Query 8

- Find the names of sailors who haue reserved a red boat .

Note !

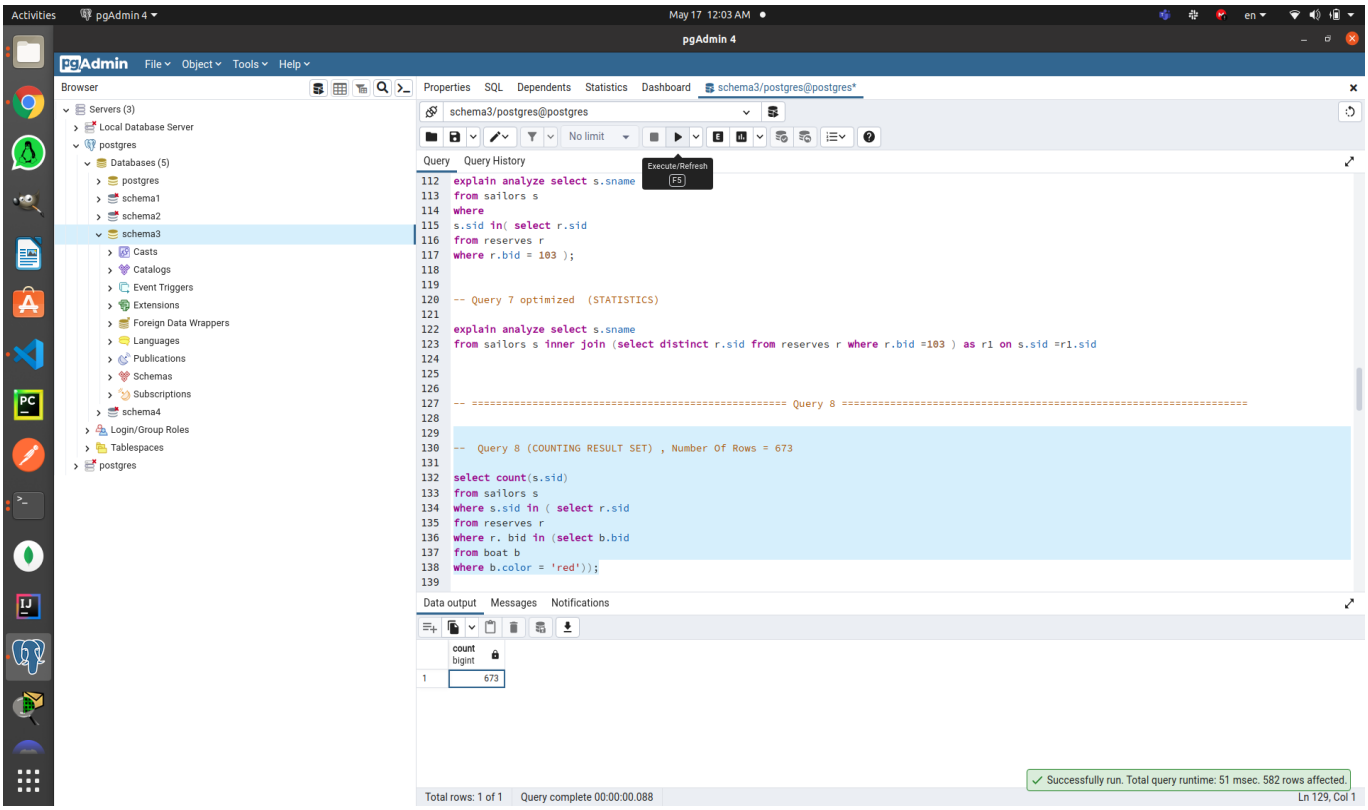
- Flags Hashjoin and HashAgg here where disabled for future after many trials and errors , I've discovered the best way to show the difference interms of the cost and to beat the Postgres Query Optimizer Algorithm to be able to show indices effect and cost differenes .
- The Execution time was changing 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 .
- I removed the primary constraint to be able to remove the already built in B-tree index to simulate no-inices behavior (I asked Prof.).

Original Query

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

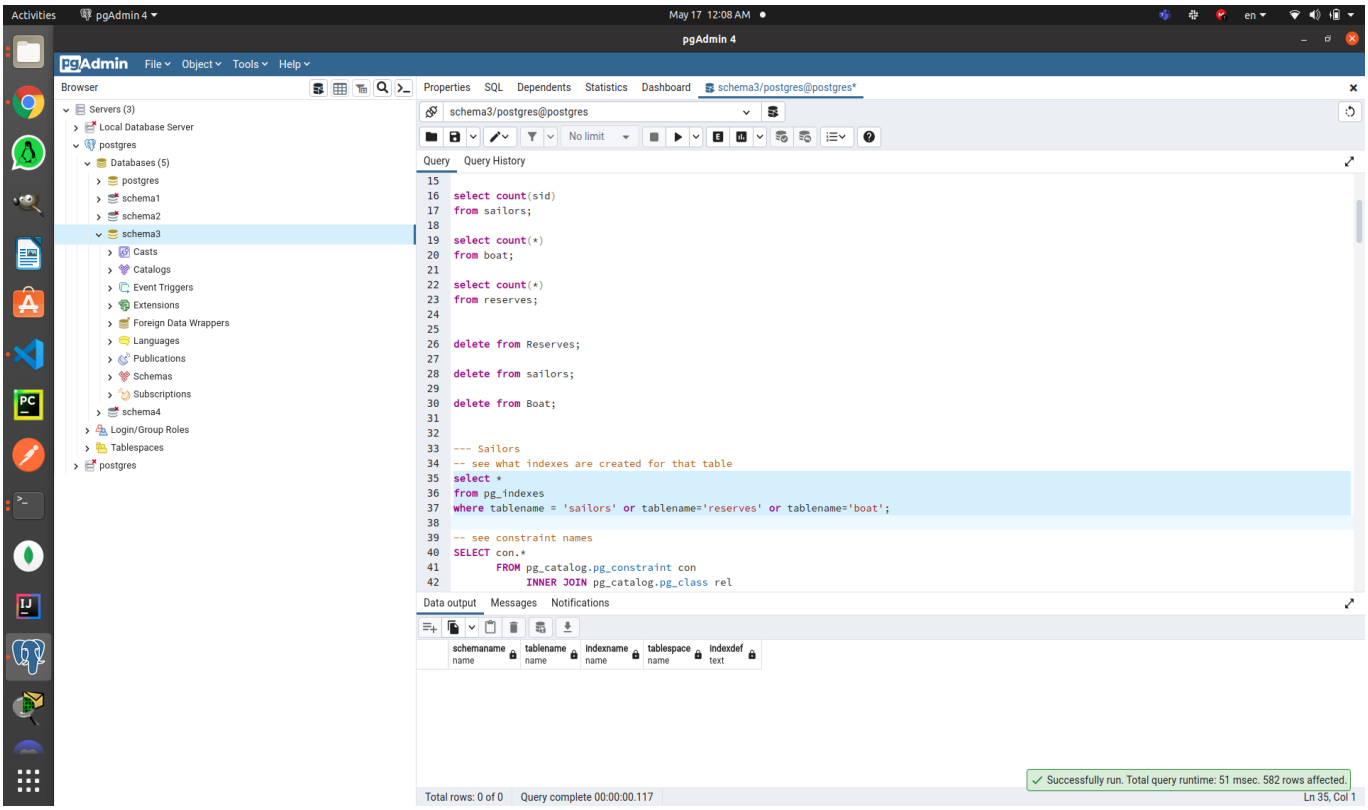
Result Set

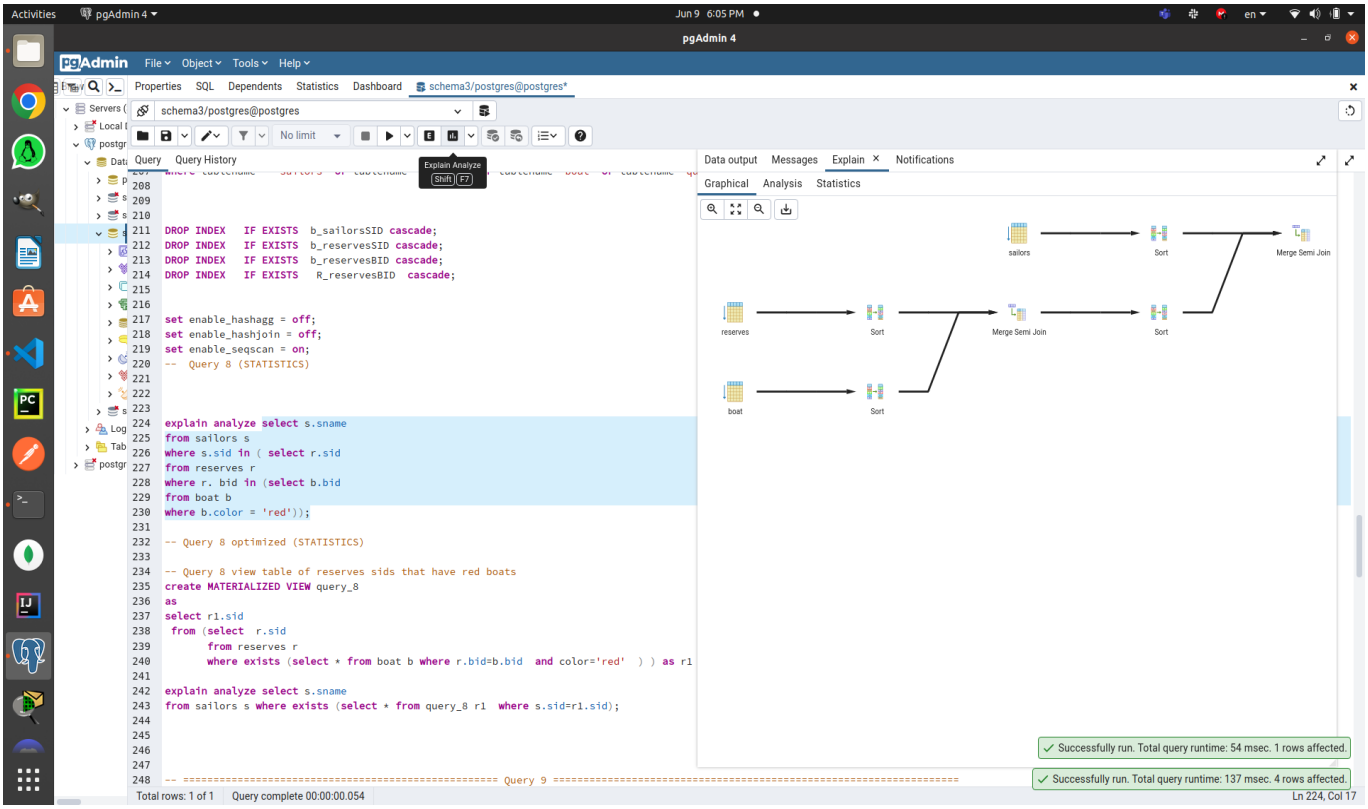
- 673 Rows

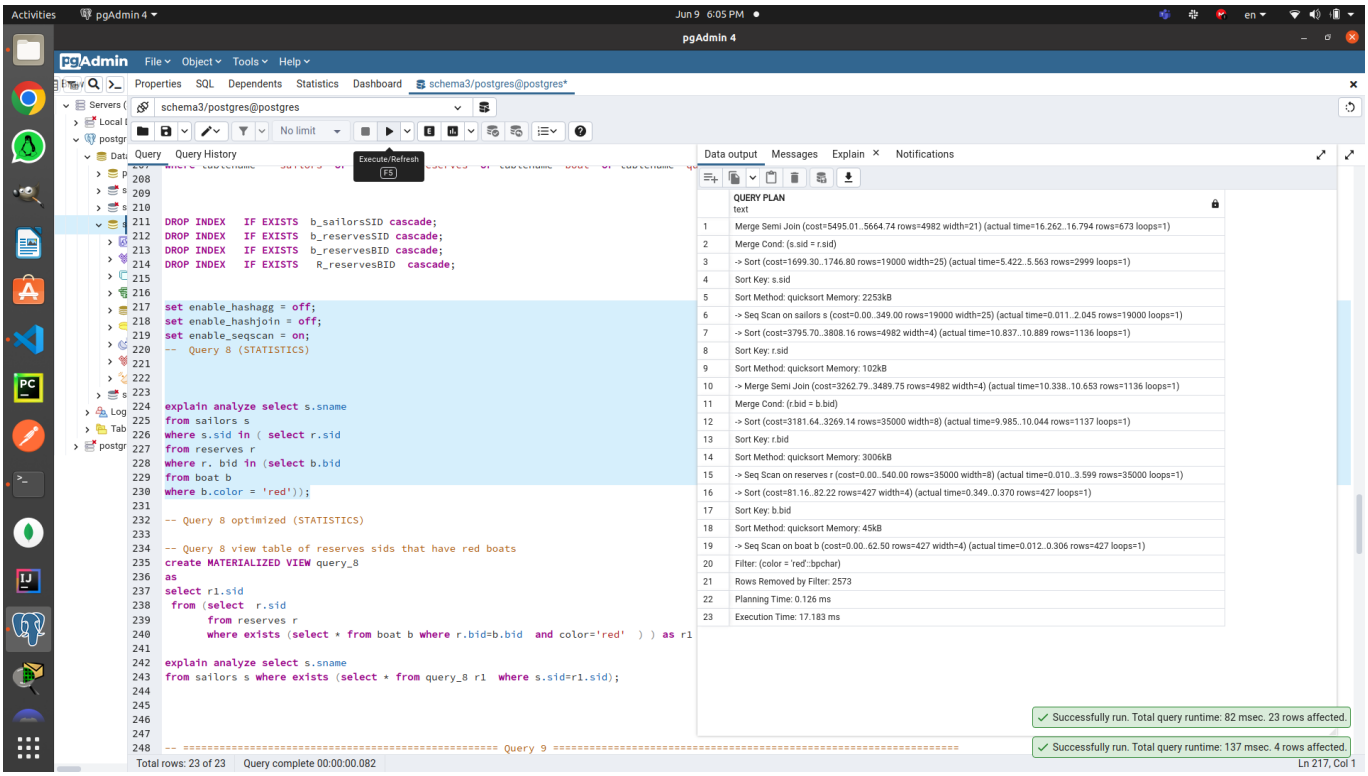


Report

1. given query without an index :


Activities
pgAdmin 4
May 17 12:08 AM
pgAdmin 4
File Object Tools Help
Browser
Servers (3)
Local Database Server
postgres
Databases (5)
postgres
schema1
schema2
schema3
Casts
Catalogs
Event Triggers
Extensions
Foreign Data Wrappers
Languages
Publications
Schemas
Subscriptions
schema4
Login/Group Roles
Tablespaces
postgres
Properties SQL Dependents Statistics Dashboard schema3/postgres@postgres*
schema3/postgres@postgres
No limit
Query Query History
15
16 select count(sid)
17 from sailors;
18
19 select count(*)
20 from boat;
21
22 select count(*)
23 from reserves;
24
25
26 delete from Reserves;
27
28 delete from sailors;
29
30 delete from Boat;
31
32
33 --- Sailors
34 -- see what indexes are created for that table
35 select *
36 from pg_indexes
37 where tablename = 'sailors' or tablename='reserves' or tablename='boat';
38
39 -- see constraint names
40 SELECT con.*
41 FROM pg_catalog.pg_constraint con
42 INNER JOIN pg_catalog.pg_class rel
Data output Messages Notifications
schemaname name
tablename name
indexname name
tablespace name
indexdef 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 9 6:05 PM
pgAdmin 4
File Object Tools Help
Properties SQL Dependents Statistics Dashboard schema3/postgres@postgres*
schema3/postgres@postgres
No limit
Query Query History
Explain Analyze Shift F7
208
209
210
211 DROP INDEX IF EXISTS b_sailorsSID cascade;
212 DROP INDEX IF EXISTS b_reservesSID cascade;
213 DROP INDEX IF EXISTS b_reservesBID cascade;
214 DROP INDEX IF EXISTS R_reservesBID cascade;
215
216
217 set enable_hashagg = off;
218 set enable_hashjoin = off;
219 set enable_seqscan = on;
220 -- Query 8 (STATISTICS)
221
222
223
224 explain analyze select s.sname
225 from sailors s
226 where s.sid in (select r.sid
227 from reserves r
228 where r. bid in (select b.bid
229 from boat b
230 where b.color = 'red'));
231
232 -- Query 8 optimized (STATISTICS)
233
234 -- Query 8 view table of reserves sids that have red boats
235 create MATERIALIZED VIEW query_8
236 as
237 select r1.sid
238 from (select r.sid
239 from reserves r
240 where exists (select * from boat b where r.bid=b.bid and color='red')) as r1
241
242 explain analyze select s.sname
243 from sailors s where exists (select * from query_8 r1 where s.sid=r1.sid);
244
245
246
247
248 -- ========== Query 9 ==========
Data output Messages Explain Notifications
Graphical Analysis Statistics
sailors
Sort
Merge Semi Join
reserves
Sort
Merge Semi Join
Sort
boat
Sort
Successfully run. Total query runtime: 54 msec. 1 rows affected.
Successfully run. Total query runtime: 137 msec. 4 rows affected.
Total rows: 1 of 1
Query complete 00:00:00.054
Ln 224, Col 17



