Schema 1

Query (1) :

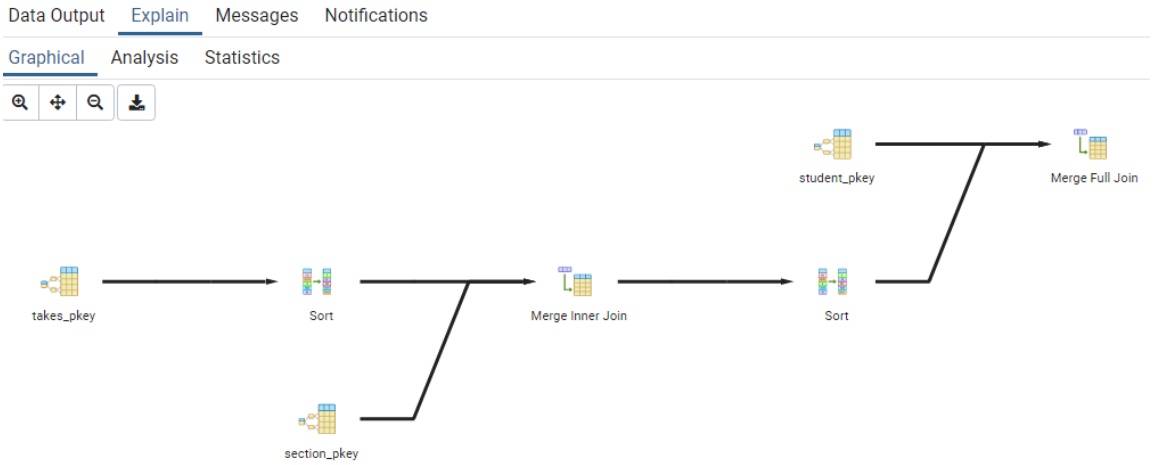
```
16  set enable_partitionwise_join= on;
17  set enable_partitionwise_aggregate= on;
18  set enable_seqscan= OFF;
19  set enable_sort= on;
20  set enable_tidscan = on;
21
22  Explain Analyze
23  select *
24  from (select *
25  from student
26  where
27  department = 'CSEN') as CS1_student
28  full outer join
29  (select *
30  from takes t inner join section s
31  on t.section_id = s.section_id
32  where semester = 1
33  and
34  year = 2019) as sem1_student
35  on CS1_student.id = sem1_student.student_id;
36
```

Query plan of Query (1) before optimization :
Cost = 2660.05 rows
Execution time = 37.852 msec

Data Output | Explain | Messages | Notifications

	QUERY PLAN text
1	Merge Full Join (cost=273.77..2660.05 rows=1249 width=64) (actual time=2.901..37.611 rows=1249 loops=1)
2	[...] Merge Cond: (student.id = t.student_id)
3	[...] -> Index Scan using student_pkey on student (cost=0.29..2377.33 rows=1114 width=24) (actual time=0.029..33.638 rows=1100 loops=1
4	[...] Filter: ((department)::text = 'CSEN'::text)
5	[...] Rows Removed by Filter: 64959
6	[...] -> Sort (cost=273.48..276.60 rows=1249 width=40) (actual time=2.867..2.992 rows=1249 loops=1)
7	[...] Sort Key: t.student_id
8	[...] Sort Method: quicksort Memory: 146kB
9	[...] -> Merge Join (cost=125.40..209.24 rows=1249 width=40) (actual time=0.735..2.222 rows=1249 loops=1)
10	[...] Merge Cond: (t.section_id = s.section_id)
11	[...] -> Sort (cost=125.13..128.25 rows=1249 width=12) (actual time=0.709..0.824 rows=1249 loops=1)
12	[...] Sort Key: t.section_id
13	[...] Sort Method: quicksort Memory: 107kB
14	[...] -> Index Scan using takes_pkey on takes t (cost=0.28..60.89 rows=1249 width=12) (actual time=0.042..0.409 rows=1249 loops=1)
15	[...] -> Index Scan using section_pkey on section s (cost=0.28..62.26 rows=1249 width=28) (actual time=0.022..0.549 rows=1249 loops=1)
16	[...] Filter: ((semester = 1) AND (year = 2019))
17	Planning Time: 0.560 ms
18	Execution Time: 37.852 ms

Physical plan of Query (1) before optimization :



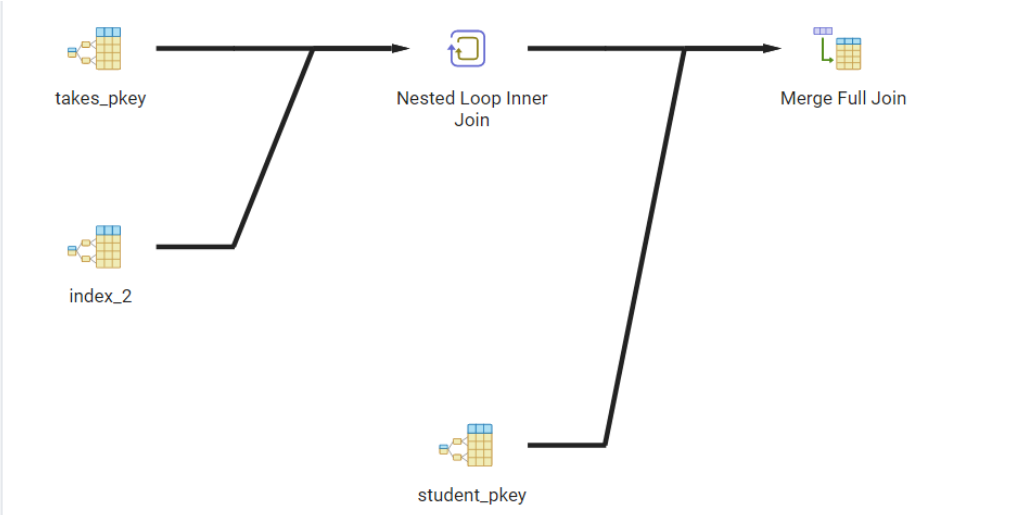
Query plan of Query (1) after using hash index on table “Section” on section_id :
Cost = 2552.35 rows
Execution time = 14.735 msec

```
22  create index index_2 on Section using hash (section_id);
23  drop index index_2;
24
25   Explain Analyze
26  select *
27  from (select *
```

Data Output | Explain | Messages | Notifications

	QUERY PLAN text
1	Merge Full Join (cost=0.57..2552.35 rows=1249 width=64) (actual time=0.045..14.650 rows=1249 loops=1)
2	[...] Merge Cond: (t.student_id = student.id)
3	[...] -> Nested Loop (cost=0.28..168.90 rows=1249 width=40) (actual time=0.025..2.080 rows=1249 loops=1)
4	[...] -> Index Scan using takes_pkey on takes t (cost=0.28..60.89 rows=1249 width=12) (actual time=0.012..0.245 rows=1249 loops=1)
5	[...] -> Index Scan using index_2 on section s (cost=0.00..0.09 rows=1 width=28) (actual time=0.001..0.001 rows=1 loops=1249)
6	[...] Index Cond: (section_id = t.section_id)
7	[...] Filter: ((semester = 1) AND (year = 2019))
8	[...] -> Index Scan using student_pkey on student (cost=0.29..2377.33 rows=1114 width=24) (actual time=0.018..12.139 rows=1100 loops=1)
9	[...] Filter: ((department)::text = 'CSEN'::text)
10	[...] Rows Removed by Filter: 64959
11	Planning Time: 0.803 ms
12	Execution Time: 14.735 ms

Physical plan of Query (1) after using hash index on table “Section” on section_id :



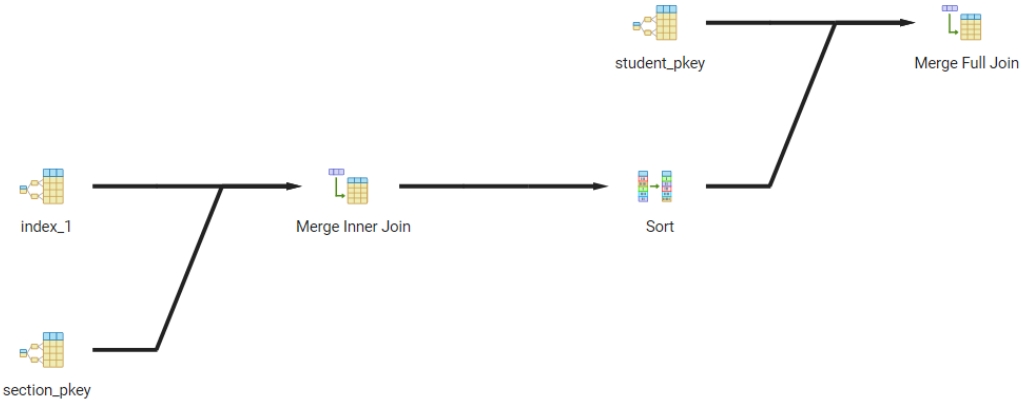
Query plan of Query (1) after using btree index on table “takes” on section_id :
Cost = 2584.81 rows
Execution time = 28.370 msec

```
22  create index index_1 on takes using Btree (section_id);
23  drop index index_1;
24  Explain Analyze
25  select *
26  from (select *
```

Data Output | Explain | Messages | Notifications

	QUERY PLAN text
1	Merge Full Join (cost=198.54..2584.81 rows=1249 width=64) (actual time=2.184..28.175 rows=1249 loops=1)
2	[...] Merge Cond: (student.id = t.student_id)
3	[...] -> Index Scan using student_pkey on student (cost=0.29..2377.33 rows=1114 width=24) (actual time=0.023..24.798 rows=1100 loops...
4	[...] Filter: ((department)::text = 'CSEN'::text)
5	[...] Rows Removed by Filter: 64959
6	[...] -> Sort (cost=198.24..201.37 rows=1249 width=40) (actual time=2.154..2.309 rows=1249 loops=1)
7	[...] Sort Key: t.student_id
8	[...] Sort Method: quicksort Memory: 146kB
9	[...] -> Merge Join (cost=0.56..134.01 rows=1249 width=40) (actual time=0.042..1.593 rows=1249 loops=1)
10	[...] Merge Cond: (t.section_id = s.section_id)
11	[...] -> Index Scan using index_1 on takes t (cost=0.28..53.01 rows=1249 width=12) (actual time=0.024..0.418 rows=1249 loops=1)
12	[...] -> Index Scan using section_pkey on section s (cost=0.28..62.26 rows=1249 width=28) (actual time=0.015..0.488 rows=1249 loops=1)
13	[...] Filter: ((semester = 1) AND (year = 2019))
14	Planning Time: 1.563 ms
15	Execution Time: 28.370 ms

Physical plan of Query (1) after using btree index on table “takes” on section_id :



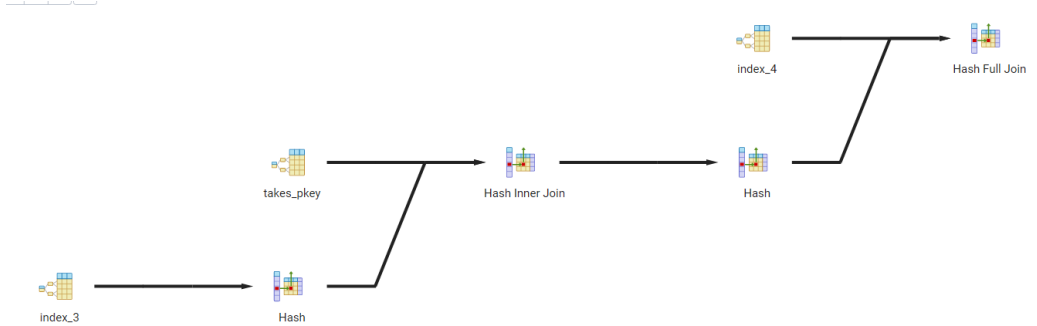
Query plan of Query (1) after using mixed indices on table “takes” on section_id :
Btree index on table section using “semester”
Hash index on table student using “department”
Cost = 2137.42 rows
Execution time = 2.322 msec

```
24
25  create index index_4 on student using hash (department);
26  create index index_3 on section using Btree (semester);
27
28  Explain Analyze
29  select *
30  from (select *
31  from student
32  where
33  department = 'CSEN') as CS1_student
34  full outer join
35  (select *
36  from takes t inner join section s
37  on t.section_id = s.section_id
38  where semester = 1
```

Data Output | Explain | Messages | Notifications

	QUERY PLAN text
1	Hash Full Join (cost=141.53..2137.42 rows=1249 width=64) (actual time=1.523..2.193 rows=1249 loops=1)
2	[...] Hash Cond: (student.id = t.student_id)
3	[...] -> Index Scan using index_4 on student (cost=0.00..1991.50 rows=1114 width=24) (actual time=0.013..0.265 rows=1100 loops=1)
4	[...] Index Cond: ((department)::text = 'CSEN'::text)
5	[...] -> Hash (cost=125.92..125.92 rows=1249 width=40) (actual time=1.499..1.500 rows=1249 loops=1)
6	[...] Buckets: 2048 Batches: 1 Memory Usage: 104kB
7	[...] -> Hash Join (cost=62.02..125.92 rows=1249 width=40) (actual time=0.509..1.163 rows=1249 loops=1)
8	[...] Hash Cond: (t.section_id = s.section_id)
9	[...] -> Index Scan using takes_pkey on takes t (cost=0.28..60.89 rows=1249 width=12) (actual time=0.024..0.253 rows=1249 loops=1)
10	[...] -> Hash (cost=46.13..46.13 rows=1249 width=28) (actual time=0.473..0.474 rows=1249 loops=1)
11	[...] Buckets: 2048 Batches: 1 Memory Usage: 90kB
12	[...] -> Index Scan using index_3 on section s (cost=0.15..46.13 rows=1249 width=28) (actual time=0.015..0.284 rows=1249 loops=1)
13	[...] Index Cond: (semester = 1)
14	[...] Filter: (year = 2019)
15	Planning Time: 0.334 ms
16	Execution Time: 2.322 ms

Physical plan of Query (1) after using mixed indices on table “takes” on section_id :
Btree index on table section using “semester”
Hash index on table student using “department”



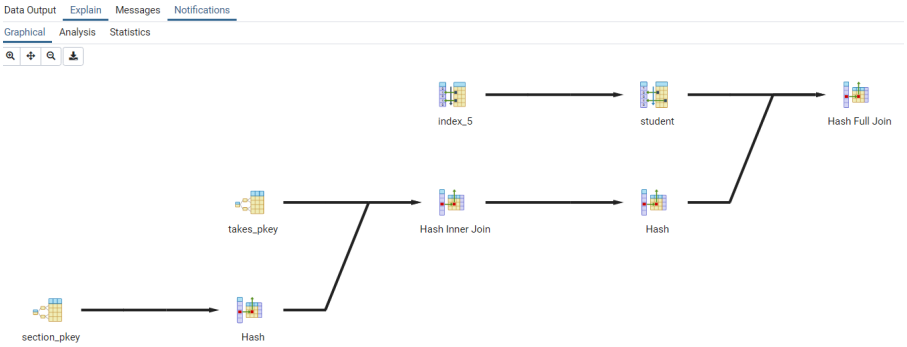
Query plan of Query (1) after using brin index on table “student” on department :
Cost = 1014.77 rows
Execution time = 4.729 msec

```
24
25 create index index_5 on student using BRIN (department);
26
27
28 Explain Analyze
29 select *
```

Data Output | Explain | Messages | Notifications

	QUERY PLAN text
1	Hash Full Join (cost=169.99..1014.77 rows=1249 width=64) (actual time=1.339..4.584 rows=1249 loops=1)
2	[...] Hash Cond: (student.id = t.student_id)
3	[...] -> Bitmap Heap Scan on student (cost=12.33..852.72 rows=1114 width=24) (actual time=0.042..2.672 rows=1100 loops=1)
4	[...] Recheck Cond: ((department)::text = 'CSEN'::text)
5	[...] Rows Removed by Index Recheck: 16308
6	[...] Heap Blocks: lossy=128
7	[...] -> Bitmap Index Scan on index_5 (cost=0.00..12.05 rows=28351 width=0) (actual time=0.033..0.034 rows=1280 loops=1)
8	[...] Index Cond: ((department)::text = 'CSEN'::text)
9	[...] -> Hash (cost=142.05..142.05 rows=1249 width=40) (actual time=1.284..1.287 rows=1249 loops=1)
10	[...] Buckets: 2048 Batches: 1 Memory Usage: 104kB
11	[...] -> Hash Join (cost=78.15..142.05 rows=1249 width=40) (actual time=0.434..0.991 rows=1249 loops=1)
12	[...] Hash Cond: (t.section_id = s.section_id)
13	[...] -> Index Scan using takes_pkey on takes t (cost=0.28..60.89 rows=1249 width=12) (actual time=0.017..0.207 rows=1249 loops=1)
14	[...] -> Hash (cost=62.26..62.26 rows=1249 width=28) (actual time=0.406..0.408 rows=1249 loops=1)
15	[...] Buckets: 2048 Batches: 1 Memory Usage: 90kB
16	[...] -> Index Scan using section_pkey on section s (cost=0.28..62.26 rows=1249 width=28) (actual time=0.011..0.254 rows=1249 loops=1)
17	[...] Filter: ((semester = 1) AND (year = 2019))
18	Planning Time: 0.554 ms
19	Execution Time: 4.729 ms

Physical plan of Query (1) after using brin index on table “student” on department :

