Hash Join in MySQL 8.0

Øystein Grøvlen
Senior Principal Engineer
Oracle
February, 2022
Agenda

1. What is hash join?
2. Hash join in MySQL 8.0
3. When to use hash join
4. How to use hash join
Hash Join
SELECT c_name, c_address, o_orderdate, o_totalprice
FROM orders JOIN customer ON o_custkey = c_custkey
WHERE o_totalprice > 500000;

1. Build phase

2. Probe phase
Hybrid Hash Join
Build phase

Build Input Chunk Files
0 1 2 3
4 5 6 7

orders

o_totalprice > 500000
hash(o_custkey)

In-memory hash table
Hybrid Hash Join
Build phase

Build Input Chunk Files

In-memory hash table

orders

in-memory hash table

hash2(o_custkey)

0 1 2 3

4 5 6 7

orders

o_totalprice > 500000

Copyright © 2022, Oracle and/or its affiliates. All rights reserved.
Hybrid Hash Join

Probe phase

In-memory hash table

Proper Input Chunk Files

0 1 2 3
4 5 6 7

customer

hash(c_custkey)

hash2(o_custkey)

Find all matches

Result

Copyright © 2022, Oracle and/or its affiliates. All rights reserved.
Hybrid Hash Join
Repeat for each pair of chunk files

In-memory hash table

Find all matches

Result
Hybrid Hash Join
Repeat for all pairs of chunk files

In-memory hash table

Find all matches

Result

Build Input Chunk Files

Prope Input Chunk Files

hash(o_custkey)

hash(c_custkey)

Copyright © 2022, Oracle and/or its affiliates. All rights reserved.
Hybrid Hash Join
Repeat for all pairs of chunk files

In-memory hash table

Build Input Chunk Files

Prope Input Chunk Files

hash(o_custkey)

hash(c_custkey)

Find all matches

Result
Hash Join in MySQL 8.0
Join Execution in MySQL 5.7

Supports only Nested-Loops Join

• For each row in left input, find all matching rows in right table

Efficient when

• An index can be used to find the matching rows
• A small subset of the rows will be read
• There are relatively few matches per row
• Most rows can be accessed in memory
Join Execution in MySQL 8.0

Supports both Nested-Loops Join and Hash Join

Hash Join

- Supports both In-memory and Hybrid Hash Join
- In-memory
  - Memory usage determined by `join_buffer_size`.
- Hybrid Hash Join
  - Max number of chunk files per input: 128
  - Automatically chosen where Block Nested Loops Join (BNL) was earlier selected
    - Usually when no indexes are available
    - NO_INDEX hint can be used to force hash join to be used
Hash Join versus Nested-Loops Join (TPC-H)
Hash join is much faster than Nested-Loops Join when there are no indexes
Hash Join can also be faster than Indexed Nested-Loops Join

**SELECT COUNT(*) FROM orders JOIN lineitem ON l_orderkey = o_orderkey WHERE o_custkey <= 1500000 * selectivity**
SELECT COUNT(*) FROM orders JOIN lineitem ON l_orderkey = o_orderkey WHERE o_custkey <= 1500000 * selectivity
When to Use Hash Join
No Applicable Index
Hash Join will automatically be selected in 8.0

SELECT s_name, c_name
FROM supplier JOIN customer ON s_phone = c_phone;

MySQL 5.7

mysql [localhost:5736] (msandbox) (dbt3_sf10) > EXPLAIN SELECT s_name, c_name FROM supplier JOIN customer ON s_phone = c_phone;

```
+----+--------+--------------------+----------+----------+--------+------+------+---------+-----------------+-----------------+-----------------+-----------------+
| id | select_type | table | partitions | type | possible_keys | key | key_len | ref | rows | filtered | Extra                  |
+----+--------+--------------------+----------+----------+--------+------+------+---------+-----------------+-----------------+-----------------+-----------------+
| 1  | SIMPLE | supplier | NULL | ALL | NULL | NULL | NULL | 100000 | 100.00 | NULL | Using where, Using join buffer (Block Nested Loop) |
| 1  | SIMPLE | customer | NULL | ALL | NULL | NULL | NULL | 1500000 | 10.00 | NULL |                                 |
```

