Query 7

- Find the names of sailors who have reserved boat 103.

Note !

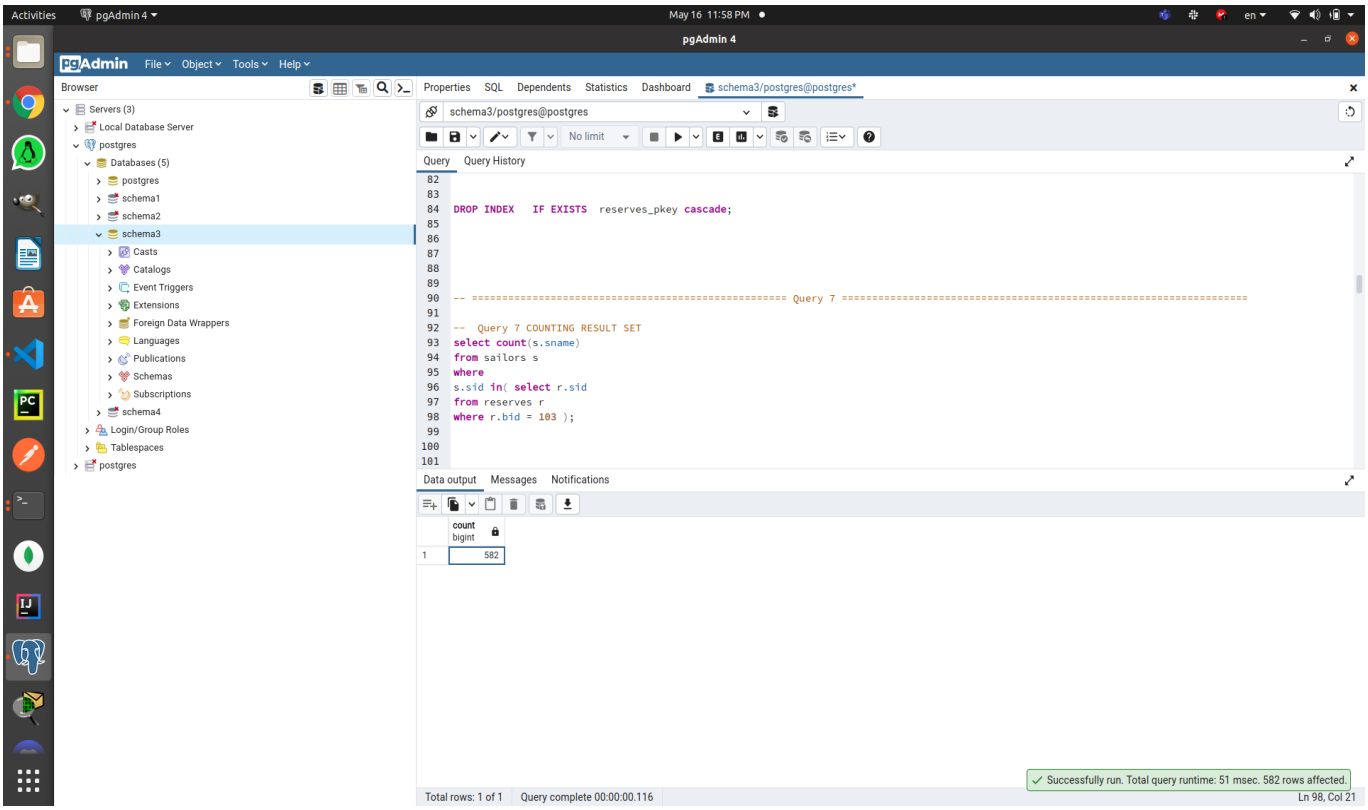
- Flags Hashjoin and HashAgg here where disabled for future after many trials and errors , I've discovered the best way to show the difference interms of the cost and to beat the Postgres Query Optimizer Algorithm to be able to show indices effect and cost differenes .
- The Execution time was changing 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 = 103 );
```

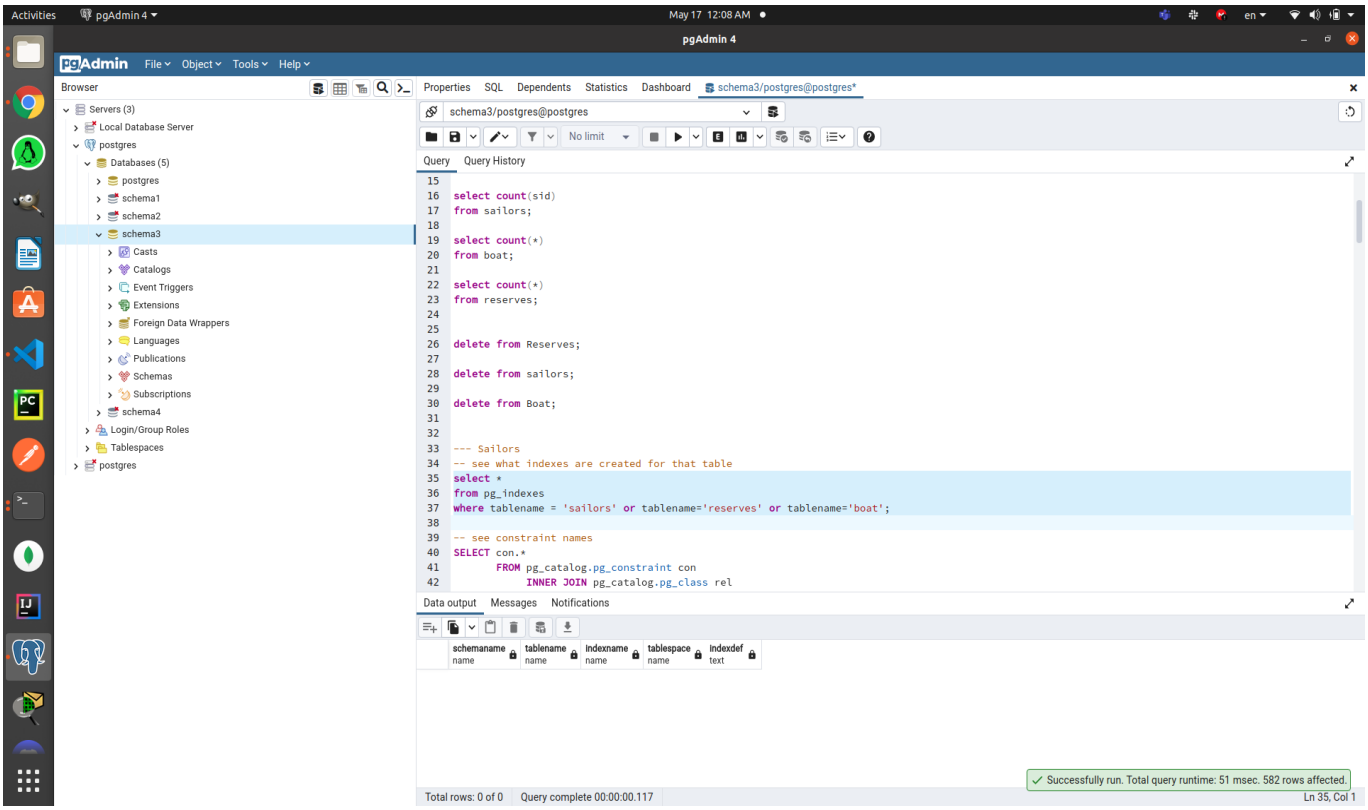
Result Set

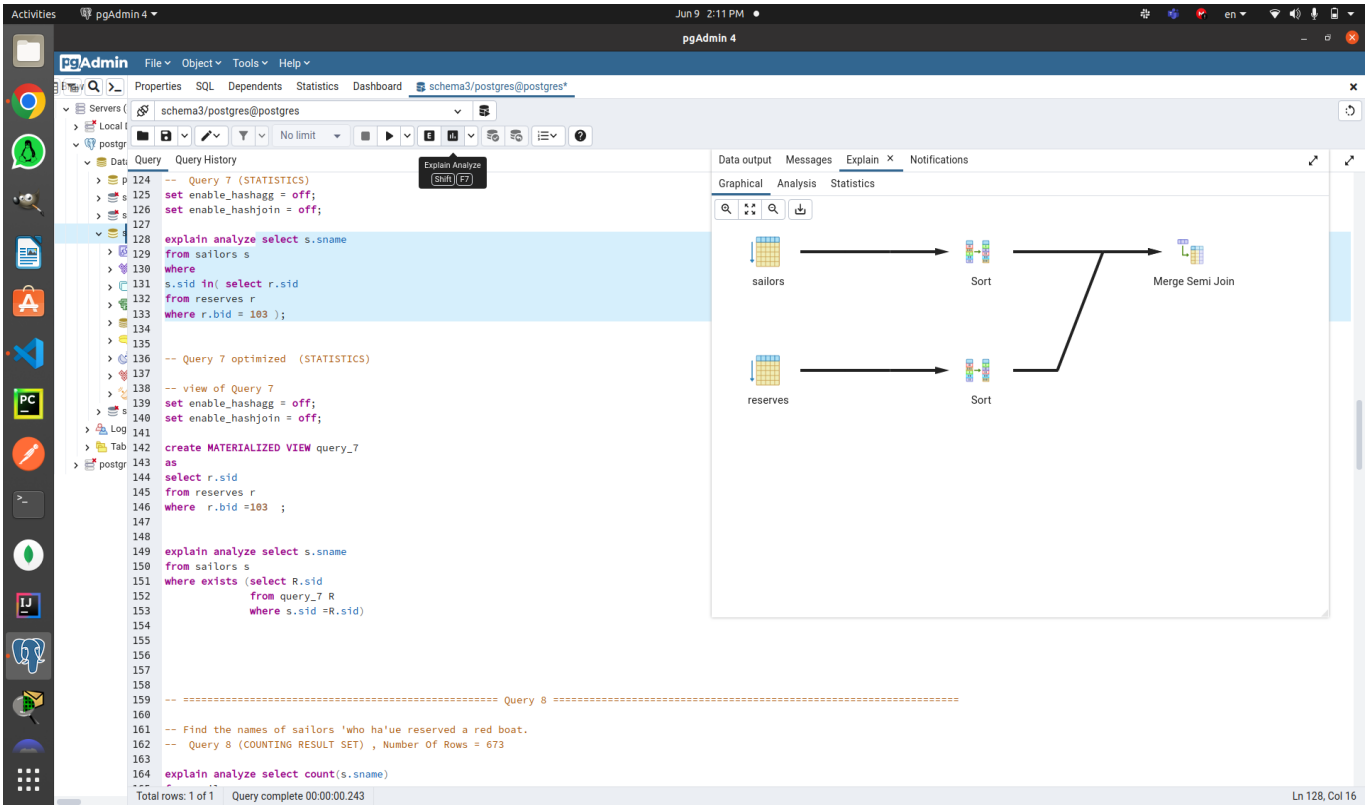
- 582 Rows

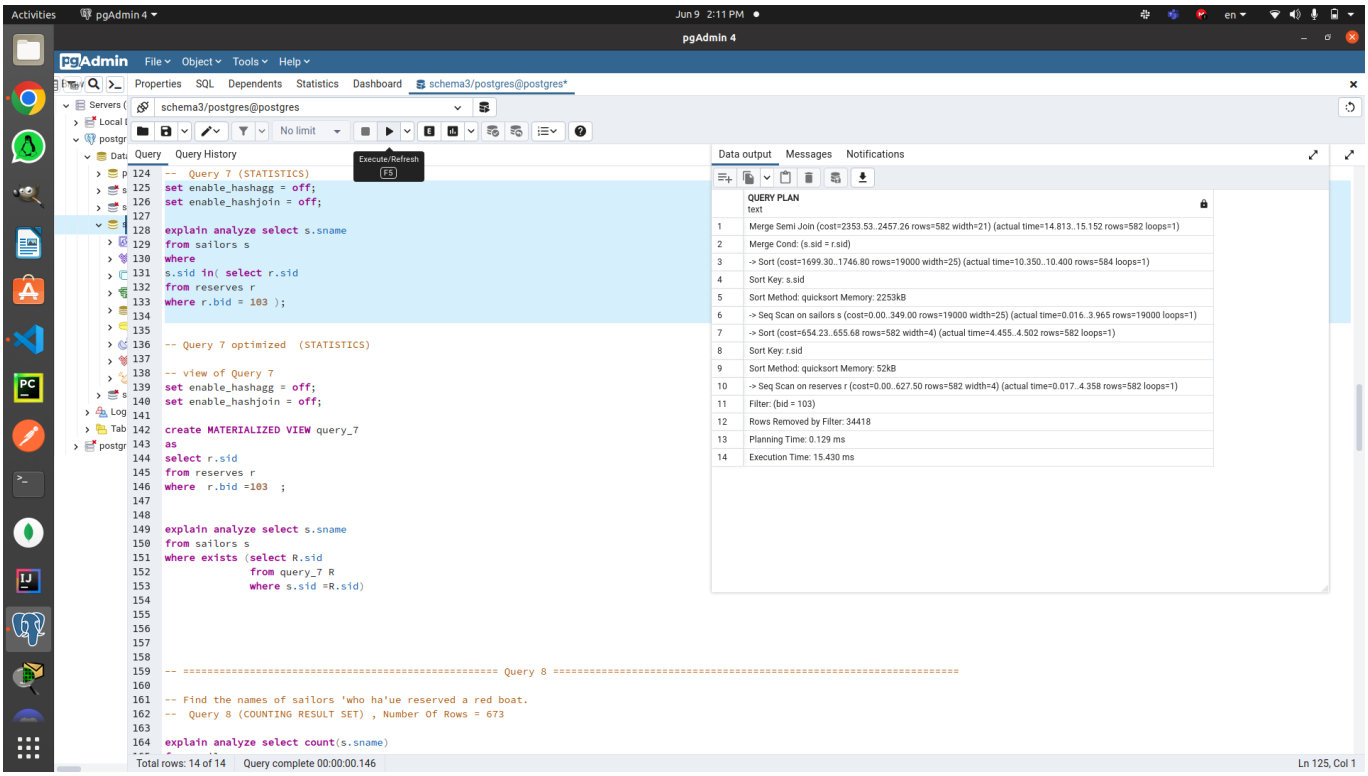


Report

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
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 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 2:11 PM
pgAdmin 4
File Object Tools Help
Properties SQL Dependents Statistics Dashboard schema3/postgres@postgres*
schema3/postgres@postgres
No limit
Explain Analyze Shift F7
Query Query History
-- Query 7 (STATISTICS)
set enable_hashagg = off;
set enable_hashjoin = off;
explain analyze select s.sname
from sailors s
where
s.sid in(select r.sid
from reserves r
where r.bid = 103);
-- Query 7 optimized (STATISTICS)
-- view of Query 7
set enable_hashagg = off;
set enable_hashjoin = off;
create MATERIALIZED VIEW query_7
as
select r.sid
from reserves r
where r.bid =103 ;
explain analyze select s.sname
from sailors s
where exists (select R.sid
from query_7 R
where s.sid =R.sid)
-- ========== Query 8 ==========
-- Find the names of sailors 'who ha'ue reserved a red boat.
