Query 9

- Find the names of sailors who have reserved both a red and a green boat.

Note !

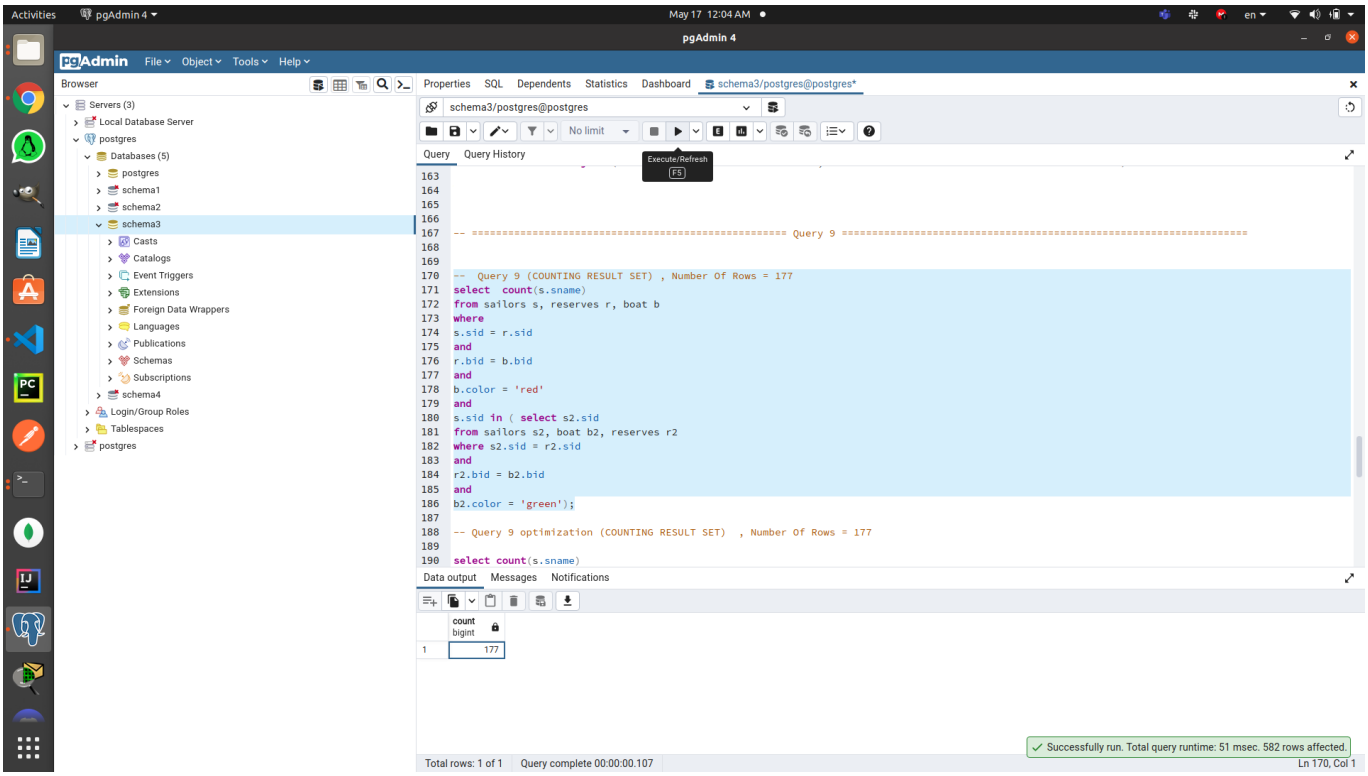
- Flags Hashjoin and HashAgg here where disabled for future after many trials and errors , I've discovered the best way to show the difference interms of the cost and to beat the Postgres Query Optimizer Algorithm to be able to show indices effect and cost differenes .
- The Execution time was changing by 10 Ms in each Execution which is considered high and I can't take it as a measurable Metric because it was Linux (Ubuntu) Operating System performance and I took permission from Prof. Wael as do not take it as my main objective I take the Overall Cost and Compare it .

Original Query

```
select s.sname
from sailors s, reserves r, boat b
where
s.sid = r.sid
and
r.bid = b.bid
and
b.color = 'red'
and
s.sid in ( select s2.sid
from sailors s2, boat b2, reserves r2
where s2.sid = r2.sid
and
r2.bid = b2.bid
and
b2.color = 'red');
```

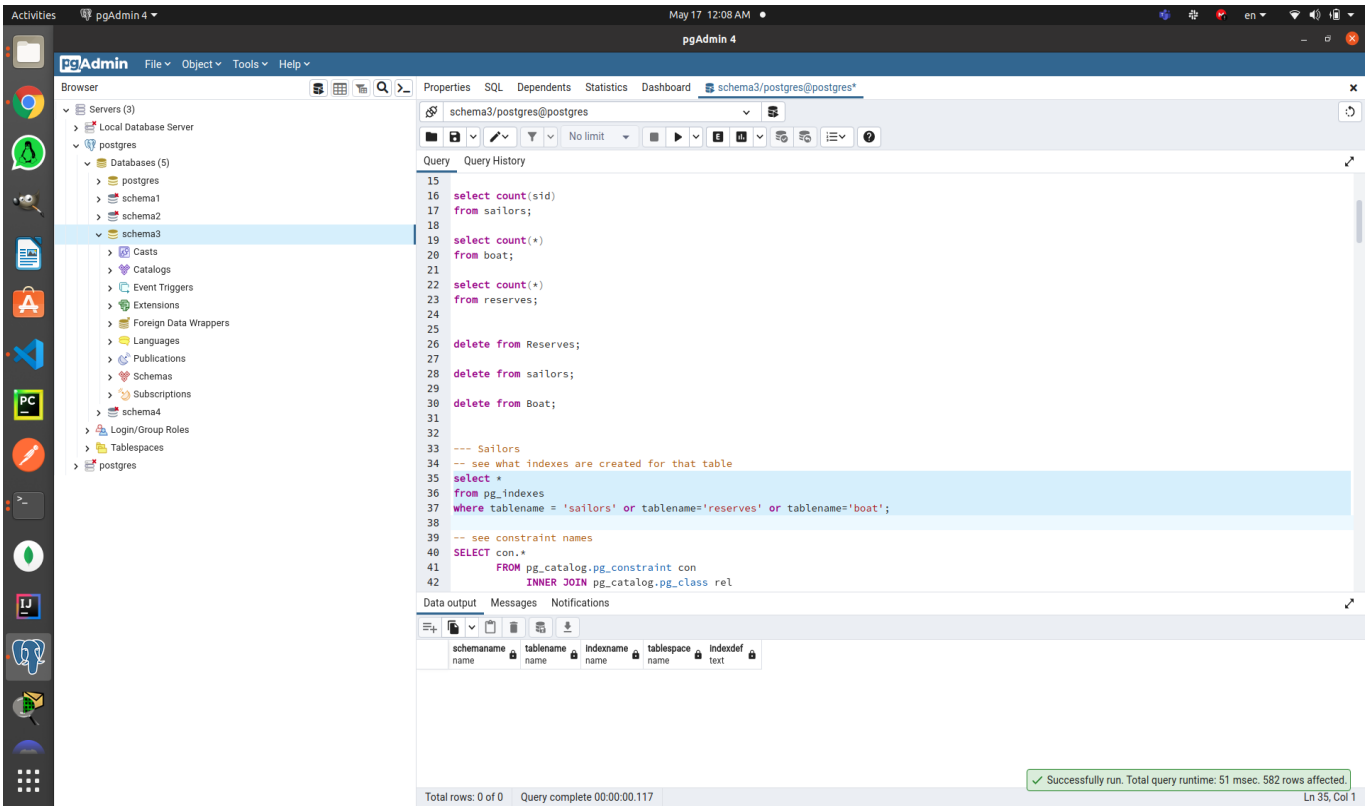
Result Set

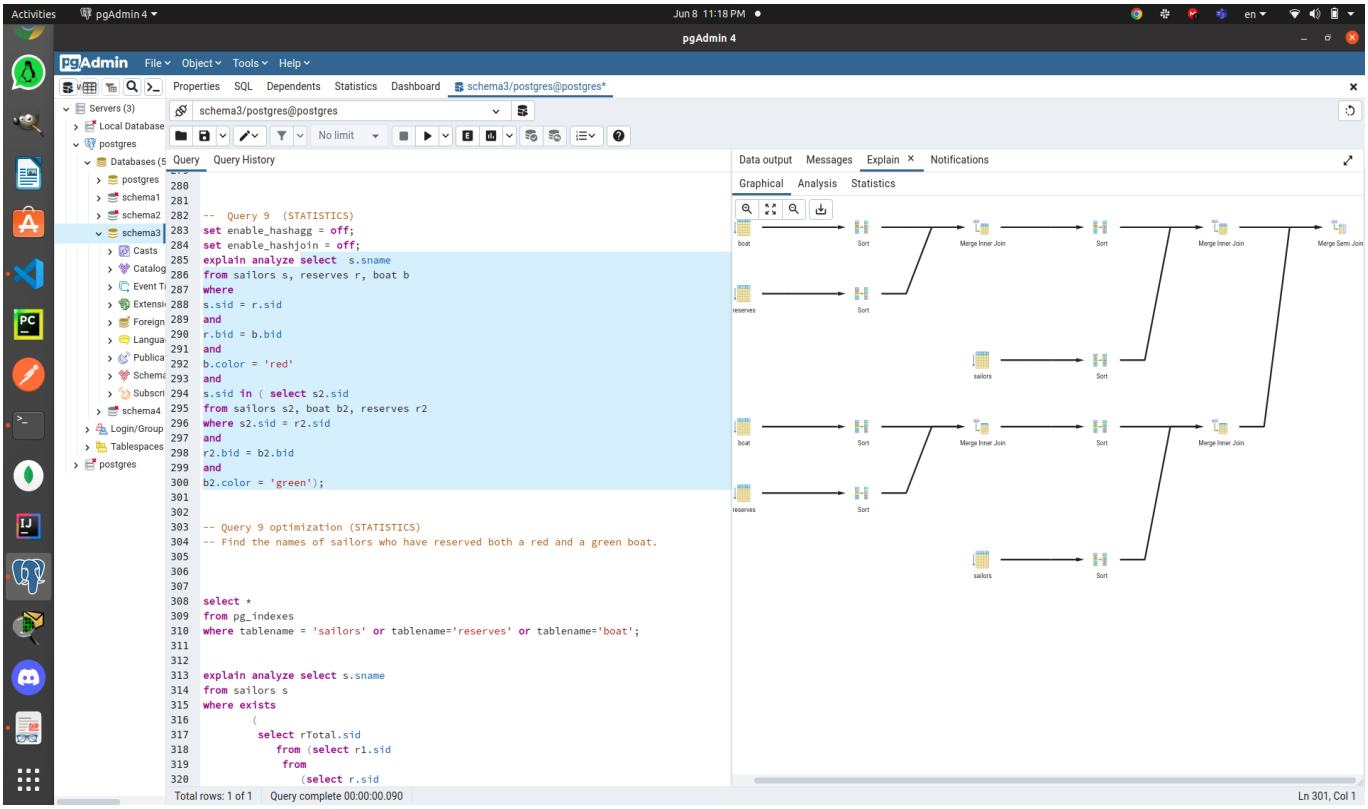
- 177 Rows

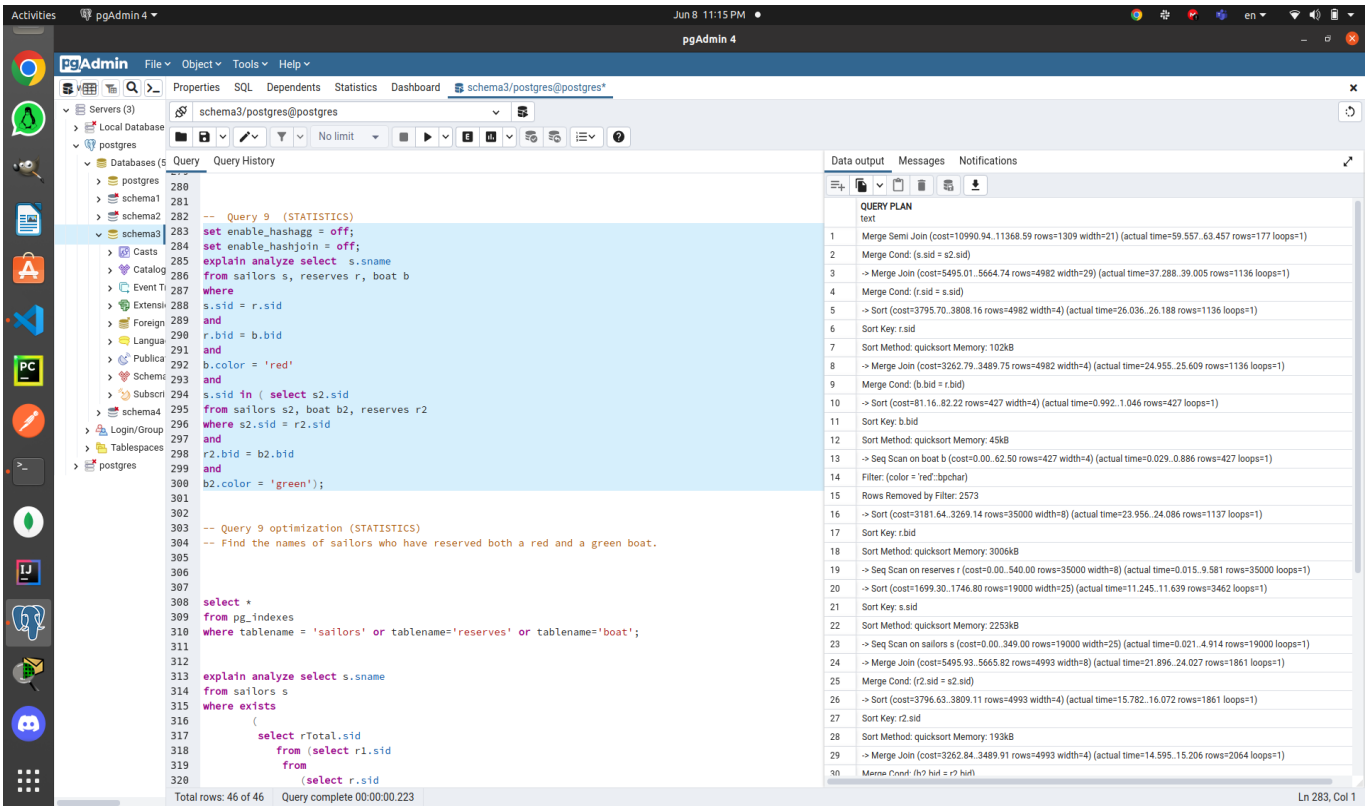


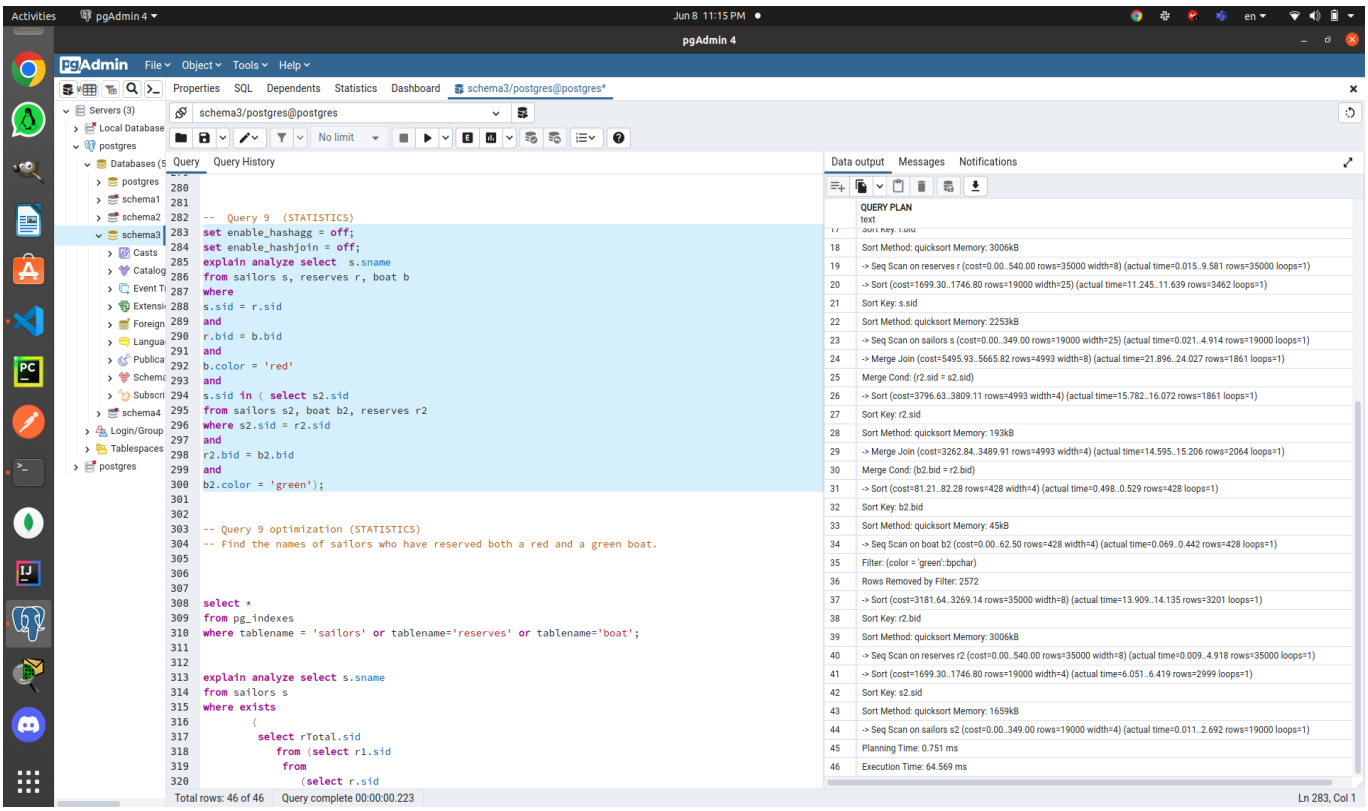
Report

1. given query without an index :


Activities
pgAdmin 4
May 17 12:08 AM
pgAdmin 4
File Object Tools Help
Browser
Properties SQL Dependents Statistics Dashboard schema3/postgres@postgres*
schema3/postgres@postgres
No limit
Query Query History
select count(sid)
from sailors;
select count(*)
from boat;
select count(*)
from reserves;
delete from Reserves;
delete from sailors;
delete from Boat;
--- Sailors
-- see what indexes are created for that table
select *
from pg_indexes
where tablename = 'sailors' or tablename='reserves' or tablename='boat';
-- see constraint names
SELECT con.*
FROM pg_catalog.pg_constraint con
INNER JOIN pg_catalog.pg_class rel
Data output Messages Notifications
schemaname tablename indexname tablespace indexdef
name name name name text
Successfully run. Total query runtime: 51 msec. 582 rows affected.
Total rows: 0 of 0
Query complete 00:00:00.117
Ln 35, Col 1


