

## Query 9

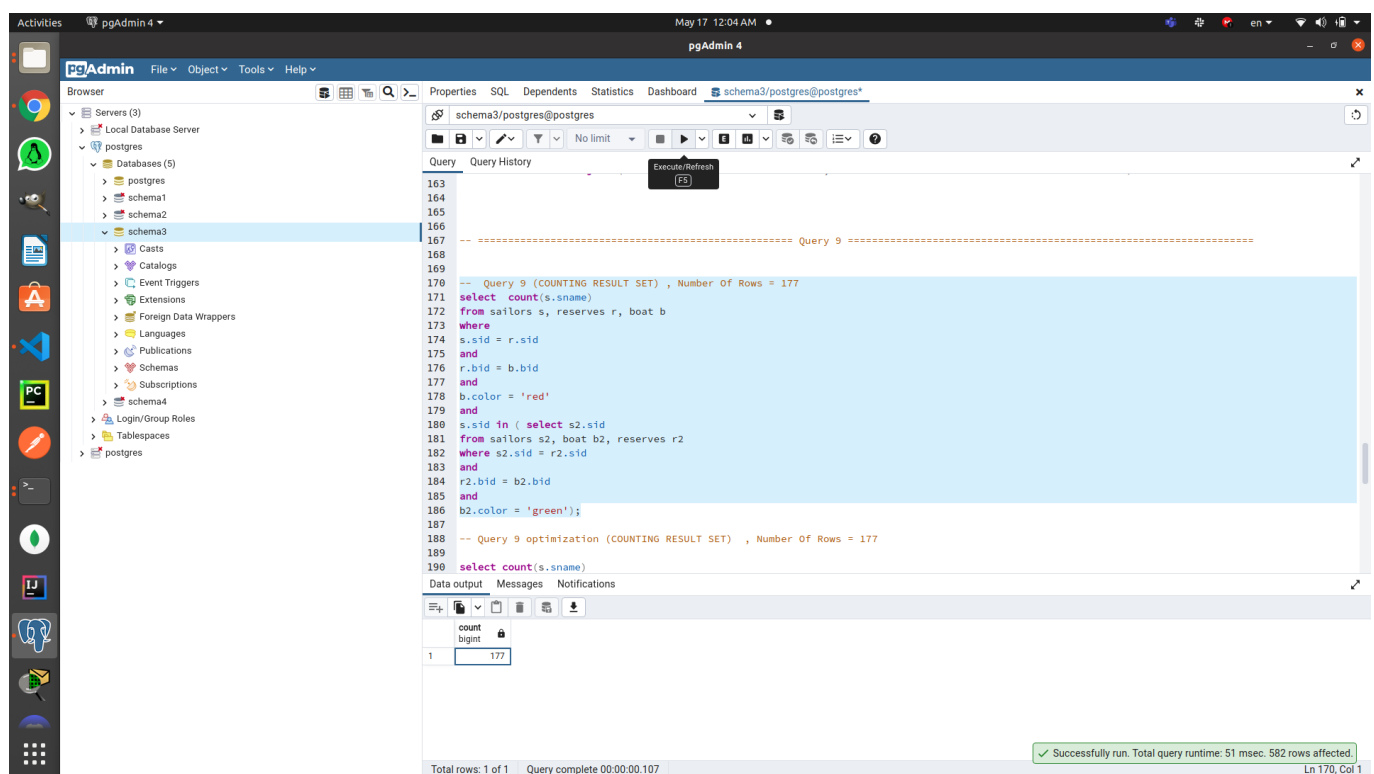
- Find the names of sailors who have reserved both a red and a green boat.