-- Query 8 (COUNTING RESULT SET) , Number Of Rows = 673
explain analyze select count(s.sname)
Data output Messages Explain Notifications
Graphical Analysis Statistics
sailors
Sort
Merge Semi Join
reserves
Sort
Total rows: 1 of 1
Query complete 00:00:00.243
Ln 128, Col 16



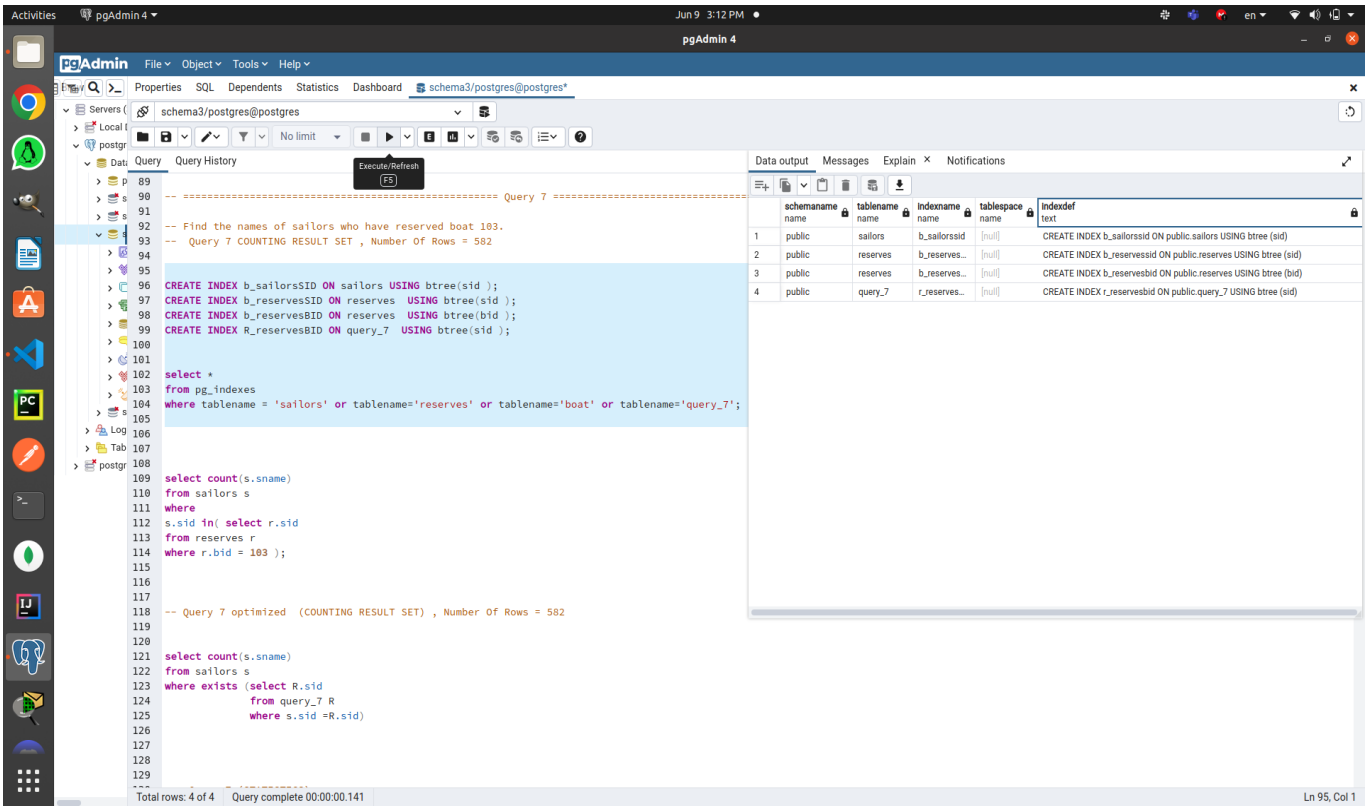
Explanation :

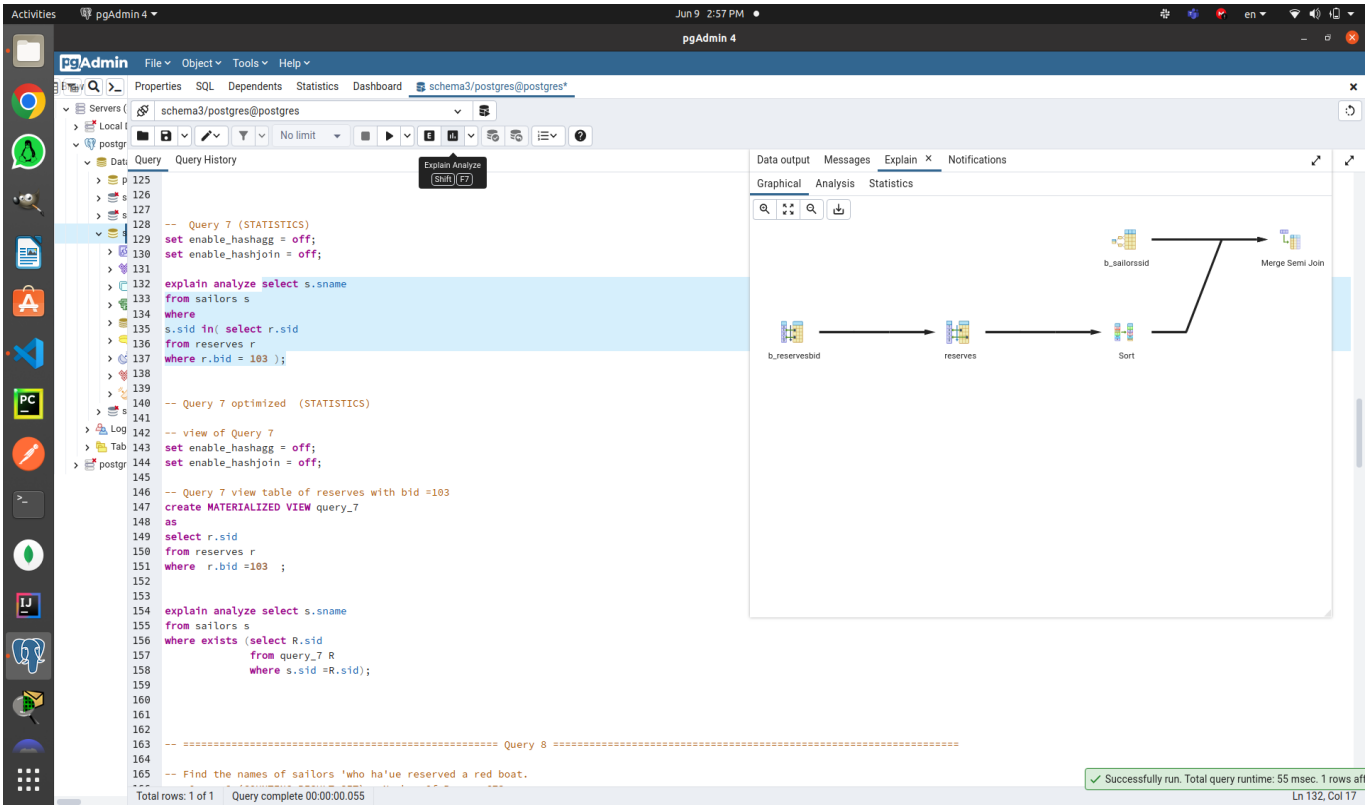
- Metrics :

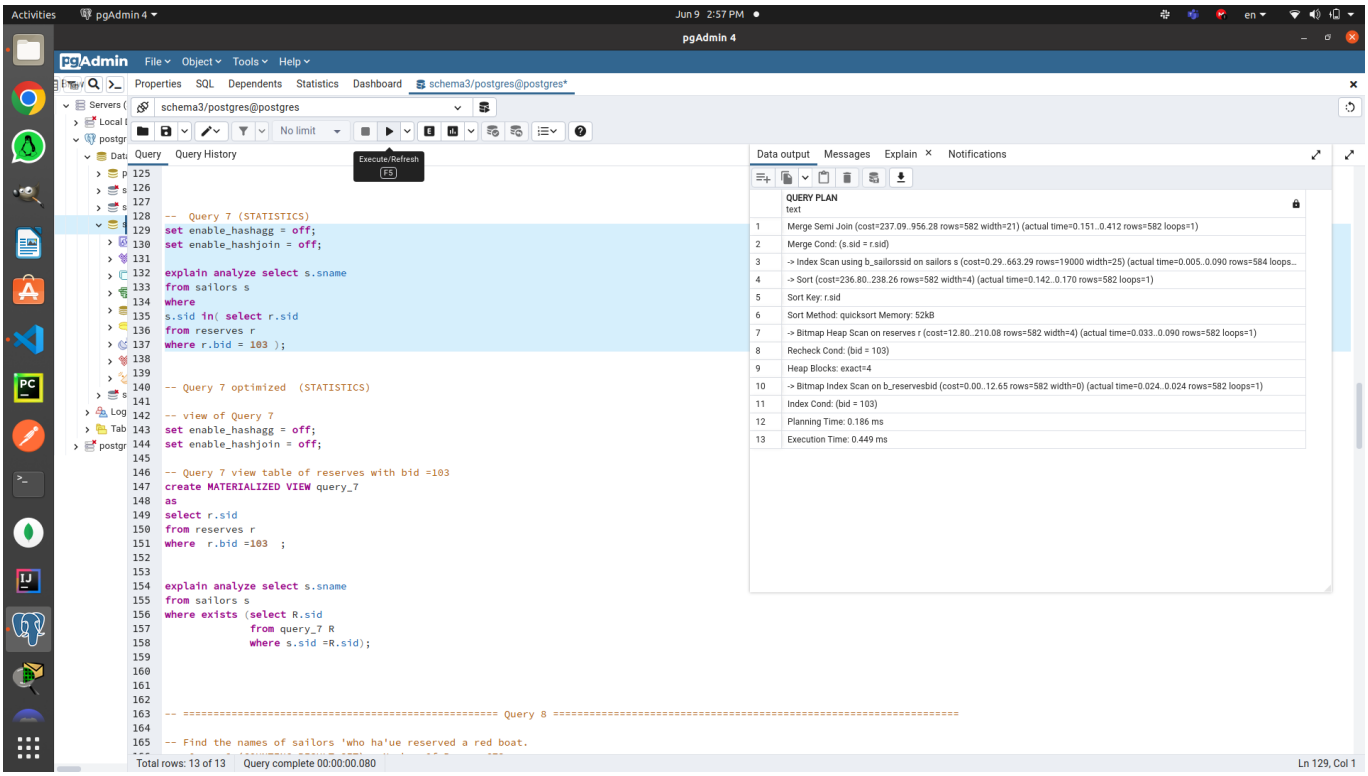
  Execution Time : 15.430 ms Total Expected Cost : 2457.26

  - 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
Data output Messages Explain Notifications
-- Find the names of sailors who have reserved boat 103.
-- Query 7 COUNTING RESULT SET , Number Of Rows = 582
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 R_reservesBID ON query_7 USING btree(sid);
select *
from pg_indexes
where tablename = 'sailors' or tablename='reserves' or tablename='boat' or tablename='query_7';
select count(s.sname)
from sailors s
where
s.sid in(select r.sid
from reserves r
where r.bid = 103);
-- Query 7 optimized (COUNTING RESULT SET) , Number Of Rows = 582
select count(s.sname)
from sailors s
where exists (select R.sid
from query_7 R
where s.sid =R.sid)
Total rows: 4 of 4 Query complete 00:00:00.141
Ln 95, 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_sailorssid
Merge Semi Join
b_reservesbid
reserves
Sort
-- Query 7 (STATISTICS)
set enable_hashagg = off;
set enable_hashjoin = off;
explain analyze select s.sname
from sailors s
where
s.sid in(select r.sid
from reserves r
where r.bid = 103);
-- Query 7 optimized (STATISTICS)
-- view of Query 7
set enable_hashagg = off;
set enable_hashjoin = off;
-- Query 7 view table of reserves with bid =103
create MATERIALIZED VIEW query_7
as
select r.sid
from reserves r
where r.bid =103 ;
explain analyze select s.sname
from sailors s
where exists (select R.sid
from query_7 R
where s.sid =R.sid);
-- Find the names of sailors 'who ha'ue reserved a red boat.
