Rewrite Your Complex MySQL Queries for Better Performance

Øystein Grøvlen

Senior Staff Engineer
Alibaba Cloud
Introduction to Subqueries

Derived Tables

Scalar Subqueries

IN/EXISTS subqueries

Use Window Functions

Optimizer Hints
Types of Subqueries

- **Scalar subqueries**
  - Returns maximum one row
  - May be used most places where a value can be used
  - Examples:
    
    ```
    SELECT a1, (SELECT AVG(b2) FROM t2) FROM t1;
    SELECT a1 FROM t1 WHERE a1 < (SELECT AVG(b2) FROM t2);
    ```

- **Non-scalar subqueries**
  - May return multiple rows
  - Examples:
    
    ```
    SELECT b1, avg_b2 FROM (SELECT b1, AVG(b2) avg_b2 FROM t2 GROUP BY b1) dt;
    SELECT a1 FROM t1 WHERE a1 IN (SELECT b2 FROM t2);
    ```
Correlated Subqueries

- Subquery refers columns of outer query
- Examples

```sql
SELECT a1, (SELECT b2 FROM t2 WHERE b1 = t1.a1) FROM t1;

SELECT a1 FROM t1 WHERE a1 < (SELECT AVG(b2) FROM t2 WHERE b1 = t1.a1);

SELECT a1 FROM t1 WHERE NOT EXISTS (SELECT 1 FROM t2 WHERE b1 = t1.a1);

SELECT t1.a1, dt.b2 FROM t1, LATERAL (SELECT b2 FROM t2 WHERE b2 = t1.a1) dt;
```

New in MySQL 8.0.14
Nested Subqueries
TPC-H Query 20: Potential Part Promotion Query

```sql
SELECT s_name, s_address
FROM supplier, nation
WHERE s_suppkey IN (  
  SELECT ps_suppkey
  FROM partsupp
  WHERE ps_partkey IN (  
    SELECT p_partkey FROM part WHERE p_name LIKE 'dodger%'  
  )  
  AND ps_availqty > (  
    SELECT 0.5 * SUM(l_quantity)
    FROM lineitem
    WHERE l_partkey = ps_partkey AND l_suppkey = ps_suppkey  
    AND l_shipdate >= '1994-01-01'
    AND l_shipdate < DATE_ADD('1994-01-01', INTERVAL '1' YEAR)
  )  
  )  
  AND n_name = 'INDIA'
ORDER BY s_name;
```
Execution of Subqueries

• Naïve approach
  • Execute subquery for each row of the outer query
  • May use indexes to speed up correlated queries
  • Example: `SELECT a1 FROM t1 WHERE a1 < (SELECT AVG(b2) FROM t2 WHERE t2.b1 = t1.a1);`

• Optimizations / Query transformations
  • Merge subquery into outer query
  • Materialization of non-correlated subqueries
  • Semi-join/Anti-join for (NOT) IN/EXISTS
  • Rewrite to MIN/MAX for queries like `<CompOp> ALL/ANY (SELECT …)`
    `SELECT a1 FROM t1 WHERE a1 > ALL (SELECT b2 FROM t2);`
  • New query transformations in 8.0
Introduction to Subqueries

Derived Tables

Scalar Subqueries

IN/EXISTS subqueries

Use Window Functions

Optimizer Hints
Execution of Derived Tables

- Materialization

```
SELECT b1, avg_b2 FROM (SELECT b1, AVG(b2) avg_b2 FROM t2 GROUP BY b1) dt;
```

1. Store the result of the subquery in a temporary table (may create indexes on temporary table if useful)

```
CREATE TEMPORARY TABLE dt AS (SELECT b1, AVG(b2) avg_b2 FROM t2 GROUP BY b1);
```

2. Execute the main query using the temporary table

```
SELECT b1, avg_b2 FROM AS dt;
DROP TEMPORARY TABLE dt;
```

- Merge into outer query (MySQL 5.7)
  - Handle derived tables the same way as views
  - Not supported for queries that contain aggregation functions, GROUP BY, LIMIT, UNION, DISTINCT, ...
### Materialized Derived Tables

Do not SELECT more data that needed by outer query

#### Query 1

```sql
mysql> SELECT AVG(o_totalprice) FROM 
    (SELECT * FROM orders ORDER BY o_totalprice DESC LIMIT 100000) dt;
```