### Original Query

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

### Result Set

- 177 Rows



The screenshot shows the pgAdmin 4 interface. The SQL editor displays the following query (lines 163-190):

```
163
164
165
166 ----- Query 9 -----
167
168
169
170 -- Query 9 (COUNTING RESULT SET) , Number Of Rows = 177
171 select count(s.sname)
172 from sailors s, reserves r, boat b
173 where
174 s.sid = r.sid
175 and
176 r.bid = b.bid
177 and
178 b.color = 'red'
179 and
180 s.sid in ( select s2.sid
181 from sailors s2, boat b2, reserves r2
182 where s2.sid = r2.sid
183 and
184 r2.bid = b2.bid
185 and
186 b2.color = 'green');
187
188 -- Query 9 optimization (COUNTING RESULT SET) , Number Of Rows = 177
189
190 select count(s.sname)
```

The Data output pane shows the result set:

count	bigint
1	177

The status bar at the bottom indicates: "Successfully run. Total query runtime: 51 msec. 582 rows affected. Ln 170, Col 1"

### Report

1. given query without an index,

pgAdmin 4

schema3/postgres@postgres\*

```
15
16 select count(sid)
17 from sailors;
18
19 select count(*)
20 from boat;
21
22 select count(*)
23 from reserves;
24
25 delete from Reserves;
26
27 delete from sailors;
28
29 delete from Boat;
30
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	tablename	indexname	tablespace	indexdef
name	name	name	name	text

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

Total rows: 0 of 0 Query complete 00:00:00.117 Ln 35, Col 1

pgAdmin 4

schema3/postgres@postgres\*

214 and

Data output Messages Explain X Notifications

Graphical Analysis Statistics

Total rows: 1 of 1 Query complete 00:00:00.123 Ln 226, Col 1

The screenshot shows the pgAdmin 4 interface. The left pane displays the database structure, including tables like 'sailors', 'reserves', and 'boats'. The main pane shows a SQL query with an 'explain analyze' statement. The query is as follows:

```

-- Query 9 (STATISTICS)
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' );

```

The right pane shows the 'QUERY PLAN' for the query. It details the execution steps, including Hash Joins, Seq Scans, and Filters, along with their respective costs, widths, and execution times.

### Explanation :

- Metrics :

**Execution Time : 33.462 ms    Total Expected Cost : 1968.71**

- First , a Sequential Scan occurred on boat b2 Table to filter the boats with color green .
- Second , a Hash Table was built on the run on b2 attributes cause a Hash Join this resulted with 1024 Buckets created ,and Memory Usage of 24 KB.
- Thirdly , a Sequential Scan occurred on reserves r2 Table to be able to hash each row's sid to the Boat b2's buckets to full inner join on the condition r2.bid= b2.bid .
- Fourthly , on the intermediate results a Hash table was built on it.
- Fifth , a Sequential Scan occurred on reserves s2 Table to be able to hash each row's sid to the Boat b2's buckets to full inner join on the condition s2.sid= r2.sid
- Sixth , we repeat the process again with s1,b1,r1 and inner join all the results together with a hash semi join and based on the conditions

1. given query with B+ trees indices only,

•

3. given query with hash indices only,

•

4. given query with BRIN indices only,

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

## 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,

pgAdmin 4

schema3/postgres/postgres\*

Query

```
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

schemaname	tablename	indexname	tablespace	indexdef
name	name	name	name	text

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

Total rows: 0 of 0 Query complete 00:00:00.117 Ln 35, Col 1

The top screenshot shows the pgAdmin 4 interface with a query window displaying the following SQL query:

```

294
295 explain analyze select s.sname
296 from sailors s
297

```

The query plan diagram shows a Hash Semi Join operation on the 'sailors' table, followed by a Hash Inner Join operation. The bottom screenshot shows a more complex query plan for a query optimization exercise, including a Hash Semi Join and Hash Inner Join, with a detailed 'Data output' window showing the query plan and execution statistics.