Successfully run. Total query runtime: 55 msec. 1 rows aff
Total rows: 1 of 1 Query complete 00:00:00.055
Ln 132, Col 17



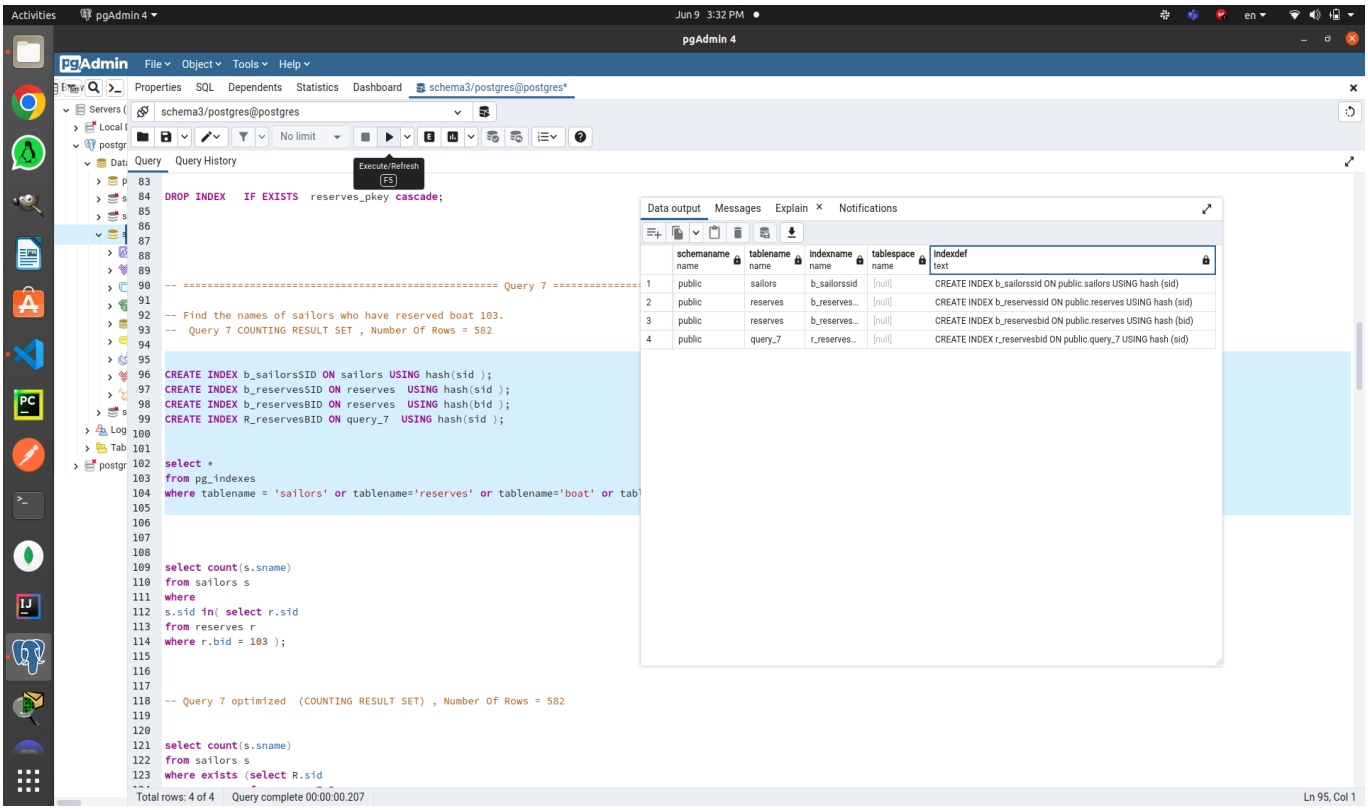
Explanation :

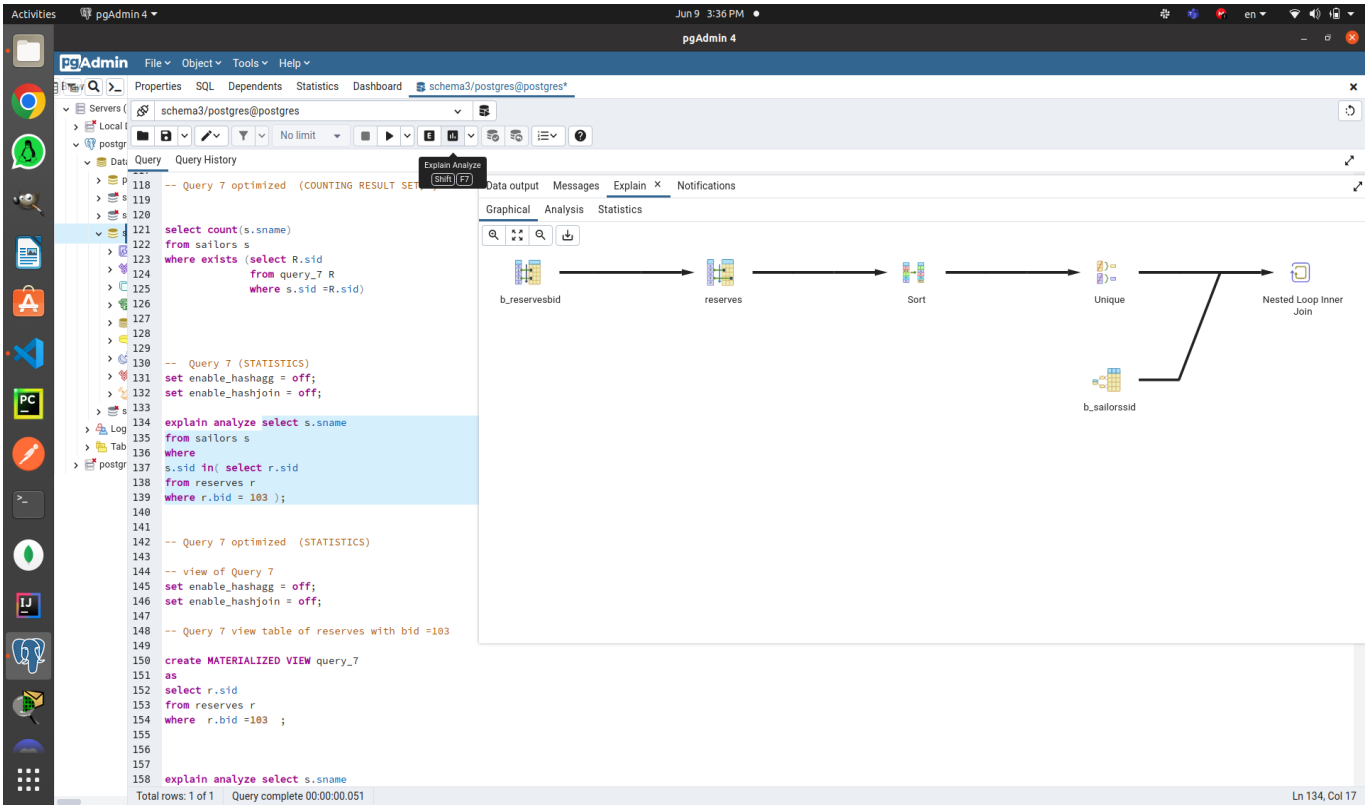
- Metrics :

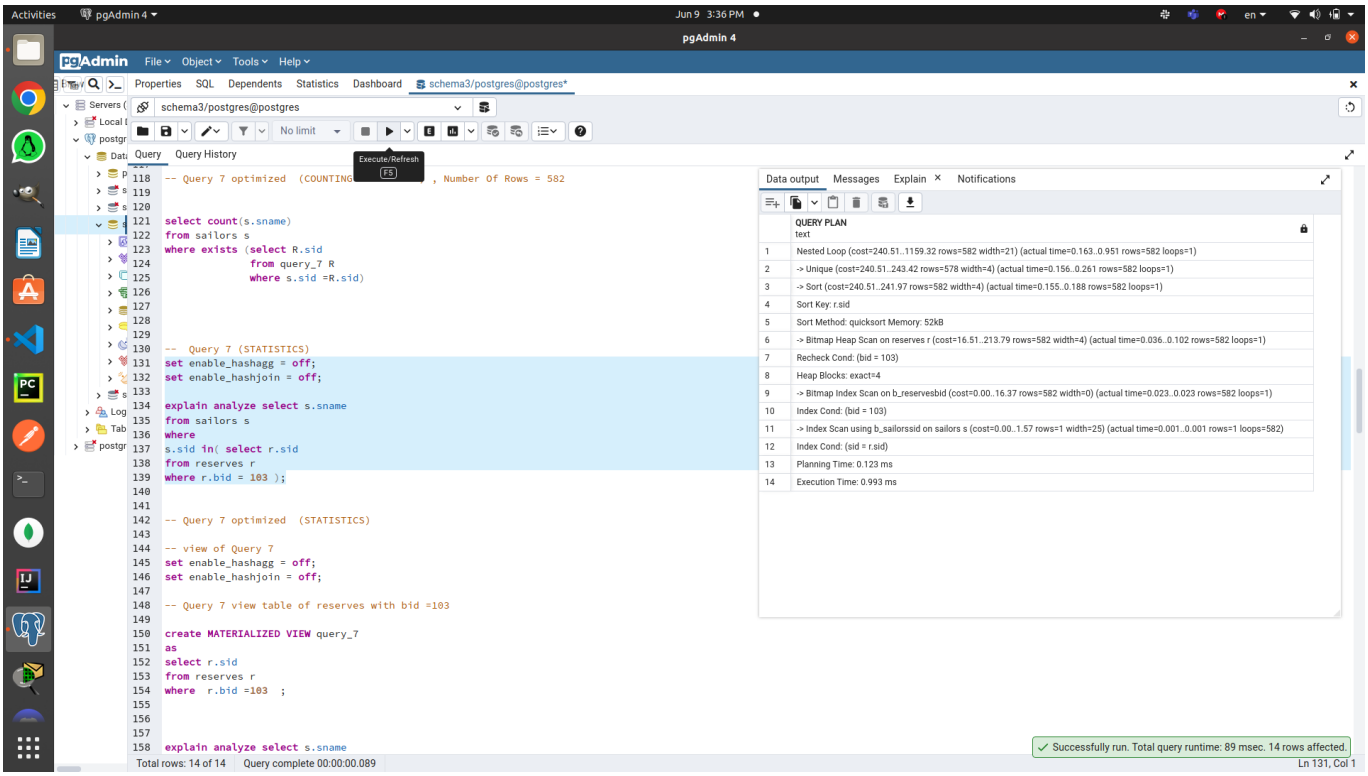
  Execution Time : 0.449 ms Total Expected Cost : 956.28

- 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 every condition of Joining in the Query the Merge Join used index scan using ZigZag algorithm .
- The Where clause condition on the (bid = 103) used B-tree which is more better as (bid = 103) are repetable and sorted after each other at the leaves of the b-tree with O(Log n) performance on all at one time .

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
DROP INDEX IF EXISTS reserves_pkey cascade;
-- Find the names of sailors who have reserved boat 103.
-- Query 7 COUNTING RESULT SET , Number Of Rows = 582
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 R_reservesBID ON query_7 USING hash(sid);
select *
from pg_indexes
where tablename = 'sailors' or tablename='reserves' or tablename='boat' or tab
select count(s.sname)
from sailors s
where
s.sid in(select r.sid
from reserves r
where r.bid = 103);
-- Query 7 optimized (COUNTING RESULT SET) , Number Of Rows = 582
select count(s.sname)
from sailors s
where exists (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 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 query_7 r_reserves... [null] CREATE INDEX r_reservesbid ON public.query_7 USING hash (sid)
Total rows: 4 of 4 Query complete 00:00:00.207
Ln 95, Col 1


pgAdmin 4
Properties SQL Dependents Statistics Dashboard schema3/postgres@postgres*
schema3/postgres@postgres
Query Query History
Explain Analyze Shift F7
-- Query 7 optimized (COUNTING RESULT SET
select count(s.sname)
from sailors s
where exists (select R.sid
from query_7 R
where s.sid =R.sid)
-- Query 7 (STATISTICS)
set enable_hashagg = off;
set enable_hashjoin = off;
explain analyze select s.sname
from sailors s
where
s.sid in(select r.sid
from reserves r
where r.bid = 103);
-- Query 7 optimized (STATISTICS)
-- view of Query 7
set enable_hashagg = off;
set enable_hashjoin = off;
-- Query 7 view table of reserves with bid =103
create MATERIALIZED VIEW query_7
as
select r.sid
from reserves r
where r.bid =103 ;
explain analyze select s.sname
Data output Messages Explain Notifications
Graphical Analysis Statistics
b_reservesbid
reserves
Sort
Unique
Nested Loop Inner Join
b_sailorssid
Total rows: 1 of 1 Query complete 00:00:00.051
Ln 134, Col 17



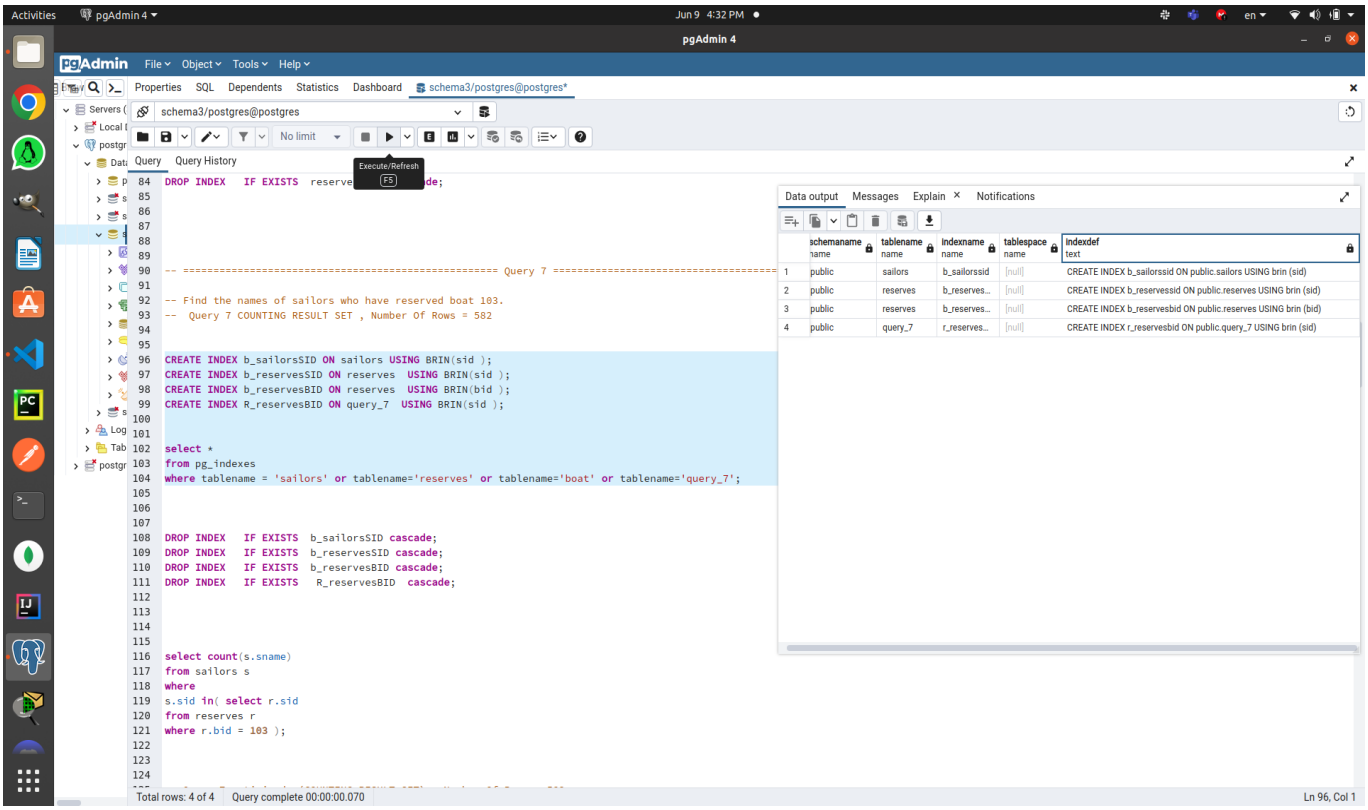
Explanation :

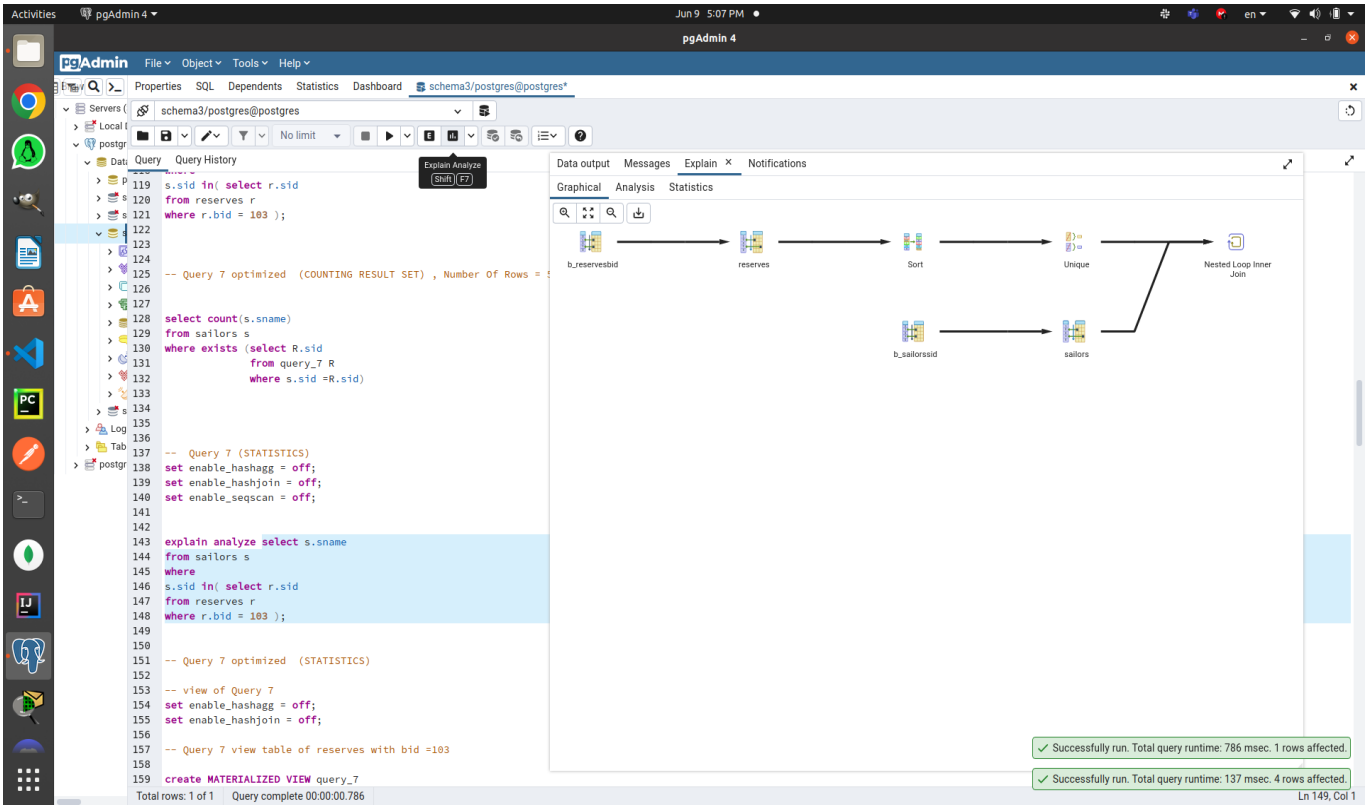
- Metrics :

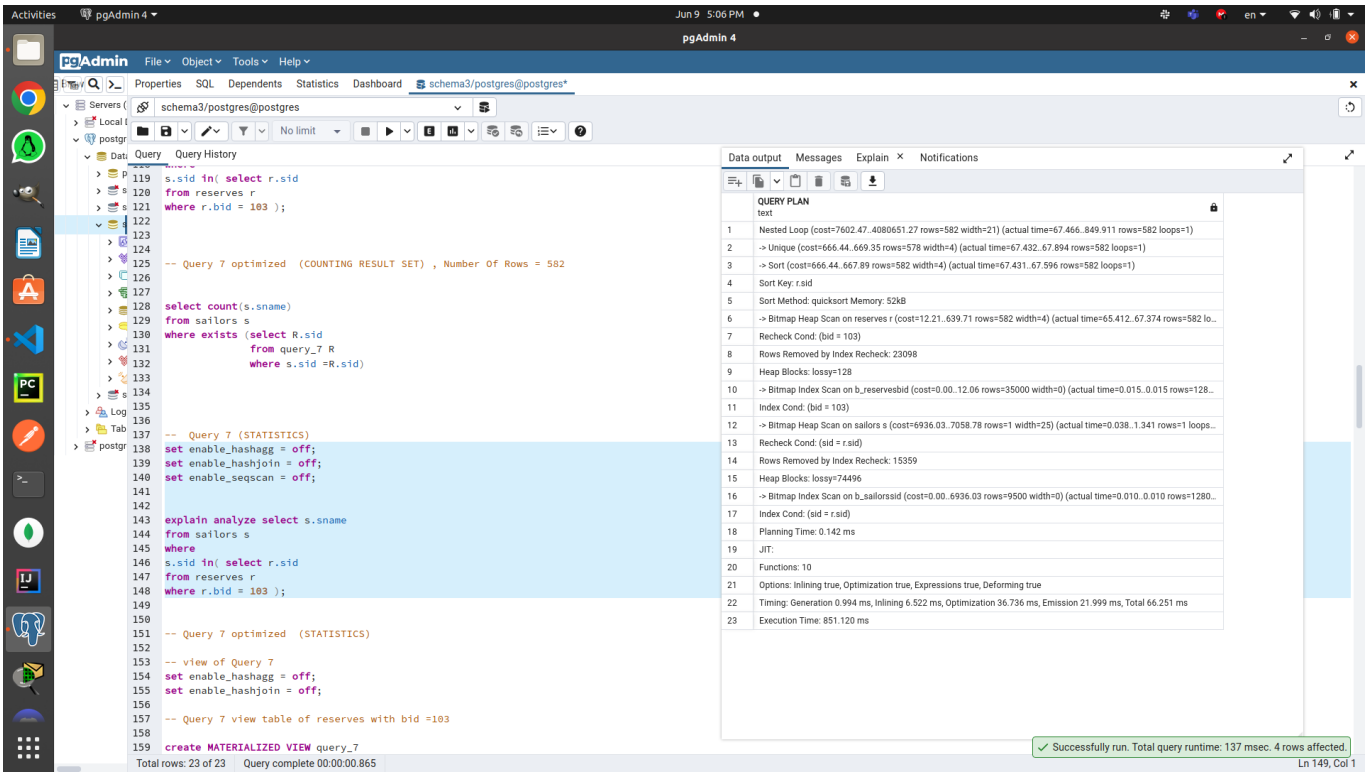
  Execution Time : 0.993 ms Total Expected Cost : 1159.32

