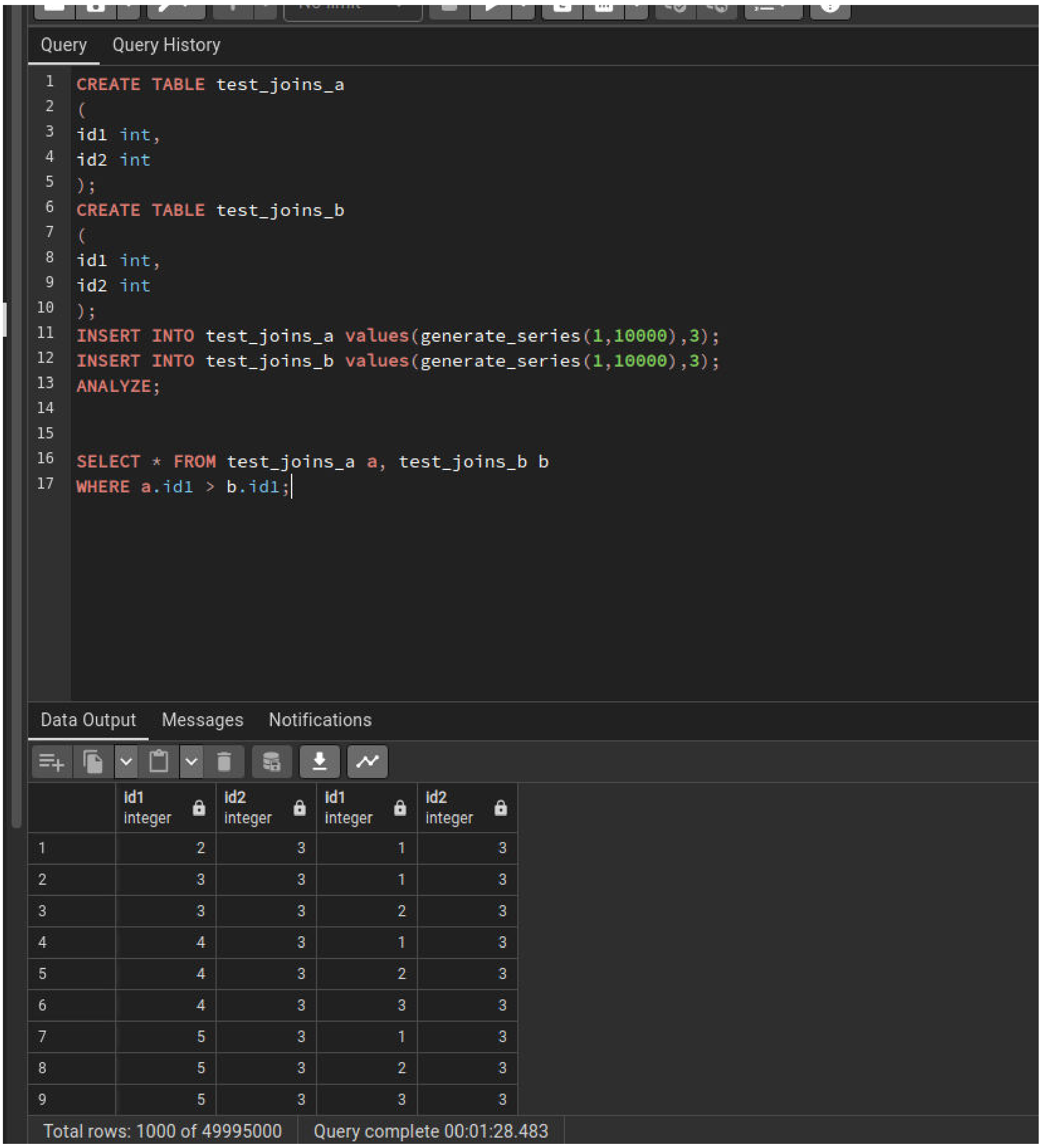
PG_DWH TASK 5

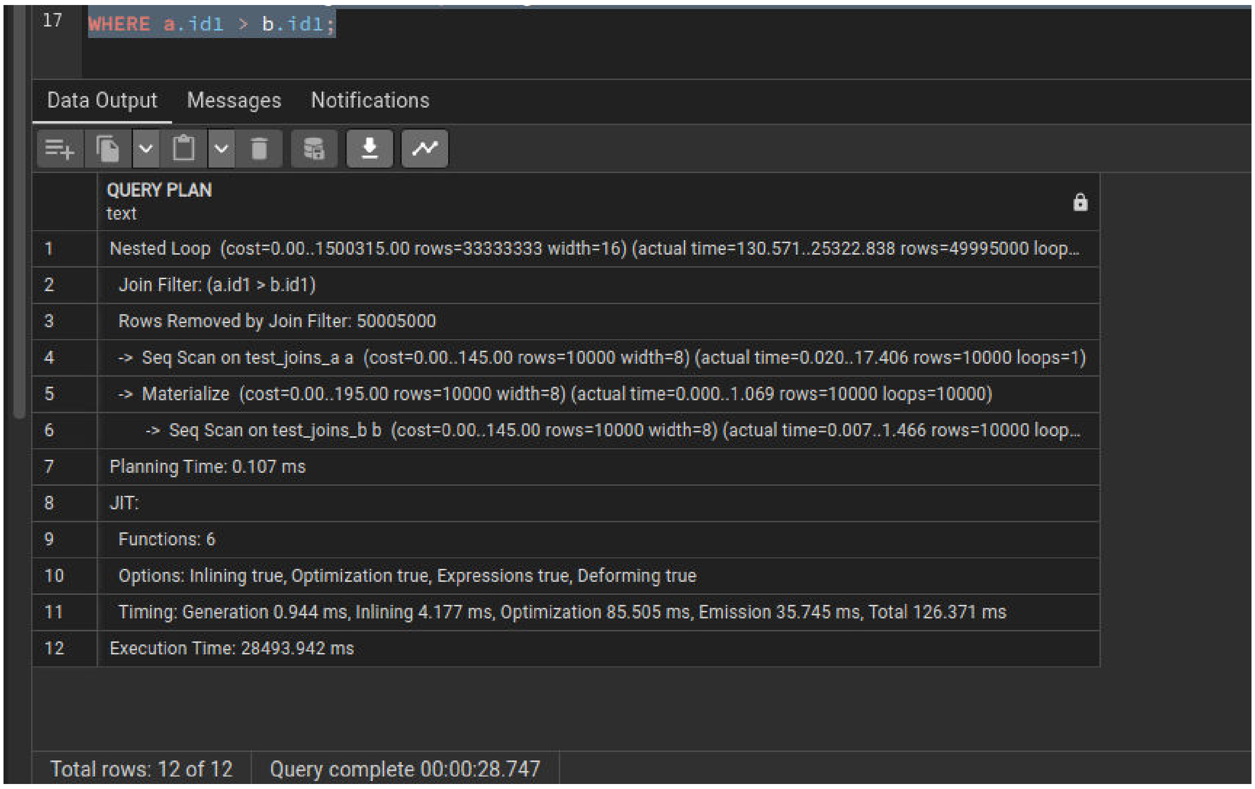


```
CREATE TABLE test_joins_a
(
id1 int,
id2 int
);
CREATE TABLE test_joins_b
(
id1 int,
id2 int
);
INSERT INTO test_joins_a values(generate_series(1,10000),3);
INSERT INTO test_joins_b values(generate_series(1,10000),3);
ANALYZE;


SELECT * FROM test_joins_a a, test_joins_b b
WHERE a.id1 > b.id1;
```

	id1 integer	id2 integer	id1 integer	id2 integer
1	2	3	1	3
2	3	3	1	3
3	3	3	2	3
4	4	3	1	3
5	4	3	2	3
6	4	3	3	3
7	5	3	1	3
8	5	3	2	3
9	5	3	3	3

