

Query 9

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

Note !

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

Original Query

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

Result Set

- 177 Rows

The screenshot shows the pgAdmin 4 interface. The left sidebar displays a tree view of database objects under 'schema3/postgres@postgres'. The main window shows a query editor with the following SQL code:

```
163
164
165
166
167 -- Query 9 (COUNTING RESULT SET) , Number Of Rows = 177
168 select count(s.sname)
169 from sailors s, reserves r, boat b
170 where
171     s.sid = r.sid
172     and
173         r.bid = b.bid
174     and
175         b.color = 'red'
176     and
177         b2.color = 'green';
178
179
180
181
182
183
184
185
186
187
188
189
190 select count(s.sname)
```

The results pane shows a single row of data:

count	bigint
1	177

At the bottom, a status bar indicates: "Total rows: 1 of 1 Query complete 00:00:00.107". A message bar at the bottom right says "Successfully run. Total query runtime: 51 msec. 582 rows affected Ln 170, Col 1".

Report

1. given query without an index :

pgAdmin 4

May 17 12:08 AM

schema3/postgres@postgres

```

15
16 select count(sid)
from sailors;
17
18 select count(*)
from boat;
19
20
21
22 select count(*)
from reserves;
23
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

schemaname	tablename	indexname	tablespace	indexdef
name	name	name	name	text

Total rows: 0 of 0 Query complete 00:00:00.17

Successfully run. Total query runtime: 51 msec. 582 rows affected.

Ln 35, Col 1

pgAdmin 4

Jun 8 11:18 PM

schema3/postgres@postgres

```

280
281
282 -- Query 9 (STATISTICS)
283 set enable_hashagg = off;
284 set enable_hashjoin = off;
285 explain analyze select s.sname
286   from sailors s, reserves r, boat b
287   where
288     s.sid = r.sid
289     and
290     r.bid = b.bid
291     and
292     b.color = 'red'
293     and
294     s.sid in ( select s2.sid
295       from sailors s2, boat b2, reserves r2
296       where s2.sid = r2.sid
297       and
298         r2.bid = b2.bid
299       and
300         b2.color = 'green');
301
302
303 -- Query 9 optimization (STATISTICS)
304 -- Find the names of sailors who have reserved both a red and a green boat.
305
306
307
308 select *
309   from pg_indexes
310  where tablename = 'sailors' or tablename='reserves' or tablename='boat';
311
312
313 explain analyze select s.sname
314   from sailors s
315  where exists
316    (
317      select rTotal.sid
318        from (select r1.sid
319          from
320            (select r.sid

```

Data output

Graphical Explain

Ln 301, Col 1

```

-- Query 9 (STATISTICS)
set enable_hashagg = off;
set enable_hashjoin = off;
explain analyze select s.sname
from sailors s, reserves r, boat b
where
    s.sid = r.sid
    and
    r.bid = b.bid
    and
    b.color = 'red'
    and
    s.sname in ( select s2.sname
    from sailors s2, boat b2, reserves r2
    where s2.sid = r2.sid
    and
    r2.bid = b2.bid
    and
    b2.color = 'green');

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

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

explain analyze select s.sname
from sailors s
where exists
(
    select rTotal.sid
    from (select r1.sid
    from
        (select r.sid

```

Total rows: 46 of 46 Query complete 00:00:00.223 Ln 283, Col 1

```

-- Query 9 (STATISTICS)
set enable_hashagg = off;
set enable_hashjoin = off;
explain analyze select s.sname
from sailors s, reserves r, boat b
where
    s.sid = r.sid
    and
    r.bid = b.bid
    and
    b.color = 'red'
    and
    s.sname in ( select s2.sname
    from sailors s2, boat b2, reserves r2
    where s2.sid = r2.sid
    and
    r2.bid = b2.bid
    and
    b2.color = 'green');

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

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

explain analyze select s.sname
from sailors s
where exists
(
    select rTotal.sid
    from (select r1.sid
    from
        (select r.sid

```

Total rows: 46 of 46 Query complete 00:00:00.223 Ln 283, Col 1

Explanation :

- Metrics :

Execution Time : 64.569 ms Total Expected Cost : 11368.59

- given query with B+ trees indices only :

Activities pgAdmin 4 Jun 9 12:37 AM ● pgAdmin 4

Servers Local [No limit] Databases Query History Execute/Refresh

```

300 b2.color = 'green';
301
302
303 -- Query 9 optimization (STATISTICS)
304 -- Find the names of sailors who have reserved both a red and a green boat.
305
306
307
308 select *
309 from pg_indexes
310 where tablename = 'sailors' or tablename='reserves' or tablename='boat';
311
312 set enable_hashagg = off;
313 set enable_hashjoin = off;
314
315 explain analyze select s.sname
316 from sailors s
317 where exists
318 (
319     select rTotal.sid
320         from (select r1.sid
321             from
322                 (select r.sid
323                  from reserves r
324                  where exists
325                      (select bid
326                         from boat b
327                         where color = 'green' and r.bid = b.bid )
328                     ) as r1
329             inner join
330                 (select r.sid
331                  from reserves r
332                  where exists
333                      (select bid
334                         from boat b
335                         where color = 'red' and r.bid = b.bid )
336                     ) as r2
337             on r2.sid = r1.sid ) as rTotal
338         where rTotal.sid=s.sid
339 )
340

```

Total rows: 4 of 4 Query complete 00:00:00.224 Ln 308, Col 1

Activities pgAdmin 4 Jun 9 12:15 AM ● pgAdmin 4

Servers (3) Local Database [No limit] Databases Query History Execute/Refresh

