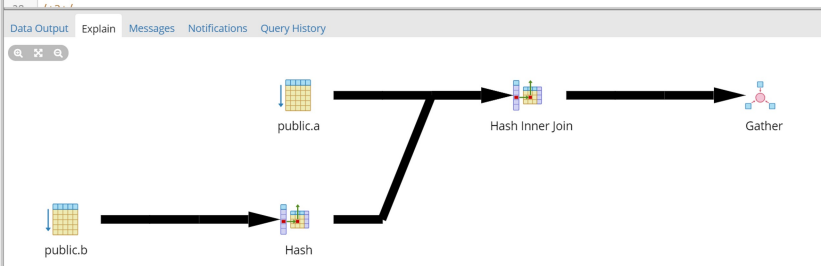
Appendix

Sunday, November 4, 2018 16:19

Section 1: Explain Query Plans

1. A.pk = B.pk

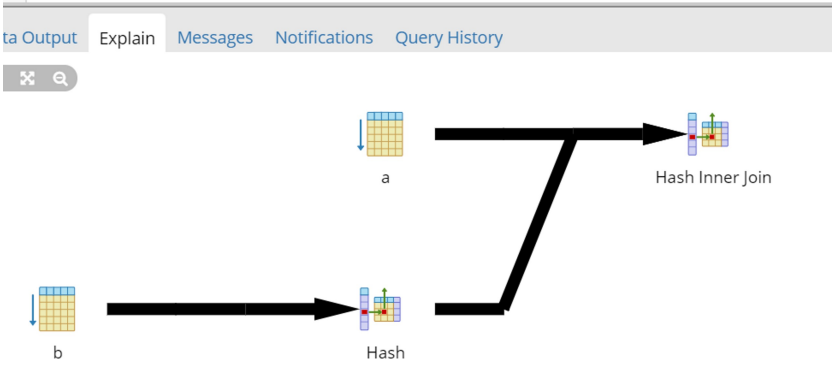
```
/** SECTION 1 **/
/*1*/
SELECT * FROM A JOIN B ON A.pk =B.pk;
```



System Chooses Hash Join. Hashes B and scans over A to do hash inner join
Expected Time:1338135.43
Expected Size:5000016 rows
T(A) = 5000000 V(A,pk) = 5000000
T(B) = 5000000 V(B,pk) = 5000000
The selected plan is hash join.

2. A.ht = B.ht

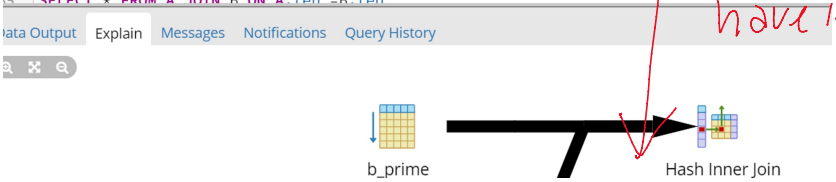
```
/*2*/
SELECT * FROM A JOIN B ON A.ht =B.ht;

/*3*/
```



T(A) = 5000000 V(A,ht) = 100000
T(B) = 5000000 V(B,ht) = 100000

3. A'.ht = B'.ht

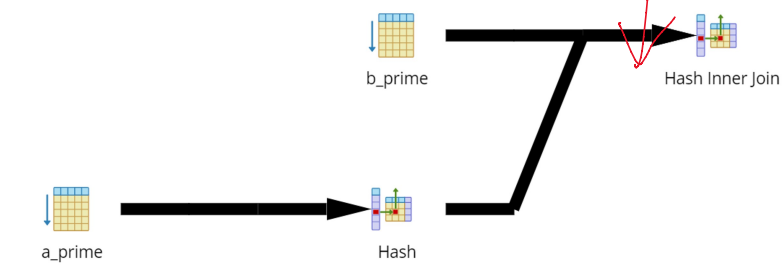
```
/*3*/
SELECT * FROM A_prime JOIN B_prime ON A_prime.ht =B_prime.ht;

/*4*/
SELECT * FROM A JOIN B ON A.ten =B.ten;
```



Same plan even though A', B' have indices on ht.

```
/*3*/
SELECT * FROM A_prime JOIN B_prime ON A_prime.ht =B_prime.ht;

/*4*/
SELECT * FROM A JOIN B ON A.ten =B.ten;
```