Total rows: 1000 of 49995000 Query complete 00:01:28.483

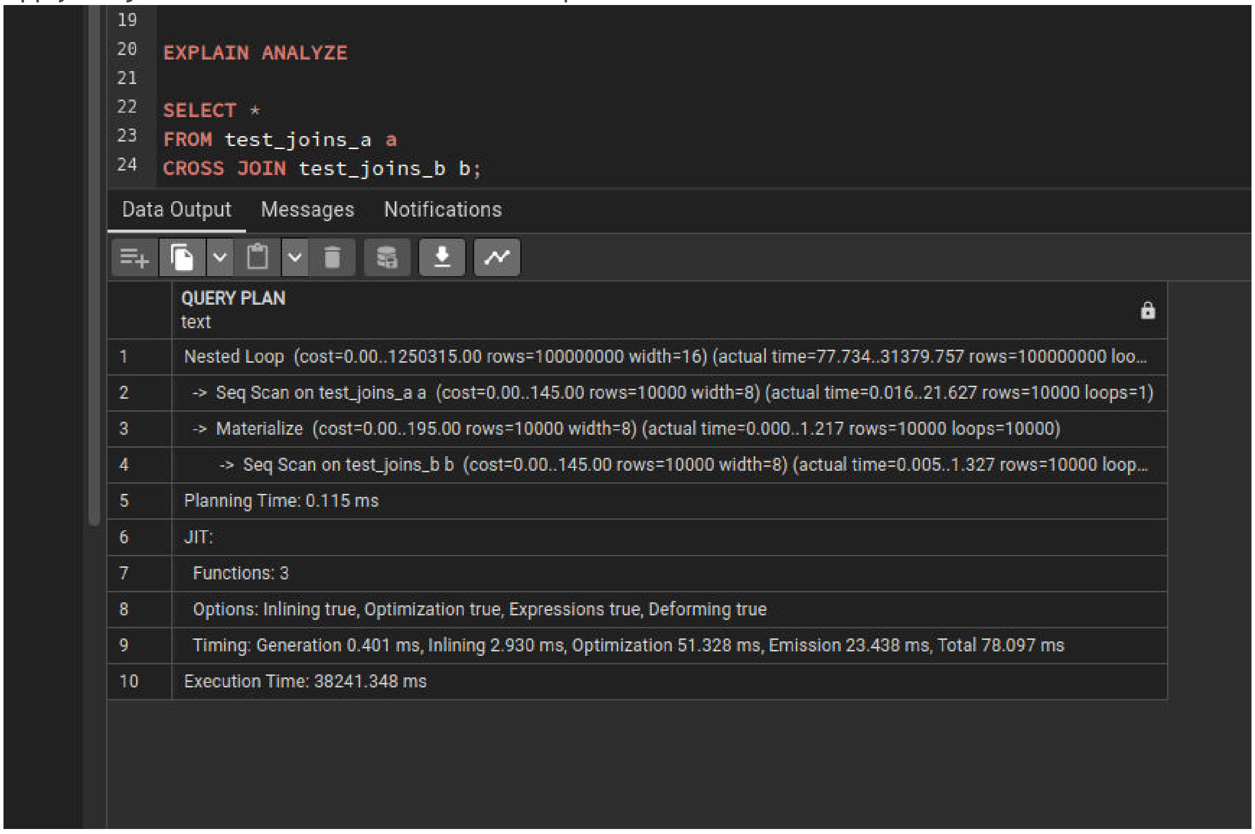
Table created and join query executed.



```
WHERE a.id1 > b.id1;
```

```
QUERY PLAN
1  Nested Loop  (cost=0.00..1500315.00 rows=33333333 width=16) (actual time=130.571..25322.838 rows=49995000 loop...
2    Join Filter: (a.id1 > b.id1)
3    Rows Removed by Join Filter: 50005000
4    ->  Seq Scan on test_joins_a a  (cost=0.00..145.00 rows=10000 width=8) (actual time=0.020..17.406 rows=10000 loops=1)
5    ->  Materialize  (cost=0.00..195.00 rows=10000 width=8) (actual time=0.000..1.069 rows=10000 loops=10000)
6          ->  Seq Scan on test_joins_b b  (cost=0.00..145.00 rows=10000 width=8) (actual time=0.007..1.466 rows=10000 loop...
7  Planning Time: 0.107 ms
8  JIT:
9    Functions: 6
10   Options: Inlining true, Optimization true, Expressions true, Deforming true
11   Timing: Generation 0.944 ms, Inlining 4.177 ms, Optimization 85.505 ms, Emission 35.745 ms, Total 126.371 ms
12 Execution Time: 28493.942 ms

Total rows: 12 of 12    Query complete 00:00:28.747
```

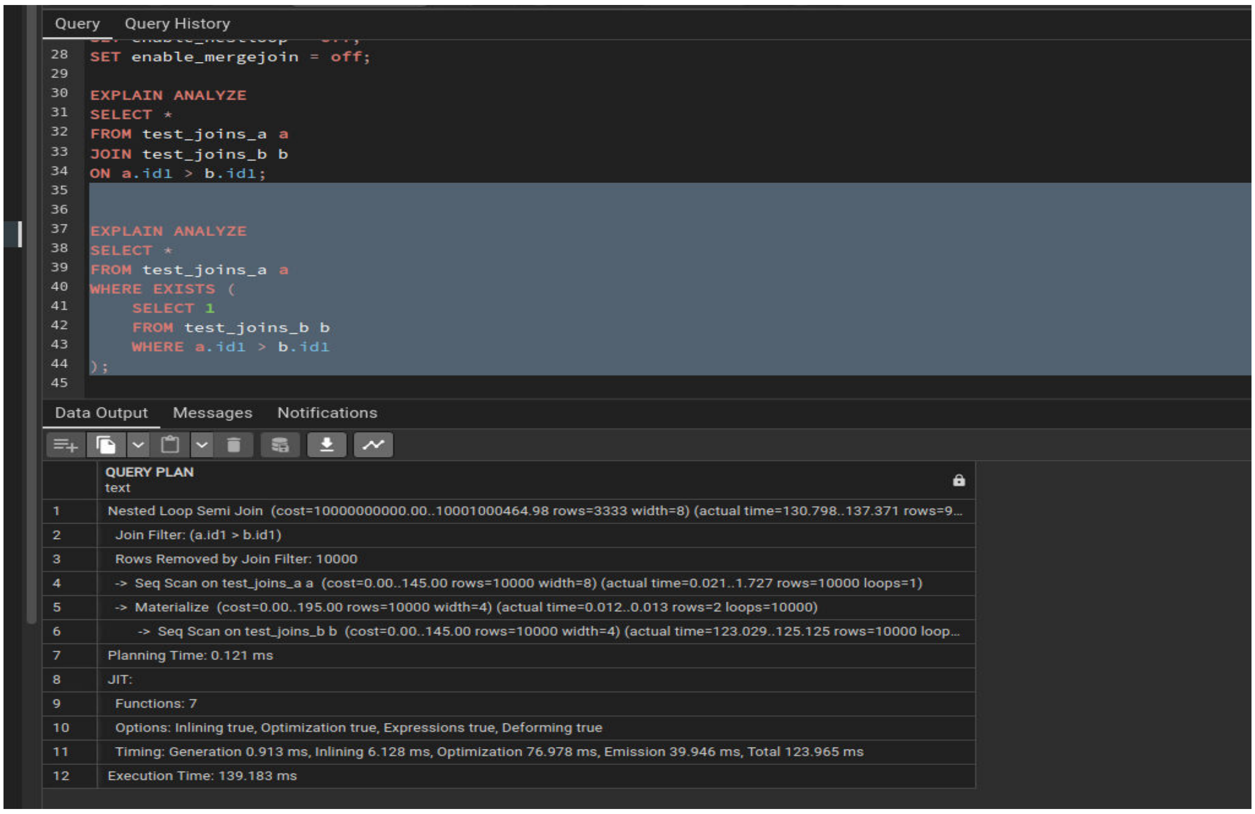
As we see postgres choose to Sequential Scan method for this join query , it is firstly scaned thru test_joins_a and then test_joins_b. For each row in `test_joins_a` , Iterate through the materialized rows from `test_joins_b` (10,000 rows for each row in `test_joins_a`).
Apply the join filter `a.id1 > b.id1` to each pair of rows.



```
EXPLAIN ANALYZE

SELECT *
FROM test_joins_a a
CROSS JOIN test_joins_b b;
```

```
QUERY PLAN
1  Nested Loop  (cost=0.00..1250315.00 rows=100000000 width=16) (actual time=77.734..31379.757 rows=100000000 loo...
2    ->  Seq Scan on test_joins_a a  (cost=0.00..145.00 rows=10000 width=8) (actual time=0.016..21.627 rows=10000 loops=1)
3    ->  Materialize  (cost=0.00..195.00 rows=10000 width=8) (actual time=0.000..1.217 rows=10000 loops=10000)
4          ->  Seq Scan on test_joins_b b  (cost=0.00..145.00 rows=10000 width=8) (actual time=0.005..1.327 rows=10000 loop...
5  Planning Time: 0.115 ms
6  JIT:
7    Functions: 3
8    Options: Inlining true, Optimization true, Expressions true, Deforming true
9    Timing: Generation 0.401 ms, Inlining 2.930 ms, Optimization 51.328 ms, Emission 23.438 ms, Total 78.097 ms
10 Execution Time: 38241.348 ms
```

