Navigating PostgreSQL — Join Strategies

Nested Loops

- Nested loops join is one of the simplest join strategies that PostgreSQL uses. It works by taking each row from one table (outer table) and finding matching rows in another table (inner table) by scanning or using an index.

- ### How Nested Loops Join Works

When PostgreSQL performs a nested loops join:

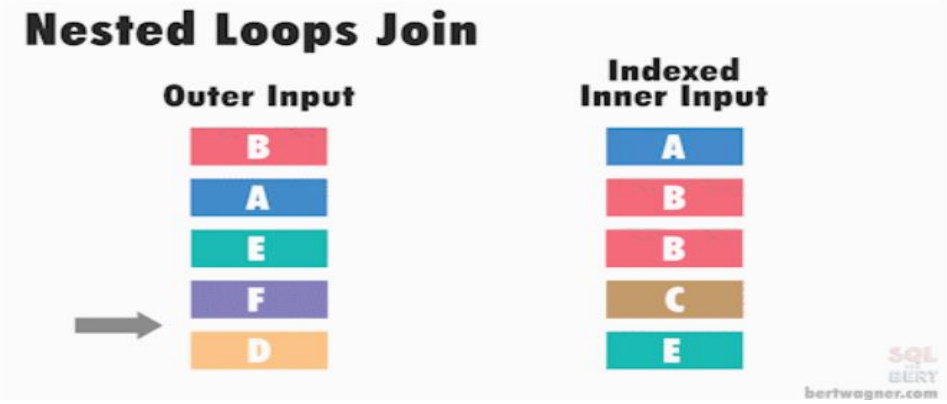
- It first gets a row from the outer table (usually the table with fewer matching rows after filters)
- For each outer row, it scans the inner table (either sequentially or using an index) to find matching rows
- When matches are found, it combines the rows according to the join condition
- This process repeats for every row from the outer table

Nested loops are most efficient when:

- The outer table has very few rows after filtering
- The inner table has an index on the join columns
- The join needs to return only a small number of rows

think of this like a double for loop.

```
Copy
for (order in orders) {
   for (product in products) {
      if (order.product_id == product.id) {
         // join the order and product
      }
   }
}
None
```



For example, in this query plan:

```
Copy
postgres=# explain select * from orders o join products p on p.id=o.product_id where o.id<107219;
                                QUERY PLAN
-------------------------------------------------------------------------------------------
 Nested Loop  (cost=0.70..14.81 rows=11 width=50)
   ->  Index Scan using orders_pkey on orders o  (cost=0.42..2.62 rows=11 width=27)
         Index Cond: (id < 107219)
   ->  Index Scan using products_pkey on products p  (cost=0.27..1.11 rows=1 width=23)
         Index Cond: (id = o.product_id)
(5 rows)
```

Note: order is the outer table and product is the inner table.

Understanding the Query Plan

- The index scan on orders returns 11 rows.
- For each of these rows, the query performs an index scan on products.
- The index scan on products returns 1 row for each of the 11 orders.

Hash Join / Hash

- Hash join is another join strategy that PostgreSQL uses. It works by creating a hash table from one of the tables and then using that hash table to find matching rows in another table.
- **How Hash Join Works**

When PostgreSQL performs a hash join:

- It creates a hash table from one of the tables (usually the smaller one)
- It then uses that hash table to find matching rows in another table
- When matches are found, it combines the rows according to the join condition

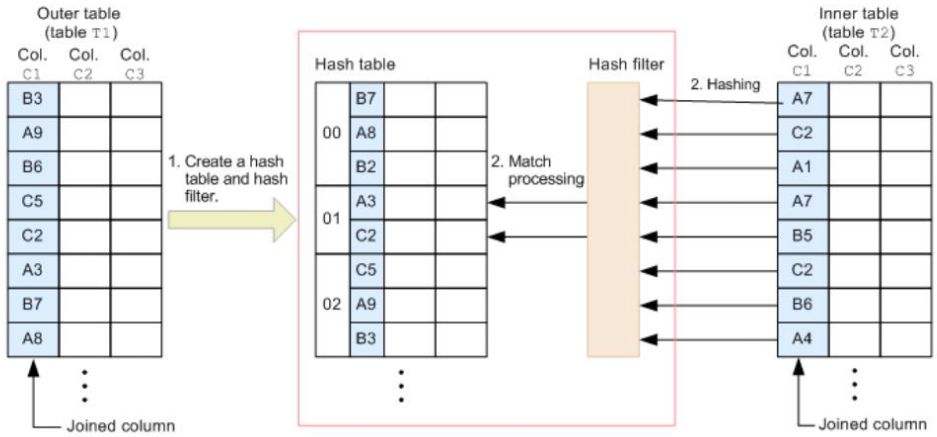
Hash join is most efficient when:

- The join needs to return only a small number of rows
- The inner table has an index on the join columns

think of this like a hash map.

```
Copy
hash_map = {}
for (product in products) {
   hash_map[product.id] = product
}

for (order in orders) {
   if (hash_map[order.product_id]) {
      // join the order and product
   }
}
None
```




```
postgres=# explain select * from orders o join products p on p.id=o.product_id where o.id<117219;
                                QUERY PLAN
-------------------------------------------------------------------------------------
 Hash Join  (cost=15.68..339.95 rows=10536 width=50)
   Hash Cond: (o.product_id = p.id)
```

```
  ->  Index Scan using orders_pkey on orders o  (cost=0.42..296.81 rows=10536 width=27)
        Index Cond: (id < 117219)
  ->  Hash  (cost=9.00..9.00 rows=500 width=23)
        ->  Seq Scan on products p  (cost=0.00..9.00 rows=500 width=23)
(6 rows)
```

Note: orders is the outer table and products is the inner table.