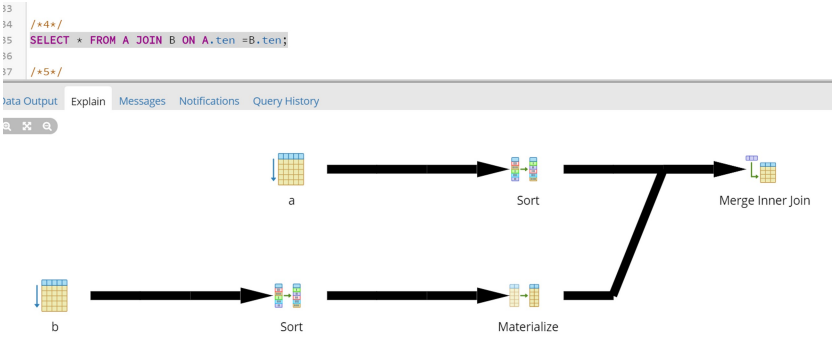
Data Output | Explain | Messages | Notifications | Query History



T(A') = 5000000 V(A',ht) = 100000
T(B') = 5000000 V(B',ht) = 100000

V(R,attrib) did not change the query plan across 1,2,3. The secondary index has no impact on the selected query plan.

4. A.ten = B.ten

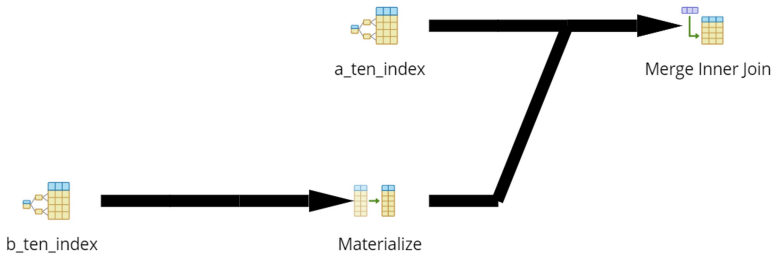


T(A) = 5000000 V(A,ten) = 10
T(B) = 5000000 V(B,ten) = 10
Query plan selects Merge join

5. A'.ten = B'.ten

```
/*5*/
SELECT * FROM A_prime JOIN B_prime ON A_prime.ten =B_prime.ten;

/*6*/
```

Data Output | Explain | Messages | Notifications | Query History

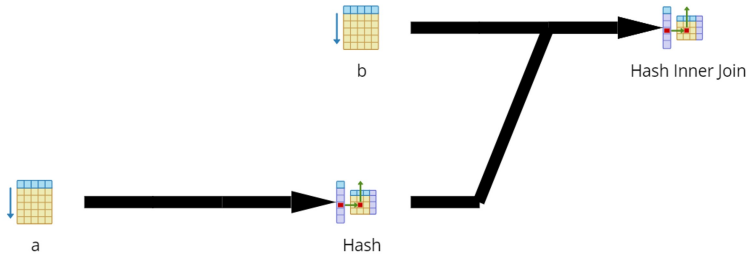


T(A) = 5000000 V(A,ten) = 10
T(B) = 5000000 V(B,ten) = 10
Query plan selects Merge join using the ten_index to shortcut data sorting

6. A.ht = B.ten

```
39
40  /*6*/
41  SELECT * FROM A JOIN B ON A.ht =B.ten;
42
43  /*7*/
```

Data Output | Explain | Messages | Notifications | Query History

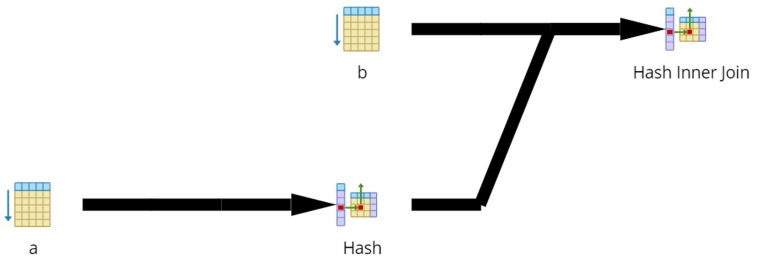


T(A)= 5000000 V(A,ht) = 100000
T(B)= 5000000 V(B,ten) = 10
Selected plan is hash join