As we can see Cross join exequtes exactly in the same method and order.



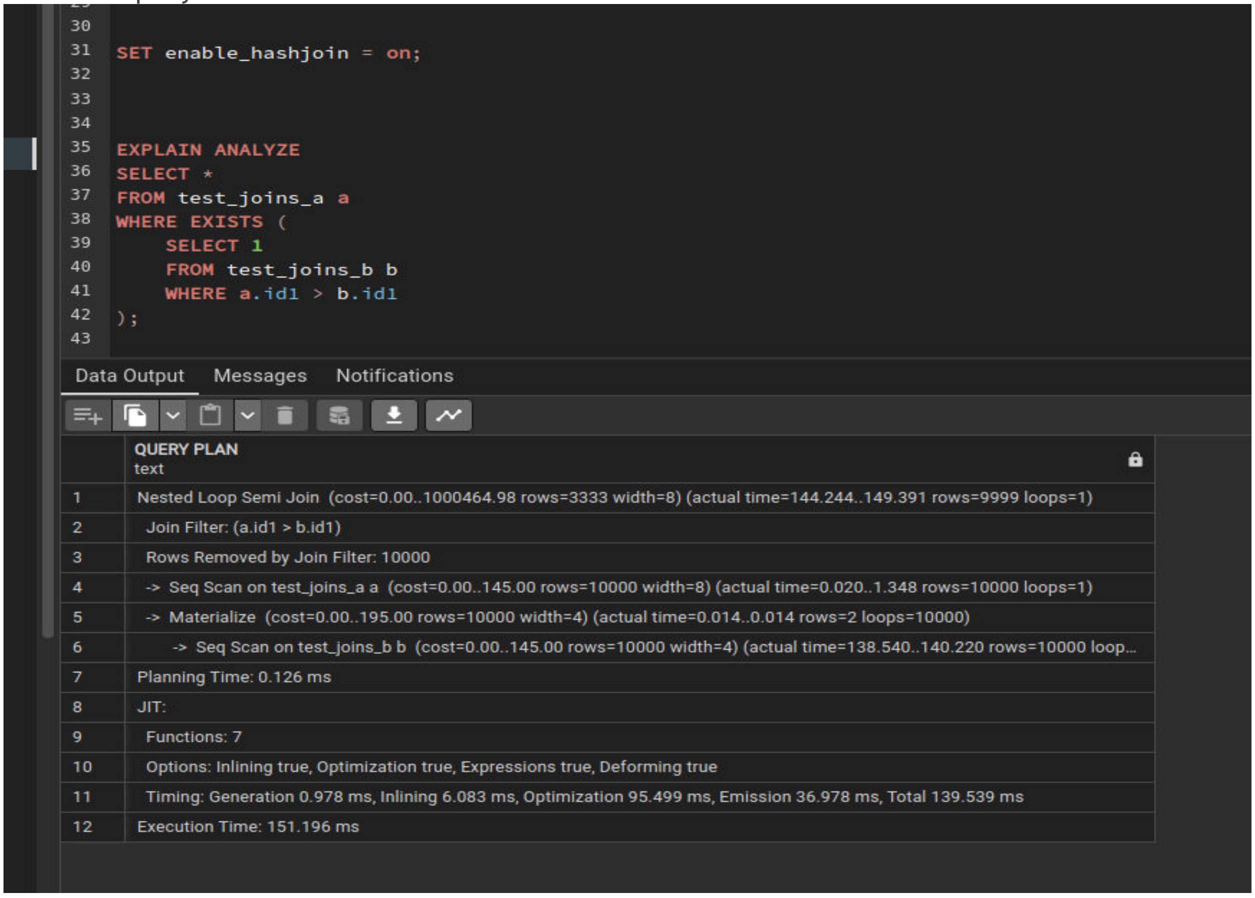
Query   Query History

```sql
SET enable_mergejoin = off;

EXPLAIN ANALYZE
SELECT *
FROM test_joins_a a
JOIN test_joins_b b
ON a.id1 > b.id1;


EXPLAIN ANALYZE
SELECT *
FROM test_joins_a a
WHERE EXISTS (
    SELECT 1
    FROM test_joins_b b
    WHERE a.id1 > b.id1
);
```

Data Output   Messages   Notifications

	QUERY PLAN text
1	Nested Loop Semi Join (cost=10000000000.00..10001000464.98 rows=3333 width=8) (actual time=130.798..137.371 rows=9...
2	Join Filter: (a.id1 > b.id1)
3	Rows Removed by Join Filter: 10000
4	-> Seq Scan on test_joins_a a (cost=0.00..145.00 rows=10000 width=8) (actual time=0.021..1.727 rows=10000 loops=1)
5	-> Materialize (cost=0.00..195.00 rows=10000 width=4) (actual time=0.012..0.013 rows=2 loops=10000)
6	-> Seq Scan on test_joins_b b (cost=0.00..145.00 rows=10000 width=4) (actual time=123.029..125.125 rows=10000 loop...
7	Planning Time: 0.121 ms
8	JIT:
9	Functions: 7
10	Options: Inlining true, Optimization true, Expressions true, Deforming true
11	Timing: Generation 0.913 ms, Inlining 6.128 ms, Optimization 76.978 ms, Emission 39.946 ms, Total 123.965 ms
12	Execution Time: 139.183 ms

Semi Join query created.