Activities
pgAdmin 4
Jun 8 11:18 PM
pgAdmin 4
File Object Tools Help
Properties SQL Dependents Statistics Dashboard schema3/postgres@postgres*
schema3/postgres@postgres
No limit
Query Query History
-- Query 9 (STATISTICS)
set enable_hashagg = off;
set enable_hashjoin = off;
explain analyze select s.sname
from sailors s, reserves r, boat b
where
s.sid = r.sid
and
r.bid = b.bid
and
b.color = 'red'
and
s.sid in (select s2.sid
from sailors s2, boat b2, reserves r2
where s2.sid = r2.sid
and
r2.bid = b2.bid
and
b2.color = 'green');
-- Query 9 optimization (STATISTICS)
-- Find the names of sailors who have reserved both a red and a green boat.
select *
from pg_indexes
where tablename = 'sailors' or tablename='reserves' or tablename='boat';
explain analyze select s.sname
from sailors s
where exists
(
select rTotal.sid
from (select r1.sid
from
(select r.sid
Data output Messages Explain Notifications
Graphical Analysis Statistics
Total rows: 1 of 1
Query complete 00:00:00.090
Ln 301, Col 1



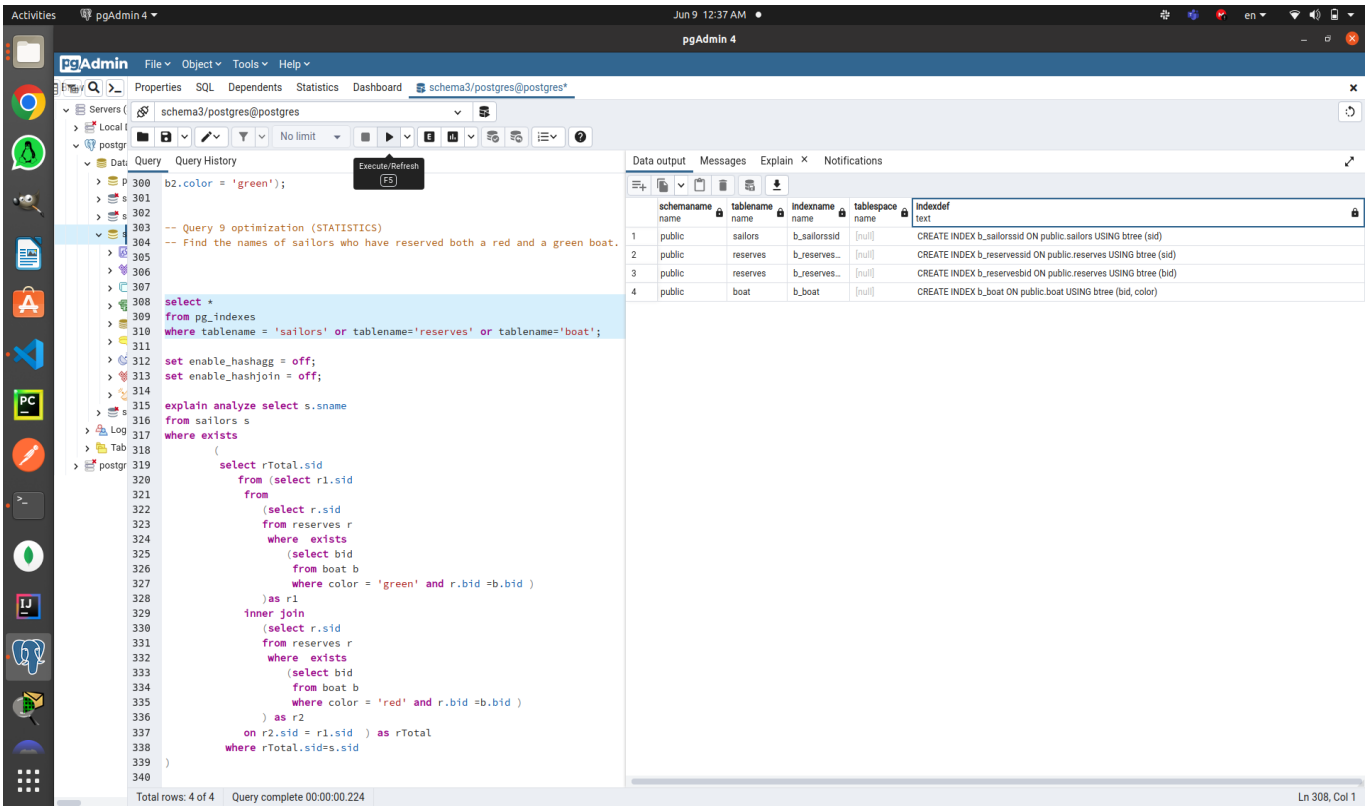


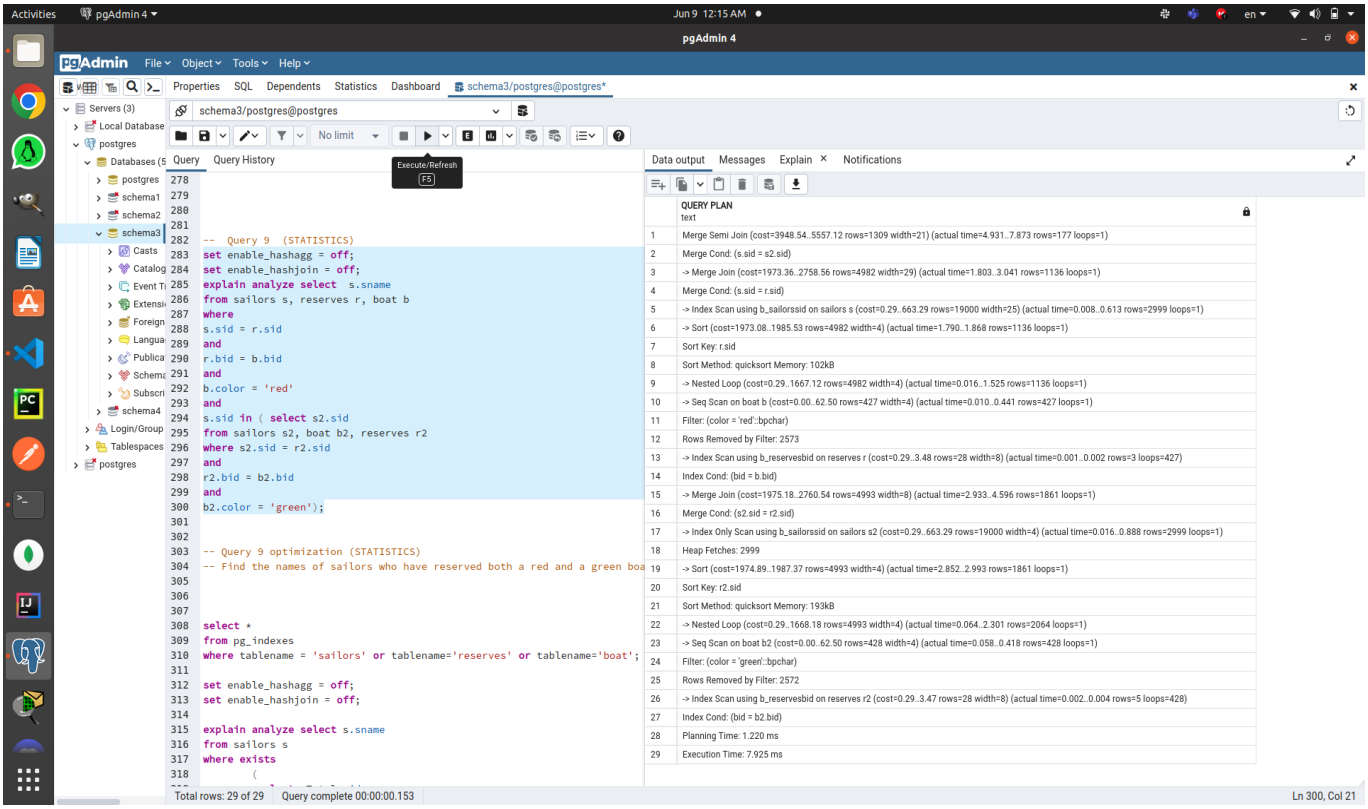
Explanation :

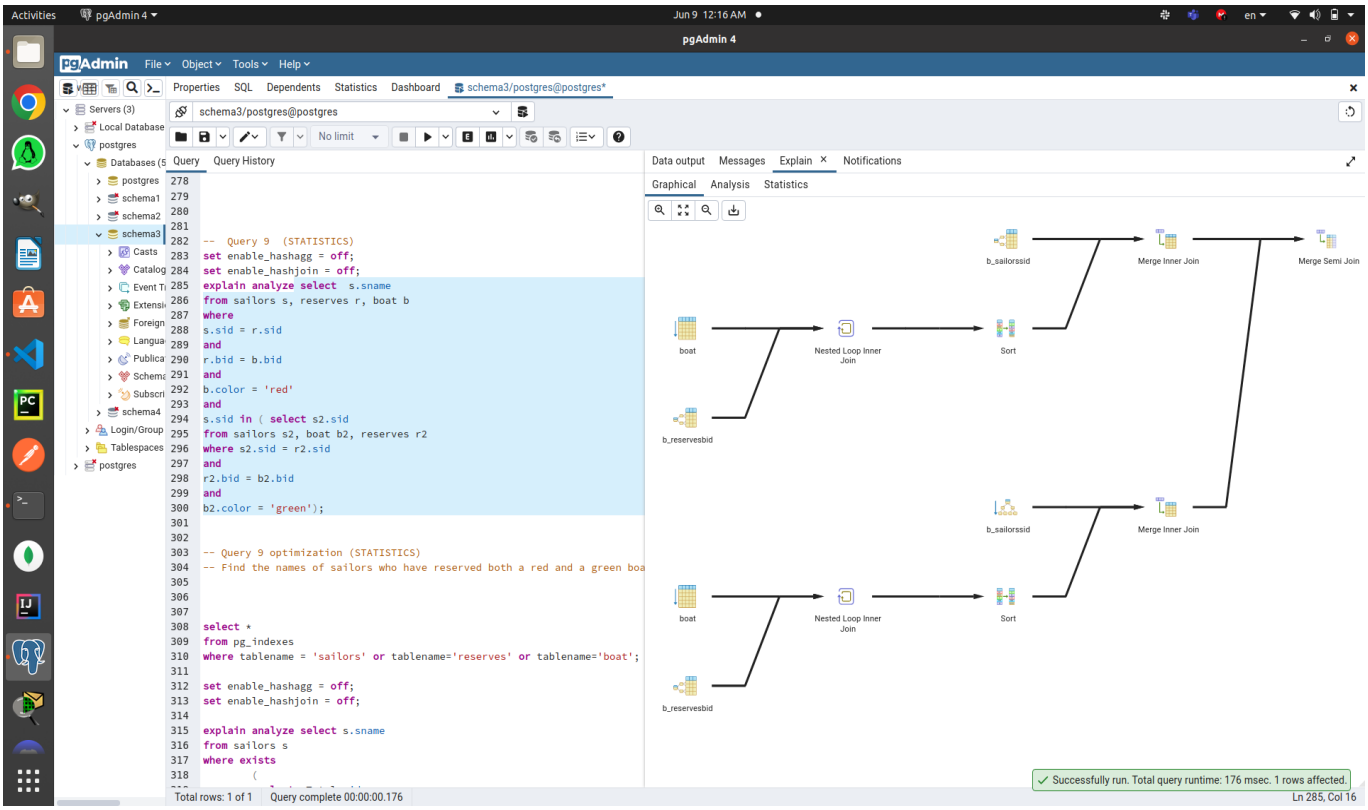
- Metrics :

Execution Time : 64.569 ms Total Expected Cost : 11368.59

2. given query with B+ trees indices only :


pgAdmin 4
Properties SQL Dependents Statistics Dashboard schema3/postgres@postgres*
schema3/postgres@postgres
Query Query History
Execute/Refresh
F5
b2.color = 'green');
-- Query 9 optimization (STATISTICS)
-- Find the names of sailors who have reserved both a red and a green boat.
select *
from pg_indexes
where tablename = 'sailors' or tablename='reserves' or tablename='boat';
set enable_hashagg = off;
set enable_hashjoin = off;
explain analyze select s.sname
from sailors s
where exists
Data output Messages Explain Notifications
schemaname tablename indexname tablespace indexdef
1 public sailors b_sailorssid [null] CREATE INDEX b_sailorssid ON public.sailors USING btree (sid)
2 public reserves b_reserves... [null] CREATE INDEX b_reservessid ON public.reserves USING btree (sid)
3 public reserves b_reserves... [null] CREATE INDEX b_reservesbid ON public.reserves USING btree (bid)
4 public boat b_boat [null] CREATE INDEX b_boat ON public.boat USING btree (bid, color)
Total rows: 4 of 4 Query complete 00:00:00.224
Ln 308, Col 1


pgAdmin 4
Properties SQL Dependents Statistics Dashboard schema3/postgres@postgres*
schema3/postgres@postgres
Query Query History
Execute/Refresh
F5
-- Query 9 (STATISTICS)
set enable_hashagg = off;
set enable_hashjoin = off;
explain analyze select s.sname
from sailors s, reserves r, boat b
where
s.sid = r.sid
and
r.bid = b.bid
and
b.color = 'red'
and
s.sid in (select s2.sid
from sailors s2, boat b2, reserves r2
where s2.sid = r2.sid
and
r2.bid = b2.bid
and
b2.color = 'green');
-- Query 9 optimization (STATISTICS)
-- Find the names of sailors who have reserved both a red and a green boa
select *
from pg_indexes
where tablename = 'sailors' or tablename='reserves' or tablename='boat';
set enable_hashagg = off;
set enable_hashjoin = off;
explain analyze select s.sname
from sailors s
where exists
Data output Messages Explain Notifications
QUERY PLAN
1 Merge Semi Join (cost=3948.54..5557.12 rows=1309 width=21) (actual time=4.931..7.873 rows=177 loops=1)
2 Merge Cond: (s.sid = s2.sid)
3 -> Merge Join (cost=1973.36..2758.56 rows=4982 width=29) (actual time=1.803..3.041 rows=1136 loops=1)
4 Merge Cond: (s.sid = r.sid)
5 -> Index Scan using b_sailorssid on sailors s (cost=0.29..663.29 rows=19000 width=25) (actual time=0.008..0.613 rows=2999 loops=1)
6 -> Sort (cost=1973.08..1985.53 rows=4982 width=4) (actual time=1.790..1.868 rows=1136 loops=1)
7 Sort Key: r.sid
8 Sort Method: quicksort Memory: 102kB
9 -> Nested Loop (cost=0.29..1667.12 rows=4982 width=4) (actual time=0.016..1.525 rows=1136 loops=1)
10 -> Seq Scan on boat b (cost=0.00..62.50 rows=427 width=4) (actual time=0.010..0.441 rows=427 loops=1)
11 Filter: (color = 'red'::bpchar)
12 Rows Removed by Filter: 2573
13 -> Index Scan using b_reservesbid on reserves r (cost=0.29..3.48 rows=28 width=8) (actual time=0.001..0.002 rows=3 loops=427)
14 Index Cond: (bid = b.bid)
15 -> Merge Join (cost=1975.18..2760.54 rows=4993 width=8) (actual time=2.933..4.596 rows=1861 loops=1)
16 Merge Cond: (s2.sid = r2.sid)
17 -> Index Only Scan using b_sailorssid on sailors s2 (cost=0.29..663.29 rows=19000 width=4) (actual time=0.016..0.888 rows=2999 loops=1)
18 Heap Fetches: 2999
19 -> Sort (cost=1974.89..1987.37 rows=4993 width=4) (actual time=2.852..2.993 rows=1861 loops=1)
20 Sort Key: r2.sid
21 Sort Method: quicksort Memory: 193kB
22 -> Nested Loop (cost=0.29..1668.18 rows=4993 width=4) (actual time=0.064..2.301 rows=2064 loops=1)
23 -> Seq Scan on boat b2 (cost=0.00..62.50 rows=428 width=4) (actual time=0.058..0.418 rows=428 loops=1)
24 Filter: (color = 'green'::bpchar)
25 Rows Removed by Filter: 2572
26 -> Index Scan using b_reservesbid on reserves r2 (cost=0.29..3.47 rows=28 width=8) (actual time=0.002..0.004 rows=5 loops=428)
27 Index Cond: (bid = b2.bid)
28 Planning Time: 1.220 ms
29 Execution Time: 7.925 ms
Total rows: 29 of 29 Query complete 00:00:00.153
Ln 300, Col 21



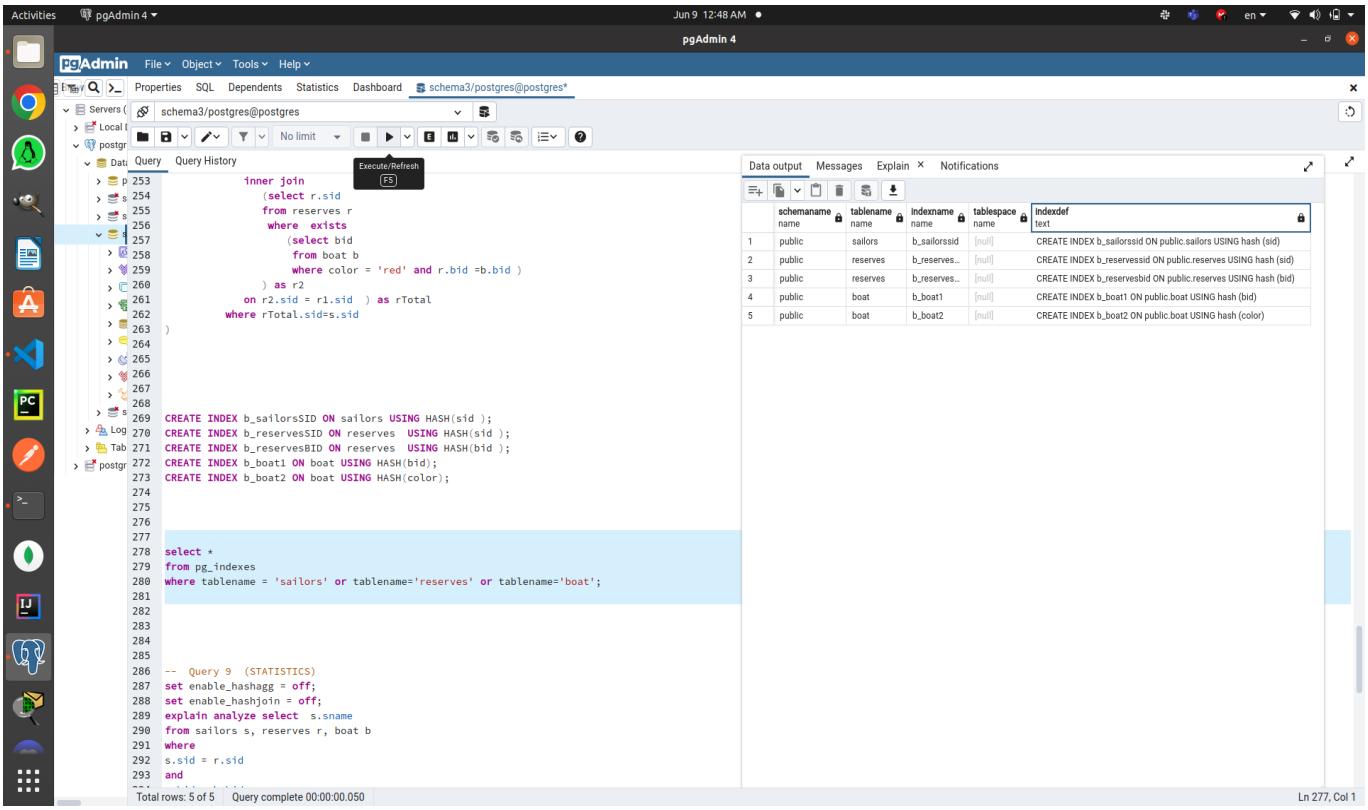
Explanation :