```

278
279
280
281
282 -- Query 9 (STATISTICS)
283 set enable_hashagg = off;
284 set enable_hashjoin = off;
285 explain analyze select s.sname
286 from sailors s, reserves r, boat b
287 where
288     s.sid = r.sid
289     and
290     r.bid = b.bid
291     and
292     b.color = 'red'
293     and
294     s.sid in ( select s2.sid
295         from sailors s2, boat b2, reserves r2
296         where s2.sid = r2.sid
297         and
298             r2.bid = b2.bid
299         and
300             b2.color = 'green' );
301
302
303 -- Query 9 optimization (STATISTICS)
304 -- Find the names of sailors who have reserved both a red and a green boat.
305
306
307
308 select *
309 from pg_indexes
310 where tablename = 'sailors' or tablename='reserves' or tablename='boat';
311
312 set enable_hashagg = off;
313 set enable_hashjoin = off;
314
315 explain analyze select s.sname
316 from sailors s
317 where exists
318 (

```

QUERY PLAN

- Merge Semi Join (cost=3948.54..5557.12 rows=1309 width=21) (actual time=4.931..7.873 rows=177 loops=1)
- Merge Cond: (s.sid = s2.sid)
- > Merge Join (cost=1973.36..2758.56 rows=4982 width=29) (actual time=1.803..3.041 rows=1136 loops=1)
- Merge Cond: (s.sid = r.sid)
- > Index Scan using b_sailorssid on sailors s (cost=0.29..663.29 rows=19000 width=25) (actual time=0.008..0.613 rows=2999 loops=1)
- > Sort (cost=1973.08..1985.53 rows=4982 width=4) (actual time=1.790..1.868 rows=1136 loops=1)
- Sort Key: r.sid
- Sort Method: quicksort Memory: 102KB
- > Nested Loop (cost=0.29..1667.12 rows=4982 width=29) (actual time=0.016..1.525 rows=1136 loops=1)
- > Seq Scan on boat b (cost=0.00..62.50 rows=427 width=4) (actual time=0.010..0.441 rows=427 loops=1)
- Filter: (color = 'red'-bpchar)
- Rows Removed by Filter: 2573
- > Index Scan using b_reservesbid on reserves r (cost=0.29..3.48 rows=28 width=8) (actual time=0.001..0.002 rows=3 loops=427)
- Index Cond: (bid = b.bid)
- > Merge Join (cost=1975.18..2760.54 rows=4993 width=8) (actual time=2.933..4.596 rows=1861 loops=1)
- Merge Cond: (s2.sid = r2.sid)
- > Index Only Scan using b_sailorssid on sailors s2 (cost=0.29..663.29 rows=19000 width=4) (actual time=0.016..0.888 rows=2999 loops=1)
- Heap Fetches: 2999
- > Sort (cost=1974.89..1987.37 rows=4993 width=4) (actual time=2.852..2.993 rows=1861 loops=1)
- Sort Key: r2.sid
- Sort Method: quicksort Memory: 193kB
- > Nested Loop (cost=0.29..1668.18 rows=4993 width=4) (actual time=0.064..2.301 rows=2064 loops=1)
- > Seq Scan on boat b2 (cost=0.00..62.50 rows=428 width=4) (actual time=0.058..0.418 rows=428 loops=1)
- Filter: (color = 'green'-bpchar)
- Rows Removed by Filter: 2572
- > Index Scan using b_reservesbid on reserves r2 (cost=0.29..3.47 rows=28 width=8) (actual time=0.002..0.004 rows=5 loops=428)
- Index Cond: (bid = b2.bid)
- Planning Time: 1.220 ms
- Execution Time: 7.925 ms

Total rows: 29 of 29 Query complete 00:00:00.153 Ln 300, Col 21

```

-- Query 9 (STATISTICS)
set enable_hashagg = off;
set enable_hashjoin = off;
explain analyze select s.sname
from sailors s, reserves r, boat b
where
    s.sid = r.sid
    and
    r.bid = b.bid
    and
    b.color = 'red';
-- Query 9 optimization (STATISTICS)
-- Find the names of sailors who have reserved both a red and a green boat
select *
from pg_indexes
where tablename = 'sailors' or tablename='reserves' or tablename='boat';
set enable_hashagg = off;
set enable_hashjoin = off;
explain analyze select s.sname
from sailors s
where exists
(
    ...
);

```

Total rows: 1 of 1 | Query complete 00:00:00.176 | ✓ Successfully run. Total query runtime: 176 msec. 1 rows affected. Ln 285, Col 16

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 .

- given query with hash indices only :

Activities pgAdmin 4 Jun 9 12:48 AM ●

File Object Tools Help

Servers Local PostgreSQL

Data Query History Execute/Refresh

```

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

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);

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

```

Total rows: 5 of 5 Query complete 00:00:00.050

Ln 277, Col 1

Activities pgAdmin 4 Jun 9 1:02 AM ●

File Object Tools Help

Servers Local PostgreSQL

Data Query History Execute/Refresh

```

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);

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

-- Query 9 (STATISTICS)
set enable_hashagg = off;
set enable_hashjoin = off;
explain analyze select s.sname
from sailors s, reserves r, boat b
where
s.sid = r.sid
and
r.bid = b.bid
and
b.color = 'red'
and
s.sid in ( select s2.sid
from sailors s2, boat b2, reserves r2
where s2.sid = r2.sid
and
r2.bid = b2.bid
and
b2.color = 'green');

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

```

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_reservesSID 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 = r2.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

The screenshot shows the pgAdmin 4 interface with the following details:

- Left Sidebar:** Contains icons for Activities, pgAdmin 4, File, Object, Tools, Help, Servers, Local, and Postgres.
- Top Bar:** Shows the date and time (Jun 9 1:10 AM) and a notifications icon.
- Central Area:**
 - Query Editor:** Displays the SQL code for Query 9, including index creation statements and a complex select statement with joins and conditions.
 - Explain Plan:** A graphical representation of the query's execution plan. It shows nested loop joins between tables: boat, reserves, sailors, and boat again. The plan indicates the use of Hash indexes for joins.
 - Bottom Status:** Shows "Total rows: 1 of 1" and "Query complete 00:00:00.091".

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.

- given query with BRIN indices only :

Activities pgAdmin 4 Jun 9 2:48 AM ● pgAdmin 4

Servers Local PostgreSQL schema3/postgres

Datad Query History Execute/Refresh