- 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 .
- 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 (sid=s.sid) which approximatly maded to be O(n) performance .
- Here it showed the improvement due to for condition of (bid = 103) with performance of O(1) .

4. given query with BRIN indices only :


Activities
pgAdmin 4
Jun 9 4:32 PM
pgAdmin 4
File Object Tools Help
Properties SQL Dependents Statistics Dashboard schema3/postgres@postgres*
schema3/postgres@postgres
Query Query History
Execute/Refresh F5
-- Find the names of sailors who have reserved boat 103.
-- Query 7 COUNTING RESULT SET , Number Of Rows = 582
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 R_reservesBID ON query_7 USING BRIN(sid);
select *
from pg_indexes
where tablename = 'sailors' or tablename='reserves' or tablename='boat' or tablename='query_7';
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;
select count(s.sname)
from sailors s
where
s.sid in(select r.sid
from reserves r
where r.bid = 103);
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 query_7 r_reserves... [null] CREATE INDEX r_reservesbid ON public.query_7 USING brin (sid)
Total rows: 4 of 4 Query complete 00:00:00.070
Ln 96, Col 1


Activities
pgAdmin 4
Jun 9 5:07 PM
pgAdmin 4
File Object Tools Help
Properties SQL Dependents Statistics Dashboard schema3/postgres@postgres*
schema3/postgres@postgres
Query Query History
Explain Analyze Shift F7
s.sid in(select r.sid
from reserves r
where r.bid = 103);
-- Query 7 optimized (COUNTING RESULT SET) , Number Of Rows =
select count(s.sname)
from sailors s
where exists (select R.sid
from query_7 R
where s.sid =R.sid)
-- Query 7 (STATISTICS)
set enable_hashagg = off;
set enable_hashjoin = off;
set enable_seqscan = off;
explain analyze select s.sname
from sailors s
where
s.sid in(select r.sid
from reserves r
where r.bid = 103);
-- Query 7 optimized (STATISTICS)
-- view of Query 7
set enable_hashagg = off;
set enable_hashjoin = off;
-- Query 7 view table of reserves with bid =103
create MATERIALIZED VIEW query_7
Data output Messages Explain Notifications
Graphical Analysis Statistics
b_reservesbid
reserves
Sort
Unique
Nested Loop Inner Join
b_sailorssid
sailors
Successfully run. Total query runtime: 786 msec. 1 rows affected.
Successfully run. Total query runtime: 137 msec. 4 rows affected.
Total rows: 1 of 1 Query complete 00:00:00.786
Ln 149, Col 1



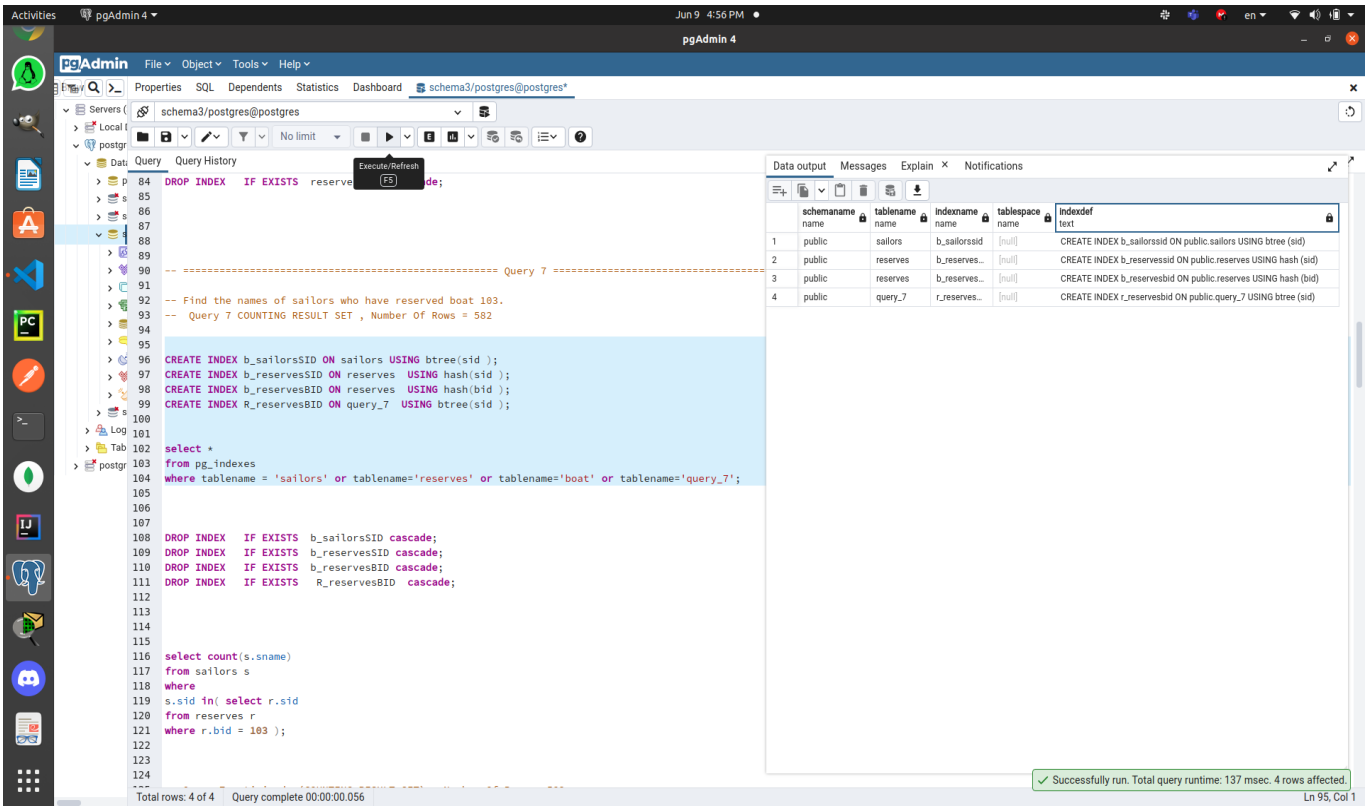
Explanation :

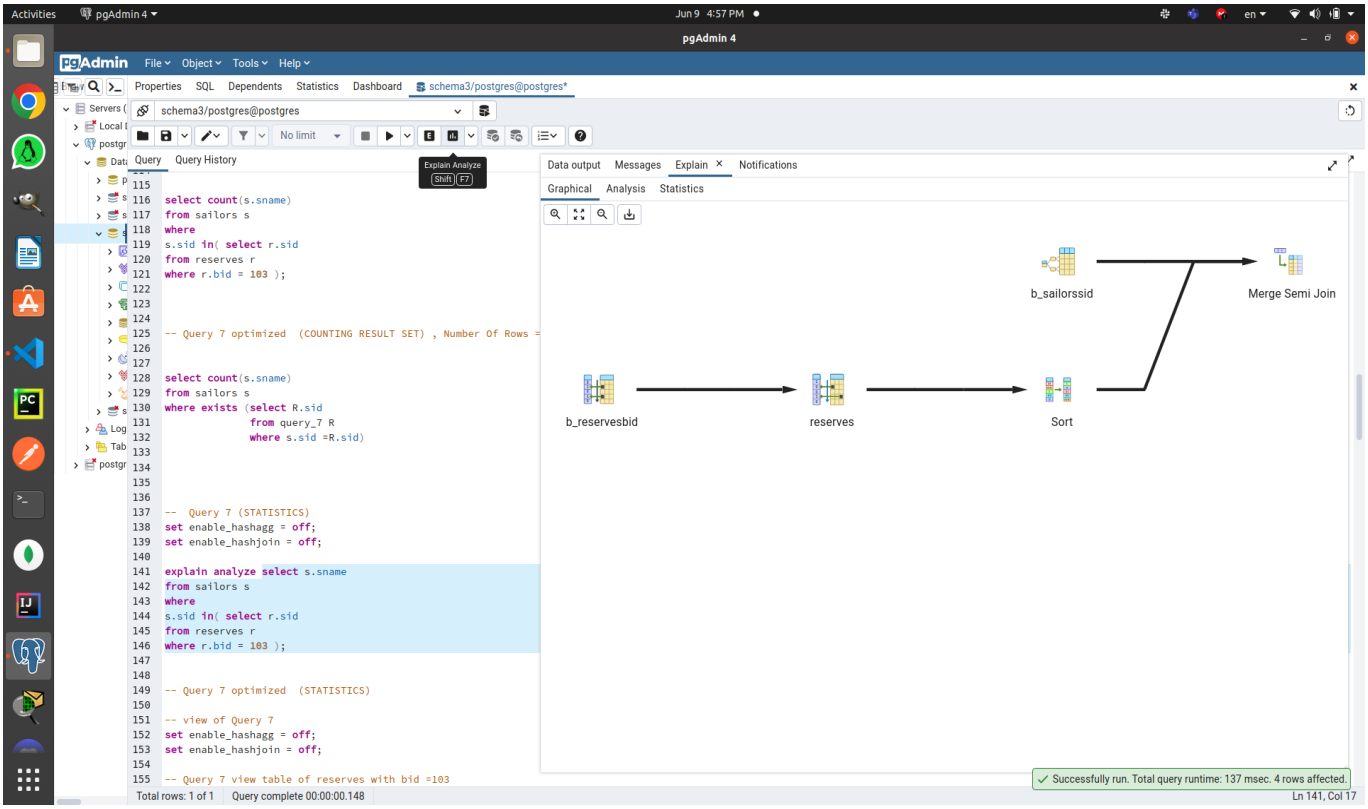
- Metrics :

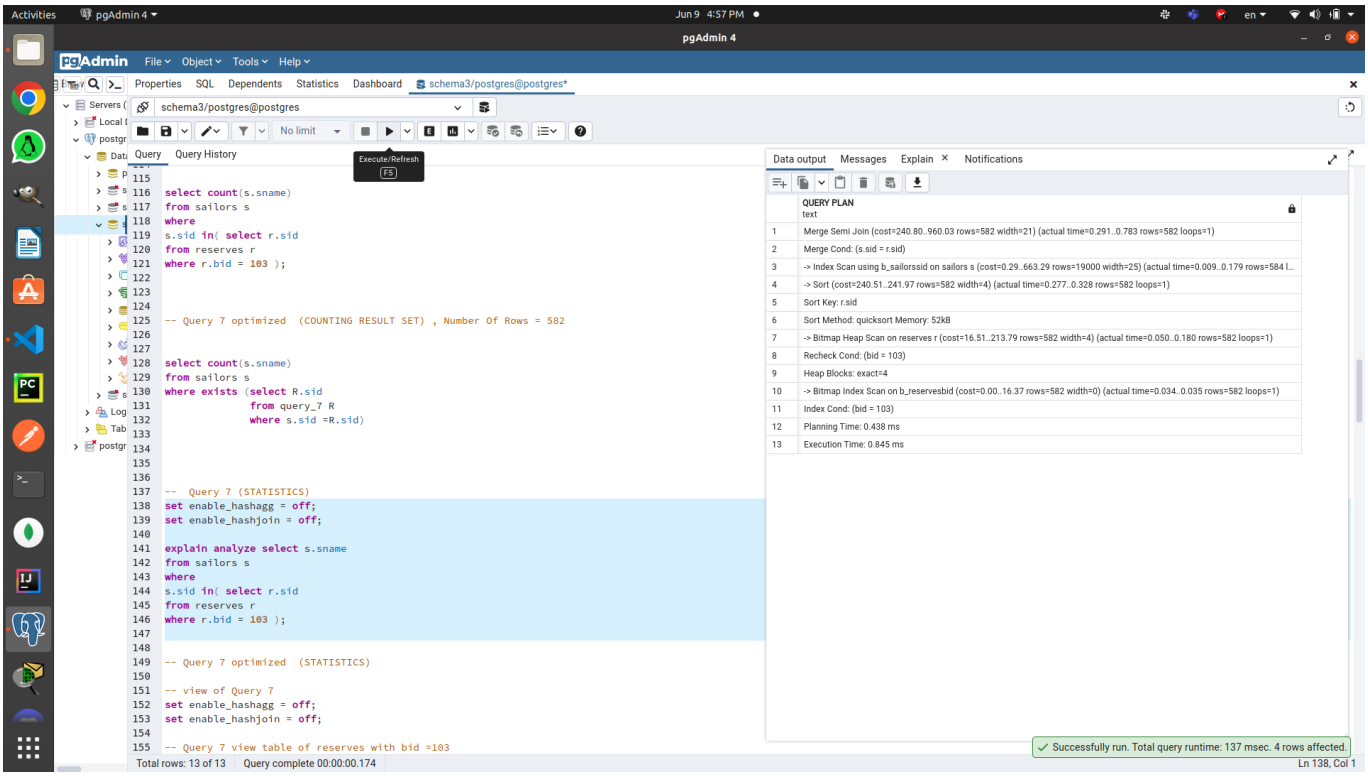
  Execution Time : 851.120 ms Total Expected Cost : 4080651.27

- 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
DROP INDEX IF EXISTS reserve de;
-- ========== Query 7 ==========
-- Find the names of sailors who have reserved boat 103.