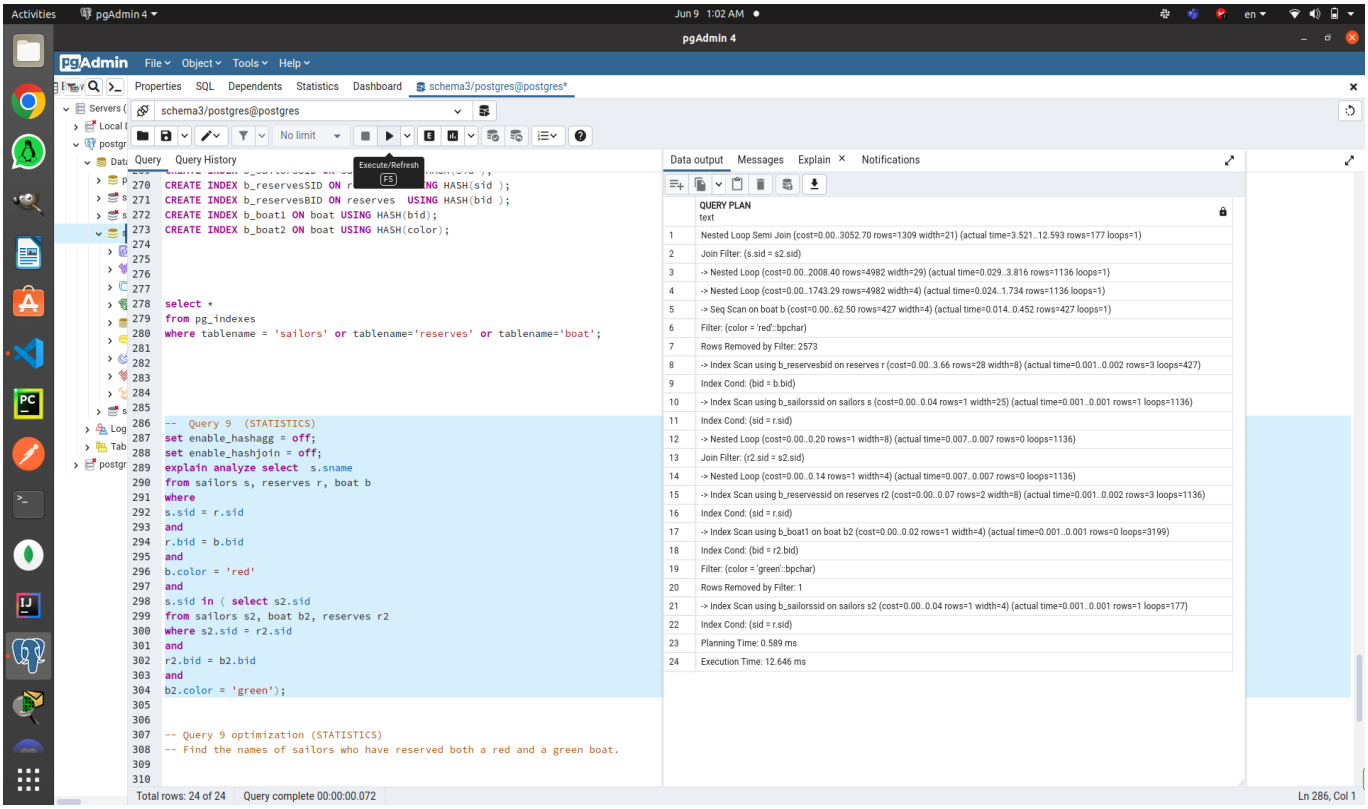
- Metrics :

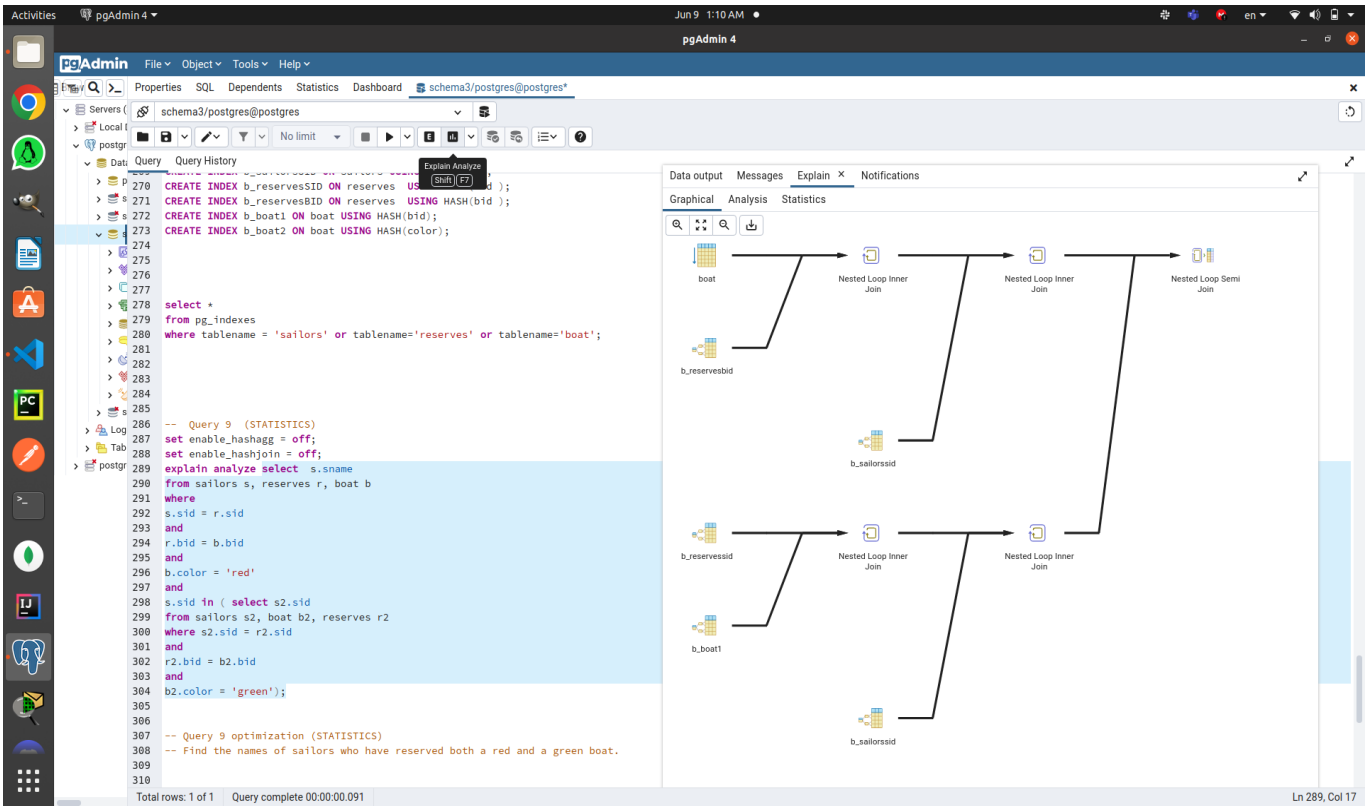
  Execution Time : 7.925 ms Total Expected Cost : 5557.12

- The B-tree helped in the performance it decreased the Execution Time and Expected Cost .
- The Query Planner used the B-tree index because B-Tree is O(Log n) performance with Exact Values
- Here it showed the improvement due to for every condition of Joining in the Query the Merge Join used index scan using ZigZag algorithm .

3. given query with hash indices only :


pgAdmin 4
Properties SQL Dependents Statistics Dashboard schema3/postgres@postgres*
schema3/postgres@postgres
Query Query History
Execute/Refresh
F5
Data output Messages Explain Notifications
schemaname name
tablename name
indexname name
tablespace name
indexdef text
1 public sailors b_sailorssid [null] CREATE INDEX b_sailorssid ON public.sailors USING hash (sid)
2 public reserves b_reserves... [null] CREATE INDEX b_reservessid ON public.reserves USING hash (sid)
3 public reserves b_reserves... [null] CREATE INDEX b_reservesbid ON public.reserves USING hash (bid)
4 public boat b_boat1 [null] CREATE INDEX b_boat1 ON public.boat USING hash (bid)
5 public boat b_boat2 [null] CREATE INDEX b_boat2 ON public.boat USING hash (color)
Total rows: 5 of 5
Query complete 00:00:00.050
Ln 277, Col 1


pgAdmin 4
Properties SQL Dependents Statistics Dashboard schema3/postgres@postgres*
schema3/postgres@postgres
Query Query History
Execute/Refresh
F5
Data output Messages Explain Notifications
QUERY PLAN text
1 Nested Loop Semi Join (cost=0.00..3052.70 rows=1309 width=21) (actual time=3.521..12.593 rows=177 loops=1)
2 Join Filter: (s.sid = s2.sid)
3 -> Nested Loop (cost=0.00..2008.40 rows=4982 width=29) (actual time=0.029..3.816 rows=1136 loops=1)
4 -> Nested Loop (cost=0.00..1743.29 rows=4982 width=4) (actual time=0.024..1.734 rows=1136 loops=1)
5 -> Seq Scan on boat b (cost=0.00..62.50 rows=427 width=4) (actual time=0.014..0.452 rows=427 loops=1)
6 Filter: (color = 'red'::bpchar)
7 Rows Removed by Filter: 2573
8 -> Index Scan using b_reservesbid on reserves r (cost=0.00..3.66 rows=28 width=8) (actual time=0.001..0.002 rows=3 loops=427)
9 Index Cond: (bid = b.bid)
10 -> Index Scan using b_sailorssid on sailors s (cost=0.00..0.04 rows=1 width=25) (actual time=0.001..0.001 rows=1 loops=1136)
11 Index Cond: (sid = r.sid)
12 -> Nested Loop (cost=0.00..0.20 rows=1 width=8) (actual time=0.007..0.007 rows=0 loops=1136)
13 Join Filter: (r2.sid = s2.sid)
14 -> Nested Loop (cost=0.00..0.14 rows=1 width=4) (actual time=0.007..0.007 rows=0 loops=1136)
15 -> Index Scan using b_reservesbid on reserves r2 (cost=0.00..0.07 rows=2 width=8) (actual time=0.001..0.002 rows=3 loops=1136)
16 Index Cond: (sid = r.sid)
17 -> Index Scan using b_boat1 on boat b2 (cost=0.00..0.02 rows=1 width=4) (actual time=0.001..0.001 rows=0 loops=3199)
18 Index Cond: (bid = r2.bid)
19 Filter: (color = 'green'::bpchar)
20 Rows Removed by Filter: 1
21 -> Index Scan using b_sailorssid on sailors s2 (cost=0.00..0.04 rows=1 width=4) (actual time=0.001..0.001 rows=1 loops=177)
22 Index Cond: (sid = r.sid)
23 Planning Time: 0.589 ms
24 Execution Time: 12.646 ms
Total rows: 24 of 24
Query complete 00:00:00.072
Ln 286, Col 1



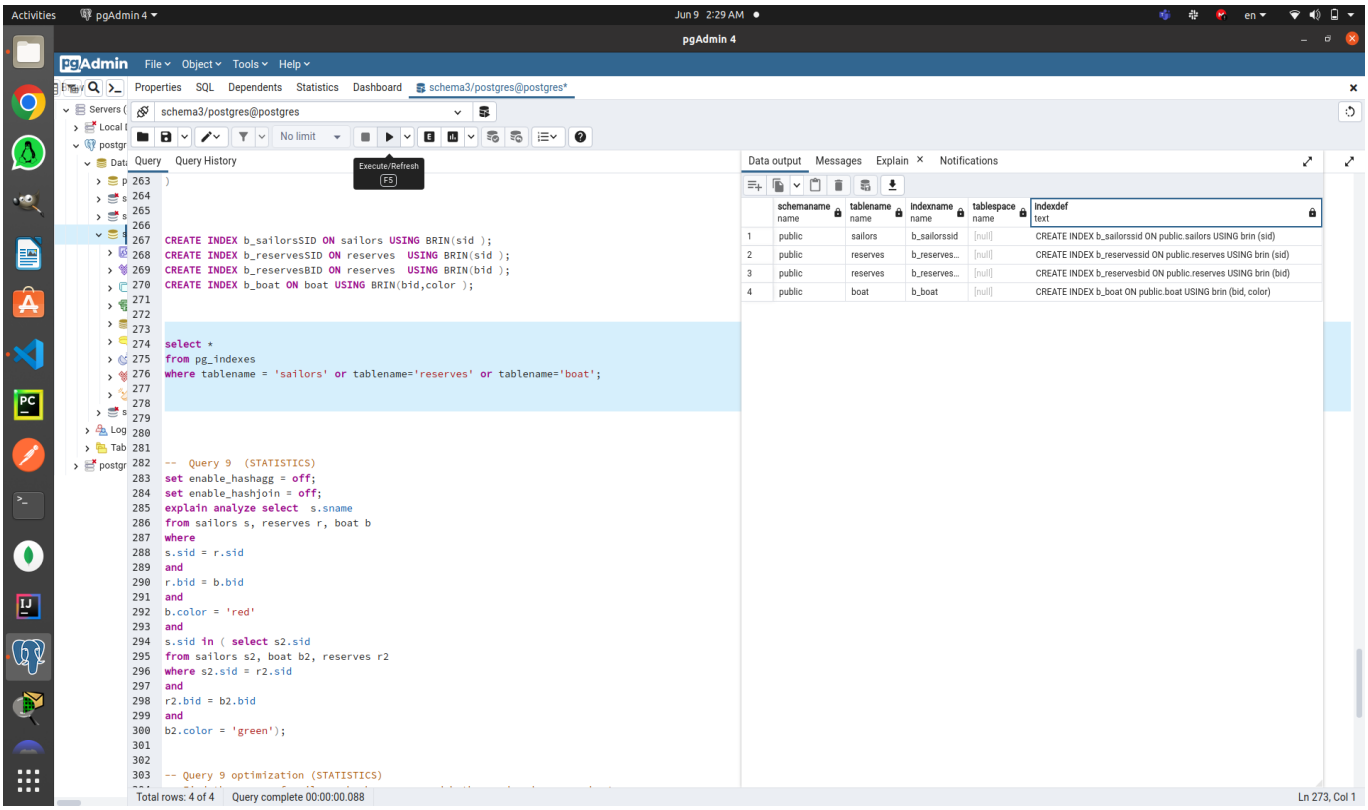
Explanation :

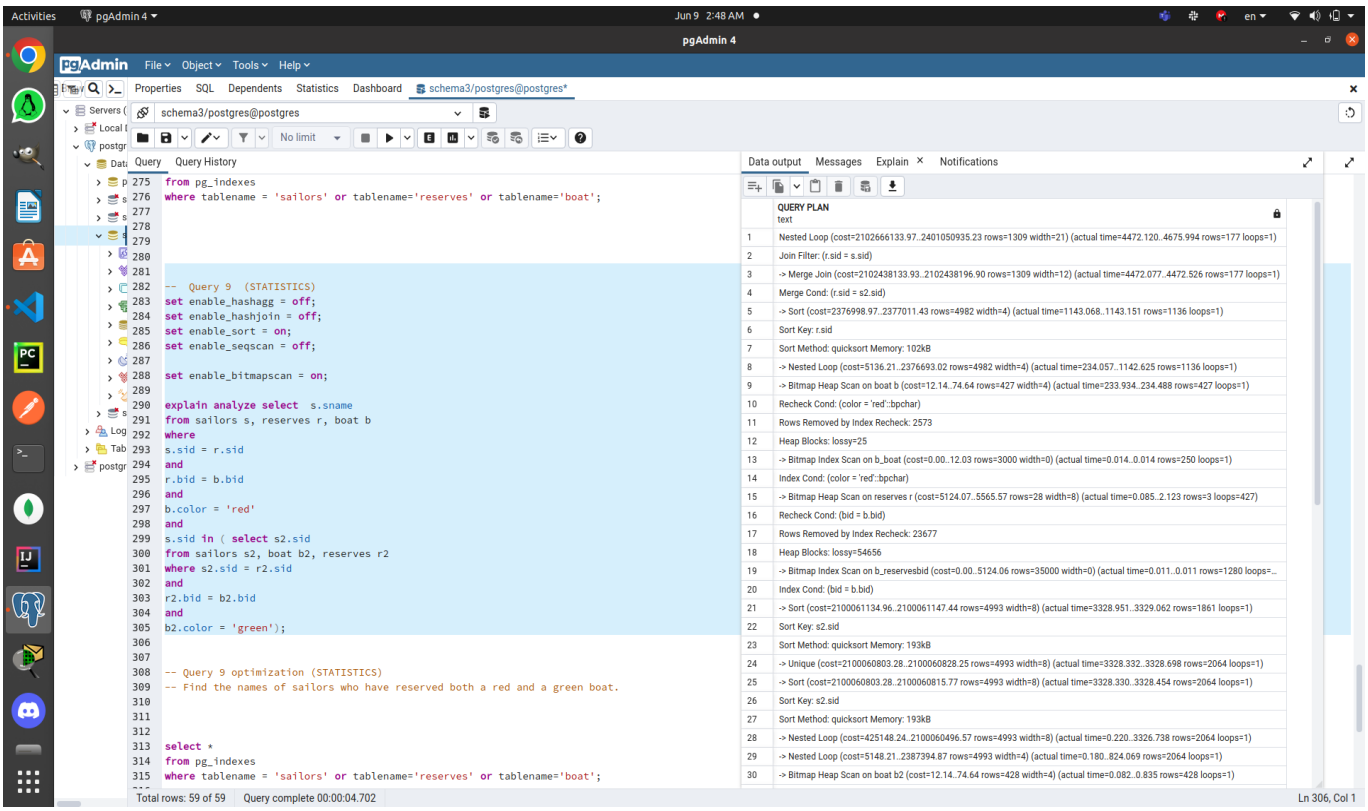
- Metrics :

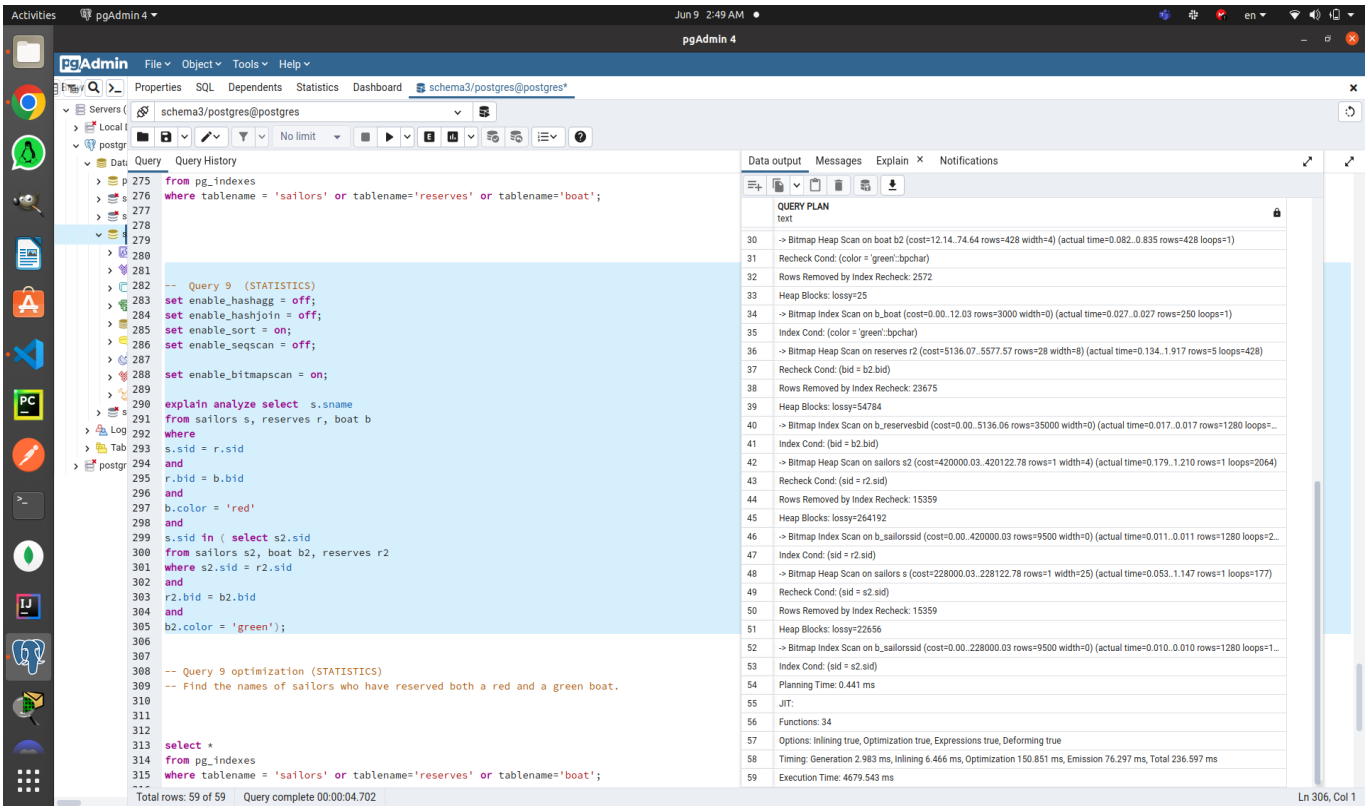
  Execution Time : 12.6 ms Total Expected Cost : 3052.12

