How to Make JOINs in ClickHouse go Brrr

9 Dec 2024
Robert Schulze
ClickHouse

Mysterious black box, not part of the presentation

Example SELECT Query

```
SELECT
    customer.name
FROM
    usage JOIN customers ON
        users.user_id =
customers.user_id
GROUP BY
    usage.customer_id
WHERE
    users.support_agreement = 'GOLD'
ORDER BY
    customers.name
```

Example SELECT Query

```
SELECT
    customer.name
FROM
    usage JOIN customers ON
        users.user_id =
customers.user_id
GROUP BY
    usage.customer_id
WHERE
    users.support_agreement = 'GOLD'
ORDER BY
    customers.name
```

Relational Operator

5 Project

2 Join

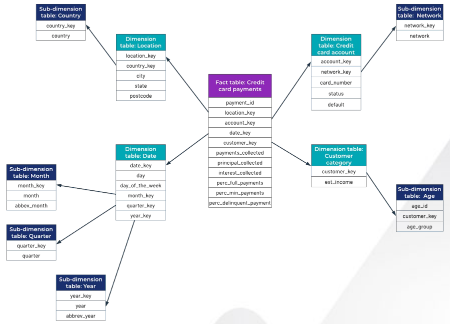
3 Aggregate

1 Filter

4 Sort

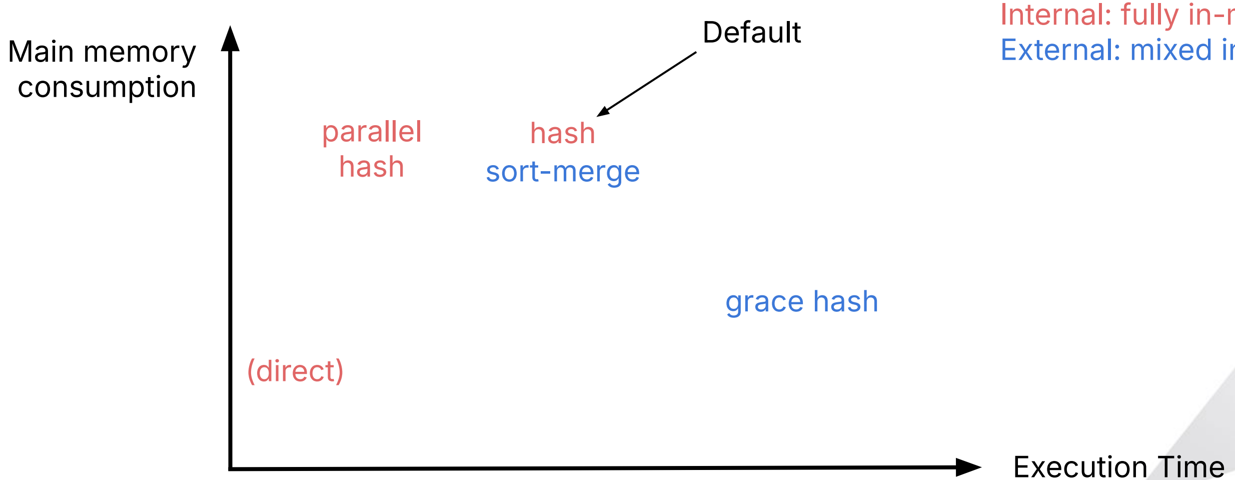
Why are JOINs important?