Understanding the Query Plan

- The index scan on orders returns 10,536 rows.
- The hash join creates a hash table from the products table (smaller table).
- For each row in the orders table, the query uses the hash table to find matching rows in the products table.
- The hash join returns 10,536 rows.

Why Did PostgreSQL Choose a Hash Join Instead of a Nested Loop?

Key Differences from Previous Query (id < 107219):

- The previous query had only 11 rows from orders, so a Nested Loop Join was efficient.
- This query has 10,536 rows, so a Nested Loop would require 10,536 index lookups in products, which would be slow.
- Instead, PostgreSQL builds a hash table once (cheap) and does fast lookups (O(1)) instead of repeated index scans (O(log N)).

Merge Join

- Merge join is another join strategy that PostgreSQL uses. It works by sorting the two tables and then merging them together.

How Merge Join Works

When PostgreSQL performs a merge join:

- It sorts the two tables by the join columns
- It then merges the two tables together
- When matches are found, it combines the rows according to the join condition

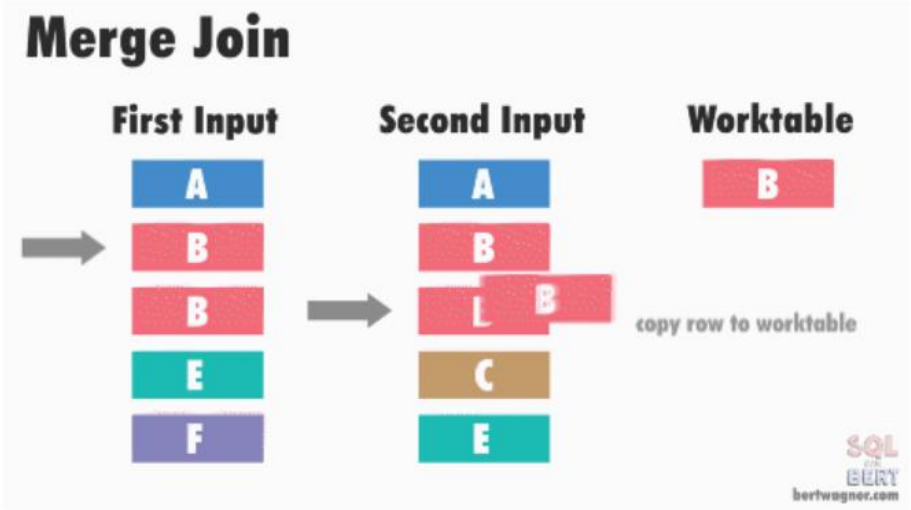
Merge join is most efficient when:

- Both tables are already sorted (or can be sorted efficiently).

- There's no suitable index for a Nested Loop.
- The tables are large, making Hash Join expensive.

think of this like a merge sort.

```
Copy
merge_sort(left, right) {
   if (left.length == 0) return right;
   if (right.length == 0) return left;
}
```




```
explain
SELECT *
FROM employees e
JOIN salary s
  ON e.id = s.employee_id
WHERE e.age < 40;

QUERY PLAN
------------------------------------------------------------------
Merge Join  (cost=198.11..268.19 rows=10 width=488)
   Merge Cond: (e.id = s.employee_id)
  ->  Index Scan using employees_id on employees e  (cost=0.29..656.28 rows=101 width=244)
        Filter: (age < 40)
  ->  Sort  (cost=197.83..200.33 rows=1000 width=244)
        Sort Key: s.employee_id
        ->  Seq Scan on salary s (cost=0.00..148.00 rows=1000 width=244)
```

Understanding the Query Plan

- The index scan on employees returns 101 rows.
- The sort operation sorts the salary table by the join column (employee_id).
- The merge join then merges the two sorted tables together.
- The merge join returns 10 rows.

Why Did PostgreSQL Choose a Merge Join Instead of Nested Loop or Hash Join?

Key Differences from Previous Queries:

- The employees table is filtered (age < 40), reducing rows to 101.
- The salary table is not indexed on employee_id, so a Hash Join would require creating a large hash table.
- The tables can be efficiently sorted and merged, making a Merge Join the best option.

Join Type	When Used	How It Works
Nested Loop	Small outer tables (<1000) Has indexes on join cols Heavily filtered outer	Iterates outer rows, checks matches in inner via index Like nested for loops
Hash Join	Medium/large tables No useful indexes Memory available	Builds hash table from small table, probes with larger O(1) hash lookups
Merge Join	Pre-sorted data Sort cost acceptable Large tables ok	Sorts inputs if needed, then walks both in parallel to match rows like merge sort

Join Type	Time Complexity	Space Complexity
Nested Loop	O(n*m) no index O(n*log m) with index n=outer, m=inner rows	O(1) minimal memory
Hash Join	O(n + m) build & probe	O(n) for hash table n = smaller table size
Merge Join	O(n log n + m log m) if sorting O(n + m) if pre-sorted	O(1) if pre-sorted O(n + m) if sorting