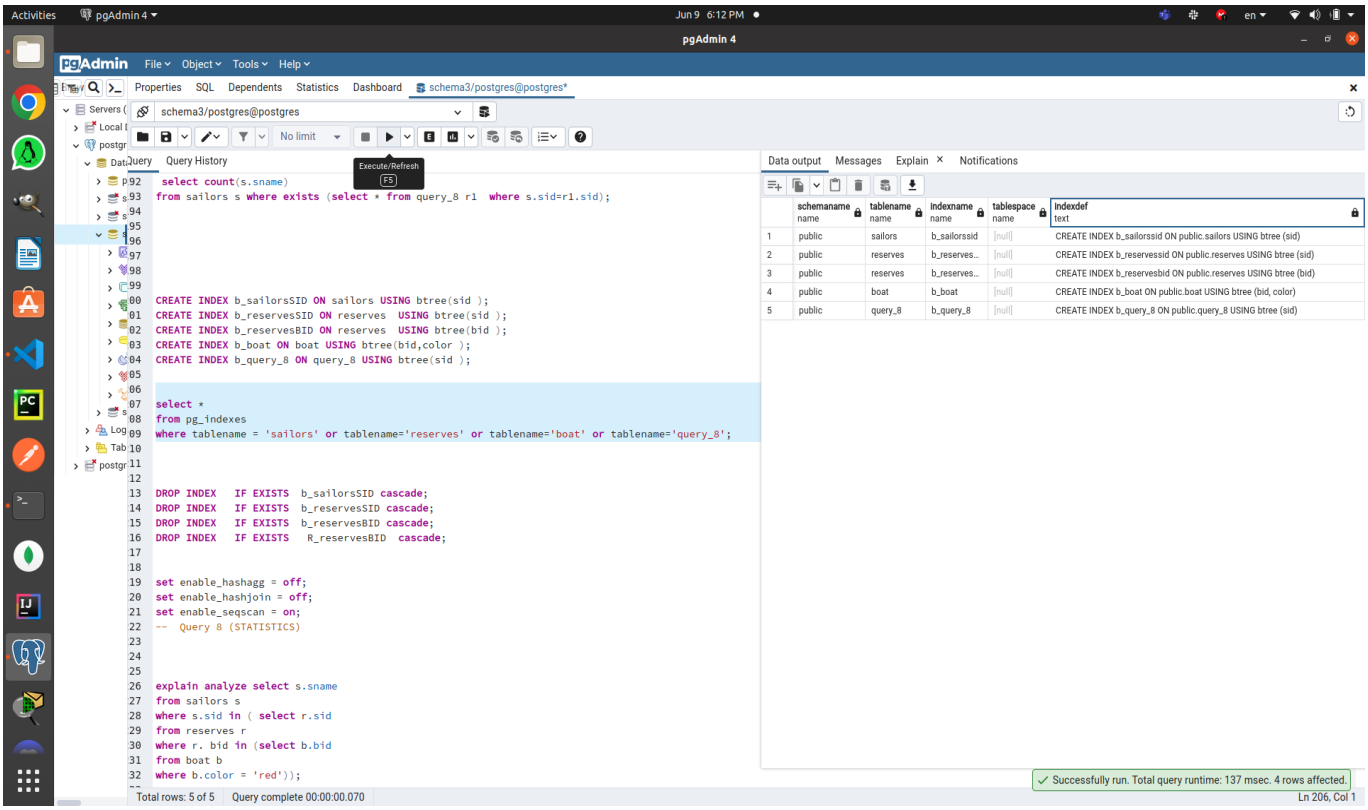
Explanation :

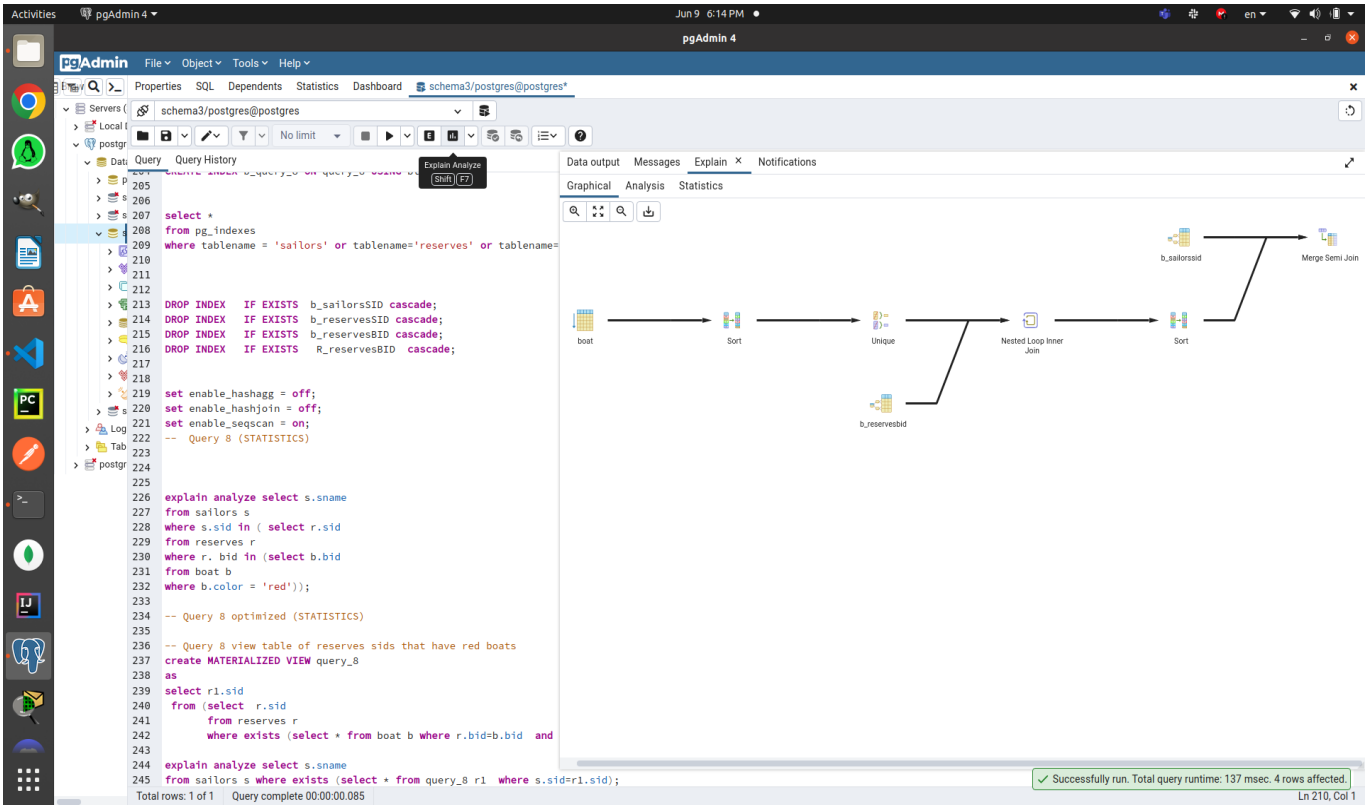
- Metrics :

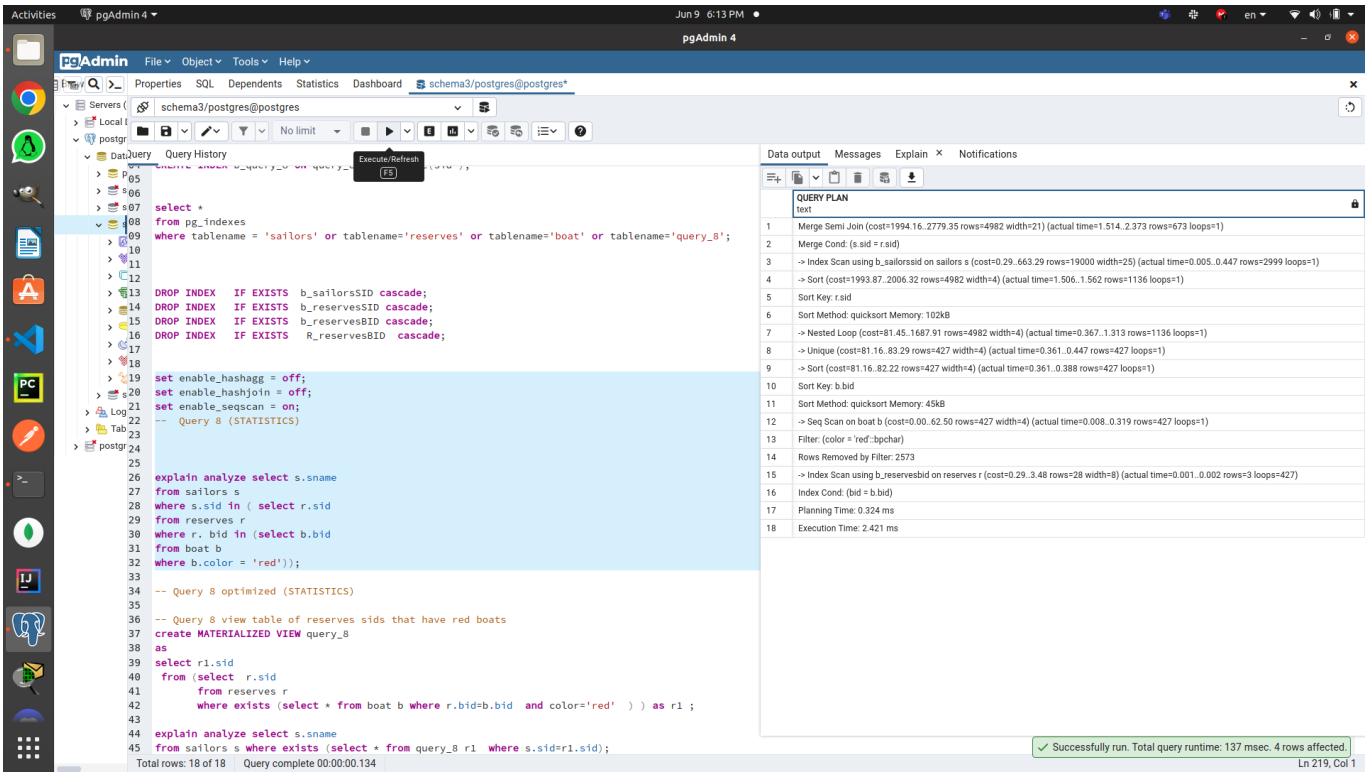
  Execution Time : 17.183 ms Total Expected Cost : 5664.74

- Here I removed all indices and the performance is not that bad due to Query optimizer and the sizee of the data is in thousands .

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
select count(s.sname)
from sailors s where exists (select * from query_8 r1 where s.sid=r1.sid);
CREATE INDEX b_sailorsSID ON sailors USING btree(sid);
CREATE INDEX b_reservesSID ON reserves USING btree(sid);
CREATE INDEX b_reservesBID ON reserves USING btree(bid);
CREATE INDEX b_boat ON boat USING btree(bid,color);
CREATE INDEX b_query_8 ON query_8 USING btree(sid);
select *
from pg_indexes
where tablename = 'sailors' or tablename='reserves' or tablename='boat' or tablename='query_8';
DROP INDEX IF EXISTS b_sailorsSID cascade;
DROP INDEX IF EXISTS b_reservesSID cascade;
DROP INDEX IF EXISTS b_reservesBID cascade;
DROP INDEX IF EXISTS R_reservesBID cascade;
set enable_hashagg = off;
set enable_hashjoin = off;
set enable_seqscan = on;
-- Query 8 (STATISTICS)
explain analyze select s.sname
from sailors s
where s.sid in (select r.sid
from reserves r
where r. bid in (select b.bid
from boat b
where b.color = 'red'));
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)
5 public query_8 b_query_8 [null] CREATE INDEX b_query_8 ON public.query_8 USING btree (sid)
Successfully run. Total query runtime: 137 msec. 4 rows affected.
Total rows: 5 of 5 Query complete 00:00:00.070
Ln 206, Col 1


pgAdmin 4
Properties SQL Dependents Statistics Dashboard schema3/postgres@postgres*
schema3/postgres@postgres
Query Query History
Explain Analyze Shift F7
select *
from pg_indexes
where tablename = 'sailors' or tablename='reserves' or tablename=
DROP INDEX IF EXISTS b_sailorsSID cascade;
DROP INDEX IF EXISTS b_reservesSID cascade;
DROP INDEX IF EXISTS b_reservesBID cascade;
DROP INDEX IF EXISTS R_reservesBID cascade;
set enable_hashagg = off;
set enable_hashjoin = off;
set enable_seqscan = on;
-- Query 8 (STATISTICS)
explain analyze select s.sname
from sailors s
where s.sid in (select r.sid
from reserves r
where r. bid in (select b.bid
from boat b
where b.color = 'red'));
-- Query 8 optimized (STATISTICS)
-- Query 8 view table of reserves sids that have red boats
create MATERIALIZED VIEW query_8
as
select r1.sid
from (select r.sid
from reserves r
where exists (select * from boat b where r.bid=b.bid and
explain analyze select s.sname
from sailors s where exists (select * from query_8 r1 where s.sid=r1.sid);
Data output Messages Explain Notifications
Graphical Analysis Statistics
boat
Sort
Unique
b_reservesbid
Nested Loop Inner Join
Sort
b_sailorssid
Merge Semi Join
Successfully run. Total query runtime: 137 msec. 4 rows affected.
Total rows: 1 of 1 Query complete 00:00:00.085
Ln 210, Col 1



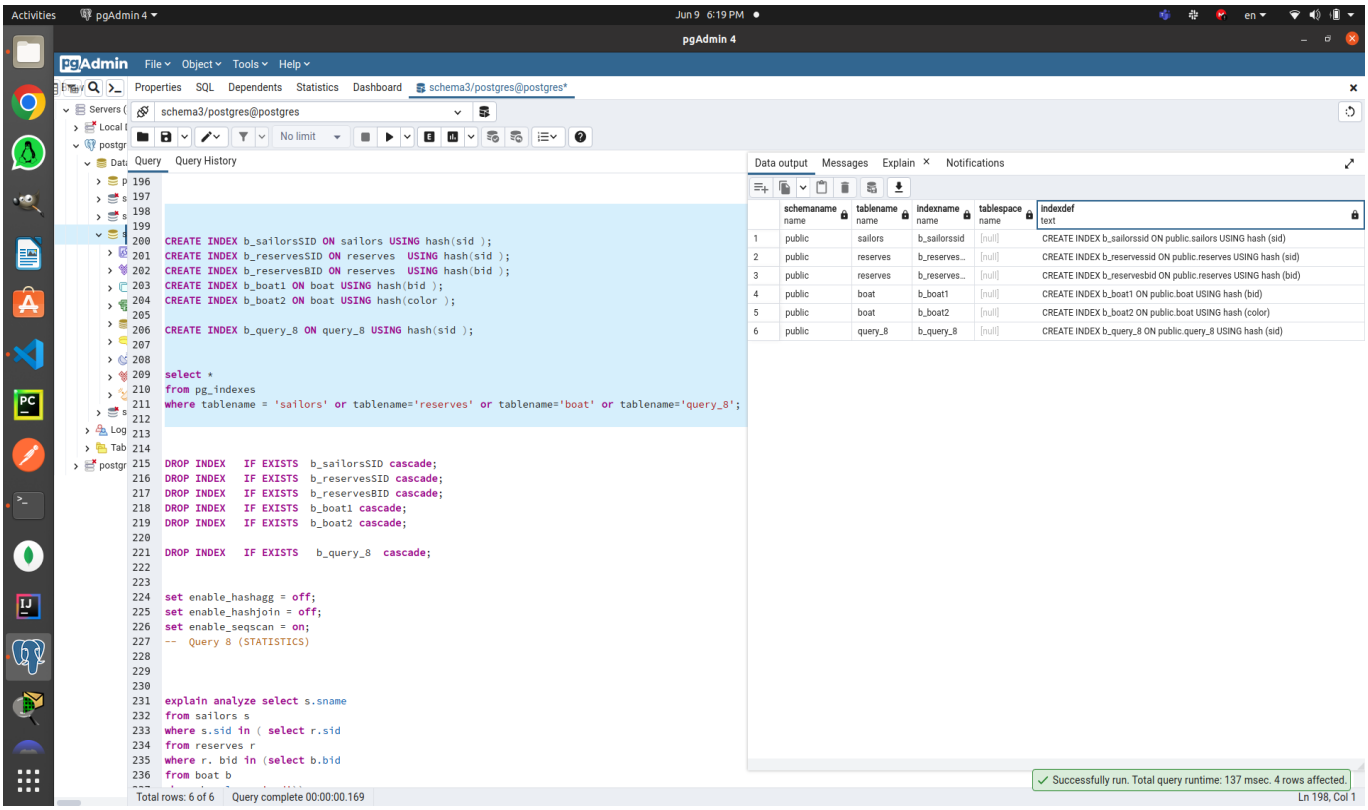
Explanation :

