Query 9

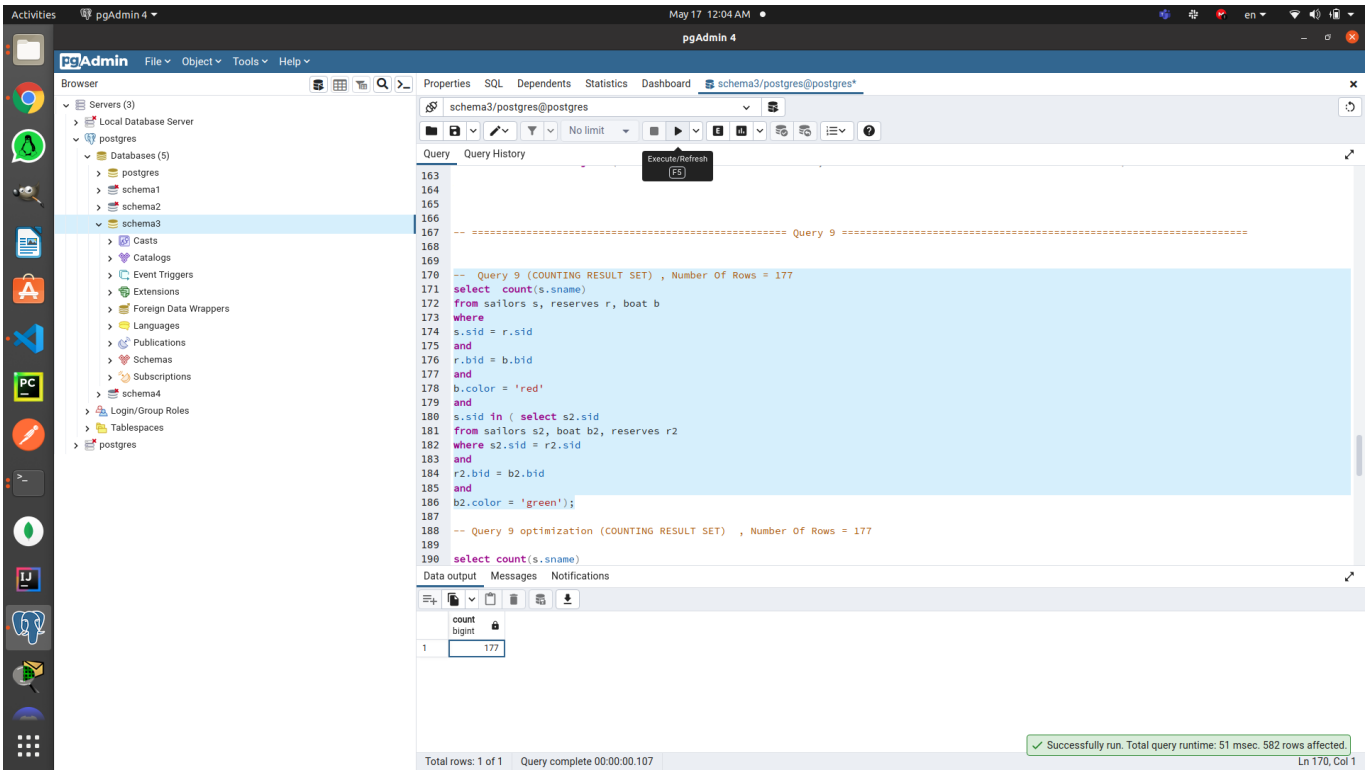
- Find the names of sailors who have reserved both a red and a green boat.