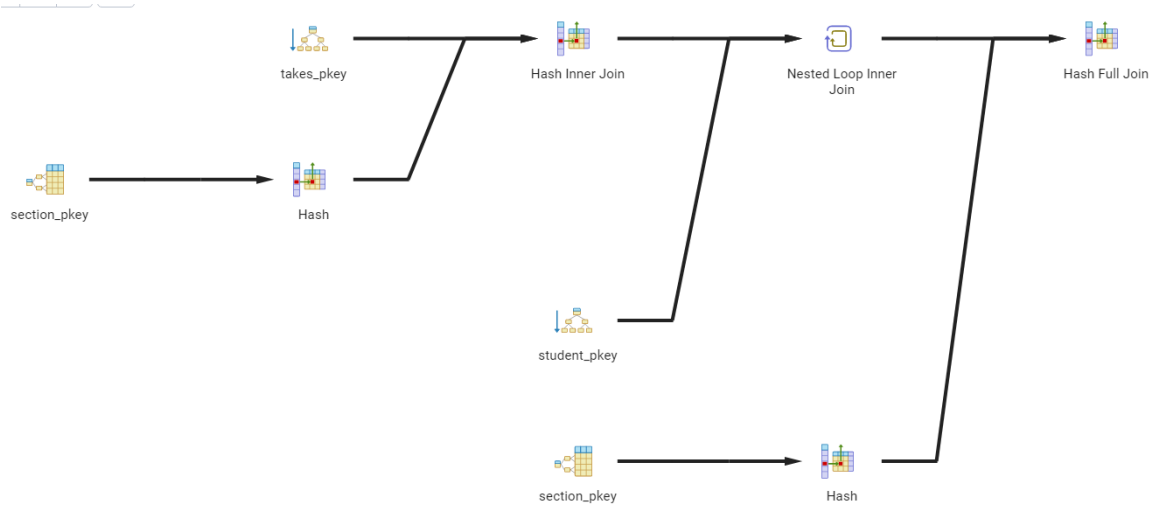


Query (1) after optimization:
Cost = 765.14 rows
Execution time = 1.588 msec

```
SET enable_seqscan = OFF;
SET enable_bitmapscan= on;
SET enable_hashagg=on;
SET enable_hashjoin=on;
SET enable_incremental_sort=on;
SET enable_indexscan=On;
SET enable_indexonlyscan=On;
SET enable_memoize=on;
SET enable_mergejoin=off;
SET enable_nestloop =on;
SET enable_sort=on;
SET enable_tidscan=on;
```

```
explain analyze
select * from (select sec.* from student s
inner join takes t on t.student_id=s.id
inner join section sec on t.section_id=sec.section_id
where sec.semester=1 and year =2019) as x
full outer join
(select * from section sec2
where sec2.semester=2 and year=2019) as y
on x.section_id=y.section_id
```

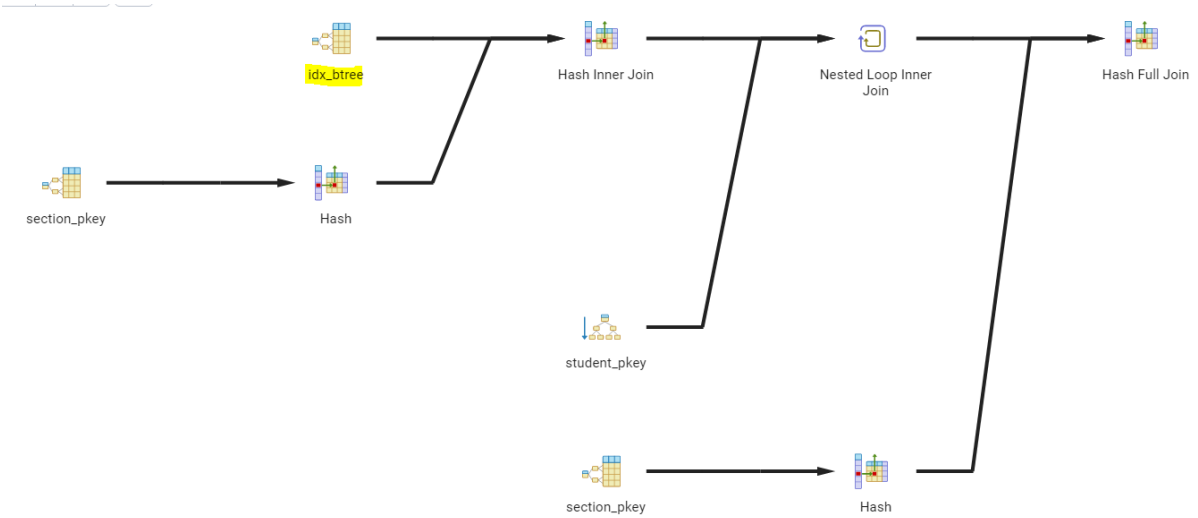
QUERY PLAN
text
Hash Full Join (cost=140.75..765.14 rows=625 width=56) (actual time=0.457..1.504 rows=1250 loops=1)
Hash Cond: (sec.section_id = sec2.section_id)
-> Nested Loop (cost=70.66..693.41 rows=624 width=28) (actual time=0.267..1.152 rows=625 loops=1)
-> Hash Join (cost=70.37..134.27 rows=624 width=32) (actual time=0.257..0.593 rows=625 loops=1)
Hash Cond: (t.section_id = sec.section_id)
-> Index Only Scan using takes_pkey on takes t (cost=0.28..60.89 rows=1249 width=8) (actual time=0.010..0.174 rows=1249 loops=1)
Heap Fetches: 1249
-> Hash (cost=62.28..62.28 rows=625 width=28) (actual time=0.241..0.242 rows=625 loops=1)
Buckets: 1024 Batches: 1 Memory Usage: 45kB
-> Index Scan using section_pkey on section sec (cost=0.28..62.28 rows=625 width=28) (actual time=0.005..0.198 rows=625 loops=1)
Filter: ((semester = 1) AND (year = 2019))
Rows Removed by Filter: 625
-> Index Only Scan using student_pkey on student s (cost=0.29..0.90 rows=1 width=4) (actual time=0.001..0.001 rows=1 loops=625)
Index Cond: (id = t.student_id)
Heap Fetches: 0
-> Hash (cost=62.28..62.28 rows=625 width=28) (actual time=0.183..0.183 rows=625 loops=1)
Buckets: 1024 Batches: 1 Memory Usage: 45kB
-> Index Scan using section_pkey on section sec2 (cost=0.28..62.28 rows=625 width=28) (actual time=0.064..0.137 rows=625 loops=1)
Filter: ((semester = 2) AND (year = 2019))
Rows Removed by Filter: 625
Planning Time: 0.235 ms
Execution Time: 1.588 ms



1) Optimized query (1) with btree:
   Cost = 757.27 rows
   Execution time = 1.572 msec

```
create index idx_btree on takes using btree(section_id)
```

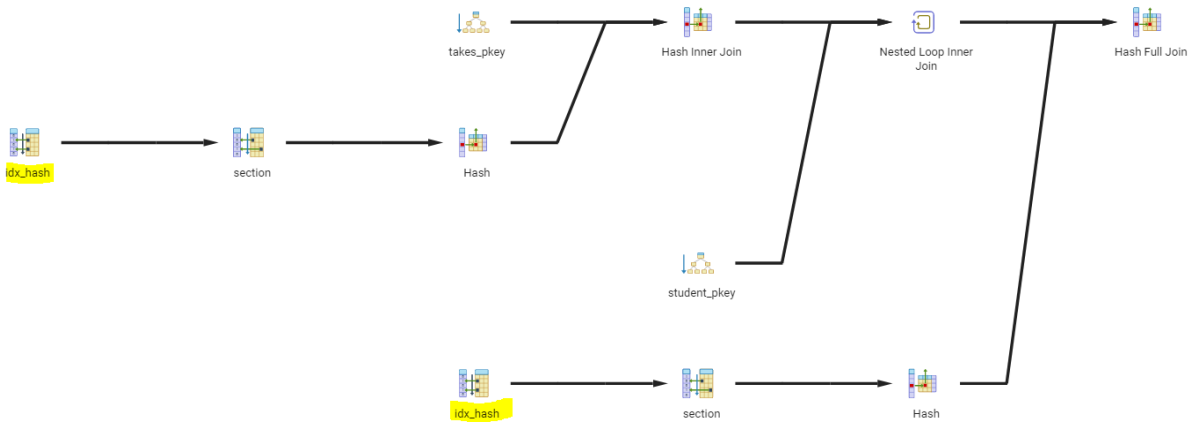
QUERY PLAN text
Hash Full Join (cost=140.75..757.27 rows=625 width=56) (actual time=0.386..1.442 rows=1250 loops=1)
Hash Cond: (sec.section_id = sec2.section_id)
-> Nested Loop (cost=70.66..685.54 rows=624 width=28) (actual time=0.200..1.093 rows=625 loops=1)
-> Hash Join (cost=70.37..126.39 rows=624 width=32) (actual time=0.192..0.486 rows=625 loops=1)
Hash Cond: (t.section_id = sec.section_id)
-> Index Scan using idx_btree on takes t (cost=0.28..53.01 rows=1249 width=8) (actual time=0.019..0.151 rows=1249 loops=1)
-> Hash (cost=62.28..62.28 rows=625 width=28) (actual time=0.166..0.166 rows=625 loops=1)
Buckets: 1024 Batches: 1 Memory Usage: 45kB
-> Index Scan using section_pkey on section sec (cost=0.28..62.28 rows=625 width=28) (actual time=0.005..0.124 rows=625 loops=1)
Filter: ((semester = 1) AND (year = 2019))
Rows Removed by Filter: 625
-> Index Only Scan using student_pkey on student s (cost=0.29..0.90 rows=1 width=4) (actual time=0.001..0.001 rows=1 loops=625)
Index Cond: (id = t.student_id)
Heap Fetches: 0
-> Hash (cost=62.28..62.28 rows=625 width=28) (actual time=0.182..0.182 rows=625 loops=1)
Buckets: 1024 Batches: 1 Memory Usage: 45kB
-> Index Scan using section_pkey on section sec2 (cost=0.28..62.28 rows=625 width=28) (actual time=0.061..0.133 rows=625 loops=1)
Filter: ((semester = 2) AND (year = 2019))
Rows Removed by Filter: 625
Planning Time: 0.656 ms
Execution Time: 1.572 ms



2) Optimized query (1) with hash:
Cost = 745.02 rows
Execution time = 2.010 msec

```
create index idx_hash on section using hash(semester)
```

QUERY PLAN text
Hash Full Join (cost=120.63..745.02 rows=625 width=56) (actual time=0.434..1.848 rows=1250 loops=1)
Hash Cond: (sec.section_id = sec2.section_id)
-> Nested Loop (cost=60.60..683.36 rows=624 width=28) (actual time=0.243..1.427 rows=625 loops=1)
-> Hash Join (cost=60.31..124.21 rows=624 width=32) (actual time=0.234..0.705 rows=625 loops=1)
Hash Cond: (t.section_id = sec.section_id)
-> Index Only Scan using takes_pkey on takes t (cost=0.28..60.89 rows=1249 width=8) (actual time=0.069..0.348 rows=1249 loops=1)
Heap Fetches: 1249
-> Hash (cost=52.22..52.22 rows=625 width=28) (actual time=0.117..0.118 rows=625 loops=1)
Buckets: 1024 Batches: 1 Memory Usage: 45kB
-> Bitmap Heap Scan on section sec (cost=32.84..52.22 rows=625 width=28) (actual time=0.021..0.076 rows=625 loops=1)
Recheck Cond: (semester = 1)
Filter: (year = 2019)
Heap Blocks: exact=5
-> Bitmap Index Scan on idx_hash (cost=0.00..32.69 rows=625 width=0) (actual time=0.016..0.016 rows=625 loops=1)
Index Cond: (semester = 1)
-> Index Only Scan using student_pkey on student s (cost=0.29..0.90 rows=1 width=4) (actual time=0.001..0.001 rows=1 loops=625)
Index Cond: (id = t.student_id)
Heap Fetches: 0
-> Hash (cost=52.22..52.22 rows=625 width=28) (actual time=0.146..0.147 rows=625 loops=1)
Buckets: 1024 Batches: 1 Memory Usage: 45kB
-> Bitmap Heap Scan on section sec2 (cost=32.84..52.22 rows=625 width=28) (actual time=0.035..0.101 rows=625 loops=1)
Recheck Cond: (semester = 2)
Filter: (year = 2019)
Heap Blocks: exact=6
-> Bitmap Index Scan on idx_hash (cost=0.00..32.69 rows=625 width=0) (actual time=0.029..0.029 rows=625 loops=1)
Index Cond: (semester = 2)
Planning Time: 0.550 ms
Execution Time: 2.010 ms



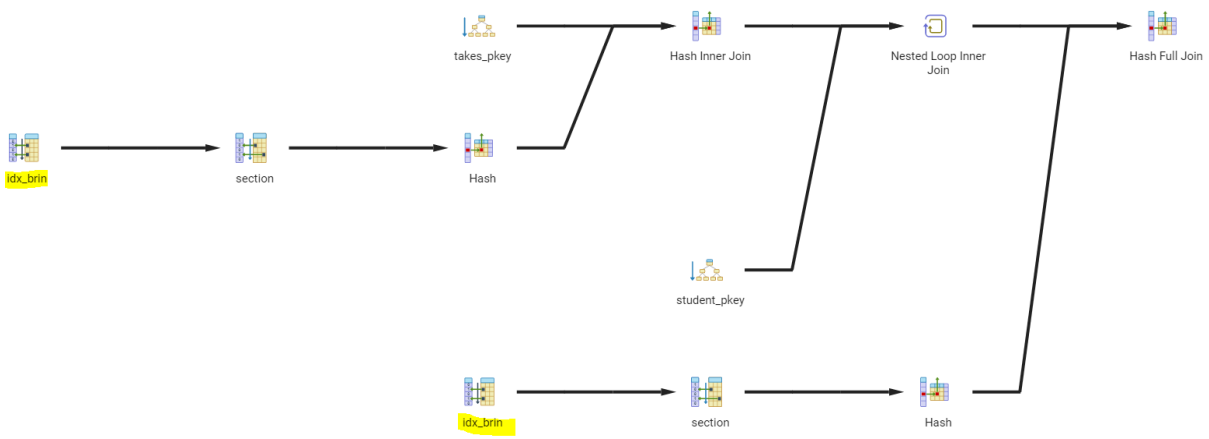
3) Optimized query (1) with brin:

Cost = 722.46 sec

Execution time = 1.541 msec

```
create index idx_brin on section using brin(semester)
```

QUERY PLAN text
Hash Full Join (cost=98.07..722.46 rows=625 width=56) (actual time=0.326..1.387 rows=1250 loops=1)
Hash Cond: (sec.section_id = sec2.section_id)
-> Nested Loop (cost=49.32..672.07 rows=624 width=28) (actual time=0.164..1.066 rows=625 loops=1)
-> Hash Join (cost=49.03..112.93 rows=624 width=32) (actual time=0.157..0.509 rows=625 loops=1)
Hash Cond: (t.section_id = sec.section_id)
-> Index Only Scan using takes_pkey on takes t (cost=0.28..60.89 rows=1249 width=8) (actual time=0.011..0.217 rows=1249 loops=1)
Heap Fetches: 1249
-> Hash (cost=40.94..40.94 rows=625 width=28) (actual time=0.141..0.141 rows=625 loops=1)
Buckets: 1024 Batches: 1 Memory Usage: 45kB
-> Bitmap Heap Scan on section sec (cost=12.19..40.94 rows=625 width=28) (actual time=0.012..0.100 rows=625 loops=1)
Recheck Cond: (semester = 1)
Rows Removed by Index Recheck: 625
Filter: (year = 2019)
Heap Blocks: lossy=10
-> Bitmap Index Scan on idx_brin (cost=0.00..12.03 rows=1250 width=0) (actual time=0.010..0.010 rows=100 loops=1)
Index Cond: (semester = 1)
-> Index Only Scan using student_pkey on student s (cost=0.29..0.90 rows=1 width=4) (actual time=0.001..0.001 rows=1 loops=625)
Index Cond: (id = t.student_id)
Heap Fetches: 0
-> Hash (cost=40.94..40.94 rows=625 width=28) (actual time=0.156..0.156 rows=625 loops=1)
Buckets: 1024 Batches: 1 Memory Usage: 45kB
-> Bitmap Heap Scan on section sec2 (cost=12.19..40.94 rows=625 width=28) (actual time=0.058..0.114 rows=625 loops=1)
Recheck Cond: (semester = 2)
Rows Removed by Index Recheck: 625
Filter: (year = 2019)
Heap Blocks: lossy=10
-> Bitmap Index Scan on idx_brin (cost=0.00..12.03 rows=1250 width=0) (actual time=0.011..0.011 rows=100 loops=1)
Index Cond: (semester = 2)
Planning Time: 0.522 ms
Execution Time: 1.541 ms

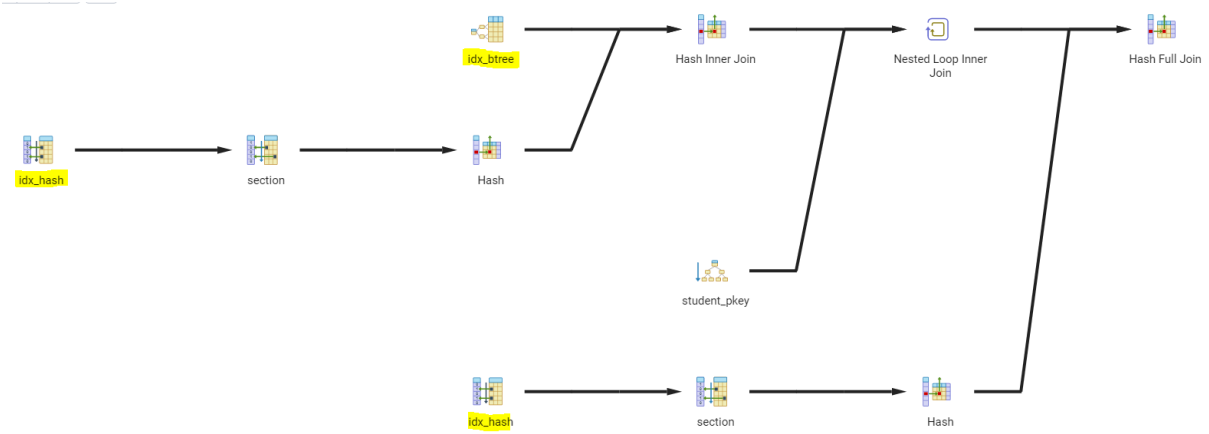


4) Optimized query (1) with mixed indices
Cost = 737.15 rows
Execution time = 1.487 msec

```
create index idx_btree on takes using btree(section_id)

create index idx_hash on section using hash(semester)
```