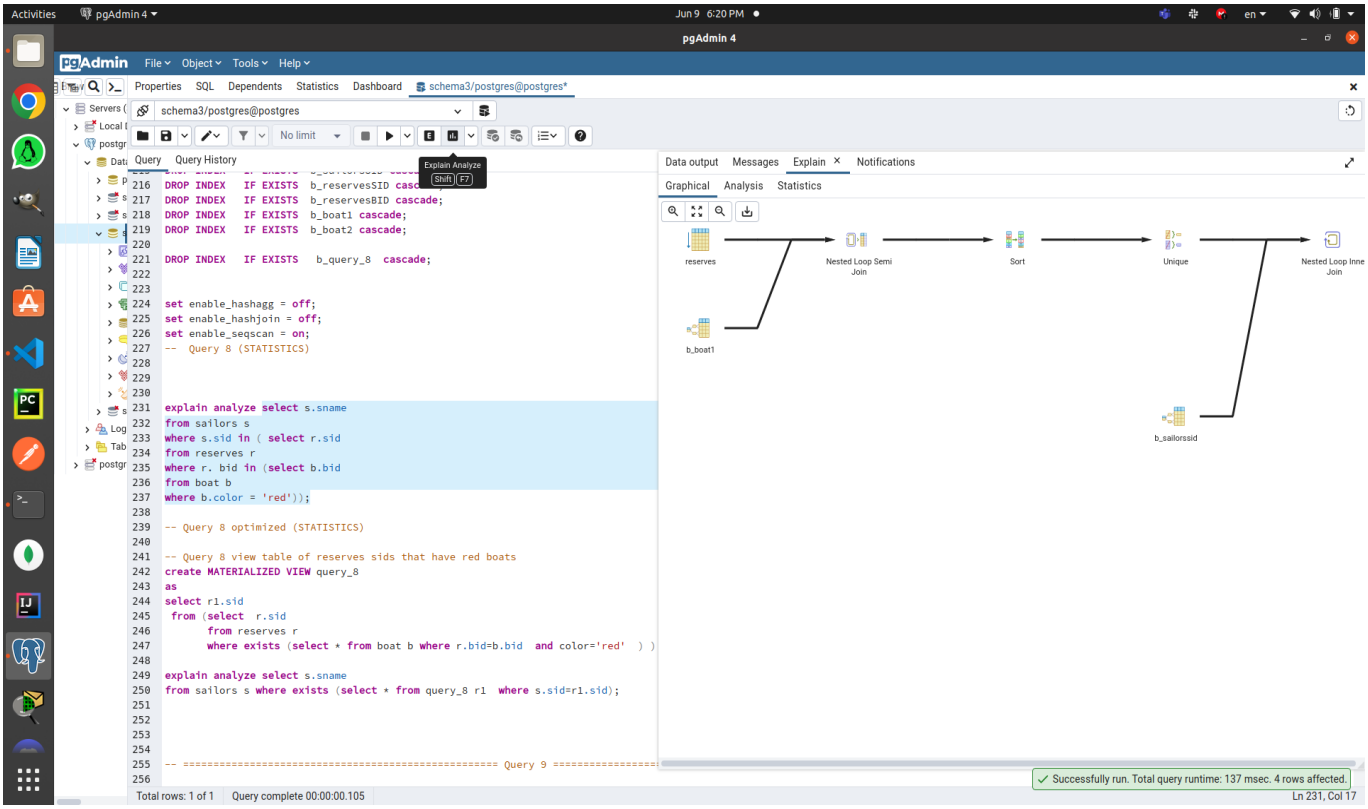
- Metrics :

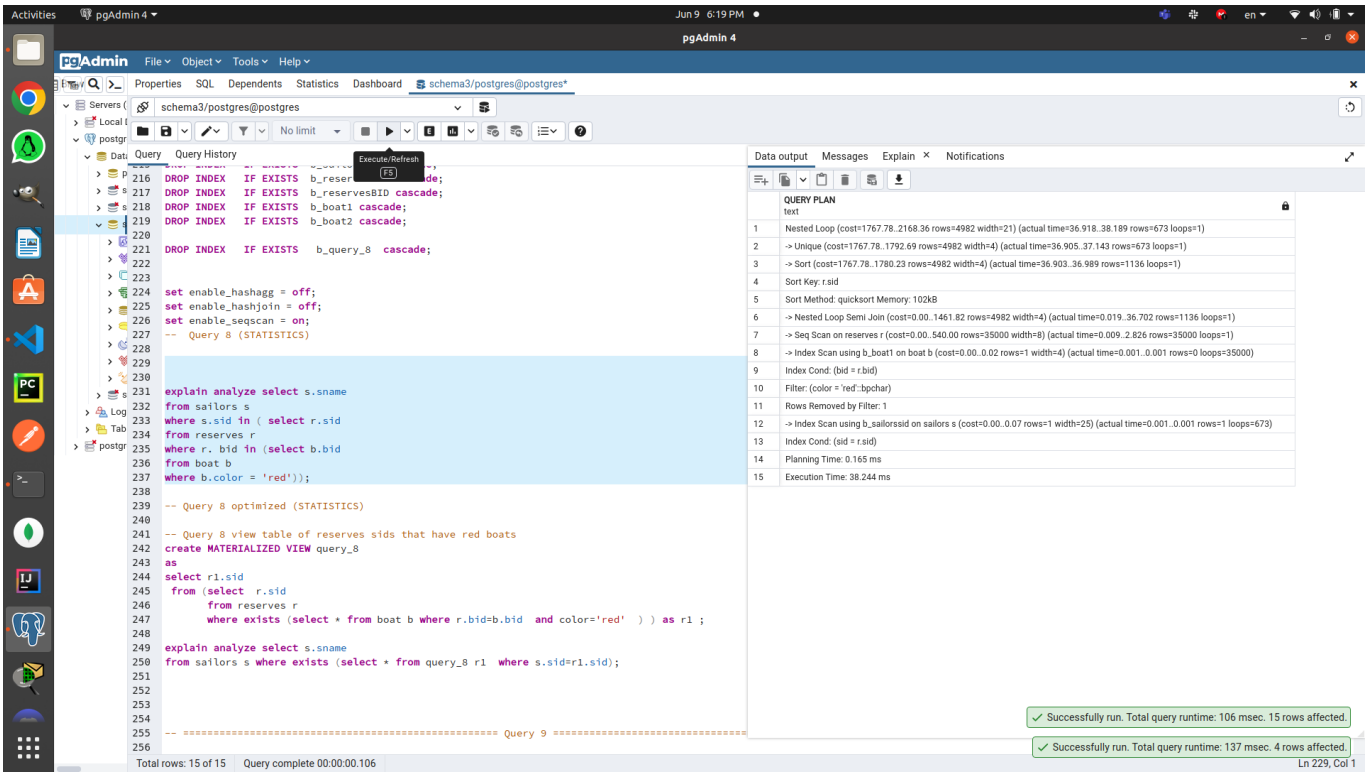
  Execution Time : 2.421 ms Total Expected Cost : 2779.35

- 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 for the attributes before the IN Operator cause the leaves are sorted and Query Planner sorted the subquery result to match to Merge Join on condition (b.bid=r.bid).
- The Query Planner used the B-tree index because B-Tree is O(Log n) performance for the attributes before the IN Operator cause the leaves are sorted and Query Planner sorted the subquery result to match to Merge Join on condition (s.sid=r.sid).

3. given query with hash indices only :


pgAdmin 4
schema3/postgres@postgres
CREATE INDEX b_sailorsSID ON sailors USING hash(sid);
CREATE INDEX b_reservesSID ON reserves USING hash(sid);
CREATE INDEX b_reservesBID ON reserves USING hash(bid);
CREATE INDEX b_boat1 ON boat USING hash(bid);
CREATE INDEX b_boat2 ON boat USING hash(color);
CREATE INDEX b_query_8 ON query_8 USING hash(sid);
select *
from pg_indexes
where tablename = 'sailors' or tablename='reserves' or tablename='boat' or tablename='query_8';
DROP INDEX IF EXISTS b_sailorsSID cascade;
DROP INDEX IF EXISTS b_reservesSID cascade;
DROP INDEX IF EXISTS b_reservesBID cascade;
DROP INDEX IF EXISTS b_boat1 cascade;
DROP INDEX IF EXISTS b_boat2 cascade;
DROP INDEX IF EXISTS b_query_8 cascade;
set enable_hashagg = off;
set enable_hashjoin = off;
set enable_seqscan = on;
-- Query 8 (STATISTICS)
explain analyze select s.sname
from sailors s
where s.sid in (select r.sid
from reserves r
where r. bid in (select b.bid
from boat b
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)
6 public query_8 b_query_8 [null] CREATE INDEX b_query_8 ON public.query_8 USING hash (sid)
Successfully run. Total query runtime: 137 msec. 4 rows affected.
Total rows: 6 of 6
Query complete 00:00:00.169
Ln 198, Col 1


pgAdmin 4
schema3/postgres@postgres
Explain Analyze
Shift F7
explain analyze select s.sname
from sailors s
where s.sid in (select r.sid
from reserves r
where r. bid in (select b.bid
from boat b
where b.color = 'red'));
-- Query 8 optimized (STATISTICS)
-- Query 8 view table of reserves sids that have red boats
create MATERIALIZED VIEW query_8
as
select r1.sid
from (select r.sid
from reserves r
where exists (select * from boat b where r.bid=b.bid and color='red'))
explain analyze select s.sname
from sailors s where exists (select * from query_8 r1 where s.sid=r1.sid);
Data output
Messages
Explain
Notifications
Graphical
Analysis
Statistics
reserves
b_boat1
Nested Loop Semi Join
Sort
Unique
b_sailorssid
Nested Loop Inne Join
Successfully run. Total query runtime: 137 msec. 4 rows affected.
Total rows: 1 of 1
Query complete 00:00:00.105
Ln 231, Col 17



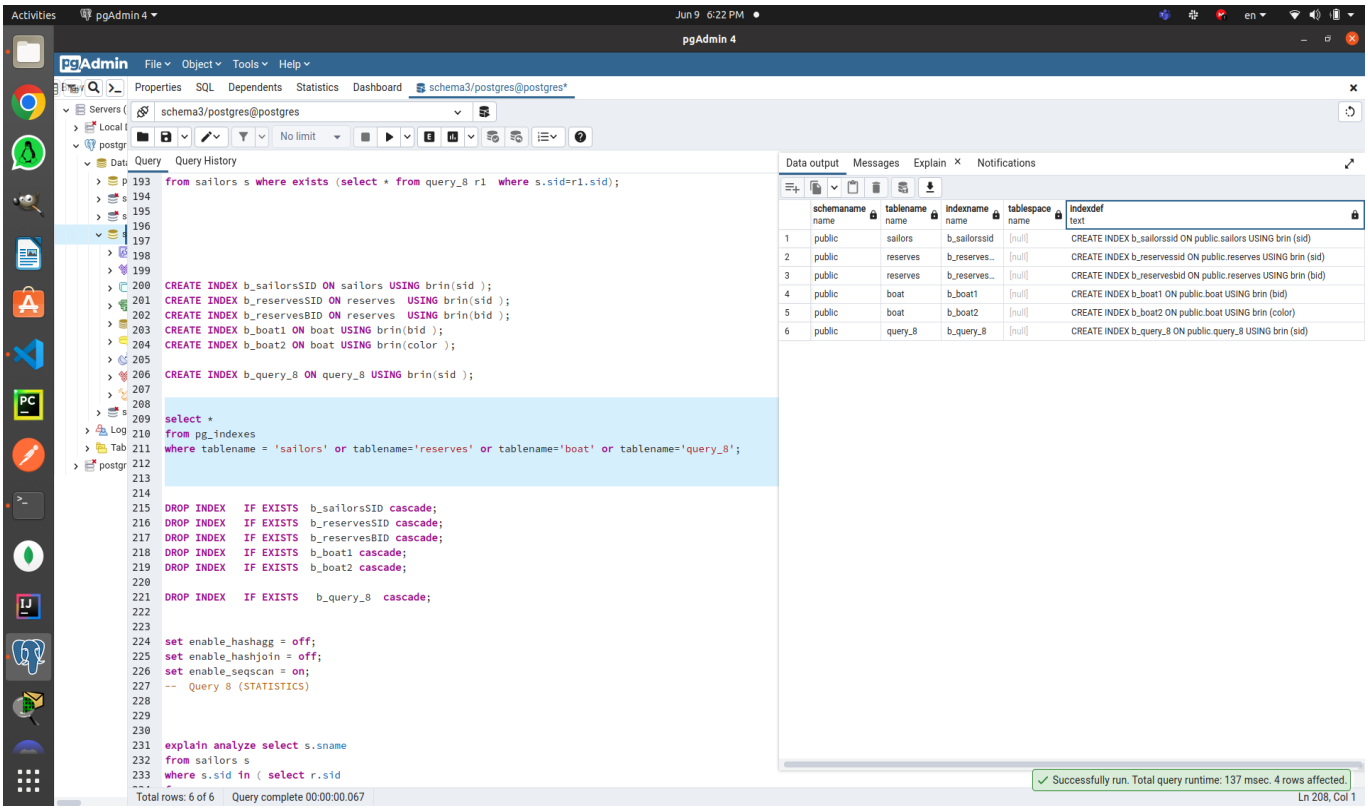
Explanation :

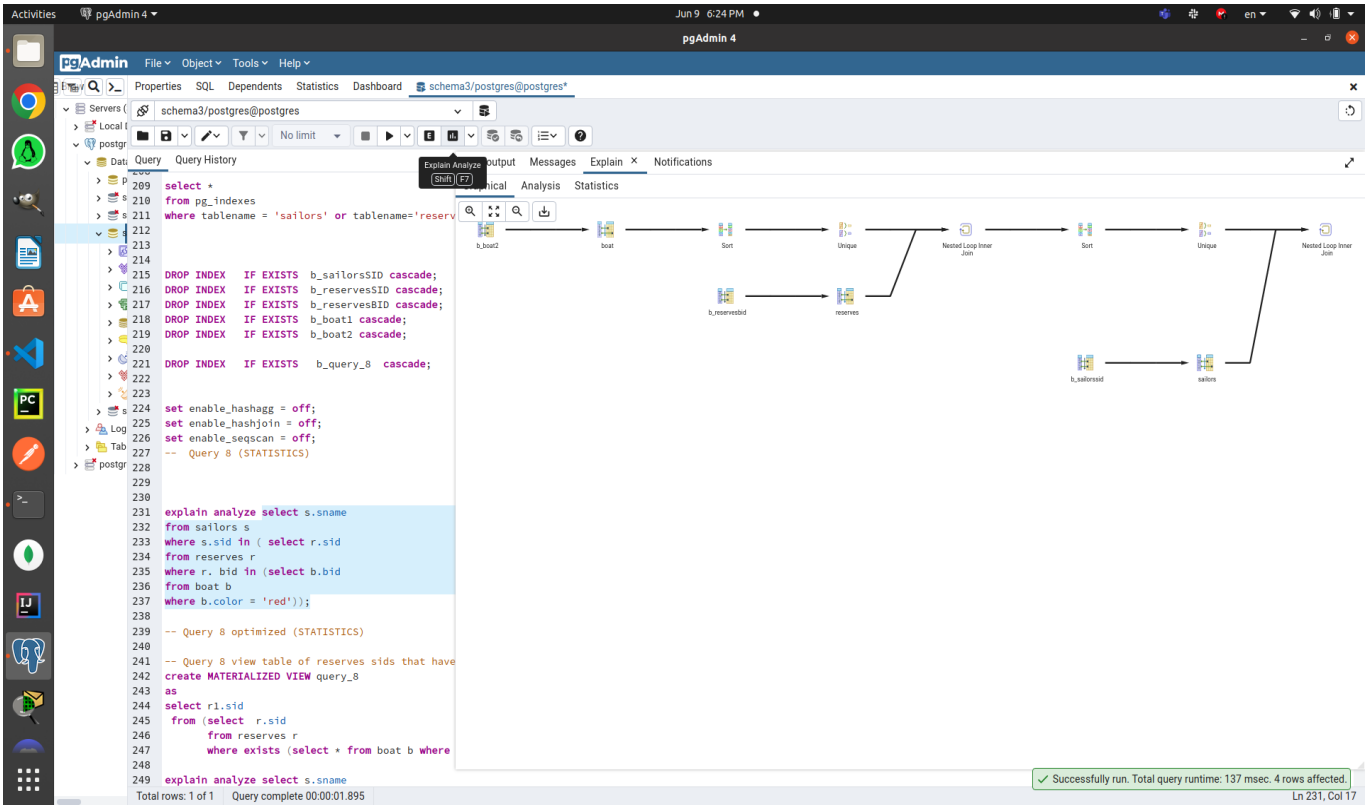
- Metrics :