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');
```

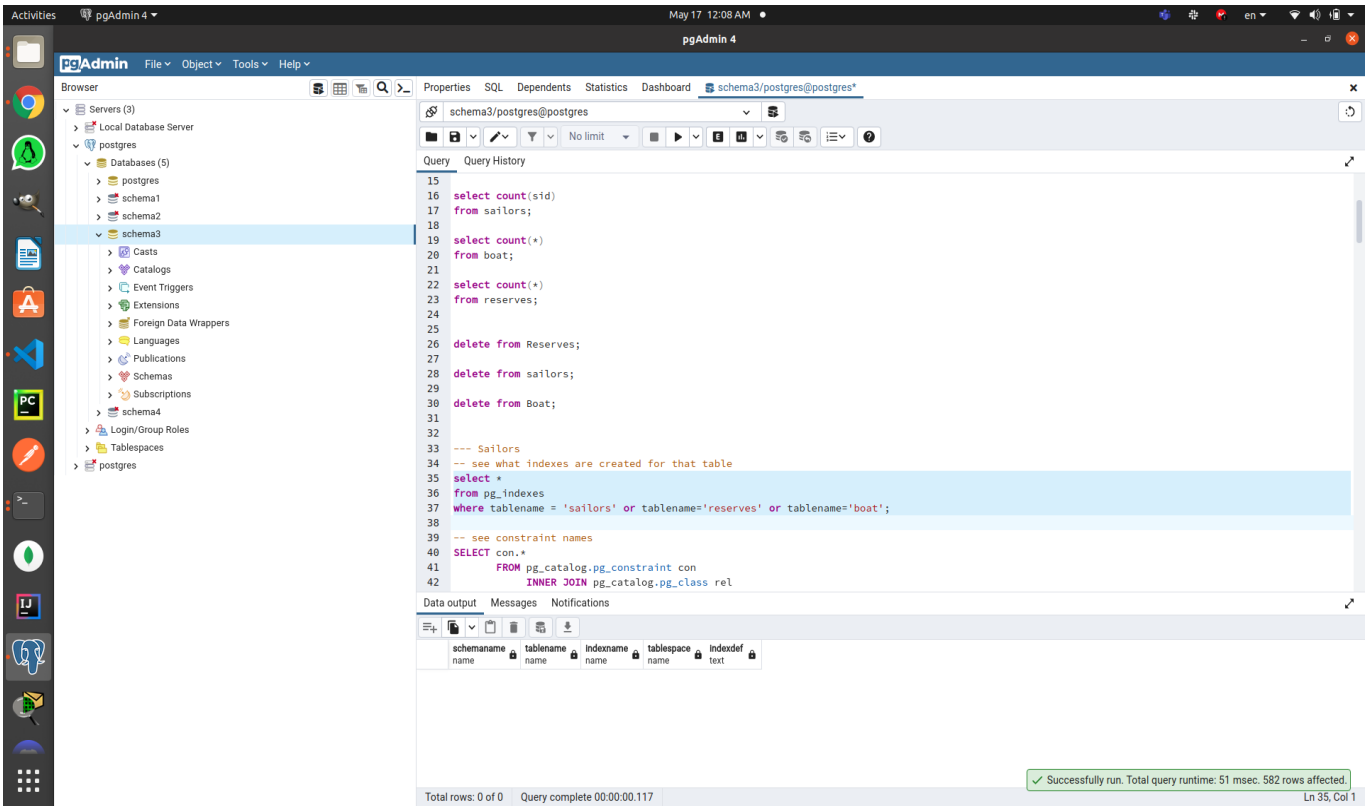
Result Set

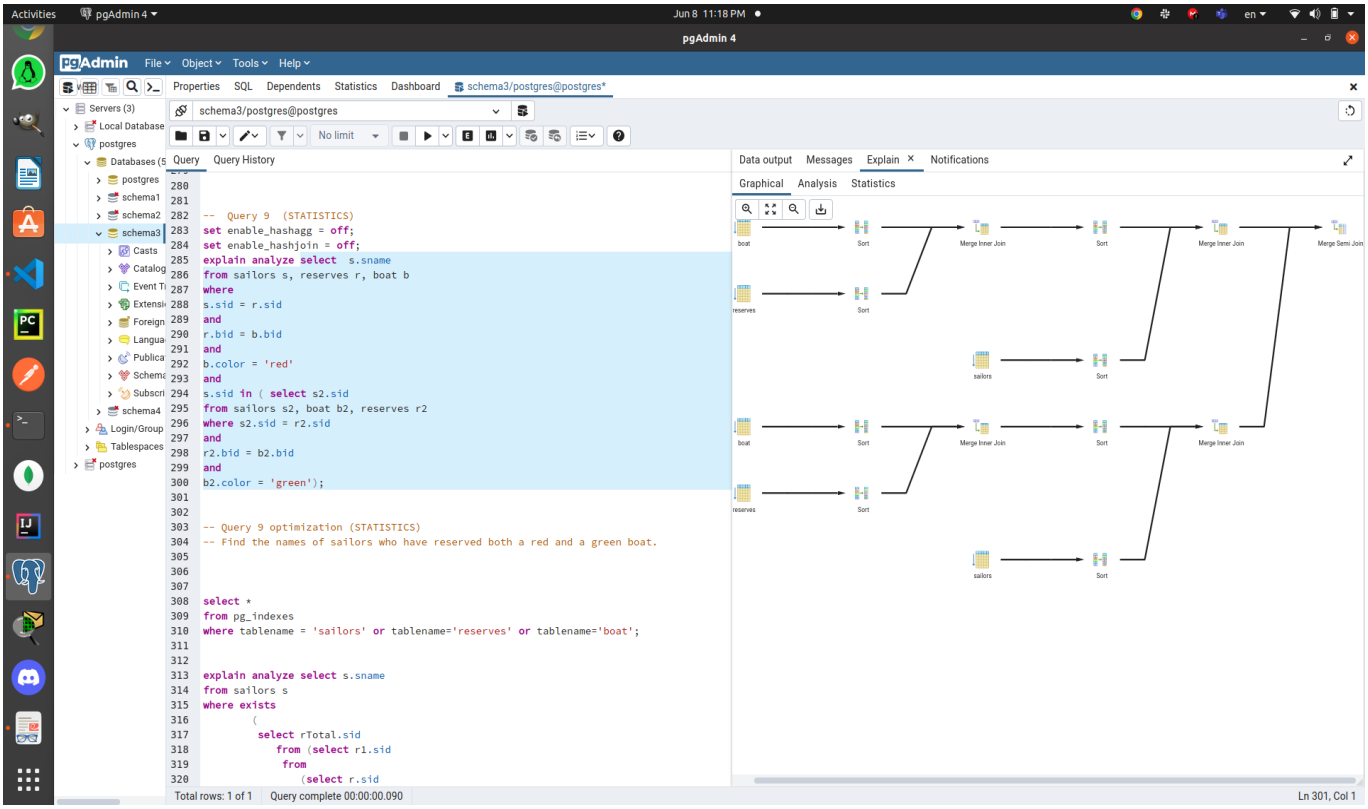
- 177 Rows

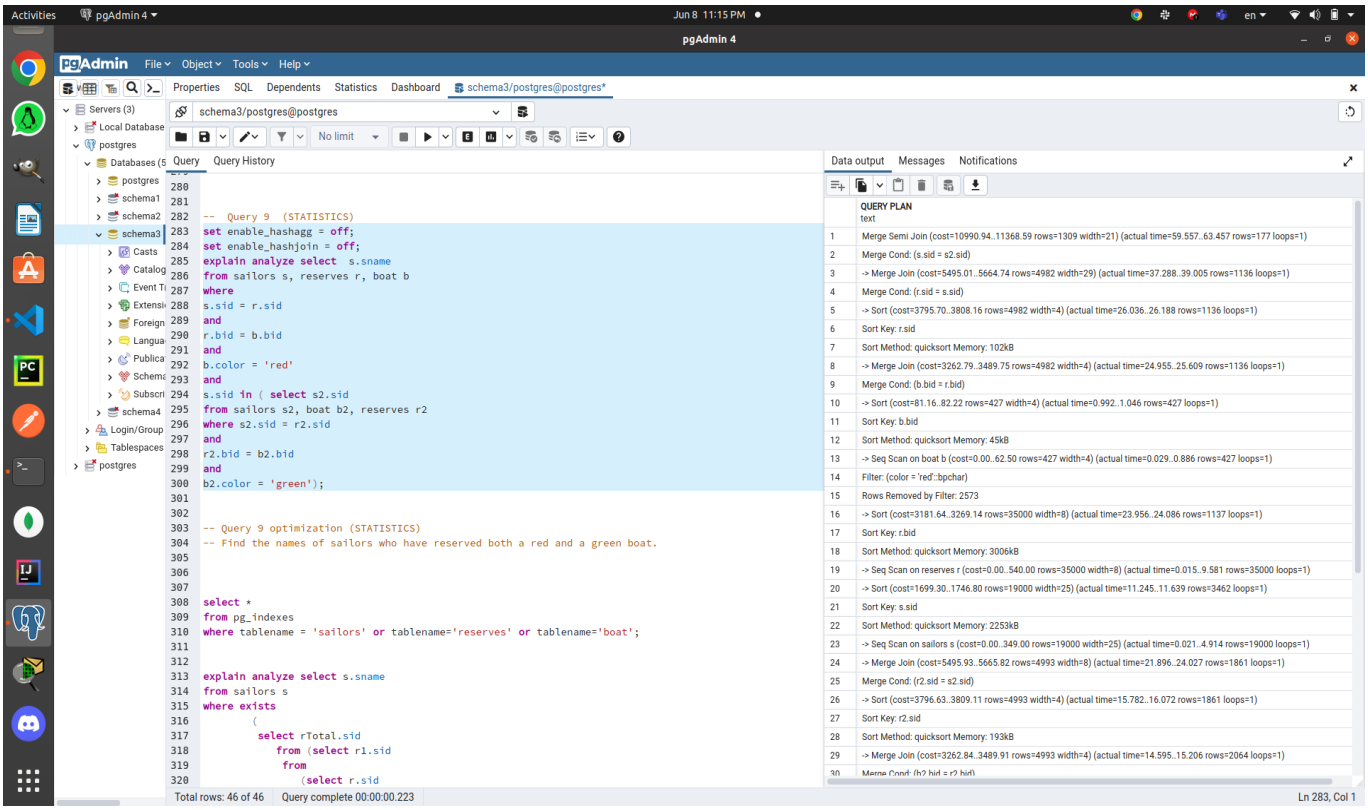


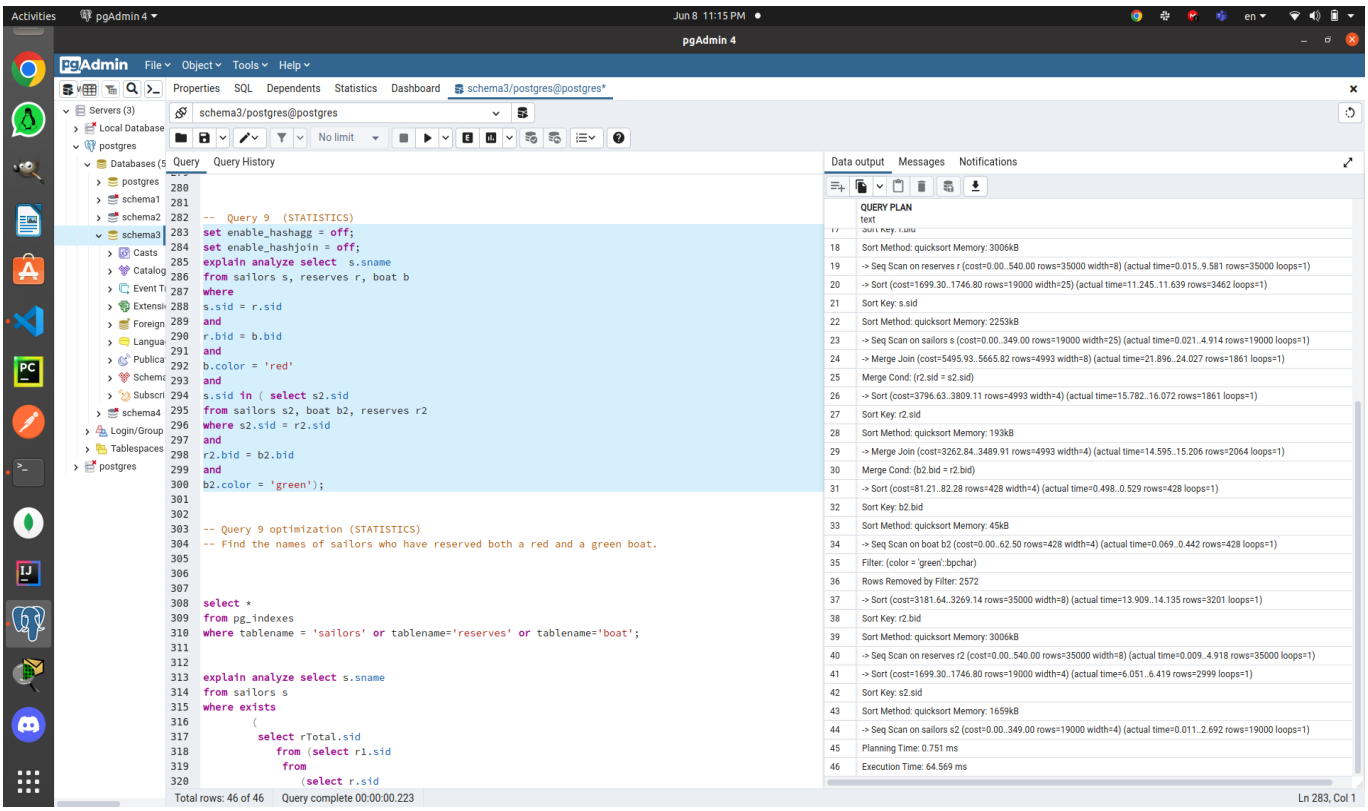
Report

1. given query without an index,









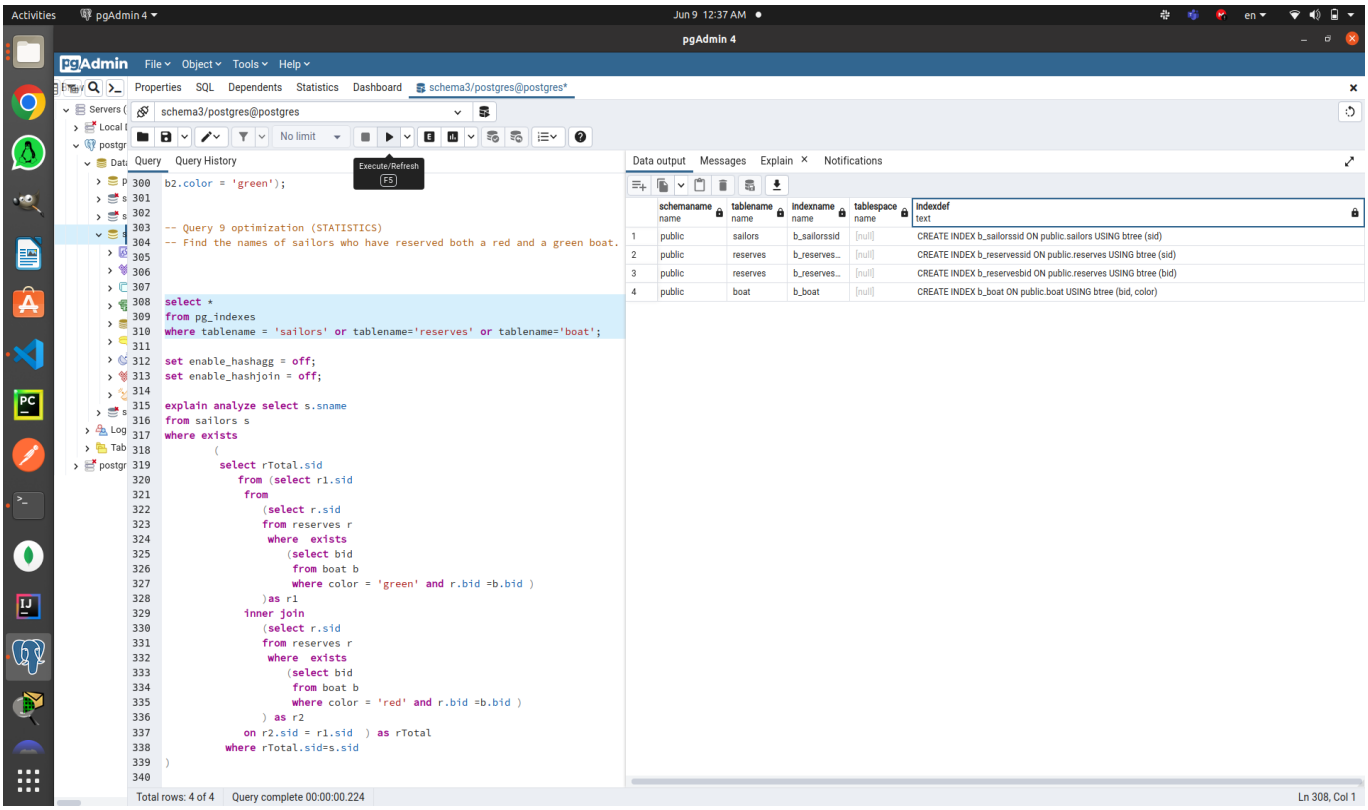
Explanation :

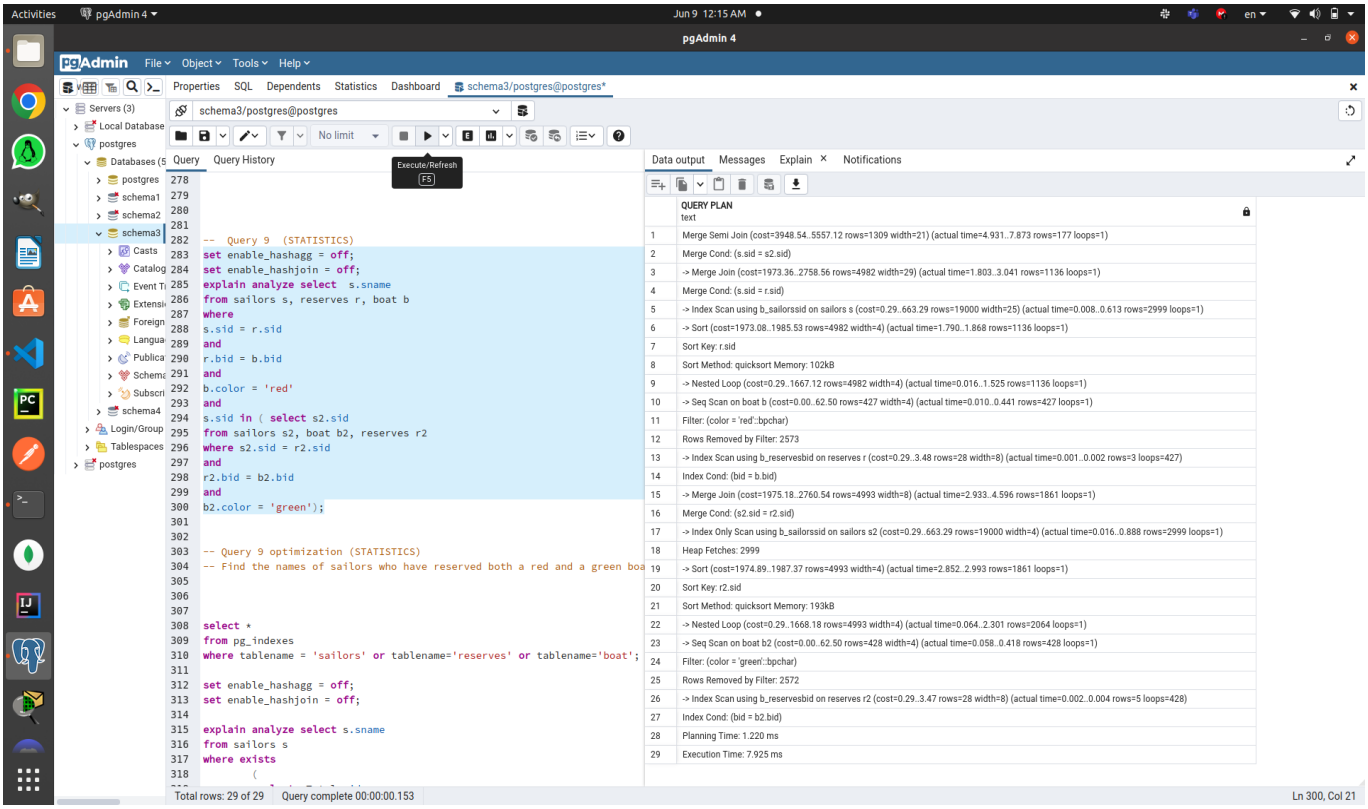
- Metrics :

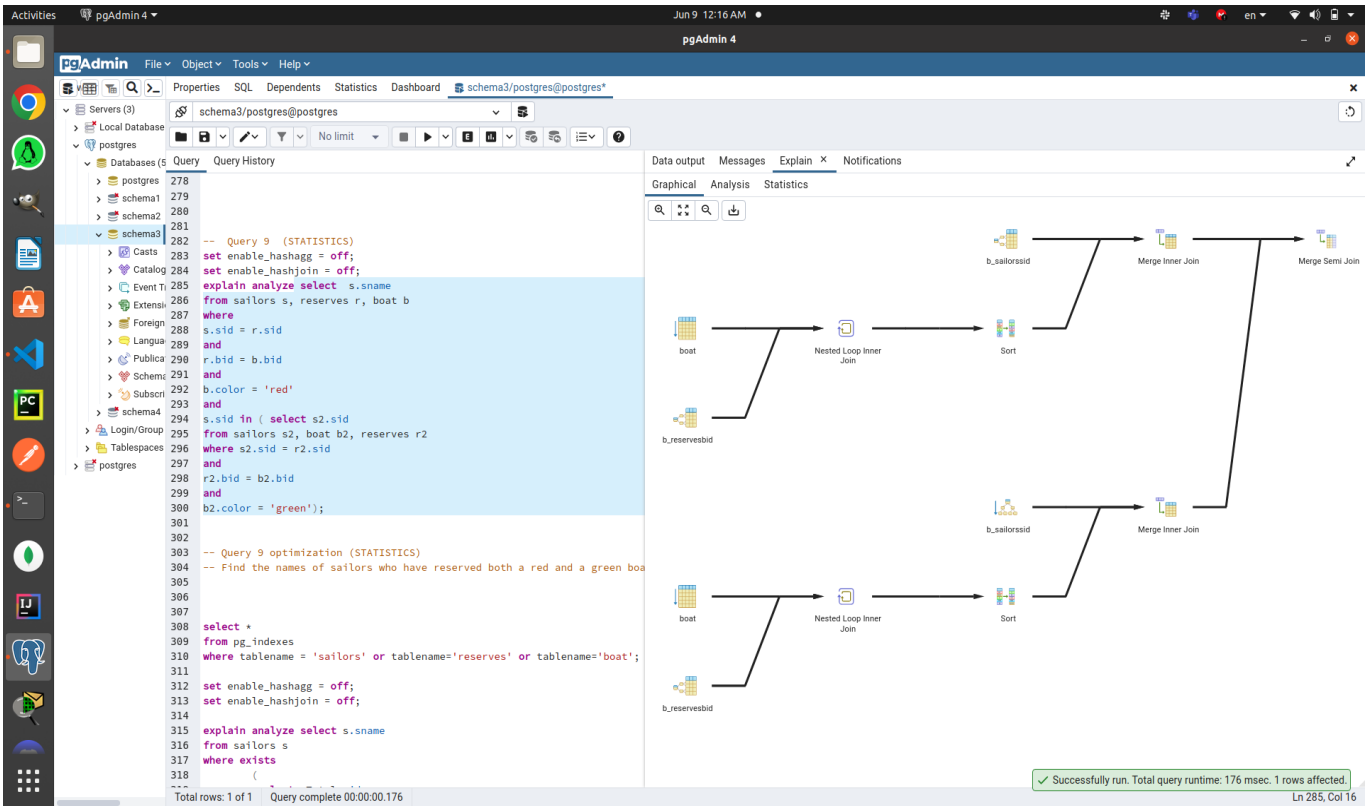
```
| Execution Time : 64.569 ms | Total Expected Cost : 11368.59 |
|----------------------------|--------------------------------|
```

- 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 .

2. given query with B+ trees indices only,







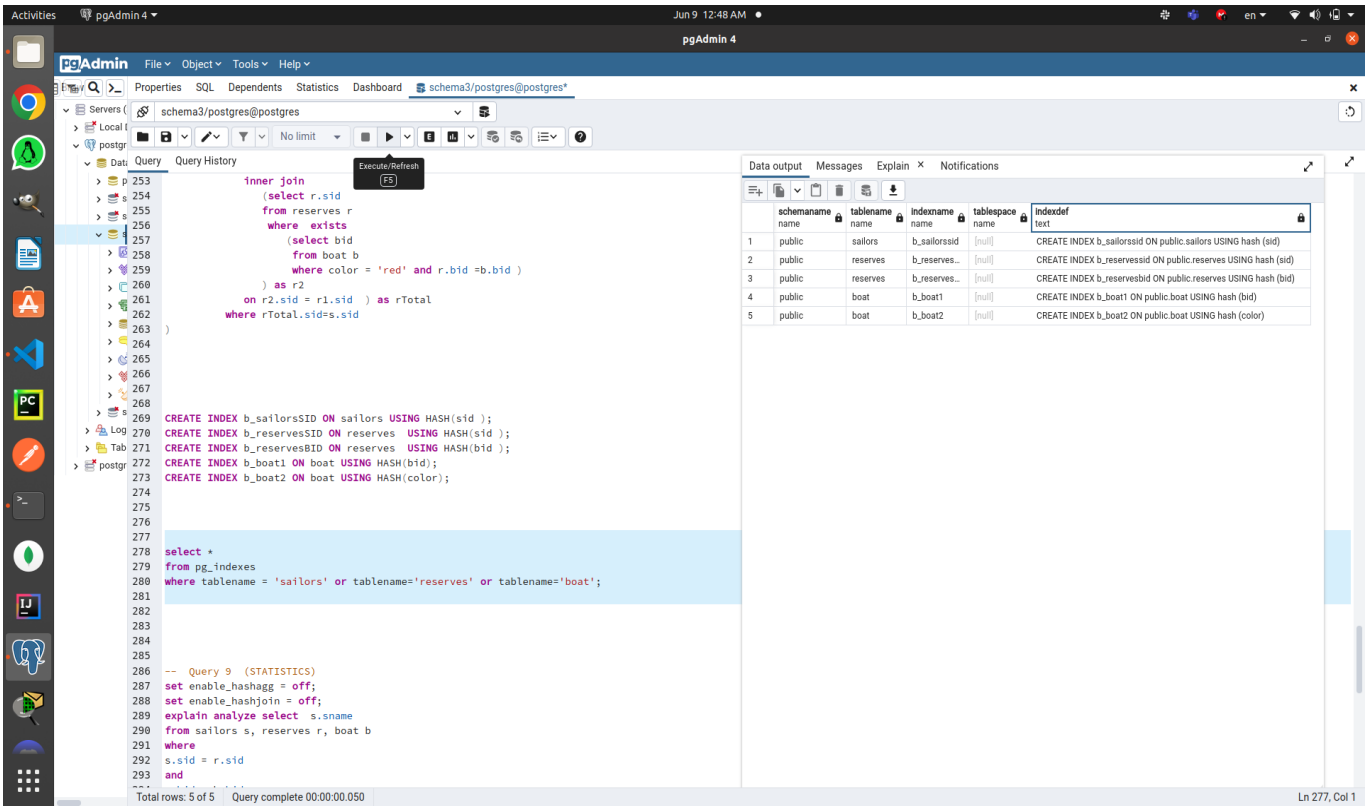
Explanation :

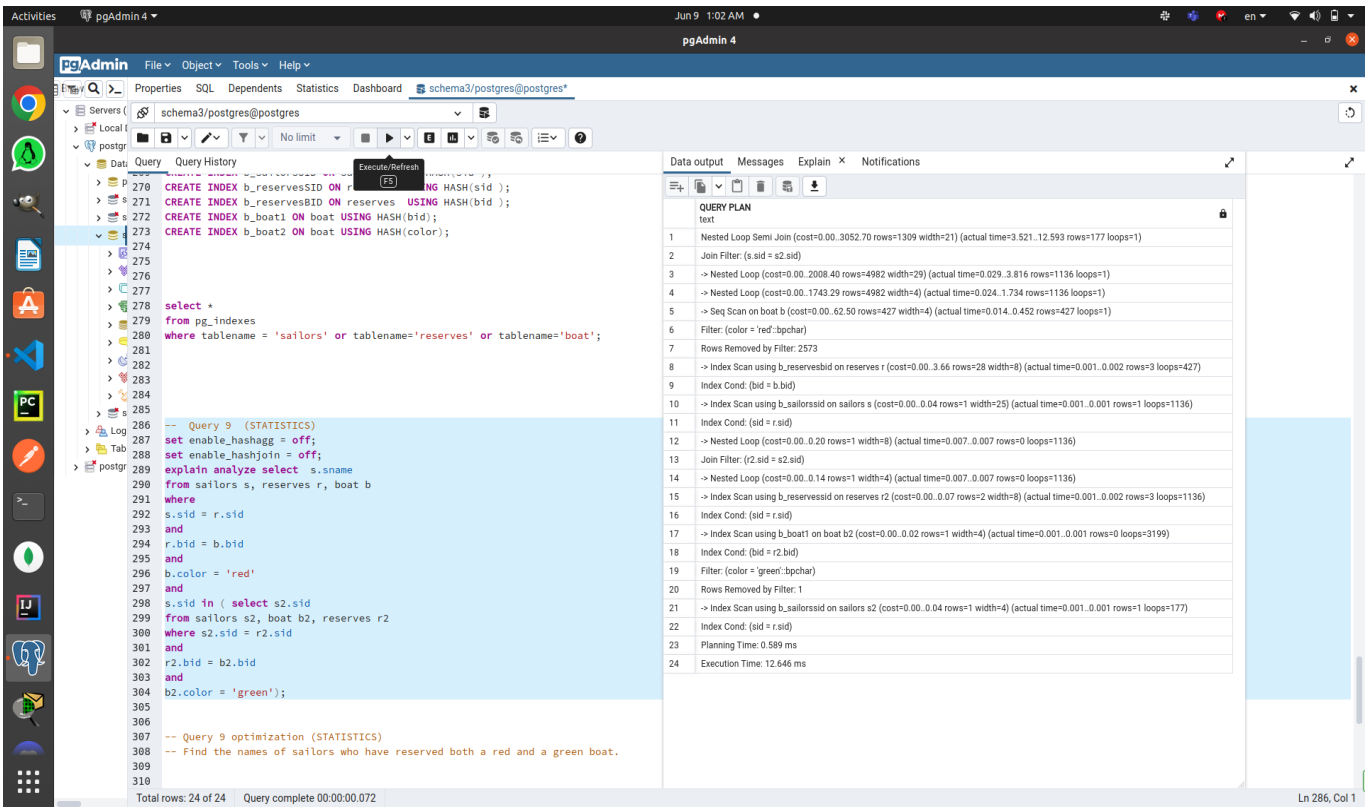
- Metrics :

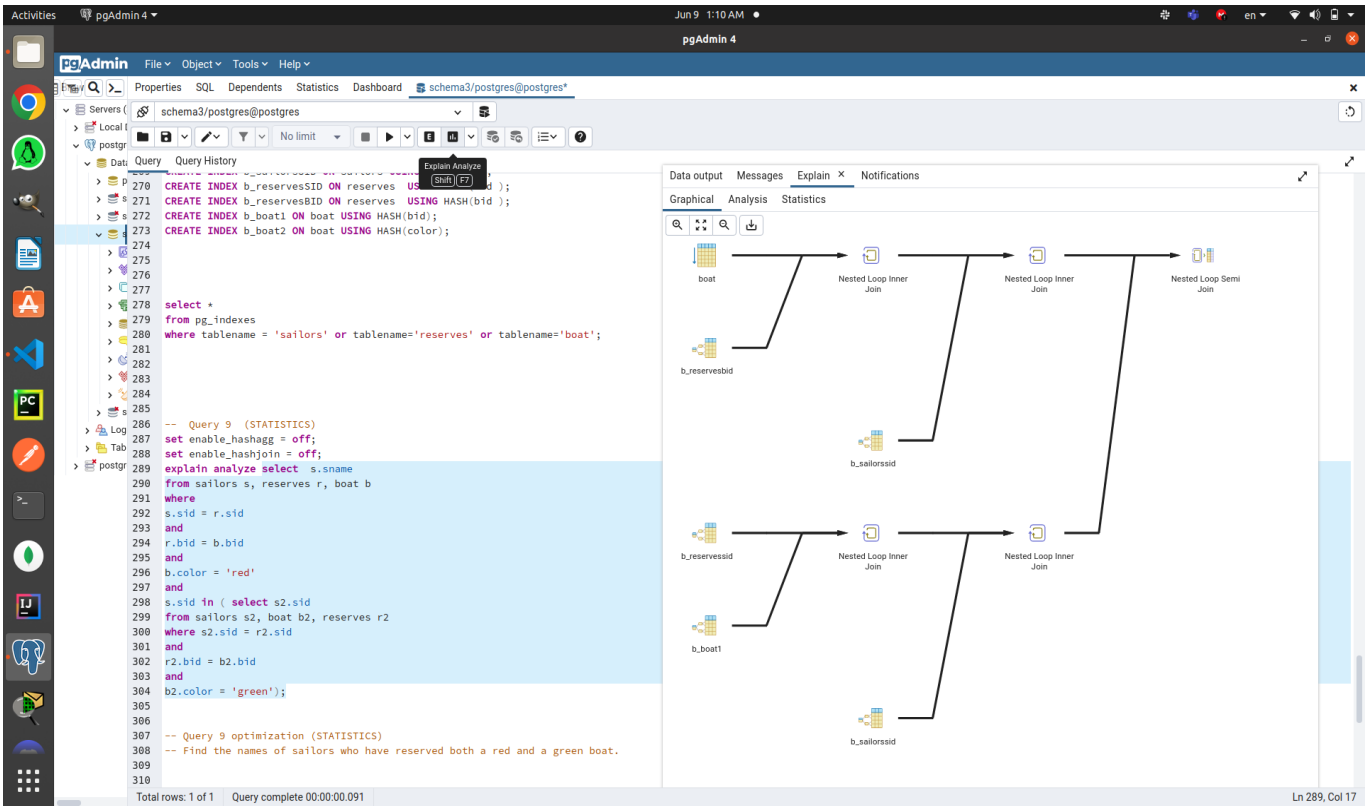
```
| Execution Time : 7.925 ms | Total Expected Cost : 5557.12 |
|---------------------------|-------------------------------|