<table>
<thead>
<tr>
<th>AVG(o_totalprice)</th>
</tr>
</thead>
<tbody>
<tr>
<td>398185.986158</td>
</tr>
</tbody>
</table>

1 row in set (24.65 sec)

#### Query 2

```sql
mysql> SELECT AVG(o_totalprice) FROM 
    (SELECT o_totalprice FROM orders ORDER BY o_totalprice DESC LIMIT 100000) dt;
```

<table>
<thead>
<tr>
<th>AVG(o_totalprice)</th>
</tr>
</thead>
<tbody>
<tr>
<td>398185.986158</td>
</tr>
</tbody>
</table>

1 row in set (8.18 sec)
SELECT b1, avg_b2
FROM (SELECT b1, AVG(b2) avg_b2 FROM t2 GROUP BY b1) dt
WHERE b1 < 10;

SELECT b1, avg_b2
FROM (SELECT b1, AVG(b2) avg_b2 FROM t2 WHERE b1 < 10 GROUP BY b1) dt;

SELECT b1, avg_b2
FROM (SELECT b1, AVG(b2) avg_b2 FROM t2 GROUP BY b1)
WHERE avg_b2 > 90;

SELECT b1, avg_b2
FROM (SELECT b1, AVG(b2) avg_b2 FROM t2 GROUP BY b1)
HAVING avg_b2 > 90)

WL#8084: Condition pushdown for materialized derived tables (MySQL 8.0.22)
Materialized Derived Tables

MySQL 8.0.22: Condition pushdown for materialized derived tables (and views)

```
EXPLAIN FORMAT=TREE SELECT b1, avg_b2
FROM (SELECT b1, AVG(b2) avg_b2 FROM t2 GROUP BY b1) dt
WHERE b1 < 10;
```

8.0.21 (1.22 ms):
- Filter: (dt.b1 < 10)
  - Table scan on dt
    - Materialize
      - Table scan on <temporary>
        - Aggregate using temporary table
          - Table scan on t2  (cost=103.65 rows=1024)

8.0.22 (0.85 ms):
- Table scan on dt  (cost=40.86 rows=341)
  - Materialize
    - Table scan on <temporary>
      - Aggregate using temporary table
        - Filter: (t2.b1 < 10)  (cost=103.65 rows=341)
          - Table scan on t2  (cost=103.65 rows=1024)
Merged Derived Tables
Not always optimal

SELECT *
FROM part p1
JOIN (SELECT * FROM part WHERE p_type LIKE '%STEEL%') p2 ON p1.p_name = p2.p_name
WHERE p1.p_type LIKE '%COPPER%';

MySQL 5.5
0.4 seconds

MySQL 5.6

MySQL 5.7
6 minutes
How to Prevent Merging of Derived Tables

• **MySQL 5.7**
  
  • Rewrite derived table so it can not be merged
    ```sql
    SELECT *
    FROM part p1
    JOIN (SELECT * FROM part WHERE p_type LIKE '%STEEL%' LIMIT 1000000) p2 ON p1.p_name=p2.p_name
    WHERE p1.p_type LIKE '%COPPER%';
    ```

• **MySQL 8.0**
  
  • Use NO_MERGE hint
    ```sql
    SELECT /*+ NO_MERGE(p2) */ *
    FROM part p1
    JOIN (SELECT * FROM part WHERE p_type LIKE '%STEEL%') p2 ON p1.p_name = p2.p_name
    WHERE p1.p_type LIKE '%COPPER%';
    ```

• MySQL 8.0.18: Not necessary since using hash join is 35% faster.
Use Common Table Expressions
MySQL 8.0

• Derived tables:

```sql
SELECT *
FROM (SELECT a, b, SUM(c) s FROM t1 GROUP BY a, b) AS d1
JOIN (SELECT a, b, SUM(c) s FROM t1 GROUP BY a, b) AS d2 ON d1.b = d2.a;
```

• Common Table Expressions (CTE):

```sql
WITH d AS (SELECT a, b, SUM(c) s FROM t1 GROUP BY a, b)
SELECT * FROM d AS d1 JOIN d AS d2 ON d1.b = d2.a
```

• Improved Readability

• CTE is only materialized once
CREATE VIEW revenue0(supplier_no, total_revenue) AS
    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;

SELECT s_suppkey, s_name, s_address, s_phone, total_revenue
FROM supplier, revenue0
WHERE s_suppkey = supplier_no
    AND total_revenue = (SELECT MAX(total_revenue) FROM revenue0)
ORDER BY s_suppkey;