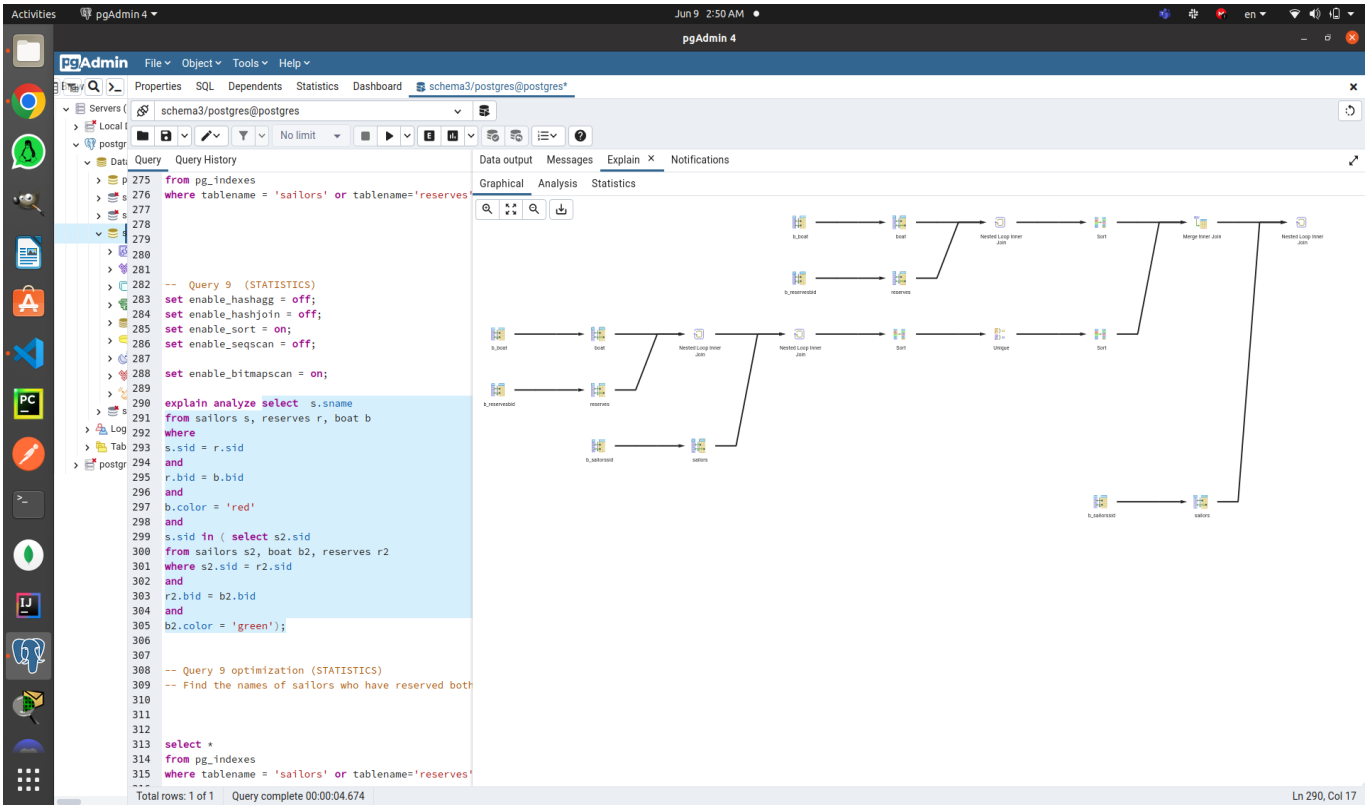
- The Hash helped in the performance it decreased the Execution Time(but in the execution time processor was overwhelmed) and Expected Cost .
- The Query Planner used the Hash index because Hash is O(1) performance with Exact Values and considered the best for exact values queries.
- Here it showed the improvement due to for every condition of Joining in the Query the Nested Loop Join using index scan using Hash based algorithm which approximatly maded to be O(n) performance.
- The Where clause condition on the color used Hash too.

4. given query with BRIN indices only :


Activities
pgAdmin 4
Jun 9 2:29 AM
pgAdmin 4
File Object Tools Help
Properties SQL Dependents Statistics Dashboard schema3/postgres@postgres*
schema3/postgres@postgres
No limit
Execute/Refresh
F5
Query Query History
263)
264
265
266
267 CREATE INDEX b_sailorsSID ON sailors USING BRIN(sid);
268 CREATE INDEX b_reservesSID ON reserves USING BRIN(sid);
269 CREATE INDEX b_reservesBID ON reserves USING BRIN(bid);
270 CREATE INDEX b_boat ON boat USING BRIN(bid,color);
271
272
273
274 select *
275 from pg_indexes
276 where tablename = 'sailors' or tablename='reserves' or tablename='boat';
277
278
279
280
281
282 -- Query 9 (STATISTICS)
283 set enable_hashagg = off;
284 set enable_hashjoin = off;
285 explain analyze select s.sname
286 from sailors s, reserves r, boat b
287 where
288 s.sid = r.sid
289 and
290 r.bid = b.bid
291 and
292 b.color = 'red'
293 and
294 s.sid in (select s2.sid
295 from sailors s2, boat b2, reserves r2
296 where s2.sid = r2.sid
297 and
298 r2.bid = b2.bid
299 and
300 b2.color = 'green');
301
302
303 -- Query 9 optimization (STATISTICS)
Data output Messages Explain Notifications
schemaname name
tablename name
indexname name
tablespace name
indexdef text
1 public sailors b_sailorssid [null] CREATE INDEX b_sailorssid ON public.sailors USING brin (sid)
2 public reserves b_reserves... [null] CREATE INDEX b_reservessid ON public.reserves USING brin (sid)
3 public reserves b_reserves... [null] CREATE INDEX b_reservesbid ON public.reserves USING brin (bid)
4 public boat b_boat [null] CREATE INDEX b_boat ON public.boat USING brin (bid, color)
Total rows: 4 of 4
Query complete 00:00:00.088
Ln 273, Col 1


Activities
pgAdmin 4
Jun 9 2:48 AM
pgAdmin 4
File Object Tools Help
Properties SQL Dependents Statistics Dashboard schema3/postgres@postgres*
schema3/postgres@postgres
No limit
Query Query History
275 from pg_indexes
276 where tablename = 'sailors' or tablename='reserves' or tablename='boat';
277
278
279
280
281
282 -- Query 9 (STATISTICS)
283 set enable_hashagg = off;
284 set enable_hashjoin = off;
285 set enable_sort = on;
286 set enable_seqscan = off;
287
288 set enable_bitmapscan = on;
289
290 explain analyze select s.sname
291 from sailors s, reserves r, boat b
292 where
293 s.sid = r.sid
294 and
295 r.bid = b.bid
296 and
297 b.color = 'red'
298 and
299 s.sid in (select s2.sid
300 from sailors s2, boat b2, reserves r2
301 where s2.sid = r2.sid
302 and
303 r2.bid = b2.bid
304 and
305 b2.color = 'green');
306
307
308 -- Query 9 optimization (STATISTICS)
309 -- Find the names of sailors who have reserved both a red and a green boat.
310
311
312
313 select *
314 from pg_indexes
315 where tablename = 'sailors' or tablename='reserves' or tablename='boat';
Data output Messages Explain Notifications
QUERY PLAN text
1 Nested Loop (cost=2102666133.97..2401050935.23 rows=1309 width=21) (actual time=4472.120..4675.994 rows=177 loops=1)
2 Join Filter: (r.sid = s.sid)
3 -> Merge Join (cost=2102438133.93..2102438196.90 rows=1309 width=12) (actual time=4472.077..4472.526 rows=177 loops=1)
4 Merge Cond: (r.sid = s2.sid)
5 -> Sort (cost=2376998.97..2377011.43 rows=4982 width=4) (actual time=1143.068..1143.151 rows=1136 loops=1)
6 Sort Key: r.sid
7 Sort Method: quicksort Memory: 102kB
8 -> Nested Loop (cost=5136.21..2376693.02 rows=4982 width=4) (actual time=234.057..1142.625 rows=1136 loops=1)
9 -> Bitmap Heap Scan on boat b (cost=12.14..74.64 rows=427 width=4) (actual time=233.934..234.488 rows=427 loops=1)
10 Recheck Cond: (color = 'red'::bpchar)
11 Rows Removed by Index Recheck: 2573
12 Heap Blocks: lossy=25
13 -> Bitmap Index Scan on b_boat (cost=0.00..12.03 rows=3000 width=0) (actual time=0.014..0.014 rows=250 loops=1)
14 Index Cond: (color = 'red'::bpchar)
15 -> Bitmap Heap Scan on reserves r (cost=5124.07..5565.57 rows=28 width=8) (actual time=0.085..2.123 rows=3 loops=427)
16 Recheck Cond: (bid = b.bid)
17 Rows Removed by Index Recheck: 23677
18 Heap Blocks: lossy=54656
19 -> Bitmap Index Scan on b_reservesbid (cost=0.00..5124.06 rows=35000 width=0) (actual time=0.011..0.011 rows=1280 loops=...
20 Index Cond: (bid = b.bid)
21 -> Sort (cost=2100061134.96..2100061147.44 rows=4993 width=8) (actual time=3328.951..3329.062 rows=1861 loops=1)
22 Sort Key: s2.sid
23 Sort Method: quicksort Memory: 193kB
24 -> Unique (cost=2100060803.28..2100060828.25 rows=4993 width=8) (actual time=3328.332..3328.698 rows=2064 loops=1)
25 -> Sort (cost=2100060803.28..2100060815.77 rows=4993 width=8) (actual time=3328.330..3328.454 rows=2064 loops=1)
26 Sort Key: s2.sid
27 Sort Method: quicksort Memory: 193kB
28 -> Nested Loop (cost=425148.24..2100060496.57 rows=4993 width=8) (actual time=0.220..3326.738 rows=2064 loops=1)
29 -> Nested Loop (cost=5148.21..2387394.87 rows=4993 width=4) (actual time=0.180..824.069 rows=2064 loops=1)
30 -> Bitmap Heap Scan on boat b2 (cost=12.14..74.64 rows=428 width=4) (actual time=0.082..0.835 rows=428 loops=1)
Total rows: 59 of 59
Query complete 00:00:04.702
Ln 306, Col 1





Explanation :

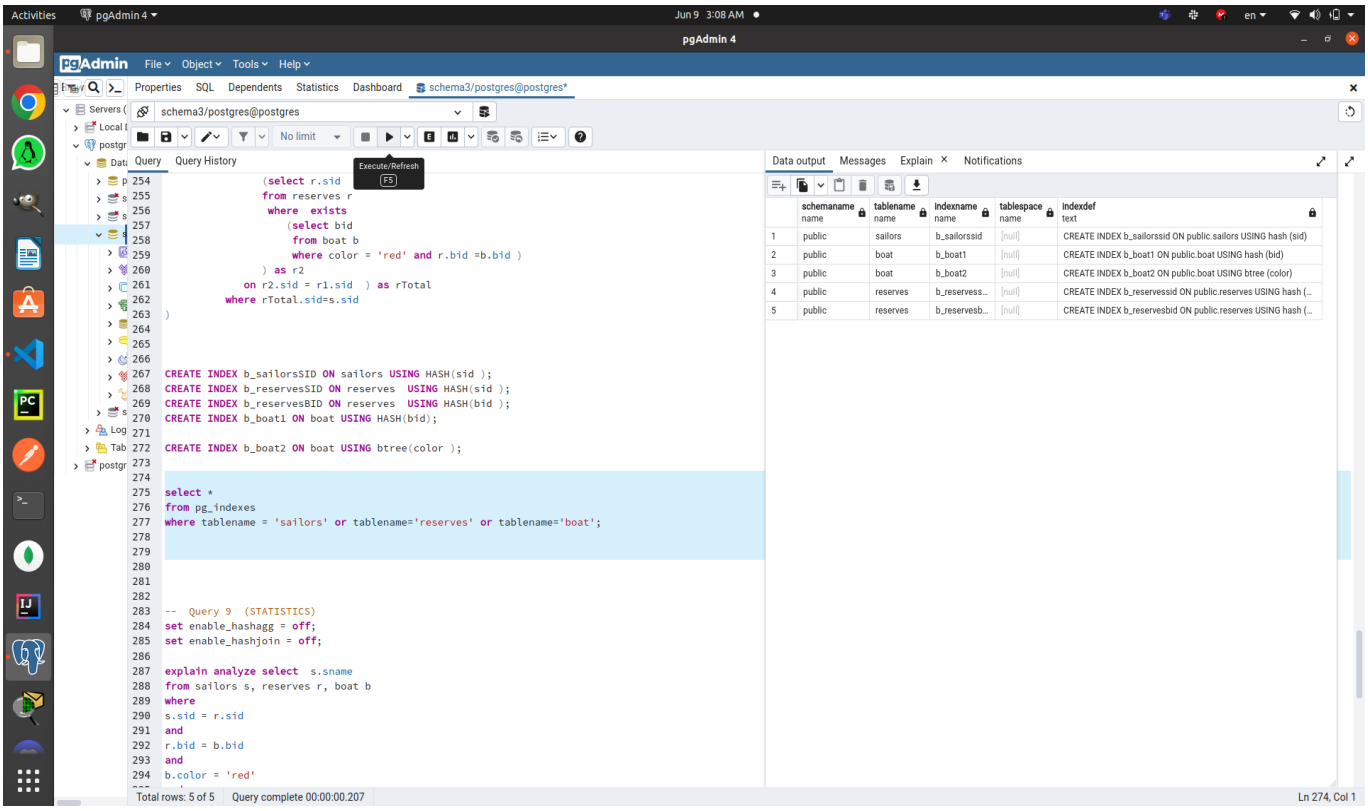
- Metrics :

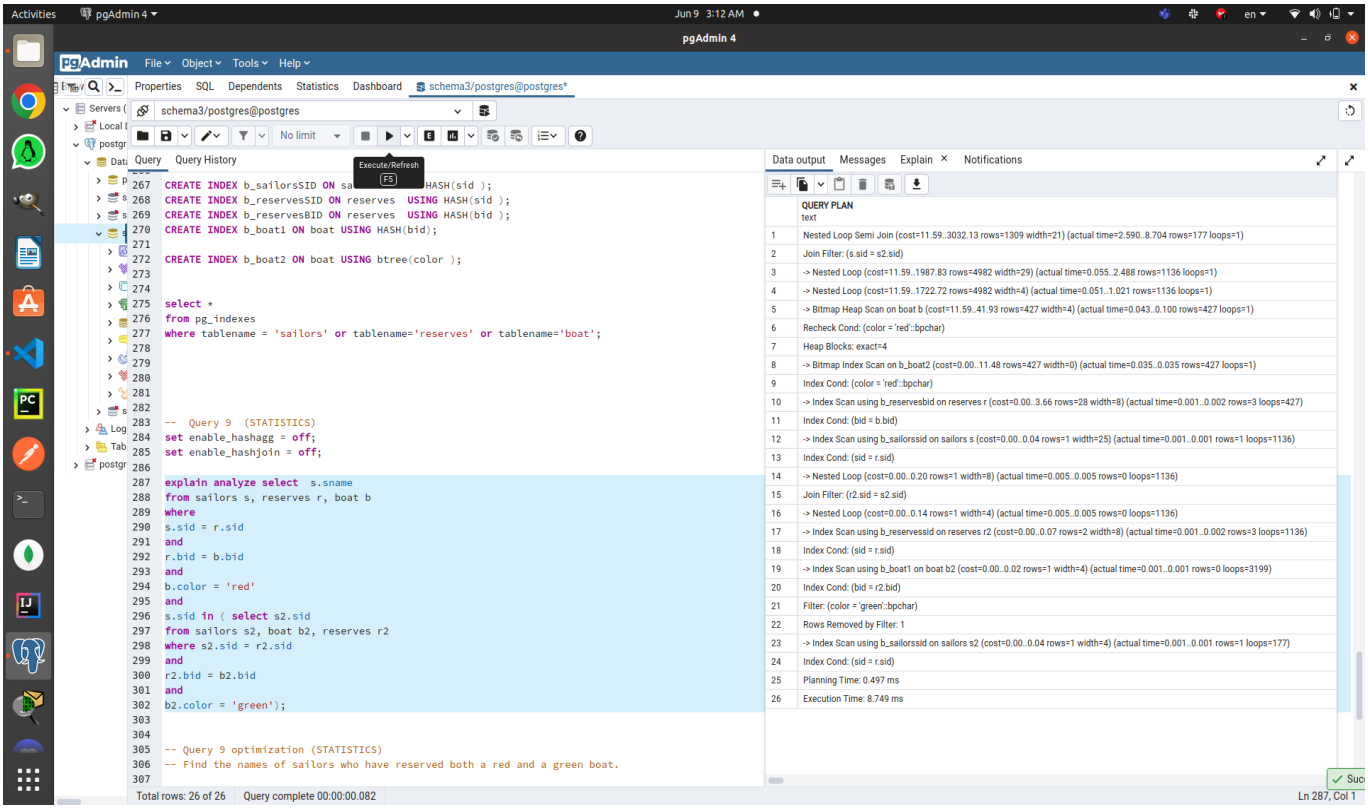
  Execution Time : 4679 ms Total Expected Cost : 2401050935.23

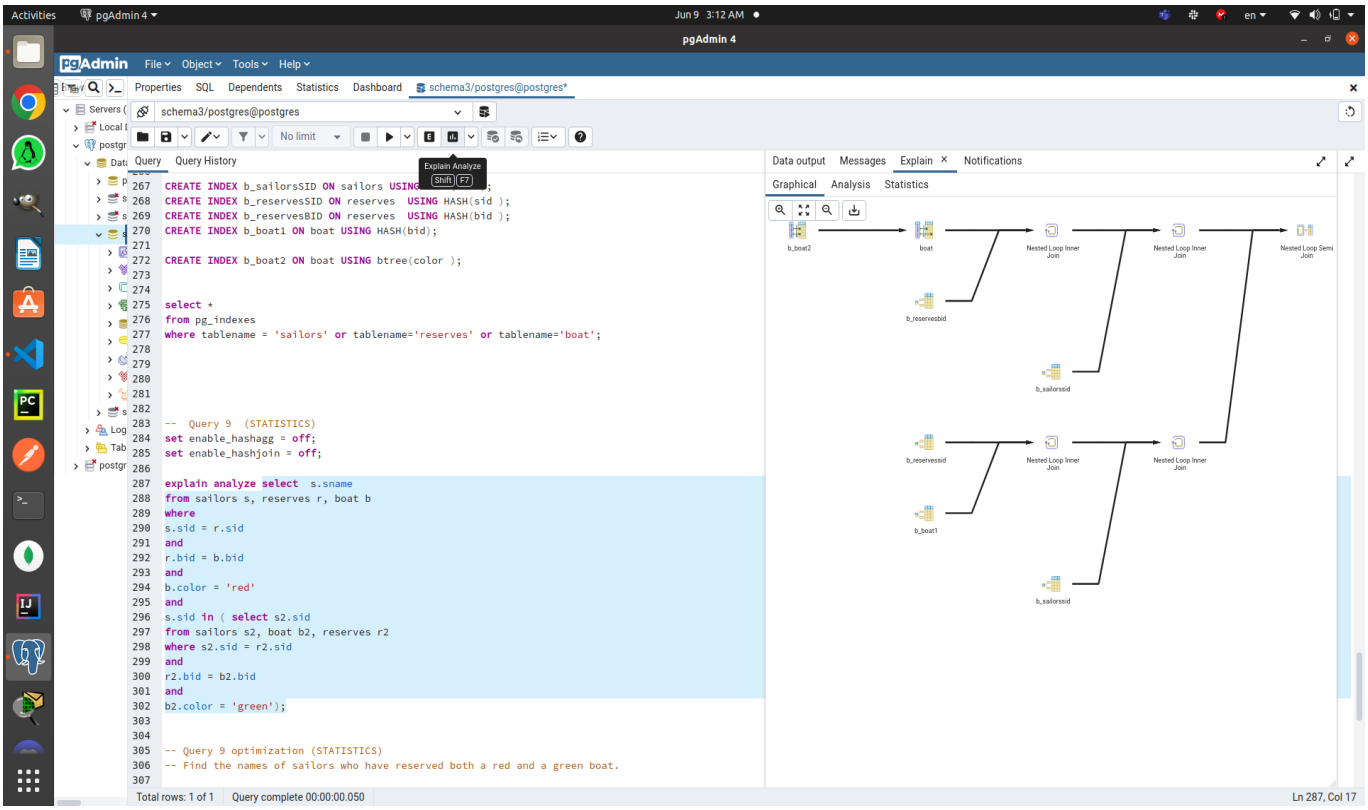
- Here the BRIN was not used in the original Query Plan settings (Hashjoin and HashAgg are off) so I've made seqscan=off too.