```

- 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 tablename indexname tablespace indexdef
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
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



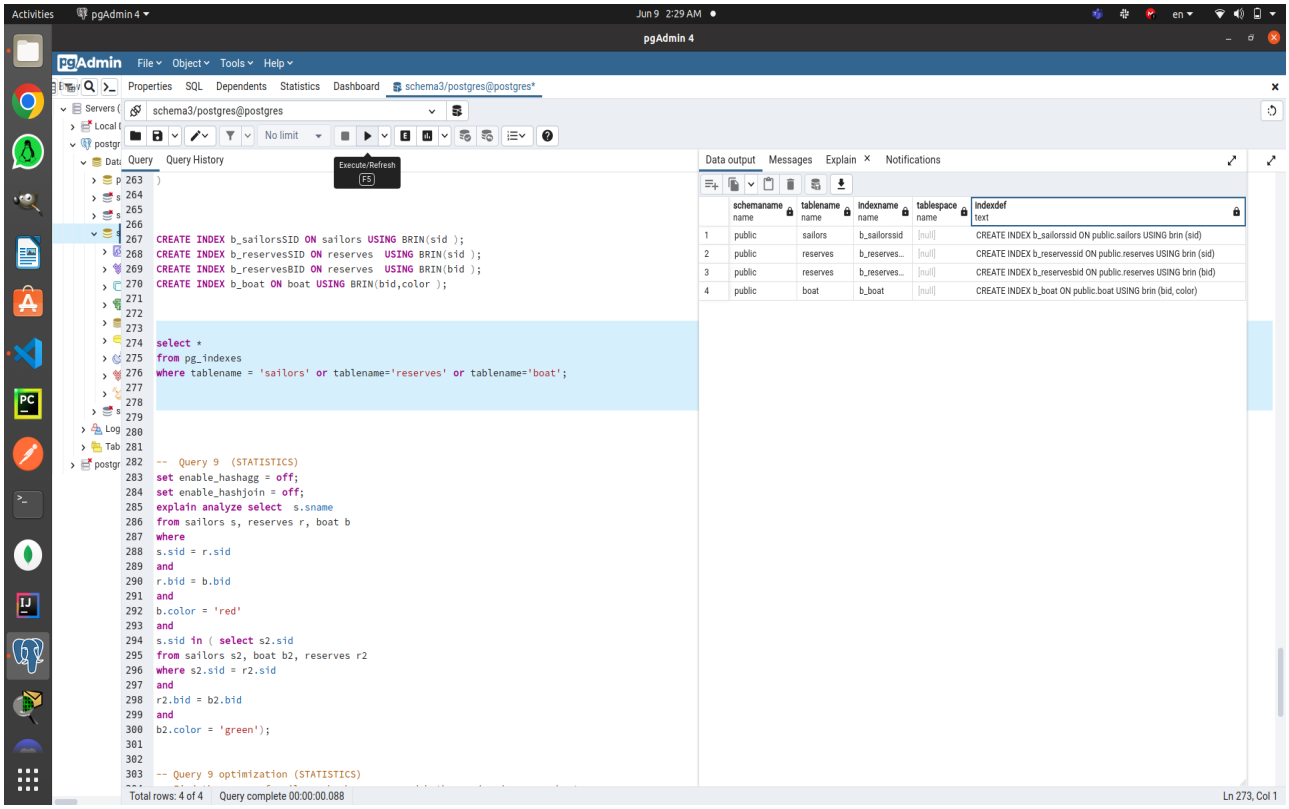
Explanation :

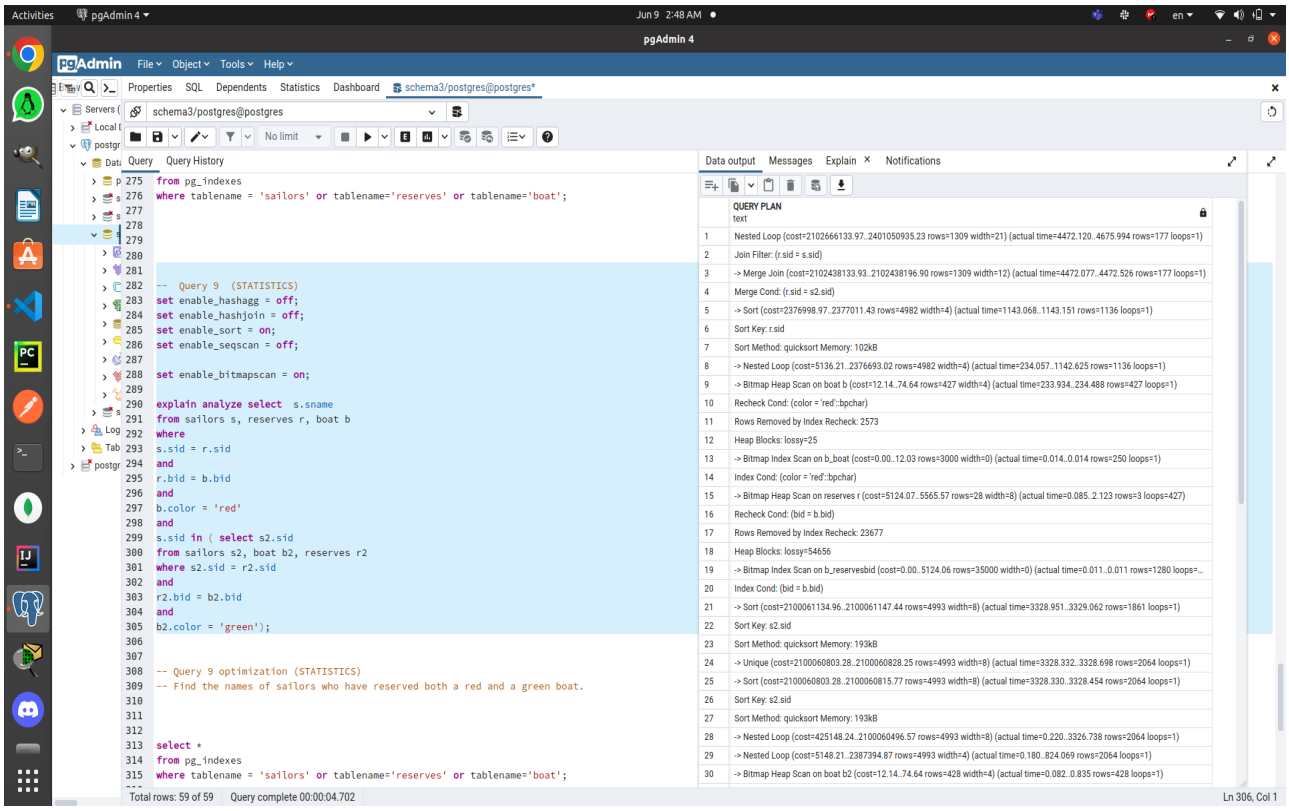
- Metrics :

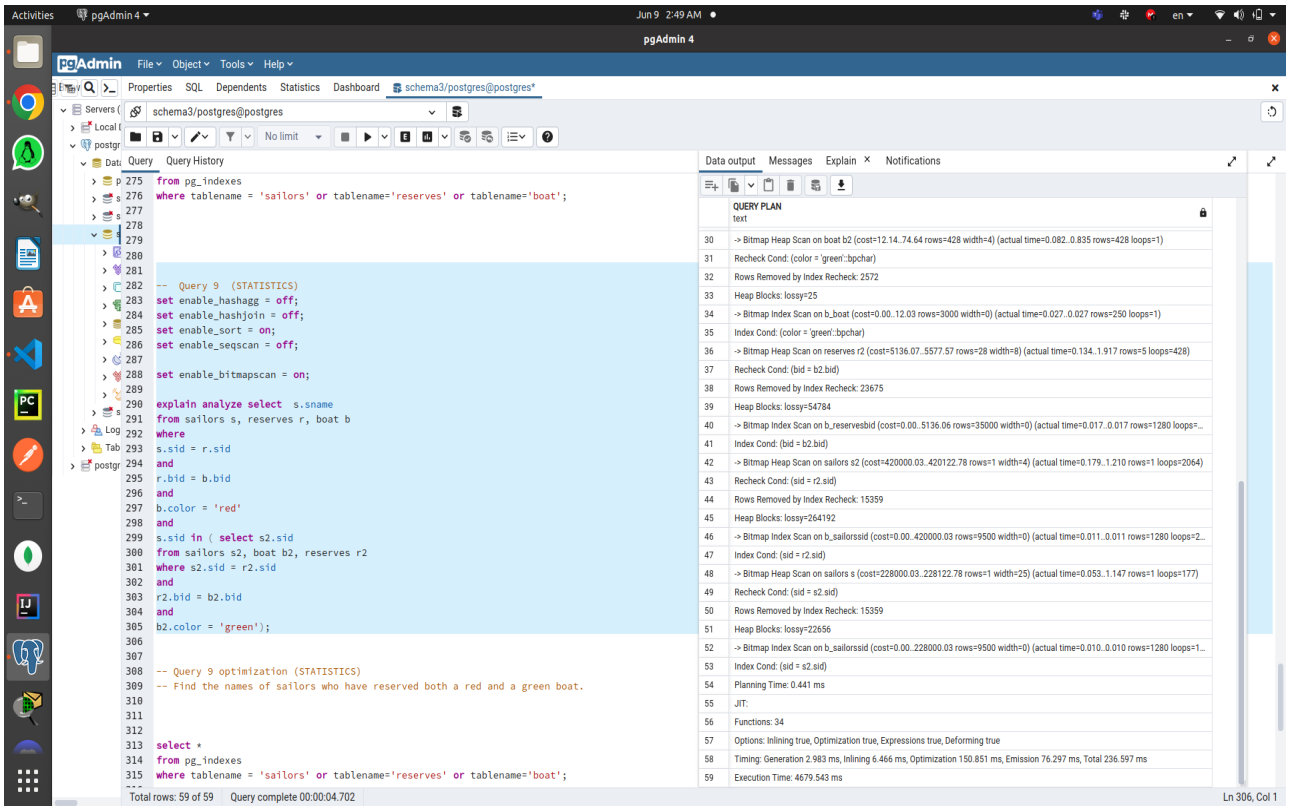
```
| Execution Time : 12.6 ms | Total Expected Cost : 3052.12 |
|--------------------------|-------------------------------|
```

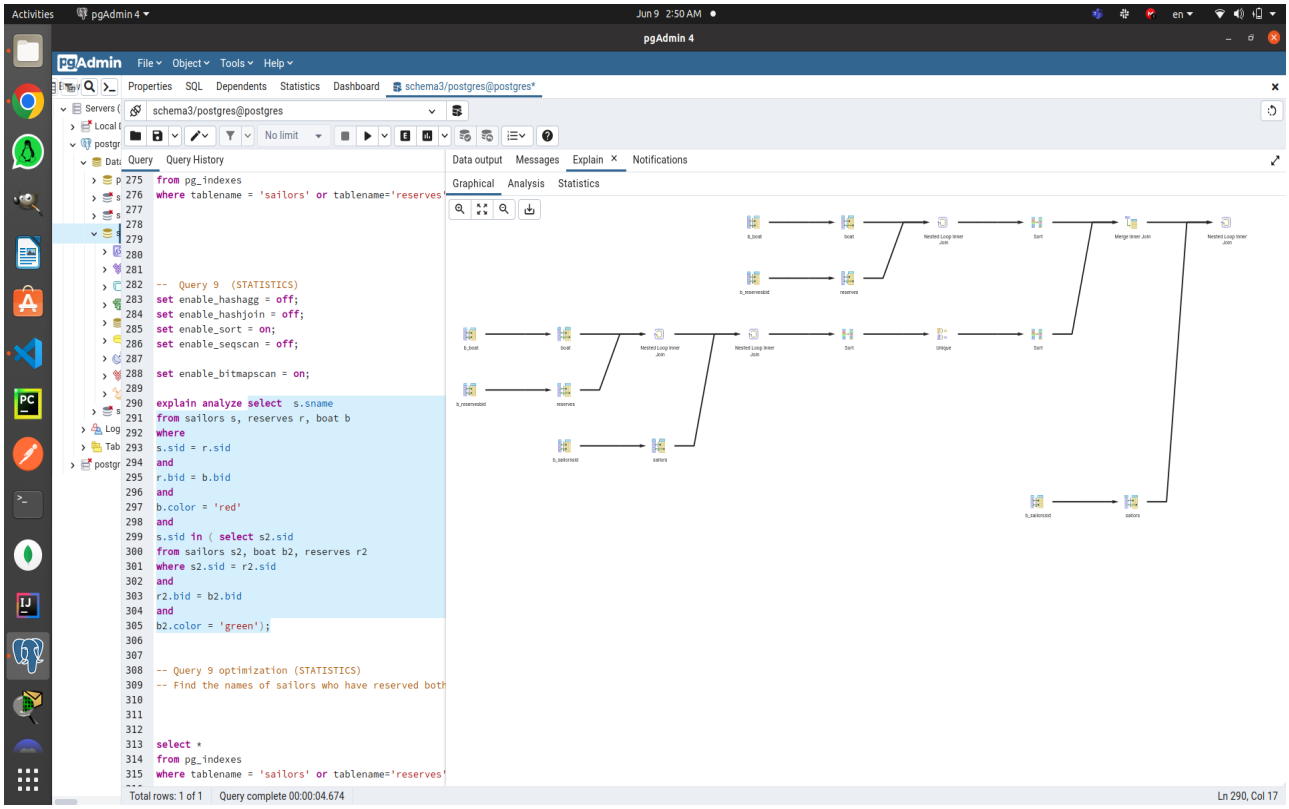
- The Hash helped in the performance it decreased the Execution Time(but in the execution time processor was overwhelmed) and Expected Cost .
- The Query Planner used the Hash index because Hash is O(1) performance with Exact Values and considered the best for exact values queries.
- Here it showed the 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,









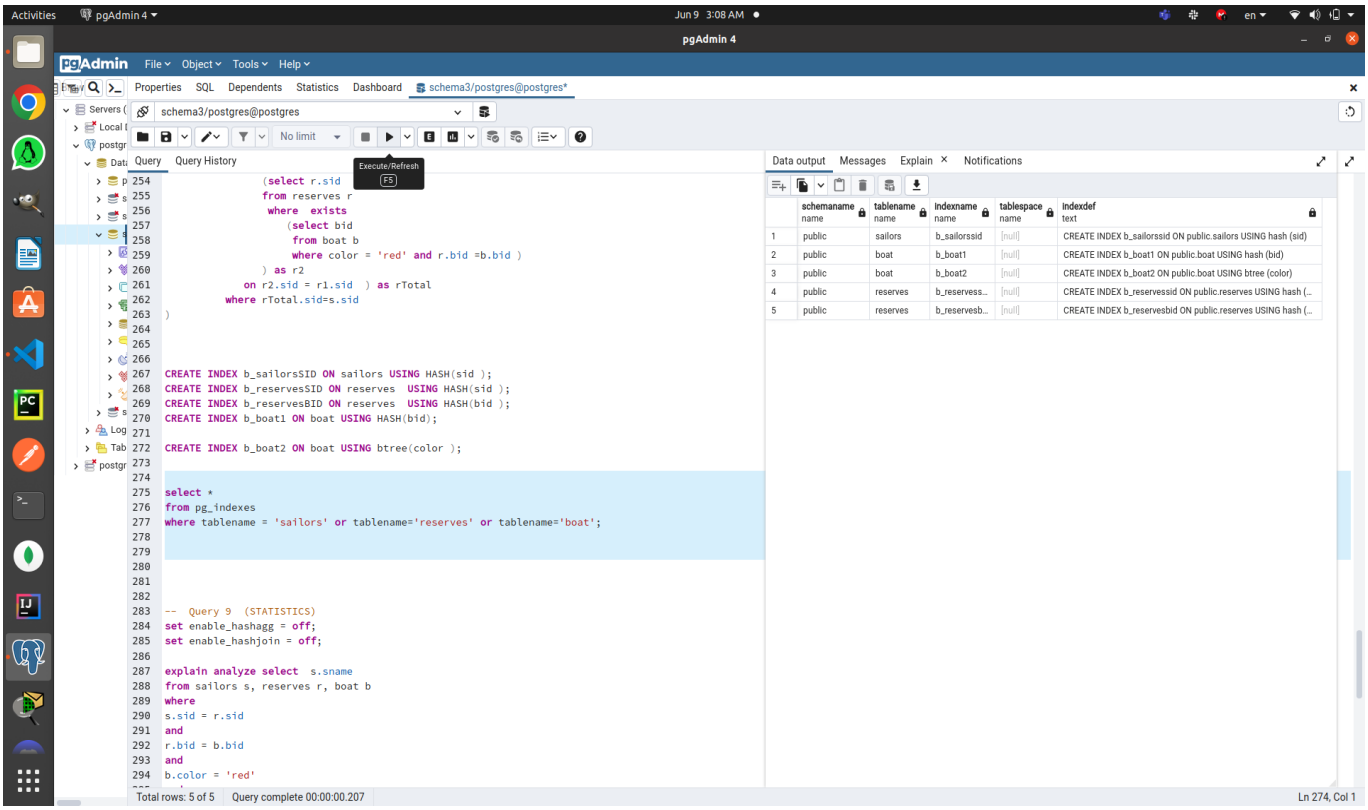