	De-normalized Schema	Normalized Schema
Table Design	Single fact table	Star or snowflake schema
Characteristics	Fast SELECTs, redundancy	Lower storage costs, fewer ETL steps, requires fast joins



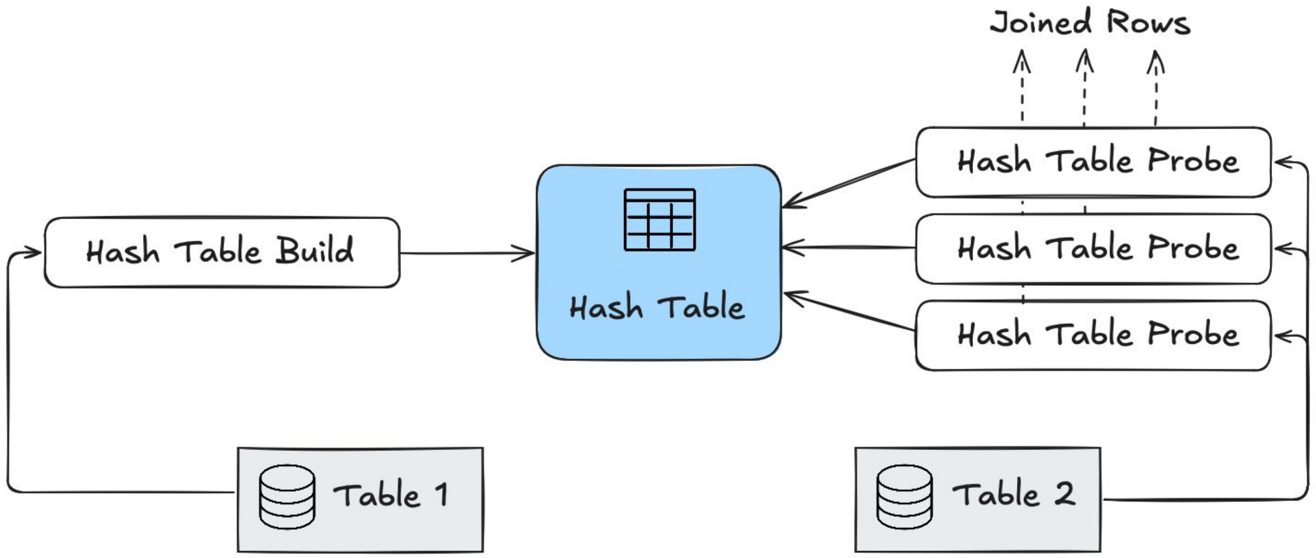
Status Quo

- Support for all major join types:
  - equi join `tab_a.col = tab_b.col`
  - non-equi join `tab_a.col > tab_b.col` v24.12
- Support for all major join algorithms