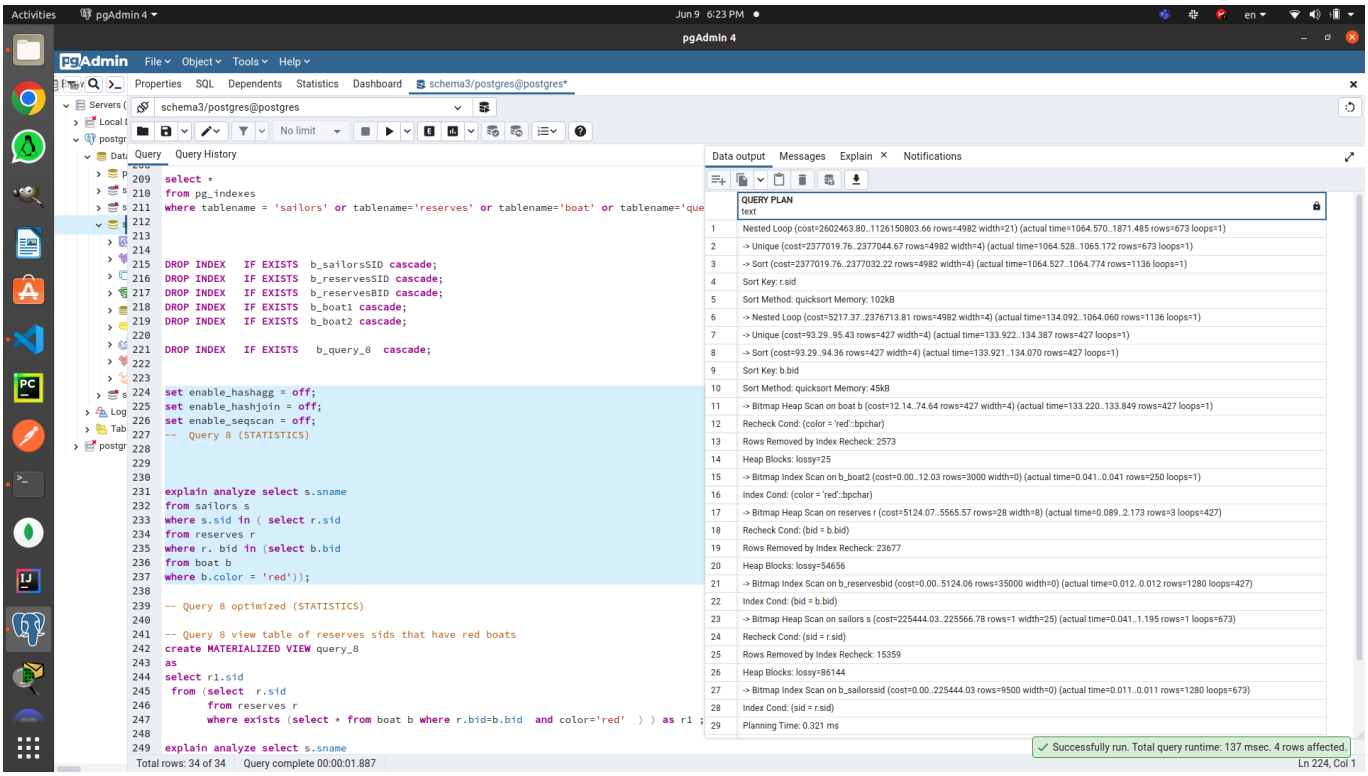
  Execution Time : 38.244 ms Total Expected Cost : 2168.36

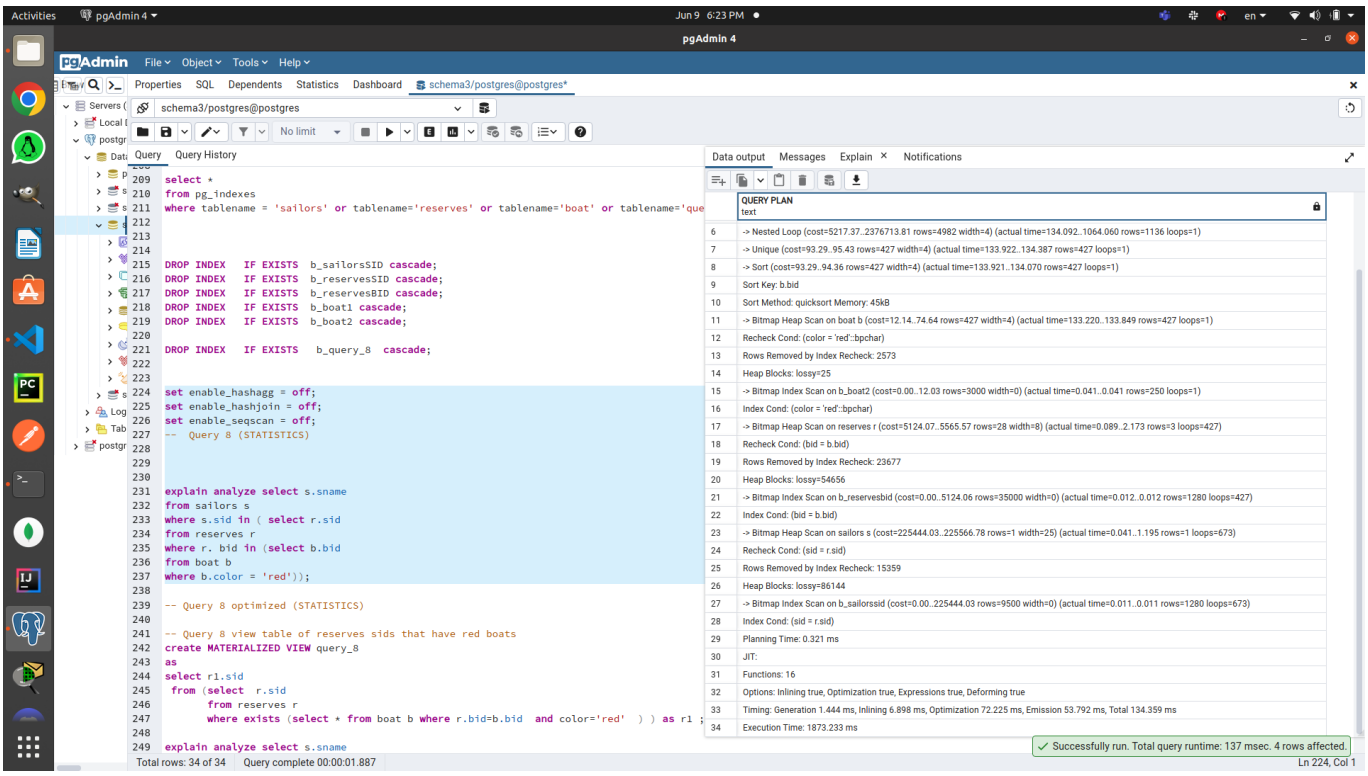
- The Hash helped in the performance it decreased the Execution Time and Expected Cost .
- The Query Planner used the Hash index because Hash is O(1) performance with Exact Values and considered the best for exact values queries (BEST OF THEM ALL).
- Here it showed the improvement due to for condition of Joining in the Query the Nested Loop Semi Join using index scan using Hash based algorithm on the condition (b.bid=r.bid) which approximatly maded to be O(n) performance .
- Here it showed the improvement due to for condition of Joining in the Query the Nested Loop Semi Join using index scan using Hash based algorithm on the condition (s.sid=r.sid) which approximatly maded to be O(n) performance .
- Here it showed the improvement due to for condition of (color = "red") with performance of O(1) .

4. given query with BRIN indices only :


pgAdmin 4
Properties SQL Dependents Statistics Dashboard schema3/postgres@postgres*
schema3/postgres@postgres
Query Query History
from sailors s where exists (select * from query_8 r1 where s.sid=r1.sid);
CREATE INDEX b_sailorsSID ON sailors USING brin(sid);
CREATE INDEX b_reservesSID ON reserves USING brin(sid);
CREATE INDEX b_reservesBID ON reserves USING brin(bid);
CREATE INDEX b_boat1 ON boat USING brin(bid);
CREATE INDEX b_boat2 ON boat USING brin(color);
CREATE INDEX b_query_8 ON query_8 USING brin(sid);
select *
from pg_indexes
where tablename = 'sailors' or tablename='reserves' or tablename='boat' or tablename='query_8';
DROP INDEX IF EXISTS b_sailorsSID cascade;
DROP INDEX IF EXISTS b_reservesSID cascade;
DROP INDEX IF EXISTS b_reservesBID cascade;
DROP INDEX IF EXISTS b_boat1 cascade;
DROP INDEX IF EXISTS b_boat2 cascade;
DROP INDEX IF EXISTS b_query_8 cascade;
set enable_hashagg = off;
set enable_hashjoin = off;
set enable_seqscan = on;
-- Query 8 (STATISTICS)
explain analyze select s.sname
from sailors s
where s.sid in (select r.sid
Data output Messages Explain Notifications
schemaname tablename indexname tablespace indexdef
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_boat1 [null] CREATE INDEX b_boat1 ON public.boat USING brin (bid)
5 public boat b_boat2 [null] CREATE INDEX b_boat2 ON public.boat USING brin (color)
6 public query_8 b_query_8 [null] CREATE INDEX b_query_8 ON public.query_8 USING brin (sid)
Successfully run. Total query runtime: 137 msec. 4 rows affected.
Total rows: 6 of 6 Query complete 00:00:00.067
Ln 208, Col 1


pgAdmin 4
Properties SQL Dependents Statistics Dashboard schema3/postgres@postgres*
schema3/postgres@postgres
Query Query History
Explain Analyze Shift F7
Data output Messages Explain Notifications
Graphical Analysis Statistics
b_boat2 boat Sort Unique Nested Loop Inner Join Sort Unique Nested Loop Inner Join
b_reservesbid reserves
b_sailorssid sailors
select *
from pg_indexes
where tablename = 'sailors' or tablename='reserv
DROP INDEX IF EXISTS b_sailorsSID cascade;
DROP INDEX IF EXISTS b_reservesSID cascade;
DROP INDEX IF EXISTS b_reservesBID cascade;
DROP INDEX IF EXISTS b_boat1 cascade;
DROP INDEX IF EXISTS b_boat2 cascade;
DROP INDEX IF EXISTS b_query_8 cascade;
set enable_hashagg = off;
set enable_hashjoin = off;
set enable_seqscan = off;
-- Query 8 (STATISTICS)
explain analyze select s.sname
from sailors s
where s.sid in (select r.sid
from reserves r
where r. bid in (select b.bid
from boat b
where b.color = 'red'));
-- Query 8 optimized (STATISTICS)
-- Query 8 view table of reserves sids that have
create MATERIALIZED VIEW query_8
as
select r1.sid
from (select r.sid
from reserves r
where exists (select * from boat b where
explain analyze select s.sname
Successfully run. Total query runtime: 137 msec. 4 rows affected.
Total rows: 1 of 1 Query complete 00:00:01.895
Ln 231, Col 17





Explanation :

- Metrics :

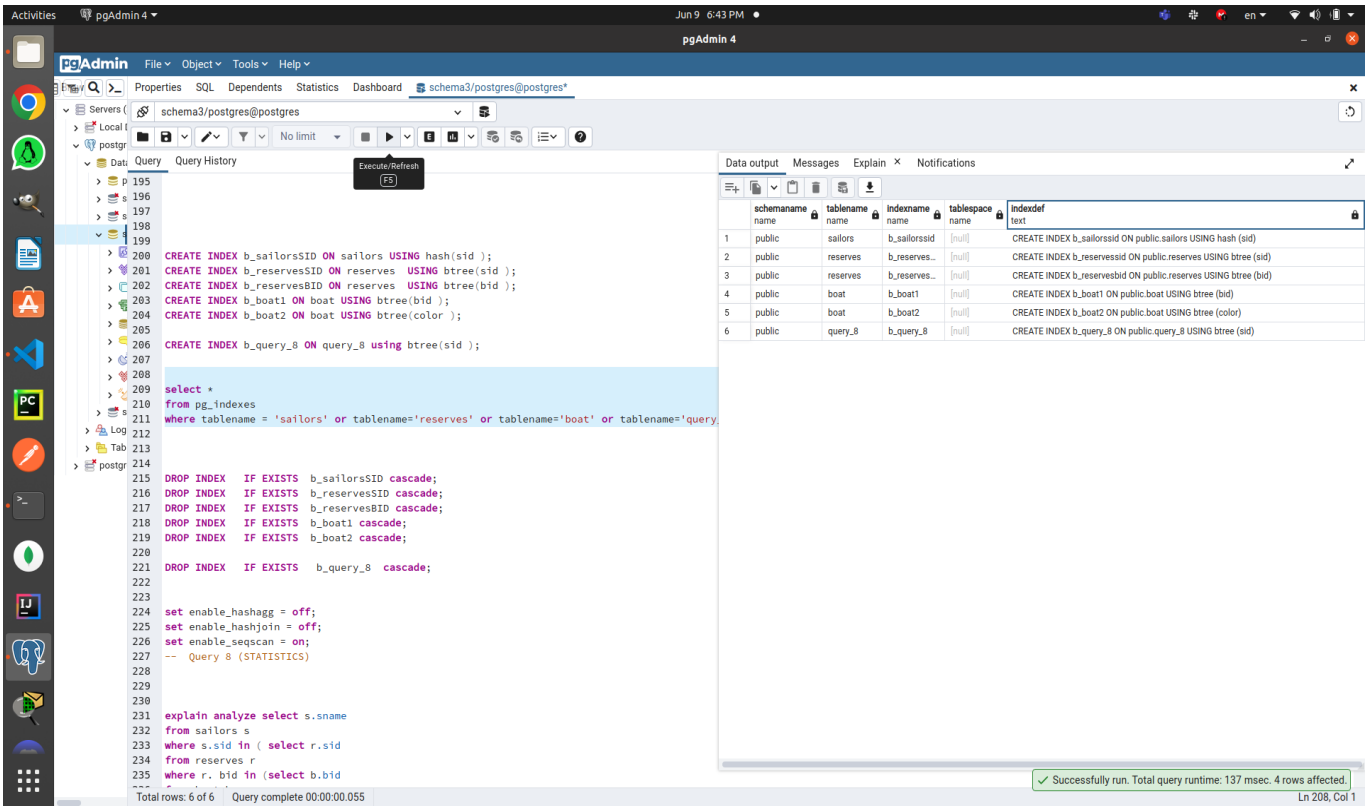
  Execution Time : 18733.23 ms Total Expected Cost : 1126150803.66

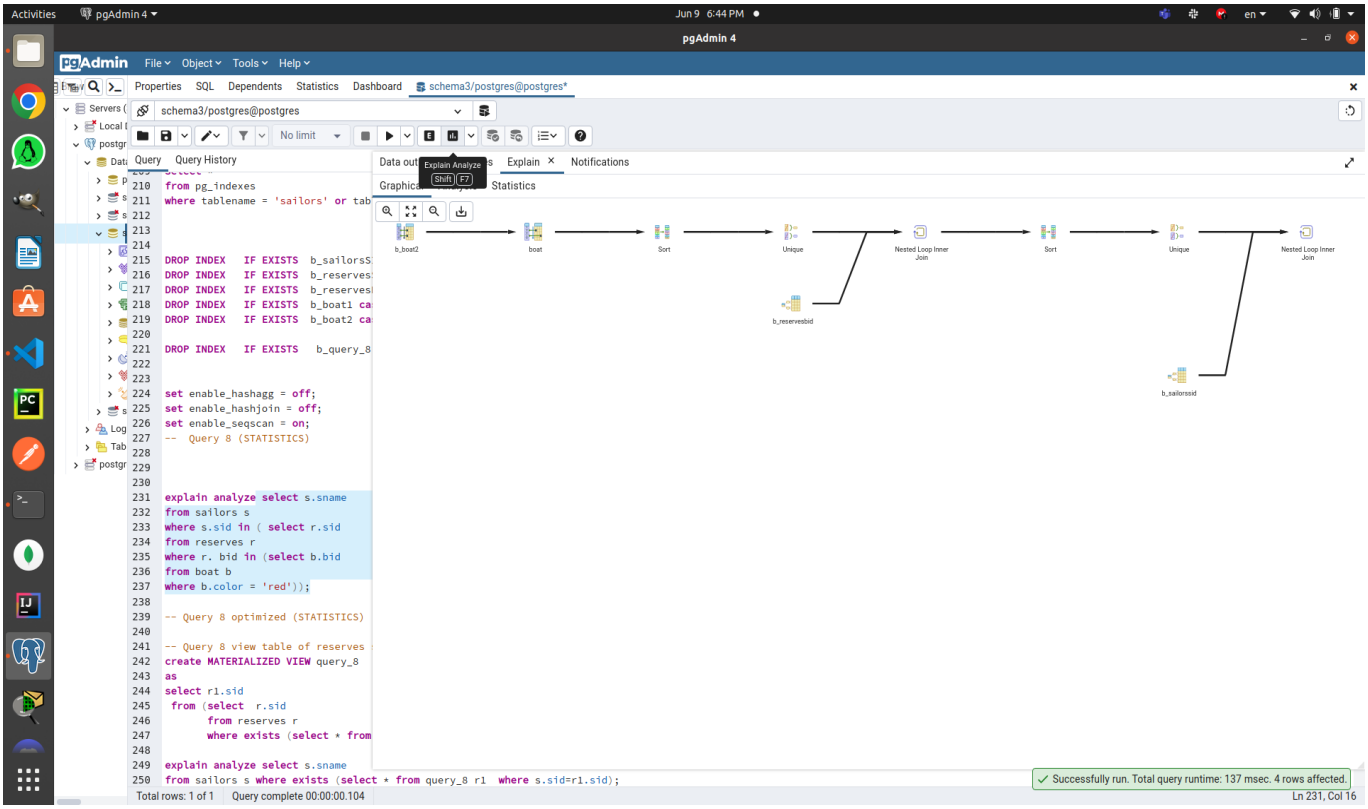
- Here the BRIN was not used in the original Query Plan settings (Hashjoin and HashAgg are off) so I've made seqscan=off too.