Explanation :

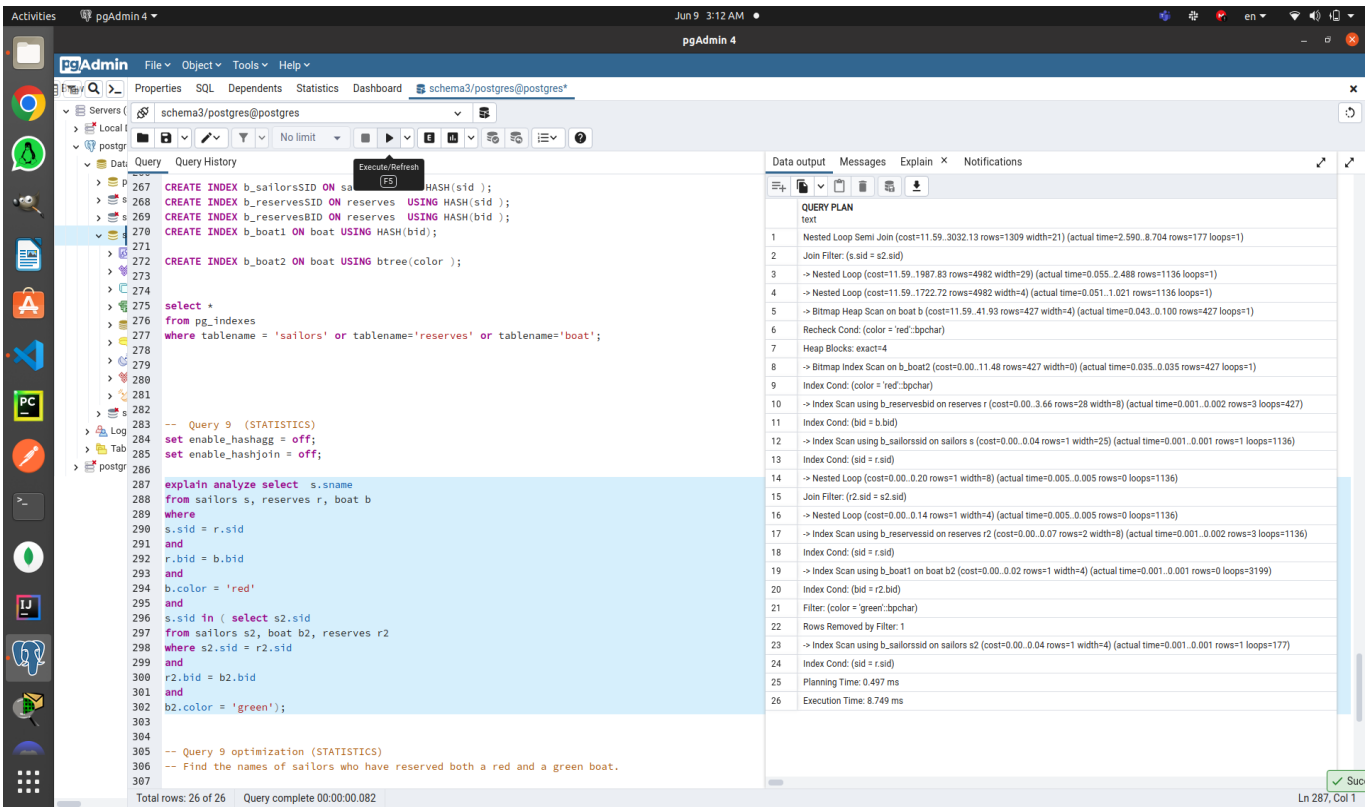
- Metrics :

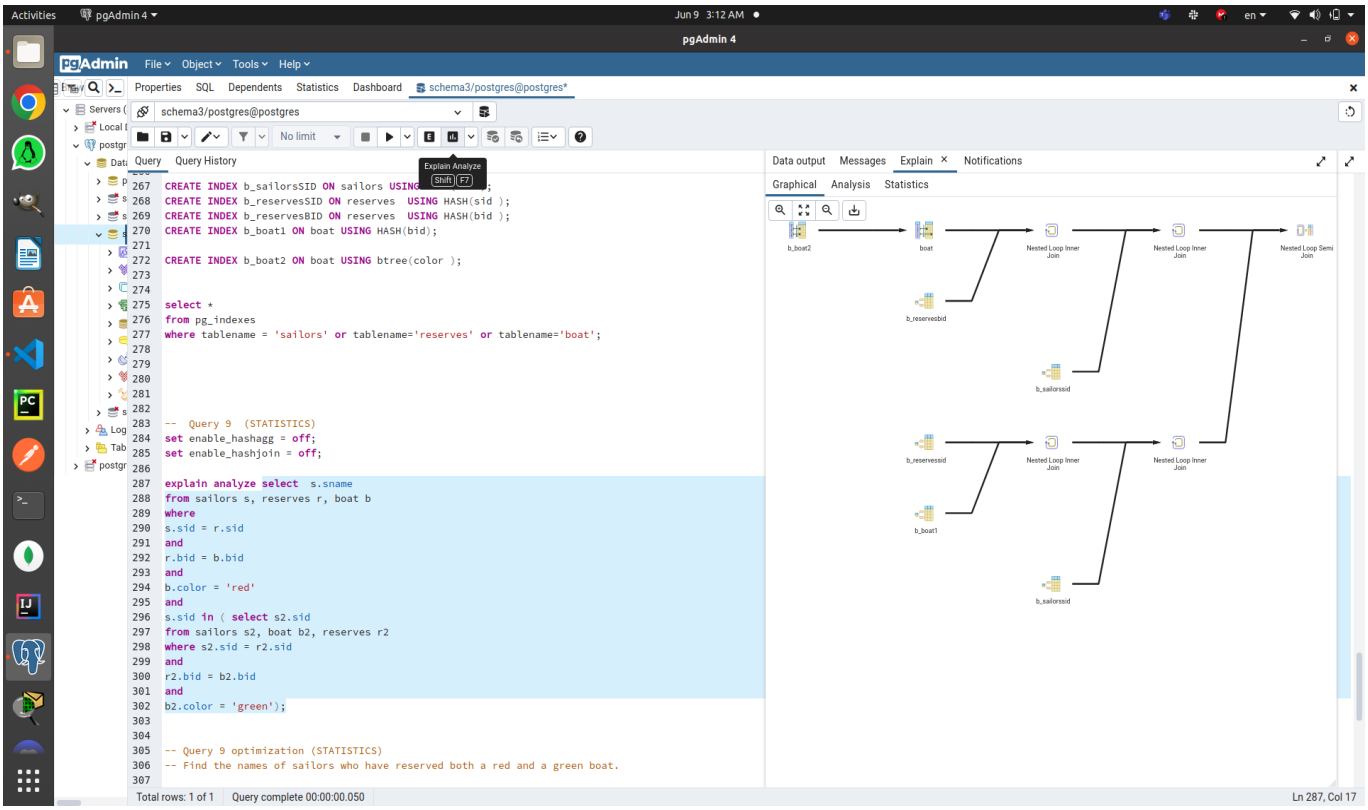
```
| Execution Time : 4679 ms | Total Expected Cost : 2401050935.23  |
|--------------------------|-------------------------------------|
```

- 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 .

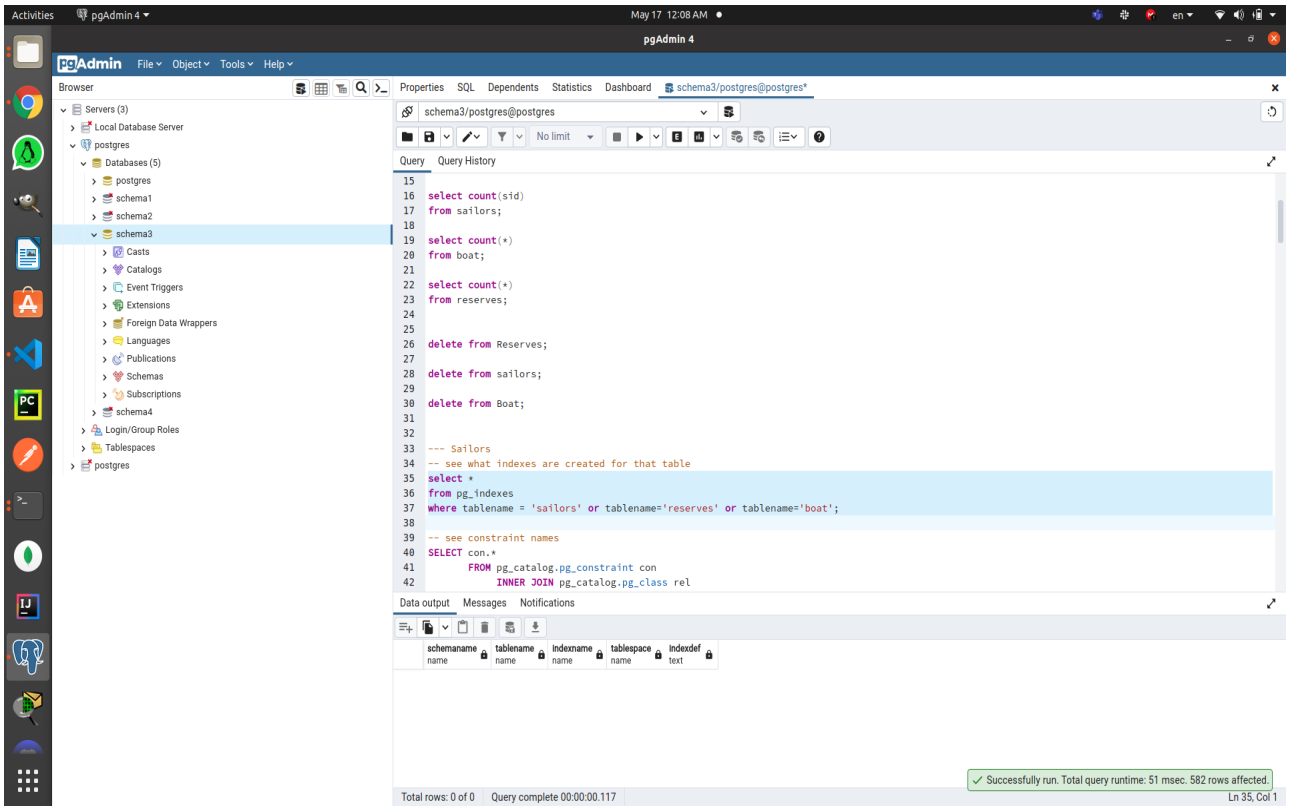
Optimized Query

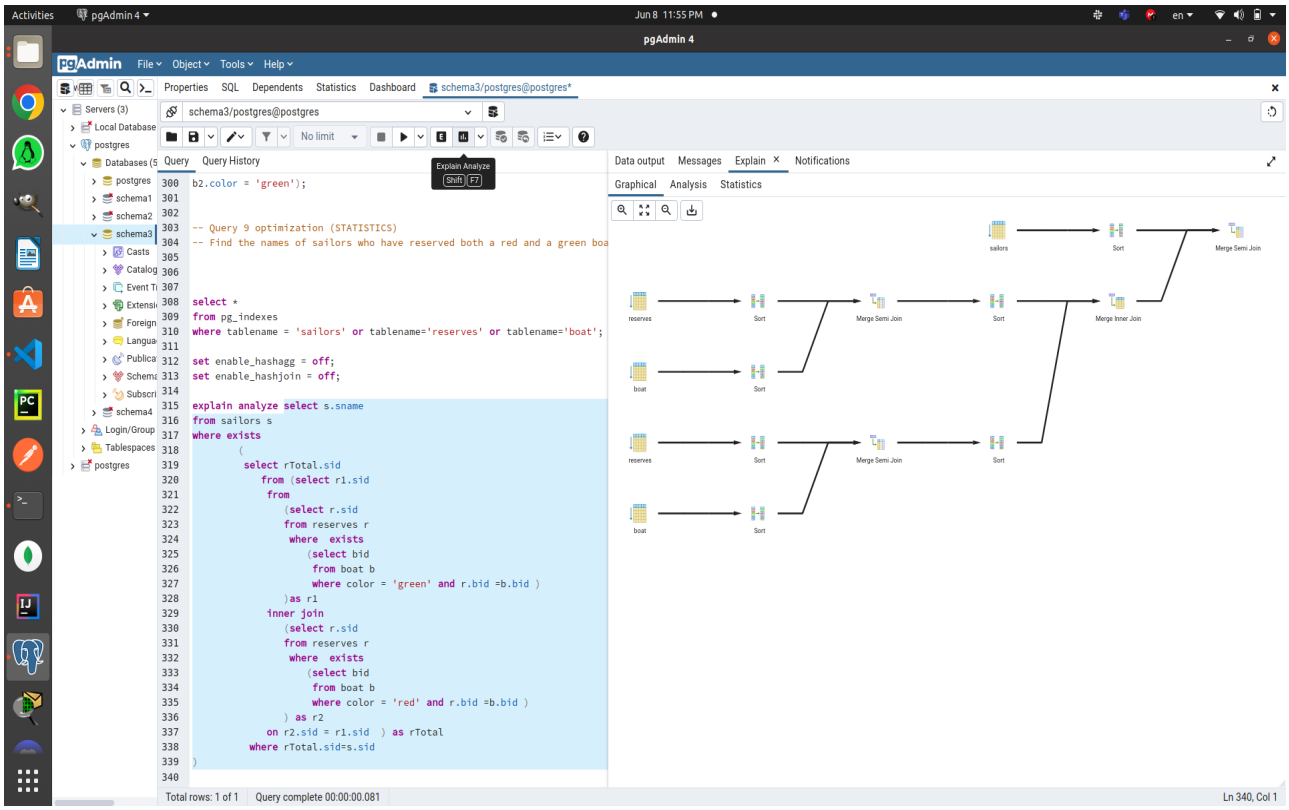
```
select s.sname
from sailors s
where exists
        (
         select rTotal.sid
            from (select r1.sid
             from
```

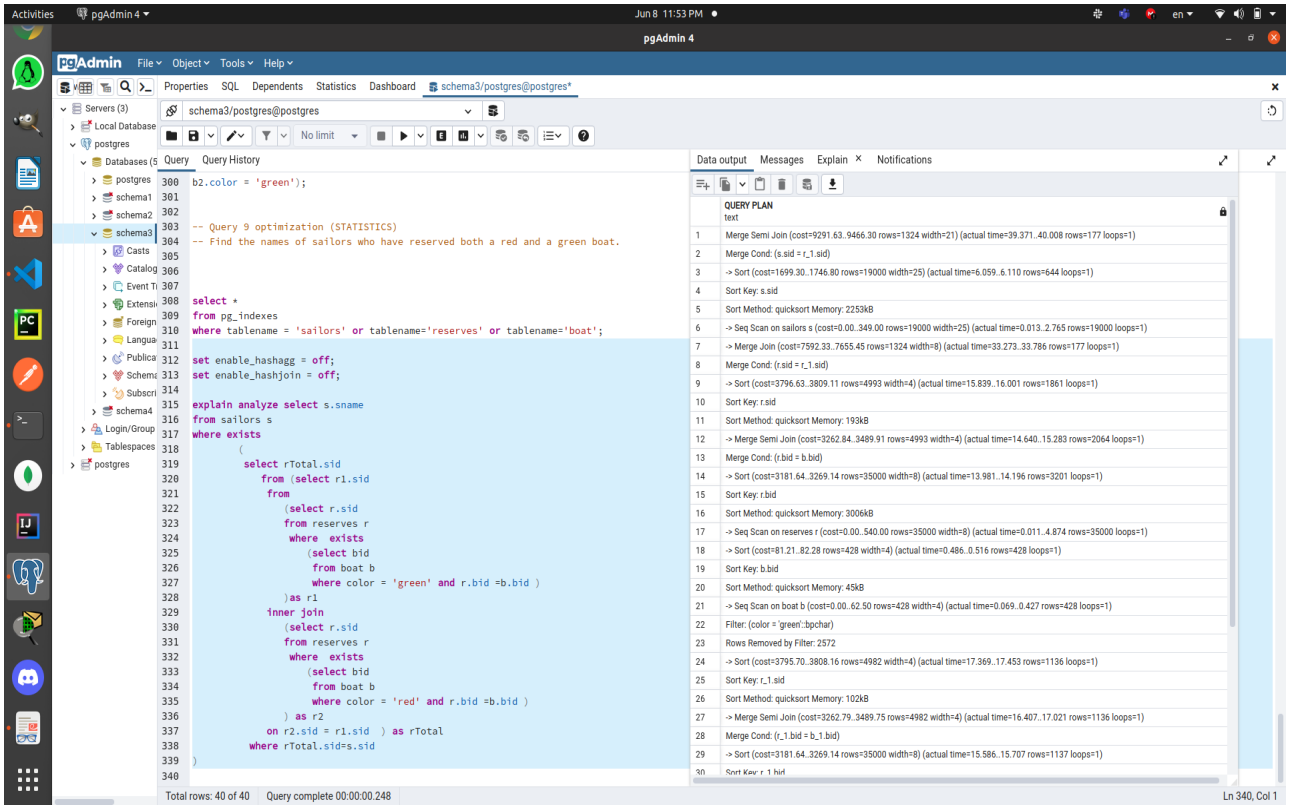
```
                (select r.sid
                from reserves r
                 where  exists
                    (select bid
                     from boat b
                     where color = 'green' and r.bid =b.bid )
                )as r1
            inner join
                (select r.sid
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