Internal: fully in-memory
External: mixed in-memory/on-disk

Hash Join

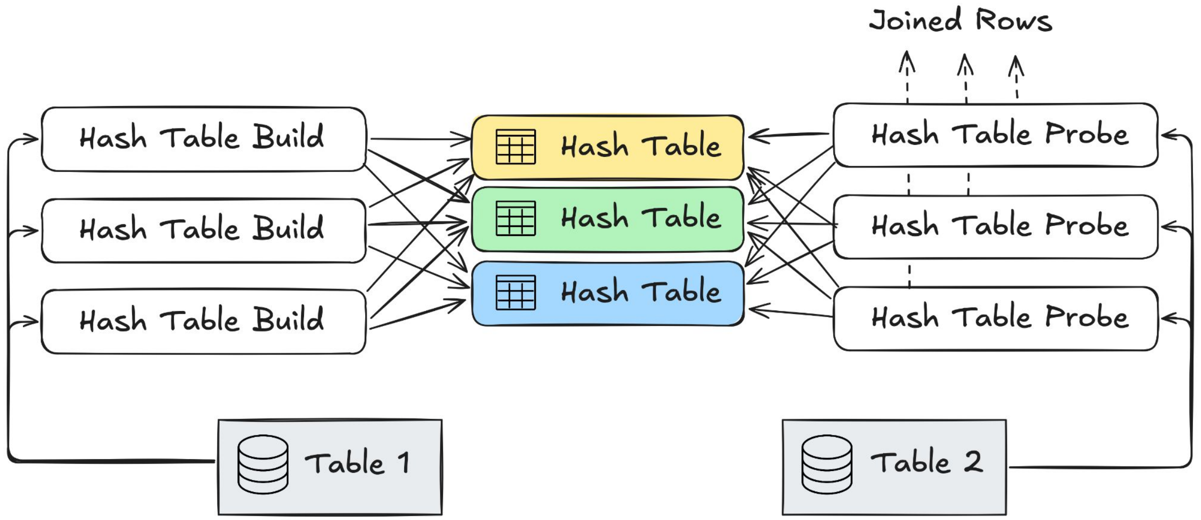


single-threaded

multi-threaded

Optimization 1: Parallel Hash Join by Default

v24.11



multi-threaded

multi-threaded

Up to 30% faster join queries

Optimization 2: Automatically Select the Build Side Table

v24.12

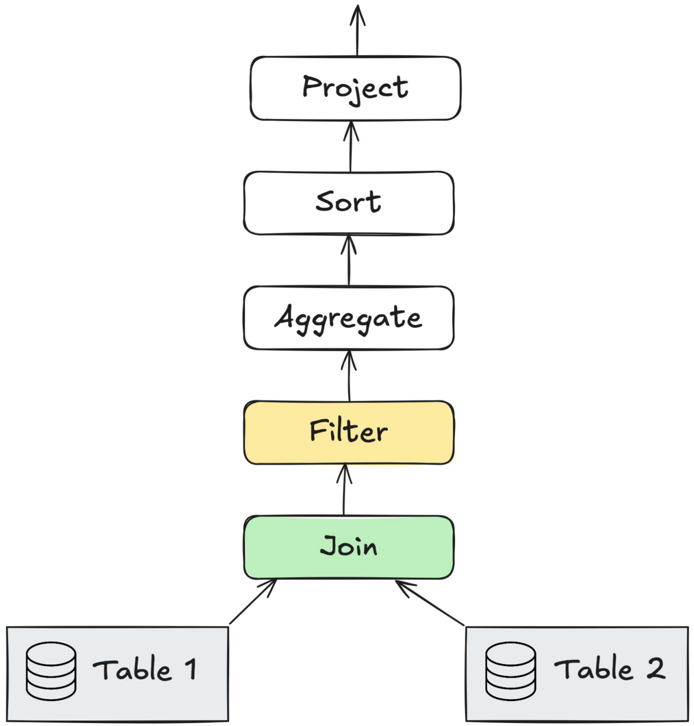
- Hash tables are ideally as small as possible for L1/L2/Lx cache locality

- Previously: Hash table always built from right table in `FROM` clause of `SELECT` query
- Now: Hash table build from the smaller of both tables