WITH revenue0 AS (...
SELECT s_suppkey, s_name, s_address, s_phone, total_revenue
FROM supplier, revenue0
WHERE s_suppkey = supplier_no
    AND total_revenue = (SELECT MAX(total_revenue) FROM revenue0)
ORDER BY s_suppkey;
Introduction to Subqueries

Derived Tables

Scalar Subqueries

IN/EXISTS subqueries

Use Window Functions

Optimizer Hints
Rewrite Scalar Subquery to Derived Table
(Or CTE in MySQL 8.0)

Q1
SELECT SUM(l_extendedprice) / 7.0 AS avg_yearly
FROM lineitem JOIN part ON p_partkey = l_partkey
WHERE p_mfgr = 'Manufacturer#1'
  AND l_quantity < (SELECT 0.2 * AVG(l_quantity) FROM lineitem WHERE l_partkey = p_partkey);

Q2
WITH pq(avg_qty, pk) AS (SELECT 0.2 * AVG(l_quantity), l_partkey
                          FROM lineitem GROUP BY l_partkey)
SELECT SUM(l_extendedprice) / 7.0 AS avg_yearly
FROM lineitem JOIN part ON p_partkey = l_partkey
  JOIN pq ON pq.pk = p_partkey
WHERE p_mfgr = 'Manufacturer#1' AND l_quantity < pq.avg_qty;

Q3
WITH pq(avg_qty, pk) AS (SELECT 0.2 * AVG(l_quantity), l_partkey
                          FROM lineitem JOIN part ON p_partkey = l_partkey
                          WHERE p_mfgr = 'Manufacturer#1' GROUP BY l_partkey)
SELECT SUM(l_extendedprice) / 7.0 AS avg_yearly
FROM lineitem JOIN pq ON pq.pk = l_partkey
WHERE l_quantity < pq.avg_qty;
Rewrite Scalar Subquery to Derived Table

Comparing Execution Times

<table>
<thead>
<tr>
<th></th>
<th>Execution time (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q1</td>
<td>109.30</td>
</tr>
<tr>
<td>Q2</td>
<td>26.60</td>
</tr>
<tr>
<td>Q3</td>
<td>9.60</td>
</tr>
</tbody>
</table>
Rewrite Scalar Subquery to Derived Table

Not always optimal

```
SELECT SUM(l_extendedprice) / 7.0 AS avg_yearly
FROM lineitem JOIN part ON p_partkey = l_partkey
WHERE p_brand = 'Brand#11' AND p_container = 'SM CAN'
    AND l_quantity < (SELECT 0.2 * AVG(l_quantity) FROM lineitem WHERE l_partkey = p_partkey);
```
Automatic Rewrite of Scalar Subqueries to Derived Tables
MySQL 8.0.21

- Only supported for non-correlated scalar subqueries
- Off by default, to enable:
  ```
  SET optimizer_switch='subquery_to-derived=on';
  ```
- Not cost-based, may give worse performance than scalar subqueries
Introduction to Subqueries

Derived Tables

Scalar Subqueries

**IN/EXISTS subqueries**

Use Window Functions

Optimizer Hints
SELECT o_orderdate, o_totalprice FROM orders
WHERE o_orderkey IN (SELECT l_orderkey FROM lineitem WHERE l_shipDate='1996-09-30');

• Semi-join
  • Introduced in MySQL 5.6
  • Inner Join + Duplicate removal
  • Opens up for more optimal "join orders", may process inner tables before outer tables
  • Can not be used if subquery contains UNION or aggregation

• Prefer IN over EXISTS

SELECT o_orderdate, o_totalprice FROM orders
WHERE EXISTS (SELECT 1 FROM lineitem WHERE l_shipDate='1996-09-30' AND o_orderkey = l_orderkey);

• MySQL 8.0.16: Automatic conversion from EXISTS to IN

IN-subqueries
Rewrite IN-Subquery to Derived Table
TPC-H Query 18: Large Volume Customer Query

Subquery SELECTs and GROUPs by l_orderkey, so it will not produce any duplicates:

```sql
SELECT c_name, c_custkey, o_orderkey, o_orderdate, o_totalprice, SUM(l_quantity)
FROM customer JOIN orders ON c_custkey = o_custkey
  JOIN lineitem ON o_orderkey = l_orderkey
WHERE o_orderkey IN ( SELECT l_orderkey FROM lineitem
  GROUP BY l_orderkey HAVING SUM(l_quantity) > 313 )
GROUP BY c_name, c_custkey, o_orderkey, o_orderdate, o_totalprice
ORDER BY o_totalprice DESC, o_orderdate LIMIT 100;
```