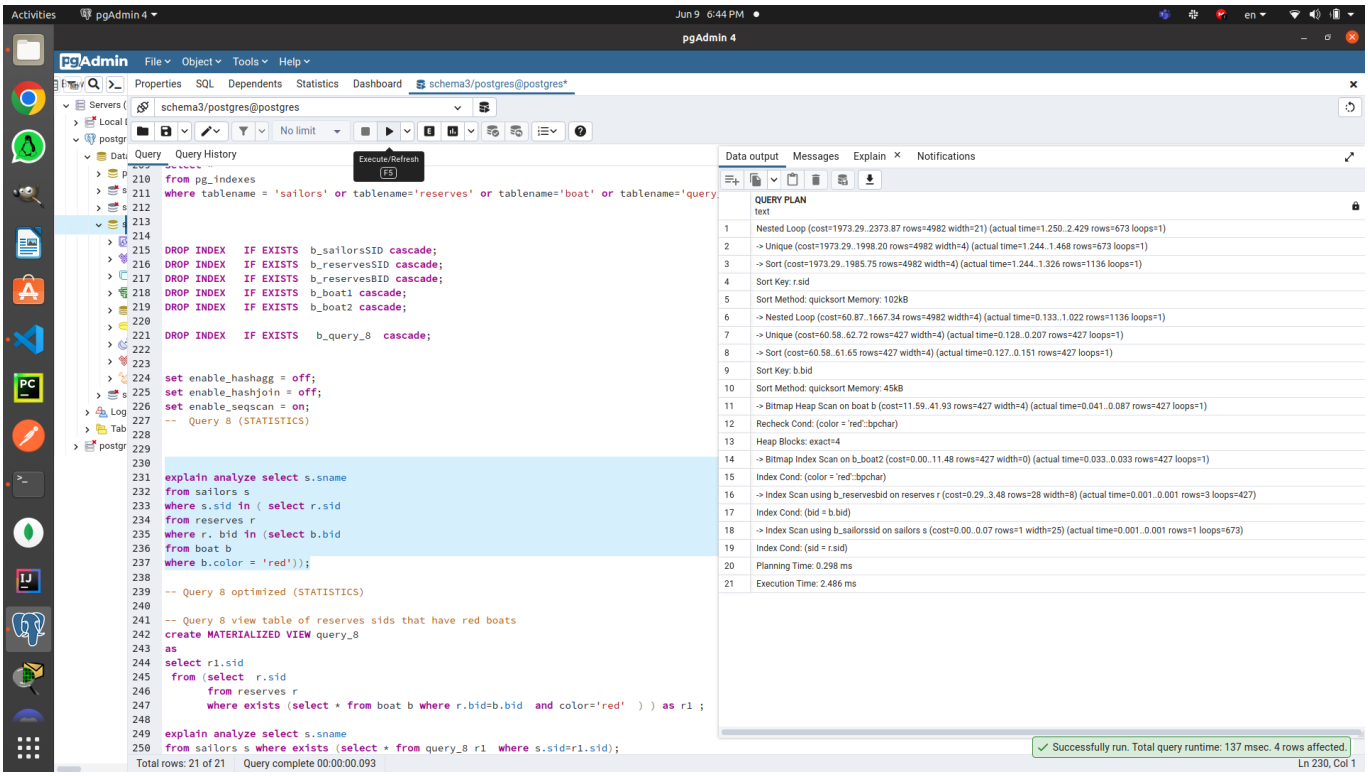
- The Execution Time and Expected Cost became the (WORST OF THEM ALL) .

- This happened because the Query Optimizer didnt used it from the first 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.
- Because there where no Aggregation used and was not low selectivity Query so it was not helpful to the performance.
- The plan performs a (Bit map index scan) on all indices and then to (Bit map heap scan) to select relevant rows .

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


pgAdmin 4
Properties SQL Dependents Statistics Dashboard schema3/postgres@postgres*
schema3/postgres@postgres
Query Query History
Execute/Refresh F5
CREATE INDEX b_sailorsSID ON sailors USING hash(sid);
CREATE INDEX b_reservesSID ON reserves USING btree(sid);
CREATE INDEX b_reservesBID ON reserves USING btree(bid);
CREATE INDEX b_boat1 ON boat USING btree(bid);
CREATE INDEX b_boat2 ON boat USING btree(color);
CREATE INDEX b_query_8 ON query_8 using btree(sid);
select *
from pg_indexes
where tablename = 'sailors' or tablename='reserves' or tablename='boat' or tablename='query
DROP INDEX IF EXISTS b_sailorsSID cascade;
DROP INDEX IF EXISTS b_reservesSID cascade;
DROP INDEX IF EXISTS b_reservesBID cascade;
DROP INDEX IF EXISTS b_boat1 cascade;
DROP INDEX IF EXISTS b_boat2 cascade;
DROP INDEX IF EXISTS b_query_8 cascade;
set enable_hashagg = off;
set enable_hashjoin = off;
set enable_seqscan = on;
-- Query 8 (STATISTICS)
explain analyze select s.sname
from sailors s
where s.sid in (select r.sid
from reserves r
where r. bid in (select b.bid
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 btree (sid)
3 public reserves b_reserves... [null] CREATE INDEX b_reservesbid ON public.reserves USING btree (bid)
4 public boat b_boat1 [null] CREATE INDEX b_boat1 ON public.boat USING btree (bid)
5 public boat b_boat2 [null] CREATE INDEX b_boat2 ON public.boat USING btree (color)
6 public query_8 b_query_8 [null] CREATE INDEX b_query_8 ON public.query_8 USING btree (sid)
Successfully run. Total query runtime: 137 msec. 4 rows affected.
Total rows: 6 of 6 Query complete 00:00:00.055
Ln 208, Col 1


pgAdmin 4
Properties SQL Dependents Statistics Dashboard schema3/postgres@postgres*
schema3/postgres@postgres
Query Query History
Explain Analyze Shift F7
Explain Notifications
Graphical Statistics
b_boat2
boat
Sort
Unique
Nested Loop Inner Join
b_reservesbid
Sort
Unique
Nested Loop Inner Join
b_sailorssid
from pg_indexes
where tablename = 'sailors' or tab
DROP INDEX IF EXISTS b_sailorsS
DROP INDEX IF EXISTS b_reserves
DROP INDEX IF EXISTS b_reserves
DROP INDEX IF EXISTS b_boat1 ca
DROP INDEX IF EXISTS b_boat2 ca
DROP INDEX IF EXISTS b_query_8
set enable_hashagg = off;
set enable_hashjoin = off;
set enable_seqscan = on;
-- Query 8 (STATISTICS)
explain analyze select s.sname
from sailors s
where s.sid in (select r.sid
from reserves r
where r. bid in (select b.bid
from boat b
where b.color = 'red'));
-- Query 8 optimized (STATISTICS)
-- Query 8 view table of reserves
create MATERIALIZED VIEW query_8
as
select r1.sid
from (select r.sid
from reserves r
where exists (select * from
explain analyze select s.sname
from sailors s where exists (select * from query_8 r1 where s.sid=r1.sid);
Successfully run. Total query runtime: 137 msec. 4 rows affected.
Total rows: 1 of 1 Query complete 00:00:00.104
Ln 231, Col 16



Explanation :

- Metrics :

  Execution Time : 2.486 ms Total Expected Cost : 2373.87

- The Query Planner used the Merge semi join by using both the ZigZag join and the Hash based algorithm together on the condition (s.sid =r.sid) which improved the Execution time and the expected cost way more better which made the join in O(n log n) .
- - The Query Planner used the Merge semi join by using both the ZigZag join and the Hash based algorithm together on the condition (r.bid = b.bid) which improved the Execution time and the expected cost way more better which made the join in O(n log n) .
- And It used the Hash indexed scan on color ='red' with performance of O(1) .
- The Mix indices helped in the performance it decreased the Execution Time and Expected Cost .

Optimized Query

```
-- Query 8 view table of reserves sids that have red boats

create MATERIALIZED VIEW query_8
as
select r1.sid
 from (select  r.sid
       from reserves r
       where exists (select * from boat b where r.bid=b.bid  and
color='red'  ) ) as r1 ;
```

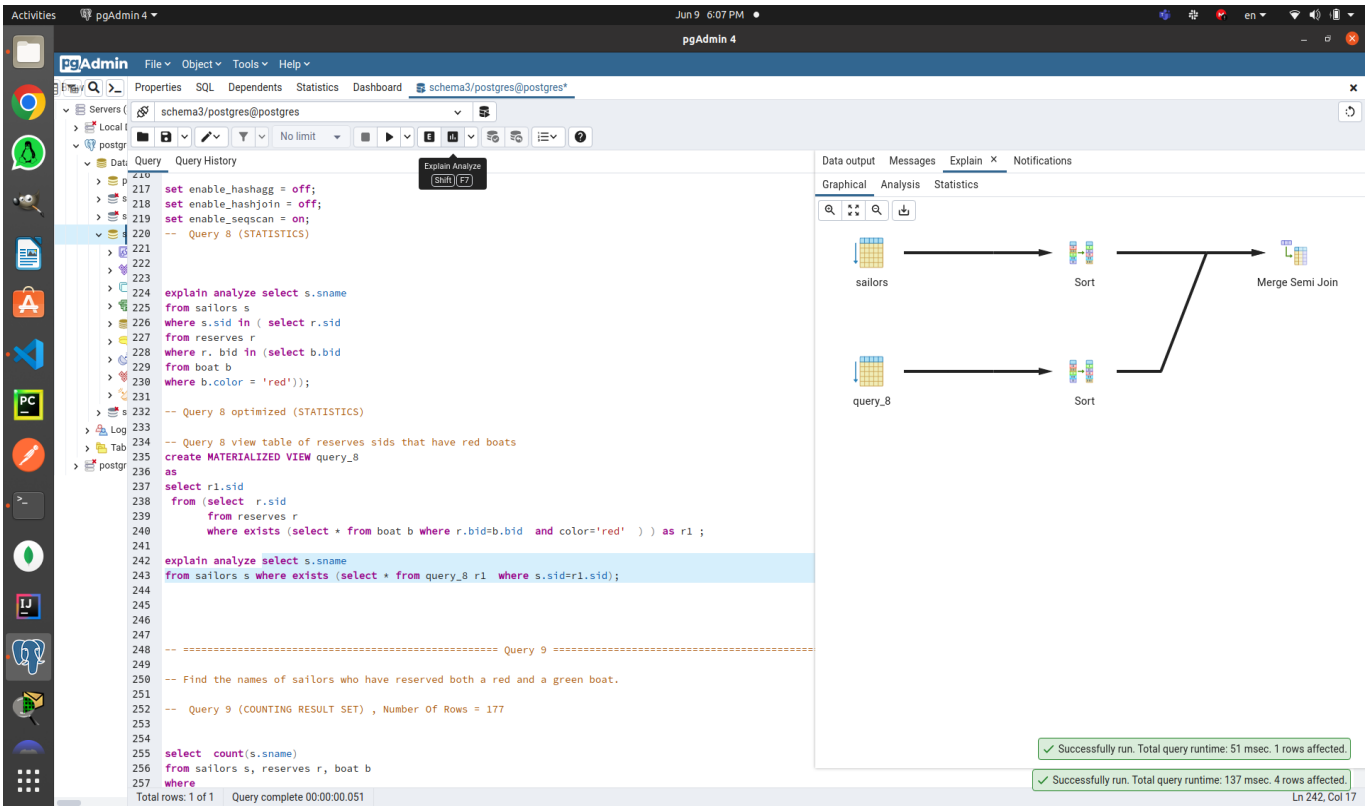
```
explain analyze select s.sname
from sailors s where exists (select * from query_8 r1  where s.sid=r1.sid);
```

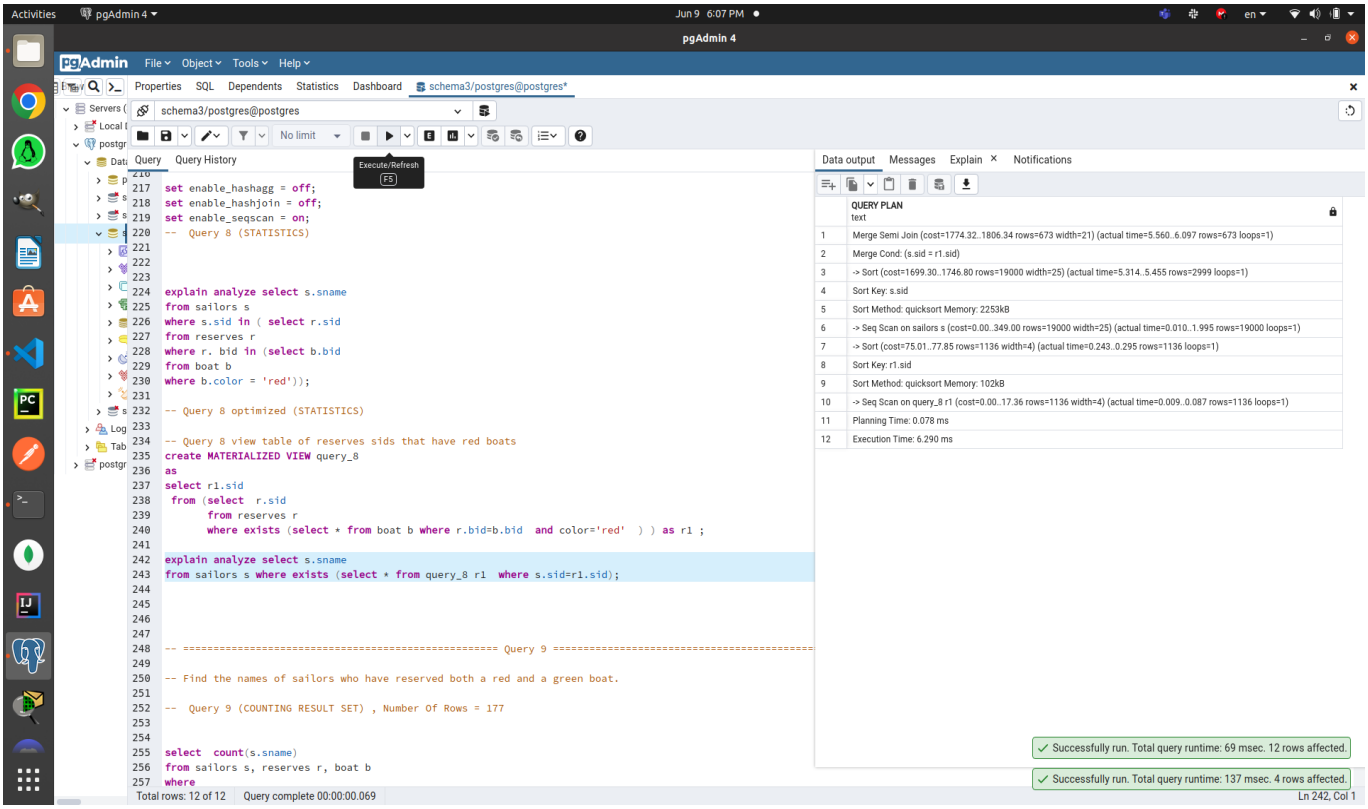
Result Set

- 673 Rows

Report

1. given query without an index :





Explanation :

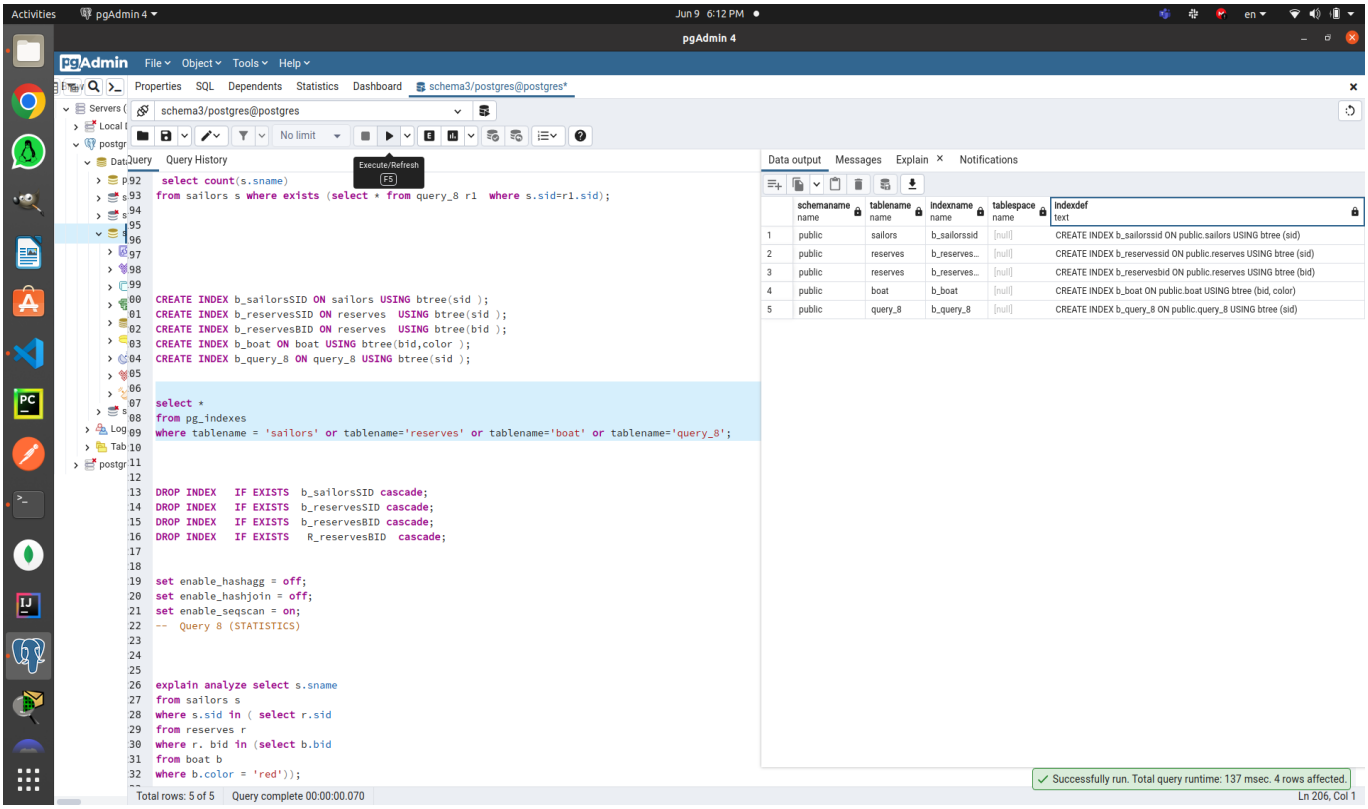
- Metrics :