7. B.ten = A.ht

```
42
43  /*7*/
44  SELECT * FROM B JOIN A ON B.ten =A.ht;
45
46  /*8*/
```

Data Output | Explain | Messages | Notifications | Query History



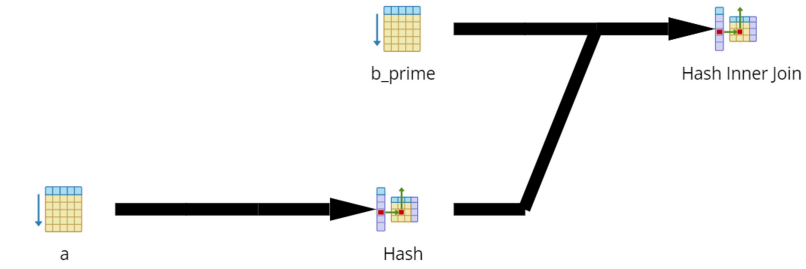
T(A)= 5000000 V(A,ht) = 100000
T(B)= 5000000 V(B,ten) = 10

Selected plan is hash join again. The optimizer chose to hash a, then join b regardless of the order that a,b appear in the JOIN statement

8. A.ht = B'.ten

```
/*8*/
SELECT * FROM A JOIN B_prime ON A.ht =B_prime.ten;

/*9*/
```

ta Output | Explain | Messages | Notifications | Query History

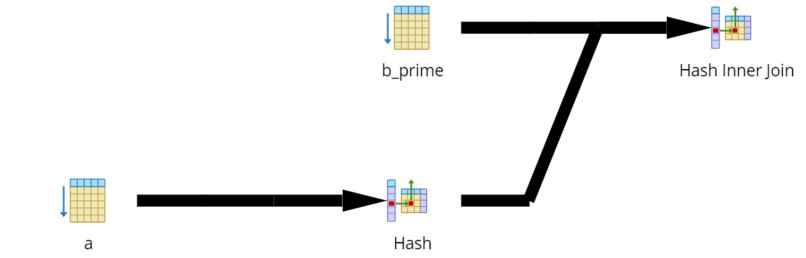


T(A) = 5000000 V(A,ht) = 100000
T(B') = 5000000 V(B',ten)= 10
Selected plan is hash join

9. B'.ten = A.ht

```
48
49  /*9*/
50  SELECT * FROM B_prime JOIN A ON B_prime.ten = A.ht;
51
52  /* 3 way joins */
53  /*10*/
54  SELECT * FROM A JOIN B ON A.ht = B.ht JOIN C ON B.ht = C.ht;
```

Data Output | Explain | Messages | Notifications | Query History

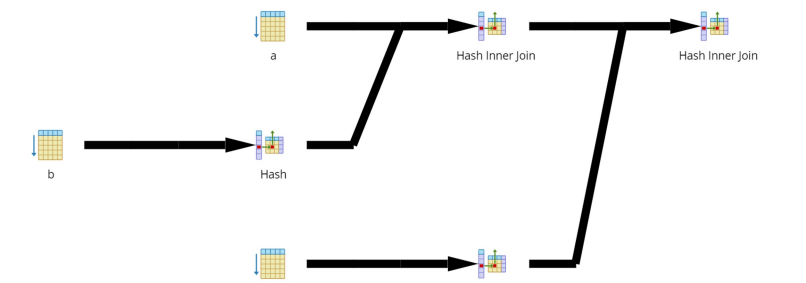


T(A) = 5000000 V(A,ht) = 100000
T(B') = 5000000 V(B',ten)= 10
Selected plan is hash join. The optimizer again chose its own order regardless of how a,b appear in the JOIN statement

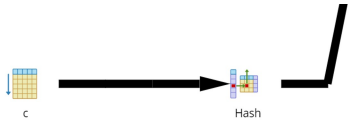
10. A.ht = B.ht = C.ht

```
/* 3 way joins */
/*10*/
SELECT * FROM A JOIN B ON A.ht = B.ht JOIN C ON B.ht = C.ht;

/*11*/
SELECT * FROM A JOIN B ON A.ten = B.ten JOIN C ON B.ten = C.ten;
```