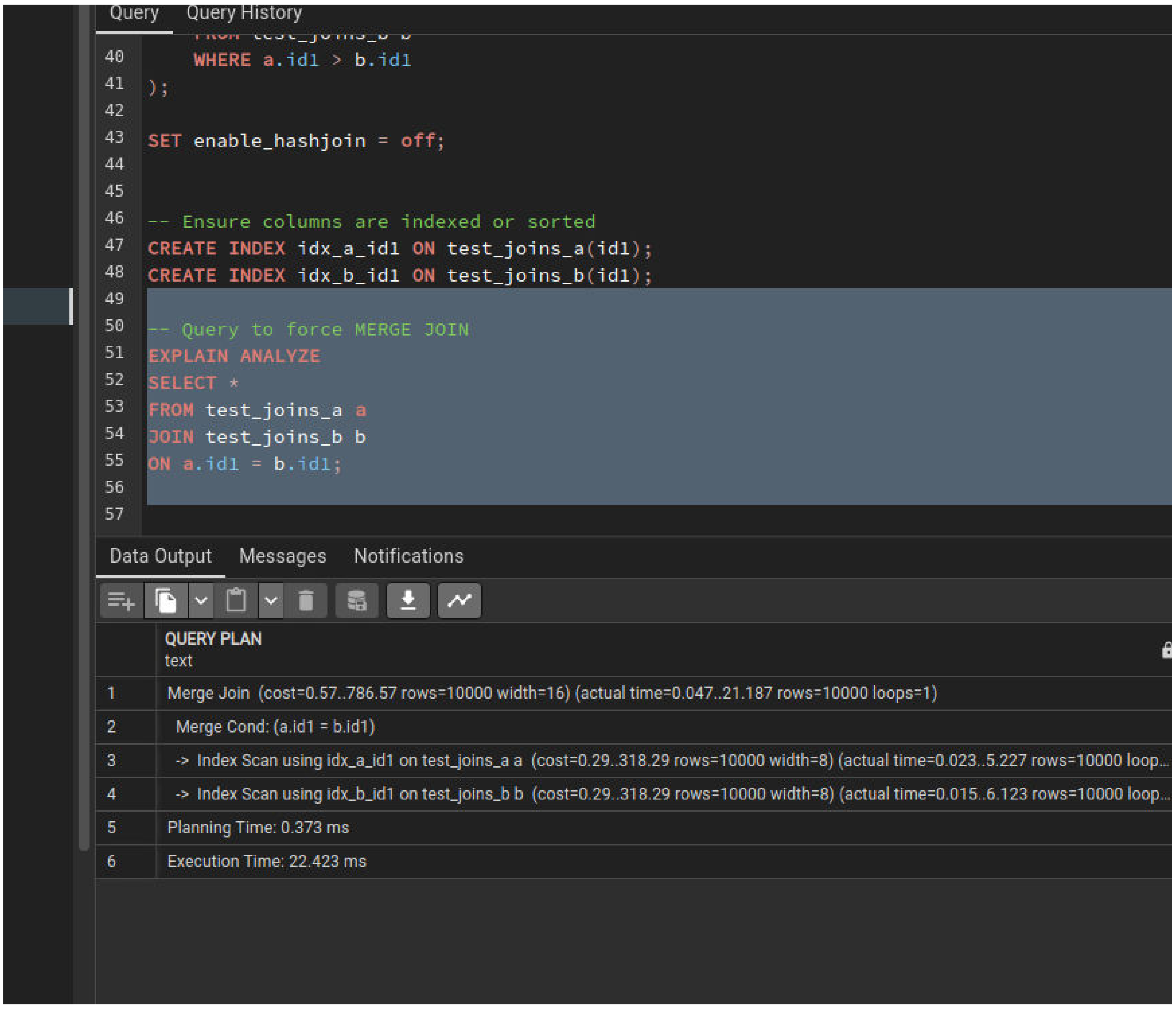


```sql
SET enable_hashjoin = on;



EXPLAIN ANALYZE
SELECT *
FROM test_joins_a a
WHERE EXISTS (
    SELECT 1
    FROM test_joins_b b
    WHERE a.id1 > b.id1
);
```

Data Output   Messages   Notifications

	QUERY PLAN text
1	Nested Loop Semi Join (cost=0.00..1000464.98 rows=3333 width=8) (actual time=144.244..149.391 rows=9999 loops=1)
2	Join Filter: (a.id1 > b.id1)
3	Rows Removed by Join Filter: 10000
4	-> Seq Scan on test_joins_a a (cost=0.00..145.00 rows=10000 width=8) (actual time=0.020..1.348 rows=10000 loops=1)
5	-> Materialize (cost=0.00..195.00 rows=10000 width=4) (actual time=0.014..0.014 rows=2 loops=10000)
6	-> Seq Scan on test_joins_b b (cost=0.00..145.00 rows=10000 width=4) (actual time=138.540..140.220 rows=10000 loop...
7	Planning Time: 0.126 ms
8	JIT:
9	Functions: 7
10	Options: Inlining true, Optimization true, Expressions true, Deforming true
11	Timing: Generation 0.978 ms, Inlining 6.083 ms, Optimization 95.499 ms, Emission 36.978 ms, Total 139.539 ms
12	Execution Time: 151.196 ms

enable_hashjoin set to off and query plan executed , then set to on.

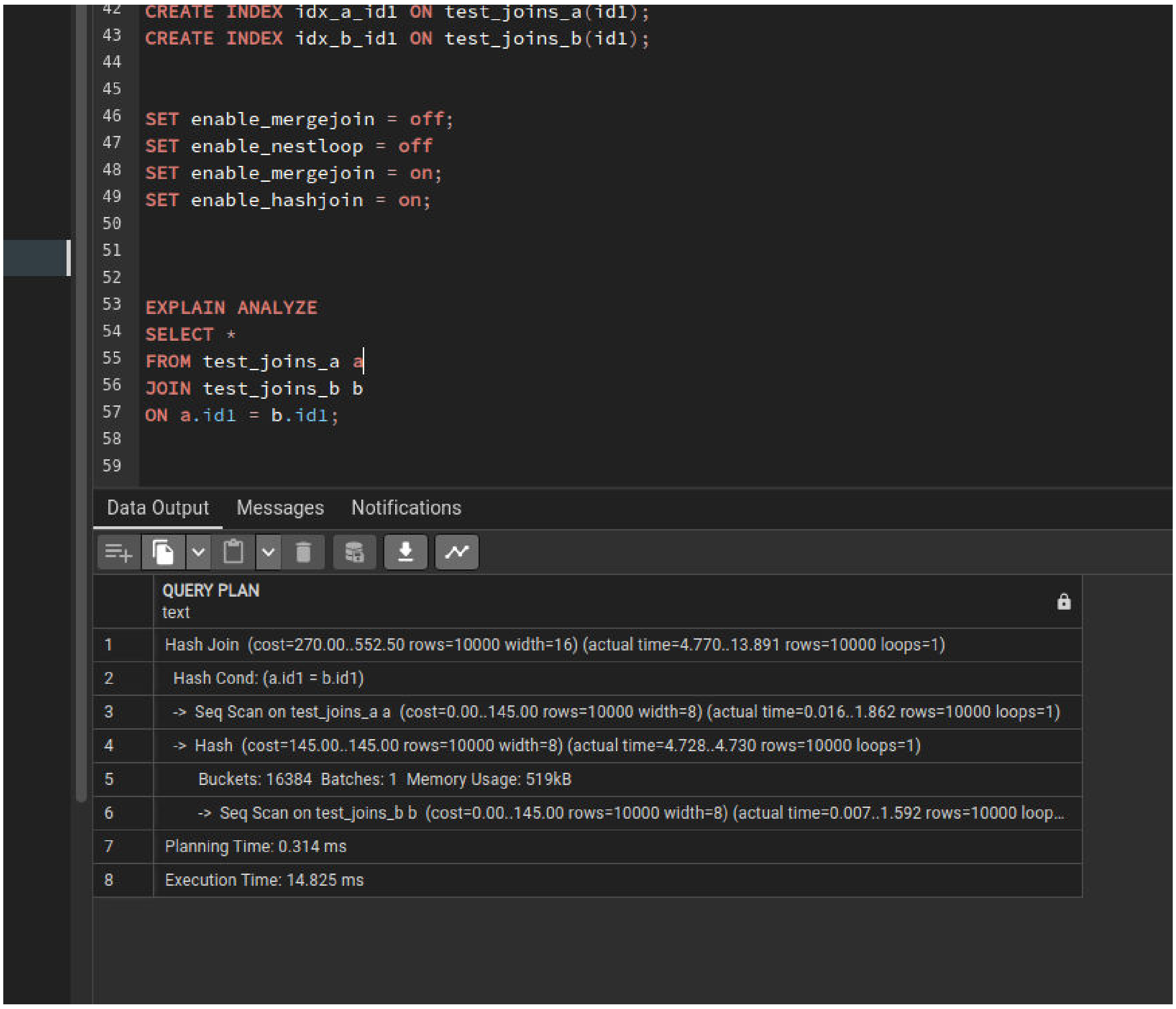

Query	Query History

```
    WHERE a.id1 > b.id1
);

SET enable_hashjoin = off;


-- Ensure columns are indexed or sorted
CREATE INDEX idx_a_id1 ON test_joins_a(id1);
CREATE INDEX idx_b_id1 ON test_joins_b(id1);

-- Query to force MERGE JOIN
EXPLAIN ANALYZE
SELECT *
FROM test_joins_a a
JOIN test_joins_b b
ON a.id1 = b.id1;
```