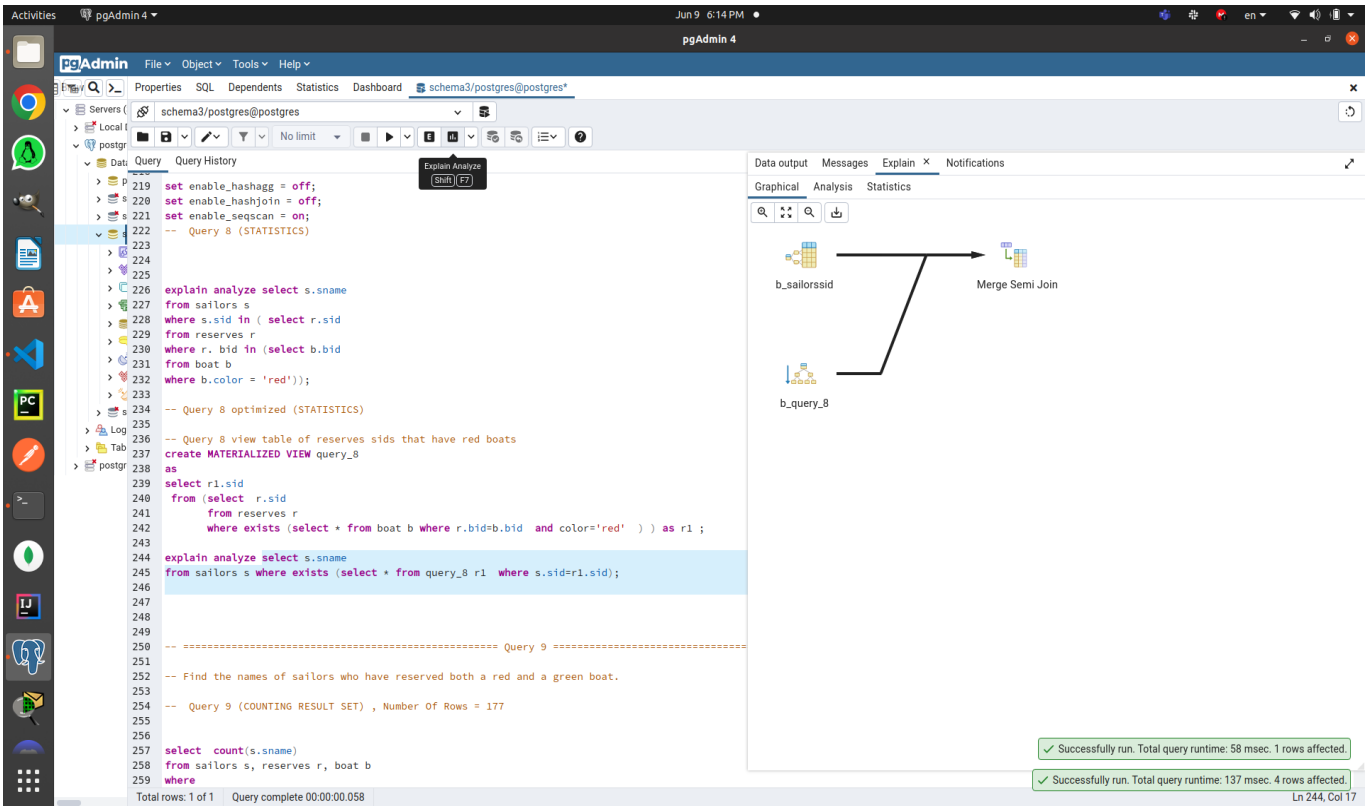
  Execution Time : 6.290 ms Total Expected Cost : 1806.34

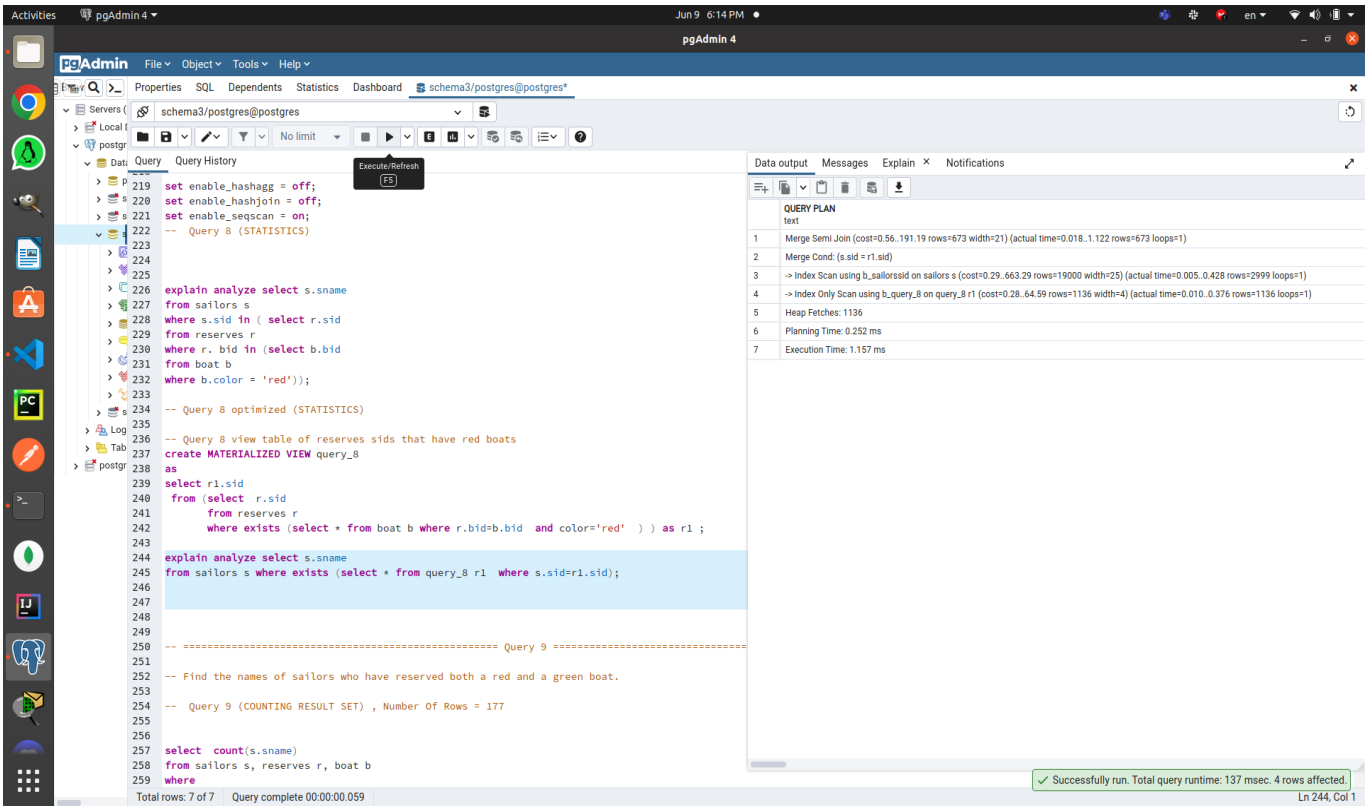
- Reason :

- This Query Improved in the Execution time and Expected Cost than the Original Query from 5664.74 to 1806.34 .
- Because I used Materialized Views which already made an Intermediate Ready Table with smaller Size that Optimized Query used it which decreased the number of steps needed(Filtration of the table over red boats reserve sids) for the Query to get Executed .
- Because the loops ends faster and exits due to I used the Exist Operator instead of the In Operator that waits till the end.

2. given query with B+ trees indices only :







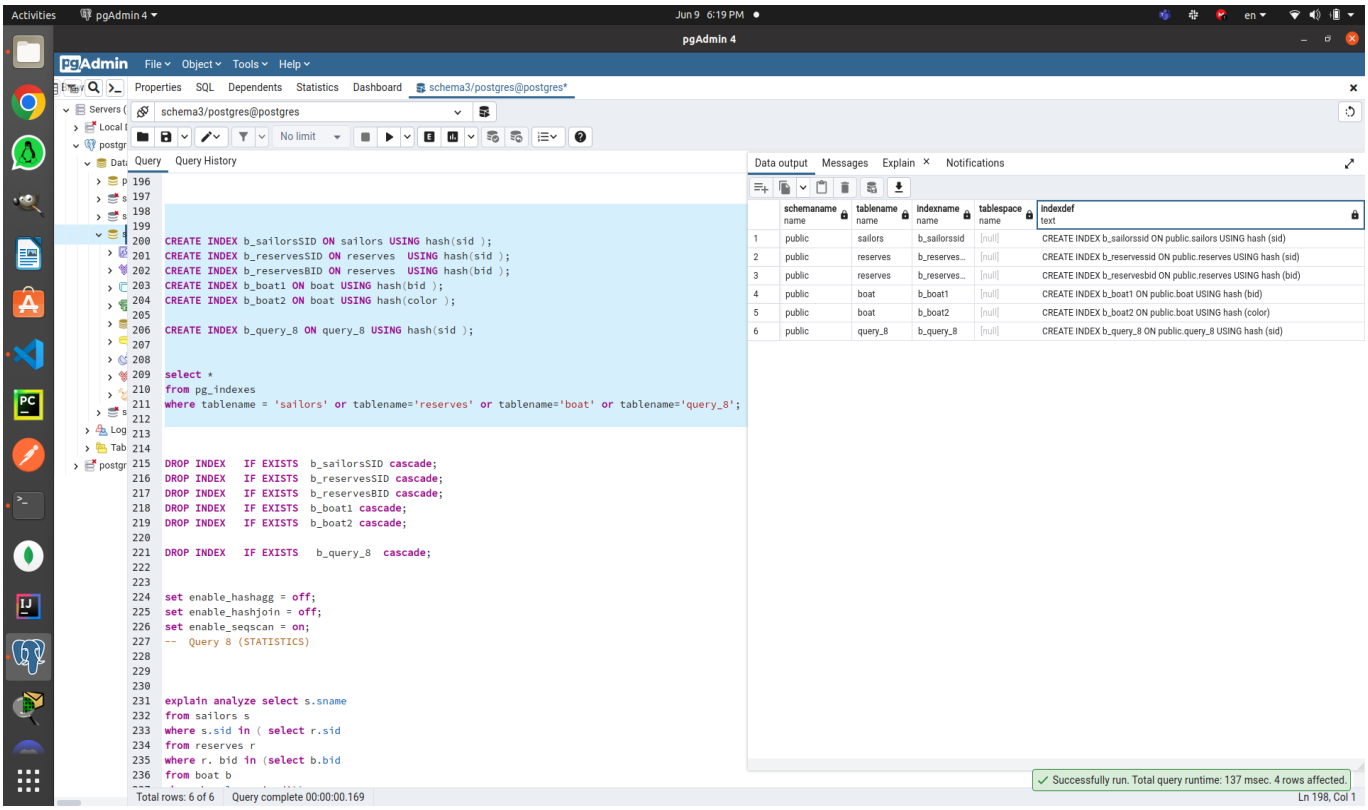
Explanation :

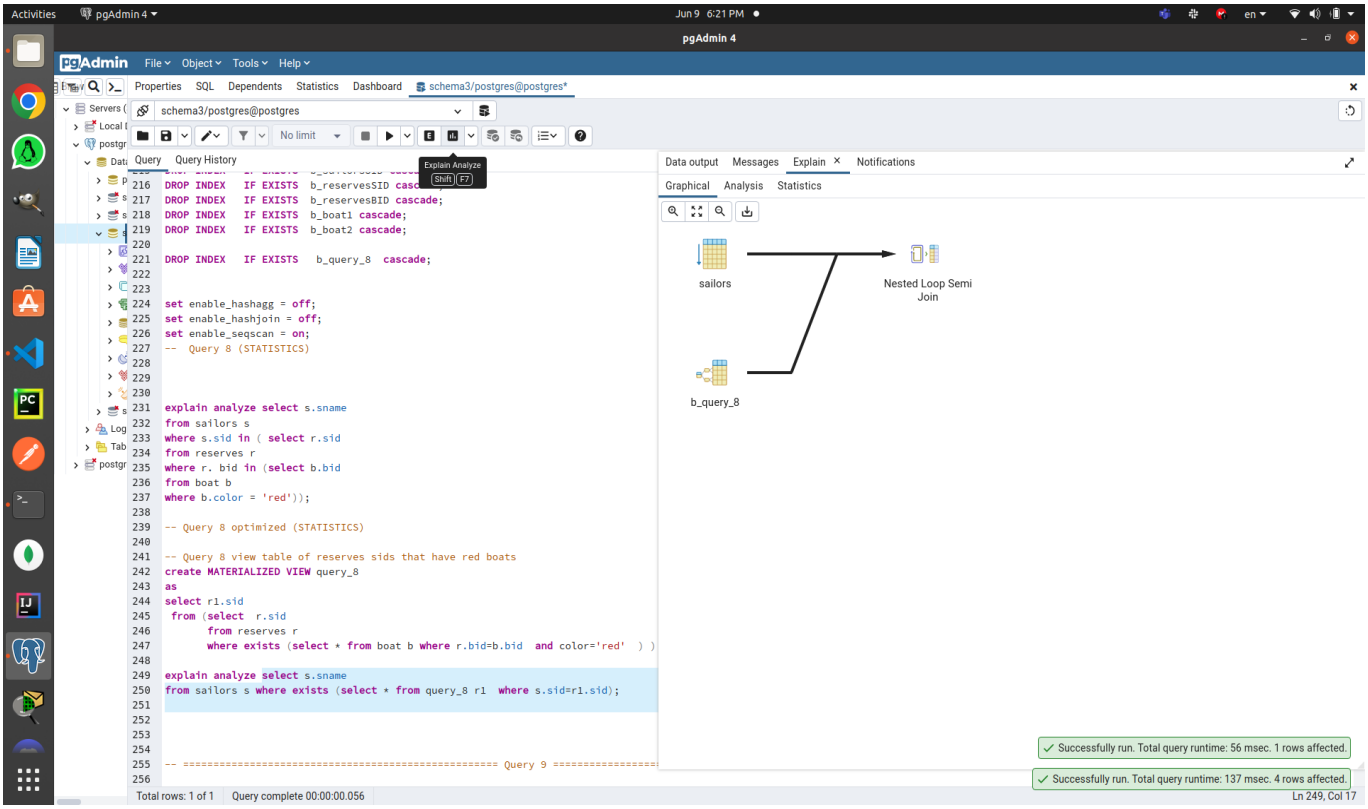
- Metrics :