ata Output | Explain | Messages | Notifications | Query History

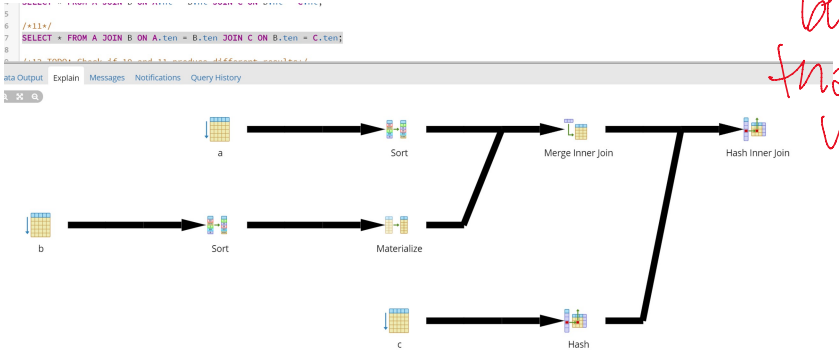


limited



T(A) = 5000000 V(A,ht) = 100000
T(B) = 5000000 V(B,ht) = 100000
T(c)= 5000000 V(C,ht) = 100000
Selected plan: hash join A,b, hash join c

11. A.ten = B.ten = C.ten



optimizer has decided its better to use hash join than to sort the 10^5 values in the attribute.

T(A) = 5000000 V(A,ten) = 10
T(B) = 5000000 V(B,ten) = 10
T(c)= 5000000 V(C,ten) = 10
Selected plan: merge join A,B, hash join c

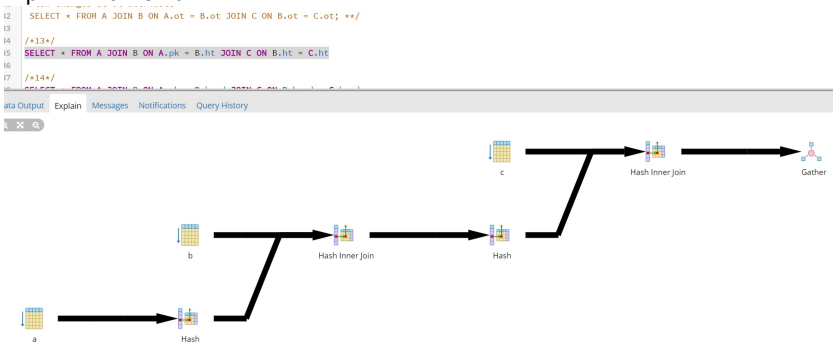
12. Do they produce different plans?

Yes, the plans are different. I found the plan changes on the ot attribute when V(R,ot) = 1000

optimizer has decided its better to sort the 10 values in the table then do merge join

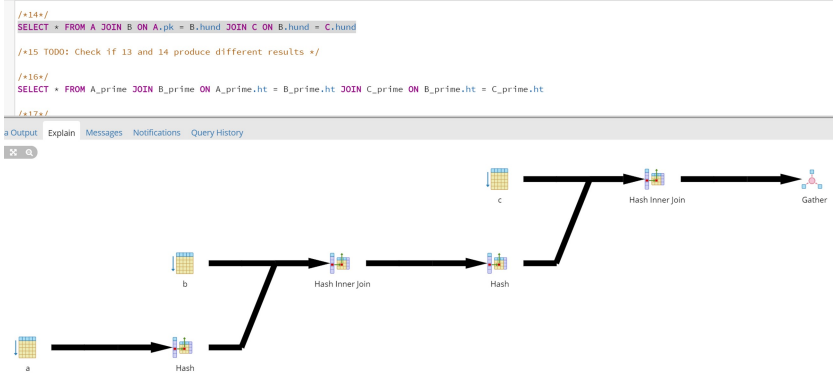
A.ot = B.ot = C.ot

13. A.pk = B.ht = C.ht



Selected pla n is to hash join a,b then hash join c
T(A) = 5000000 V(A,pk) = 5000000
T(B) = 5000000 V(B,ht) = 100000
T(C) = 5000000 V(C,ht) = 100000