- The Execution Time and Expected Cost became the Worst of all .
- This happened because the Query Optimizer didnt used it from the place due to BRIN Usage here was not suitable so we have used it to simulate seqscan behaviour only we traveresed it all and followed all its pointers so it is worst index to use in this case.

5. given query with mixed indices (any mix of your choice) :







Explanation :

- Metrics :

  Execution Time : 4679 ms Total Expected Cost : 3032.13

- The Query Planner used the Hash index because Hash is O(1) performance with Exact Values and considered the best for exact values queries.
- Here it showed the improvement due to for every condition of Joining in the Query the Nested Loop Join using index scan using Hash based algorithm which approximatly maded to be O(n) performance.
- The Where clause condition on the color used B-tree which is more better as colors are repetable and sorted after each other at the leaves of the b-tree with O(Log n) performance on all at one time little bit better than Hash O(1) but calculation of hashfunction and stored in different buckets .

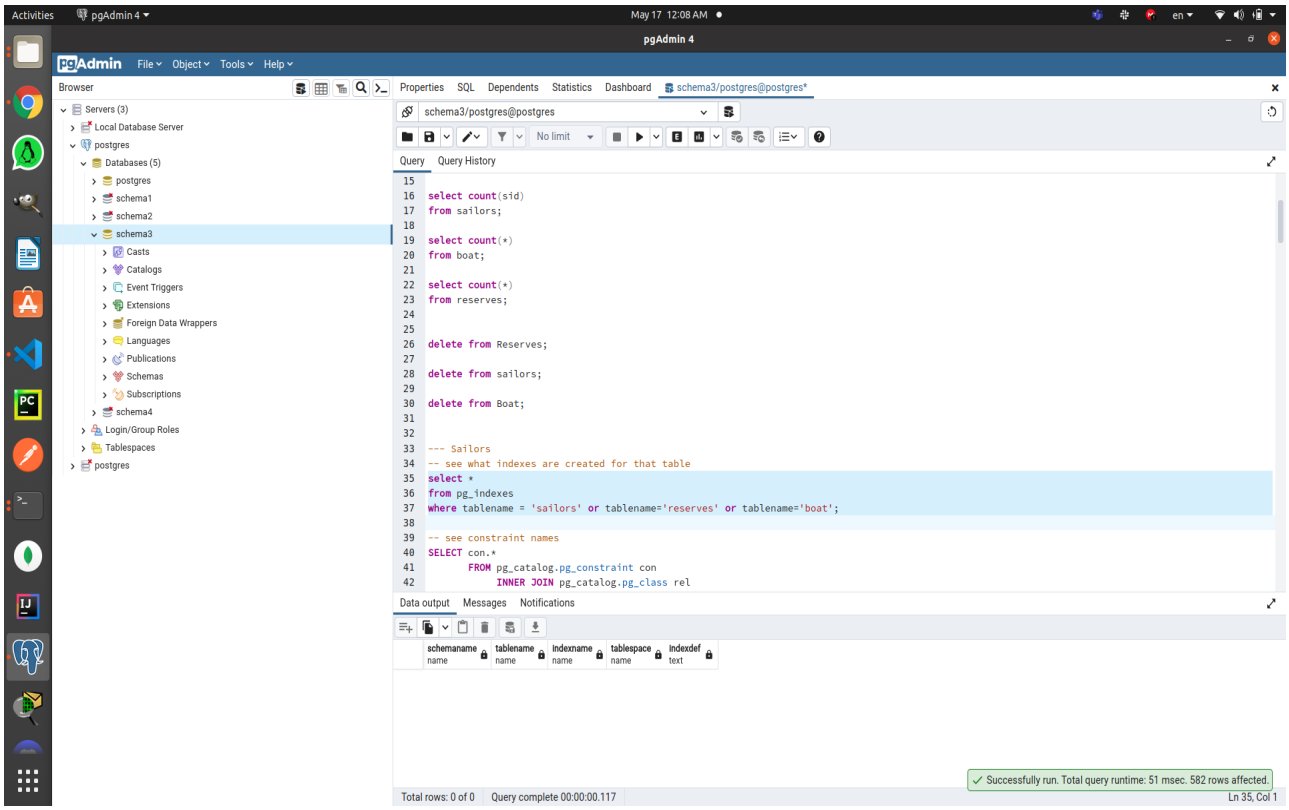
Optimized Query

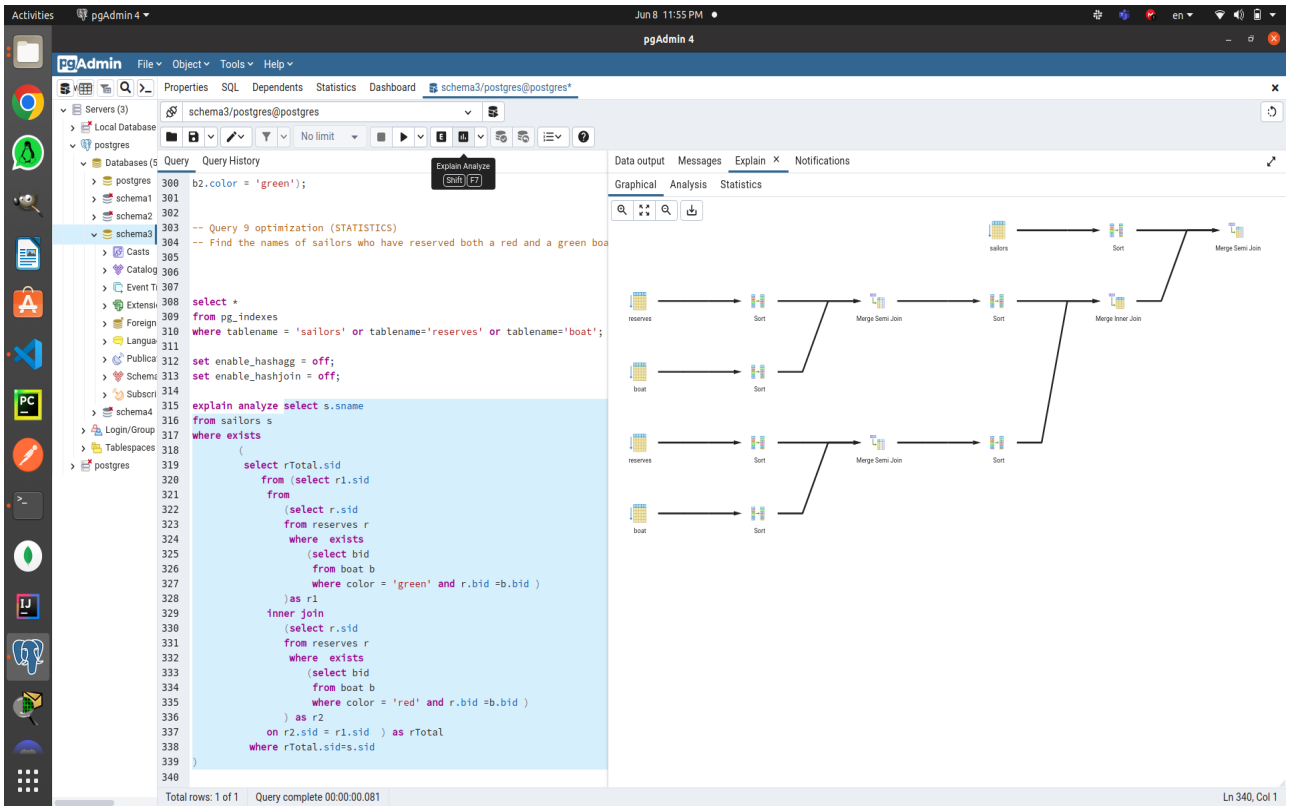
```
select s.sname
from sailors s
where exists
        (
         select rTotal.sid
            from (select r1.sid
             from
                (select r.sid
                from reserves r
                 where  exists
```

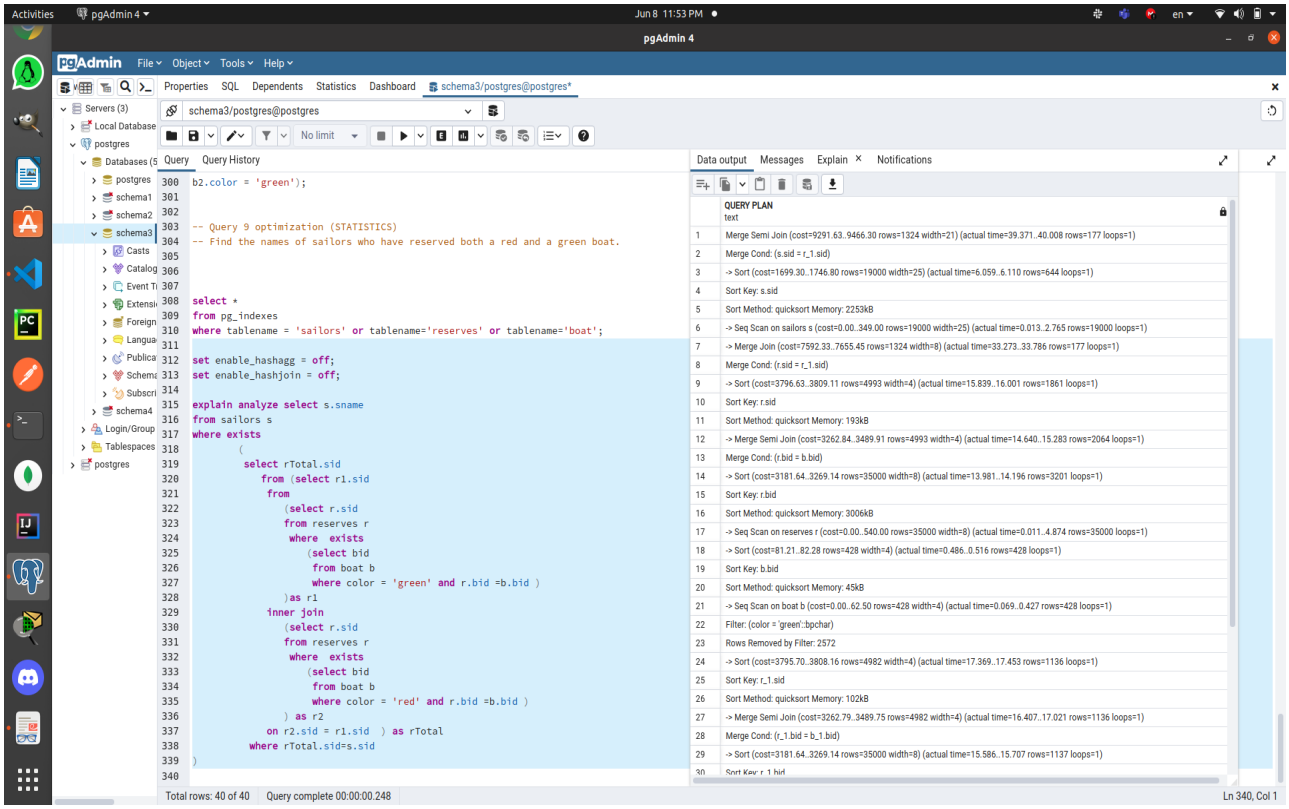
```
                    (select bid
                     from boat b
                     where color = 'green' and r.bid =b.bid )
                )as r1
            inner join
                (select r.sid
                from reserves r
                 where  exists
                    (select bid
                     from boat b
                     where color = 'red' and r.bid =b.bid )
                ) as r2
                on r2.sid = r1.sid  ) as rTotal
            where rTotal.sid=s.sid
)
```

Report

1. given query without an index,

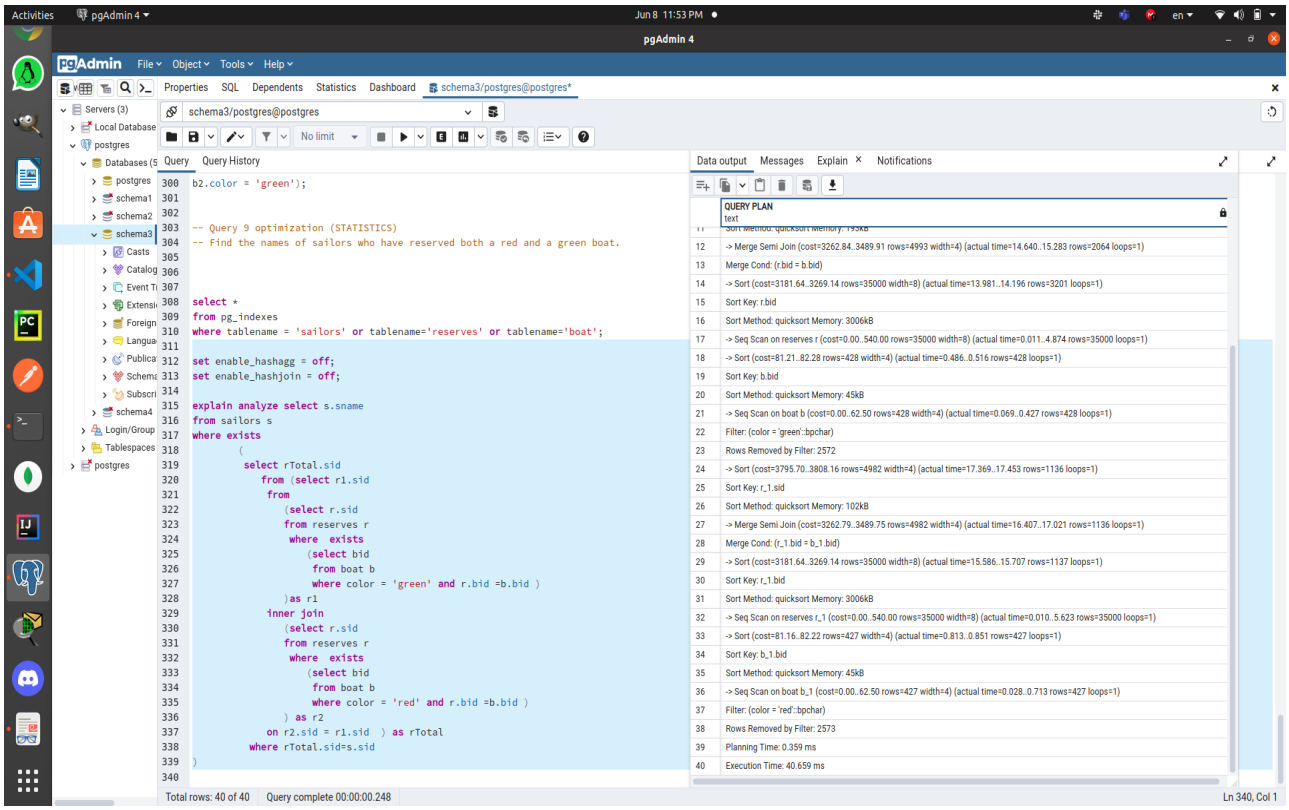







```
b2.color = 'green');


-- Query 9 optimization (STATISTICS)
-- Find the names of sailors who have reserved both a red and a green boat.


select *
from pg_indexes
where tablename = 'sailors' or tablename='reserves' or tablename='boat';

set enable_hashagg = off;
set enable_hashjoin = off;

explain analyze select s.sname
from sailors s
where exists
        (
        select rTotal.sid
           from (select r1.sid
            from
               (select r.sid
               from reserves r
                where  exists
                   (select bid
                    from boat b
                    where color = 'green' and r.bid =b.bid )
               )as r1
            inner join
               (select r.sid
               from reserves r
                where  exists
                   (select bid
                    from boat b
                    where color = 'red' and r.bid =b.bid )
               ) as r2
            on r2.sid = r1.sid  ) as rTotal
         where rTotal.sid=s.sid
)
```