QUERY PLAN text
Hash Full Join (cost=120.63..737.15 rows=625 width=56) (actual time=0.284..1.347 rows=1250 loops=1)
Hash Cond: (sec.section_id = sec2.section_id)
-> Nested Loop (cost=60.60..675.48 rows=624 width=28) (actual time=0.148..1.035 rows=625 loops=1)
-> Hash Join (cost=60.31..116.33 rows=624 width=32) (actual time=0.139..0.435 rows=625 loops=1)
Hash Cond: (t.section_id = sec.section_id)
-> Index Scan using idx_btree on takes t (cost=0.28..53.01 rows=1249 width=8) (actual time=0.019..0.154 rows=1249 loops=1)
-> Hash (cost=52.22..52.22 rows=625 width=28) (actual time=0.114..0.114 rows=625 loops=1)
Buckets: 1024 Batches: 1 Memory Usage: 45kB
-> Bitmap Heap Scan on section sec (cost=32.84..52.22 rows=625 width=28) (actual time=0.017..0.075 rows=625 loops=1)
Recheck Cond: (semester = 1)
Filter: (year = 2019)
Heap Blocks: exact=5
-> Bitmap Index Scan on idx_hash (cost=0.00..32.69 rows=625 width=0) (actual time=0.012..0.013 rows=625 loops=1)
Index Cond: (semester = 1)
-> Index Only Scan using student_pkey on student s (cost=0.29..0.90 rows=1 width=4) (actual time=0.001..0.001 rows=1 loops=625)
Index Cond: (id = t.student_id)
Heap Fetches: 0
-> Hash (cost=52.22..52.22 rows=625 width=28) (actual time=0.130..0.130 rows=625 loops=1)
Buckets: 1024 Batches: 1 Memory Usage: 45kB
-> Bitmap Heap Scan on section sec2 (cost=32.84..52.22 rows=625 width=28) (actual time=0.025..0.085 rows=625 loops=1)
Recheck Cond: (semester = 2)
Filter: (year = 2019)
Heap Blocks: exact=6
-> Bitmap Index Scan on idx_hash (cost=0.00..32.69 rows=625 width=0) (actual time=0.020..0.020 rows=625 loops=1)
Index Cond: (semester = 2)
Planning Time: 0.705 ms
Execution Time: 1.487 ms



Schema 2

Query (2) before optimization :

```
set enable_async_append = on;
set enable_bitmapscan= on;
set enable_gathermerge = on;
set enable_hashagg = on;
set enable_hashjoin = on;
set enable_incremental_sort = On;
set enable_indexscan= on;
set enable_indexonlyscan = off;
set enable_material= on;
set enable_memoize= on;
set enable_mergejoin= on;
set enable_nestloop= on;
set enable_parallel_append= off;
set enable_parallel_hash= off;
set enable_partition_pruning= on;
set enable_partitionwise_join= off;
set enable_partitionwise_aggregate= off;
set enable_seqscan= on;
set enable_sort= off;
set enable_tidscan = on;
```

```
22 Explain Analyze
23 select distinct pnumber
24 from project
25 where pnumber in
26 (select pnumber
27 from project p, department d, employee e
28 where e.dno=d.dnumber and p.dnumber=d.dnumber
29 and
30 d.mgr_snn=ssn
31 and
32 e.lname='Employee1' )
33 or
34 pnumber in
35 (select pno
36 from works_on, employee
37 where essn=ssn and lname='Employee1' );
38
```

Messages | Data Output

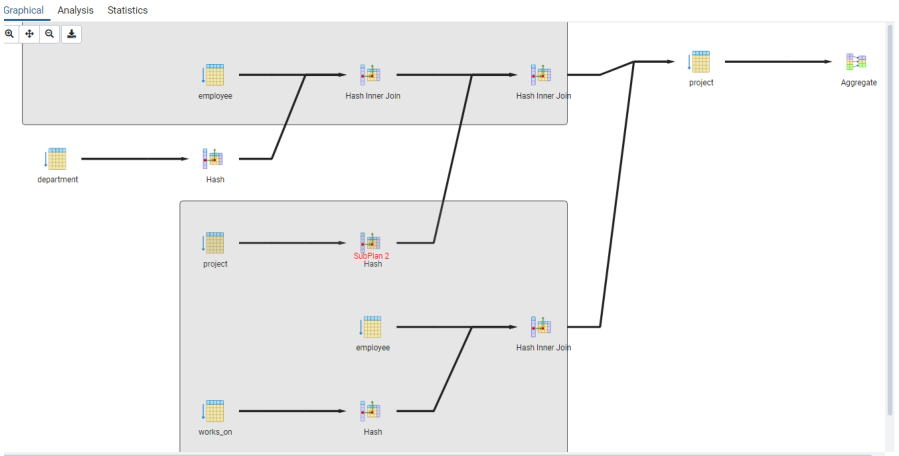
	QUERY PLAN text
1	HashAggregate (cost=2064.31..2133.31 rows=6900 width=4) (actual time=23.344..24.588 rows=9200 loops=1)
2	[...] Group Key: project.pnumber
3	[...] Batches: 1 Memory Usage: 913kB
4	[...] -> Seq Scan on project (cost=1777.06..2047.06 rows=6900 width=4) (actual time=16.882..20.378 rows=9200 loops=1)
5	[...] Filter: ((hashed SubPlan 1) OR (hashed SubPlan 2))
6	[...] SubPlan 1
7	[...] -> Hash Join (cost=344.75..891.59 rows=31 width=4) (actual time=4.510..13.233 rows=9200 loops=1)
8	[...] Hash Cond: (e.dno = p.dnumber)
9	[...] -> Hash Join (cost=5.75..551.98 rows=1 width=8) (actual time=0.111..7.013 rows=150 loops=1)
10	[...] Hash Cond: ((e.dno = d.dnumber) AND (e.ssn = d.mgr_snn))

Query plan of Query (2) before optimization :
Cost = 2133.31 rows
Execution time = 16.532 msec

Messages | Data Output

	QUERY PLAN text
1	HashAggregate (cost=2064.31..2133.31 rows=6900 width=4) (actual time=14.761..15.897 rows=9200 loops=1)
2	[...] Group Key: project.pnumber
3	[...] Batches: 1 Memory Usage: 913kB
4	[...] -> Seq Scan on project (cost=1777.06..2047.06 rows=6900 width=4) (actual time=10.150..12.666 rows=9200 loops=1)
5	[...] Filter: ((hashed SubPlan 1) OR (hashed SubPlan 2))
6	[...] SubPlan 1
7	[...] -> Hash Join (cost=344.75..891.59 rows=31 width=4) (actual time=2.173..7.426 rows=9200 loops=1)
8	[...] Hash Cond: (e.dno = p.dnumber)
9	[...] -> Hash Join (cost=5.75..551.98 rows=1 width=8) (actual time=0.059..3.983 rows=150 loops=1)
10	[...] Hash Cond: ((e.dno = d.dnumber) AND (e.ssn = d.mgr_snn))
11	[...] -> Seq Scan on employee e (cost=0.00..463.00 rows=15854 width=8) (actual time=0.006..2.107 rows=15854 loops=1)
12	[...] Filter: (lname = 'Employee1'::bpchar)
13	[...] Rows Removed by Filter: 146
14	[...] -> Hash (cost=3.50..3.50 rows=150 width=8) (actual time=0.045..0.046 rows=150 loops=1)
15	[...] Buckets: 1024 Batches: 1 Memory Usage: 14kB
16	[...] -> Seq Scan on department d (cost=0.00..3.50 rows=150 width=8) (actual time=0.009..0.023 rows=150 loops=1)
17	[...] -> Hash (cost=224.00..224.00 rows=9200 width=8) (actual time=2.059..2.062 rows=9200 loops=1)
18	[...] Buckets: 16384 Batches: 1 Memory Usage: 488kB
19	[...] -> Seq Scan on project p (cost=0.00..224.00 rows=9200 width=8) (actual time=0.006..0.994 rows=9200 loops=1)
20	[...] SubPlan 2
21	[...] -> Hash Join (cost=251.48..863.10 rows=8917 width=4) (never executed)
22	[...] Hash Cond: (employee.ssn = works_on.essn)
23	[...] -> Seq Scan on employee (cost=0.00..463.00 rows=15854 width=4) (never executed)
24	[...] Filter: (lname = 'Employee1'::bpchar)
25	[...] -> Hash (cost=138.99..138.99 rows=8999 width=8) (never executed)
26	[...] -> Seq Scan on works_on (cost=0.00..138.99 rows=8999 width=8) (never executed)
27	Planning Time: 0.338 ms
28	Execution Time: 16.523 ms

Physical plan of Query (2) before optimization :



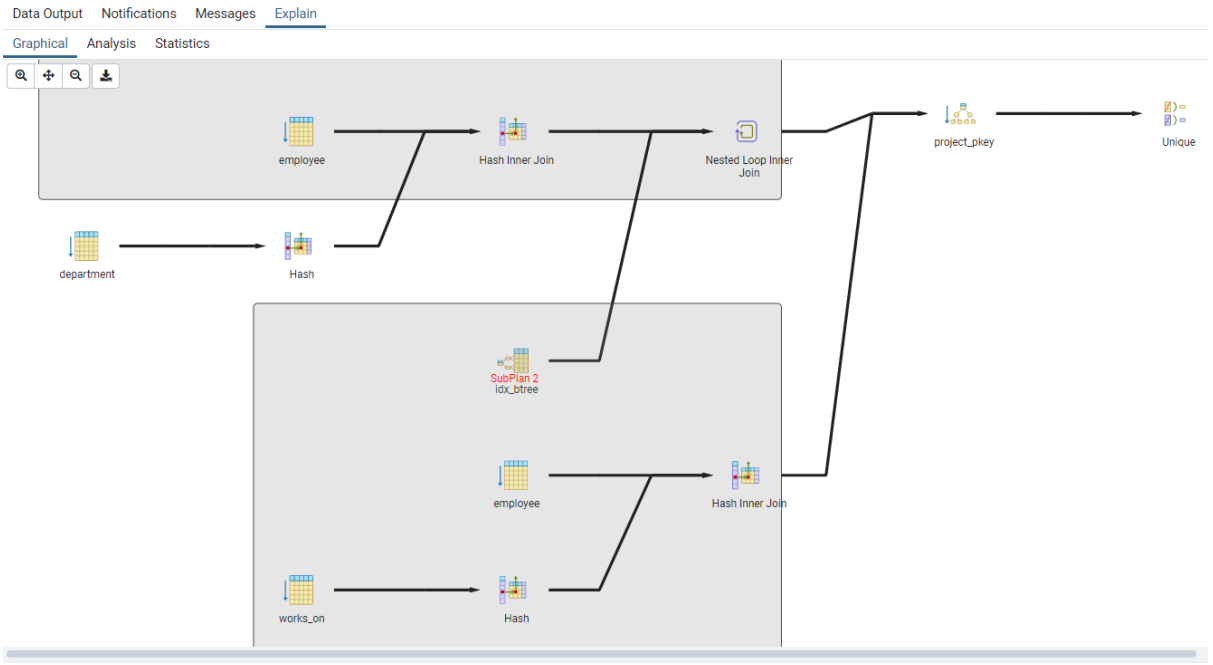
Query plan of Query (2) after using Btree index on table “project” using (dnumber):
Cost = 1752.99 rows

```
24  create index idx_btree on project using btree (dnumber);
25
26  Explain Analyze
27  select distinct pnumber
28  from project
```

Data Output | Notifications | Messages | Explain

	QUERY PLAN text
1	Unique (cost=1439.74..1752.99 rows=6900 width=4) (actual time=12.351..15.737 rows=9200 loops=1)
2	[...] -> Index Only Scan using project_pkey on project (cost=1439.74..1735.74 rows=6900 width=4) (actual time=12.349..1...
3	[...] Filter: ((hashed SubPlan 1) OR (hashed SubPlan 2))
4	[...] Heap Fetches: 0
5	[...] SubPlan 1
6	[...] -> Nested Loop (cost=6.04..553.98 rows=31 width=4) (actual time=0.135..9.470 rows=9200 loops=1)
7	[...] -> Hash Join (cost=5.75..551.98 rows=1 width=8) (actual time=0.096..4.397 rows=150 loops=1)
8	[...] Hash Cond: ((e.dno = d.dnumber) AND (e.ssn = d.mgr_snn))
9	[...] -> Seq Scan on employee e (cost=0.00..463.00 rows=15854 width=8) (actual time=0.010..2.263 rows=15854 loops=1)
10	[...] Filter: (lname = 'Employee1'::bpchar)
11	[...] Rows Removed by Filter: 146
12	[...] -> Hash (cost=3.50..3.50 rows=150 width=8) (actual time=0.071..0.071 rows=150 loops=1)
13	[...] Buckets: 1024 Batches: 1 Memory Usage: 14kB
14	[...] -> Seq Scan on department d (cost=0.00..3.50 rows=150 width=8) (actual time=0.012..0.035 rows=150 loops=1)
15	[...] -> Index Scan using idx_btree on project p (cost=0.29..1.39 rows=61 width=8) (actual time=0.002..0.027 rows=61 loop...
16	[...] Index Cond: (dnumber = e.dno)
17	[...] SubPlan 2
18	[...] -> Hash Join (cost=251.48..863.10 rows=8917 width=4) (never executed)

Physical plan of Query (2) after using Btree index on table “project” using (dnumber):



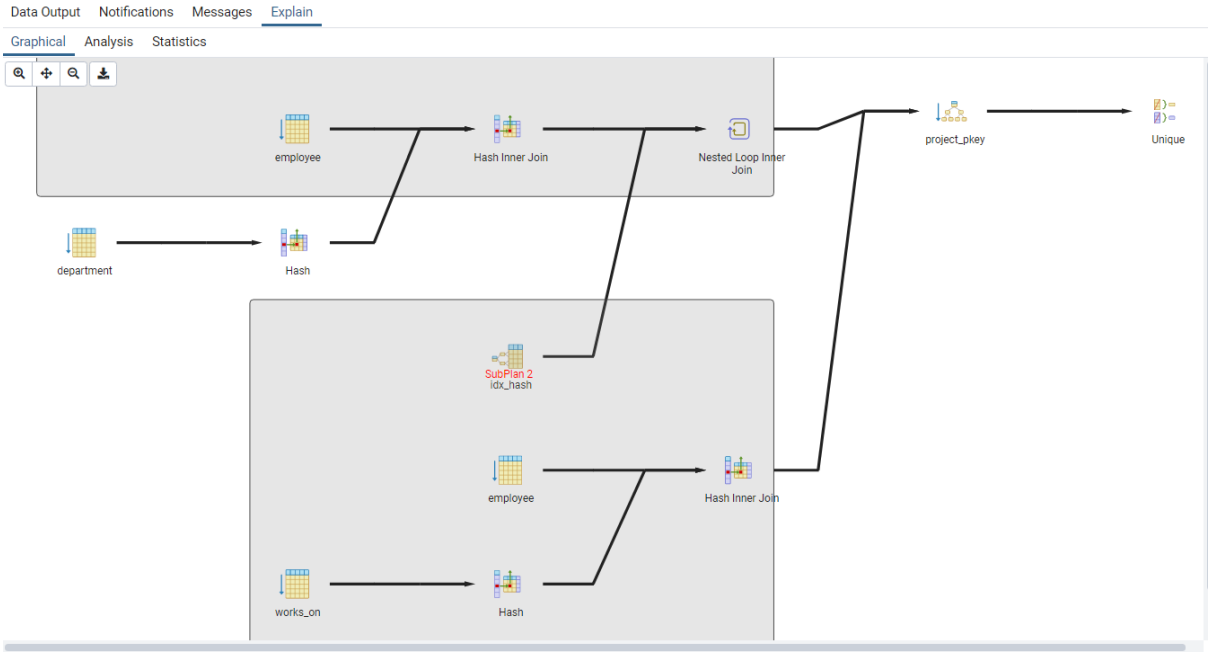
Query plan of Query (2) after using hash index on table “project” using (dnumber):
Cost = 1752.71 rows

```
23
24  create index idx_hash on project using hash (dnumber);
25
```

Data Output | Notifications | Messages | Explain

	QUERY PLAN text
1	Unique (cost=1439.46..1752.71 rows=6900 width=4) (actual time=20.862..26.497 rows=9200 loops=1)
2	[...] -> Index Only Scan using project_pkey on project (cost=1439.46..1735.46 rows=6900 width=4) (actual time=20.859..2...
3	[...] Filter: ((hashed SubPlan 1) OR (hashed SubPlan 2))
4	[...] Heap Fetches: 0
5	[...] SubPlan 1
6	[...] -> Nested Loop (cost=5.75..553.70 rows=31 width=4) (actual time=0.214..16.209 rows=9200 loops=1)
7	[...] -> Hash Join (cost=5.75..551.98 rows=1 width=8) (actual time=0.121..6.219 rows=150 loops=1)
8	[...] Hash Cond: ((e.dno = d.dnumber) AND (e.ssn = d.mgr_snn))
9	[...] -> Seq Scan on employee e (cost=0.00..463.00 rows=15854 width=8) (actual time=0.013..3.196 rows=15854 loops=1)
10	[...] Filter: (lname = 'Employee1'::bpchar)
11	[...] Rows Removed by Filter: 146
12	[...] -> Hash (cost=3.50..3.50 rows=150 width=8) (actual time=0.088..0.089 rows=150 loops=1)
13	[...] Buckets: 1024 Batches: 1 Memory Usage: 14kB
14	[...] -> Seq Scan on department d (cost=0.00..3.50 rows=150 width=8) (actual time=0.014..0.043 rows=150 loops=1)
15	[...] -> Index Scan using idx_hash on project p (cost=0.00..1.11 rows=61 width=8) (actual time=0.004..0.059 rows=61 loop...