```

267 CREATE INDEX b_sailorsSID ON sailors USING BRIN(sid );
268 CREATE INDEX b_reservesID ON reserves USING BRIN(sid );
269 CREATE INDEX b_reservesBID ON reserves USING BRIN(bid );
270 CREATE INDEX b_boat ON boat USING BRIN(bid,color );
271
272
273
274 select *
275 from pg_indexes
276 where tablename = 'sailors' or tablename='reserves' or tablename='boat';
277
278
279
280 Log
281 Tab
282
283 -- Query 9 (STATISTICS)
284 set enable_hashagg = off;
285 set enable_hashjoin = off;
286 explain analyze select s.sname
287 from sailors s, reserves r, boat b
288 where
289 s.sid = r.sid
290 and
291 r.bid = b.bid
292 and
293 b.color = 'red'
294 and
295 s.sid in ( select s2.sid
296 from sailors s2, boat b2, reserves r2
297 where s2.sid = r2.sid
298 and
299 r2.bid = b2.bid
300 and
301 b2.color = 'green');
302
303 -- Query 9 optimization (STATISTICS)
304

```

Total rows: 4 of 4 Query complete 00:00:00.088

Ln 273, Col 1

Activities pgAdmin 4 Jun 9 2:48 AM ● pgAdmin 4

Servers Local PostgreSQL schema3/postgres

Datad Query History Execute/Refresh

```

275 from pg_indexes
276 where tablename = 'sailors' or tablename='reserves' or tablename='boat';
277
278
279
280
281
282 -- Query 9 (STATISTICS)
283 set enable_hashagg = off;
284 set enable_hashjoin = off;
285 set enable_sort = on;
286 set enable_seqscan = off;
287
288 set enable_bitmapscan = on;
289
290 explain analyze select s.sname
291 from sailors s, reserves r, boat b
292 where
293 s.sid = r.sid
294 and
295 r.bid = b.bid
296 and
297 b.color = 'red'
298 and
299 s.sid in ( select s2.sid
300 from sailors s2, boat b2, reserves r2
301 where s2.sid = r2.sid
302 and
303 r2.bid = b2.bid
304 and
305 b2.color = 'green');
306
307
308 -- Query 9 optimization (STATISTICS)
309 -- Find the names of sailors who have reserved both a red and a green boat.
310
311
312
313 select *
314 from pg_indexes
315 where tablename = 'sailors' or tablename='reserves' or tablename='boat';
316

```

Total rows: 59 of 59 Query complete 00:00:04.702

Ln 306, Col 1

Activities pgAdmin 4 Jun 9 2:49 AM ● pgAdmin 4

Servers Local PostgreSQL Data Query History

```

> p 275 from pg_indexes
> s 276 where tablename = 'sailors' or tablename='reserves' or tablename='boat';
> s 277
> s 278
> s 279
> s 280
> s 281
> s 282 -- Query 9 (STATISTICS)
> s 283 set enable_hashagg = off;
> s 284 set enable_hashjoin = off;
> s 285 set enable_sort = on;
> s 286 set enable_seqscan = off;
> s 287
> s 288 set enable_bitmapscan = on;
> s 289
> s 290 explain analyze select s.sname
> s 291 from sailors s, reserves r, boat b
> s 292 where
> s 293 s.sid = r.sid
> s 294 and
> s 295 r.bid = b.bid
> s 296 and
> s 297 b.color = 'red'
> s 298 and
> s 299 s.sid in (select s2.sid
> s 300 from sailors s2, boat b2, reserves r2
> s 301 where s2.sid = r2.sid
> s 302 and
> s 303 r2.bid = b2.bid
> s 304 and
> s 305 b2.color = 'green');
> s 306
> s 307
> s 308 -- Query 9 optimization (STATISTICS)
> s 309 -- Find the names of sailors who have reserved both a red and a green boat.
> s 310
> s 311
> s 312
> s 313 select *
> s 314 from pg_indexes
> s 315 where tablename = 'sailors' or tablename='reserves' or tablename='boat';

```

Total rows: 59 of 59 Query complete 00:00:04.702 Ln 306, Col 1

Activities pgAdmin 4 Jun 9 2:50 AM ● pgAdmin 4

Servers Local PostgreSQL Data Query History

```

> p 275 from pg_indexes
> s 276 where tablename = 'sailors' or tablename='reserves'
> s 277
> s 278
> s 279
> s 280
> s 281
> s 282 -- Query 9 (STATISTICS)
> s 283 set enable_hashagg = off;
> s 284 set enable_hashjoin = off;
> s 285 set enable_sort = on;
> s 286 set enable_seqscan = off;
> s 287
> s 288 set enable_bitmapscan = on;
> s 289
> s 290 explain analyze select s.sname
> s 291 from sailors s, reserves r, boat b
> s 292 where
> s 293 s.sid = r.sid
> s 294 and
> s 295 r.bid = b.bid
> s 296 and
> s 297 b.color = 'red'
> s 298 and
> s 299 s.sid in (select s2.sid
> s 300 from sailors s2, boat b2, reserves r2
> s 301 where s2.sid = r2.sid
> s 302 and
> s 303 r2.bid = b2.bid
> s 304 and
> s 305 b2.color = 'green');
> s 306
> s 307
> s 308 -- Query 9 optimization (STATISTICS)
> s 309 -- Find the names of sailors who have reserved both
> s 310
> s 311
> s 312
> s 313 select *
> s 314 from pg_indexes
> s 315 where tablename = 'sailors' or tablename='reserves'

```

Total rows: 1 of 1 Query complete 00:00:04.674 Ln 290, Col 17

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 traversed 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) :

pgAdmin 4

```

SELECT r.sid
FROM reserves r
WHERE EXISTS
    (SELECT bid
     FROM boat b
     WHERE color = 'red' AND r.bid = b.bid)
        ) AS rTotal
    ON r2.sid = r1.sid ) AS rTotal
    WHERE rTotal.sid = s.sid
);

CREATE INDEX b_sailorssID ON sailors USING HASH(sid);
CREATE INDEX b_reservesID 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 btree(color);

SELECT *
FROM pg_indexes
WHERE tablename = 'sailors' OR tablename='reserves' OR tablename='boat';

```

Total rows: 5 of 5 Query complete 00:00:00.207

Ln 274, Col 1

pgAdmin 4