MySQL 8.0

mysql [localhost:8028] (msandbox) (dbt3_sf10) > EXPLAIN SELECT s_name, c_name FROM supplier JOIN customer ON s_phone = c_phone;

```
+----+--------+--------------------+----------+----------+--------+------+------+---------+-----------------+-----------------+-----------------+-----------------+
| id | select_type | table | partitions | type | possible_keys | key | key_len | ref | rows | filtered | Extra                  |
+----+--------+--------------------+----------+----------+--------+------+------+---------+-----------------+-----------------+-----------------+-----------------+
| 1  | SIMPLE | supplier | NULL | ALL | NULL | NULL | NULL | 100000 | 100.00 | NULL | Using where, Using join buffer (hash join) |
| 1  | SIMPLE | customer | NULL | ALL | NULL | NULL | NULL | 1500000 | 10.00 | NULL |                                 |
```
No Applicable Index
Hash Join will automatically be selected in 8.0

```sql
SELECT s_name, c_name
FROM supplier JOIN customer ON s_phone = c_phone;
```
IO-bound queries

```
SELECT COUNT(*) FROM orders JOIN lineitem ON l_orderkey = o_orderkey WHERE o_custkey <= 1500000 * selectivity
```
IO-bound queries

SELECT COUNT(*) FROM orders JOIN lineitem ON l_orderkey = o_orderkey WHERE o_custkey <= 1500000 * selectivity
SELECT c_count, COUNT(*) AS custdist
FROM (
    SELECT c_custkey, COUNT(o_orderkey) AS c_count
    FROM customer LEFT OUTER JOIN orders
    ON c_custkey = o_custkey AND o_comment NOT LIKE '%express%requests%'
    GROUP BY c_custkey
) AS c_orders
GROUP BY c_count
ORDER BY custdist DESC, c_count DESC;
Selective Conditions on Multiple Tables
Hash join Enables Early Filtering

SELECT *
FROM t1 JOIN t2 ON t1.a = t2.a
WHERE t1.b > 5 AND t2.c < 9;
Selective Conditions on Multiple Tables
TPC-H Q5 (Local Supplier Volume Query)

```
SELECT n_name, sum(l_extendedprice * (1 - l_discount)) as revenue
FROM customer JOIN orders ON c_custkey = o_custkey
JOIN lineitem ON l_orderkey = o_orderkey
JOIN supplier ON l_suppkey = s_suppkey AND c_nationkey = s_nationkey
JOIN nation ON s_nationkey = n_nationkey
JOIN region ON n_regionkey = r_regionkey
WHERE r_name = 'ASIA'
AND o_orderdate >= '1995-01-01' AND o_orderdate < DATE_ADD('1995-01-01', INTERVAL '1' YEAR)
GROUP BY n_name ORDER BY revenue desc;
```
How to Use Hash Join
EXPLAIN Shows Whether Hash Join is Used

EXPLAIN FORMAT=TRADITIONAL:

<table>
<thead>
<tr>
<th>id</th>
<th>select_type</th>
<th>table</th>
<th>partitions</th>
<th>type</th>
<th>possible_keys</th>
<th>key</th>
<th>key_len</th>
<th>ref</th>
<th>rows</th>
<th>filtered</th>
<th>Extra</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>SIMPLE</td>
<td>orders</td>
<td>NULL</td>
<td>ALL</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>15000000</td>
<td>5.02</td>
<td>Using where</td>
</tr>
<tr>
<td>1</td>
<td>SIMPLE</td>
<td>lineitem</td>
<td>NULL</td>
<td>ALL</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>59986052</td>
<td>0.00</td>
<td>Using where; Using join buffer (hash join)</td>
</tr>
</tbody>
</table>

EXPLAIN FORMAT=TREE:

```
- Aggregate: count(0) (cost=4513965784213.93 rows=3009309)
  - Inner hash join (lineitem.l_orderkey = orders.o_orderkey) (cost=4513965483283.02 rows=3009309)
    - Table scan on lineitem (cost=0.19 rows=59986052)
    - Hash
      - Filter: (orders.o_custkey <= <cache>((1500000 * 0.05))) (cost=1530473.77 rows=752502)
        - Table scan on orders (cost=1530473.77 rows=15000000)
```