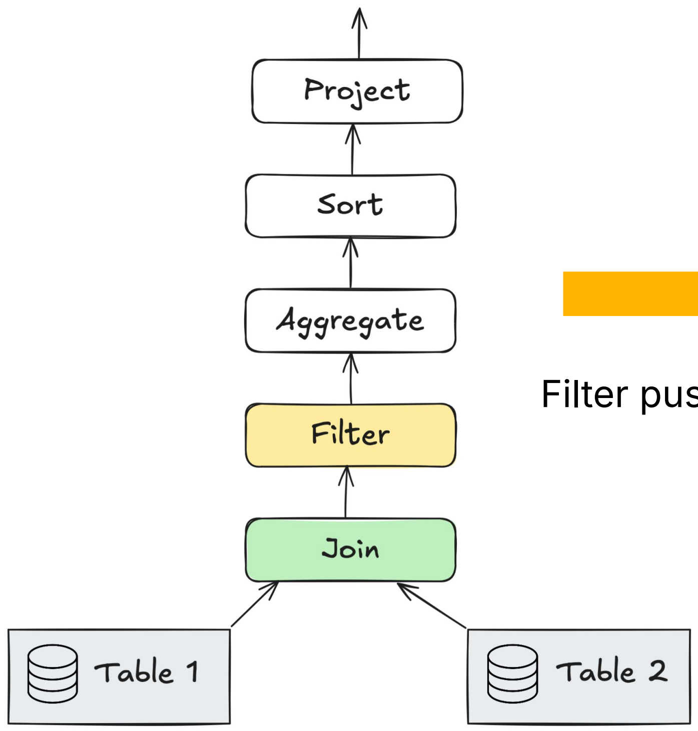
Up to 40% faster join queries

Filter Pushdown

```
SELECT
    customer.name
FROM
    usage JOIN customers ON
        users.user_id =
customers.user_id
GROUP BY
    usage.customer_id
WHERE
    users.support_agreement = 'GOLD'
ORDER BY
    customers.name
```

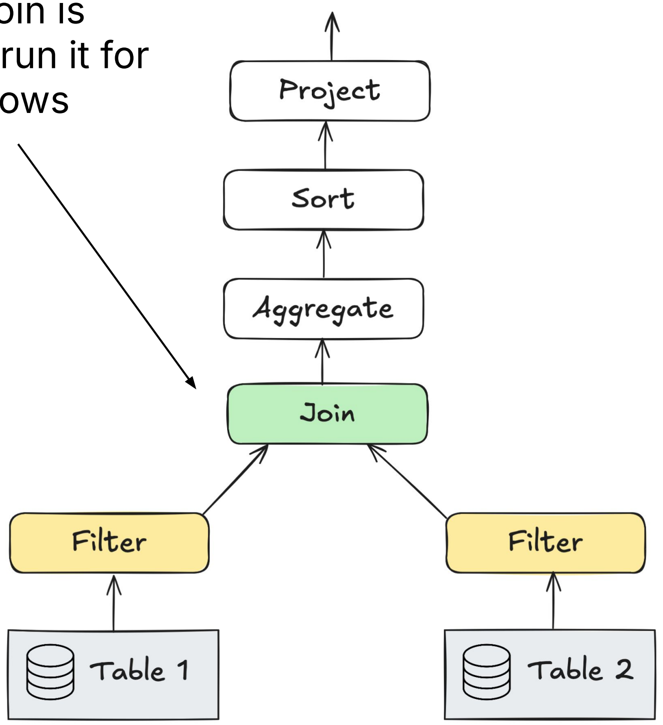


Filter Pushdown



Filter pushdown

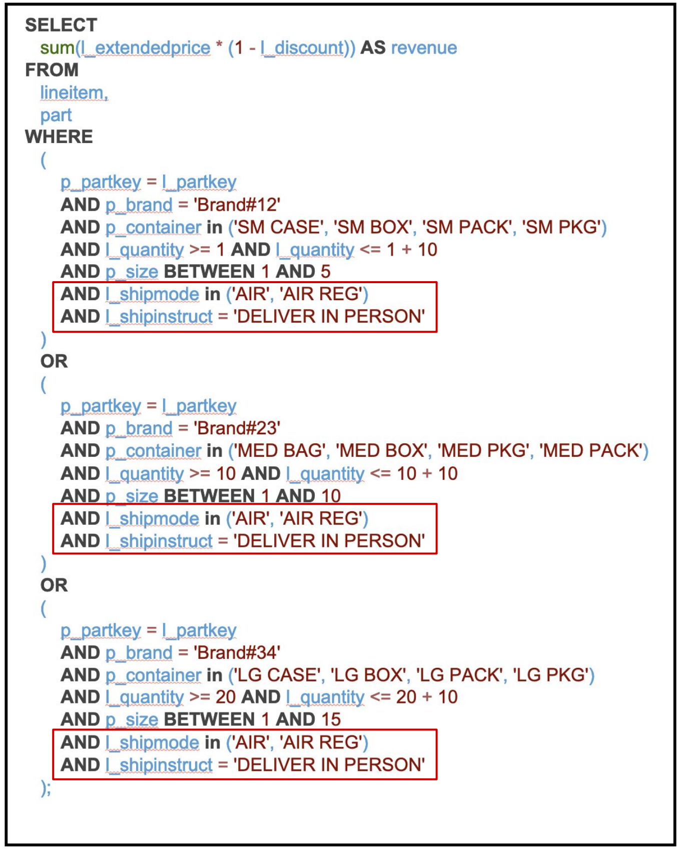
Idea: Join is costly, run it for fewer rows