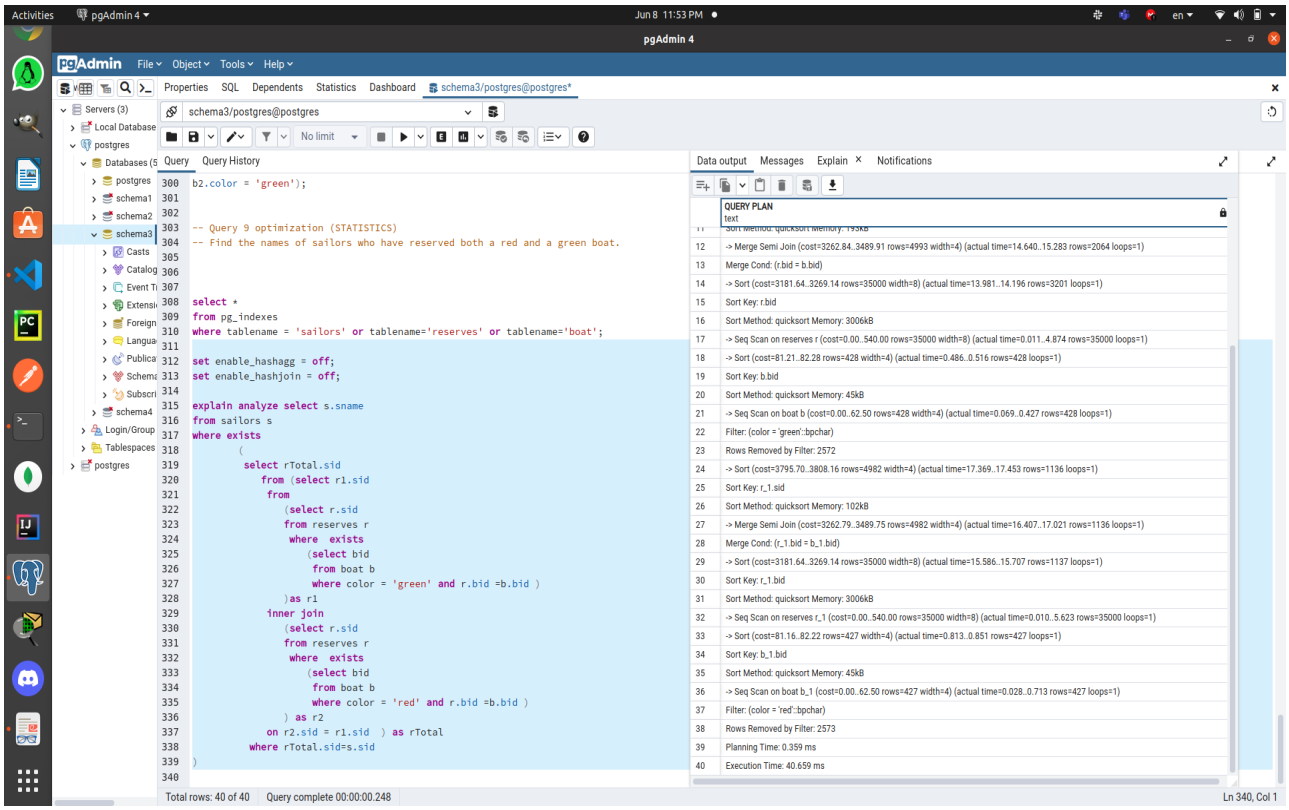
Report

1. given query without an index,




Activities
pgAdmin 4
Jun 8 11:55 PM
pgAdmin 4
File Object Tools Help
Properties SQL Dependents Statistics Dashboard schema3/postgres@postgres*
schema3/postgres@postgres
Explain Analyze
Shift F7
Query Query History
Data output Messages Explain Notifications
Graphical Analysis Statistics
sailors
Sort
Merge Semi Join
reserves
Merge Inner Join
boat
Total rows: 1 of 1
Query complete 00:00:00.081
Ln 340, Col 1