Probe Input  Build Input
Disable Indexes to Force Hash Join
TPC-H Q13 (Customer Distribution Query)

```
SELECT c_count, COUNT(*) AS custdist
FROM (SELECT c_custkey, COUNT(o_orderkey) AS c_count
      FROM customer LEFT OUTER JOIN orders
      ON c_custkey = o_custkey AND o_comment NOT LIKE '%express%requests%' 
      GROUP BY c_custkey
    ) AS c_orders
GROUP BY c_count
ORDER BY custdist DESC, c_count DESC;
```
Use NO_INDEX Hint to Force Hash Join

TPC-H Q13 (Customer Distribution Query)

```sql
SELECT c_count, COUNT(*) AS custdist
FROM (SELECT /*+ NO_INDEX(orders, i_o_custkey) */ c_custkey, COUNT(o_orderkey) AS c_count
    FROM customer LEFT OUTER JOIN orders
    ON c_custkey = o_custkey AND o_comment NOT LIKE '%express%requests%'
    GROUP BY c_custkey
) AS c_orders
GROUP BY c_count
ORDER BY custdist DESC, c_count DESC;
```

<table>
<thead>
<tr>
<th>id</th>
<th>select_type</th>
<th>table</th>
<th>partitions</th>
<th>type</th>
<th>possible_keys</th>
<th>key</th>
<th>key_len</th>
<th>ref</th>
<th>rows</th>
<th>filtered</th>
<th>Extra</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>PRIMARY</td>
<td>&lt;derived2&gt;</td>
<td>NULL</td>
<td>ALL</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>22500000000000</td>
<td>100.00</td>
<td>Using temporary; Using filesort</td>
</tr>
<tr>
<td>2</td>
<td>DERIVED</td>
<td>customer</td>
<td>NULL</td>
<td>index</td>
<td>PRIMARY,i_c_nationkey</td>
<td>i_c_nationkey</td>
<td>5</td>
<td>NULL</td>
<td>1500000</td>
<td>100.00</td>
<td>Using index; Using temporary</td>
</tr>
<tr>
<td>2</td>
<td>DERIVED</td>
<td>orders</td>
<td>NULL</td>
<td>ALL</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>1500000</td>
<td>100.00</td>
<td>Using where; Using join buffer (hash join)</td>
</tr>
</tbody>
</table>
SELECT n_name, sum(l_extendedprice * (1 - l_discount)) as revenue
FROM customer JOIN orders ON c_custkey = o_custkey
    JOIN lineitem ON l_orderkey = o_orderkey
    JOIN supplier ON l_suppkey = s_suppkey AND c_nationkey = s_nationkey
    JOIN nation ON s_nationkey = n_nationkey
    JOIN region ON n_regionkey = r_regionkey
WHERE r_name = 'ASIA'
    AND o_orderdate >= '1995-01-01' AND o_orderdate < DATE_ADD('1995-01-01', INTERVAL '1' YEAR)
GROUP BY n_name ORDER BY revenue desc;