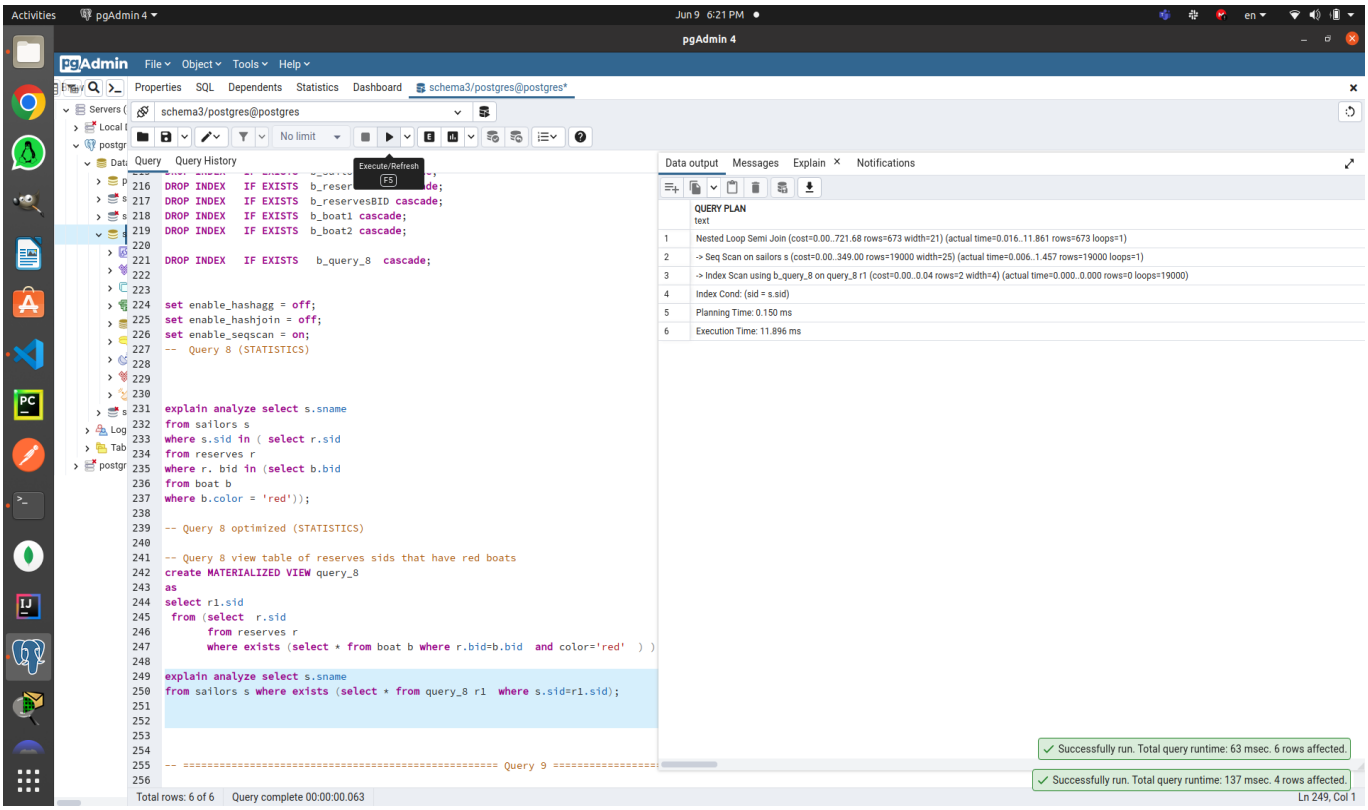
  Execution Time : 1.157 ms Total Expected Cost :191.19

- The B-tree helped in the performance it decreased the Execution Time and Expected Cost (BEST OF THEM ALL).
- 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 (s.sid=R.sid (it used the index that was built on query_8 view)) condition of Joining in the Query the Merge Semi Join used index scan using ZigZag and algorithm and these columns where built on it an b-tree index.

1. given query with hash indices only :


pgAdmin 4
Properties SQL Dependents Statistics Dashboard schema3/postgres@postgres*
schema3/postgres@postgres
Query Query History
Data output Messages Explain Notifications
CREATE INDEX b_sailorsSID ON sailors USING hash(sid);
CREATE INDEX b_reservesSID ON reserves USING hash(sid);
CREATE INDEX b_reservesBID ON reserves USING hash(bid);
CREATE INDEX b_boat1 ON boat USING hash(bid);
CREATE INDEX b_boat2 ON boat USING hash(color);
CREATE INDEX b_query_8 ON query_8 USING hash(sid);
select *
from pg_indexes
where tablename = 'sailors' or tablename='reserves' or tablename='boat' or tablename='query_8';
DROP INDEX IF EXISTS b_sailorsSID cascade;
DROP INDEX IF EXISTS b_reservesSID cascade;
DROP INDEX IF EXISTS b_reservesBID cascade;
DROP INDEX IF EXISTS b_boat1 cascade;
DROP INDEX IF EXISTS b_boat2 cascade;
DROP INDEX IF EXISTS b_query_8 cascade;
set enable_hashagg = off;
set enable_hashjoin = off;
set enable_seqscan = on;
-- Query 8 (STATISTICS)
explain analyze select s.sname
from sailors s
where s.sid in (select r.sid
from reserves r
where r. bid in (select b.bid
from boat b
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)
6 public query_8 b_query_8 [null] CREATE INDEX b_query_8 ON public.query_8 USING hash (sid)
Successfully run. Total query runtime: 137 msec. 4 rows affected.
Total rows: 6 of 6 Query complete 00:00:00.169
Ln 198, Col 1


pgAdmin 4
Properties SQL Dependents Statistics Dashboard schema3/postgres@postgres*
schema3/postgres@postgres
Query Query History
Explain Analyze Shift F7
Data output Messages Explain Notifications
Graphical Analysis Statistics
sailors
Nested Loop Semi Join
b_query_8
DROP INDEX IF EXISTS b_reservesSID cascade;
DROP INDEX IF EXISTS b_reservesBID cascade;
DROP INDEX IF EXISTS b_boat1 cascade;
DROP INDEX IF EXISTS b_boat2 cascade;
DROP INDEX IF EXISTS b_query_8 cascade;
set enable_hashagg = off;
set enable_hashjoin = off;
set enable_seqscan = on;
-- Query 8 (STATISTICS)
explain analyze select s.sname
from sailors s
where s.sid in (select r.sid
from reserves r
where r. bid in (select b.bid
from boat b
where b.color = 'red'));
-- Query 8 optimized (STATISTICS)
-- Query 8 view table of reserves sids that have red boats
create MATERIALIZED VIEW query_8
as
select r1.sid
from (select r.sid
from reserves r
where exists (select * from boat b where r.bid=b.bid and color='red'))
explain analyze select s.sname
from sailors s where exists (select * from query_8 r1 where s.sid=r1.sid);
-- ========== Query 9 ==========
Successfully run. Total query runtime: 56 msec. 1 rows affected.
Successfully run. Total query runtime: 137 msec. 4 rows affected.
Total rows: 1 of 1 Query complete 00:00:00.056
Ln 249, Col 17



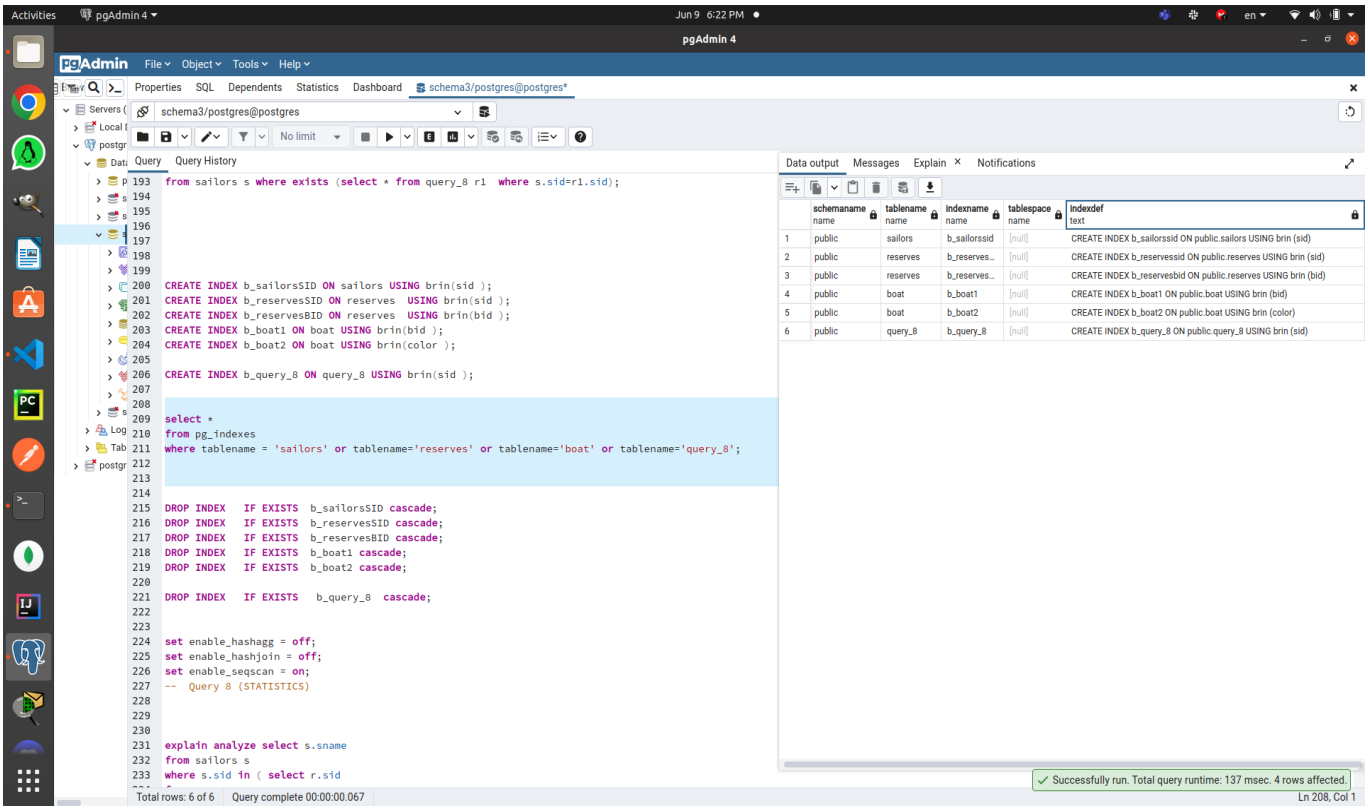
Explanation :

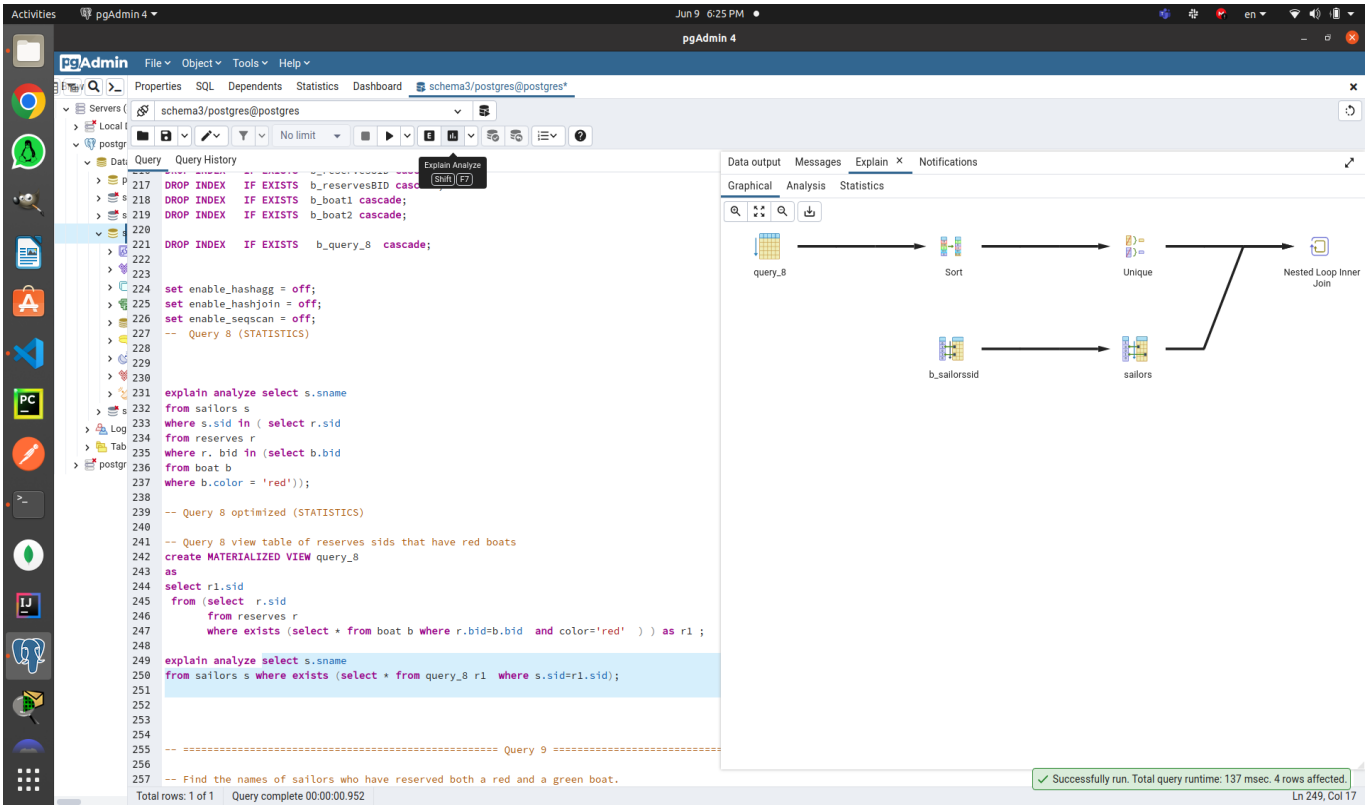
- Metrics :

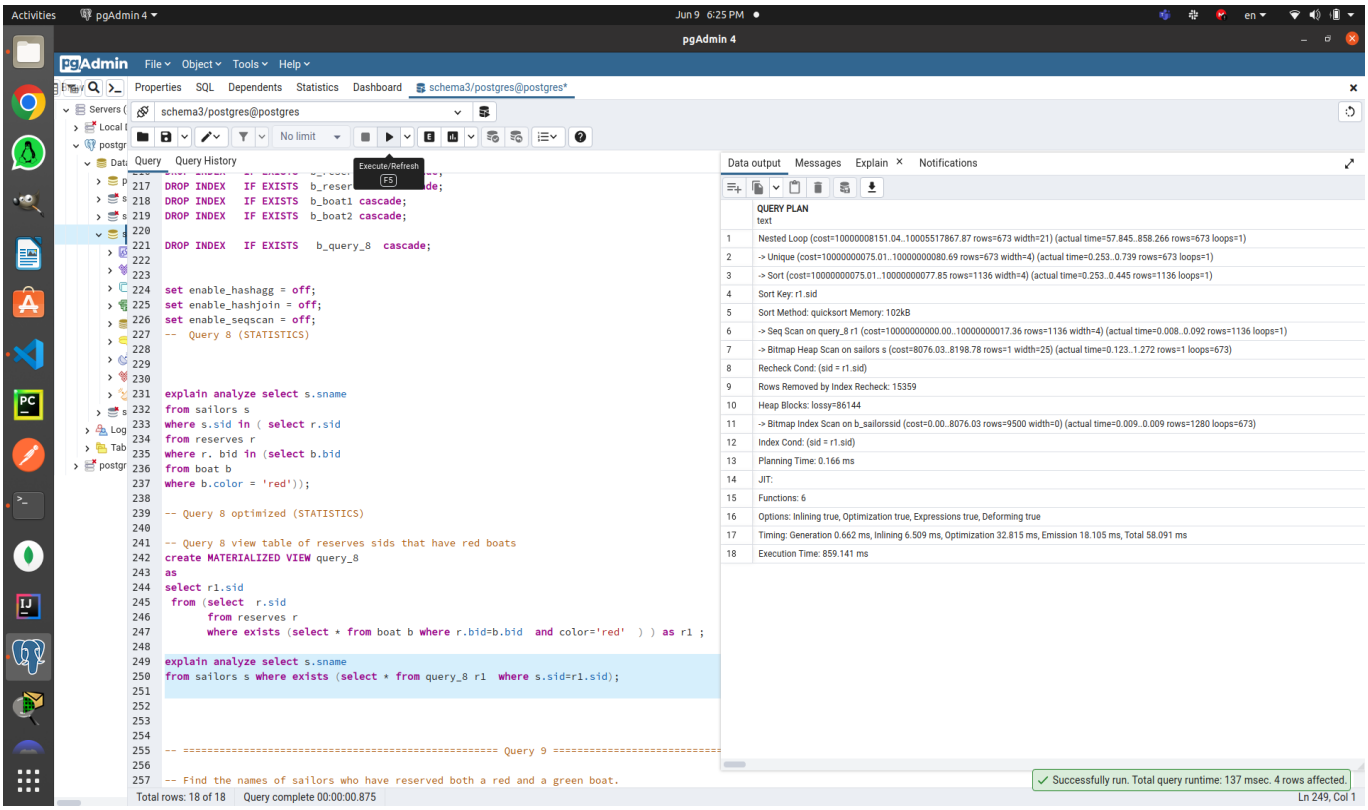
  Execution Time : 11.896 ms Total Expected Cost : 721.68

- 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 Semi Join using index scan using Hash based algorithm on the condition (sid(from query_8 view)=s.sid) which approximatly maded to be O(n) performance.