Physical plan of Query (2) after using hash index on table “project” using (dnumber):



Query plan of Query (2) after using mixed indices:
Using btree index on table “project” using (pnumber)
Using hash index on table “project” using (dnumber)
Cost = 1752.71 rows
Execution time = 23.377 msec

```
23 create index idx_btree on project using btree (pnumber);
24 create index idx_hash on project using hash (dnumber);
25
26 Explain Analyze
27 select distinct pnumber
28 from project
29 where pnumber in
30 (select pnumber
```

Data Output | Notifications | Messages | Explain

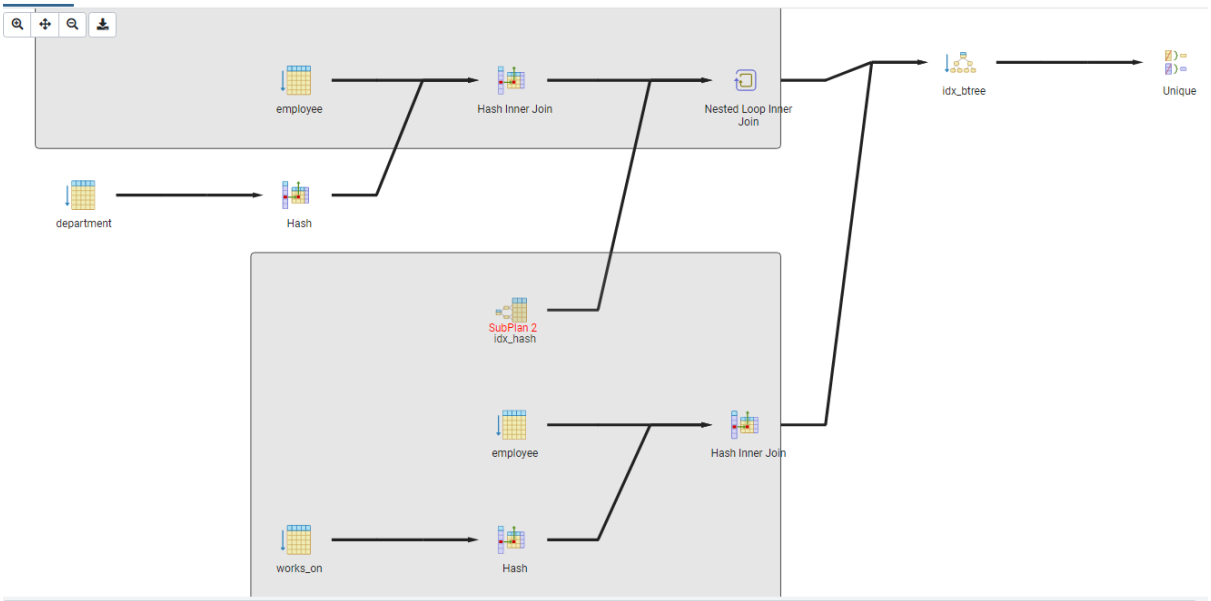
	QUERY PLAN text
1	Unique (cost=1439.46..1752.71 rows=6900 width=4) (actual time=17.731..22.850 rows=9200 loops=1)
2	[...] -> Index Only Scan using idx_btree on project (cost=1439.46..1735.46 rows=6900 width=4) (actual time=17.729..21.316 rows=9200 loops=1)
3	[...] Filter: ((hashed SubPlan 1) OR (hashed SubPlan 2))
4	[...] Heap Fetches: 0
5	[...] SubPlan 1
6	[...] -> Nested Loop (cost=5.75..553.70 rows=31 width=4) (actual time=0.149..13.372 rows=9200 loops=1)
7	[...] -> Hash Join (cost=5.75..551.98 rows=1 width=8) (actual time=0.138..5.369 rows=150 loops=1)
8	[...] Hash Cond: ((e.dno = d.dnumber) AND (e.ssn = d.mgr_snn))
9	[...] -> Seq Scan on employee e (cost=0.00..463.00 rows=15854 width=8) (actual time=0.013..2.701 rows=15854 loops=1)
10	[...] Filter: (lname = 'Employee1'::bpchar)
11	[...] Rows Removed by Filter: 146
12	[...] -> Hash (cost=3.50..3.50 rows=150 width=8) (actual time=0.100..0.100 rows=150 loops=1)
13	[...] Buckets: 1024 Batches: 1 Memory Usage: 14kB
14	[...] -> Seq Scan on department d (cost=0.00..3.50 rows=150 width=8) (actual time=0.014..0.042 rows=150 loops=1)
15	[...] -> Index Scan using idx_hash on project p (cost=0.00..1.11 rows=61 width=8) (actual time=0.003..0.046 rows=61 loops=150)
16	[...] Index Cond: (dnumber = e.dno)
17	[...] SubPlan 2
18	[...] -> Hash Join (cost=251.48..863.10 rows=8917 width=4) (never executed)
19	[...] Hash Cond: (employee.ssn = works_on.essn)
20	[...] -> Seq Scan on employee (cost=0.00..463.00 rows=15854 width=4) (never executed)

17	[...] SubPlan 2
18	[...] -> Hash Join (cost=251.48..863.10 rows=8917 width=4) (never executed)
19	[...] Hash Cond: (employee.ssn = works_on.essn)
20	[...] -> Seq Scan on employee (cost=0.00..463.00 rows=15854 width=4) (never executed)
21	[...] Filter: (lname = 'Employee1'::bpchar)
22	[...] -> Hash (cost=138.99..138.99 rows=8999 width=8) (never executed)
23	[...] -> Seq Scan on works_on (cost=0.00..138.99 rows=8999 width=8) (never executed)
24	Planning Time: 1.383 ms
25	Execution Time: 23.377 ms

Physical plan of Query (2) after using mixed indices:
Using btree index on table “project” using (pnumber)
Using hash index on table “project” using (dnumber)

Data Output | Notifications | Messages | Explain

Graphical | Analysis | Statistics



Query plan of Query (2) before optimization using brin index on table “ project” using (dnumber) :
Cost = 3758.87 rows

Execution time = 137.086 msec
Using Brin index increases cost so using brin index was not efficient as Brin index is only efficient with queries that have exact values .

```
22  create index idx_brin on project using brin (dnumber);
23  Explain Analyze
24  select distinct pnumber
25  from project
26  where pnumber in
27  (select pnumber
28  from project p , department d, employee e
29  where e.dno=d.dnumber and p.dnumber= d.dnumber
30  and
31  d.mgr_snn=ssn
32  and
33  e.lname='Employee1' )
34  or
35  pnumber in
36  (select pno
37  from works_on, employee
38  where essn=ssn and lname='Employee1' );
```

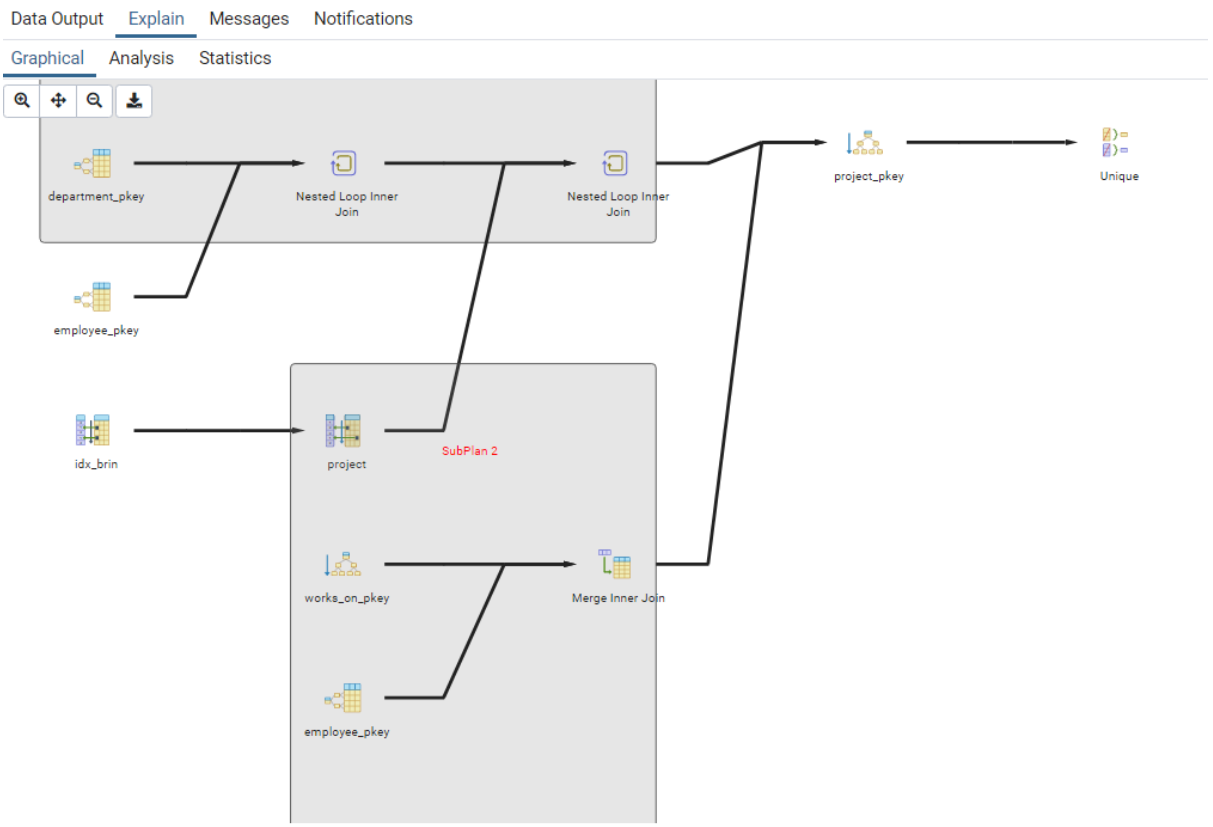
Data Output | Explain | Messages | Notifications

	QUERY PLAN text
1	Unique (cost=3445.62..3758.87 rows=6900 width=4) (actual time=134.218..136.748 rows=9200 loops=1)
2	[...] -> Index Only Scan using project_pkey on project (cost=3445.62..3741.62 rows=6900 width=4) (actual time=134.218..135.833 rows=9200 loops=1)
3	[...] Filter: ((hashed SubPlan 1) OR (hashed SubPlan 2))
4	[...] Heap Fetches: 0
5	[...] SubPlan 1
6	[...] -> Nested Loop (cost=1800.51..2634.86 rows=31 width=4) (actual time=0.032..131.602 rows=9200 loops=1)
7	[...] Join Filter: (e.dno = p.dnumber)
8	[...] -> Nested Loop (cost=0.43..715.02 rows=1 width=8) (actual time=0.013..0.625 rows=150 loops=1)
9	[...] -> Index Scan using department_pkey on department d (cost=0.14..15.39 rows=150 width=8) (actual time=0.005..0.082 rows=150 loops=1)
10	[...] -> Index Scan using employee_pkey on employee e (cost=0.29..4.65 rows=1 width=8) (actual time=0.003..0.003 rows=1 loops=150)
11	[...] Index Cond: (ssn = d.mgr_snn)
12	[...] Filter: ((lname = 'Employee1'::bpchar) AND (d.dnumber = dno))
13	[...] -> Bitmap Heap Scan on project p (cost=1800.08..1919.08 rows=61 width=8) (actual time=0.018..0.862 rows=61 loops=150)
14	[...] Recheck Cond: (dnumber = d.dnumber)

Data Output | Explain | Messages | Notifications

	QUERY PLAN text
14	[...] Recheck Cond: (dnumber = d.dnumber)
15	[...] Rows Removed by Index Recheck: 9139
16	[...] Heap Blocks: lossy=19800
17	[...] -> Bitmap Index Scan on idx_brin (cost=0.00..1800.06 rows=9200 width=0) (actual time=0.010..0.010 rows=1320 loops=150)
18	[...] Index Cond: (dnumber = d.dnumber)
19	[...] SubPlan 2
20	[...] -> Merge Join (cost=0.57..788.10 rows=8917 width=4) (never executed)
21	[...] Merge Cond: (works_on.essn = employee.ssn)
22	[...] -> Index Only Scan using works_on_pkey on works_on (cost=0.29..243.27 rows=8999 width=8) (never executed)
23	[...] Heap Fetches: 0
24	[...] -> Index Scan using employee_pkey on employee (cost=0.29..730.28 rows=15854 width=4) (never executed)
25	[...] Filter: (lname = 'Employee1'::bpchar)
26	Planning Time: 0.746 ms
27	Execution Time: 137.086 ms

Physical plan of Query (2) before optimization using brin index on table “ project” using (dnumber) :

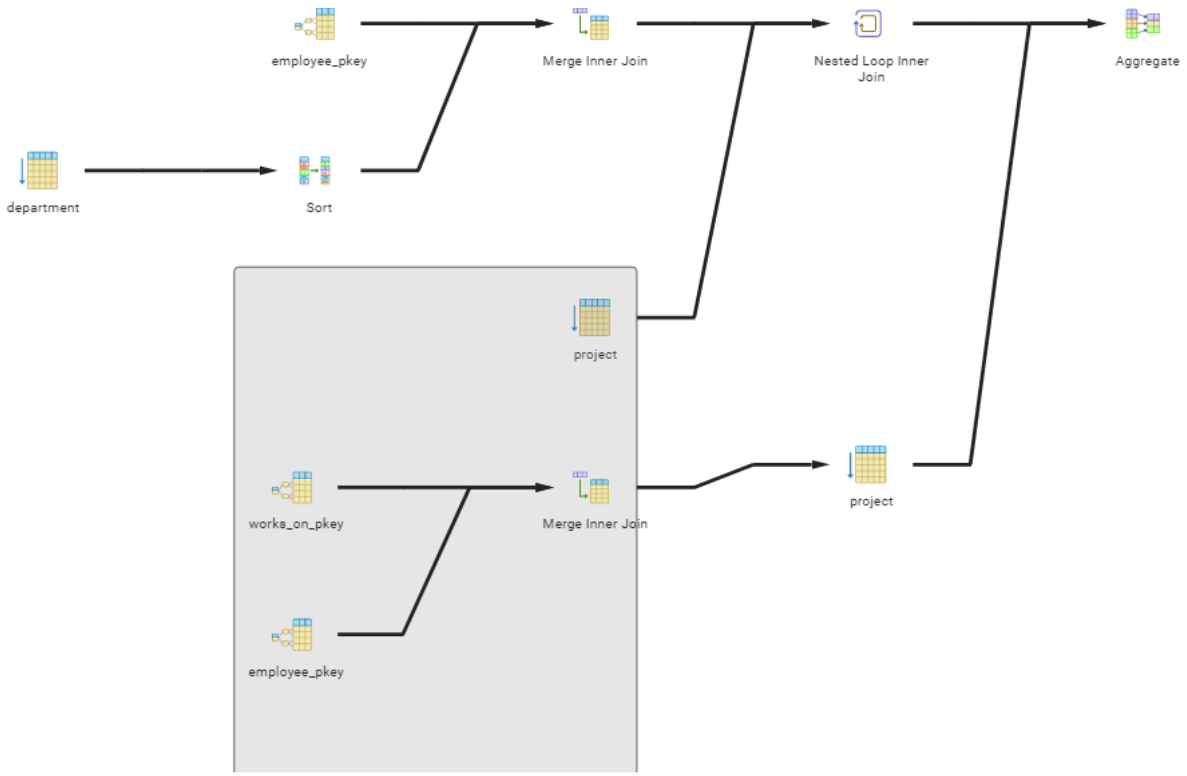


Query (2) after optimization:
Cost = 1275.43 rows
Execution time = 5.333 msec

```
set enable_async_append = on;
set enable_bitmapscan= on;
set enable_gathermerge = on;
set enable_hashagg = on;
set enable_hashjoin = on;
set enable_incremental_sort = On;
set enable_indexscan= on;
set enable_indexonlyscan = off;
set enable_material= on;
set enable_memoize= on;
set enable_mergejoin= on;
set enable_nestloop= on;
set enable_parallel_append= off;
set enable_parallel_hash= off;
set enable_partition_pruning= on;
set enable_partitionwise_join= off;
set enable_partitionwise_aggregate= off;
set enable_seqscan= on;
set enable_sort= on;
set enable_tidscan = on;
```

```
Explain Analyze
(select distinct pnumber
from project p
where exists
(select pnumber
from project p
 natural join department d
 inner join employee e on e.dno=d.dnumber
 and p.dnumber =  d.dnumber
 and d.mgr_snn=ssn
 where e.lname='Employee1')
or
pnumber in (select pno from works_on
            inner join (select ssn from employee
                        where lname='Employee1')
            as f on essn=f.ssn));
```