14. A.pk = B.hund = C.hund



Selected pla n is to hash join a,b then hash join c

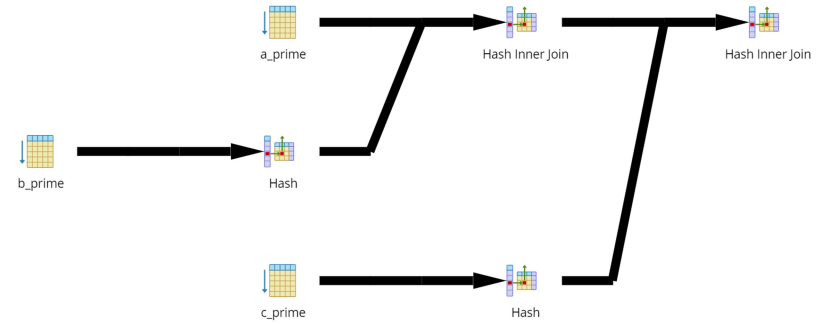
T(A) = 5000000 V(A,pk) = 5000000
T(B) = 5000000 V(B,hund) = 100
T(C) = 5000000 V(C,hund) = 100

15. Do they produce different plans?
    No, 13 and 14 produce the same plan.

16. A'.ht = B'.ht = C'.ht

```
/*16*/
SELECT * FROM A_prime JOIN B_prime ON A_prime.ht = B_prime.ht JOIN C_prime ON B_prime.ht = C_prime.ht

/*17*/
SELECT * FROM A_prime JOIN B_prime ON A_prime.hund = B_prime.hund JOIN C_prime ON B_prime.hund = C_prime.hund
```

Output | Explain | Messages | Notifications | Query History

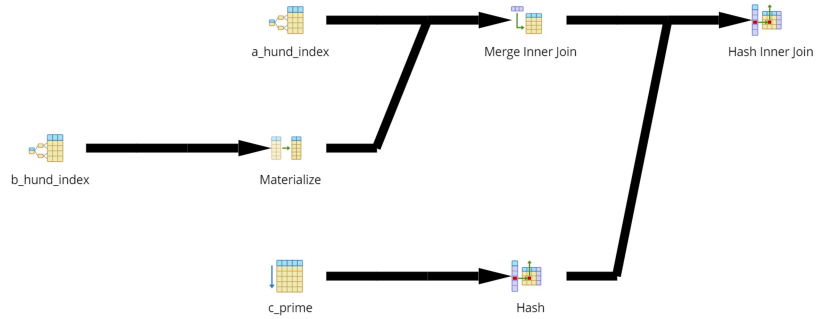


T(A) = 5000000 V(A,ht) = 100000
T(B) = 5000000 V(B,ht) = 100000
T(c)= 5000000 V(C,ht) = 100000
Selected plan: hash join A,b, hash join c. The query plan does not change from 10, so the secondary index has no effect on the plan

17. A'.hund = B'.hund = C'.hund

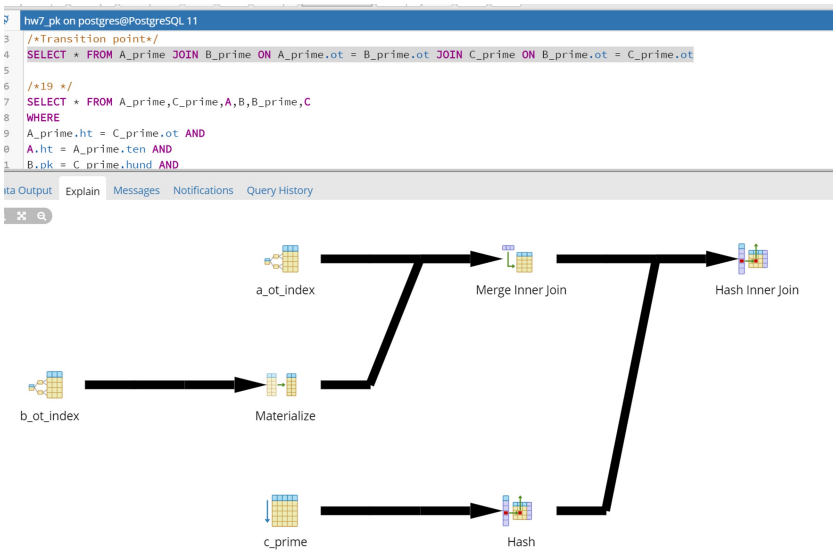
```
/*17*/
SELECT * FROM A_prime JOIN B_prime ON A_prime.hund = B_prime.hund JOIN C_prime ON B_prime.hund = C_prime.hund

/*18 */
SELECT * FROM A_prime JOIN B_prime ON A_prime.ht = B_prime.ten JOIN C_prime ON B_prime.ten = C_prime.ht
```