4. given query with BRIN indices only :


pgAdmin 4
Properties SQL Dependents Statistics Dashboard schema3/postgres@postgres*
schema3/postgres@postgres
Query Query History
from sailors s where exists (select * from query_8 r1 where s.sid=r1.sid);
CREATE INDEX b_sailorsSID ON sailors USING brin(sid);
CREATE INDEX b_reservesSID ON reserves USING brin(sid);
CREATE INDEX b_reservesBID ON reserves USING brin(bid);
CREATE INDEX b_boat1 ON boat USING brin(bid);
CREATE INDEX b_boat2 ON boat USING brin(color);
CREATE INDEX b_query_8 ON query_8 USING brin(sid);
select *
from pg_indexes
where tablename = 'sailors' or tablename='reserves' or tablename='boat' or tablename='query_8';
DROP INDEX IF EXISTS b_sailorsSID cascade;
DROP INDEX IF EXISTS b_reservesSID cascade;
DROP INDEX IF EXISTS b_reservesBID cascade;
DROP INDEX IF EXISTS b_boat1 cascade;
DROP INDEX IF EXISTS b_boat2 cascade;
DROP INDEX IF EXISTS b_query_8 cascade;
set enable_hashagg = off;
set enable_hashjoin = off;
set enable_seqscan = on;
-- Query 8 (STATISTICS)
explain analyze select s.sname
from sailors s
where s.sid in (select r.sid
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_boat1 [null] CREATE INDEX b_boat1 ON public.boat USING brin (bid)
5 public boat b_boat2 [null] CREATE INDEX b_boat2 ON public.boat USING brin (color)
6 public query_8 b_query_8 [null] CREATE INDEX b_query_8 ON public.query_8 USING brin (sid)
Successfully run. Total query runtime: 137 msec. 4 rows affected.
Total rows: 6 of 6 Query complete 00:00:00.067
Ln 208, Col 1


pgAdmin 4
Properties SQL Dependents Statistics Dashboard schema3/postgres@postgres*
schema3/postgres@postgres
Query Query History
Explain Analyze Shift F7
DROP INDEX IF EXISTS b_reservesBID cascade;
DROP INDEX IF EXISTS b_boat1 cascade;
DROP INDEX IF EXISTS b_boat2 cascade;
DROP INDEX IF EXISTS b_query_8 cascade;
set enable_hashagg = off;
set enable_hashjoin = off;
set enable_seqscan = off;
-- Query 8 (STATISTICS)
explain analyze select s.sname
from sailors s
where s.sid in (select r.sid
from reserves r
where r. bid in (select b.bid
from boat b
where b.color = 'red'));
-- Query 8 optimized (STATISTICS)
-- Query 8 view table of reserves sids that have red boats
create MATERIALIZED VIEW query_8
as
select r1.sid
from (select r.sid
from reserves r
where exists (select * from boat b where r.bid=b.bid and color='red')) as r1 ;
explain analyze select s.sname
from sailors s where exists (select * from query_8 r1 where s.sid=r1.sid);
-- Find the names of sailors who have reserved both a red and a green boat.
Data output Messages Explain Notifications
Graphical Analysis Statistics
query_8
Sort
Unique
Nested Loop Inner Join
b_sailorssid
sailors
Successfully run. Total query runtime: 137 msec. 4 rows affected.
Total rows: 1 of 1 Query complete 00:00:00.952
Ln 249, Col 17



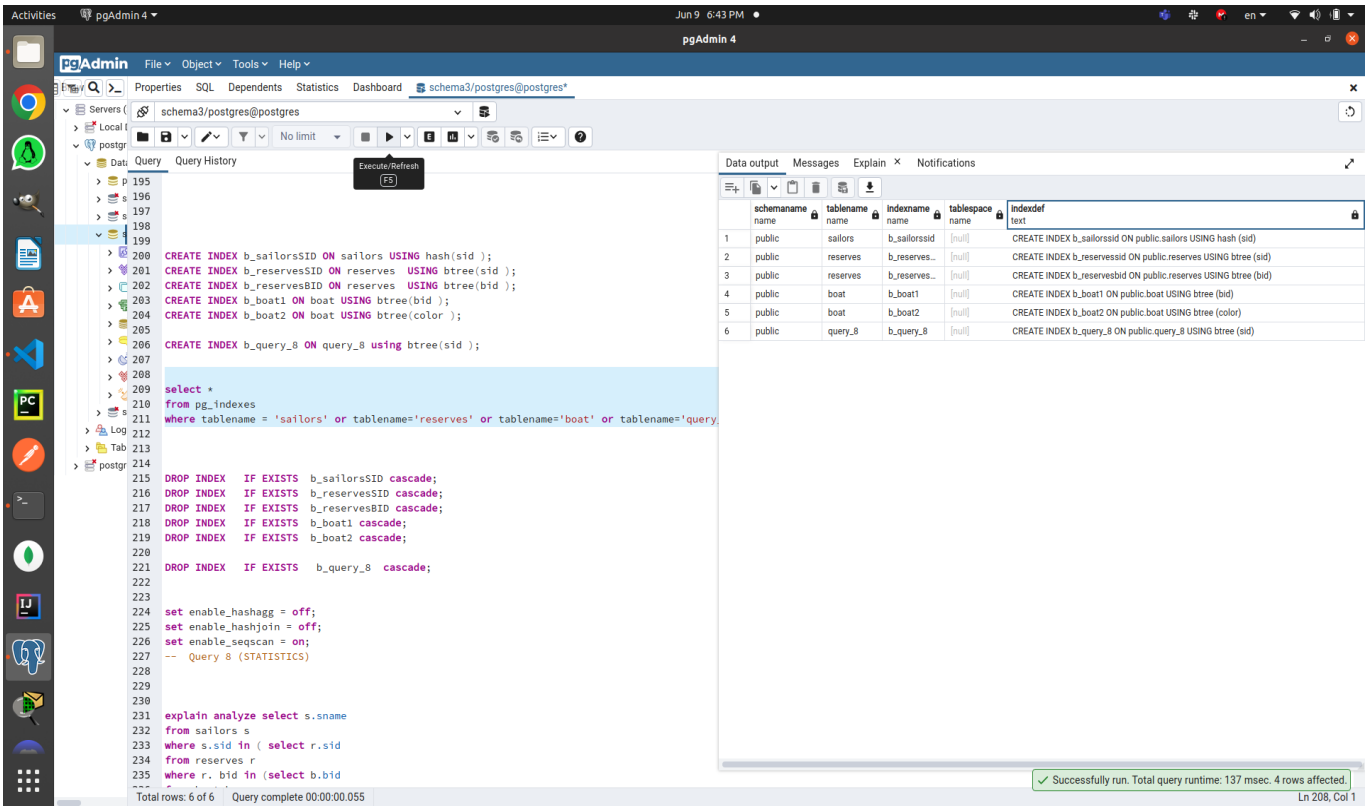
Explanation :

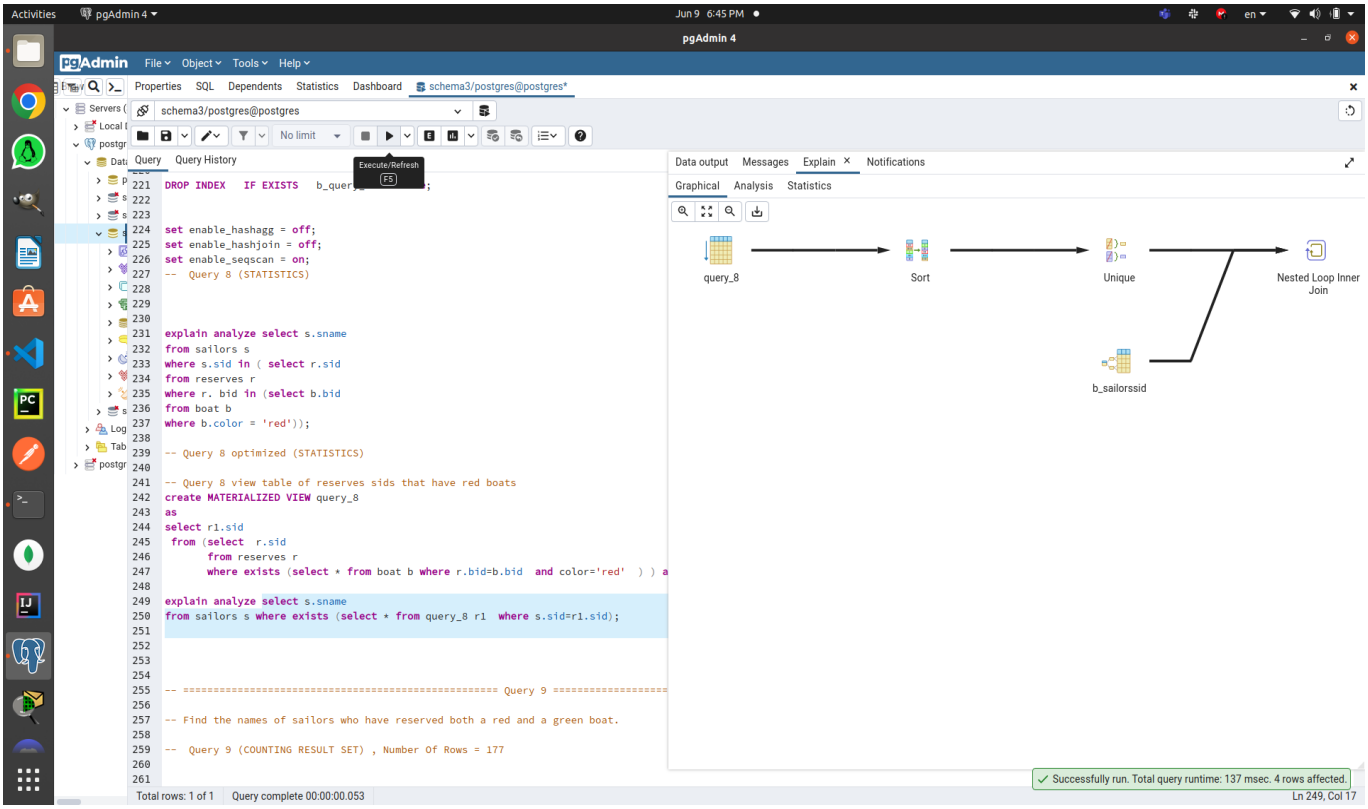
- Metrics :

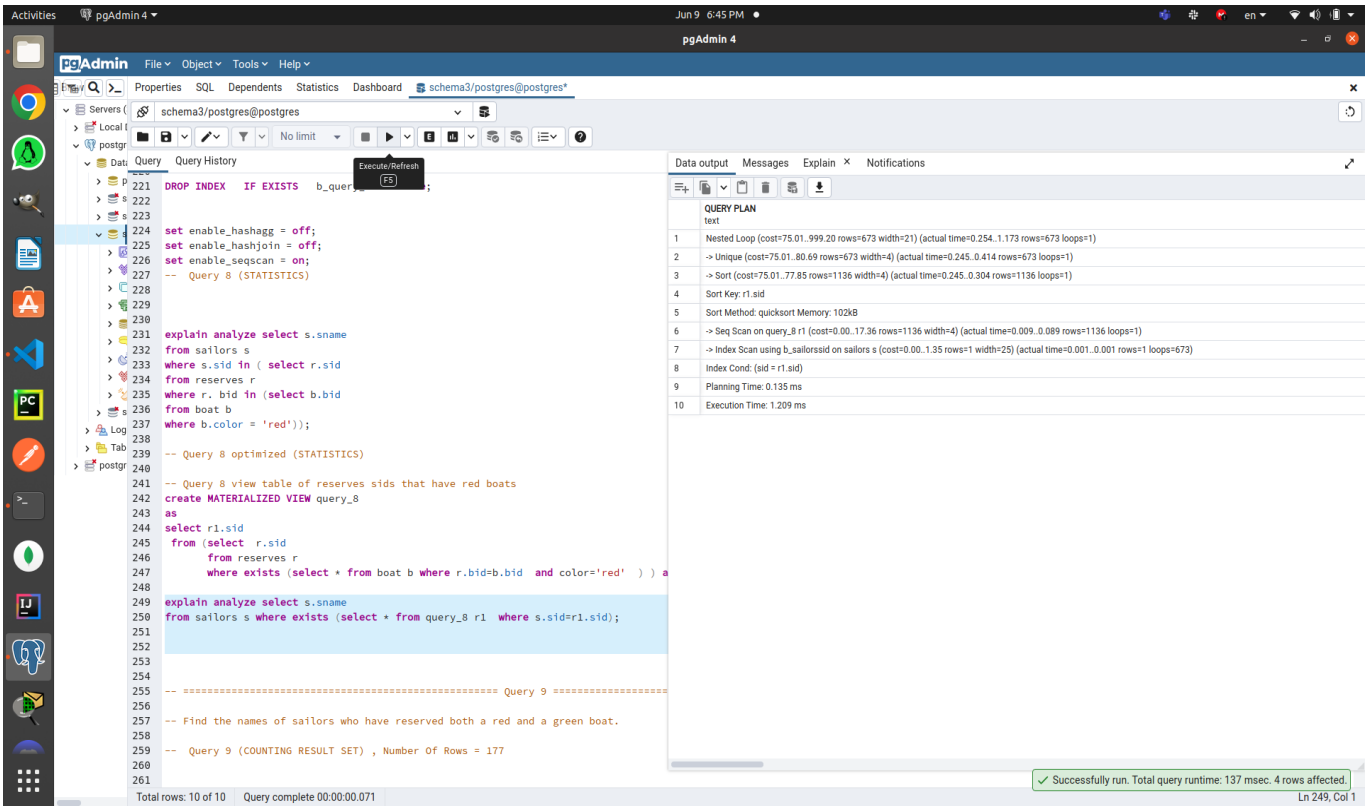
  Execution Time : 859.141 ms Total Expected Cost : 10005517867.87

- 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 THEM ALL) .
- This happened because the Query Optimizer didnt used it from the first 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.
- Because there where no Aggregation used and was not low selectivity Query so it was not helpful to the performance.
- The plan performs a (Bit map index scan) on all indices and then to (Bit map heap scan) to select relevant rows .

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


pgAdmin 4
Properties SQL Dependents Statistics Dashboard schema3/postgres@postgres*
schema3/postgres@postgres
Query Query History
Execute/Refresh
F5
CREATE INDEX b_sailorsSID ON sailors USING hash(sid);
CREATE INDEX b_reservesSID ON reserves USING btree(sid);
CREATE INDEX b_reservesBID ON reserves USING btree(bid);
CREATE INDEX b_boat1 ON boat USING btree(bid);
CREATE INDEX b_boat2 ON boat USING btree(color);
CREATE INDEX b_query_8 ON query_8 using btree(sid);
select *
from pg_indexes
where tablename = 'sailors' or tablename='reserves' or tablename='boat' or tablename='query_
DROP INDEX IF EXISTS b_sailorsSID cascade;
DROP INDEX IF EXISTS b_reservesSID cascade;
DROP INDEX IF EXISTS b_reservesBID cascade;
DROP INDEX IF EXISTS b_boat1 cascade;
DROP INDEX IF EXISTS b_boat2 cascade;
DROP INDEX IF EXISTS b_query_8 cascade;
set enable_hashagg = off;
set enable_hashjoin = off;
set enable_seqscan = on;
-- Query 8 (STATISTICS)
explain analyze select s.sname
from sailors s
where s.sid in (select r.sid
from reserves r
where r. bid in (select b.bid
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 btree (sid)
3 public reserves b_reserves... [null] CREATE INDEX b_reservesbid ON public.reserves USING btree (bid)
4 public boat b_boat1 [null] CREATE INDEX b_boat1 ON public.boat USING btree (bid)
5 public boat b_boat2 [null] CREATE INDEX b_boat2 ON public.boat USING btree (color)
6 public query_8 b_query_8 [null] CREATE INDEX b_query_8 ON public.query_8 USING btree (sid)
Successfully run. Total query runtime: 137 msec. 4 rows affected.
Total rows: 6 of 6 Query complete 00:00:00.055
Ln 208, Col 1


pgAdmin 4
Properties SQL Dependents Statistics Dashboard schema3/postgres@postgres*
schema3/postgres@postgres
Query Query History
Execute/Refresh
F5
DROP INDEX IF EXISTS b_query_8 cascade;
set enable_hashagg = off;
set enable_hashjoin = off;
set enable_seqscan = on;
-- Query 8 (STATISTICS)
explain analyze select s.sname
from sailors s
where s.sid in (select r.sid
from reserves r
where r. bid in (select b.bid
from boat b
where b.color = 'red'));
-- Query 8 optimized (STATISTICS)
-- Query 8 view table of reserves sids that have red boats
create MATERIALIZED VIEW query_8
as
select r1.sid
from (select r.sid
from reserves r
where exists (select * from boat b where r.bid=b.bid and color='red')) a
explain analyze select s.sname
from sailors s where exists (select * from query_8 r1 where s.sid=r1.sid);
-- Find the names of sailors who have reserved both a red and a green boat.
-- Query 9 (COUNTING RESULT SET) , Number Of Rows = 177
Data output Messages Explain Notifications
Graphical Analysis Statistics
query_8
Sort
Unique
Nested Loop Inner Join
b_sailorssid
Successfully run. Total query runtime: 137 msec. 4 rows affected.
Total rows: 1 of 1 Query complete 00:00:00.053
Ln 249, Col 17



Explanation :

- Metrics :

  Execution Time : 1.209 ms Total Expected Cost : 999.20

- The Query Planner used the Merge join by using both the ZigZag join and the Hash based algorithm together which improved the Execution time and the expected cost way more better which made the join in O(n log n).

- The Mix indices helped in the performance it decreased the Execution Time and Expected Cost .