Data Output	Messages	Notifications

```
QUERY PLAN
text
1  Merge Join  (cost=0.57..786.57 rows=10000 width=16) (actual time=0.047..21.187 rows=10000 loops=1)
2    Merge Cond: (a.id1 = b.id1)
3    -> Index Scan using idx_a_id1 on test_joins_a a  (cost=0.29..318.29 rows=10000 width=8) (actual time=0.023..5.227 rows=10000 loop...
4    -> Index Scan using idx_b_id1 on test_joins_b b  (cost=0.29..318.29 rows=10000 width=8) (actual time=0.015..6.123 rows=10000 loop...
5  Planning Time: 0.373 ms
6  Execution Time: 22.423 ms
```

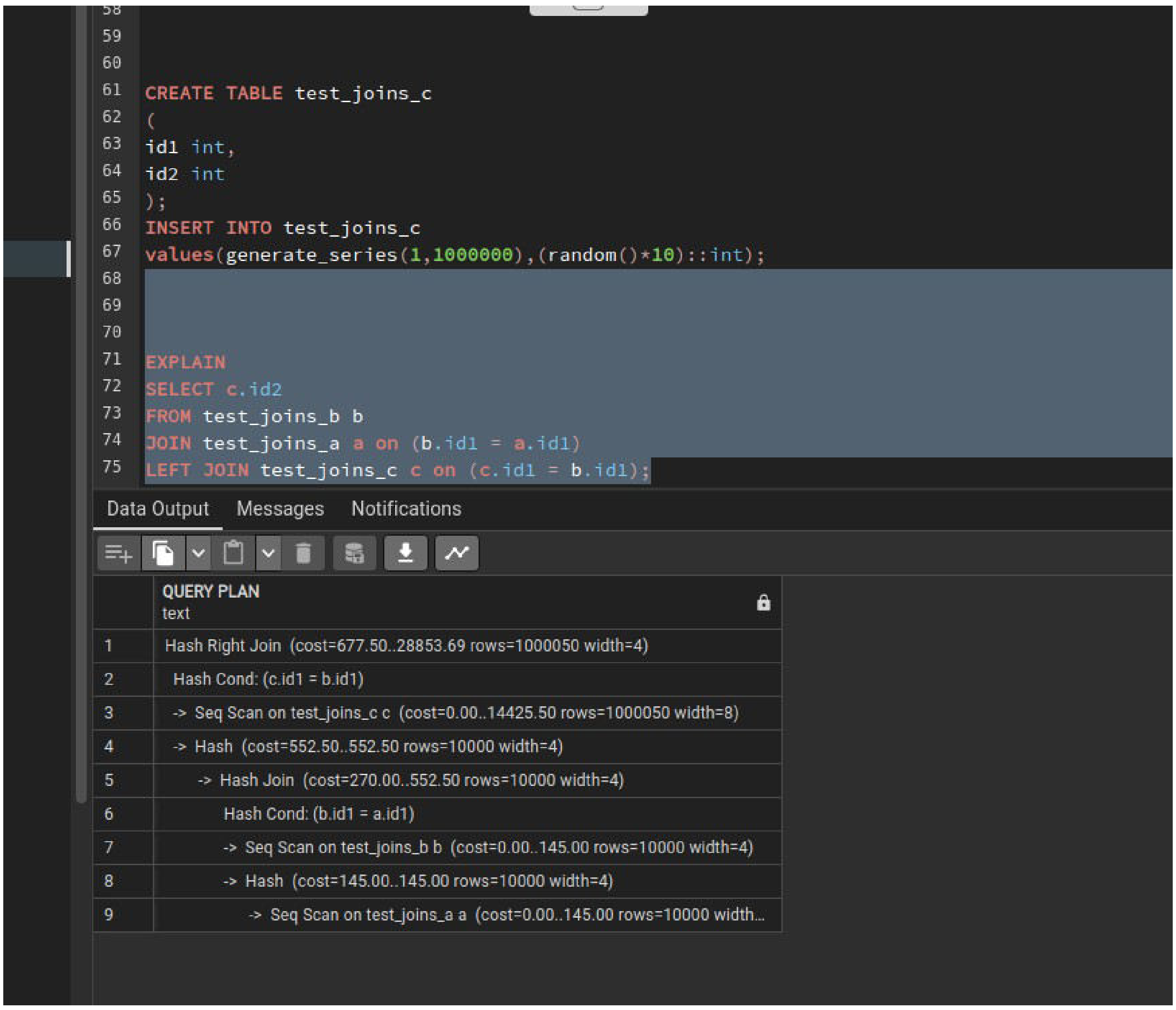
Query for Merge join , first I disabled hashjoin then created index on both tables(make sure the join condition columns are sorted) and execute query.



```
CREATE INDEX idx_a_id1 ON test_joins_a(id1);
CREATE INDEX idx_b_id1 ON test_joins_b(id1);


SET enable_mergejoin = off;
SET enable_nestloop = off
SET enable_mergejoin = on;
SET enable_hashjoin = on;



EXPLAIN ANALYZE
SELECT *
FROM test_joins_a a
JOIN test_joins_b b
ON a.id1 = b.id1;
```

```
QUERY PLAN
text
1  Hash Join  (cost=270.00..552.50 rows=10000 width=16) (actual time=4.770..13.891 rows=10000 loops=1)
2    Hash Cond: (a.id1 = b.id1)
3    ->  Seq Scan on test_joins_a a  (cost=0.00..145.00 rows=10000 width=8) (actual time=0.016..1.862 rows=10000 loops=1)
4    ->  Hash  (cost=145.00..145.00 rows=10000 width=8) (actual time=4.728..4.730 rows=10000 loops=1)
5          Buckets: 16384  Batches: 1  Memory Usage: 519kB
6          ->  Seq Scan on test_joins_b b  (cost=0.00..145.00 rows=10000 width=8) (actual time=0.007..1.592 rows=10000 loop...
7  Planning Time: 0.314 ms
8  Execution Time: 14.825 ms
```