TPC-H Q19: Runtime down from ∞ to 7.6 sec

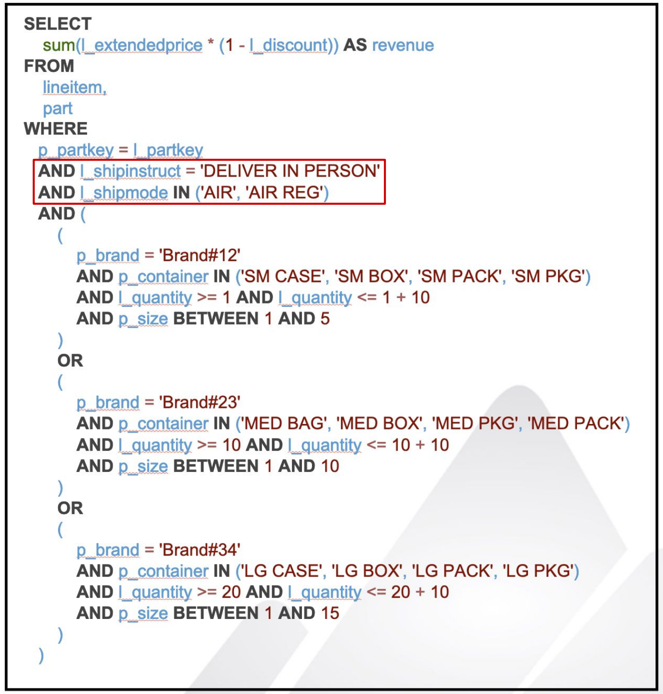
Optimization 3: Pushdown Filters Aggressively

v24.12



```
SELECT
  sum(l_extendedprice * (1 - l_discount)) AS revenue
FROM
  lineitem,
  part
WHERE
  (
    p_partkey = l_partkey
    AND p_brand = 'Brand#12'
    AND p_container in ('SM CASE', 'SM BOX', 'SM PACK', 'SM PKG')
    AND l_quantity >= 1 AND l_quantity <= 1 + 10
    AND p_size BETWEEN 1 AND 5
    AND l_shipmode in ('AIR', 'AIR REG')
    AND l_shipinstruct = 'DELIVER IN PERSON'
  )
  OR
  (
    p_partkey = l_partkey
    AND p_brand = 'Brand#23'
    AND p_container in ('MED BAG', 'MED BOX', 'MED PKG', 'MED PACK')
    AND l_quantity >= 10 AND l_quantity <= 10 + 10
    AND p_size BETWEEN 1 AND 10
    AND l_shipmode in ('AIR', 'AIR REG')
    AND l_shipinstruct = 'DELIVER IN PERSON'
  )
  OR
  (
    p_partkey = l_partkey
    AND p_brand = 'Brand#34'
    AND p_container in ('LG CASE', 'LG BOX', 'LG PACK', 'LG PKG')
    AND l_quantity >= 20 AND l_quantity <= 20 + 10
    AND p_size BETWEEN 1 AND 15
    AND l_shipmode in ('AIR', 'AIR REG')
    AND l_shipinstruct = 'DELIVER IN PERSON'
  );
```