--> Sort: revenue DESC
--> Table scan on <temporary>
    --> Aggregate using temporary table
        --> Nested loop inner join (cost=4288570.15 rows=261515)
            --> Nested loop inner join (cost=2457968.58 rows=5230290)
                --> Nested loop inner join (cost=1905724.98 rows=1307877)
                    --> Nested loop inner join (cost=30335.99 rows=30000)
                        --> Nested loop inner join (cost=1.88 rows=5)
                            --> Filter: (region.r_name = 'ASIA') (cost=0.75 rows=1)
                                --> Table scan on region (cost=0.75 rows=5)
                                    --> Index lookup on nation using i_r_regionkey (n.regionkey=region.r.regionkey) (cost=1.12 rows=5)
                                        --> Covering index lookup on customer using i_c_nationkey (c.nationkey=nation.n.nationkey) (cost=1266.82 rows=60000)
                                            --> Filter: ((orders.o_orderDATE >= DATE '1995-01-01') and (orders.o_orderDATE < <coalesce(('1995-01-01' + interval '1' year)),)) (cost=3.75 rows=4)
                                                --> Index lookup on orders using i_o_custkey (o_custkey=customer.c_custkey) (cost=3.75 rows=15)
                                                    --> Filter: (lineitem.l_suppkey is not null) (cost=0.25 rows=4)
                                                        --> Index lookup on lineitem using PRIMARY (l_orderkey=orders.o_orderkey) (cost=0.25 rows=4)
                                                            --> Filter: (supplier.s_nationkey = nation.n.nationkey) (cost=0.25 rows=0)
                                                                --> Single-row index lookup on supplier using PRIMARY (s_suppkey=lineitem.l_suppkey) (cost=0.25 rows=1)
Use NO_INDEX Hint to Force Hash Join

TPC-H Q5 (Local Supplier Volume Query)

```sql
SELECT /*+ JOIN_PREFIX(region, nation, customer, orders) NO_INDEX(orders) */
    n_name, sum(l_extendedprice * (1 - l_discount)) as revenue
FROM customer JOIN orders ON c_custkey = o_custkey
JOIN lineitem ON 1_orderkey = o_orderkey
JOIN supplier ON 1_suppkey = s_suppkey
AND c_nationkey = s_nationkey
JOIN nation ON s_nationkey = n_nationkey
JOIN region ON n_regionkey = r_regionkey
WHERE r_name = 'ASIA'
AND o_orderdate >= '1995-01-01' AND o_orderdate < DATE_ADD('1995-01-01', INTERVAL '1' YEAR)
GROUP BY n_name ORDER BY revenue desc;
```
Join Buffer Size Matters
But not that much

```sql
SELECT COUNT(*) FROM orders JOIN lineitem ON l_orderkey = o_orderkey WHERE o_custkey <= 1500000 * selectivity
```
Is Join Buffer Big Enough for In-Memory Hash Join?

Check `performance_schema.memory_summary_global_by_event_name`

```sql
TRUNCATE performance_schema.memory_summary_global_by_event_name;
< Run query >
SELECT event_name, count_alloc, format_bytes(sum_number_of_bytes_alloc) "Total usage",
    high_count_used, format_bytes(high_number_of_bytes_used) "Max. usage"
FROM performance_schema.memory_summary_global_by_event_name
WHERE event_name LIKE 'memory/sql/hash_join';
```

```sql
SET join_buffer_size = 32*1024*1024;
```

```
+-----------------+-----------+----------+-------------+----------+
<table>
<thead>
<tr>
<th>event_name</th>
<th>count_alloc</th>
<th>Total usage</th>
<th>high_count_used</th>
<th>Max. usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>memory/sql/hash_join</td>
<td>465</td>
<td>305.95 MiB</td>
<td>17</td>
<td>15.00 MiB</td>
</tr>
</tbody>
</table>
```

```sql
SET join_buffer_size = 1024*024*1024;
```

```
+-----------------+-----------+----------+-------------+----------+
<table>
<thead>
<tr>
<th>event_name</th>
<th>count_alloc</th>
<th>Total usage</th>
<th>high_count_used</th>
<th>Max. usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>memory/sql/hash_join</td>
<td>23</td>
<td>350.68 MiB</td>
<td>23</td>
<td>350.68 MiB</td>
</tr>
</tbody>
</table>
```
Add Histograms
Reduced performance without histogram on `o_custkey`

```sql
SELECT COUNT(*) FROM orders JOIN lineitem ON l_orderkey = o_orderkey WHERE o_custkey <= 1500000 * selectivity
```
Summary
Summary

• Hash join in MySQL 8.0 gives better join performance when
  • No index is available
  • Query is IO-bound
  • Large part of a table will be accessed
  • Selective conditions on multiple tables
• Use NO_INDEX() hint to force hash join to be used
• Increasing `join_buffer_size` may improve performance
• Remember to create histograms!
  • `ANALYZE TABLE ... UPDATE HISTOGRAM ...`