	QUERY PLAN text
1	Merge Semi Join (cost=9291.63..9466.30 rows=1324 width=21) (actual time=39.371..40.008 rows=177 loops=1)
2	Merge Cond: (s.sid = r_1.sid)
3	-> Sort (cost=1699.30..1746.80 rows=19000 width=25) (actual time=6.059..6.110 rows=644 loops=1)
4	Sort Key: s.sid
5	Sort Method: quicksort Memory: 2253kB
6	-> Seq Scan on sailors s (cost=0.00..349.00 rows=19000 width=25) (actual time=0.013..2.765 rows=19000 loops=1)
7	-> Merge Join (cost=7592.33..7655.45 rows=1324 width=8) (actual time=33.273..33.786 rows=177 loops=1)
8	Merge Cond: (r.sid = r_1.sid)
9	-> Sort (cost=3796.63..3809.11 rows=4993 width=4) (actual time=15.839..16.001 rows=1861 loops=1)
10	Sort Key: r.sid
11	Sort Method: quicksort Memory: 193kB
12	-> Merge Semi Join (cost=3262.84..3489.91 rows=4993 width=4) (actual time=14.640..15.283 rows=2064 loops=1)
13	Merge Cond: (r.bid = b.bid)
14	-> Sort (cost=3181.64..3269.14 rows=35000 width=8) (actual time=13.981..14.196 rows=3201 loops=1)
15	Sort Key: r.bid
16	Sort Method: quicksort Memory: 3006kB
17	-> Seq Scan on reserves r (cost=0.00..540.00 rows=35000 width=8) (actual time=0.011..4.874 rows=35000 loops=1)
18	-> Sort (cost=81.21..82.28 rows=428 width=4) (actual time=0.486..0.516 rows=428 loops=1)
19	Sort Key: b.bid
20	Sort Method: quicksort Memory: 45kB
21	-> Seq Scan on boat b (cost=0.00..62.50 rows=428 width=4) (actual time=0.069..0.427 rows=428 loops=1)
22	Filter: (color = 'green'::bpchar)
23	Rows Removed by Filter: 2572
24	-> Sort (cost=3795.70..3808.16 rows=4982 width=4) (actual time=17.369..17.453 rows=1136 loops=1)
25	Sort Key: r_1.sid
26	Sort Method: quicksort Memory: 102kB
27	-> Merge Semi Join (cost=3262.79..3489.75 rows=4982 width=4) (actual time=16.407..17.021 rows=1136 loops=1)
28	Merge Cond: (r_1.bid = b_1.bid)
29	-> Sort (cost=3181.64..3269.14 rows=35000 width=8) (actual time=15.586..15.707 rows=1137 loops=1)
30	Sort Key: r_1.bid



Explanation :

- Metrics :

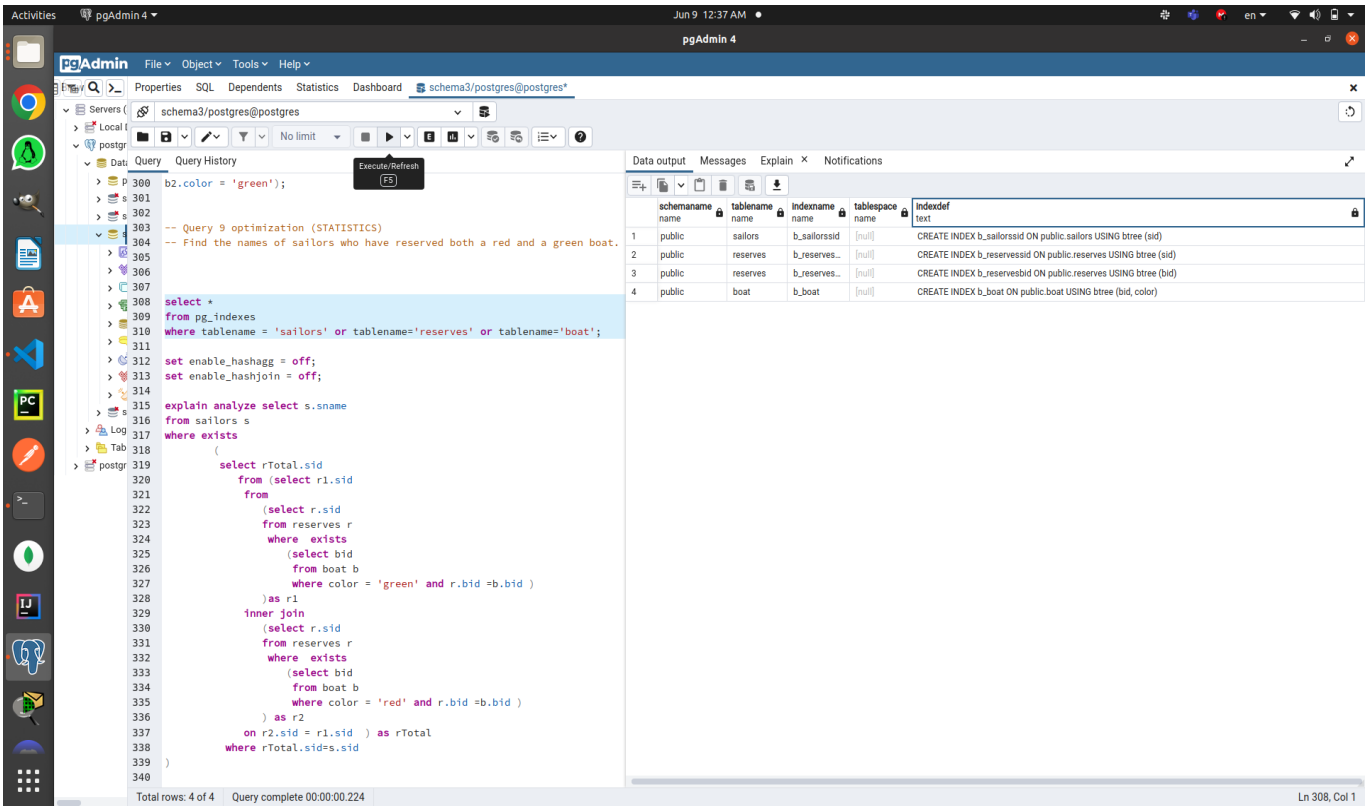
  Execution Time : 40.659 ms Total Expected Cost : 9466.30

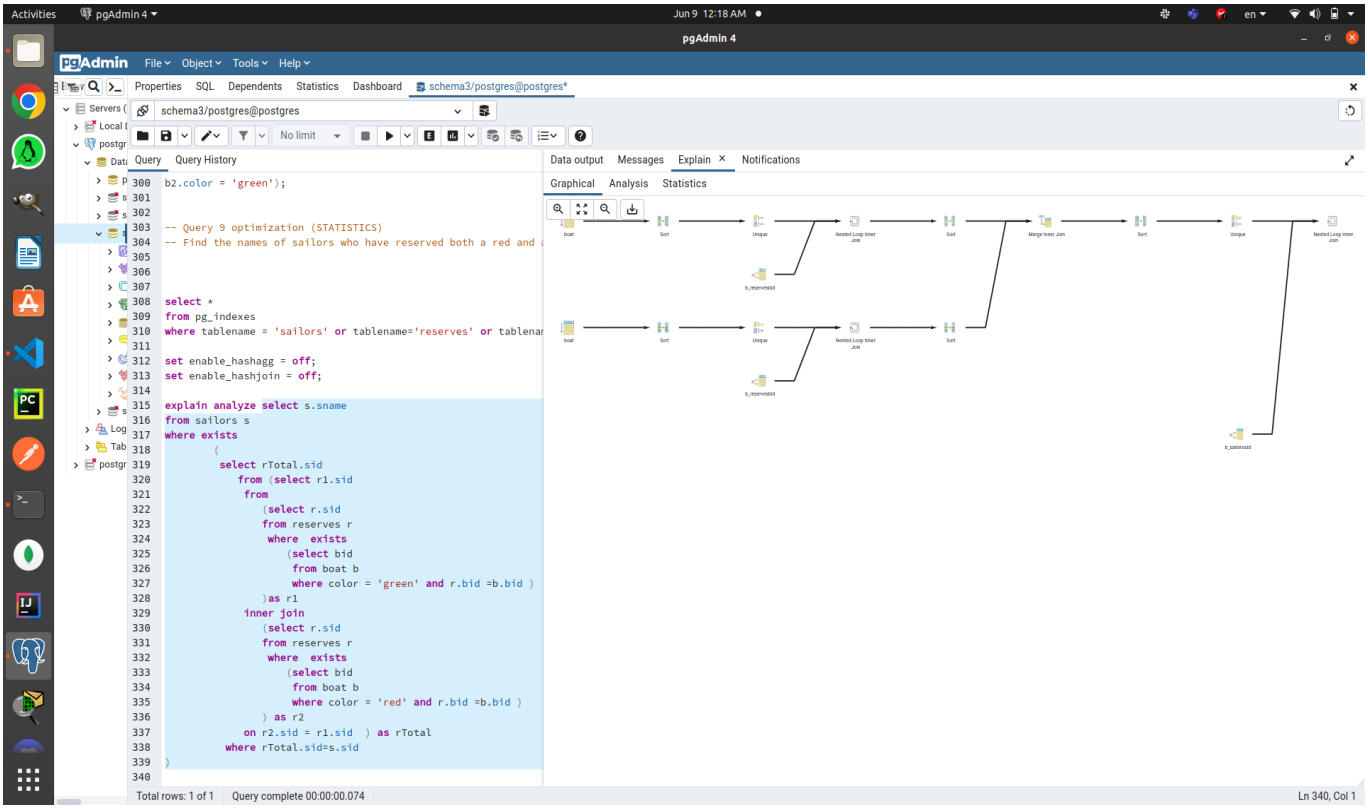
- Same flags is set to all here too.

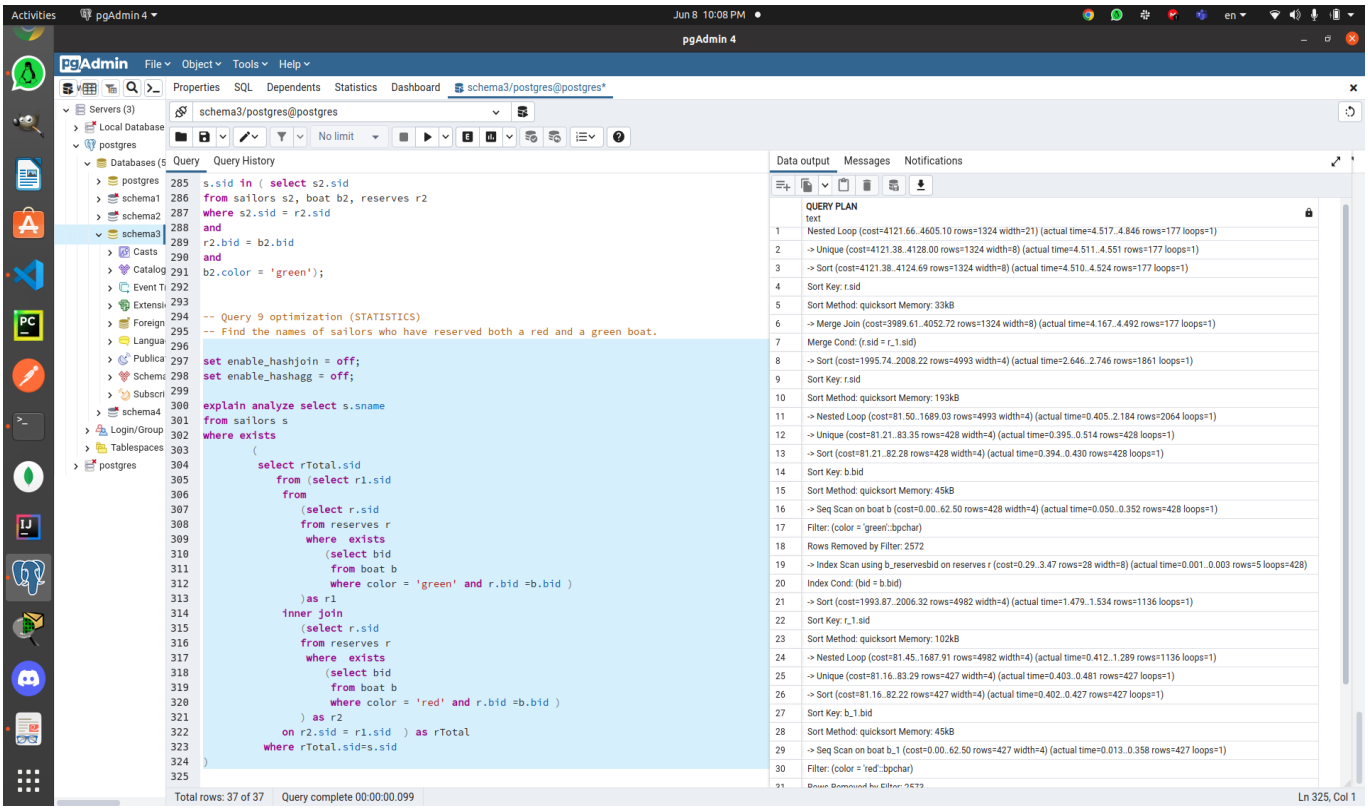
- Reason :

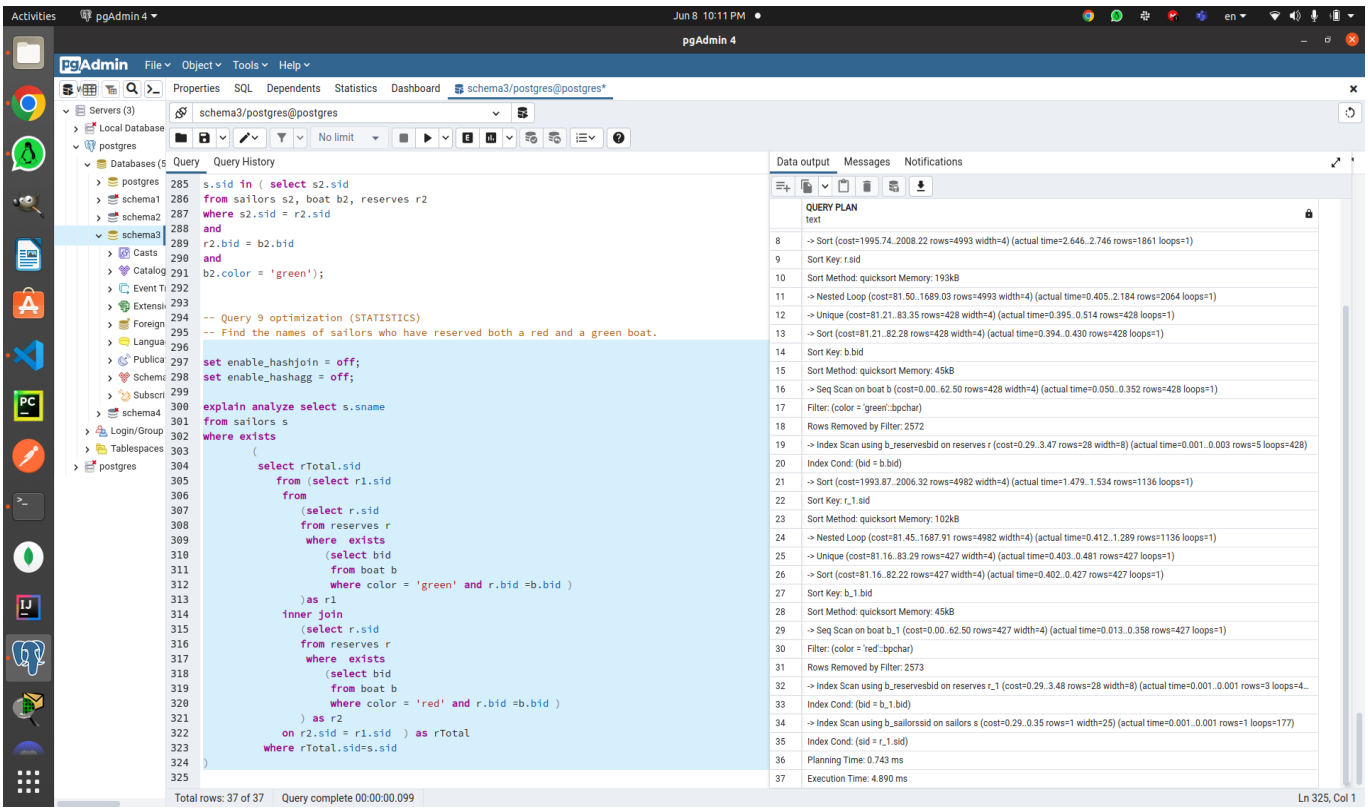
  - This Query Improved in the Execution time and Expected Cost than the Original Query from 11368.59 to 9466.30.
  - Because the loops ends faster and exits due to I used the Exist Operator instead of the In Operator so it helped in the intermediate results.
  - I have used the table Sailors once instead of twiceas used in original Query .
  - 6 less steps of the Query Plan where made in the Optimized one which helped in decreasing the cost.

2. given query with B+ trees indices only,


Activities
pgAdmin 4
Jun 9 12:37 AM
pgAdmin 4
File Object Tools Help
Properties SQL Dependents Statistics Dashboard schema3/postgres@postgres*
schema3/postgres@postgres
No limit
Execute/Refresh
F5
Query Query History
Data output Messages Explain Notifications
b2.color = 'green');
-- Query 9 optimization (STATISTICS)
-- Find the names of sailors who have reserved both a red and a green boat.
select *
from pg_indexes
where tablename = 'sailors' or tablename='reserves' or tablename='boat';
set enable_hashagg = off;
set enable_hashjoin = off;
explain analyze select s.sname
from sailors s
where exists
(select rTotal.sid
from (select r1.sid
from
(select r.sid
from reserves r
where exists
(select bid
from boat b
where color = 'green' and r.bid =b.bid)
)as r1
inner join
(select r.sid
from reserves r
where exists
(select bid
from boat b
where color = 'red' and r.bid =b.bid)
) as r2
on r2.sid = r1.sid) as rTotal
where rTotal.sid=s.sid
)
schemaname name
tablename name
indexname name
tablespace name
indexdef text
1 public sailors b_sailorssid [null] CREATE INDEX b_sailorssid ON public.sailors USING btree (sid)
2 public reserves b_reserves... [null] CREATE INDEX b_reservessid ON public.reserves USING btree (sid)
3 public reserves b_reserves... [null] CREATE INDEX b_reservesbid ON public.reserves USING btree (bid)
4 public boat b_boat [null] CREATE INDEX b_boat ON public.boat USING btree (bid, color)
Total rows: 4 of 4
Query complete 00:00:00.224
Ln 308, Col 1