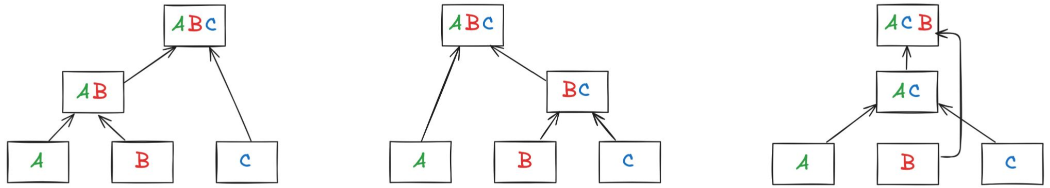
Applying De Morgan's Law enables filter pushdown



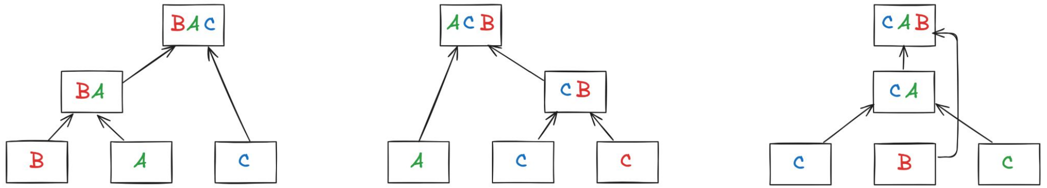
```
SELECT
  sum(l_extendedprice * (1 - l_discount)) AS revenue
FROM
  lineitem,
  part
WHERE
  p_partkey = l_partkey
  AND l_shipinstruct = 'DELIVER IN PERSON'
  AND l_shipmode IN ('AIR', 'AIR REG')
  AND (
    (
      p_brand = 'Brand#12'
      AND p_container IN ('SM CASE', 'SM BOX', 'SM PACK', 'SM PKG')
      AND l_quantity >= 1 AND l_quantity <= 1 + 10
      AND p_size BETWEEN 1 AND 5
    )
    OR
    (
      p_brand = 'Brand#23'
      AND p_container IN ('MED BAG', 'MED BOX', 'MED PKG', 'MED PACK')
      AND l_quantity >= 10 AND l_quantity <= 10 + 10
      AND p_size BETWEEN 1 AND 10
    )
    OR
    (
      p_brand = 'Brand#34'
      AND p_container IN ('LG CASE', 'LG BOX', 'LG PACK', 'LG PKG')
      AND l_quantity >= 20 AND l_quantity <= 20 + 10
      AND p_size BETWEEN 1 AND 15
    )
  )
```

The rabbit hole goes deeper …

- Observation 1: JOINs are associative



- Observation 2: JOINs are commutative (switch build & probe sides)



- Observation 3: As more tables are joined, the number of possible join orders explodes

$$NumJoins(N) = \frac{(2N-2)!}{(N-1)!}$$

$$NumJoins(3) = 12$$

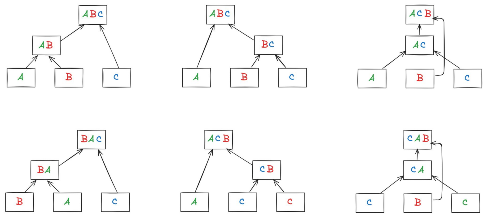
$$NumJoins(7) \sim 17mil$$

Work in Progress: Join Reordering

v25.x

- Performance is mostly influenced by the JOIN order
- To find a good JOIN order, we need ...

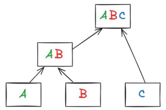
Optimization Algorithm



Enumerate exhaustively vs. sub-set

Cost Model

Input: join order



Output: estimated costs

$$C = \begin{cases} |R| & \text{if R is base table} \\ |R| + C(|S|) + C(|T|) & \text{if R is a join between S and T} \end{cases}$$

Statistics about Base Tables

Cost models need statistics about values in a column:

- how many distinct values?
- top-10 most frequent values, etc.

Summary

- First JOIN optimizations will be in v24.11 / v24.12
- More foundational optimizations for JOINs (join reordering) are work-in-progress and expected in v25.x.