Output | Explain | Messages | Notifications | Query History



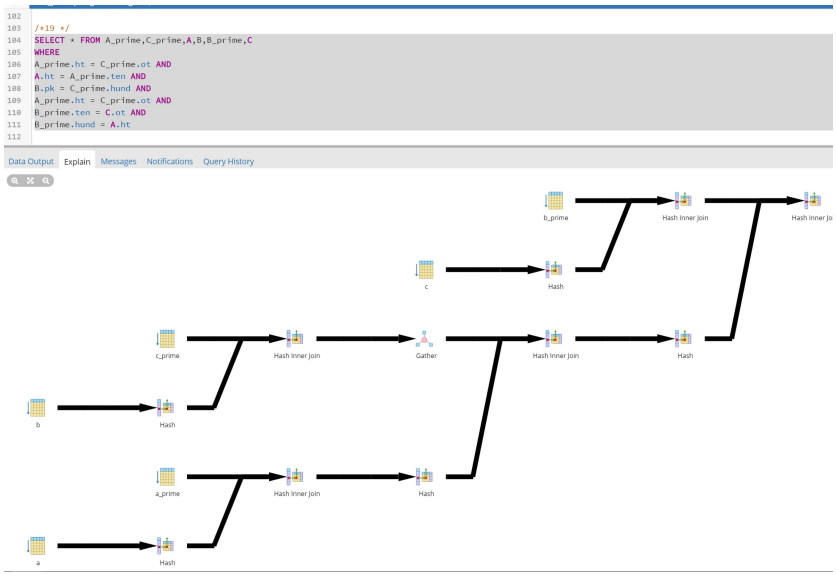
T(A) = 5000000 V(A,hund) = 100
T(B) = 5000000 V(B,hund) = 100
T(c)= 5000000 V(C,hund) = 100
Selected plan: merge join A,b, hash join c. The query plan changes from 16.

18. Do 16 and 17 produce different results?
    Yes. 16 chooses hash join to join a,b. 17 chooses merge join. The transition point is at the ot attribute



Section 2:

19. A'.ht = C'.ot AND
    A.ht = A'.ten AND
    B.pk = C'.hund AND
    A'.ht = C'.ot AND
    B'.ten = C.ot AND
    B'.hund = A.ht

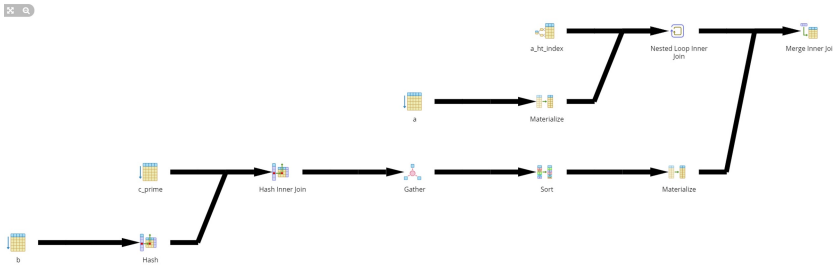


The estimated size in the last hash inner join is over 3 quadrillion rows (3,172,245,629,574,755)

20. A'ht = C'.ot AND
    A'.ten = 5 AND
    A.ht < A'.ten AND
    B.pk = C'.hund AND
    B.ot < 500 AND
    A'.ht = C'.ot

```
/*20*/
SELECT * FROM A_prime,C_prime,A,B
WHERE
A_prime.ht = C_prime.ot AND
A_prime.ten = 5 AND
A.ht < A_prime.ten AND
B.pk = C_prime.hund AND
B.ot < 500 AND
A_prime.ht = C_prime.ot
```

Output | Explain | Messages | Notifications | Query History

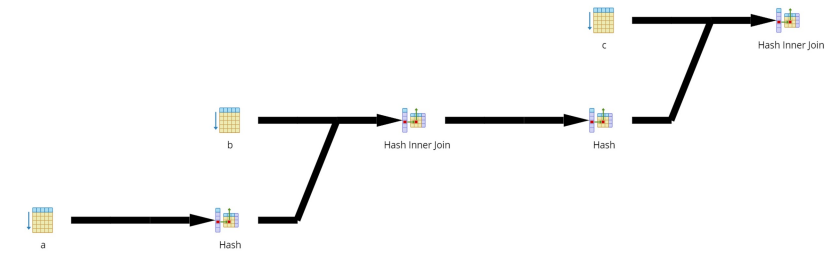


The estimated size of the last merge inner join is: over 21 trillion rows(21,554,197,796,542)!
The compiler did not notice the equivalent predicate.