-- Query 7 COUNTING RESULT SET , Number Of Rows = 582
CREATE INDEX b_sailorsSID ON sailors USING btree(sid);
CREATE INDEX b_reservesSID ON reserves USING hash(sid);
CREATE INDEX b_reservesBID ON reserves USING hash(bid);
CREATE INDEX R_reservesBID ON query_7 USING btree(sid);
select *
from pg_indexes
where tablename = 'sailors' or tablename='reserves' or tablename='boat' or tablename='query_7';
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;
select count(s.sname)
from sailors s
where
s.sid in(select r.sid
from reserves r
where r.bid = 103);
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 hash (sid)
3 public reserves b_reserves... [null] CREATE INDEX b_reservesbid ON public.reserves USING hash (bid)
4 public query_7 r_reserves... [null] CREATE INDEX r_reservesbid ON public.query_7 USING btree (sid)
Successfully run. Total query runtime: 137 msec. 4 rows affected.
Total rows: 4 of 4 Query complete 00:00:00.056
Ln 95, Col 1


pgAdmin 4
Properties SQL Dependents Statistics Dashboard schema3/postgres@postgres*
schema3/postgres@postgres
Query Query History
Explain Analyze Shift F7
select count(s.sname)
from sailors s
where
s.sid in(select r.sid
from reserves r
where r.bid = 103);
-- Query 7 optimized (COUNTING RESULT SET) , Number Of Rows =
select count(s.sname)
from sailors s
where exists (select R.sid
from query_7 R
where s.sid =R.sid)
-- Query 7 (STATISTICS)
set enable_hashagg = off;
set enable_hashjoin = off;
explain analyze select s.sname
from sailors s
where
s.sid in(select r.sid
from reserves r
where r.bid = 103);
-- Query 7 optimized (STATISTICS)
-- view of Query 7
set enable_hashagg = off;
set enable_hashjoin = off;
-- Query 7 view table of reserves with bid =103
Data output Messages Explain Notifications
Graphical Analysis Statistics