Since no duplicate removal is needed, we can rewrite to use INNER JOIN:

```sql
WITH l2 AS (SELECT l_orderkey FROM lineitem
  GROUP BY l_orderkey HAVING SUM(l_quantity) > 313 )
SELECT c_name, c_custkey, o_orderkey, o_orderdate, o_totalprice, SUM(l_quantity)
FROM customer JOIN orders ON c_custkey = o_custkey
  JOIN lineitem ON o_orderkey = lineitem.l_orderkey
  JOIN l2 ON o_orderkey = l2.l_orderkey
GROUP BY c_name, c_custkey, o_orderkey, o_orderdate, o_totalprice
ORDER BY o_totalprice DESC, o_orderdate LIMIT 100;
```
Rewrite IN-Subquery to Derived Table

Improved performance with JOIN
Introduction to Subqueries

Derived Tables

Scalar Subqueries

IN/EXISTS subqueries

Use Window Functions

Optimizer Hints
Use Window Functions Instead of Subqueries for Aggregation
TPC-H Query 17: Small-Quantity-Order Revenue Query

```
SELECT SUM(l_extendedprice) / 7.0 AS avg_yearly
FROM lineitem, part
WHERE p_partkey = l_partkey AND p_brand = 'Brand#11' AND p_container = 'SM CAN'
  AND l_quantity < (SELECT 0.2 * AVG(l_quantity) FROM lineitem WHERE l_partkey = p_partkey);

WITH win AS (  
  SELECT l_extendedprice, l_quantity, 
     AVG(l_quantity) OVER (PARTITION BY p_partkey) avg_l_quantity  
  FROM lineitem, part  
  WHERE p_partkey = l_partkey AND p_brand = 'Brand#11' AND p_container = 'SM CAN'
)
SELECT SUM(l_extendedprice) / 7.0 AS avg_yearly
FROM win
WHERE l_quantity < 0.2 * avg_l_quantity;
```
Window Functions Instead of Subqueries for Aggregation

Execution Time (seconds)

- Q2
- Q11
- Q15
- Q17

Original Window function

Alibaba Cloud
Introduction to Subqueries

 Derived Tables

 Scalar Subqueries

 IN/EXISTS subqueries

 Use Window Functions

 Optimizer Hints
Optimizer Hints

- Syntax: `SELECT /*+ hints */` ...

- Subquery hints:
  - MERGE(), NO MERGE(): Merge derived table/View/CTE? (MySQL 8.0)
  - SEMIJOIN(), NO SEMIJOIN(): Whether to use semi-join, and which strategy to (not) use
  - SUBQUERY(): Which subquery strategy to use for non-correlated subqueries (Materialization or not)
  - DERIVED_CONDITION_PUSHDOWN(), NO_DERIVED_CONDITION_PUSHDOWN() (MySQL 8.0.22)

- Other very useful hints:
  - JOIN_PREFIX(), JOIN_ORDER(), JOIN_SUFFIX(): Affect join order (MySQL 8.0)
  - INDEX(), NO_INDEX(): Which index to use (MySQL 8.0.20)
Query Rewrite Plugin

• Rewrite problematic queries without the need to make application changes
  • Add hints
  • Rewrite queries

• Add rewrite rules to table:

  ```sql
  INSERT INTO query_rewrite.rewrite_rules (pattern, replacement) VALUES
  ('"SELECT * FROM t1 WHERE a > ? AND b = ?"',
   '"SELECT /*+ INDEX(a_idx) */ * FROM t1 WHERE a > ? AND b = ?"');
  ```

• Pre- and post-parse query rewrite APIs
  • Users can write their own plug-ins
Summary

• Do only select columns in derived table that will be used by outer query
• Push conditions into subqueries
• Be aware that sometimes the automatic merge of derived tables is not optimal
• MySQL 8.0: Use Common Table Expressions (CTEs) instead of derived tables.
• Scalar subqueries may be rewritten to derived tables, but will not always improve performance.
• Prefer IN over EXISTS
• If semijoin does not apply, check if IN-subquery can be replaced by derived table.
• MySQL 8.0: Use window functions to avoid referring the same table twice.
Thank you