Activities
pgAdmin 4
Jun 9 12:18 AM
pgAdmin 4
File Object Tools Help
Properties SQL Dependents Statistics Dashboard schema3/postgres@postgres*
schema3/postgres@postgres
No limit
Query Query History
Data output Messages Explain Notifications
Graphical Analysis Statistics
b2.color = 'green');
-- Query 9 optimization (STATISTICS)
-- Find the names of sailors who have reserved both a red and
select *
from pg_indexes
where tablename = 'sailors' or tablename='reserves' or tablenar
set enable_hashagg = off;
set enable_hashjoin = off;
explain analyze select s.sname
from sailors s
where exists
(select rTotal.sid
from (select r1.sid
from
(select r.sid
from reserves r
where exists
(select bid
from boat b
where color = 'green' and r.bid =b.bid)
)as r1
inner join
(select r.sid
from reserves r
where exists
(select bid
from boat b
where color = 'red' and r.bid =b.bid)
) as r2
on r2.sid = r1.sid) as rTotal
where rTotal.sid=s.sid
)
Total rows: 1 of 1
Query complete 00:00:00.074
Ln 340, Col 1






```
s.sid in ( select s2.sid
from sailors s2, boat b2, reserves r2
where s2.sid = r2.sid
and
r2.bid = b2.bid
and
b2.color = 'green');


-- Query 9 optimization (STATISTICS)
-- Find the names of sailors who have reserved both a red and a green boat.

set enable_hashjoin = off;
set enable_hashagg = off;

explain analyze select s.sname
from sailors s
where exists
        (
        select rTotal.sid
           from (select r1.sid
            from
               (select r.sid
               from reserves r
                where  exists
                   (select bid
                    from boat b
                    where color = 'green' and r.bid =b.bid )
               )as r1
            inner join
               (select r.sid
               from reserves r
                where  exists
                   (select bid
                    from boat b
                    where color = 'red' and r.bid =b.bid )
               ) as r2
            on r2.sid = r1.sid  ) as rTotal
         where rTotal.sid=s.sid
)
```

```
QUERY PLAN
text
Nested Loop (cost=4121.66..4605.10 rows=1324 width=21) (actual time=4.517..4.846 rows=177 loops=1)
-> Unique (cost=4121.38..4128.00 rows=1324 width=8) (actual time=4.511..4.551 rows=177 loops=1)
-> Sort (cost=4121.38..4124.69 rows=1324 width=8) (actual time=4.510..4.524 rows=177 loops=1)
Sort Key: r.sid
Sort Method: quicksort Memory: 33kB
-> Merge Join (cost=3989.61..4052.72 rows=1324 width=8) (actual time=4.167..4.492 rows=177 loops=1)
Merge Cond: (r.sid = r_1.sid)
-> Sort (cost=1995.74..2008.22 rows=4993 width=4) (actual time=2.646..2.746 rows=1861 loops=1)
Sort Key: r.sid
Sort Method: quicksort Memory: 193kB
-> Nested Loop (cost=81.50..1689.03 rows=4993 width=4) (actual time=0.405..2.184 rows=2064 loops=1)
-> Unique (cost=81.21..83.35 rows=428 width=4) (actual time=0.395..0.514 rows=428 loops=1)
-> Sort (cost=81.21..82.28 rows=428 width=4) (actual time=0.394..0.430 rows=428 loops=1)
Sort Key: b.bid
Sort Method: quicksort Memory: 45kB
-> Seq Scan on boat b (cost=0.00..62.50 rows=428 width=4) (actual time=0.050..0.352 rows=428 loops=1)
Filter: (color = 'green'::bpchar)
Rows Removed by Filter: 2572
-> Index Scan using b_reservesbid on reserves r (cost=0.29..3.47 rows=28 width=8) (actual time=0.001..0.003 rows=5 loops=428)
Index Cond: (bid = b.bid)
-> Sort (cost=1993.87..2006.32 rows=4982 width=4) (actual time=1.479..1.534 rows=1136 loops=1)
Sort Key: r_1.sid
Sort Method: quicksort Memory: 102kB
-> Nested Loop (cost=81.45..1687.91 rows=4982 width=4) (actual time=0.412..1.289 rows=1136 loops=1)
-> Unique (cost=81.16..83.29 rows=427 width=4) (actual time=0.403..0.481 rows=427 loops=1)
-> Sort (cost=81.16..82.22 rows=427 width=4) (actual time=0.402..0.427 rows=427 loops=1)
Sort Key: b_1.bid
Sort Method: quicksort Memory: 45kB
-> Seq Scan on boat b_1 (cost=0.00..62.50 rows=427 width=4) (actual time=0.013..0.358 rows=427 loops=1)
Filter: (color = 'red'::bpchar)
Rows Removed by Filter: 2573
-> Index Scan using b_reservesbid on reserves r_1 (cost=0.29..3.48 rows=28 width=8) (actual time=0.001..0.001 rows=3 loops=4...
Index Cond: (bid = b_1.bid)
-> Index Scan using b_sailorssid on sailors s (cost=0.29..0.35 rows=1 width=25) (actual time=0.001..0.001 rows=1 loops=177)
Index Cond: (sid = r_1.sid)
Planning Time: 0.743 ms
Execution Time: 4.890 ms
```

Explanation :

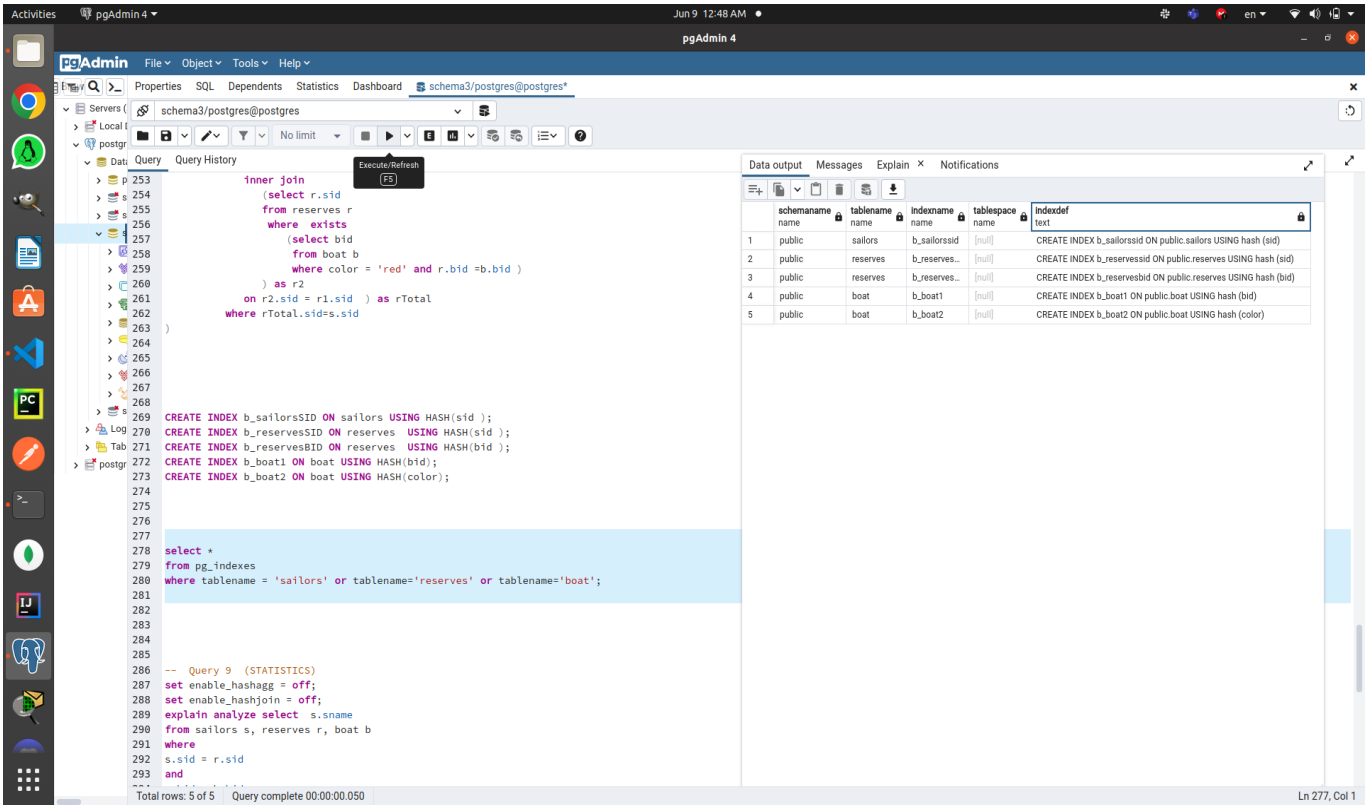
- Metrics :

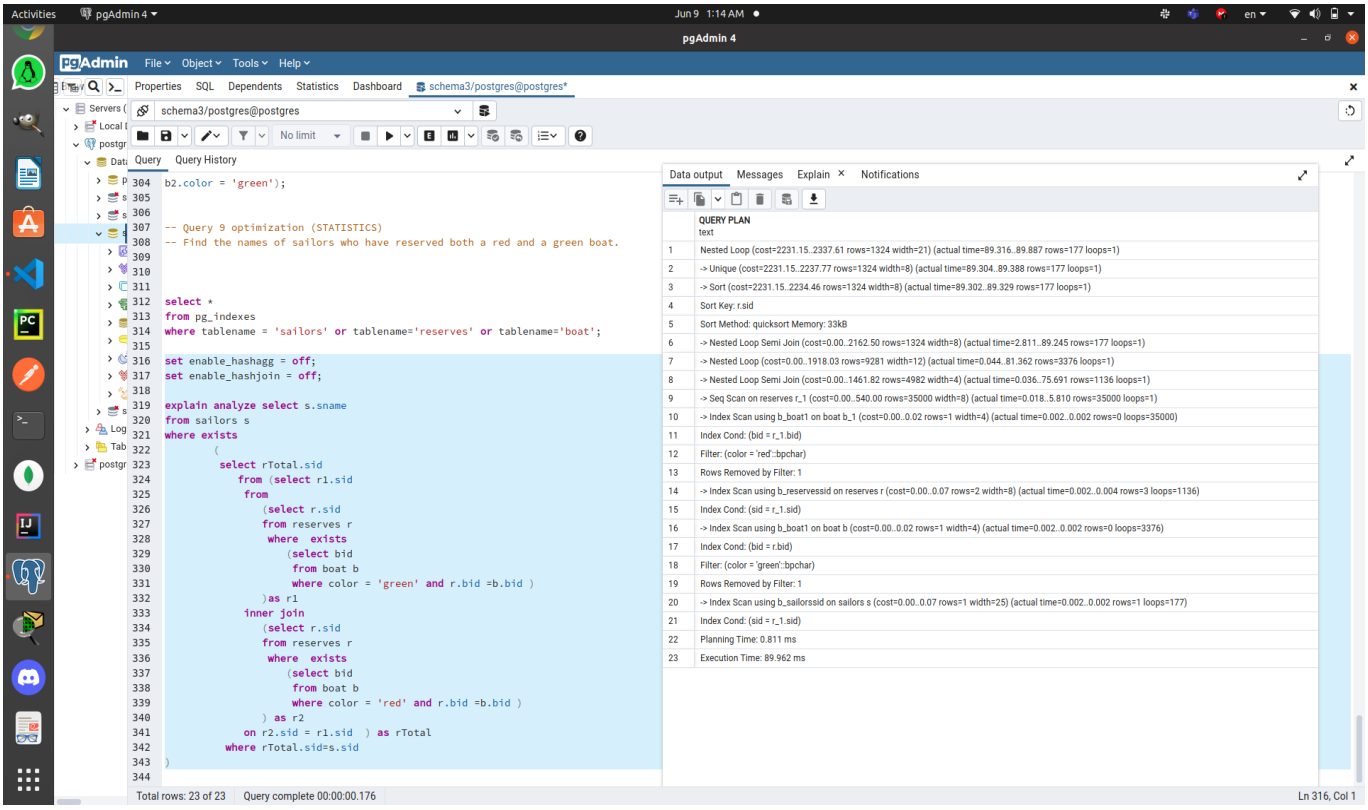
  Execution Time : 4.890 ms Total Expected Cost : 4605.10

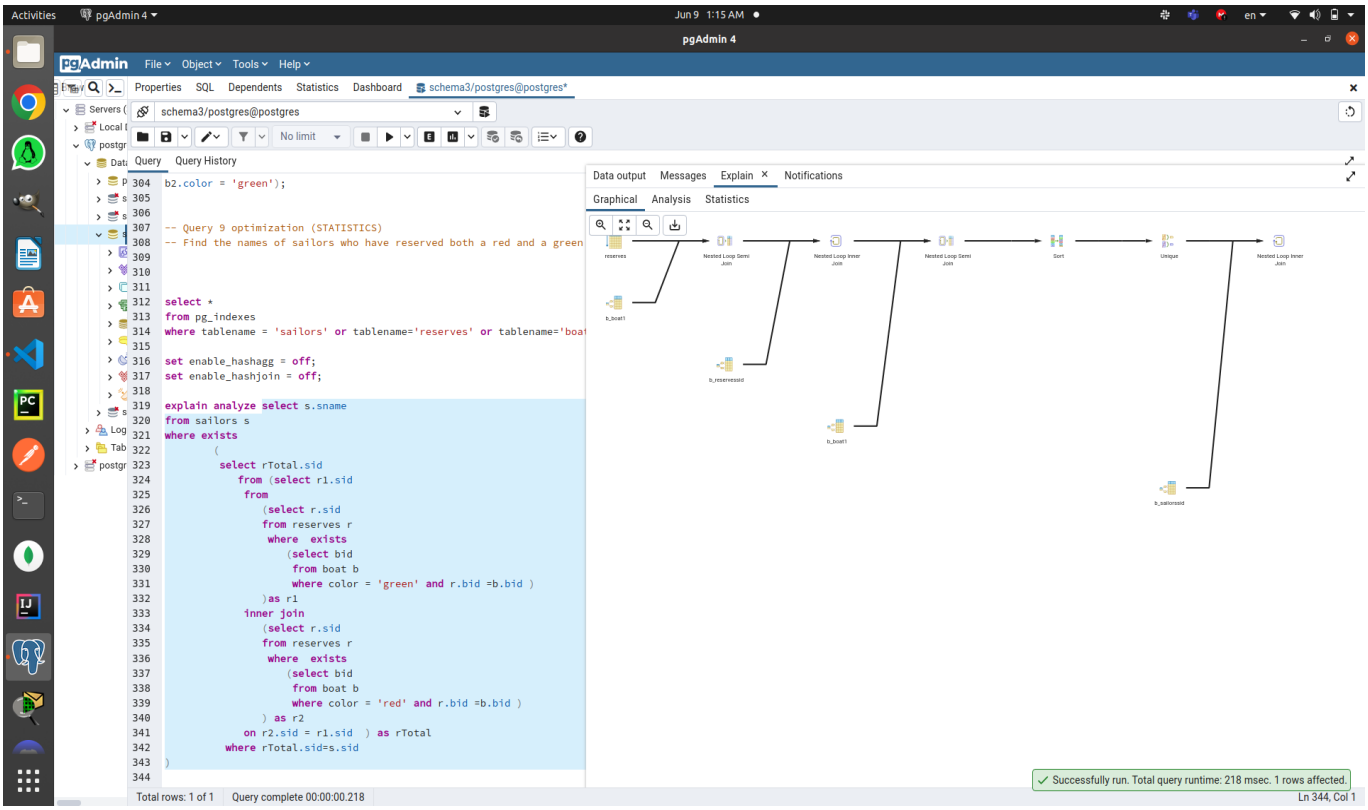
- The B-tree helped in the performance it decreased the Execution Time and Expected Cost .
- The Query Planner used the B-tree index because B-Tree is O(Log n) performance with Exact Values

- Here it showed the improvement due to for every condition of Joining in the Query the Merge Join used index scan using ZigZag algorithm .

3. given query with hash indices only,







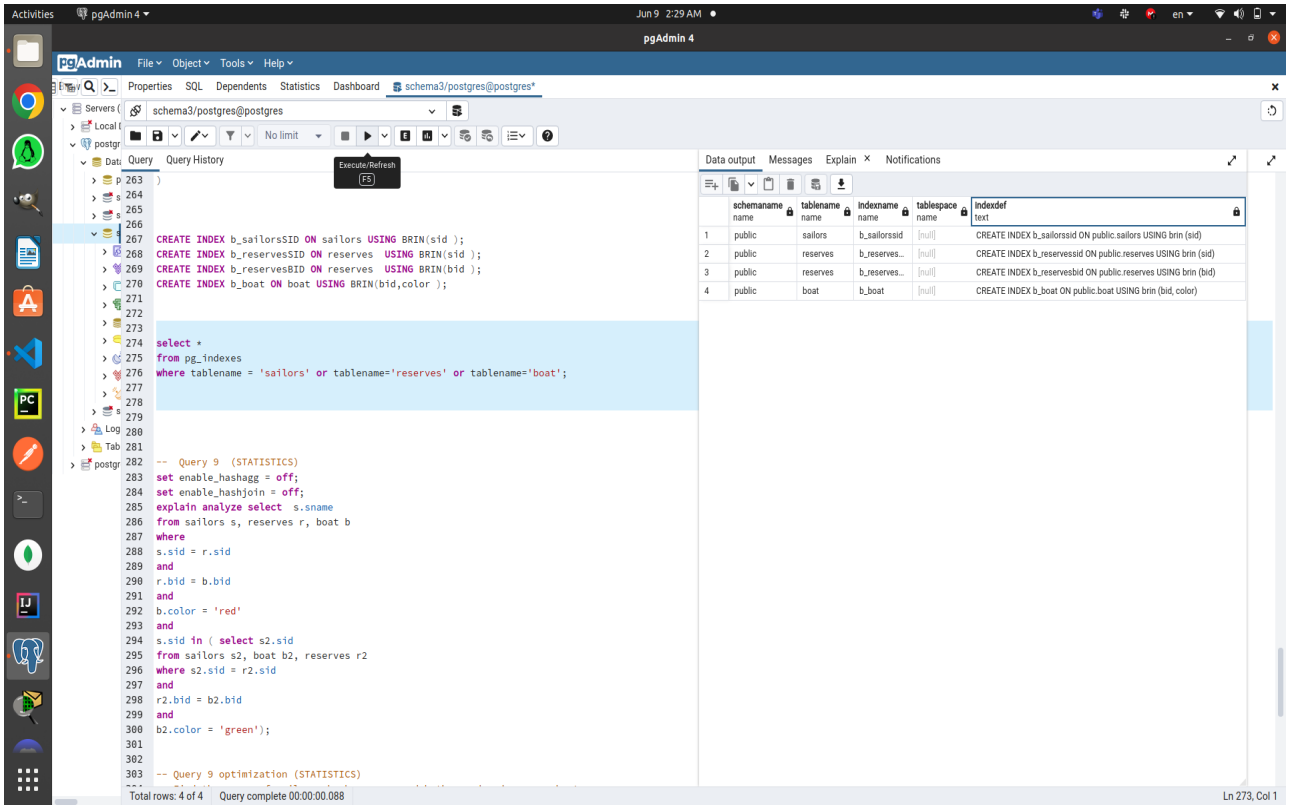
Explanation :

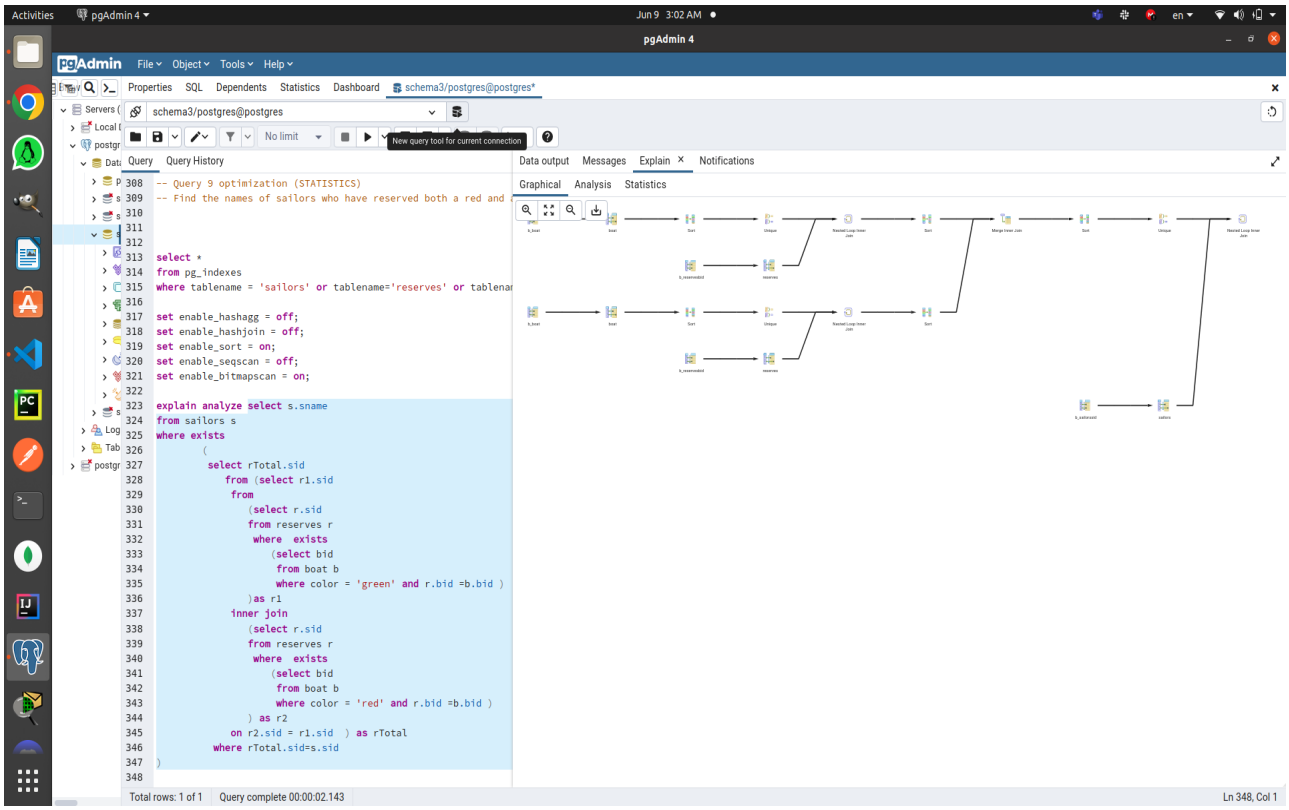
- Metrics :