QUERY PLAN text
HashAggregate (cost=1206.43..1275.43 rows=6900 width=4) (actual time=3.303..4.579 rows=9200 loops=1)
Group Key: p.pnumber
Batches: 1 Memory Usage: 913kB
InitPlan 1 (returns $0)
-> Nested Loop (cost=9.21..358.05 rows=46 width=0) (actual time=0.053..0.055 rows=1 loops=1)
Join Filter: (d.dnumber = p_1.dnumber)
-> Merge Join (cost=9.21..19.05 rows=1 width=8) (actual time=0.048..0.049 rows=1 loops=1)
Merge Cond: (e.ssn = d.mgr_snn)
Join Filter: (d.dnumber = e.dno)
-> Index Scan using employee_pkey on employee e (cost=0.29..730.28 rows=15854 width=8) (actual time=0.007..0.008 rows=1 loops=1)
Filter: (lname = 'Employee1'::bpchar)
-> Sort (cost=8.92..9.30 rows=150 width=8) (actual time=0.036..0.037 rows=1 loops=1)
Sort Key: d.mgr_snn
Sort Method: quicksort Memory: 32kB
-> Seq Scan on department d (cost=0.00..3.50 rows=150 width=8) (actual time=0.008..0.019 rows=150 loops=1)
-> Seq Scan on project p_1 (cost=0.00..224.00 rows=9200 width=4) (actual time=0.004..0.004 rows=1 loops=1)
-> Seq Scan on project p (cost=925.39..1172.39 rows=6900 width=4) (actual time=0.092..1.201 rows=9200 loops=1)
Filter: ($0 OR (hashed SubPlan 2))
SubPlan 2
-> Merge Join (cost=0.57..903.10 rows=8917 width=4) (never executed)
Merge Cond: (works_on.essn = employee.ssn)
-> Index Scan using works_on_pkey on works_on (cost=0.29..358.27 rows=8999 width=8) (never executed)
-> Index Scan using employee_pkey on employee (cost=0.29..730.28 rows=15854 width=4) (never executed)
Filter: (lname = 'Employee1'::bpchar)
Planning Time: 0.551 ms
Execution Time: 5.333 ms

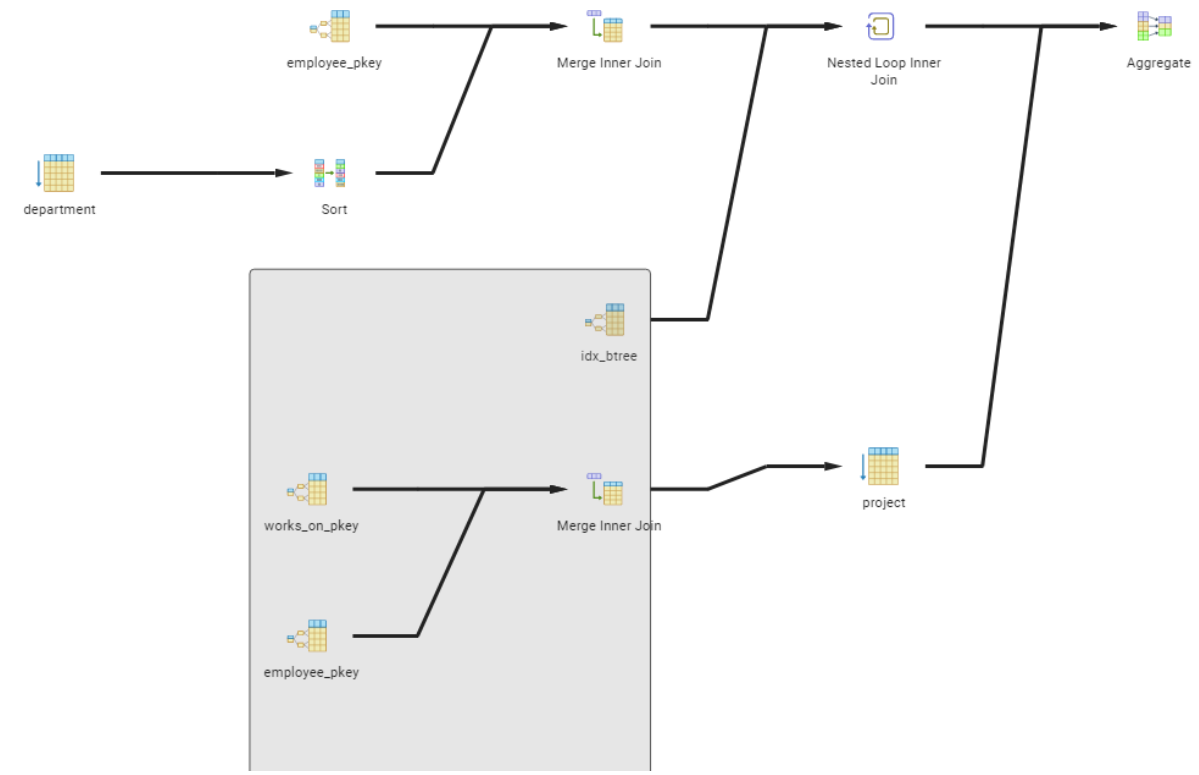


Query(2) optimized with btree index:
Cost = 1268.39 rows
Execution time = 4.502 msec

```
create index idx_btree on project using btree(dnumber);
```

QUERY PLAN text
HashAggregate (cost=1199.39..1268.39 rows=6900 width=4) (actual time=3.004..4.050 rows=9200 loops=1)
Group Key: p.pnumber
Batches: 1 Memory Usage: 913kB
InitPlan 1 (returns $1)
-> Nested Loop (cost=9.49..21.20 rows=46 width=0) (actual time=0.055..0.056 rows=1 loops=1)
Join Filter: (d.dnumber = p_1.dnumber)
-> Merge Join (cost=9.21..19.05 rows=1 width=8) (actual time=0.048..0.049 rows=1 loops=1)
Merge Cond: (e.ssn = d.mgr_snn)
Join Filter: (d.dnumber = e.dno)
-> Index Scan using employee_pkey on employee e (cost=0.29..730.28 rows=15854 width=8) (actual time=0.007..0.007 rows=1 loops=1)
Filter: (lname = 'Employee1'::bpchar)
-> Sort (cost=8.92..9.30 rows=150 width=8) (actual time=0.038..0.039 rows=1 loops=1)
Sort Key: d.mgr_snn
Sort Method: quicksort Memory: 32kB
-> Seq Scan on department d (cost=0.00..3.50 rows=150 width=8) (actual time=0.008..0.020 rows=150 loops=1)
-> Index Scan using idx_btree on project p_1 (cost=0.29..1.39 rows=61 width=4) (actual time=0.005..0.006 rows=1 loops=1)
Index Cond: (dnumber = e.dno)
-> Seq Scan on project p (cost=925.39..1172.39 rows=6900 width=4) (actual time=0.065..1.049 rows=9200 loops=1)
Filter: ($1 OR (hashed SubPlan 2))
SubPlan 2
-> Merge Join (cost=0.57..903.10 rows=8917 width=4) (never executed)

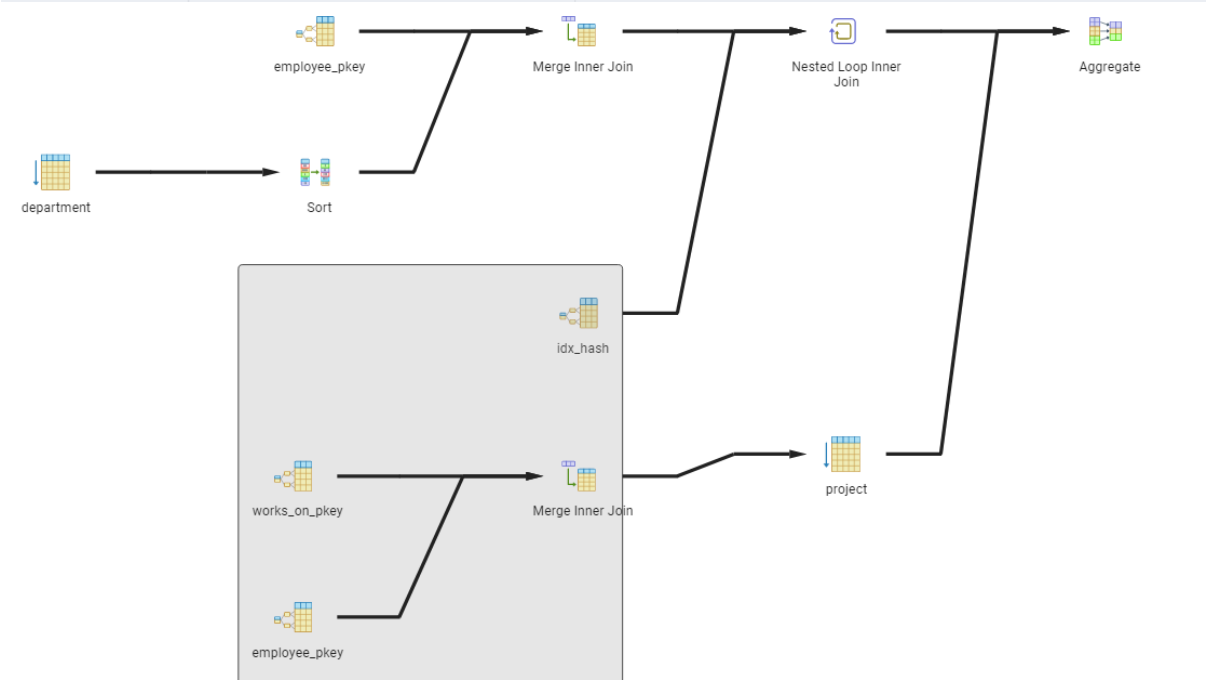
Merge Cond: (works_on.essn = employee.ssn)
-> Index Scan using works_on_pkey on works_on (cost=0.29..358.27 rows=8999 width=8) (never executed)
-> Index Scan using employee_pkey on employee (cost=0.29..730.28 rows=15854 width=4) (never executed)
Filter: (lname = 'Employee1'::bpchar)
Planning Time: 1.364 ms
Execution Time: 4.502 ms



Query(2) optimized with hashindex:
Cost = 1268.10 rows
Execution time = 5.259 msec

```
create index idx_hash on project using hash(dnumber);
```

QUERY PLAN text
HashAggregate (cost=1199.10..1268.10 rows=6900 width=4) (actual time=3.860..4.829 rows=9200 loops=1)
Group Key: p.pnumber
Batches: 1 Memory Usage: 913kB
InitPlan 1 (returns $1)
-> Nested Loop (cost=9.21..20.93 rows=46 width=0) (actual time=0.052..0.054 rows=1 loops=1)
Join Filter: (d.dnumber = p_1.dnumber)
-> Merge Join (cost=9.21..19.05 rows=1 width=8) (actual time=0.042..0.044 rows=1 loops=1)
Merge Cond: (e.ssn = d.mgr_snn)
Join Filter: (d.dnumber = e.dno)
-> Index Scan using employee_pkey on employee e (cost=0.29..730.28 rows=15854 width=8) (actual time=0.005..0.005 rows=1 loops=1)
Filter: (lname = 'Employee1'::bpchar)
-> Sort (cost=8.92..9.30 rows=150 width=8) (actual time=0.035..0.036 rows=1 loops=1)
Sort Key: d.mgr_snn
Sort Method: quicksort Memory: 32kB
-> Seq Scan on department d (cost=0.00..3.50 rows=150 width=8) (actual time=0.008..0.019 rows=150 loops=1)
-> Index Scan using idx_hash on project p_1 (cost=0.00..1.11 rows=61 width=4) (actual time=0.008..0.009 rows=1 loops=1)
Index Cond: (dnumber = e.dno)
-> Seq Scan on project p (cost=925.39..1172.39 rows=6900 width=4) (actual time=0.060..1.298 rows=9200 loops=1)
Filter: ($1 OR (hashed SubPlan 2))
SubPlan 2
-> Merge Join (cost=0.57..903.10 rows=8917 width=4) (never executed)
Merge Cond: (works_on.essn = employee.ssn)
-> Index Scan using works_on_pkey on works_on (cost=0.29..358.27 rows=8999 width=8) (never executed)
-> Index Scan using employee_pkey on employee (cost=0.29..730.28 rows=15854 width=4) (never executed)
Filter: (lname = 'Employee1'::bpchar)
Planning Time: 0.702 ms
Execution Time: 5.259 ms



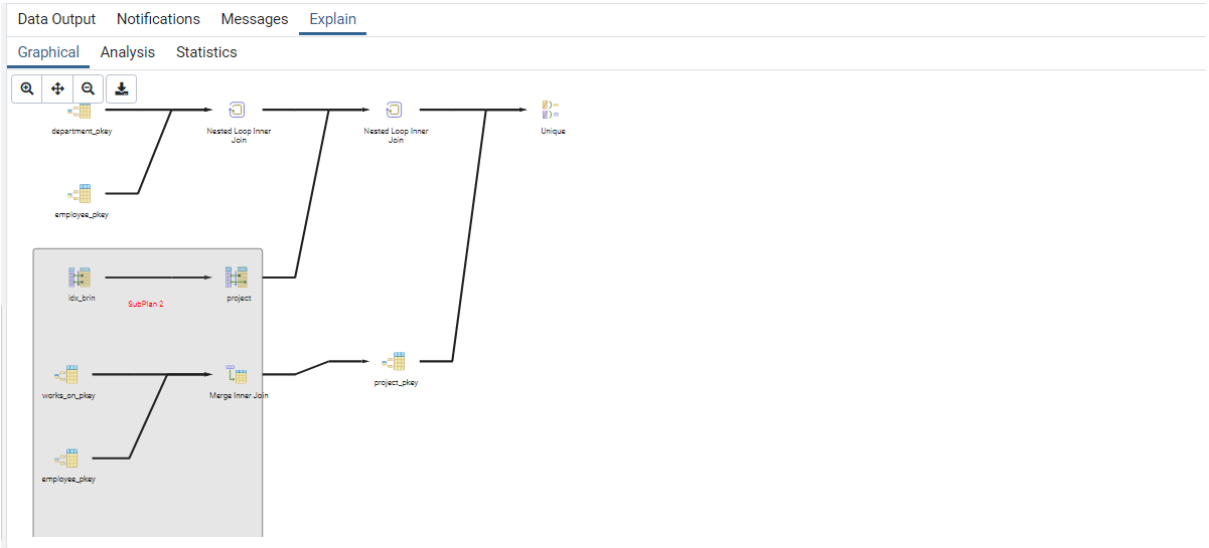
Query(2) optimized with brin index on table “project” using “ number”:
Cost = 3169.57 rows
Execution time = 2.935 msec
Using Brin index increases cost so using brin index was not efficient as Brin index is only efficient with queries that have exact values .

```
24
25 create index idx_brin on project using brin (dnumber);
26 Explain Analyze
27 select distinct pnumber
28 from project p
29 where exists ( select pnumber from project p natural join
30               department d inner join employee e on e.dno=d.dnumber and p.dnumber=d.dnumber
31               and d.mgr_snn = ssn
```

Data Output | Notifications | Messages | Explain

	QUERY PLAN text
1	Unique (cost=2744.32..3169.57 rows=6900 width=4) (actual time=0.092..2.686 rows=9200 loops=1)
2	[...] InitPlan 1 (returns $3)
3	[...] -> Nested Loop (cost=1800.51..2634.71 rows=46 width=0) (actual time=0.076..0.079 rows=1 loops=1)
4	[...] -> Nested Loop (cost=0.43..715.02 rows=1 width=8) (actual time=0.029..0.030 rows=1 loops=1)
5	[...] -> Index Scan using department_pkey on department d (cost=0.14..15.39 rows=150 width=8) (actual time=0.019..0.019 rows=1 l...
6	[...] -> Index Scan using employee_pkey on employee e (cost=0.29..4.65 rows=1 width=8) (actual time=0.008..0.009 rows=1 loops=1)
7	[...] Index Cond: (ssn = d.mgr_snn)
8	[...] Filter: ((lname = 'Employee1'::bpchar) AND (d.dnumber = dno))
9	[...] -> Bitmap Heap Scan on project p_1 (cost=1800.08..1919.08 rows=61 width=4) (actual time=0.043..0.044 rows=1 loops=1)
10	[...] Recheck Cond: (dnumber = d.dnumber)
11	[...] Heap Blocks: lossy=1
12	[...] -> Bitmap Index Scan on idx_brin (cost=0.00..1800.06 rows=9200 width=0) (actual time=0.039..0.039 rows=1320 loops=1)
13	[...] Index Cond: (dnumber = d.dnumber)
14	[...] -> Index Scan using project_pkey on project p (cost=925.68..1333.68 rows=6900 width=4) (actual time=0.091..1.597 rows=9200 ...
15	[...] Filter: ($3 OR (hashed SubPlan 2))
16	[...] SubPlan 2
17	[...] -> Merge Join (cost=0.57..903.10 rows=8917 width=4) (never executed)
18	[...] Merge Cond: (works_on.essn = employee.ssn)
19	[...] -> Index Scan using works_on_pkey on works_on (cost=0.29..358.27 rows=8999 width=8) (never executed)
20	[...] -> Index Scan using employee_pkey on employee (cost=0.29..730.28 rows=15854 width=4) (never executed)
21	[...] Filter: (lname = 'Employee1'::bpchar)
22	Planning Time: 0.505 ms
23	Execution Time: 2.935 ms