Activities
pgAdmin 4
Jun 8 11:53 PM
pgAdmin 4
File Object Tools Help
Properties SQL Dependents Statistics Dashboard schema3/postgres@postgres*
schema3/postgres@postgres
Query Query History
Data output Messages Explain Notifications
QUERY PLAN
text
Total rows: 40 of 40
Query complete 00:00:00.248
Ln 340, Col 1



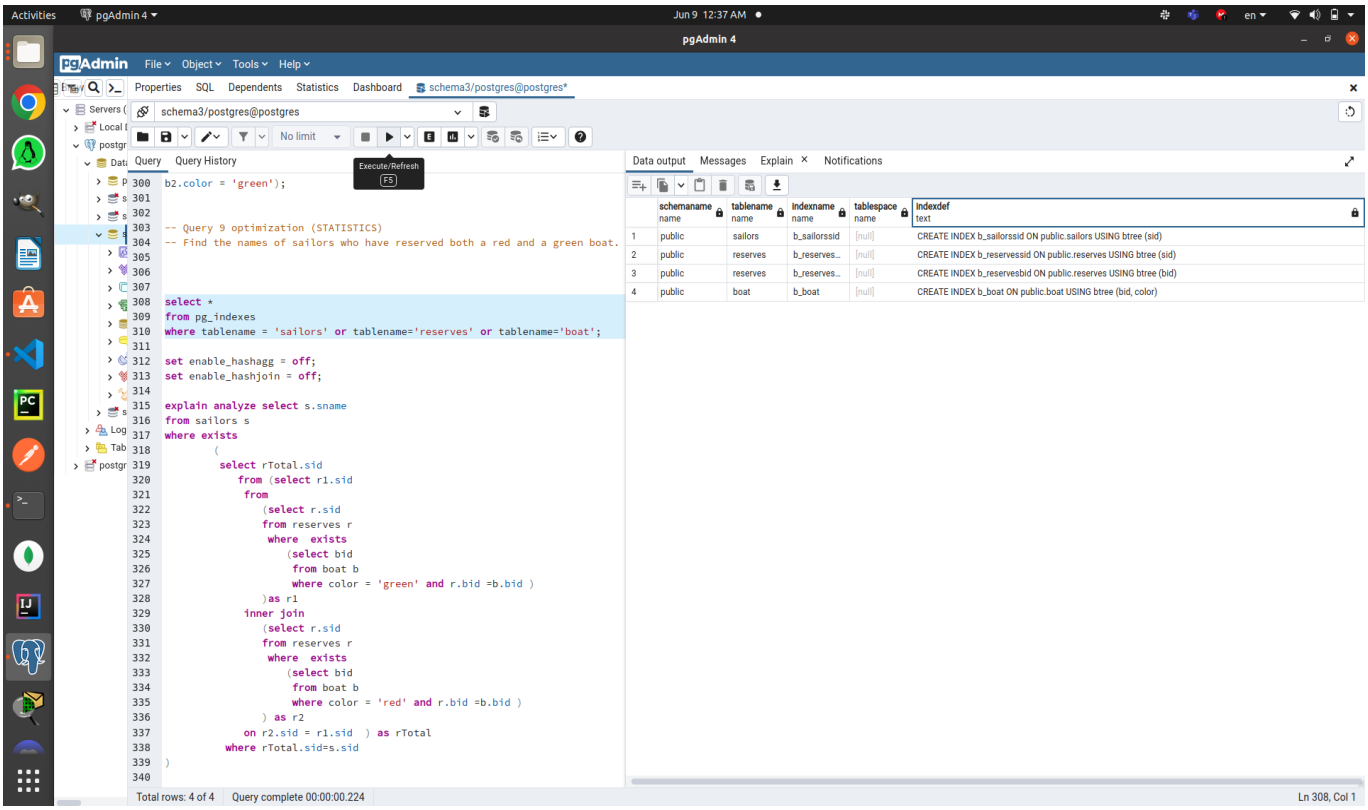
Explanation :

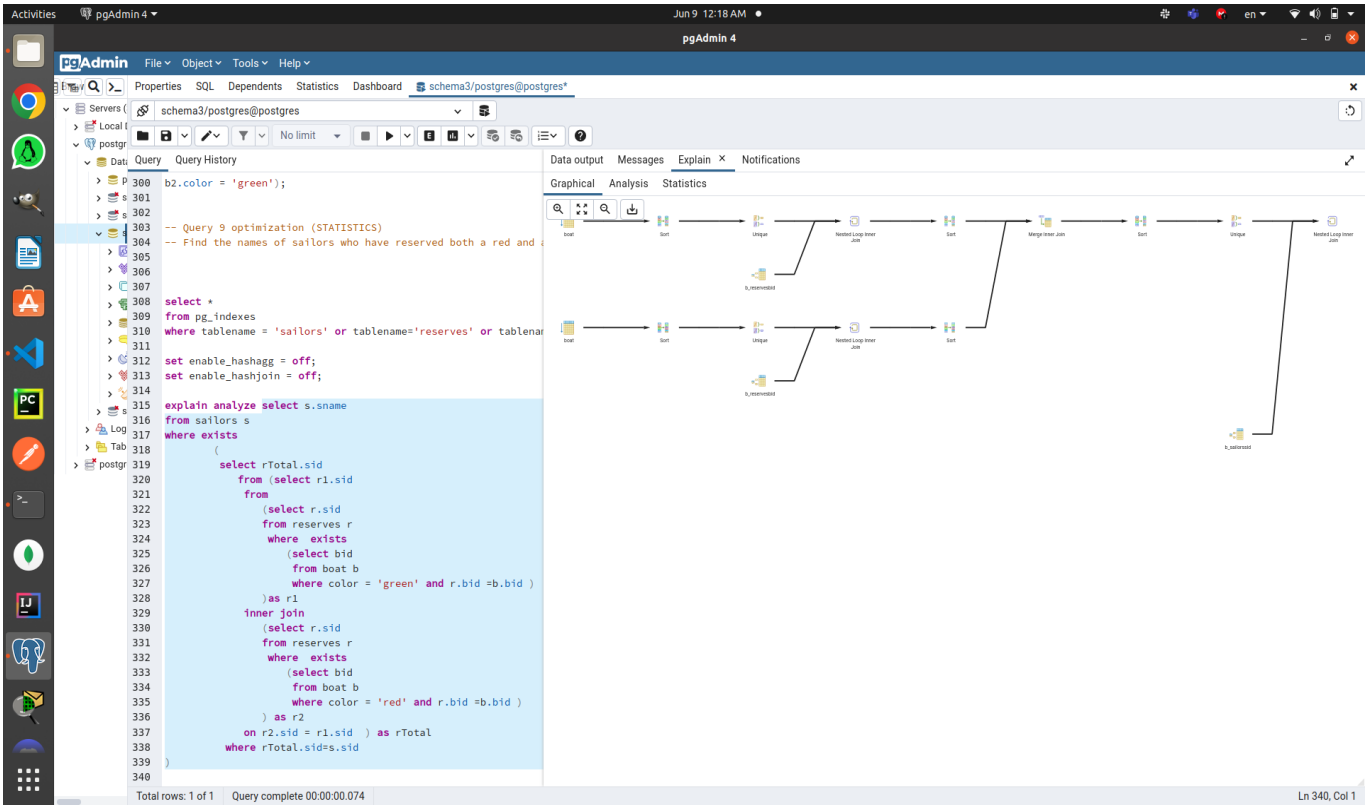
- Metrics :

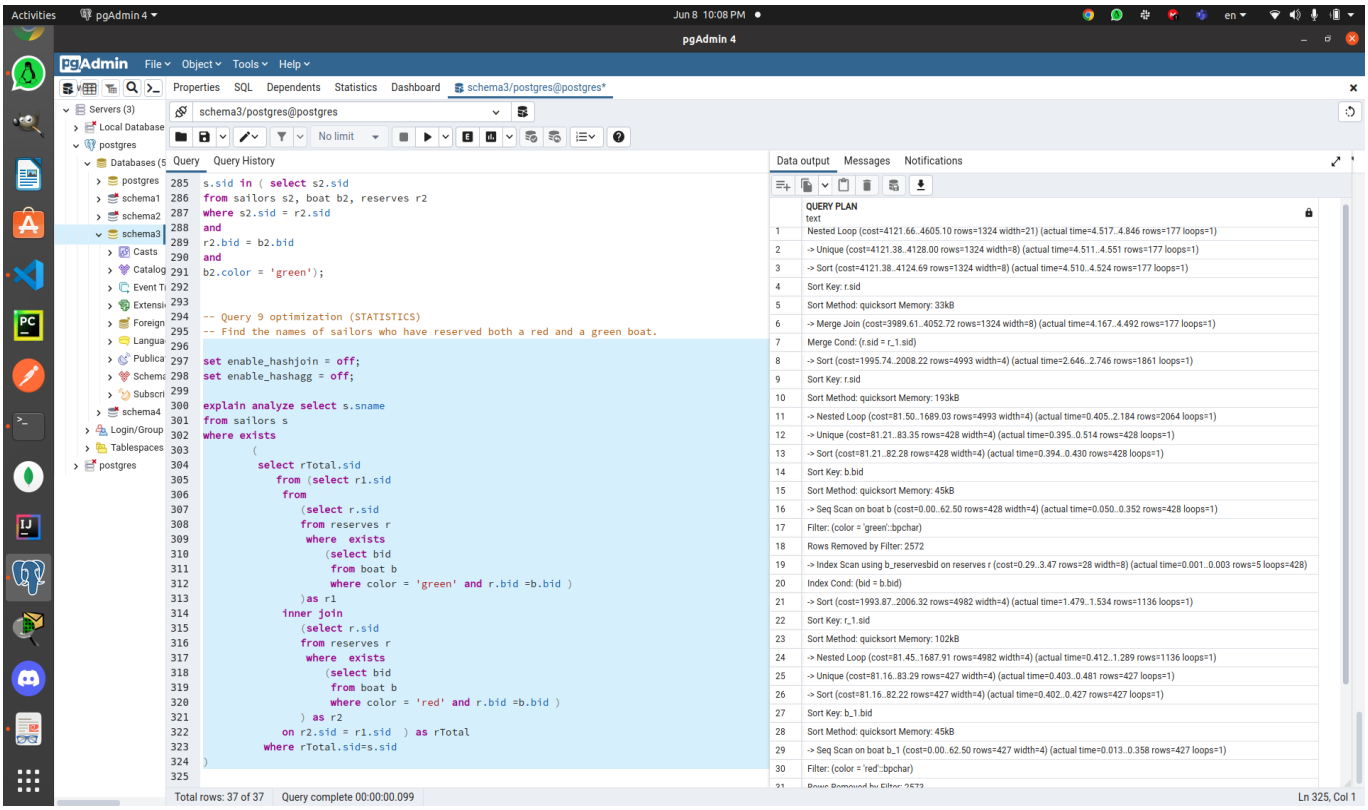
```
| Execution Time : 40.659 ms | Total Expected Cost : 9466.30 |
|----------------------------|-------------------------------|
```

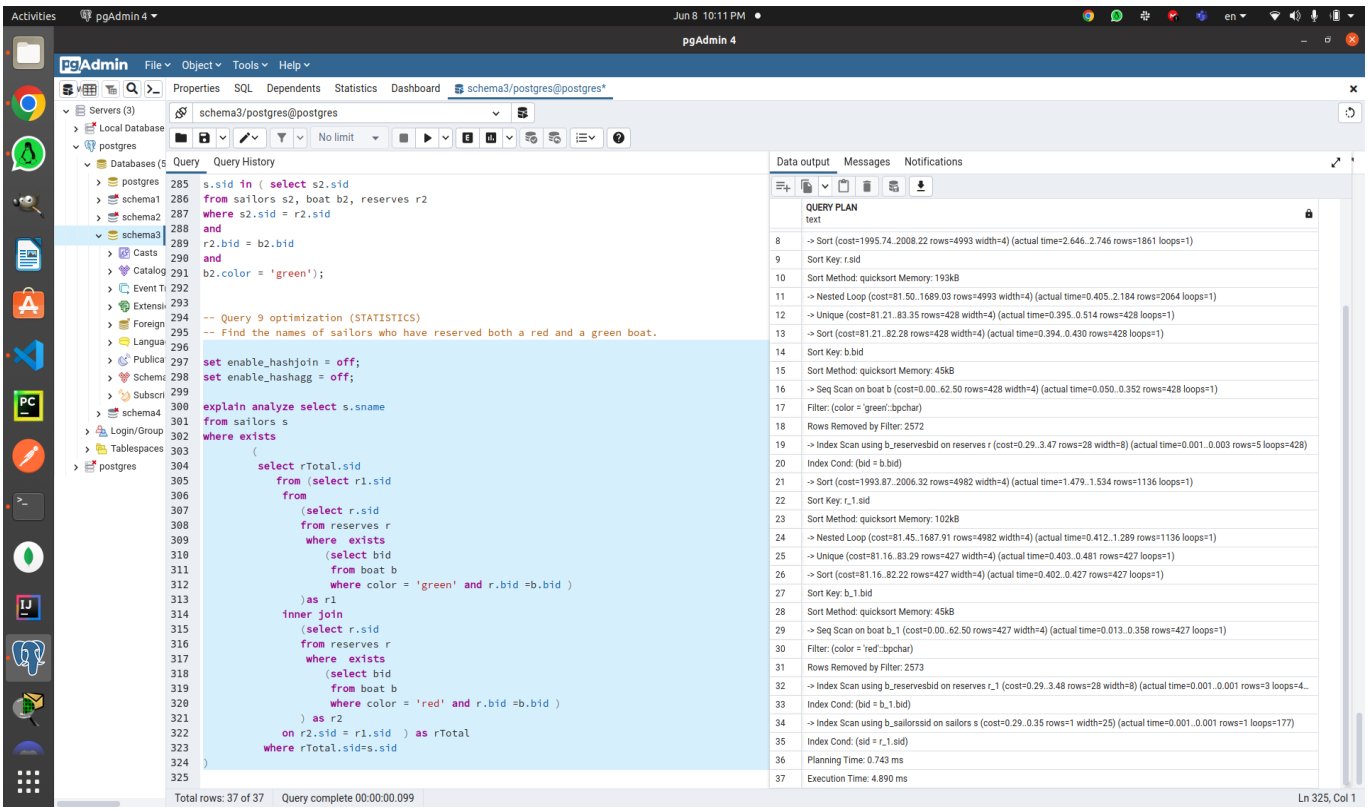
- Same flags is set to all here too.
- Reason :
    - This Query Improved in the Execution time and Expected Cost than the Original Query from 11368.59 to 9466.30.
    - Because the loops ends faster and exits due to I used the Exist Operator instead of the In Operator so it helped in the intermediate 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,


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 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_reservesbid [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
Data output Messages Explain Notifications
Graphical Analysis Statistics
Total rows: 1 of 1 Query complete 00:00:00.074
Ln 340, Col 1