b_sailorssid
Merge Semi Join
b_reservesbid
reserves
Sort
Successfully run. Total query runtime: 137 msec. 4 rows affected.
Total rows: 1 of 1 Query complete 00:00:00.148
Ln 141, Col 17



Explanation :

- Metrics :

  Execution Time : 0.845 ms Total Expected Cost : 960.03

- 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).
- And It used the Hash indexed scan on bid =103.
- The Mix indices helped in the performance it decreased the Execution Time and Expected Cost .

Optimized Query

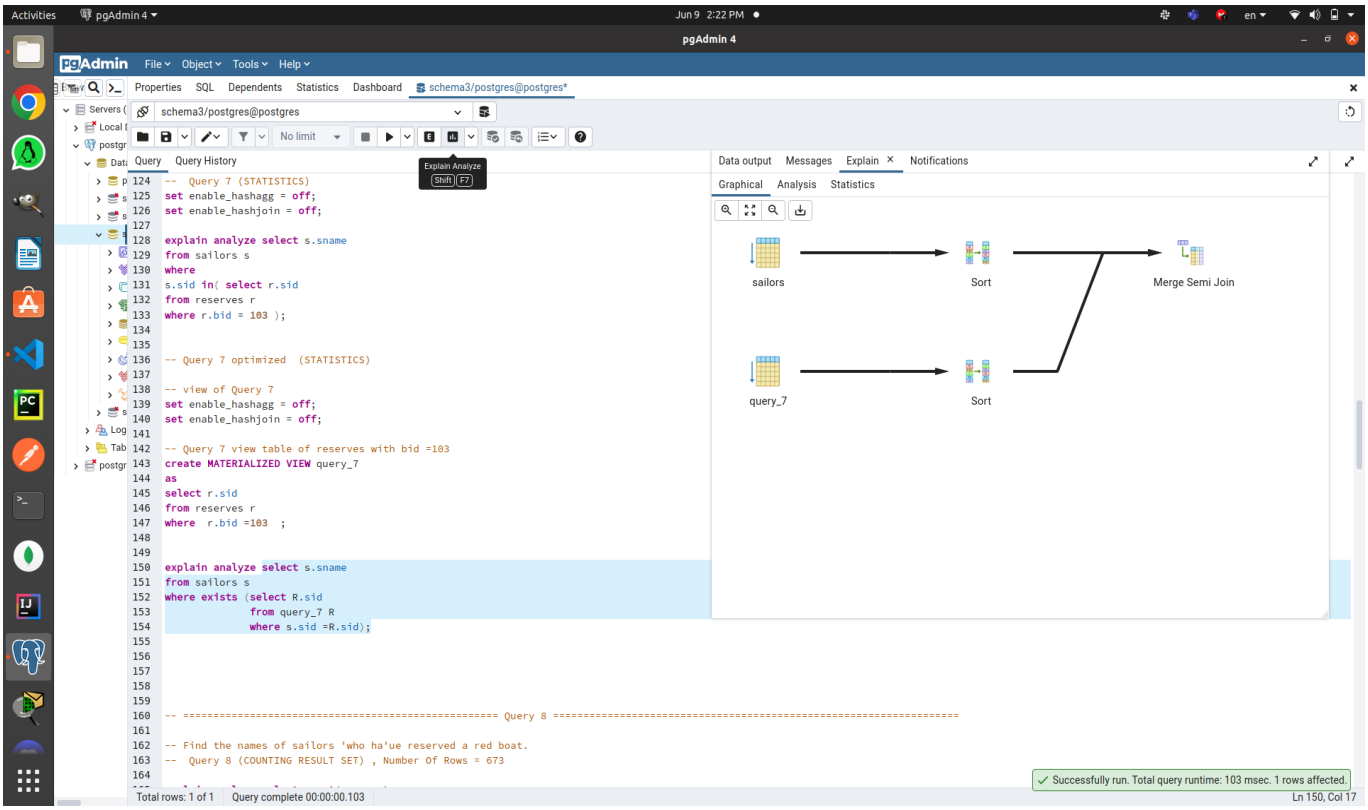
```
-- Query 7 view table of reserves with bid = 103

create MATERIALIZED VIEW query_7
as
select r.sid
from reserves r
where  r.bid =103  ;


select s.sname
from sailors s
where exists (select R.sid
              from query_7 R
              where s.sid =R.sid);
```

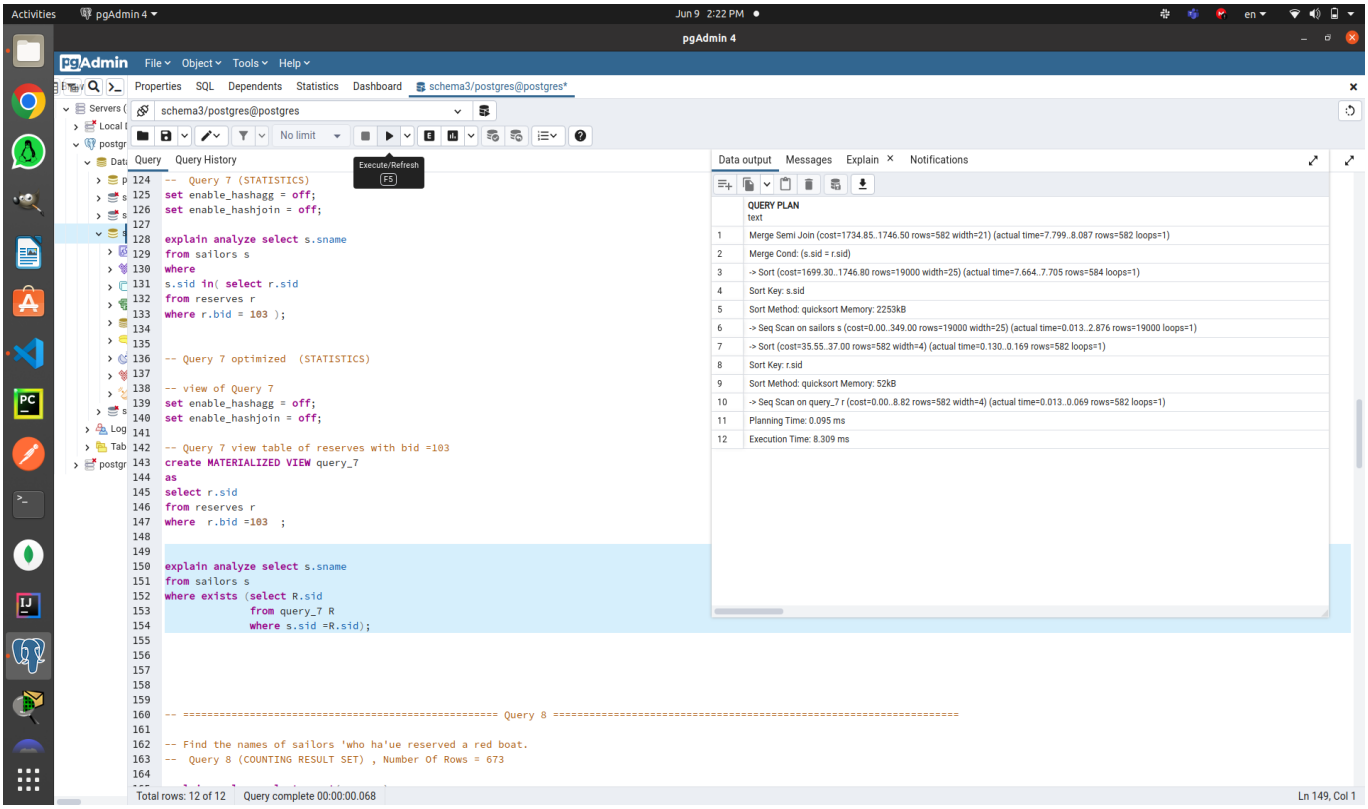
Result Set

- 582 Rows

Report

1. given query without an index :





Explanation :

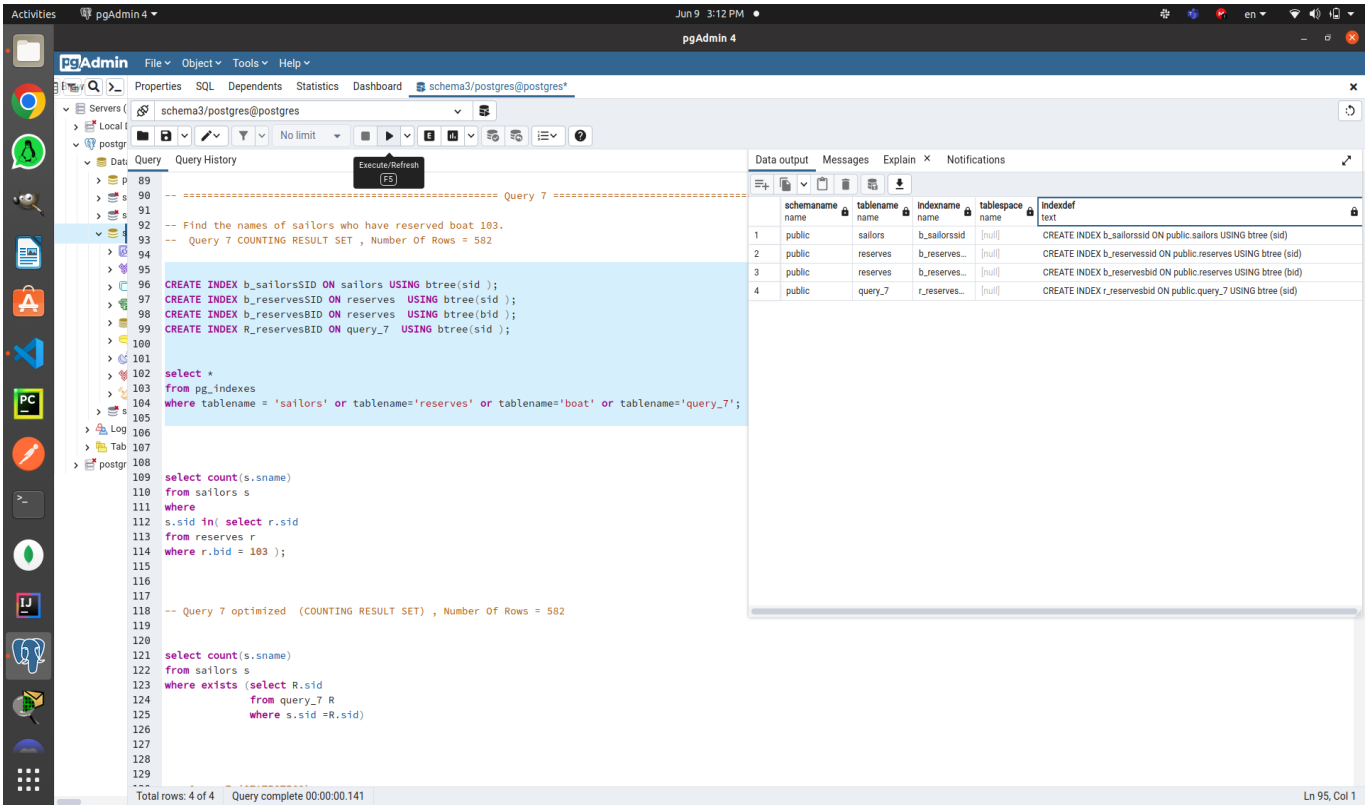
- Metrics :

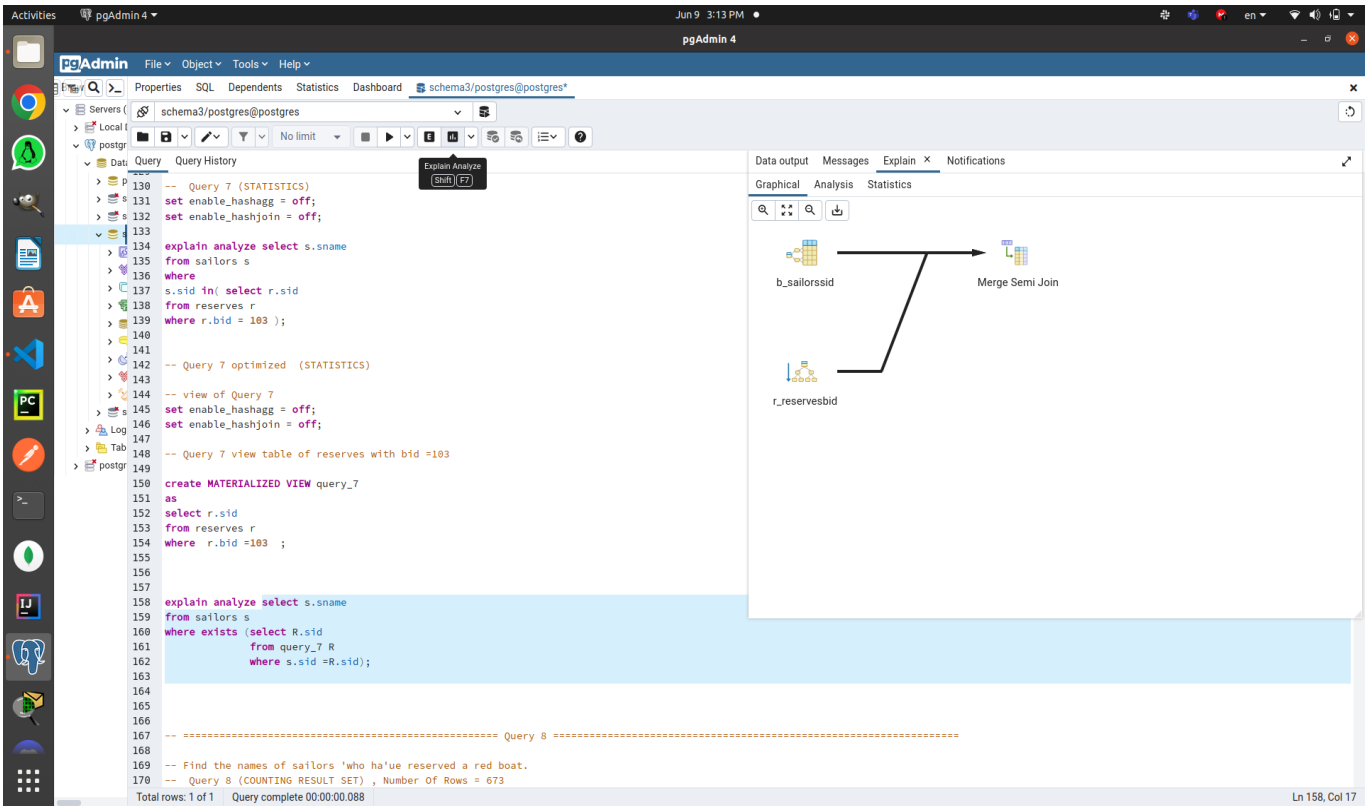
  Execution Time : 8.309 ms Total Expected Cost : 1746.50

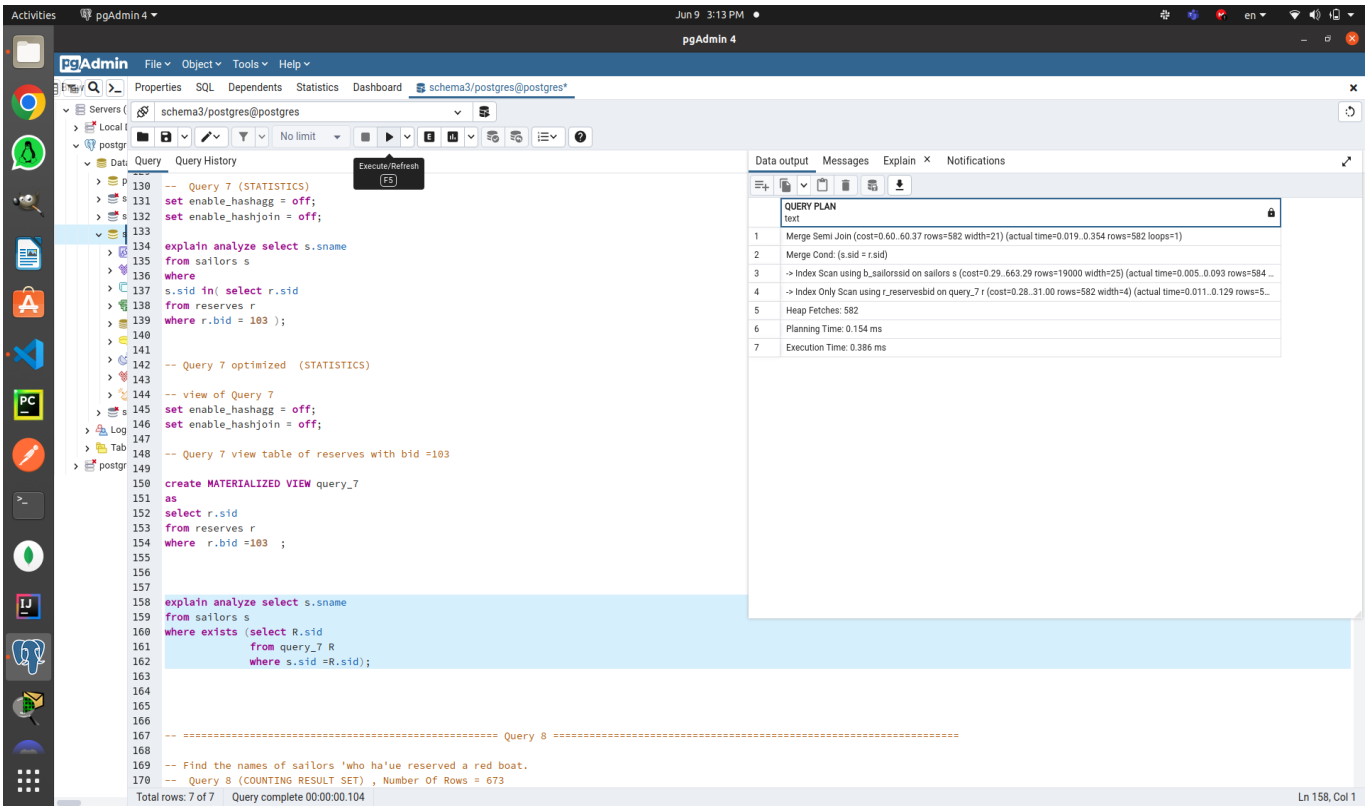
- Reason :

- This Query Improved in the Execution time and Expected Cost than the Original Query from 2475.26 to 1746.50 .
- 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 r.bid =103) 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 so it helped in the intermediate results.

2. given query with B+ trees indices only :







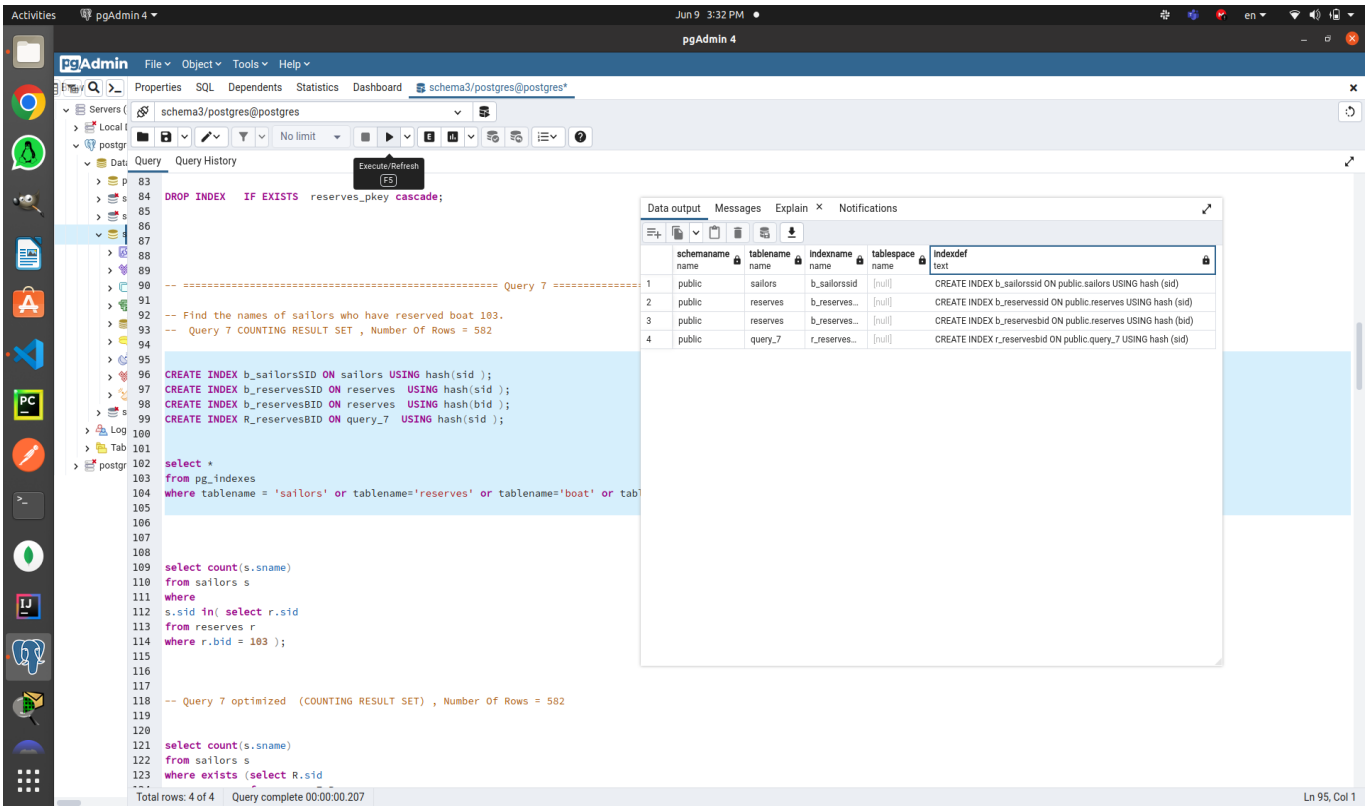
Explanation :

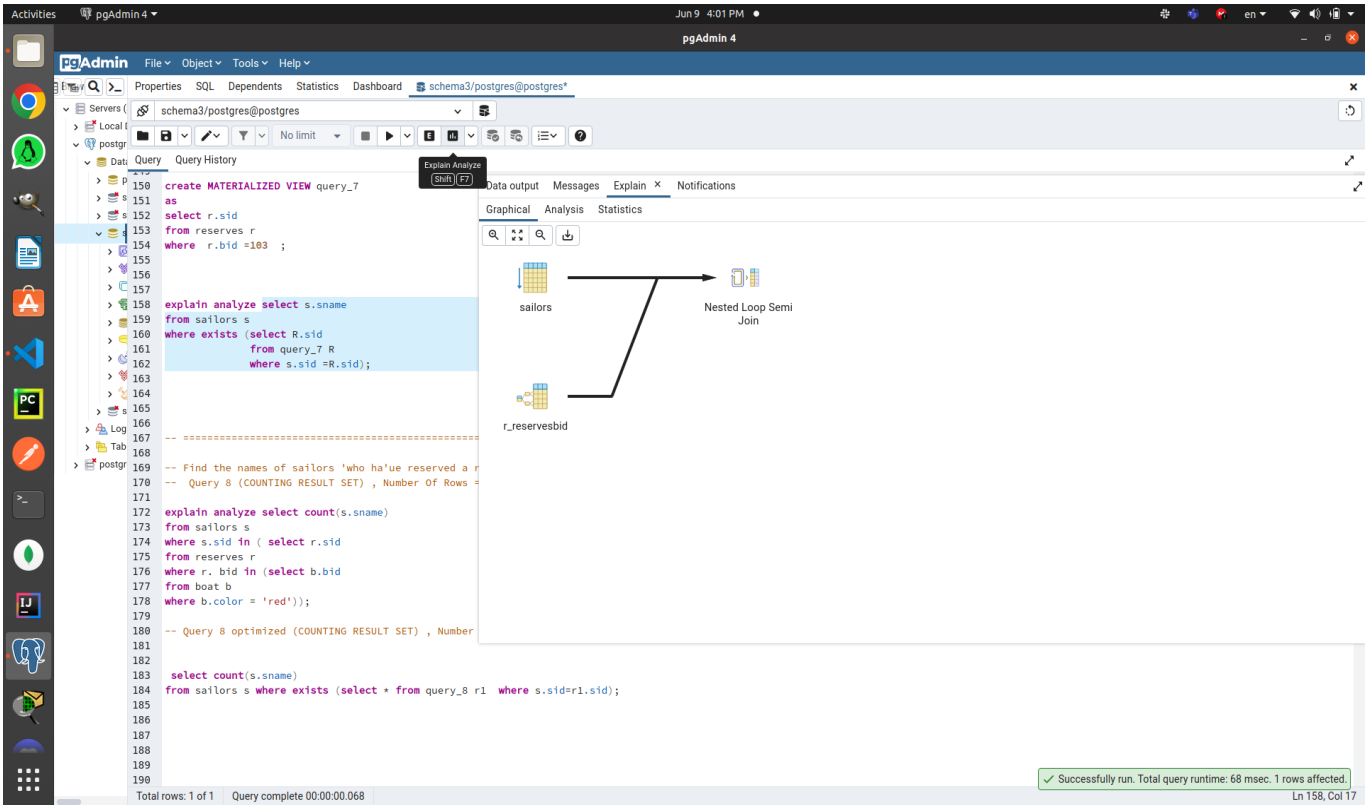
- Metrics :

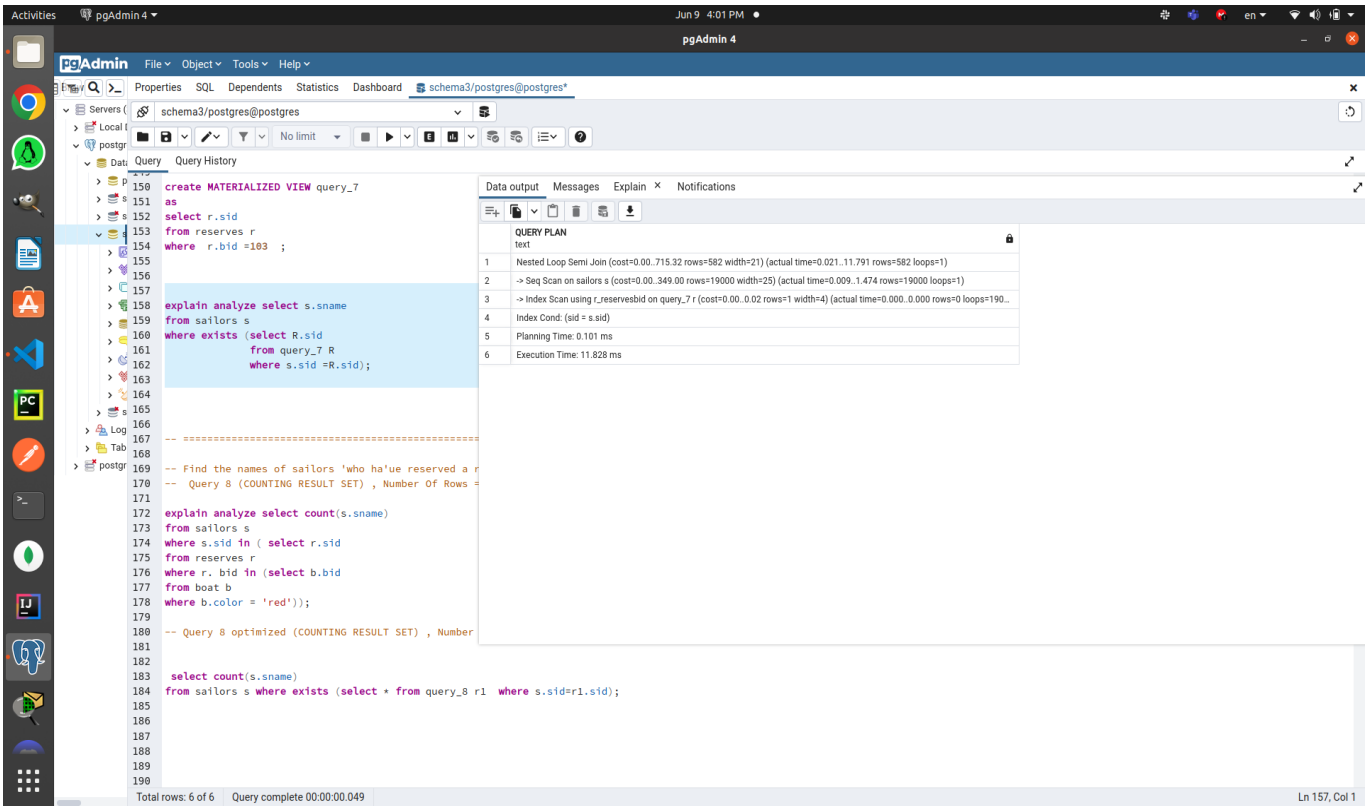
  Execution Time : 0.386 ms Total Expected Cost : 60.37

- 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_7 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.