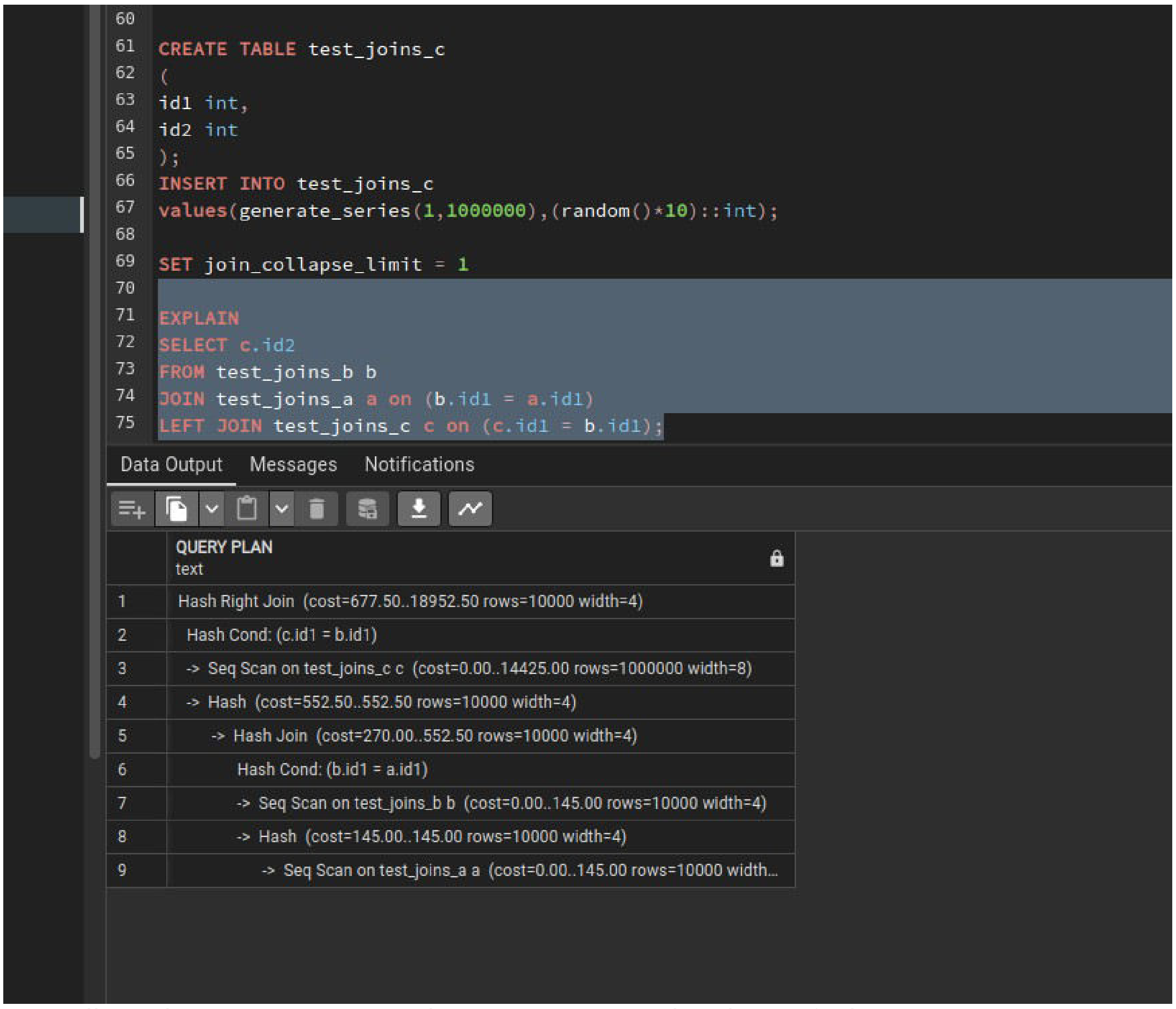
After disabling merge join we can see that postgres choose hash join method for execution.



```
CREATE TABLE test_joins_c
(
id1 int,
id2 int
);
INSERT INTO test_joins_c
values(generate_series(1,1000000),(random()*10)::int);



EXPLAIN
SELECT c.id2
FROM test_joins_b b
JOIN test_joins_a a on (b.id1 = a.id1)
LEFT JOIN test_joins_c c on (c.id1 = b.id1);
```

	QUERY PLAN text
1	Hash Right Join (cost=677.50..28853.69 rows=1000050 width=4)
2	Hash Cond: (c.id1 = b.id1)
3	-> Seq Scan on test_joins_c c (cost=0.00..14425.50 rows=1000050 width=8)
4	-> Hash (cost=552.50..552.50 rows=10000 width=4)
5	-> Hash Join (cost=270.00..552.50 rows=10000 width=4)
6	Hash Cond: (b.id1 = a.id1)
7	-> Seq Scan on test_joins_b b (cost=0.00..145.00 rows=10000 width=4)
8	-> Hash (cost=145.00..145.00 rows=10000 width=4)
9	-> Seq Scan on test_joins_a a (cost=0.00..145.00 rows=10000 width...

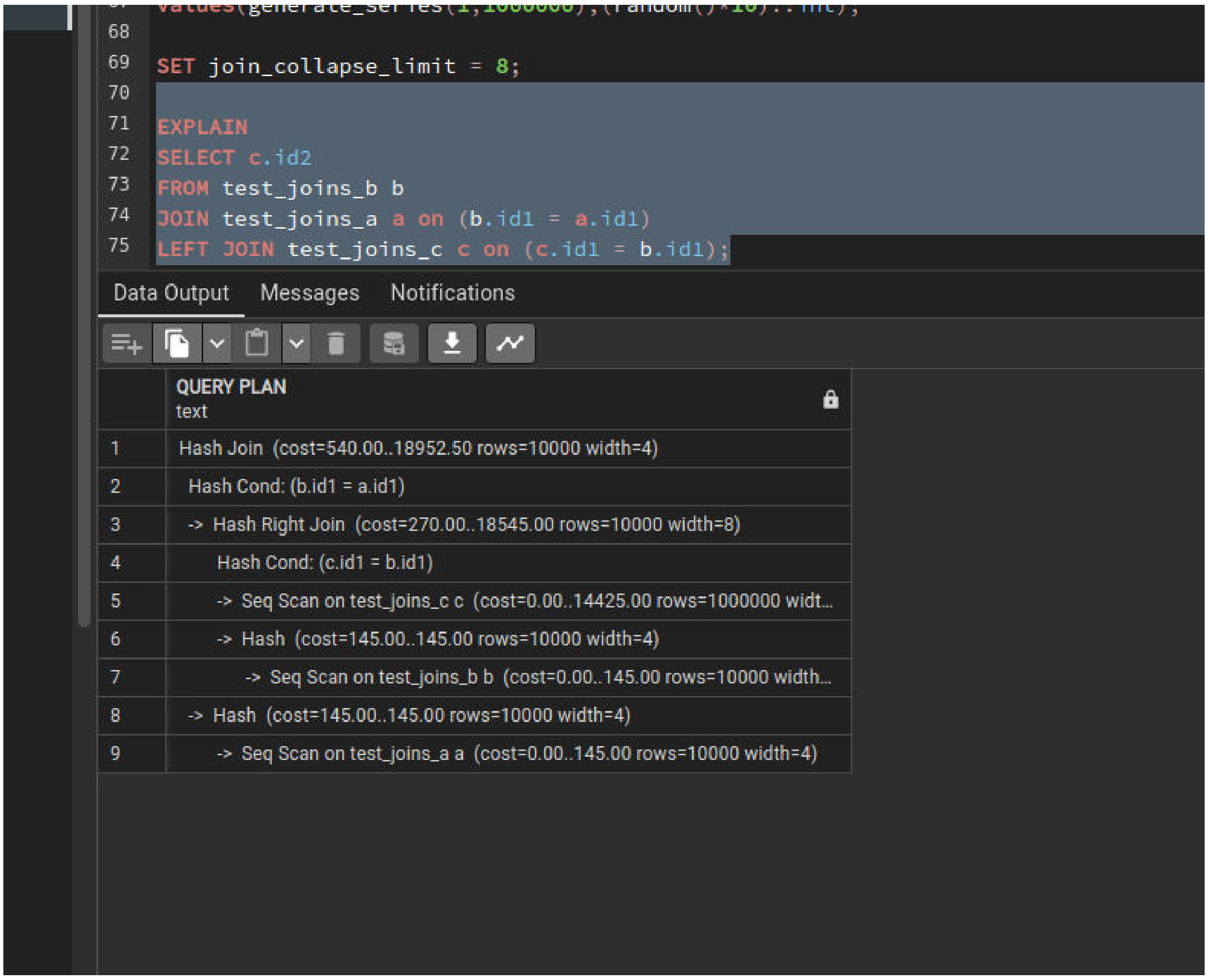
Table created and in query plan we see that first it secuentially scan on test_joins_a table , then Hashing it , next it goes to other table secuentially then hashed it too , then hash join between tables.After than it starts same process with third table , first sequentially then hash join with other joined table.



```
CREATE TABLE test_joins_c
(
id1 int,
id2 int
);
INSERT INTO test_joins_c
values(generate_series(1,1000000),(random()*10)::int);

SET join_collapse_limit = 1

EXPLAIN
SELECT c.id2
FROM test_joins_b b
JOIN test_joins_a a on (b.id1 = a.id1)
LEFT JOIN test_joins_c c on (c.id1 = b.id1);
```

	QUERY PLAN text
1	Hash Right Join (cost=677.50..18952.50 rows=10000 width=4)
2	Hash Cond: (c.id1 = b.id1)
3	-> Seq Scan on test_joins_c c (cost=0.00..14425.00 rows=1000000 width=8)
4	-> Hash (cost=552.50..552.50 rows=10000 width=4)
5	-> Hash Join (cost=270.00..552.50 rows=10000 width=4)
6	Hash Cond: (b.id1 = a.id1)
7	-> Seq Scan on test_joins_b b (cost=0.00..145.00 rows=10000 width=4)
8	-> Hash (cost=145.00..145.00 rows=10000 width=4)
9	-> Seq Scan on test_joins_a a (cost=0.00..145.00 rows=10000 width...

join_collapse_limit set to 1 , we see that it start to use Hash Right join for last step.