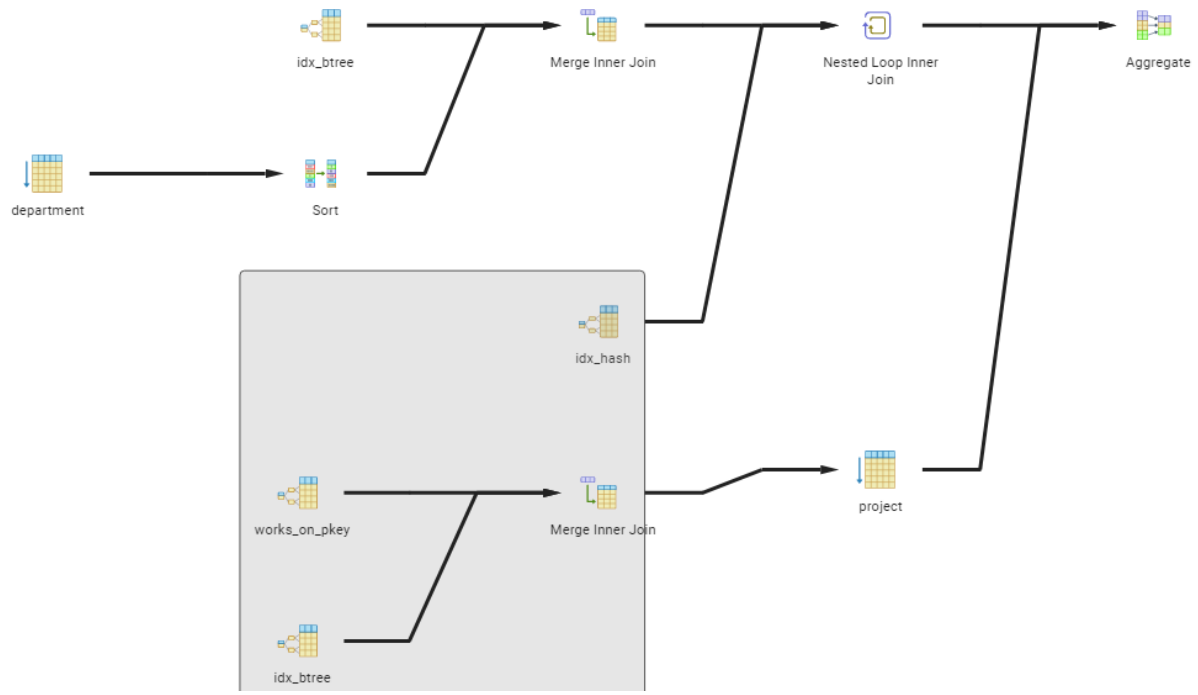
Physical plan of query (2) after optimization using brin index on table “project” using “ number” :



Query(2) optimized with mixed indices:
Cost = 1268.10 rows
Execution time = 4.098 msec

```
create index idx_btree on employee using btree(ssn);
create index idx_hash on project using hash(dnumber);
```

QUERY PLAN text
HashAggregate (cost=1199.10..1268.10 rows=6900 width=4) (actual time=2.694..3.586 rows=9200 loops=1)
Group Key: p.pnumber
Batches: 1 Memory Usage: 913kB
InitPlan 1 (returns $1)
-> Nested Loop (cost=9.21..20.93 rows=46 width=0) (actual time=0.055..0.057 rows=1 loops=1)
Join Filter: (d.dnumber = p_1.dnumber)
-> Merge Join (cost=9.21..19.05 rows=1 width=8) (actual time=0.046..0.047 rows=1 loops=1)
Merge Cond: (e.ssn = d.mgr_snn)
Join Filter: (d.dnumber = e.dno)
-> Index Scan using idx_btree on employee e (cost=0.29..730.28 rows=15854 width=8) (actual time=0.008..0.008 rows=1 loops=1)
Filter: (lname = 'Employee1'::bpchar)
-> Sort (cost=8.92..9.30 rows=150 width=8) (actual time=0.035..0.035 rows=1 loops=1)
Sort Key: d.mgr_snn
Sort Method: quicksort Memory: 32kB
-> Seq Scan on department d (cost=0.00..3.50 rows=150 width=8) (actual time=0.007..0.018 rows=150 loops=1)
-> Index Scan using idx_hash on project p_1 (cost=0.00..1.11 rows=61 width=4) (actual time=0.008..0.008 rows=1 loops=1)
Index Cond: (dnumber = e.dno)
-> Seq Scan on project p (cost=925.39..1172.39 rows=6900 width=4) (actual time=0.065..0.883 rows=9200 loops=1)
Filter: ($1 OR (hashed SubPlan 2))
SubPlan 2
-> Merge Join (cost=0.57..903.10 rows=8917 width=4) (never executed)
Merge Cond: (works_on.essn = employee.ssn)
-> Index Scan using works_on_pkey on works_on (cost=0.29..358.27 rows=8999 width=8) (never executed)
-> Index Scan using idx_btree on employee (cost=0.29..730.28 rows=15854 width=4) (never executed)
Filter: (lname = 'Employee1'::bpchar)
Planning Time: 0.619 ms
Execution Time: 4.098 ms



Query (3) before optimization :

SET enable_async_append= on;
SET enable_bitmapscan= on;
SET enable_gathermerge=off;
SET enable_hashagg=on;
SET enable_hashjoin=off;
SET enable_incremental_sort=on;
SET enable_indexscan=off;
SET enable_indexonlyscan=off;
SET enable_material=off;
SET enable_memoize=on;
SET enable_mergejoin=on;
SET enable_nestloop =off;
SET enable_parallel_append=on;
SET enable_parallel_hash=off;
SET enable_partition_pruning=on;
SET enable_partitionwise_join =off;
SET enable_partitionwise_aggregate =on;
SET enable_sort=on;
SET enable_tidscan=off;

```
23  select lname, fname
24  from employee
25  where salary > all (
26  select salary
27  from employee
28  where dno=5 );
29
30
```

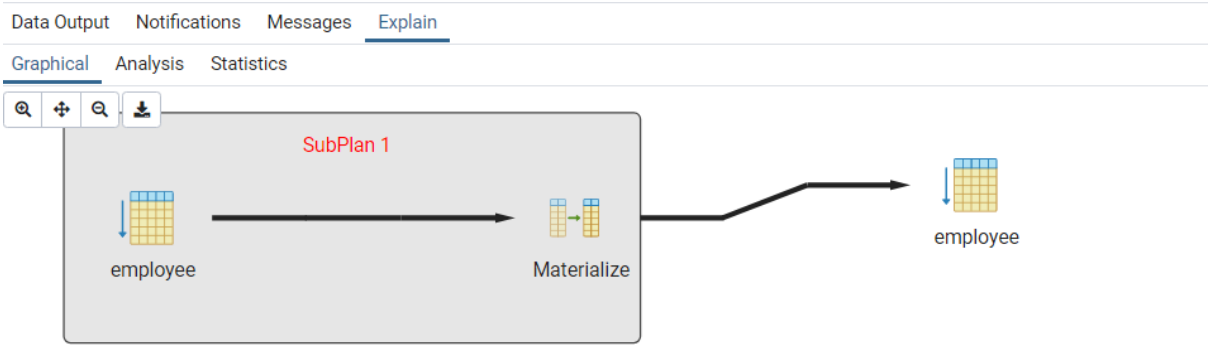
Query plan of Query (3) before optimization :
Cost = 3710763 rows
Execution time = 127.200 msec

```
22  Explain Analyze
23  select lname, fname
24  from employee
25  where salary > all (
26  select salary
27  from employee
28  where dno=5 );
29
```

Data Output | Notifications | Messages | Explain

	QUERY PLAN text
1	Seq Scan on employee (cost=0.00..3710763.00 rows=8000 width=42) (actual time=1.731..126.843 rows=15330 loops=1)
2	[...] Filter: (SubPlan 1)
3	[...] Rows Removed by Filter: 670
4	[...] SubPlan 1
5	[...] -> Materialize (cost=0.00..463.52 rows=105 width=4) (actual time=0.000..0.003 rows=101 loops=16000)
6	[...] -> Seq Scan on employee employee_1 (cost=0.00..463.00 rows=105 width=4) (actual time=0.004..1.655 rows=105 loops=1)
7	[...] Filter: (dno = 5)
8	[...] Rows Removed by Filter: 15895
9	Planning Time: 0.098 ms
10	Execution Time: 127.200 ms

Physical plan of Query (3) before optimization :



Query plan of Query (3) before optimization using after btree index on table “employee” using (dno) :
Cost = 1688963.25 rows
Execution time = 759.698 msec

```
23  Create index idx_btree on employee using btree (dno);
24
25  Explain Analyze
26  select lname, fname
27  from employee
28  where salary > all (
29  select salary
30  from employee
31  where dno=5 );
32
33
```

Data Output | Notifications | Messages | Explain

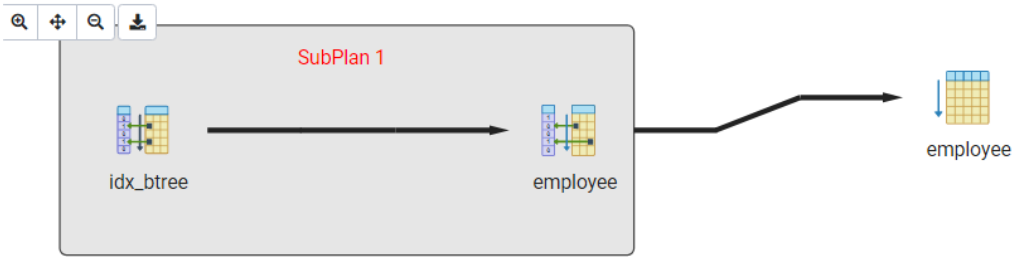
	QUERY PLAN text
1	Seq Scan on employee (cost=0.00..1688963.25 rows=8000 width=42) (actual time=1.848..758.865 rows=15330 loops=1)
2	[...] Filter: (SubPlan 1)
3	[...] Rows Removed by Filter: 670
4	[...] SubPlan 1
5	[...] -> Bitmap Heap Scan on employee employee_1 (cost=5.10..205.70 rows=105 width=4) (actual time=0.010..0.040 rows=101 loops=16000)
6	[...] Recheck Cond: (dno = 5)
7	[...] Heap Blocks: exact=1610985
8	[...] -> Bitmap Index Scan on idx_btree (cost=0.00..5.07 rows=105 width=0) (actual time=0.006..0.006 rows=105 loops=16000)
9	[...] Index Cond: (dno = 5)
10	Planning Time: 0.084 ms
11	Execution Time: 759.698 ms

physical plan of Query (3) before optimization after using btree index on table “employee” using (dno):

```
23  Create index idx_btree on employee using btree (dno);
24
25  Explain Analyze
26  select lname, fname
27  from employee
28  where salary > all (
29  select salary
30  from employee
31  where dno=5 );
32
```

Data Output Notifications Messages Explain

Graphical Analysis Statistics



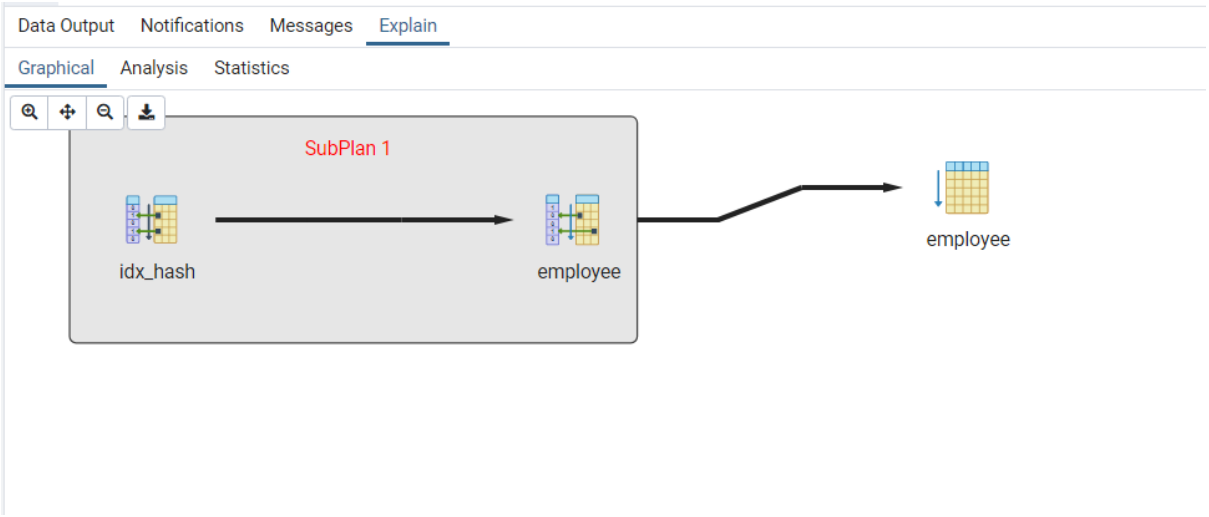
Query plan of Query (3) before optimization after using hash index on table “employee” using (dno) :
Cost = 1688963.25 rows
Execution time = 741.750 msec

```
23
24  Create index idx_hash on employee using btree (dno);
25
26  Explain Analyze
27  select lname, fname
28  from employee
29  where salary > all (
30  select salary
31  from employee
32  where dno=5 );
```

Data Output Notifications Messages Explain

	QUERY PLAN text
1	Seq Scan on employee (cost=0.00..1688963.25 rows=8000 width=42) (actual time=2.087..741.129 rows=15330 loops=1)
2	[...] Filter: (SubPlan 1)
3	[...] Rows Removed by Filter: 670
4	[...] SubPlan 1
5	[...] -> Bitmap Heap Scan on employee employee_1 (cost=5.10..205.70 rows=105 width=4) (actual time=0.009..0.039 rows=101 loops=16000)
6	[...] Recheck Cond: (dno = 5)
7	[...] Heap Blocks: exact=1610985
8	[...] -> Bitmap Index Scan on idx_hash (cost=0.00..5.07 rows=105 width=0) (actual time=0.006..0.006 rows=105 loops=16000)
9	[...] Index Cond: (dno = 5)
10	Planning Time: 0.266 ms
11	Execution Time: 741.750 ms

Physical plan of Query (3) before optimization after using hash index on table “employee” using (dno):



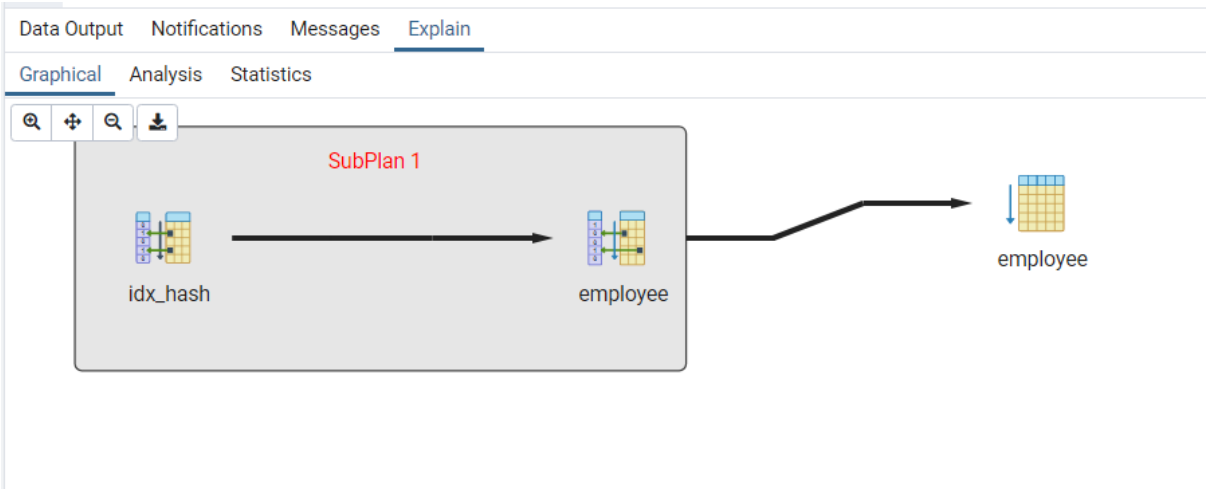
Query plan of Query (3) before optimization after using mixed indices :
Using btree index on table “Employee” using (salary)
Using hash index on table “Employee” using (dno)
Cost = 1684403.25 rows
Execution time = 814.693 msec

```
23   create index idx_btree on employee using btree (salary);
24  create index idx_hash on employee using hash (dno);
25
26
27  Explain Analyze
28  select lname, fname
29  from employee
30  where salary > all (
31  select salary
32  from employee
33  where dno=5 );
34
```

Data Output | Notifications | Messages | Explain

	QUERY PLAN text
1	Seq Scan on employee (cost=0.00..1684403.25 rows=8000 width=42) (actual time=1.990..813.990 rows=15330 loops=1)
2	[...] Filter: (SubPlan 1)
3	[...] Rows Removed by Filter: 670
4	[...] SubPlan 1
5	[...] -> Bitmap Heap Scan on employee employee_1 (cost=4.81..205.42 rows=105 width=4) (actual time=0.010..0.044 rows=101 loops=16000)
6	[...] Recheck Cond: (dno = 5)
7	[...] Heap Blocks: exact=1610985
8	[...] -> Bitmap Index Scan on idx_hash (cost=0.00..4.79 rows=105 width=0) (actual time=0.006..0.006 rows=105 loops=16000)
9	[...] Index Cond: (dno = 5)
10	Planning Time: 0.091 ms
11	Execution Time: 814.693 ms

Physical plan of Query (3) before optimization after using mixed indices :
Using btree index on table “Employee” using (salary)
Using hash index on table “Employee” using (dno)