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
DROP INDEX IF EXISTS reserves_pkey cascade;
-- Query 7 ==========
-- Find the names of sailors who have reserved boat 103.
-- Query 7 COUNTING RESULT SET , Number Of Rows = 582
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 R_reservesBID ON query_7 USING hash(sid);
select *
from pg_indexes
where tablename = 'sailors' or tablename='reserves' or tablename='boat' or tab
select count(s.sname)
from sailors s
where
s.sid in(select r.sid
from reserves r
where r.bid = 103);
-- Query 7 optimized (COUNTING RESULT SET) , Number Of Rows = 582
select count(s.sname)
from sailors s
where exists (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 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 query_7 r_reserves... [null] CREATE INDEX r_reservesbid ON public.query_7 USING hash (sid)
Total rows: 4 of 4 Query complete 00:00:00.207
Ln 95, Col 1


pgAdmin 4
Properties SQL Dependents Statistics Dashboard schema3/postgres@postgres*
schema3/postgres@postgres
Query Query History
Explain Analyze Shift F7
create MATERIALIZED VIEW query_7
as
select r.sid
from reserves r
where r.bid =103 ;
explain analyze select s.sname
from sailors s
where exists (select R.sid
from query_7 R
where s.sid =R.sid);
-- Find the names of sailors 'who ha'ue reserved a r
-- Query 8 (COUNTING RESULT SET) , Number Of Rows =
explain analyze select count(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 (COUNTING RESULT SET) , Number
select count(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
sailors
r_reservesbid
Nested Loop Semi Join
Successfully run. Total query runtime: 68 msec. 1 rows affected.
Total rows: 1 of 1 Query complete 00:00:00.068
Ln 158, Col 17



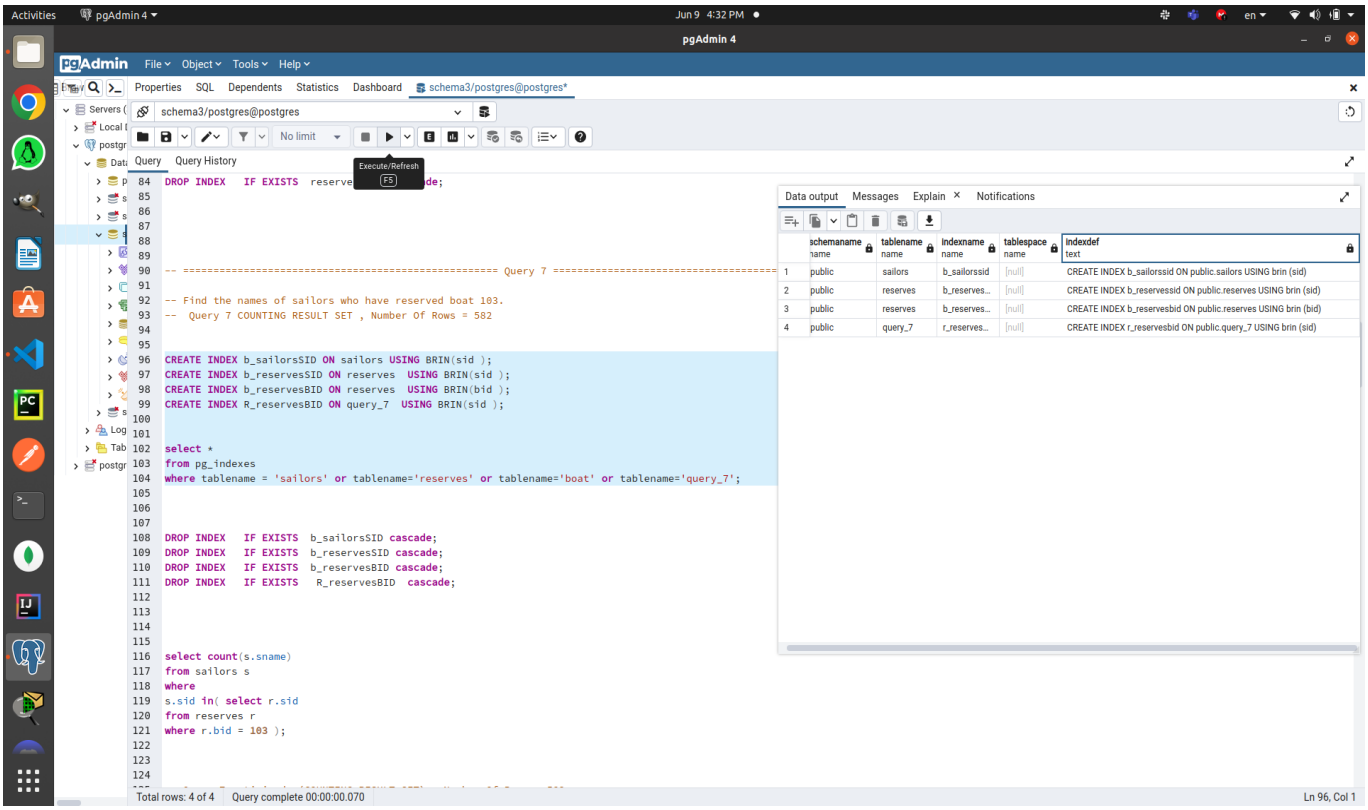
Explanation :

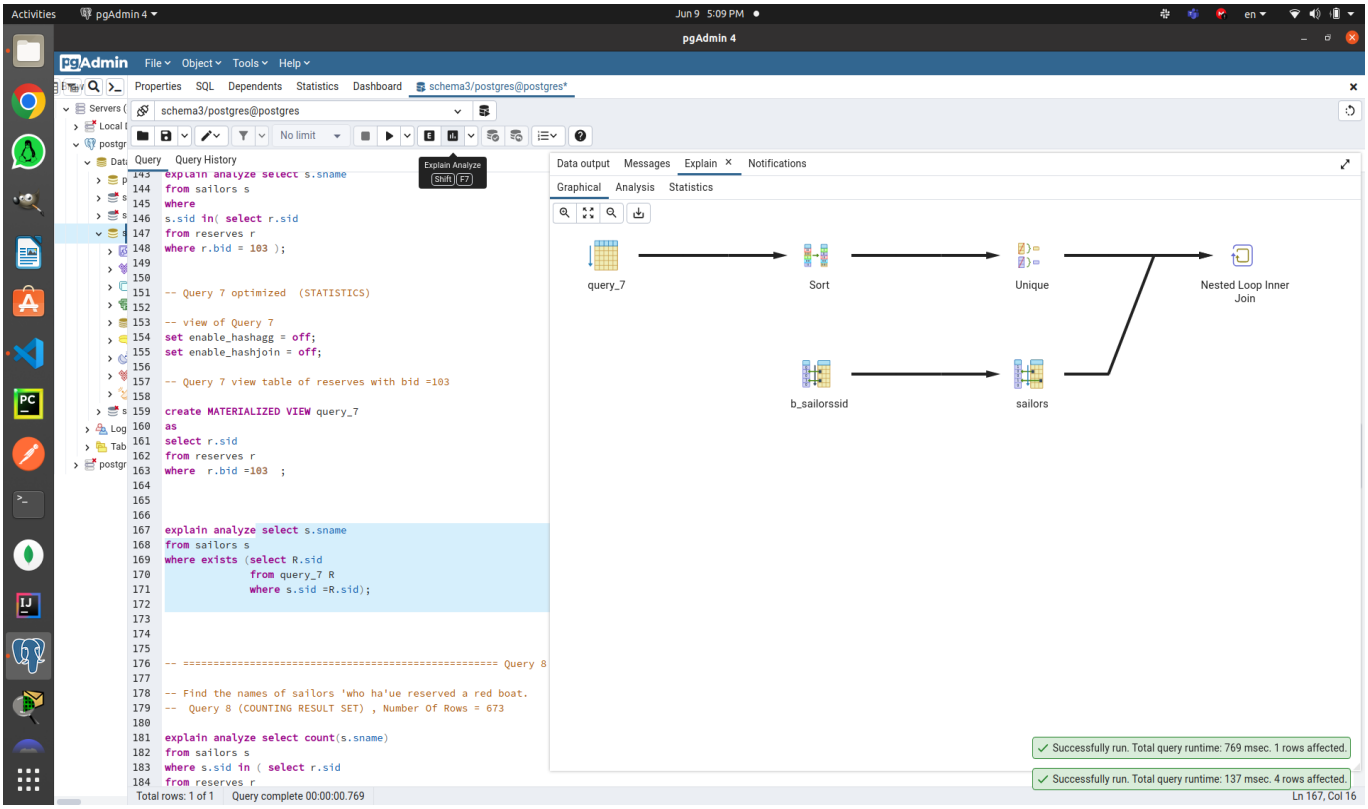
- Metrics :

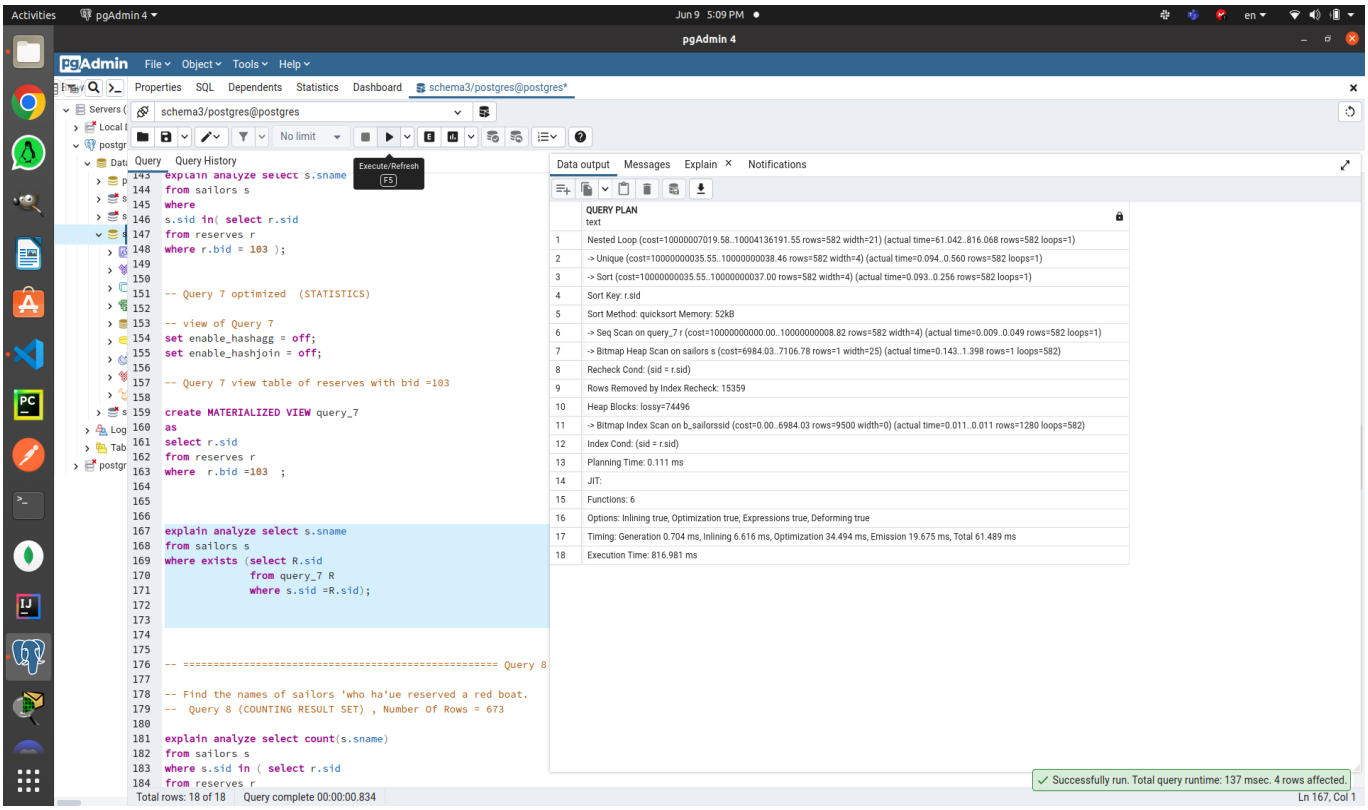
  Execution Time : 11.828 ms Total Expected Cost : 715.32

- 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_7 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
Execute/Refresh F5
-- ========== Query 7 ==========
-- Find the names of sailors who have reserved boat 103.