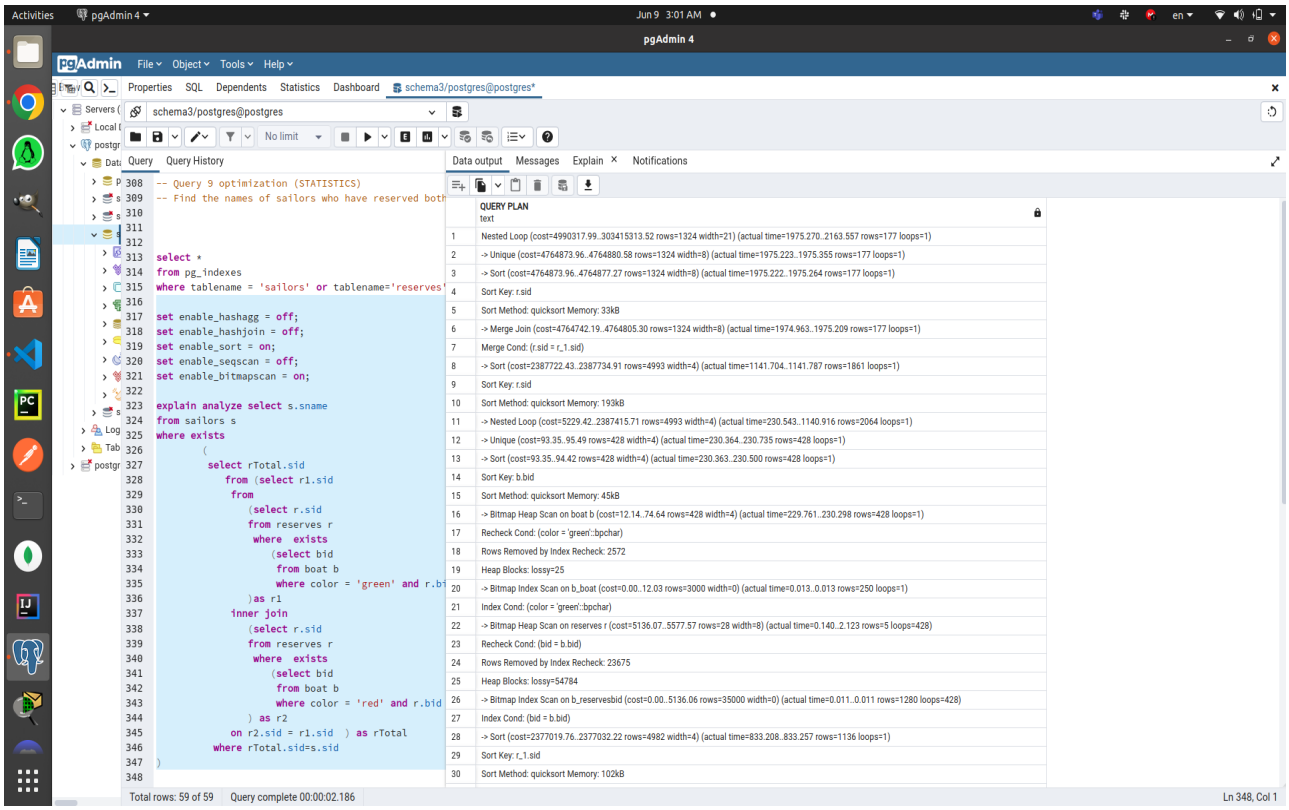
  Execution Time : 12.646 ms Total Expected Cost : 2337.61

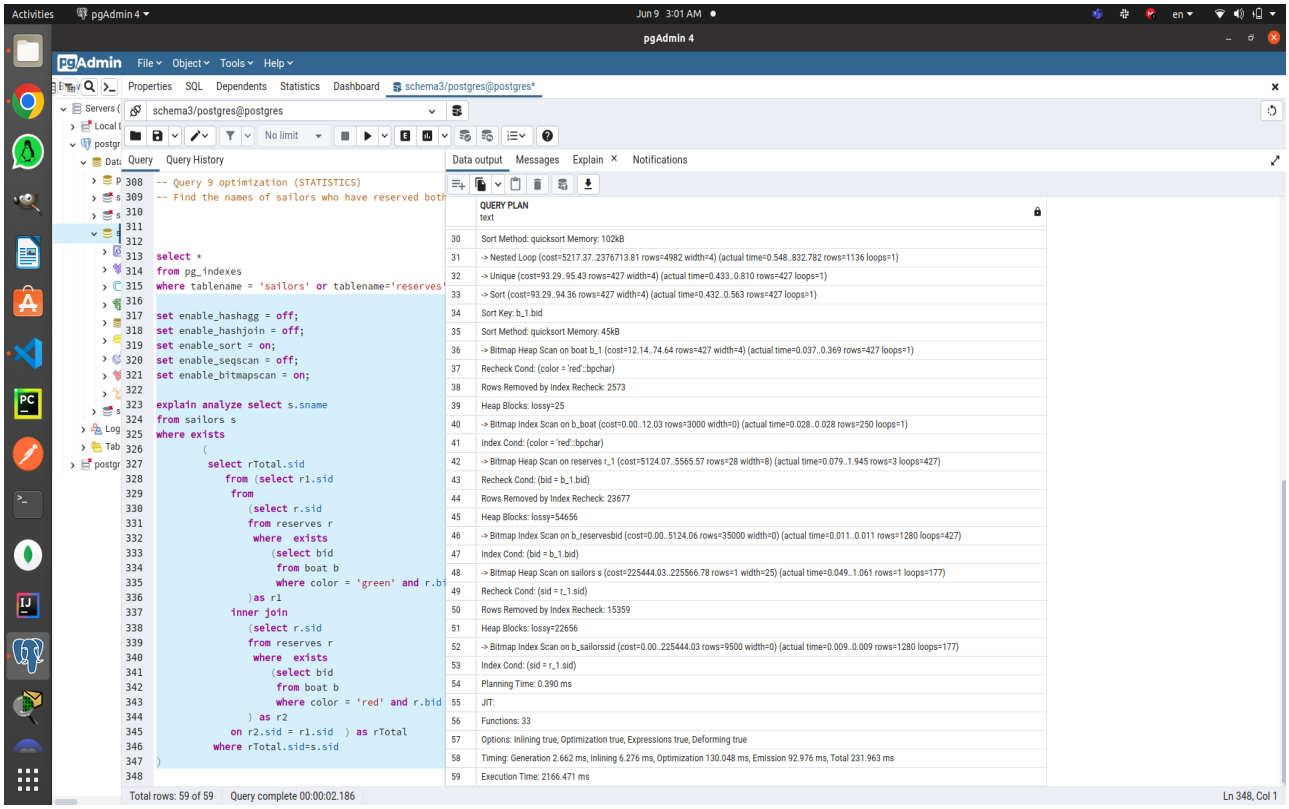
- The Hash helped in the performance it decreased the Execution Time(but in the execution time processor was overwhelmed + Linux performance) and Expected Cost .
- The Query Planner used the Hash index because Hash is O(1) performance with Exact Values and considered the best for exact values queries.
- Here it showed the improvement due to for every condition of Joining in the Query the Nested Loop Join using index scan using Hash based algorithm which approximatly maded to be O(n) performance.
- The Where clause condition on the color used Hash too.

4. given query with BRIN indices only,









Explanation :

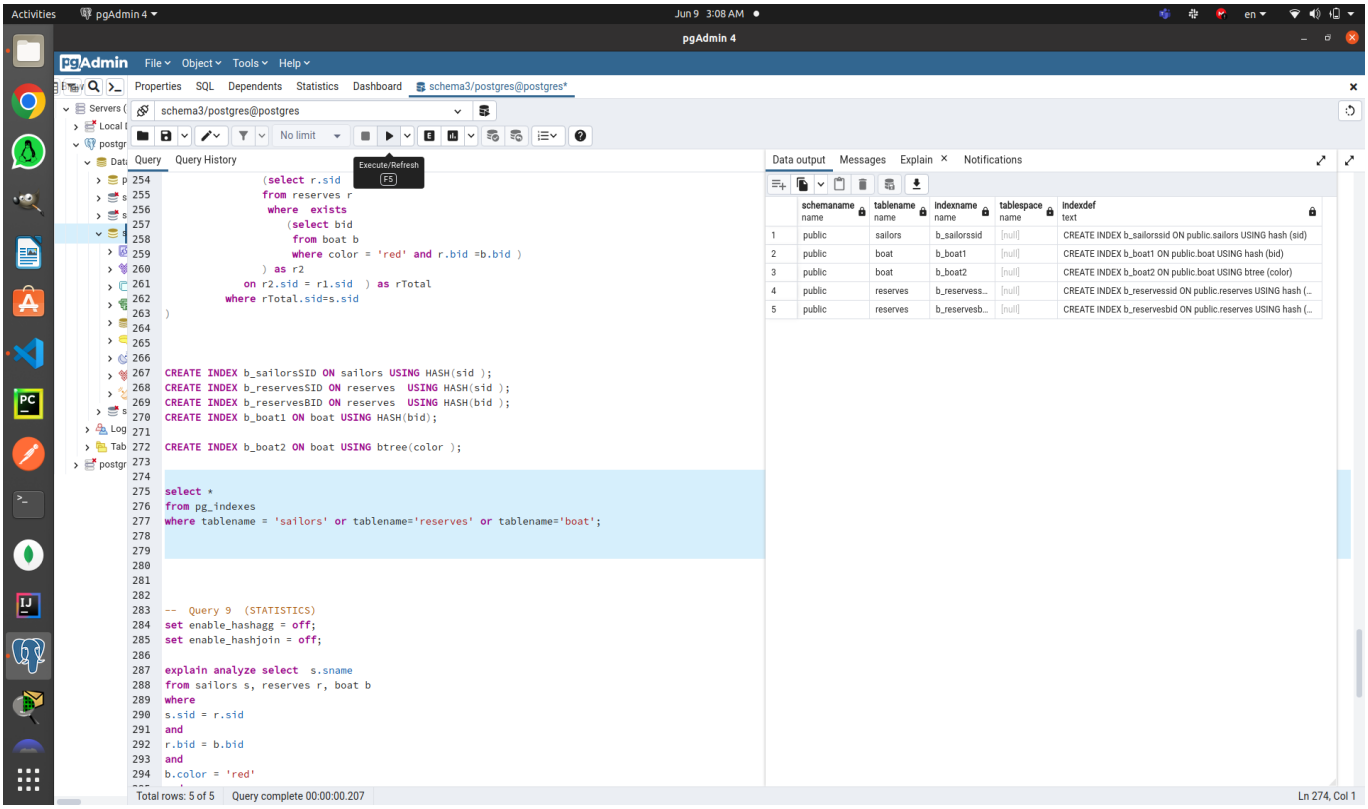
- Metrics :

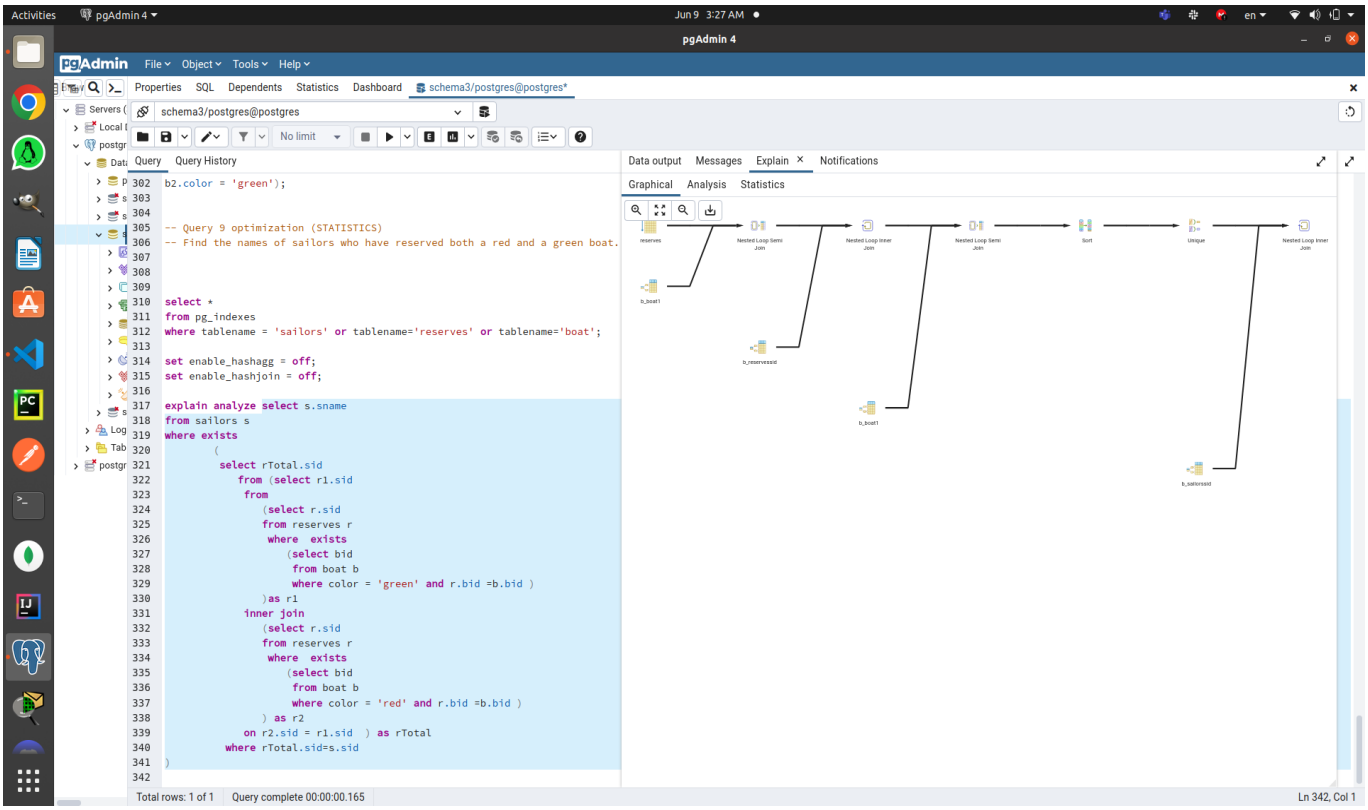
Execution Time : 4679 ms	**Total Expected Cost : 303415313.52**

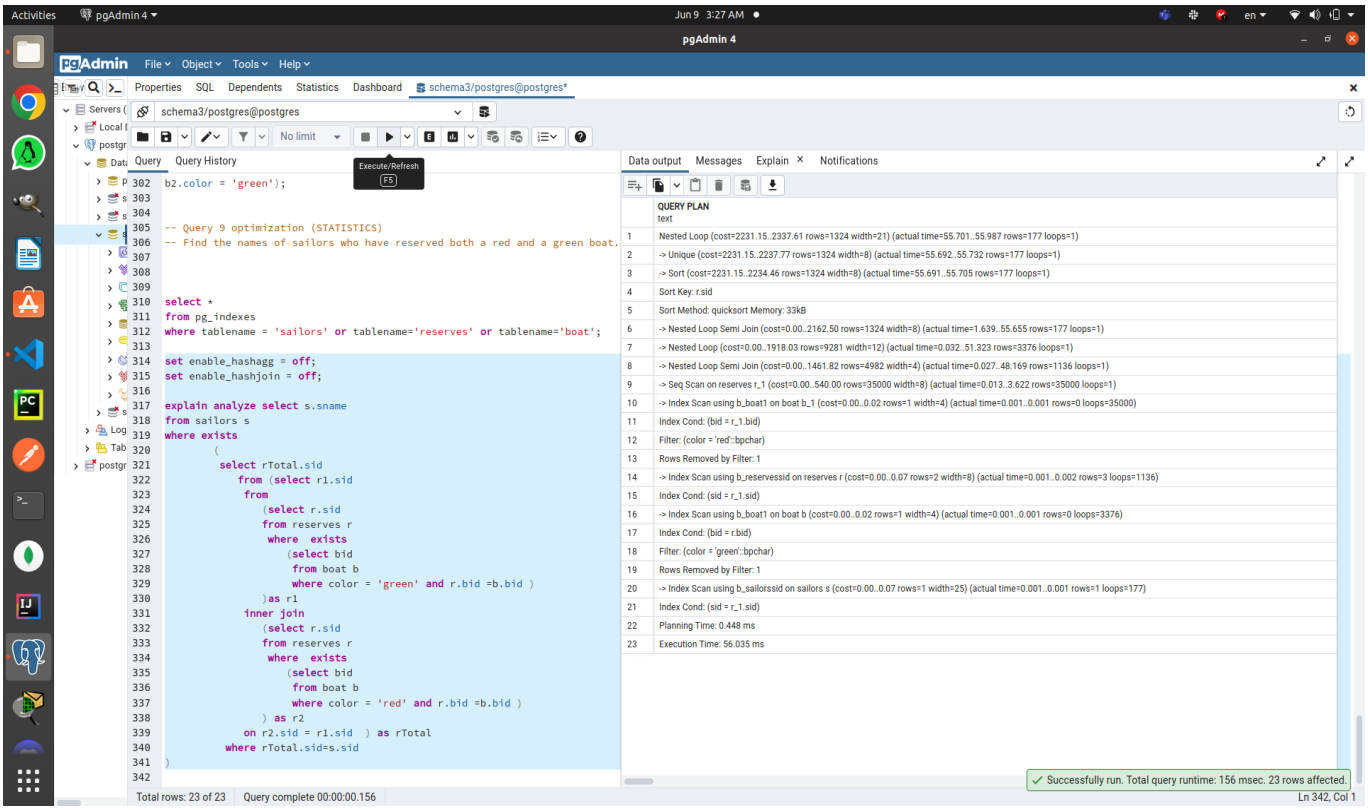
- Here the BRIN was not used in the original Query Plan settings (Hashjoin and HashAgg are off) so I've made seqscan=off too.

- The Execution Time and Expected Cost became the Worst of all .
- This happened because the Query Optimizer didnt used it from the place due to BRIN Usage here was not suitable so we have used it to .

5. given query with mixed indices (any mix of your choice)







Explanation :

- Metrics :

  Execution Time : 4679 ms Total Expected Cost : 2337.61

- The Query Planner used the Hash index because Hash is O(1) performance with Exact Values and considered the best for exact values queries.

- Here it showed the improvement due to for every condition of Joining in the Query the Nested Loop Join using index scan using Hash based algorithm which approximatly maded to be O(n) performance.

- The Where clause condition on the color used B-tree which is more better as colors are repetable and sorted after each other at the leaves of the b-tree with O(Log n) performance on all at one time little bit better than Hash O(1) but calculation of hashfunction and stored in different buckets .