>
<td>Using index</td>
</tr>
</tbody>
</table>
#4: Handler operations

Run FLUSH STATUS before the query to reset the counters.

```
FLUSH STATUS;
QUERY;
SHOW STATUS LIKE 'Handler%';
```

```
mysql> show status like 'Handler%';
+-----------------------------+-----------+
| Variable_name               | Value     |
+-----------------------------+-----------+
| Handler_commit              |         1 |
| Handler_delete              |         0 |
| Handler_discover            |         0 |
| Handler_prepare             |         0 |
| Handler_read_first          |         0 |
| Handler_read_key            |     61341 |
| Handler_read_last           |         0 |
| 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 |
| Handler_read_next           | 198008370 |
+-----------------------------+-----------+
```
#4: Problematic part

```sql
LEFT JOIN
(metadata_table1 m7 JOIN metadata_table2 m8 ON m7.attr1 = m8.attr1
AND m7.const = 'some text')
ON tmain.id = m7.tmain_id AND tmain.important_flag=1;

SELECT AVG(LENGTH(m1.somefield))
FROM metadata_table1 m1
JOIN metadata_table2 m2
ON m1.attr1 = m2.attr1
AND m1.const='some text'
```

```
+----+-------------+----------+--------+-------+-------------+
<table>
<thead>
<tr>
<th>id</th>
<th>select_type</th>
<th>table</th>
<th>type</th>
<th>rows</th>
<th>Extra</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>SIMPLE</td>
<td>m1</td>
<td>ref</td>
<td>19764</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>SIMPLE</td>
<td>m2</td>
<td>eq_ref</td>
<td>102</td>
<td>Using index</td>
</tr>
</tbody>
</table>
+----+-------------+----------+--------+-------+-------------+
```

MySQL executes the join like this. One “problematic part takes roughly 3 minutes, so it adds up.
#4: Rewrite

```
SELECT tmain.id
FROM main_table tmain
LEFT JOIN metadata_table1 m1 ON tmain.id=m1.tmain_id
LEFT JOIN metadata_table2 m2 ON m1.attr1 = m2.attr1
AND m1.const = 'some text'
WHERE tmain.important_flag = 1;
```

- In order to leave out () LEFT JOIN has to be used to find non-matching elements.
- Runtime for query is 0.2 sec instead of 3 minutes.

```
+----+-------------+----------+--------+------+-------------+
| id | select_type | table    | type   | rows | Extra       |
+----+-------------+----------+--------+------|-------------+
| 1  | SIMPLE      | tmain    | ref    | 1786 |             |
| 1  | SIMPLE      | m1       | ref    | 4    |             |
| 1  | SIMPLE      | m2       | eq_ref | 1    | Using index |
+----+-------------+----------+--------+------|-------------+
```
After applying the tuning to the original query, I expected it to execute in a few seconds. It executed in 4 minutes instead. Used Handler operations looked good. Took a look at explain, and explain took 4 minutes also.

- This means the optimizer spends time in constructing the query plan.
- Solution: STRAIGHT_JOIN and/or optimizer_search_depth
Q & A
Thank you for attention.