```

CREATE INDEX b_sailorssID ON sailors USING HASH(sid);
CREATE INDEX b_reservesID 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 btree(color);

SELECT *
FROM pg_indexes
WHERE tablename = 'sailors' OR tablename='reserves' OR tablename='boat';

-- Query 9 (STATISTICS)
SET enable_hashagg = off;
SET enable_hashjoin = off;

EXPLAIN ANALYZE SELECT s.sname
FROM sailors s, reserves r, boat b
WHERE
s.sid = r.sid
AND
r.bid = b.bid
AND
b.color = 'red'
```
Total rows: 5 of 5 Query complete 00:00:00.207

```

Ln 287, Col 1

QUERY PLAN

```

1 Nested Loop Semi Join (cost=11.59..3022.13 rows=1309 width=21) (actual time=2.590..8.704 rows=177 loops=1)
 2 Join Filter: (s.sid = r2.sid)
 3 -> Nested Loop (cost=11.59..1987.83 rows=4982 width=29) (actual time=0.055..2.488 rows=1136 loops=1)
 4 -> Nested Loop (cost=11.59..1722.72 rows=4982 width=4) (actual time=0.051..1.021 rows=1136 loops=1)
 5 -> Bitmap Heap Scan on boat b (cost=11.59..41.93 rows=427 width=4) (actual time=0.043..0.100 rows=427 loops=1)
 6 Recheck Cond: (color = 'red')::bpchar
 7 Heap Blocks: exact4
 8 -> Bitmap Index Scan on b_boat2 (cost=0.00..11.48 rows=427 width=0) (actual time=0.035..0.035 rows=427 loops=1)
 9 Index Cond: (color = 'red')::bpchar
 10 -> 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)
 11 Index Cond: (bid = b.bid)
 12 -> 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)
 13 Index Cond: (sid = r.sid)
 14 -> Nested Loop (cost=0.00..0.20 rows=1 width=8) (actual time=0.005..0.005 rows=0 loops=1136)
 15 Join Filter: (r2.sid = s2.sid)
 16 -> Nested Loop (cost=0.00..0.14 rows=1 width=4) (actual time=0.005..0.005 rows=0 loops=1136)
 17 -> Index Scan using b_reservesID on reserves r2 (cost=0.00..0.07 rows=2 width=8) (actual time=0.001..0.002 rows=3 loops=1136)
 18 Index Cond: (sid = r2.sid)
 19 -> 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)
 20 Index Cond: (bid = r2.bid)
 21 Filter: (color = 'green')::bpchar
 22 Rows Removed by Filter: 1
 23 -> 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)
 24 Index Cond: (sid = r2.sid)
 25 Planning Time: 0.497 ms
 26 Execution Time: 8.749 ms

```

Total rows: 26 of 26    Query complete 00:00:00.082

Ln 287, Col 1

The screenshot shows the pgAdmin 4 interface with the 'Explain Analyze' tool open. The left pane displays the query code, which includes several CREATE INDEX statements and a complex SELECT query involving multiple tables (sailors, reserves, boat) and their respective indexes. The right pane shows the detailed execution plan, specifically the 'Graphical' tab of the Explain Analyze window. The plan illustrates a Nested Loop Semi Join with three main stages of Nested Loop Inner Joins. The first stage joins b\_boat2 and boat. The second stage joins b\_reservesid and b\_sailorsid. The third stage joins b\_reservesid and b\_boat1. The final stage is the Nested Loop Semi Join. The nodes in the plan represent different table scans and index lookups, with arrows indicating the flow of data between them.

```

CREATE INDEX b_sailorsSID ON sailors USING hash(sid);
CREATE INDEX b_reservesID ON reserves USING HASH(sid);
CREATE INDEX b_reservesID ON reserves USING HASH(bid);
CREATE INDEX b_boat1 ON boat USING HASH(bid);
CREATE INDEX b_boat2 ON boat USING btree(color);

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

-- Query 9 (STATISTICS)
set enable_hashagg = off;
set enable_hashjoin = off;

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

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

```

### 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 repeatable 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,

The screenshot shows the pgAdmin 4 interface. The left sidebar displays the database browser with several servers and databases listed. The main area is a query editor containing the following SQL code:

```

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

```

The status bar at the bottom indicates "Successfully run. Total query runtime: 51 msec. 582 rows affected." and "Ln 35, Col 1".

pgAdmin 4 • Jun 8 11:55 PM

```

Explain Analyze [Shift+F7]
Data output Messages Explain Notifications
Graphical Analysis Statistics
Graphical

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

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

set enable_hashagg = off;
set enable_hashjoin = off;

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

```

Total rows: 1 of 1    Query complete 00:00:00.081

pgAdmin 4 • Jun 8 11:53 PM

```

Explain Analyze [Shift+F7]
Data output Messages Explain Notifications
QUERY PLAN
text
1 Merge Semi Join (cost=9291.63..9466.30 rows=1324 width=21) (actual time=39.371..40.008 rows=177 loops=1)
 2 Merge Cond: (s.sid = r1.sid)
 3 > Sort (cost=1699.30..1746.80 rows=19000 width=25) (actual time=6.059..6.110 rows=644 loops=1)
 4 Sort Key: s.sid
 5 Sort Method: quicksort Memory: 2259kB
 6 > Seq Scan on sailors s (cost=0.00..349.00 rows=19000 width=25) (actual time=0.013..2.765 rows=19000 loops=1)
 7 > Merge Join (cost=7592.33..7655.45 rows=1324 width=8) (actual time=33.273..33.786 rows=177 loops=1)
 8 Merge Cond: (r.sid = r1.sid)
 9 > Sort (cost=3796.63..3809.11 rows=4993 width=4) (actual time=15.839..16.001 rows=1861 loops=1)
 10 Sort Key: r.sid
 11 Sort Method: quicksort Memory: 193kB
 12 > Merge Semi Join (cost=3262.84..3489.91 rows=4993 width=4) (actual time=14.640..15.283 rows=2064 loops=1)
 13 Merge Cond: (r.bid = b.bid)
 14 > Sort (cost=3181.64..3269.14 rows=35000 width=8) (actual time=13.981..14.196 rows=3201 loops=1)
 15 Sort Key: r.bid
 16 Sort Method: quicksort Memory: 3006kB
 17 > Seq Scan on reserves r (cost=0.00..540.00 rows=35000 width=8) (actual time=0.011..4.874 rows=35000 loops=1)
 18 > Sort (cost=81.21..82.28 rows=428 width=4) (actual time=0.486..0.516 rows=428 loops=1)
 19 Sort Key: b.bid
 20 Sort Method: quicksort Memory: 45kB
 21 > Seq Scan on boat b (cost=0.00..62.50 rows=428 width=4) (actual time=0.069..0.427 rows=428 loops=1)
 22 Filter: (color = 'green')::bpchar
 23 Rows Removed by Filter: 2572
 24 > Sort (cost=395.70..3808.16 rows=4982 width=4) (actual time=17.369..17.453 rows=1136 loops=1)
 25 Sort Key: r1.sid
 26 Sort Method: quicksort Memory: 102kB
 27 > Merge Semi Join (cost=3262.79..3489.75 rows=4982 width=4) (actual time=16.407..17.021 rows=1136 loops=1)
 28 Merge Cond: (r1.bid = b1.bid)
 29 > Sort (cost=3181.64..3269.14 rows=35000 width=8) (actual time=15.586..15.707 rows=1137 loops=1)
 30 Sort Key: r1.bid