Query plan of Query (3) before optimization after using brin index on table “employee” using (dno) :
Cost = 10003710775.12 rows
Execution time = 143.580 msec
Using Brin index increases cost to a very big value so using brin index was not efficient as Brin index is only efficient with queries that have exact values .

```
23
24  create index idx_brin on employee using brin (dno);
25
26  Explain Analyze
27  select lname, fname
28  from employee
29  where salary > all (
30  select salary
31  from employee
32  where dno=5 );
33
34
35
```

Data Output | Explain | Messages | Notifications

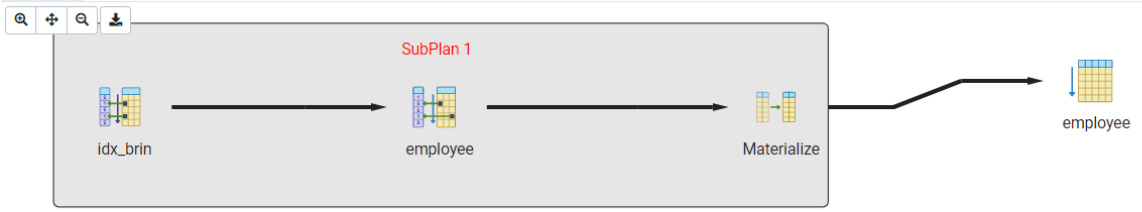
	QUERY PLAN text
1	Seq Scan on employee (cost=10000000012.12..10003710775.12 rows=8000 width=42) (actual time=3.558..143.129 rows=15330 loops=1)
2	[...] Filter: (SubPlan 1)
3	[...] Rows Removed by Filter: 670
4	[...] SubPlan 1
5	[...] -> Materialize (cost=12.12..475.65 rows=105 width=4) (actual time=0.000..0.003 rows=101 loops=16000)
6	[...] -> Bitmap Heap Scan on employee employee_1 (cost=12.12..475.12 rows=105 width=4) (actual time=0.031..3.412 rows=105 loops=1)
7	[...] Recheck Cond: (dno = 5)
8	[...] Rows Removed by Index Recheck: 15895
9	[...] Heap Blocks: lossy=263
10	[...] -> Bitmap Index Scan on idx_brin (cost=0.00..12.10 rows=16000 width=0) (actual time=0.023..0.023 rows=2630 loops=1)
11	[...] Index Cond: (dno = 5)
12	Planning Time: 0.197 ms
13	Execution Time: 143.580 ms

Physical of Query (3) before optimization after using brin index on table “employee” using (dno) :

```
23
24  create index idx_brin on employee using brin (dno);
25
26 Explain Analyze
27 select lname, fname
28 from employee
29 where salary > all (
30 select salary
31 from employee
32 where dno=5 );
33
34
35
```

Data Output | Explain | Messages | Notifications

Graphical | Analysis | Statistics



Query (3) after optimization :
Cost = 926.27 rows
Execution time = 4.106 msec

```
32 --optimized query
33 Explain analyze
34 select lname, fname
35 from employee
36 where salary >  (
37 select max (salary)
38 from employee
39 where dno=5 );
40
```

Query plan of optimized query (3) :

```
44
```

Data Output | Notifications | Messages | Explain

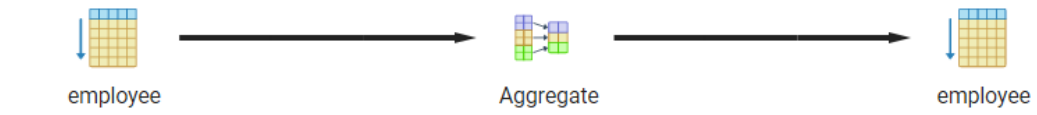
	QUERY PLAN text
1	Seq Scan on employee (cost=463.27..926.27 rows=5333 width=42) (actual time=1.469..3.792 rows=15330 loops=1)
2	[...] Filter: (salary > $0)
3	[...] Rows Removed by Filter: 670
4	[...] InitPlan 1 (returns $0)
5	[...] -> Aggregate (cost=463.26..463.27 rows=1 width=4) (actual time=1.448..1.449 rows=1 loops=1)
6	[...] -> Seq Scan on employee employee_1 (cost=0.00..463.00 rows=105 width=4) (actual time=0.004..1.440 rows=105 loops=1)
7	[...] Filter: (dno = 5)
8	[...] Rows Removed by Filter: 15895
9	Planning Time: 0.129 ms
10	Execution Time: 4.106 ms

Physical plan of optimized query (3) :

```
34  select lname, fname
35  from employee
36  where salary >  (
37  select max (salary)
38  from employee
39  where dno=5 );
40
41
42
43
44
```

Data Output | Notifications | Messages | Explain

Graphical | Analysis | Statistics



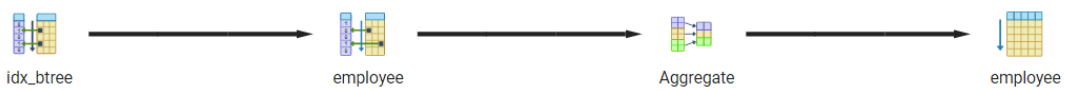
Query plan of optimized query (3) after using btree index on table “employee” using “dno”:
Cost = 668.97 rows
Execution time = 2.262 msec

```
33  --optimized query
34  Explain analyze
35  select lname, fname
36  from employee
37  where salary >  (
38  select max (salary)
39  from employee
40  where dno=5 );
41
42  create index idx_btree on employee using btree (dno);
43
44
45
```

Data Output | Notifications | Messages | Explain

	QUERY PLAN text
1	Seq Scan on employee (cost=205.97..668.97 rows=5333 width=42) (actual time=0.137..1.969 rows=15330 loops=1)
2	[...] Filter: (salary > $0)
3	[...] Rows Removed by Filter: 670
4	[...] InitPlan 1 (returns $0)
5	[...] -> Aggregate (cost=205.96..205.97 rows=1 width=4) (actual time=0.117..0.117 rows=1 loops=1)
6	[...] -> Bitmap Heap Scan on employee employee_1 (cost=5.10..205.70 rows=105 width=4) (actual time=0.031..0.104 rows=105...
7	[...] Recheck Cond: (dno = 5)
8	[...] Heap Blocks: exact=105
9	[...] -> Bitmap Index Scan on idx_btree (cost=0.00..5.07 rows=105 width=0) (actual time=0.021..0.021 rows=105 loops=1)
10	[...] Index Cond: (dno = 5)
11	Planning Time: 0.368 ms
12	Execution Time: 2.262 ms

Physical plan of optimized query (3) after using btree index on table “employee” using “dno”:



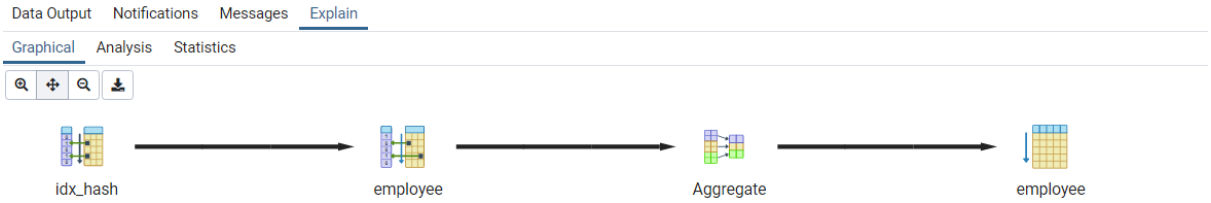
Query plan of optimized query (3) after using hash index on table “employee” using “dno”:
Cost = 668.69 rows
Execution time = 2.245 msec

```
33  --optimized query
34  Explain analyze
35  select lname, fname
36  from employee
37  where salary >  (
38  select max (salary)
39  from employee
40  where dno=5 );
41
42  create index idx_hash on employee using hash (dno);
43
44
45
```

Data Output | Notifications | Messages | Explain

	QUERY PLAN text
1	Seq Scan on employee (cost=205.69..668.69 rows=5333 width=42) (actual time=0.113..1.958 rows=15330 loops=1)
2	[...] Filter: (salary > $0)
3	[...] Rows Removed by Filter: 670
4	[...] InitPlan 1 (returns $0)
5	[...] -> Aggregate (cost=205.68..205.69 rows=1 width=4) (actual time=0.094..0.095 rows=1 loops=1)
6	[...] -> Bitmap Heap Scan on employee employee_1 (cost=4.81..205.42 rows=105 width=4) (actual time=0.024..0.087 rows=105...
7	[...] Recheck Cond: (dno = 5)
8	[...] Heap Blocks: exact=105
9	[...] -> Bitmap Index Scan on idx_hash (cost=0.00..4.79 rows=105 width=0) (actual time=0.013..0.013 rows=105 loops=1)
10	[...] Index Cond: (dno = 5)
11	Planning Time: 0.371 ms
12	Execution Time: 2.245 ms

Physical plan of optimized query (3) after using hash index on table “employee” using “dno”:



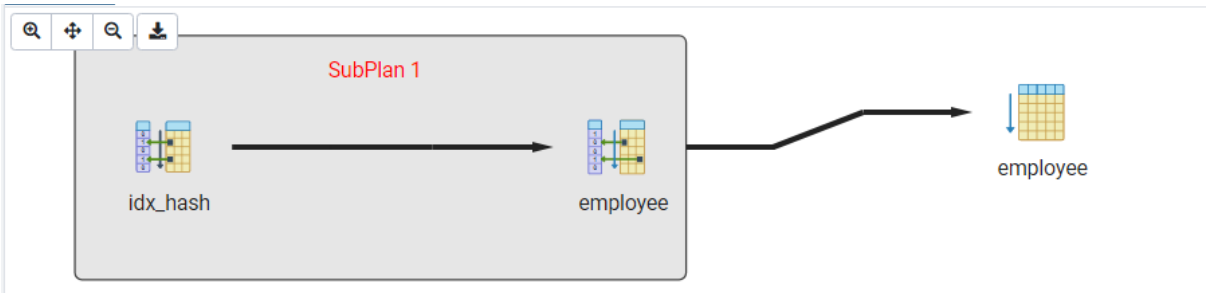
Query plan of optimized query (3) after using mixed indices :
Using btree index on table “Employee “ using “salary”
Using hash index on table “Employee” using “dno”
Cost = 476.97 rows
Execution time = 8.370 msec

```
54  --optimized query
55  Explain analyze
56  select lname, fname
57  from employee
58  where salary >  (
59  select max (salary)
60  from employee
61  where dno=5 );
62
63   create index idx_btree on employee using btree (salary);
64    create index idx_hash on employee using hash (dno);
65
```

Messages | Data Output

	QUERY PLAN text
1	Seq Scan on employee (cost=13.97..476.97 rows=5333 width=42) (actual time=6.029..8.039 rows=15330 loops=1)
2	[...] Filter: (salary > $1)
3	[...] Rows Removed by Filter: 670
4	[...] InitPlan 2 (returns $1)
5	[...] -> Result (cost=13.96..13.97 rows=1 width=4) (actual time=6.002..6.004 rows=1 loops=1)
6	[...] InitPlan 1 (returns $0)
7	[...] -> Limit (cost=0.29..13.96 rows=1 width=4) (actual time=5.999..6.000 rows=1 loops=1)
8	[...] -> Index Scan Backward using idx_btree on employee employee_1 (cost=0.29..1436.45 rows=105 width=4) (actual time=5.998..5.998 rows=1 loops=1)
9	[...] Index Cond: (salary IS NOT NULL)
10	[...] Filter: (dno = 5)
11	[...] Rows Removed by Filter: 15330
12	Planning Time: 0.131 ms
13	Execution Time: 8.370 ms

Physical plan of optimized query (3) after using mixed indices :
Using btree index on table “Employee “ using “salary”
Using hash index on table “Employee” using “dno”



Query plan of optimized query (3) after using brin index on table “employee” using “dno”:
Cost = 10000000938.39 rows
Execution time = 5.402 msec
Using Brin index increases cost to a very big value so using brin index was not efficient as Brin index is only efficient with queries that have exact values .

```
22
23  create index idx_brin on employee using brin (dno);
24 --optimized query
25 Explain analyze
26 select lname, fname
27 from employee
28 where salary >  (
29 select max (salary)
30 from employee
31 where dno=5 );
32
33
```

Data Output | Explain | Messages | Notifications

	QUERY PLAN text
1	Seq Scan on employee (cost=10000000475.39..10000000938.39 rows=5333 width=42) (actual time=2.388..5.013 rows=15330 loops=1)
2	[...] Filter: (salary > $0)
3	[...] Rows Removed by Filter: 670
4	[...] InitPlan 1 (returns $0)
5	[...] -> Aggregate (cost=475.38..475.39 rows=1 width=4) (actual time=2.365..2.365 rows=1 loops=1)
6	[...] -> Bitmap Heap Scan on employee employee_1 (cost=12.12..475.12 rows=105 width=4) (actual time=0.024..2.356 rows=105 loops=1)
7	[...] Recheck Cond: (dno = 5)
8	[...] Rows Removed by Index Recheck: 15895
9	[...] Heap Blocks: lossy=263
10	[...] -> Bitmap Index Scan on idx_brin (cost=0.00..12.10 rows=16000 width=0) (actual time=0.018..0.018 rows=2630 loops=1)
11	[...] Index Cond: (dno = 5)
12	Planning Time: 0.246 ms
13	Execution Time: 5.402 ms

Physical plan of optimized query (3) after using brin index on table “employee” using “dno”:

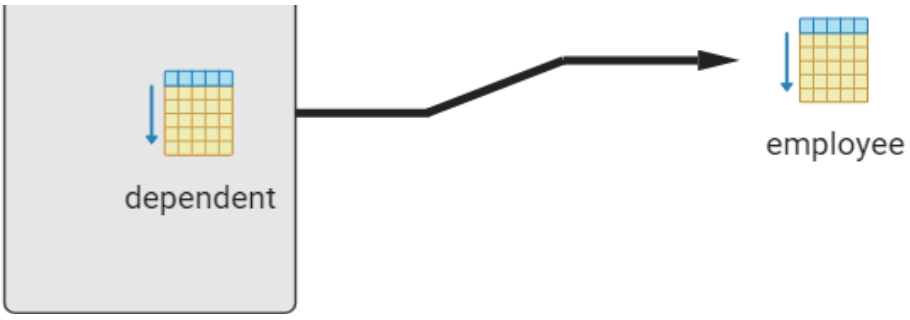
Query 4

- **Original Query**
  - **The original query without any index and its query plan and graph**
    - ➔ Cost: 140603.00
    - ➔ Execution time: 2858.827ms

```
SET enable_seqscan = On;
SET enable_async_append= on;
SET enable_bitmapscan= on;
SET enable_gathermerge=on;
SET enable_hashagg=on;
SET enable_hashjoin=on;
SET enable_incremental_sort=on;
SET enable_indexscan=On;
SET enable_indexonlyscan=On;
SET enable_material=on;
SET enable_memoize=on;
SET enable_mergejoin=on;
SET enable_sort=on;
SET enable_tidscan=on;
```

```
Explain analyze
select e.fname, e.lname
from employee as e
where e.ssn in ( select essn
                 from dependent as d
                 where e.fname = d.dependent_name
                 and e.sex = d.sex );
```

QUERY PLAN text
Seq Scan on employee e (cost=0.00..140603.00 rows=8000 width=42) (actual time=0.080..2858.689 rows=600 loops=1)
Filter: (SubPlan 1)
Rows Removed by Filter: 15400
SubPlan 1
-> Seq Scan on dependent d (cost=0.00..16.02 rows=601 width=4) (actual time=0.164..0.167 rows=11 loops=16000)
Filter: ((e.fname = dependent_name) AND (e.sex = sex))
Rows Removed by Filter: 578
Planning Time: 0.646 ms
Execution Time: 2858.827 ms

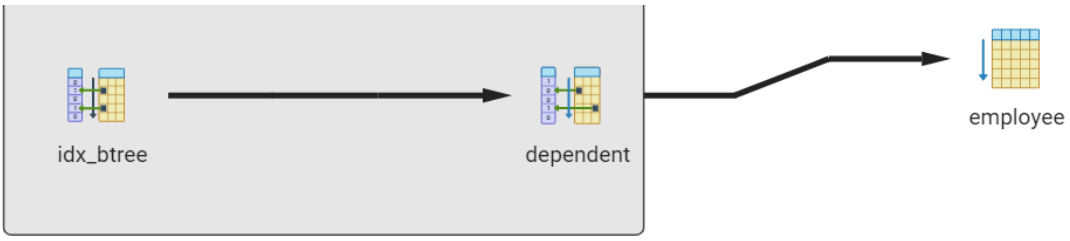


- **The original query with BTree index and its query plan and graph**
  - ➔ Cost: 10000345527.00
  - ➔ Execution time: 219.762ms

  The B+Tree I constructed did not optimize the query performance, as B+Trees are not effective with Queries that are working/returning all/most of the data.

```
create index idx_btree on dependent using btree(sex)
```

QUERY PLAN text
Seq Scan on employee e (cost=10000000000.00..10000345527.00 rows=8000 width=42) (actual time=0.071..219.687 rows=600 loops=1)
Filter: (SubPlan 1)
Rows Removed by Filter: 15400
SubPlan 1
-> Bitmap Heap Scan on dependent d (cost=12.81..28.82 rows=601 width=4) (actual time=0.003..0.005 rows=11 loops=16000)
Recheck Cond: (e.sex = sex)
Filter: (e.fname = dependent_name)
Rows Removed by Filter: 0
Heap Blocks: exact=2189
-> Bitmap Index Scan on idx_btree (cost=0.00..12.66 rows=601 width=0) (actual time=0.003..0.003 rows=23 loops=16000)
Index Cond: (sex = e.sex)
Planning Time: 0.355 ms
Execution Time: 219.762 ms