-- Query 7 COUNTING RESULT SET , Number Of Rows = 582
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 R_reservesBID ON query_7 USING BRIN(sid);
select *
from pg_indexes
where tablename = 'sailors' or tablename='reserves' or tablename='boat' or tablename='query_7';
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;
select count(s.sname)
from sailors s
where
s.sid in(select r.sid
from reserves r
where r.bid = 103);
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 query_7 r_reserves... [null] CREATE INDEX r_reservesbid ON public.query_7 USING brin (sid)
Total rows: 4 of 4 Query complete 00:00:00.070
Ln 96, Col 1


pgAdmin 4
Properties SQL Dependents Statistics Dashboard schema3/postgres@postgres*
schema3/postgres@postgres
Query Query History
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 = 103);
-- Query 7 optimized (STATISTICS)
-- view of Query 7
set enable_hashagg = off;
set enable_hashjoin = off;
-- Query 7 view table of reserves with bid =103
create MATERIALIZED VIEW query_7
as
select r.sid
from reserves r
where r.bid =103 ;
explain analyze select s.sname
from sailors s
where exists (select R.sid
from query_7 R
where s.sid =R.sid);
-- ========== Query 8
-- Find the names of sailors 'who ha'ue reserved a red boat.
-- Query 8 (COUNTING RESULT SET) , Number Of Rows = 673
explain analyze select count(s.sname)
from sailors s
where s.sid in (select r.sid
from reserves r
Data output Messages Explain Notifications
Graphical Analysis Statistics
query_7
Sort
Unique
Nested Loop Inner Join
b_sailorssid
sailors
Successfully run. Total query runtime: 769 msec. 1 rows affected.
Successfully run. Total query runtime: 137 msec. 4 rows affected.
Total rows: 1 of 1 Query complete 00:00:00.769
Ln 167, Col 16



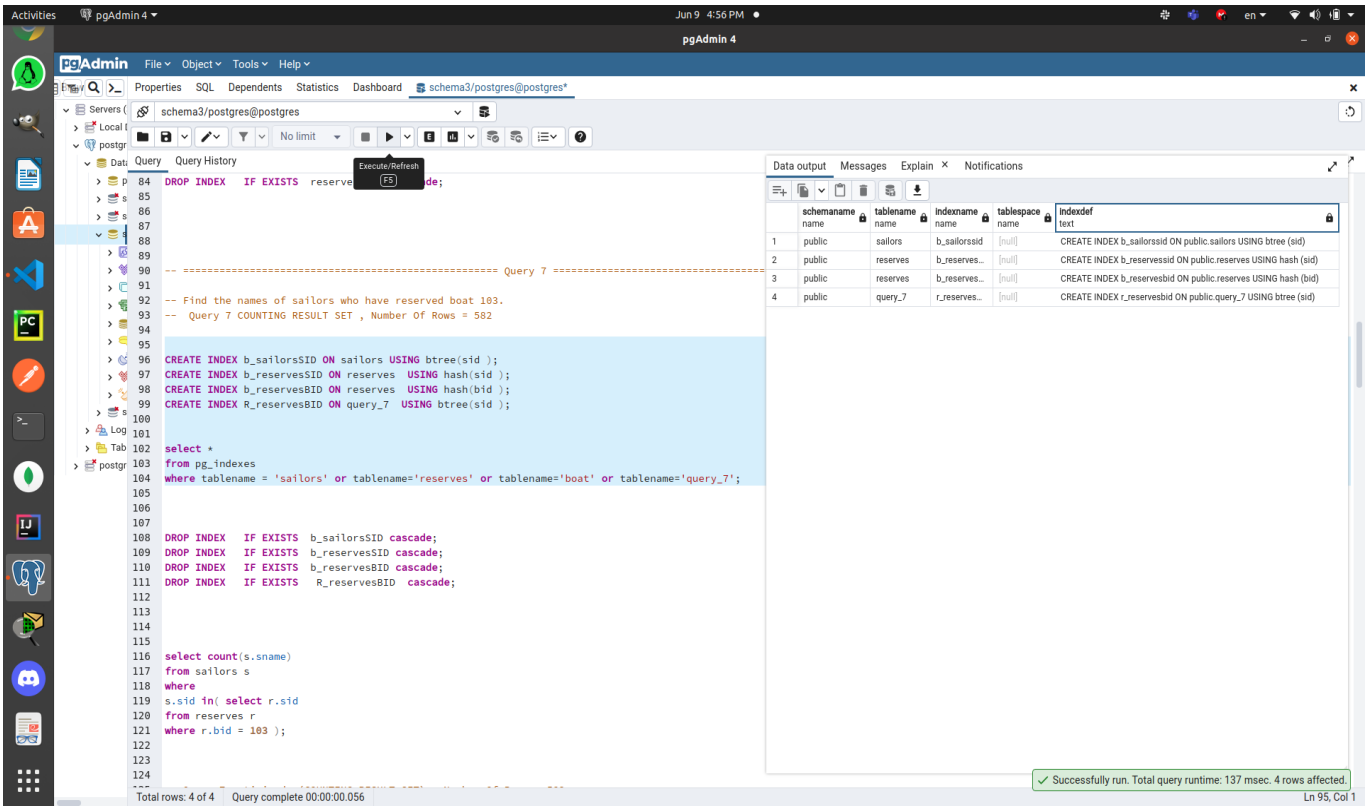
Explanation :

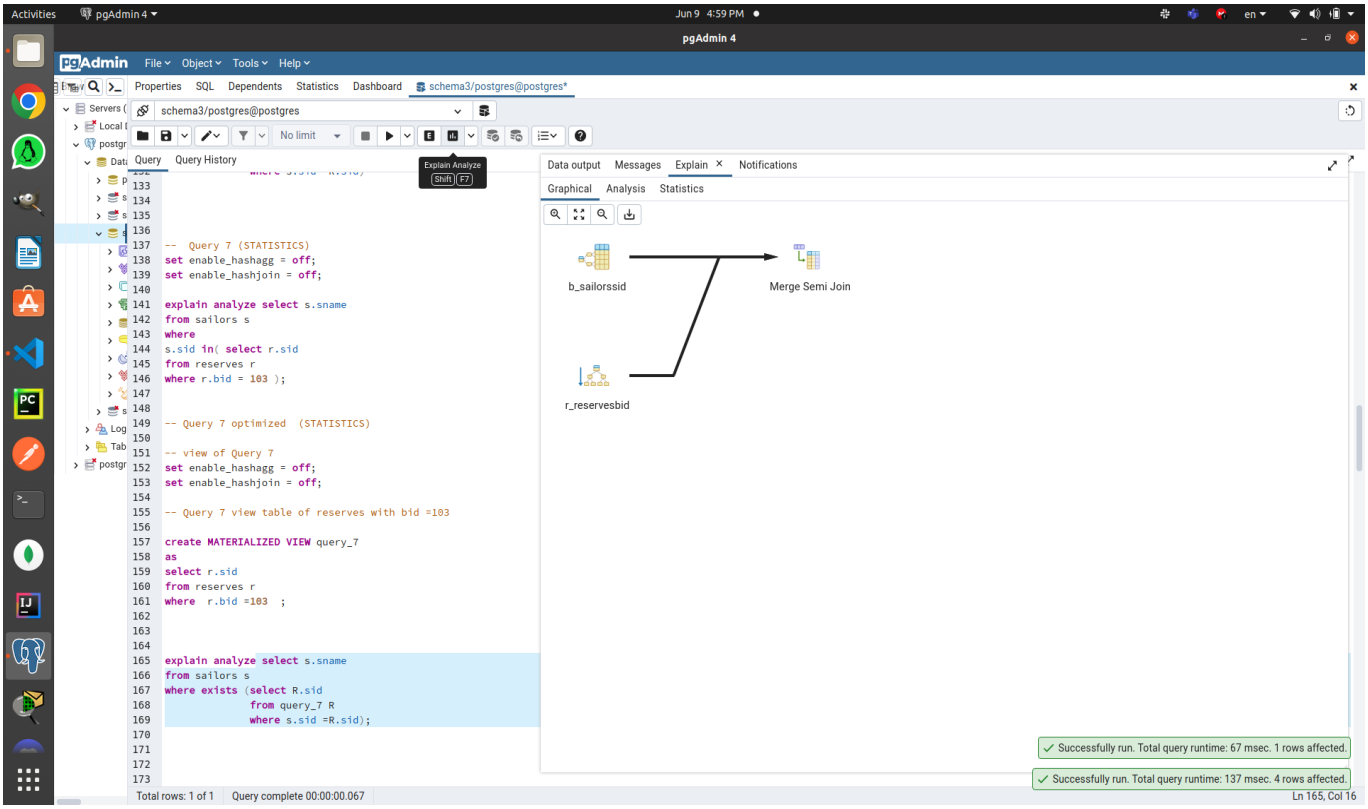
- Metrics :

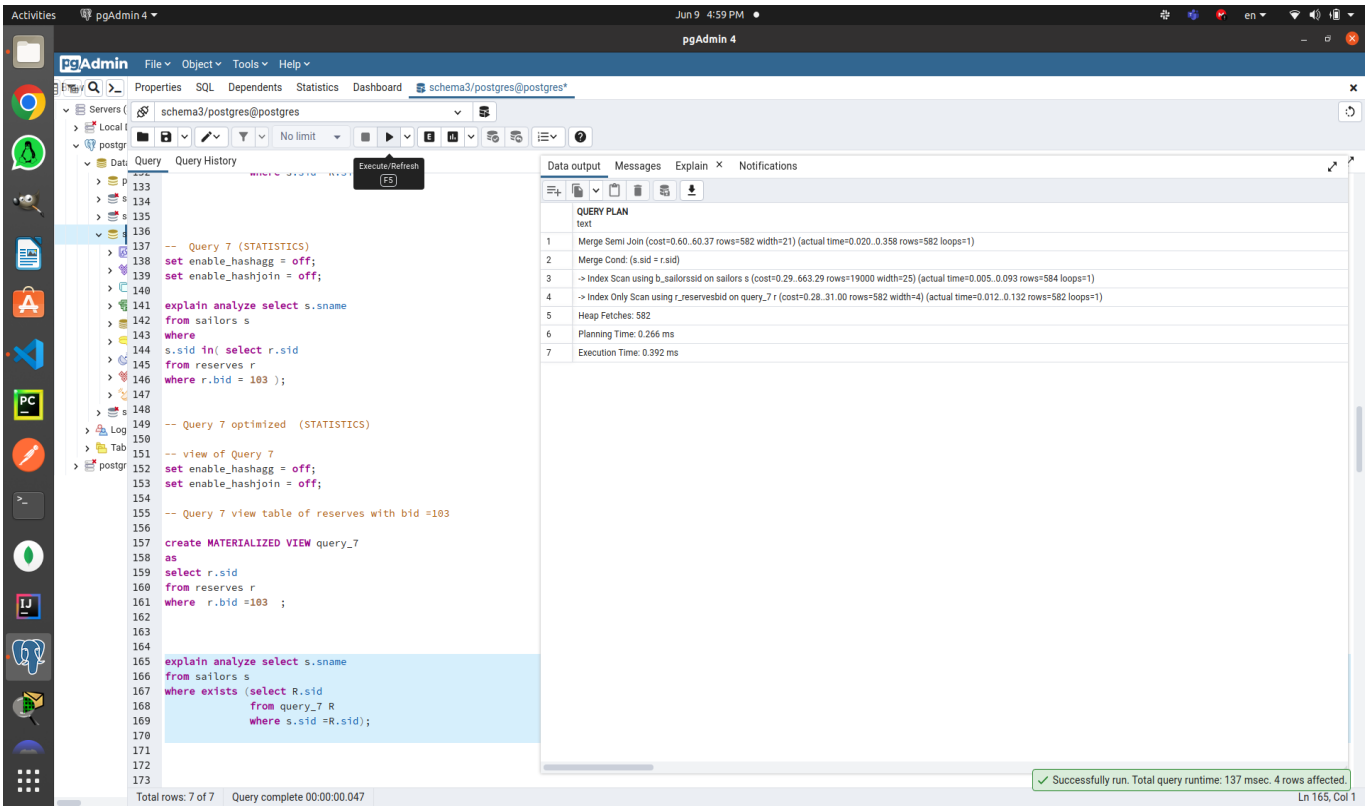
  Execution Time : 816.981 ms Total Expected Cost : 10004136191.55

- 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
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 hash (sid)
3 public reserves b_reserves... [null] CREATE INDEX b_reservesbid ON public.reserves USING hash (bid)
4 public query_7 r_reserves... [null] CREATE INDEX r_reservesbid ON public.query_7 USING btree (sid)
Successfully run. Total query runtime: 137 msec. 4 rows affected.
Total rows: 4 of 4 Query complete 00:00:00.056
Ln 95, 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_sailorssid
Merge Semi Join
r_reservesbid
Successfully run. Total query runtime: 67 msec. 1 rows affected.
Successfully run. Total query runtime: 137 msec. 4 rows affected.
Total rows: 1 of 1 Query complete 00:00:00.067
Ln 165, Col 16



Explanation :

- Metrics :

  Execution Time : 0.392 ms Total Expected Cost : 60.37

- The Query Planner used the Merge join by using both the ZigZag join and the Hash based algorithm together on 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 Mix indices helped in the performance it decreased the Execution Time and Expected Cost (BEST OF THEM ALL).