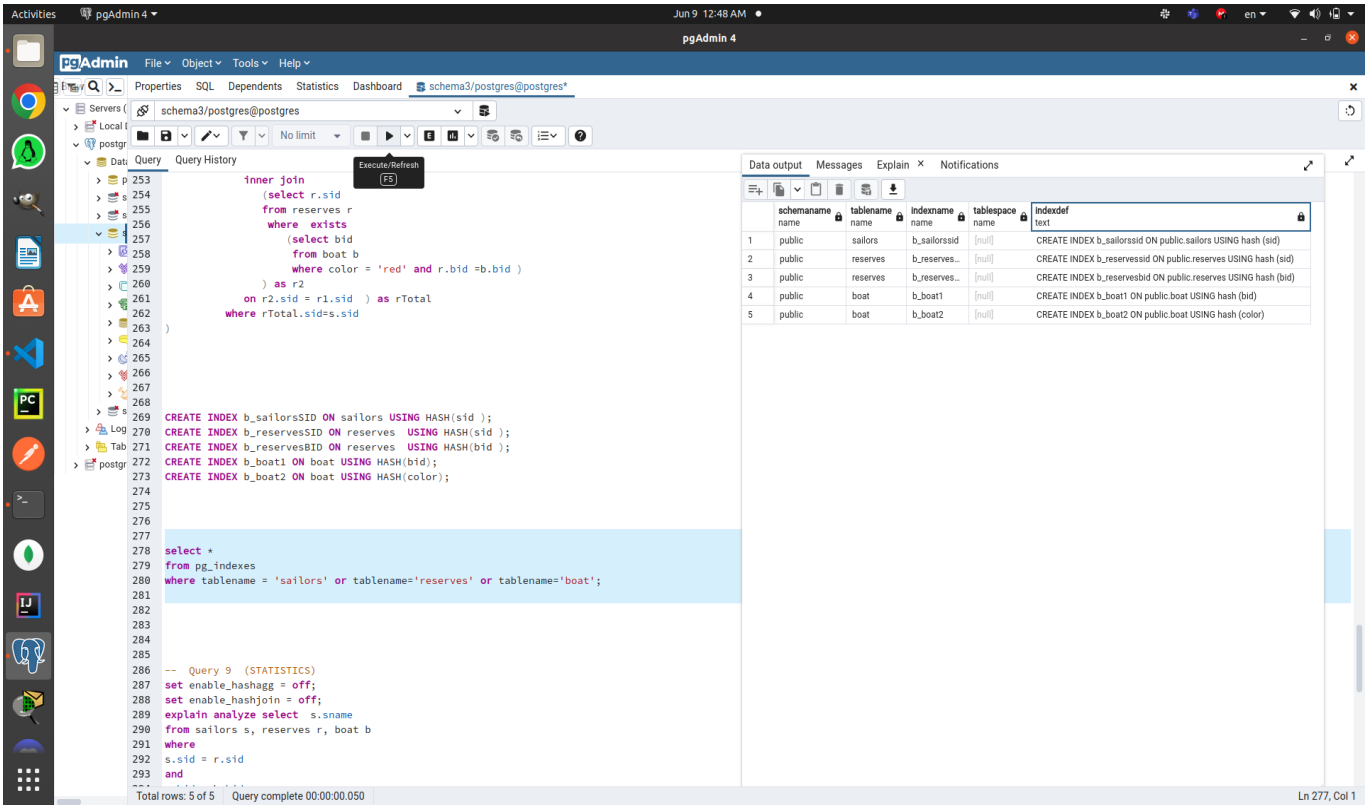
Explanation :

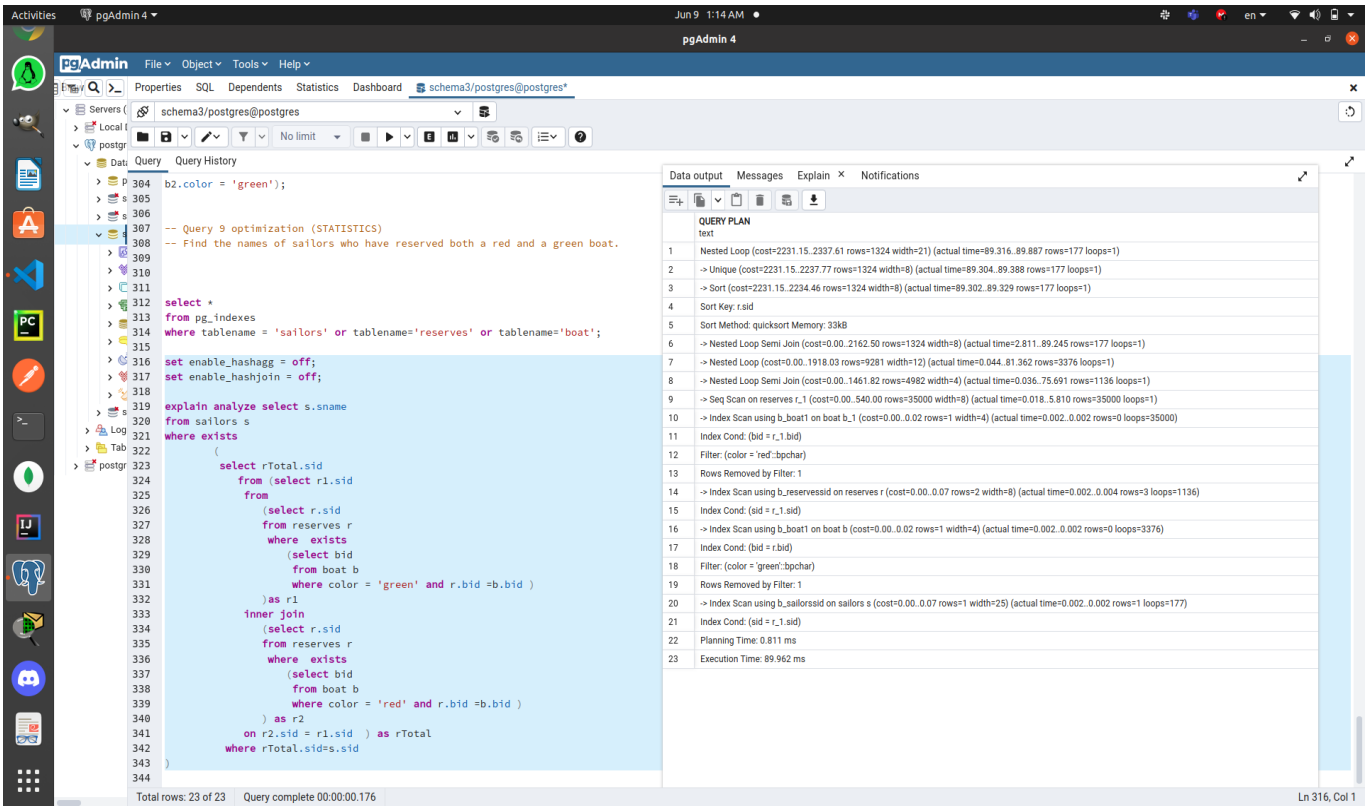
- Metrics :

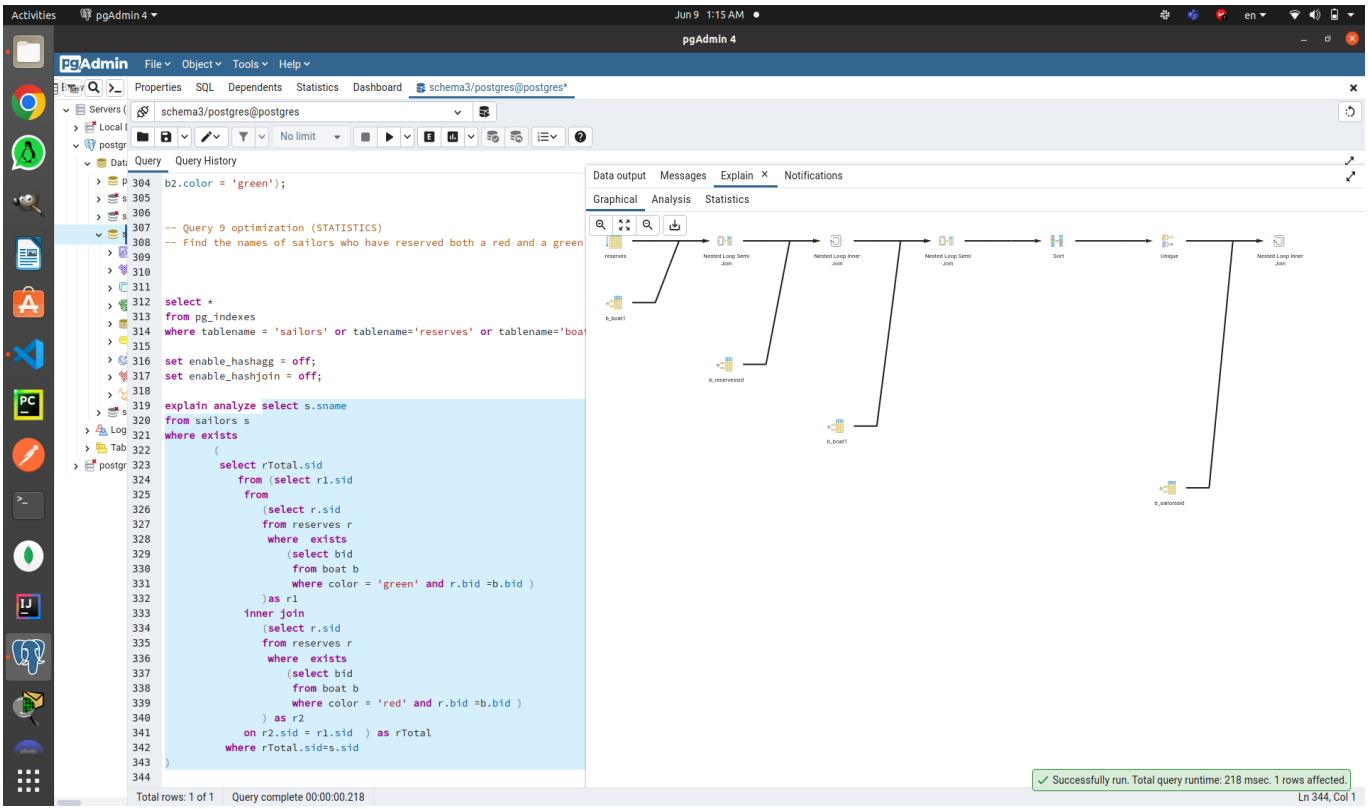
```
| Execution Time : 4.890 ms | Total Expected Cost : 4605.10|
|---------------------------|------------------------------|
```

- 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,





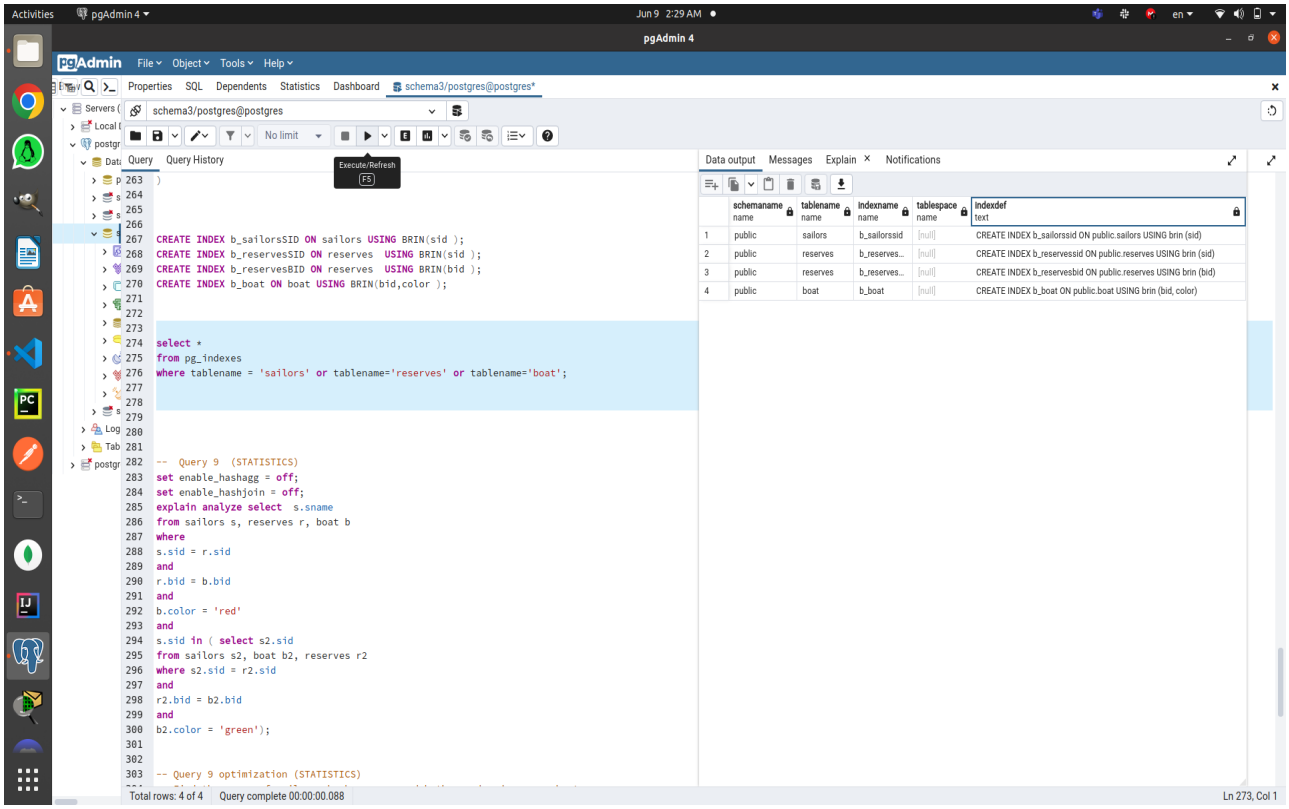


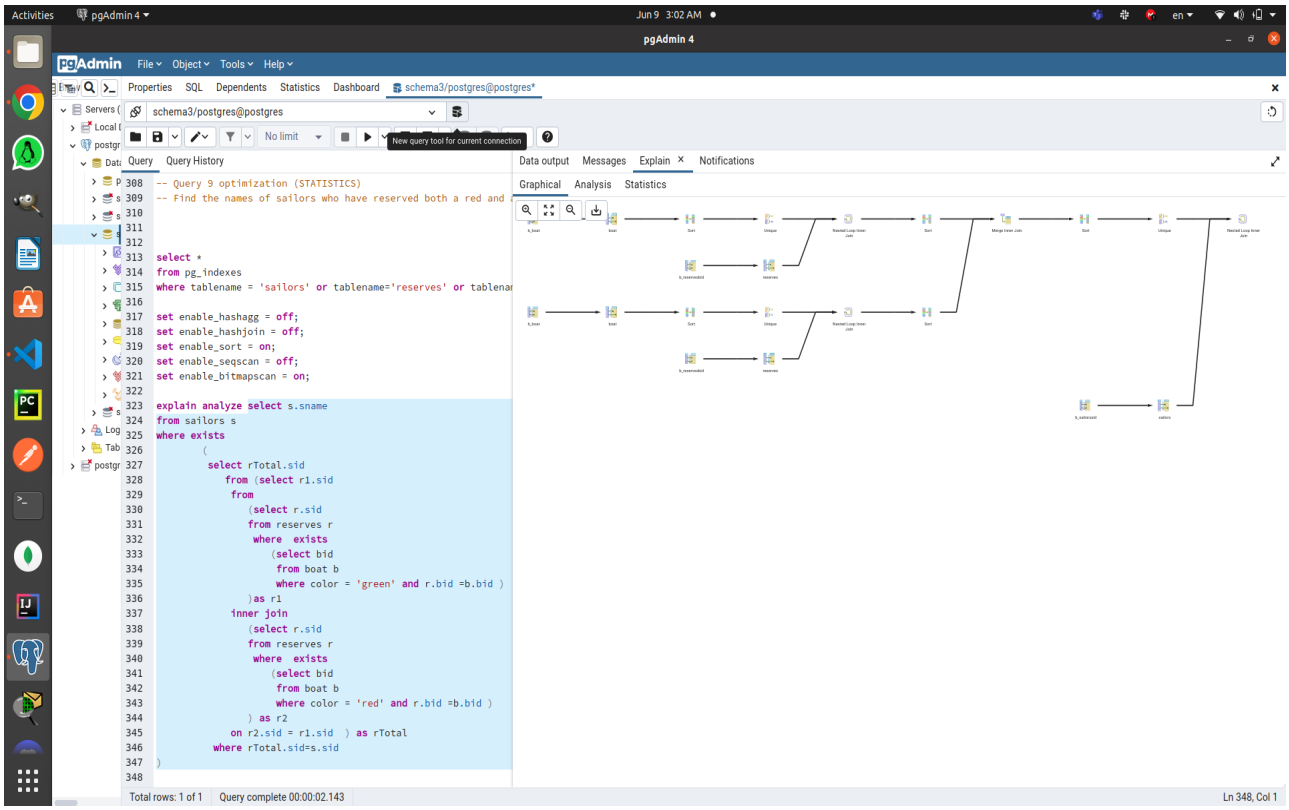
- Metrics :