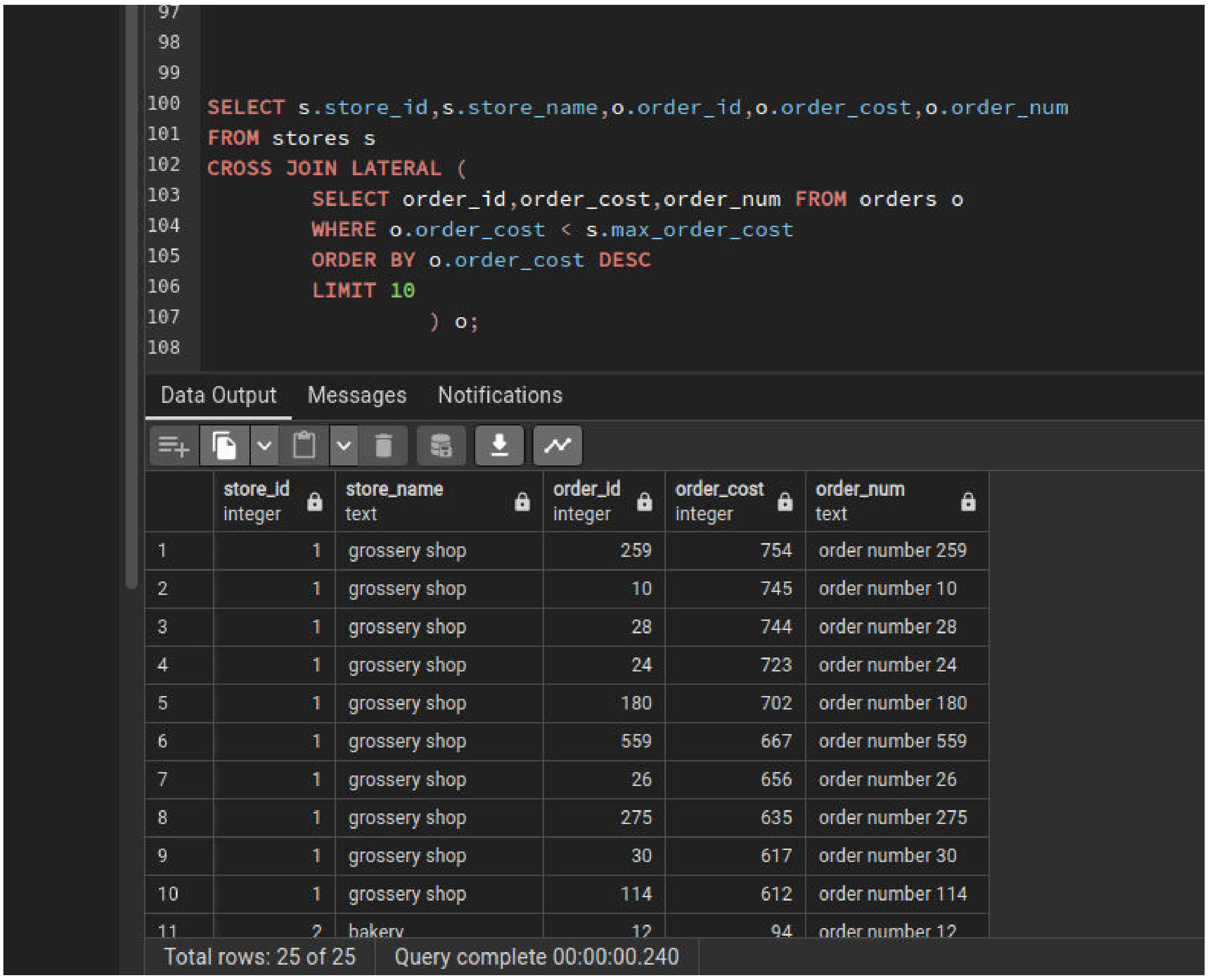


```
values(generate_series(1,1000000),(random()*10)::int);

SET join_collapse_limit = 8;

EXPLAIN
SELECT c.id2
FROM test_joins_b b
JOIN test_joins_a a on (b.id1 = a.id1)
LEFT JOIN test_joins_c c on (c.id1 = b.id1);
```

Data Output   Messages   Notifications

```
QUERY PLAN
text
1  Hash Join  (cost=540.00..18952.50 rows=10000 width=4)
2    Hash Cond: (b.id1 = a.id1)
3    ->  Hash Right Join  (cost=270.00..18545.00 rows=10000 width=8)
4          Hash Cond: (c.id1 = b.id1)
5          ->  Seq Scan on test_joins_c c  (cost=0.00..14425.00 rows=1000000 widt...
6          ->  Hash  (cost=145.00..145.00 rows=10000 width=4)
7                ->  Seq Scan on test_joins_b b  (cost=0.00..145.00 rows=10000 width...
8    ->  Hash  (cost=145.00..145.00 rows=10000 width=4)
9          ->  Seq Scan on test_joins_a a  (cost=0.00..145.00 rows=10000 width=4)
```

Returned to 8.



```
SELECT s.store_id,s.store_name,o.order_id,o.order_cost,o.order_num
FROM stores s
CROSS JOIN LATERAL (
        SELECT order_id,order_cost,order_num FROM orders o
        WHERE o.order_cost < s.max_order_cost
        ORDER BY o.order_cost DESC
        LIMIT 10
                ) o;
```