The bottom screenshot shows the pgAdmin 4 interface with a query window displaying the following SQL query:

```

299
300 -- Query 9 optimization (STATISTICS)
301 -- Find the names of sailors who have reserved both a red and a green boat.
302
303
304 explain analyze select s.sname
305 from sailors s
306 where exists
307 (
308   select rTotal.sid
309   from (select r1.sid
310         from reserves r1
311         where exists
312             (select bid
313              from boat b
314              where color = 'green' and r1.bid = b.bid)
315        ) as r1
316   inner join
317   (select r2.sid
318    from reserves r2
319    where exists
320        (select bid
321         from boat b
322         where color = 'red' and r2.bid = b.bid)
323        ) as r2
324   on r2.sid = r1.sid ) as rTotal
325 where rTotal.sid=s.sid
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344

```

The 'Data output' window shows the following query plan and execution statistics:

Step	Operation	Cost	Rows	Width	Actual Time	Actual Rows	Loops
1	Hash Semi Join	(cost=1621.20..2034.80 rows=1324 width=21)	13.866	16.943	rows=177 loops=1		
2	Hash Cond	(s.sid = r1.sid)					
3	Seq Scan on sailors s	(cost=0.00..349.00 rows=19000 width=25)	0.007	1.498	rows=19000 loops=1		
4	Seq Scan on reserves r	(cost=1604.65..1604.65 rows=1324 width=8)	13.819	13.824	rows=177 loops=1		
5	Buckets	2048 Batches: 1 Memory Usage: 23K					
6	Seq Scan on reserves r	(cost=0.00..540.00 rows=35000 width=8)	0.005	2.795	rows=35000 loops=1		
7	Hash Cond	(r1.sid = r2.sid)					
8	Hash Semi Join	(cost=67.85..755.27 rows=4993 width=4)	0.503	6.706	rows=2064 loops=1		
9	Hash Cond	(r1.bid = b.bid)					
10	Seq Scan on reserves r	(cost=0.00..540.00 rows=35000 width=8)	0.005	2.795	rows=35000 loops=1		
11	Hash	(cost=62.50..62.50 rows=428 width=4)	0.360	0.360	rows=428 loops=1		
12	Buckets	1024 Batches: 1 Memory Usage: 24K					
13	Seq Scan on boat b	(cost=0.00..62.50 rows=428 width=4)	0.046	0.311	rows=428 loops=1		
14	Filter	(color = 'green'::bpchar)					
15	Rows Removed by Filter	2572					
16	Hash	(cost=755.14..755.14 rows=4982 width=4)	6.886	6.887	rows=1136 loops=1		
17	Buckets	8192 Batches: 1 Memory Usage: 104K					
18	Hash Semi Join	(cost=67.84..755.14 rows=4982 width=8)	0.357	6.742	rows=1136 loops=1		
19	Hash Cond	(r1.bid = b1.bid)					
20	Seq Scan on reserves r1	(cost=0.00..540.00 rows=35000 width=8)	0.004	2.869	rows=35000 loops=1		
21	Hash	(cost=62.50..62.50 rows=427 width=4)	0.350	0.350	rows=427 loops=1		
22	Buckets	1024 Batches: 1 Memory Usage: 24K					
23	Seq Scan on boat b1	(cost=0.00..62.50 rows=427 width=4)	0.004	0.303	rows=427 loops=1		
24	Filter	(color = 'red'::bpchar)					
25	Rows Removed by Filter	2573					
26	Planning Time	0.338 ms					
27	Execution Time	16.981 ms					

### Explanation :

- First , a Sequential Scan occurred on Reserves Table to filter the reserved boats with bid 103 and it costs 0..627 rows, and read through 35000 Rows and 582 Rows are left after the filtration of the where condition.
- Second , a Hash Table was built on the run on r.bid of the 582 Rows and it costed 627.50...627.50 rows, this resulted with 1024 Buckets created , and Memory Usage of 29 KB.
- Thirdly , a Sequential Scan occurred on Sailors Table to be able to hash each row's sid to the Reserves's buckets to full inner join on the condition 00..349 rows, and it read through

19000 rows.

- Fourthly, a Hash Semi Join occurred to produce the result set of the condition of  $s.sid = r.sid$ , and the reason it is Hash Semi Join In the first query, only the  $r.sid$  needs to be saved from the reserves into the hash table, because that is the only data needed to implement the semi-join, it costs 634.77 ... 1084.60 rows.
- Execution Time : 10.462 ms

2. given query with B+ trees indices only,

- 

3. given query with hash indices only,

- 

4. given query with BRIN indices only,

- 

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

-