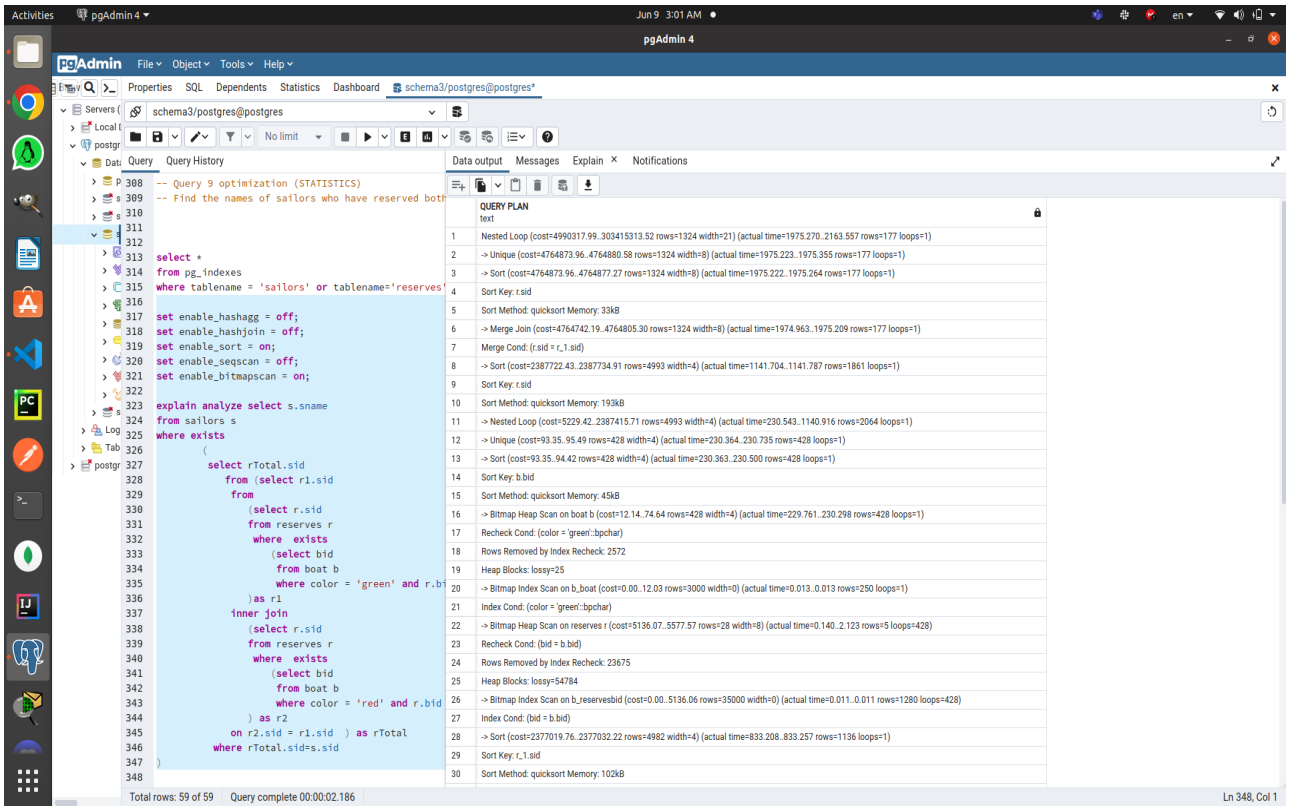
```
| Execution Time : 12.646 ms | Total Expected Cost : 2337.61|
|----------------------------|------------------------------|
```

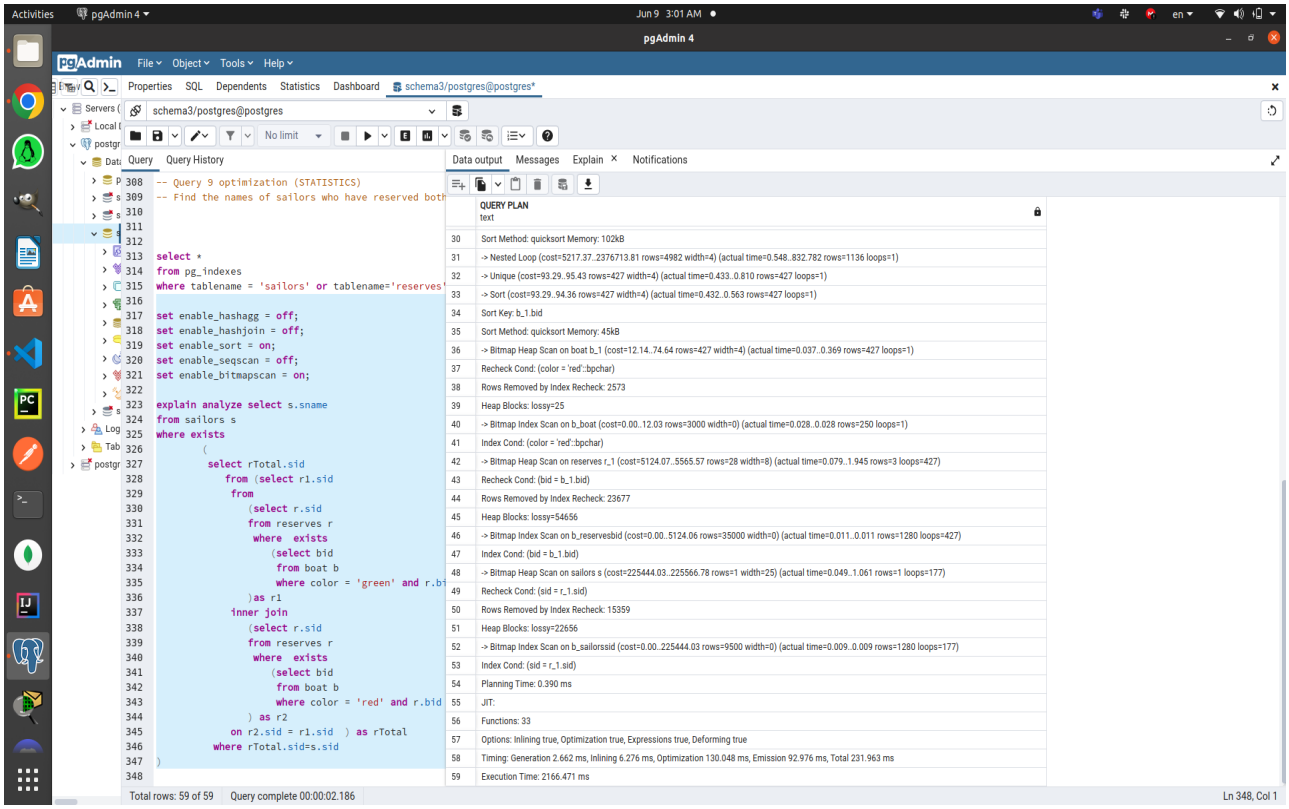
- The Hash helped in the performance it decreased the Execution Time(but in the execution time processor was overwhelmed + Linux performance) and Expected Cost .
- The Query Planner used the Hash index because Hash is O(1) performance with Exact Values and considered the best for exact values queries.
- 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,









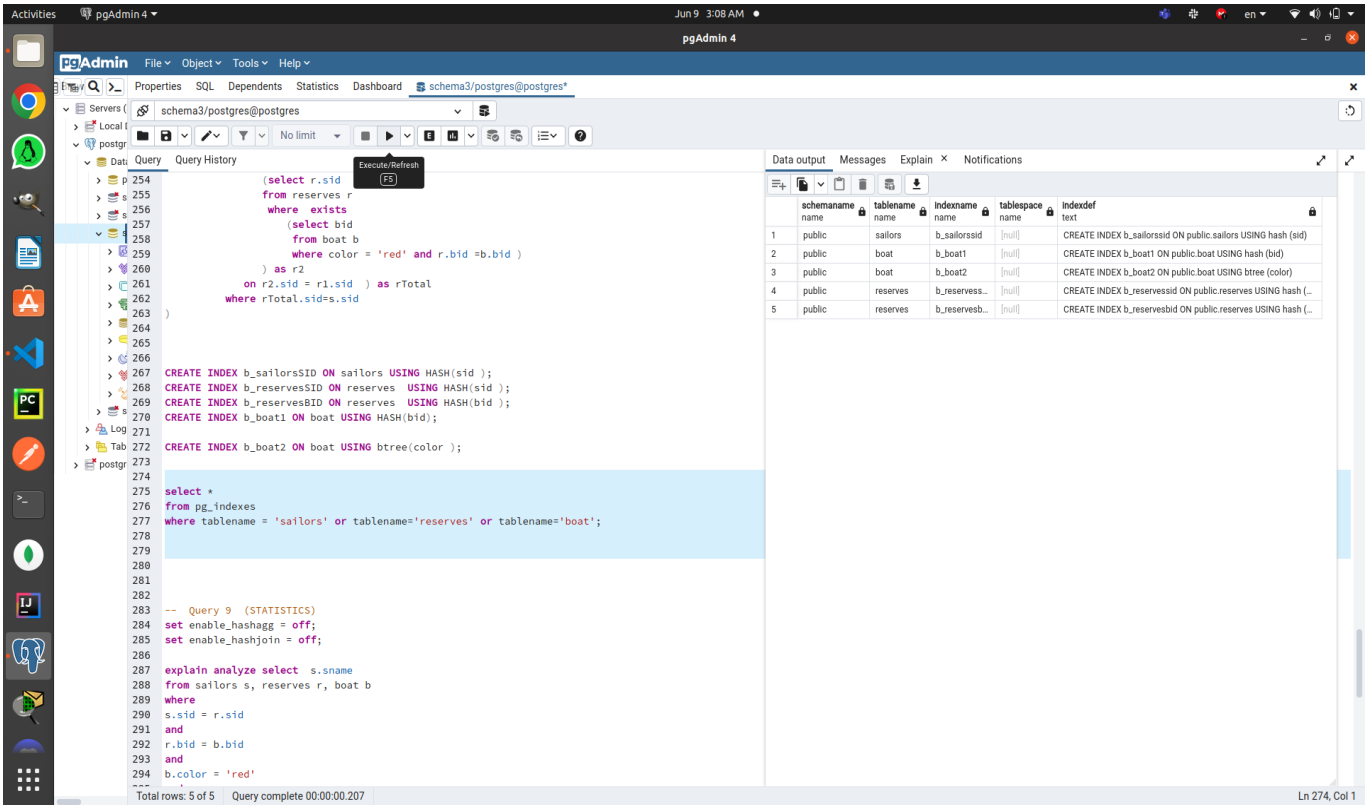
Explanation :

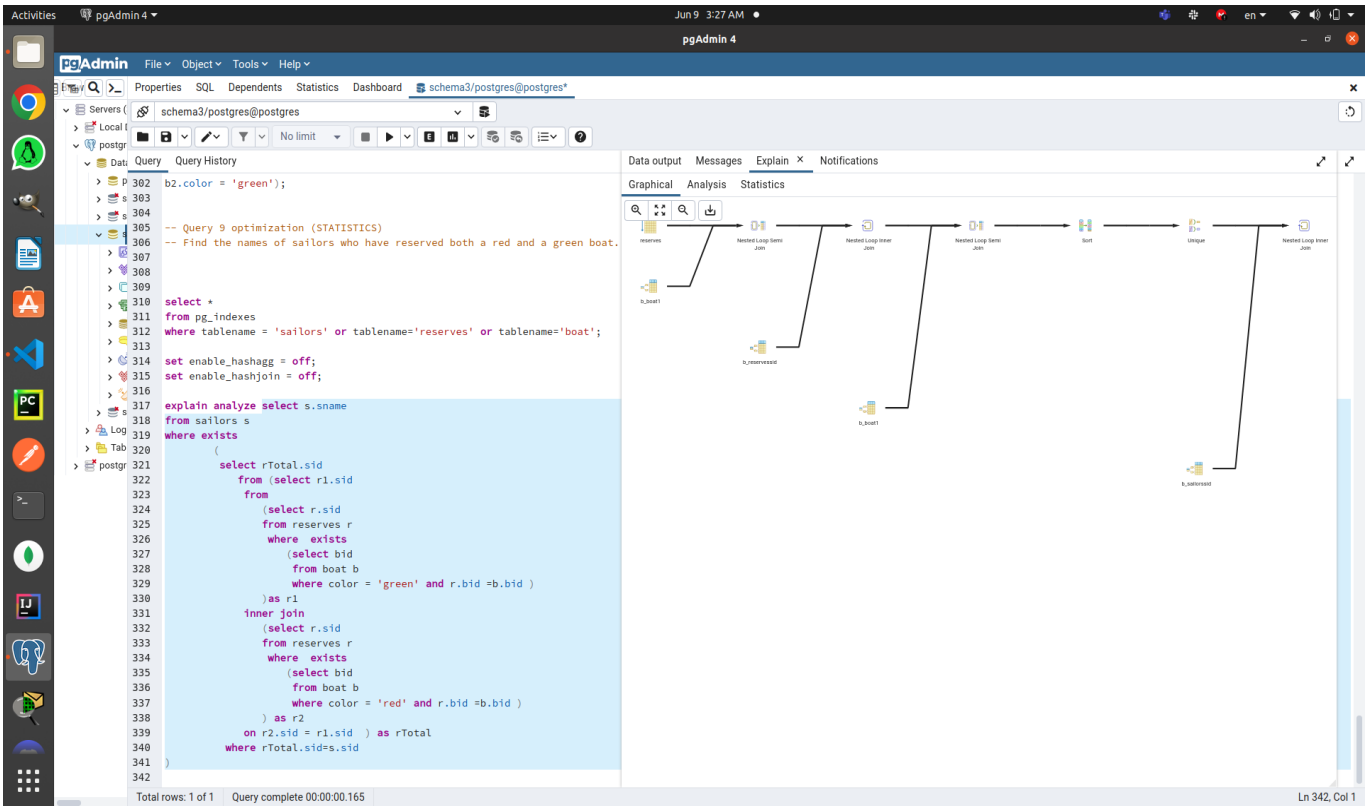
- Metrics :

```
| 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)





 mix  mix

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 .