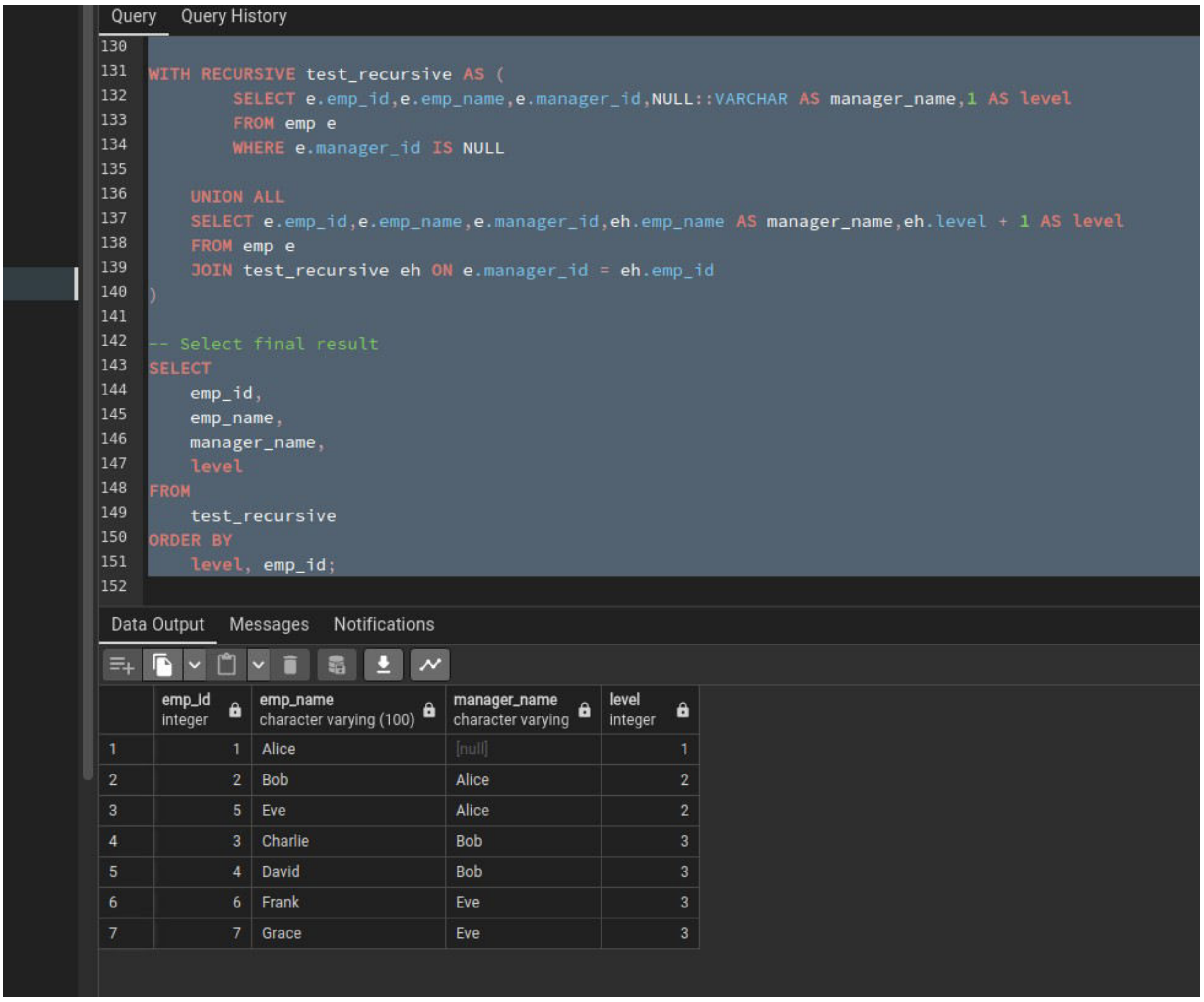
Data Output Messages Notifications

	store_id integer	store_name text	order_id integer	order_cost integer	order_num text
1	1	grossery shop	259	754	order number 259
2	1	grossery shop	10	745	order number 10
3	1	grossery shop	28	744	order number 28
4	1	grossery shop	24	723	order number 24
5	1	grossery shop	180	702	order number 180
6	1	grossery shop	559	667	order number 559
7	1	grossery shop	26	656	order number 26
8	1	grossery shop	275	635	order number 275
9	1	grossery shop	30	617	order number 30
10	1	grossery shop	114	612	order number 114
11	2	bakery	12	94	order number 12

Total rows: 25 of 25 Query complete 00:00:00.240

s

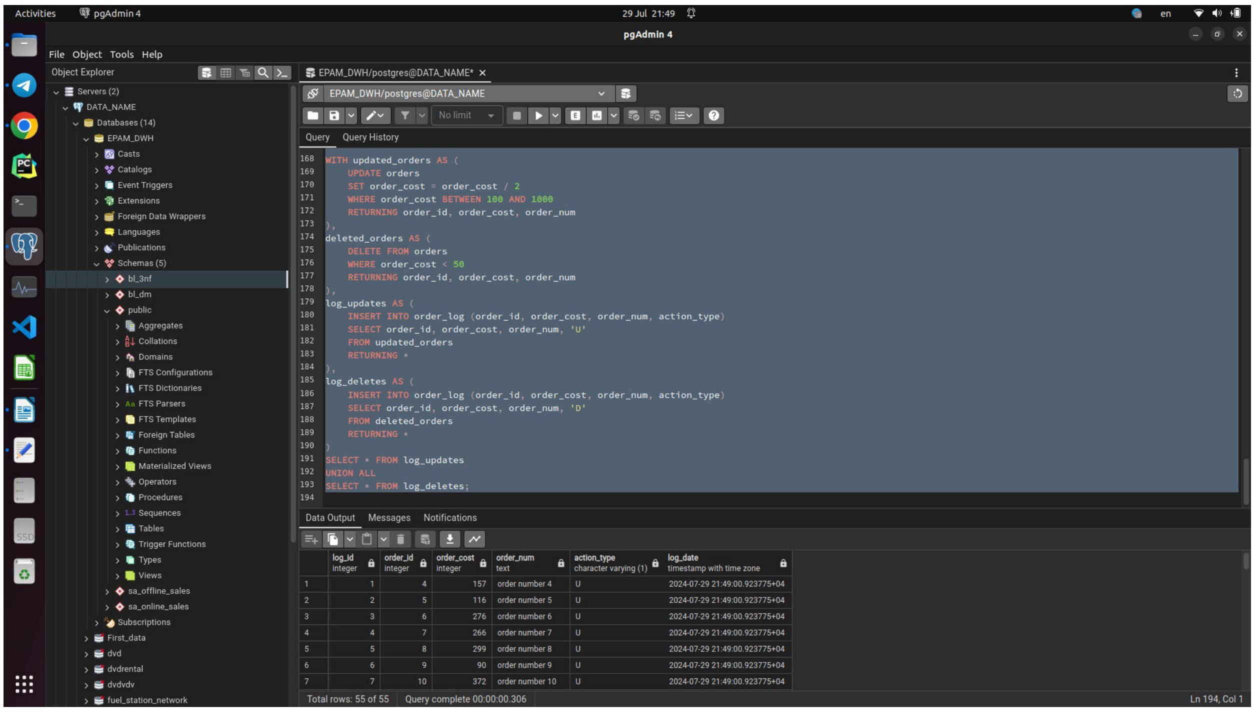
Query for TOP 10 of orders by cost for each store used LATERAL JOIN.



```sql

WITH RECURSIVE test_recursive AS (
        SELECT e.emp_id,e.emp_name,e.manager_id,NULL::VARCHAR AS manager_name,1 AS level
        FROM emp e
        WHERE e.manager_id IS NULL

    UNION ALL
    SELECT e.emp_id,e.emp_name,e.manager_id,eh.emp_name AS manager_name,eh.level + 1 AS level
    FROM emp e
    JOIN test_recursive eh ON e.manager_id = eh.emp_id
)

-- Select final result
SELECT
    emp_id,
    emp_name,
    manager_name,
    level
FROM
    test_recursive
ORDER BY
    level, emp_id;

```

	emp_id integer	emp_name character varying (100)	manager_name character varying	level integer
1	1	Alice	[null]	1
2	2	Bob	Alice	2
3	5	Eve	Alice	2
4	3	Charlie	Bob	3
5	4	David	Bob	3
6	6	Frank	Eve	3
7	7	Grace	Eve	3

Recursive query for selectiong emloyees.



```sql
WITH updated_orders AS (
    UPDATE orders
    SET order_cost = order_cost / 2
    WHERE order_cost BETWEEN 100 AND 1000
    RETURNING order_id, order_cost, order_num
),
deleted_orders AS (
    DELETE FROM orders
    WHERE order_cost < 50
    RETURNING order_id, order_cost, order_num
),
log_updates AS (
    INSERT INTO order_log (order_id, order_cost, order_num, action_type)
    SELECT order_id, order_cost, order_num, 'U'
    FROM updated_orders
    RETURNING *
),
log_deletes AS (
    INSERT INTO order_log (order_id, order_cost, order_num, action_type)
    SELECT order_id, order_cost, order_num, 'D'
    FROM deleted_orders
    RETURNING *
)
SELECT * FROM log_updates
UNION ALL
SELECT * FROM log_deletes;
```

	log_id integer	order_id integer	order_cost integer	order_num text	action_type character varying (1)	log_date timestamp with time zone
1	1	4	157	order number 4	U	2024-07-29 21:49:00.923775+04
2	2	5	116	order number 5	U	2024-07-29 21:49:00.923775+04
3	3	6	276	order number 6	U	2024-07-29 21:49:00.923775+04
4	4	7	266	order number 7	U	2024-07-29 21:49:00.923775+04
5	5	8	299	order number 8	U	2024-07-29 21:49:00.923775+04
6	6	9	90	order number 9	U	2024-07-29 21:49:00.923775+04
7	7	10	372	order number 10	U	2024-07-29 21:49:00.923775+04

Total rows: 55 of 55 Query complete 00:00:00.306

Changing data usin CTE’s ,