```

Total rows: 40 of 40    Query complete 00:00:00.248

The screenshot shows the pgAdmin 4 interface. On the left is a tree view of database objects: Servers (3), Local Database (1), and the 'postgres' database containing Databases (5), Query (1), and Query History (1). The main area displays a SQL query and its corresponding query plan.

```

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

SELECT *
FROM pg_indexes
WHERE tablename = 'sailors' OR tablename='reserves' OR tablename='boat';

EXPLAIN ANALYZE SELECT s.sname
FROM sailors s
WHERE EXISTS (
 SELECT rTotal.sid
 FROM (SELECT r1.sid
 FROM (
 SELECT r.sid
 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
);

```

The query plan (right pane) details the execution steps:

- 1 -> Merge Semi Join (cost=3262.84..3489.91 rows=4993 width=4) (actual time=14.640..15.283 rows=2064 loops=1)
- 12 -> Sort (cost=3181.64..3269.14 rows=35000 width=8) (actual time=13.981..14.196 rows=3201 loops=1)
- 13 Sort Key: r.bid
- 14 Sort Method: quicksort Memory: 3006KB
- 15 -> Seq Scan on reserves r (cost=0.00..540.00 rows=35000 width=8) (actual time=0.011..4.874 rows=35000 loops=1)
- 16 Sort Key: r.bid
- 17 Sort Method: quicksort Memory: 45kB
- 18 -> Sort (cost=81.21..82.28 rows=428 width=4) (actual time=0.486..0.516 rows=428 loops=1)
- 19 Sort Key: b.bid
- 20 Sort Method: quicksort Memory: 45kB
- 21 -> Seq Scan on boat b (cost=0.00..62.50 rows=428 width=4) (actual time=0.069..0.427 rows=428 loops=1)
- 22 Filter: (color = 'green'::bpchar)
- 23 Rows Removed by Filter: 2572
- 24 -> Sort (cost=3795.70..3808.16 rows=4982 width=4) (actual time=17.369..17.453 rows=1136 loops=1)
- 25 Sort Key: r\_1.sid
- 26 Sort Method: quicksort Memory: 102kB
- 27 -> Merge Semi Join (cost=3262.79..3489.75 rows=4982 width=4) (actual time=16.407..17.021 rows=1136 loops=1)
- 28 Merge Cond: (r\_1.bid = b\_1.bid)
- 29 -> Sort (cost=3181.64..3269.14 rows=35000 width=8) (actual time=15.586..15.707 rows=1137 loops=1)
- 30 Sort Key: r\_1.bid
- 31 Sort Method: quicksort Memory: 3006KB
- 32 -> Seq Scan on reserves r\_1 (cost=0.00..540.00 rows=35000 width=8) (actual time=0.010..5.623 rows=35000 loops=1)
- 33 -> Sort (cost=81.16..82.22 rows=427 width=4) (actual time=0.813..0.851 rows=427 loops=1)
- 34 Sort Key: b\_1.bid
- 35 Sort Method: quicksort Memory: 45kB
- 36 -> Seq Scan on boat b\_1 (cost=0.00..62.50 rows=427 width=4) (actual time=0.028..0.713 rows=427 loops=1)
- 37 Filter: (color = 'red'::bpchar)
- 38 Rows Removed by Filter: 2573
- 39 Planning Time: 0.359 ms
- 40 Execution Time: 40.659 ms