21. A.ht = B.tt = C.ot

```
/*21*/
SELECT * FROM A
JOIN B ON A.ht = B.tt
JOIN C ON B.tt = C.ot

/*22*/
SELECT * FROM C
JOIN B ON C.ot = B.tt
```

ata Output | Explain | Messages | Notifications | Query History

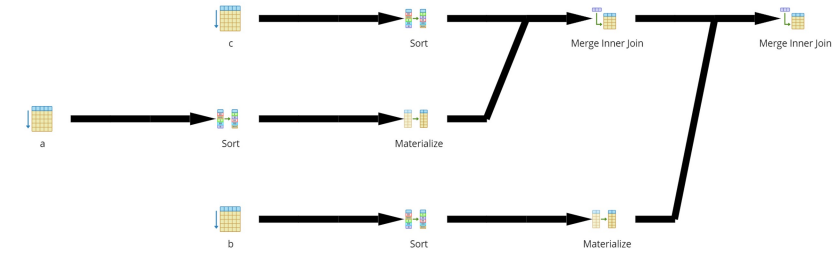


T(A) = 5000000 V(A,ht) = 100000
T(B) = 5000000 V(B,tt) = 10000
T(C) = 5000000 V(C,ot) = 1000

22. C.ot = B.tt = A.ht

```
/*22*/
SELECT * FROM C
JOIN B ON C.ot = B.tt
JOIN A ON B.tt = A.ht
```

ata Output | Explain | Messages | Notifications | Query History

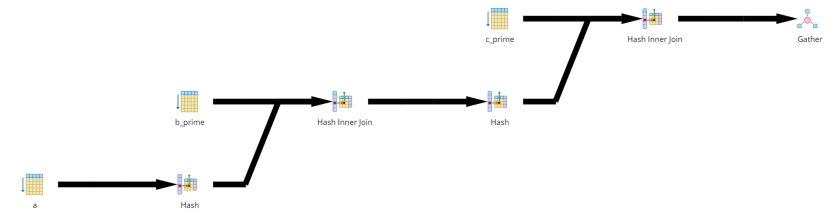


The query plan changes between 21 and 22 even though the only difference is the order of the tables in the query.
T(A) = 5000000 V(A,ht) = 100000
T(B) = 5000000 V(B,tt) = 10000
T(C) = 5000000 V(C,ot) = 1000

23. A.pk = B'.tt = C'.ot

```
/*23*/
SELECT * FROM A
JOIN B_prime ON A.pk = B_prime.tt
JOIN C_prime ON B_prime.tt = C_prime.ot

/*24*/
SELECT *FROM C_prime
JOIN B_prime ON C_prime.ot = B_prime.tt
JOIN A ON B_prime.tt = A.pk
```

Data Output | Explain | Messages | Notifications | Query History

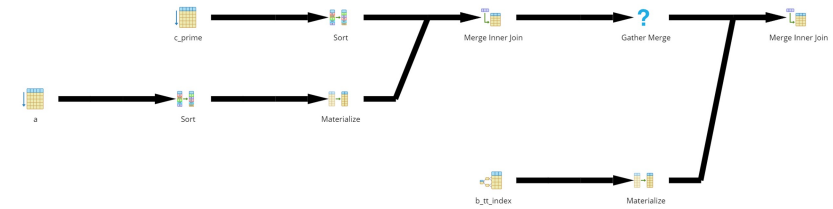


T(A) = 5000000 T(A,pk) = 5000000
T(B) = 5000000 T(B,tt) = 10000
T(C) = 5000000 T(C,ot) = 1000

24. C'.ot = B'.tt = A.pk

```
/*24*/
SELECT *FROM C_prime
JOIN B_prime ON C_prime.ot = B_prime.tt
JOIN A ON B_prime.tt = A.pk
```

Data Output | Explain | Messages | Notifications | Query History



T(A) = 5000000 T(A,pk) = 5000000
T(B) = 5000000 T(B,tt) = 10000
T(C) = 5000000 T(C,ot) = 1000
Again, the query plan changes between 23 and 24 even though the only difference is the order of the tables in the query