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,

Activities pgAdmin 4 Jun 9 12:37 AM ● pgAdmin 4

Servers schema3/postgres Local Data Query History Execute/Refresh

```

b2.color = 'green';

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

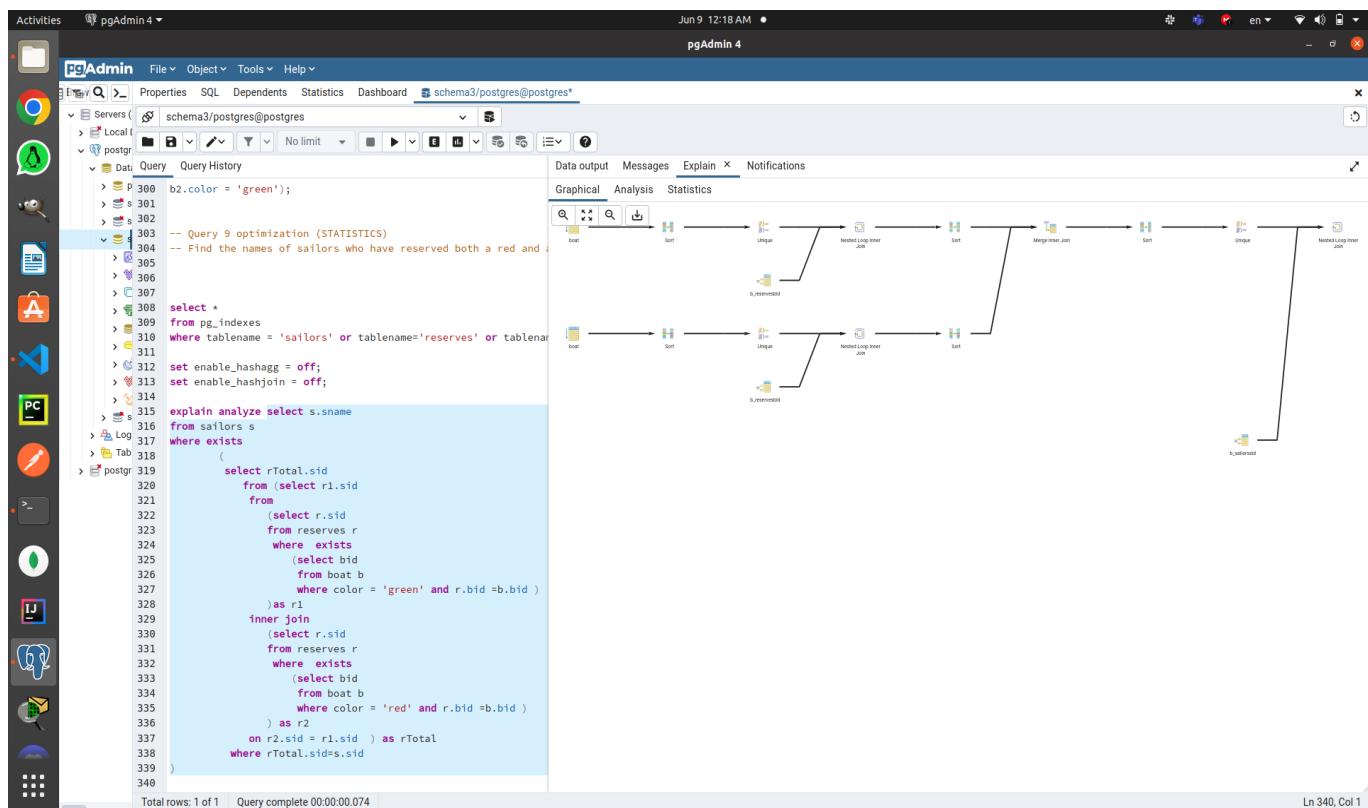
select *
from pg_indexes
where tablename = 'sailors' or tablename='reserves' or tablename='boat';

set enable_hashagg = off;
set enable_hashjoin = off;

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

```

Total rows: 4 of 4    Query complete 00:00:00.224    Ln 308, Col 1



The screenshot shows the pgAdmin 4 interface with the following details:

- Left Sidebar:** Shows the PostgreSQL database structure under "Servers (3)" and "Local Database".
- Top Bar:** Activities, pgAdmin 4, File, Object, Tools, Help.
- Central Area:**
  - Query Editor:** Displays the SQL code for Query 9, which finds sailors who have reserved both a red and a green boat. The code includes several nested queries and uses the EXPLAIN ANALYZE command to analyze the execution plan.
  - Data Output:** Shows the "QUERY PLAN" section with 30 numbered steps detailing the execution plan, including nested loops, sort operations, and index scans.
- Bottom Status:** Total rows: 37 of 37, Query complete 00:00:00.099, Ln 325, Col 1.

This screenshot shows the pgAdmin 4 interface with the same query and structure as the first one, but with a different execution plan due to a bug fix. The "QUERY PLAN" section shows 37 numbered steps, indicating a more efficient execution path.

## 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,

pgAdmin 4 - Jun 9 12:48 AM ● pgAdmin 4

```

Activities pgAdmin 4 ▾ File Object Tools Help
Servers schema3/postgres@postgres Local Postgr Data Query Jun 9 12:48 AM ● pgAdmin 4
Properties SQL Dependents Statistics Dashboard
schema3/postgres@postgres
No limit
Data History Execute Refresh
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

CREATE INDEX b_sailorssid ON sailors USING HASH(sid);
CREATE INDEX b_reservesID 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);

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

-- Query 9 (STATISTICS)
set enable_hashagg = off;
set enable_hashjoin = off;
explain analyze select s.sname
from sailors s, reserves r, boat b
where
s.sid = r.sid
and
r.bid = b.bid
;
```

Total rows: 5 of 5    Query complete 00:00:00.050

Ln 277, Col 1

pgAdmin 4 - Jun 9 1:14 AM ● pgAdmin 4

```

Activities pgAdmin 4 ▾ File Object Tools Help
Servers schema3/postgres@postgres Local Postgr Data Query Jun 9 1:14 AM ● pgAdmin 4
Properties SQL Dependents Statistics Dashboard
schema3/postgres@postgres
No limit
Data History Execute Refresh
b2.color = 'green');

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

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

set enable_hashagg = off;
set enable_hashjoin = off;

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

Total rows: 23 of 23 Query complete 00:00:00.176
```

Ln 316, Col 1

```

 graph LR
 A[t_reserves] --> B[Nested Loop Semi Join]
 B --> C[Nested Loop Inner Join]
 C --> D[Sort]
 D --> E[Unique]
 E --> F[Nested Loop Inner Join]
 G[t_reserv] --> B
 H[t_reserve] --> C
 I[t_sailorid] --> F

```

The screenshot shows the pgAdmin 4 interface with the 'Explain' tab selected. The query being explained is:

```

-- Find the names of sailors who have reserved both a red and a green boat
select * from pg_indexes where tablename = 'sailors' or tablename='reserves' or tablename='boats'
set enable_hashagg = off;
set enable_hashjoin = off;
explain analyze select s.sname
from sailors s
where exists
(
select rTotal.sid
from (select r1.sid
from
(select r.sid
from reserves r
where exists
(select bid
from boat b
where color = 'green' and r.bid = b.bid)
) as r1
inner join
(select r.sid
from reserves r
where exists
(select bid
from boat b
where color = 'red' and r.bid = b.bid)
) as r2
on r2.sid = r1.sid) as rTotal
where rTotal.sid=s.sid
)

```

The execution plan is as follows:

- Data output
- Messages
- Explain** (selected)
- Notifications

The plan consists of the following steps:

- Nested Loop Semi Join (between t\_reserves and t\_reserv)
- Nested Loop Inner Join (between t\_reserv and t\_reserve)
- Sort
- Unique
- Nested Loop Inner Join (between t\_reserve and t\_sailorid)

Success message: Successfully run. Total query runtime: 218 msec. 1 rows affected.

### Explanation :

- Metrics :

**Execution Time : 12.646 ms    Total Expected Cost : 2337.61**

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

#### 4. given query with BRIN indices only,

pgAdmin 4 - Jun 9 2:29 AM

```

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_boat ON boat USING BRIN(bid,color);

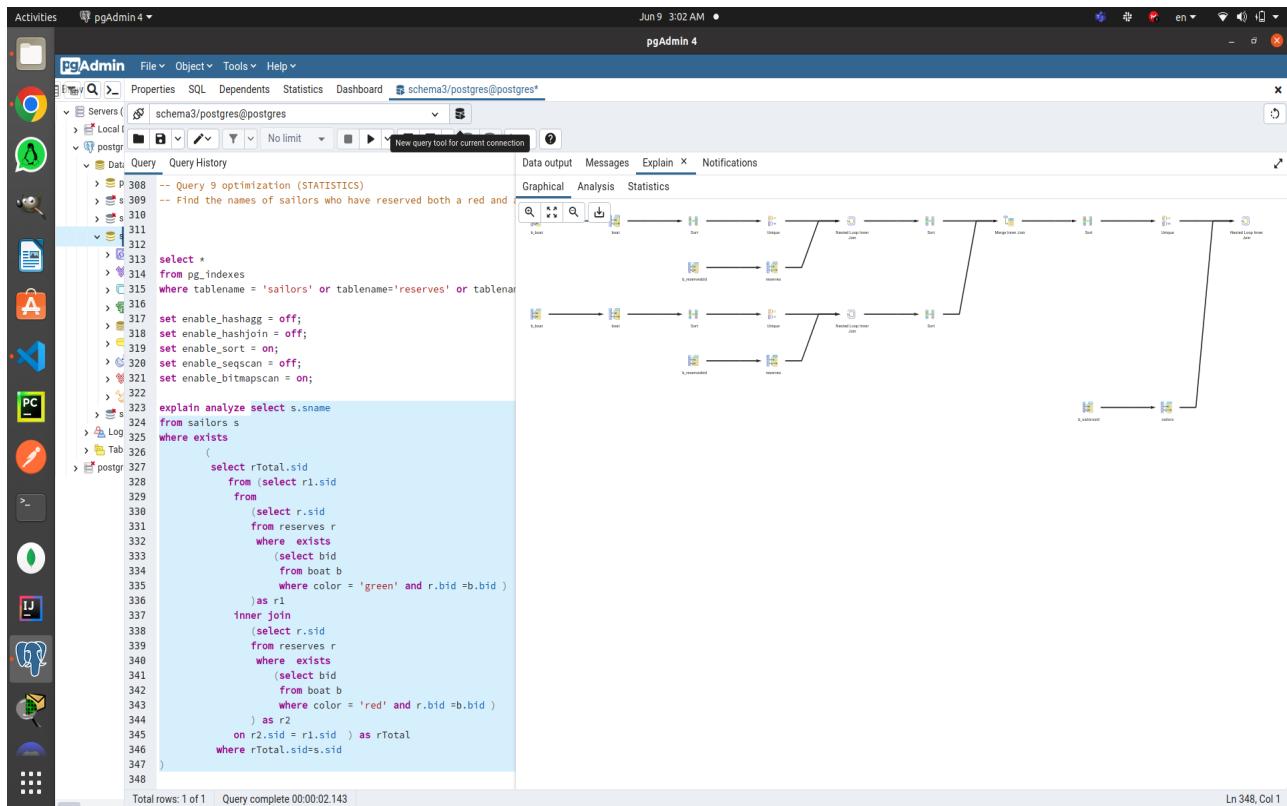
select *
from pg_indexes
where tablename = 'sailors' or tablename='reserves' or tablename='boat';

-- Query 9 (STATISTICS)
set enable_hashagg = off;
set enable_hashjoin = off;
explain analyze select s.sname
from sailors s, reserves r, boat b
where
s.sid = r.sid
and
r.bid = b.bid
and
b.color = 'red'
and
s.sid in (select s2.sid
from sailors s2, boat b2, reserves r2
where s2.sid = r2.sid
and
r2.bid = b2.bid
and
b2.color = 'green');

```

Total rows: 4 of 4    Query complete 00:00:00.088

Ln 273, Col 1



```

Activities pgAdmin 4 Jun 9 3:01 AM ● pgAdmin 4
Servers schema3/postgres Local No limit Data Query History Properties SQL Dependents Statistics Dashboard schema3/postgres@postgres*
P 308 -- Query 9 optimization (STATISTICS)
s 309 -- Find the names of sailors who have reserved both
e 310
i 311
o 312
n 313
r 314
t 315
w 316
d 317
h 318
s 319
l 320
q 321
b 322
x 323
e 324
L 325
T 326
p 327
Total rows: 59 of 59 Query complete 00:00:02.186
Jun 9 3:01 AM ● pgAdmin 4
Servers schema3/postgres Local No limit Data Query History Properties SQL Dependents Statistics Dashboard schema3/postgres@postgres*
P 308 -- Query 9 optimization (STATISTICS)
s 309 -- Find the names of sailors who have reserved both
e 310
i 311
o 312
n 313
r 314
t 315
w 316
d 317
h 318
s 319
l 320
q 321
b 322
x 323
e 324
L 325
T 326
p 327
Total rows: 59 of 59 Query complete 00:00:02.186

```

The screenshot shows two instances of pgAdmin 4. Both instances have the same session details at the top: "Activities pgAdmin 4", "Servers schema3/postgres@postgres\*", "Local", "No limit", "Data", "Query History", "Properties", "SQL", "Dependents", "Statistics", "Dashboard". The timestamp "Jun 9 3:01 AM" and application name "pgAdmin 4" are also present.

The left sidebar contains various icons for file operations, database management, and system monitoring.

The main area displays a SQL query and its corresponding query plan. The SQL query is:

```

-- Query 9 optimization (STATISTICS)
-- Find the names of sailors who have reserved both
SELECT *
FROM pg_indexes
WHERE tablename = 'sailors' OR tablename='reserves'
SET enable_hashagg = off;
SET enable_hashjoin = off;
SET enable_sort = on;
SET enable_seqscan = off;
SET enable_bitmapscan = on;

EXPLAIN ANALYSE SELECT s.sname
FROM sailors s
WHERE EXISTS
(
 SELECT rTotal.sid
 FROM (SELECT r1.sid
 FROM reserves r
 WHERE EXISTS
 (
 SELECT bid
 FROM boat b
 WHERE color = 'green' AND r.bid = r1.sid
) AS r1
 INNER JOIN
 (
 SELECT r2.sid
 FROM reserves r
 WHERE EXISTS
 (
 SELECT bid
 FROM boat b
 WHERE color = 'red' AND r.bid = r2.sid
) AS r2
 ON r2.sid = r1.sid) AS rTotal
 WHERE rTotal.sid=s.sid
);

```

The first instance of pgAdmin shows a very detailed and long query plan (30+ lines) with numerous specific statistics and execution details. The second instance shows a much shorter and simplified query plan (59 lines), indicating that some parts of the original plan were collapsed or simplified.

### 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)

pgAdmin 4 - Jun 9 3:08 AM • pgAdmin 4

```

SELECT r.sid
FROM reserves r
WHERE EXISTS
 (SELECT bid
 FROM boat b
 WHERE color = 'red' AND r.bid = b.bid)
) AS r2
 ON r2.sid = r1.sid) AS rTotal
WHERE rTotal.sid = s.sid
;
```

| schemaname | tablename | indexname     | tablespace | indexdef                                                       |
|------------|-----------|---------------|------------|----------------------------------------------------------------|
| public     | sailors   | b_sailorssid  | [null]     | CREATE INDEX b_sailorssid ON public.sailors USING hash (sid)   |
| public     | boat      | b_boat        | [null]     | CREATE INDEX b_boat1 ON public.boat USING hash (bid)           |
| public     | boat      | b_boat2       | [null]     | CREATE INDEX b_boat2 ON public.boat USING btree (color)        |
| public     | reserves  | b_reservesid  | [null]     | CREATE INDEX b_reservesid ON public.reserves USING hash (sid)  |
| public     | reserves  | b_reservesbid | [null]     | CREATE INDEX b_reservesbid ON public.reserves USING hash (bid) |

pgAdmin 4 - Jun 9 3:27 AM • pgAdmin 4

```

SELECT *
FROM pg_indexes
WHERE tablename = 'sailors' OR tablename='reserves' OR tablename='boat';
;
```

The explain plan diagram illustrates the execution flow of the query. It starts with a 'Nested Loop Join' between the 'reserves' table (labeled 'b\_reserves') and the 'boat' table (labeled 'b\_boat'). The result of this join is then used in another 'Nested Loop Join' with the 'boat' table (labeled 'b\_boat1'). This results in a 'Sort' operation, which is then followed by a 'Unique' operation to remove duplicates. Finally, a 'Nested Loop Join' is performed with the 'sailors' table (labeled 'b\_sailors') to produce the final result.

```

b2.color = 'green');

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

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

set enable_hashagg = off;
set enable_hashjoin = off;

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

```

QUERY PLAN

```

text
1 Nested Loop (cost=2231.15..2337.61 rows=1324 width=21) (actual time=55.701..55.987 rows=177 loops=1)
2 -> Unique (cost=2231.15..2237.77 rows=1324 width=8) (actual time=55.692..55.732 rows=177 loops=1)
3 -> Sort (cost=2231.15..2234.46 rows=1324 width=8) (actual time=55.691..55.705 rows=177 loops=1)
4 Sort Key: r.sid
5 Sort Method: quicksort Memory: 33kB
6 -> Nested Loop Semi Join (cost=0.00..2162.50 rows=1324 width=8) (actual time=1.639..55.655 rows=177 loops=1)
7 -> Nested Loop (cost=0.00..1918.03 rows=9281 width=12) (actual time=0.032..51.323 rows=3376 loops=1)
8 -> Nested Loop Semi Join (cost=0.00..1461.82 rows=4982 width=4) (actual time=0.027..48.169 rows=1136 loops=1)
9 -> Seq Scan on reserves r_1 (cost=0.00..540.00 rows=35000 width=8) (actual time=0.013..3.622 rows=35000 loops=1)
10 -> Index Scan using b_boat on boat b_1 (cost=0.00..0.02 rows=1 width=4) (actual time=0.001..0.001 rows=0 loops=35000)
11 Index Cond: (bid = r_1.bid)
12 Filter: (color = 'red':bpchar)
13 Rows Removed by Filter: 1
14 -> Index Scan using b_reservesid on reserves r (cost=0.00..0.07 rows=2 width=8) (actual time=0.001..0.002 rows=3 loops=1136)
15 Index Cond: (sid = r_1.sid)
16 -> Index Scan using b_boat on boat b (cost=0.00..0.02 rows=1 width=4) (actual time=0.001..0.001 rows=0 loops=3376)
17 Index Cond: (bid = r.bid)
18 Filter: (color = 'green':bpchar)
19 Rows Removed by Filter: 1
20 -> Index Scan using b_sailorssid on sailors s (cost=0.00..0.07 rows=1 width=25) (actual time=0.001..0.001 rows=1 loops=177)
21 Index Cond: (sid = r_1.sid)
22 Planning Time: 0.448 ms
23 Execution Time: 56.035 ms

```

Successfully run. Total query runtime: 156 msec. 23 rows affected.

Total rows: 23 of 23    Query complete 00:00:00.156    Ln 342, Col 1

### 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 approximatlly maded to be O(n) performance.
- The Where clause condition on the color used B-tree which is more better as colors are repeatable 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 .