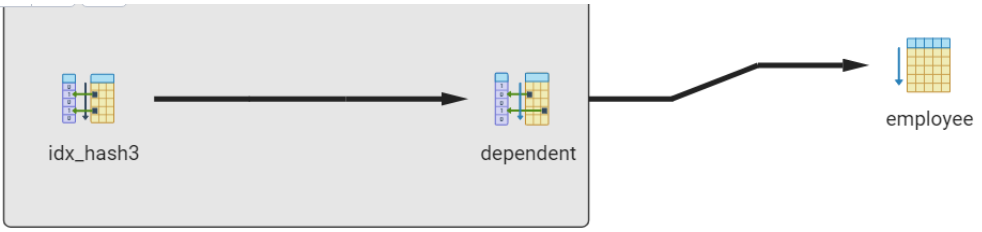


- **The original query with Hash index and its query plan and graph**
  - ➔ Cost: 10000535127.00
  - ➔ Execution time: 2651.535ms

  The Hash index I constructed did not optimize the query performance, as Hash indices are not effective with Queries returning all/most of the data.

```
create index idx_hash3 on dependent using hash(dependent_name)
```

QUERY PLAN text
Seq Scan on employee e (cost=10000000000.00..10000535127.00 rows=8000 width=42) (actual time=0.041..2651.470 rows=600 loops=1)
Filter: (SubPlan 1)
Rows Removed by Filter: 15400
SubPlan 1
-> Bitmap Heap Scan on dependent d (cost=24.66..40.67 rows=601 width=4) (actual time=0.154..0.157 rows=11 loops=16000)
Recheck Cond: (e.fname = dependent_name)
Filter: (e.sex = sex)
Rows Removed by Filter: 573
Heap Blocks: exact=108939
-> Bitmap Index Scan on idx_hash3 (cost=0.00..24.51 rows=601 width=0) (actual time=0.020..0.020 rows=595 loops=16000)
Index Cond: (dependent_name = e.fname)
Planning Time: 0.118 ms
Execution Time: 2651.535 ms



- **The original query with brin index and its query plan and graph**
  - ➔ Cost: 10000335519.00 rows
  - ➔ Execution time: 1716.555 msec

Using Brin index increases cost to a very big value so using brin index was not efficient as Brin index is only efficient with queries that have exact values ..

```
create index idx_brin on dependent using brin (dependent_name);

Explain Analyze
select e.fname, e.lname
from employee as e
where e.ssn in (
select essn
from dependent as d
where e.fname = d.dependent_name
and
e.sex = d.sex );
```

Data Output | Explain | Messages | Notifications

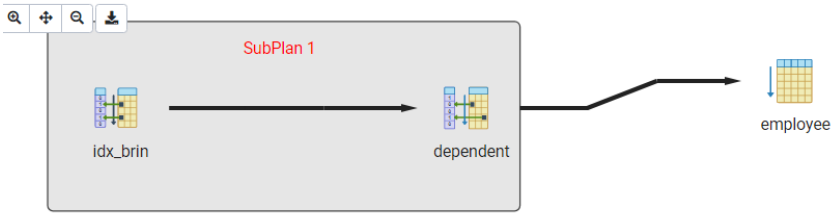
	QUERY PLAN text
1	Seq Scan on employee e (cost=10000000000.00..10000335519.00 rows=8000 width=42) (actual time=0.049..1704.797 rows=600 loops=1)
2	[...] Filter: (SubPlan 1)
3	[...] Rows Removed by Filter: 15400
4	[...] SubPlan 1
5	[...] -> Bitmap Heap Scan on dependent d (cost=12.18..28.20 rows=601 width=4) (actual time=0.100..0.102 rows=11 loops=16000)
6	[...] Recheck Cond: (e.fname = dependent_name)
7	[...] Filter: (e.sex = sex)
8	[...] Rows Removed by Filter: 573
9	[...] Heap Blocks: lossy=108939
10	[...] -> Bitmap Index Scan on idx_brin (cost=0.00..12.03 rows=601 width=0) (actual time=0.007..0.007 rows=69 loops=16000)
11	[...] Index Cond: (dependent_name = e.fname)
12	Planning Time: 0.341 ms
13	Execution Time: 1716.555 ms

```
create index idx_brin on dependent using brin (dependent_name);

Explain Analyze
select e.fname, e.lname
from employee as e
where e.ssn in (
select essn
from dependent as d
where e.fname = d.dependent_name
and
e.sex = d.sex );
```

Data Output | Explain | Messages | Notifications

Graphical | Analysis | Statistics

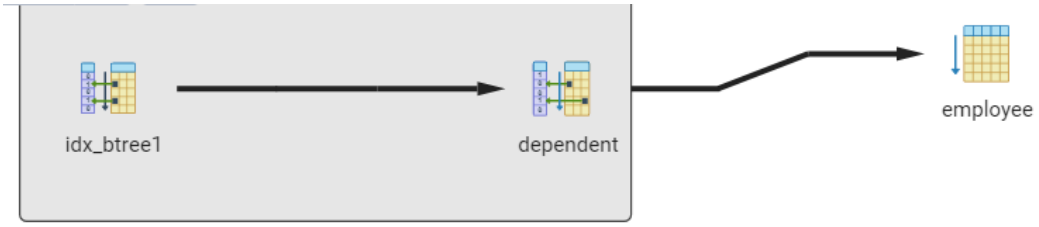


- **The original query with mixed index and its query plan and graph**
  - ➔ Cost: 10000345527.00
  - ➔ Execution time: 202.865ms

  The Hash index & B+Tree I constructed did not optimize the query performance, as Hash indices and B+ Trees are not effective with Queries returning all/most of the data.

```
create index idx_btree1 on dependent using btree(sex)
create index idx_hash4 on employee using hash(ssn)
```

QUERY PLAN text
Seq Scan on employee e (cost=10000000000.00..10000345527.00 rows=8000 width=42) (actual time=0.042..202.765 rows=600 loops=1)
Filter: (SubPlan 1)
Rows Removed by Filter: 15400
SubPlan 1
-> Bitmap Heap Scan on dependent d (cost=12.81..28.82 rows=601 width=4) (actual time=0.003..0.005 rows=11 loops=16000)
Recheck Cond: (e.sex = sex)
Filter: (e.fname = dependent_name)
Rows Removed by Filter: 0
Heap Blocks: exact=2189
-> Bitmap Index Scan on idx_btree1 (cost=0.00..12.66 rows=601 width=0) (actual time=0.003..0.003 rows=23 loops=16000)
Index Cond: (sex = e.sex)
Planning Time: 0.478 ms
Execution Time: 202.865 ms

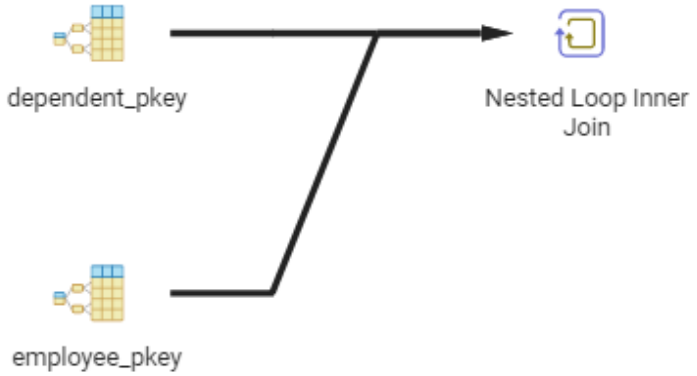


- # Alternative Query
  - **The alternative query without any index and its query plan and graph**
    - ➔ Cost: 1477.98
    - ➔ Execution time: 0.894 ms

```
SET enable_seqscan = off;
SET enable_async_append= on;
SET enable_bitmapscan= on;
SET enable_gathermerge=on;
SET enable_hashagg=on;
SET enable_hashjoin=off;
SET enable_incremental_sort=on;
SET enable_indexscan=on;
SET enable_indexonlyscan=on;
SET enable_material=on;
SET enable_memoize=on;
SET enable_mergejoin=off;
SET enable_nestloop =on;
SET enable_parallel_append=on;
SET enable_parallel_hash=on;
SET enable_partition_pruning=on;
SET enable_partitionwise_join =on;
SET enable_partitionwise_aggregate =on;
SET enable_sort=on;
SET enable_tidscan=on;
--create index indexn4 on dependent using btree(essn,dependent_name,sex)
--drop index indexn4
explain analyze

select e.fname, e.lname
from employee e inner join dependent d on d.essn=e.ssn
where e.fname = d.dependent_name
and
e.sex = d.sex;
```

	QUERY PLAN text
1	Nested Loop (cost=0.56..1477.98 rows=22 width=42) (actual time=0.024..0.861 rows=600 loops=1)
2	-> Index Scan using dependent_pkey on dependent d (cost=0.28..51.16 rows=601 width=27) (actual time=0.013..0.086 rows=601 loops=1)
3	-> Index Scan using employee_pkey on employee e (cost=0.29..2.36 rows=1 width=48) (actual time=0.001..0.001 rows=1 loops=601)
4	Index Cond: (ssn = d.essn)
5	Filter: ((d.dependent_name = fname) AND (d.sex = sex))
6	Rows Removed by Filter: 0
7	Planning Time: 0.166 ms
8	Execution Time: 0.894 ms



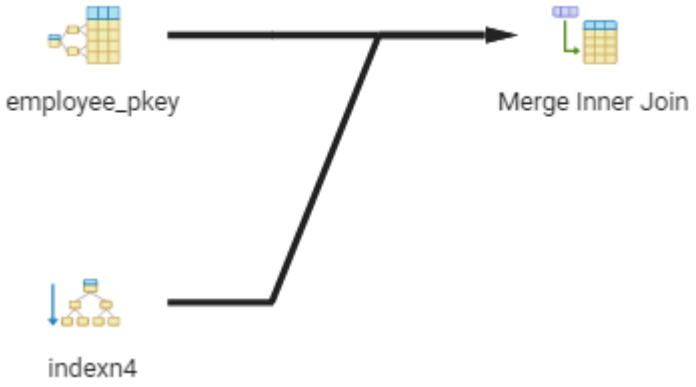
- **The alternative query with BTree index and its query plan and graph**
  - ➔ Cost: 89.39
  - ➔ Execution time: 1.379 ms

```
SET enable_seqscan = off;
SET enable_async_append= on;
SET enable_bitmapscan= on;
SET enable_gathermerge=on;
SET enable_hashagg=on;
SET enable_hashjoin=off;
SET enable_incremental_sort=on;
SET enable_indexscan=on;
SET enable_indexonlyscan=on;
SET enable_material=on;
SET enable_memoize=on;
SET enable_mergejoin=on;
SET enable_nestloop =on;
SET enable_parallel_append=on;
SET enable_parallel_hash=on;
SET enable_partition_pruning=on;
SET enable_partitionwise_join =on;
SET enable_partitionwise_aggregate =on;
SET enable_sort=on;
SET enable_tidscan=on;
--create index indexn4 on dependent using btree(essn,dependent_name,sex)
--drop index indexn4
explain analyze

select e.fname, e.lname
from employee e inner join dependent d on d.essn=e.ssn
where e.fname = d.dependent_name
and
e.sex = d.sex;

```

	QUERY PLAN text
1	Merge Join (cost=0.56..89.39 rows=22 width=42) (actual time=0.040..1.241 rows=600 loops=1)
2	Merge Cond: (e.ssn = d.essn)
3	Join Filter: ((e.fname = d.dependent_name) AND (e.sex = d.sex))
4	Rows Removed by Join Filter: 1
5	-> Index Scan using employee_pkey on employee e (cost=0.29..690.28 rows=16000 width=48) (actual time=0.014..0.206 rows=602 loops...
6	-> Index Only Scan using indexn4 on dependent d (cost=0.28..51.16 rows=601 width=27) (actual time=0.017..0.349 rows=601 loops=1)
7	Heap Fetches: 601
8	Planning Time: 2.685 ms
9	Execution Time: 1.379 ms



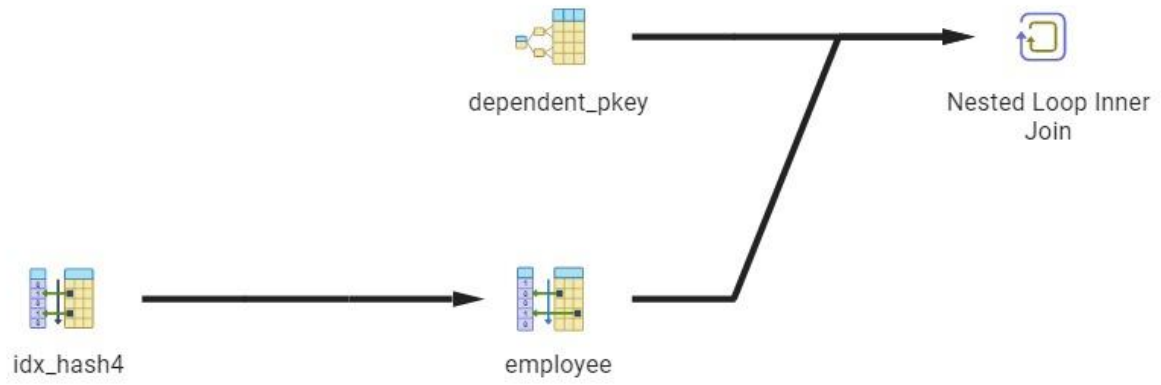
- **The alternative query with Hash index and its query plan and graph**
  - ➔ Cost: 10000002740.35
  - ➔ Execution time: 3.491 ms

  And so here we can see how the cost of my query increased that high while trying to optimize it using hash indices. So it doesn't enhance my performance at all ,however it makes it worse. Also this might be because a hash based index should be used when you want to enhance the performance of SQL queries on exact value and do not care (have) any other query types and as we can see here we are not doing our query on exact values, hence it doesn't work and was very inefficient for me to use.

Query Query History

```
1  SET enable_seqscan = Off;
2  SET enable_async_append= on;
3  SET enable_bitmapscan= on;
4  SET enable_gathermerge=on;
5  SET enable_hashagg=off;
6  SET enable_hashjoin=off;
7  SET enable_incremental_sort=on;
8  SET enable_indexscan=Off;
9  SET enable_indexonlyscan=On;
10 SET enable_material=on;
11 SET enable_memoize=on;
12 SET enable_mergejoin=on;
13 SET enable_sort=on;
14 SET enable_tidscan=on;
15
16
17 --create index indexn44 on employee using hash(ssn)
18 --drop index indexn44
19 explain analyze
20
21 select e.fname, e.lname
22 from employee e inner join dependent d on d.essn=e.ssn
23 where e.fname = d.dependent_name
24 and
25 e.sex = d.sex;
```

QUERY PLAN text
Nested Loop (cost=10000000000.72..10000002740.35 rows=22 width=42) (actual time=0.045..3.419 rows=600 loops=1)
-> Index Scan using dependent_pkey on dependent d (cost=10000000000.27..10000000051.16 rows=601 width=27) (actual time=0.004..0.080 rows=601 loops=1)
-> Bitmap Heap Scan on employee e (cost=0.45..4.46 rows=1 width=48) (actual time=0.002..0.002 rows=1 loops=601)
Recheck Cond: (ssn = d.essn)
Filter: ((d.dependent_name = fname) AND (d.sex = sex))
Rows Removed by Filter: 0
Heap Blocks: exact=601
-> Bitmap Index Scan on idx_hash4 (cost=0.00..0.45 rows=1 width=0) (actual time=0.001..0.001 rows=1 loops=601)
Index Cond: (ssn = d.essn)
Planning Time: 0.299 ms
Execution Time: 3.491 ms



- **The alternative query with brin index and its query plan and graph**

  My query using BRIN indices only did not work because the query is not an analytical query and does not involve narrow-range search on columns with a huge range of values.

So it wasn't an efficient way for optimizing my query to do better performance.

- **The alternative query with mixed index and its query plan and graph**

  ➔ Cost: 89.39
  ➔ Execution time:0.552

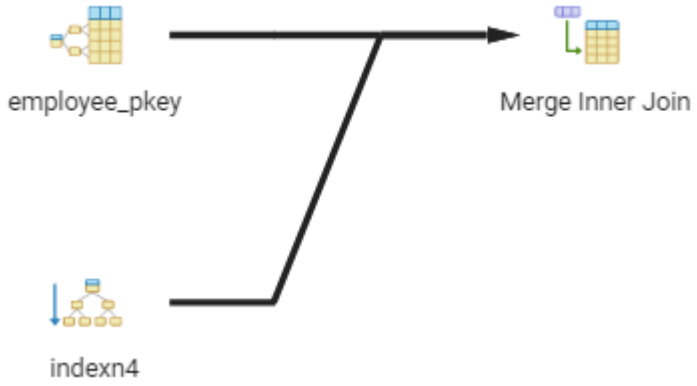
```
SET enable_seqscan = off;
SET enable_async_append= on;
SET enable_bitmapscan= on;
SET enable_gathermerge=on;
SET enable_hashagg=on;
SET enable_hashjoin=off;
SET enable_incremental_sort=on;
SET enable_indexscan=on;
SET enable_indexonlyscan=on;
SET enable_material=on;
SET enable_memoize=on;
SET enable_mergejoin=on;
SET enable_nestloop =on;
SET enable_parallel_append=on;
SET enable_parallel_hash=on;
SET enable_partition_pruning=on;
SET enable_partitionwise_join =on;
SET enable_partitionwise_aggregate =on;
SET enable_sort=on;
SET enable_tidscan=on;

--create index indexn4 on dependent using btree(essn,dependent_name,sex)
--create index indexn44 on dependent using hash(sex)
--drop index indexn4
explain analyze

select e.fname, e.lname
from employee e inner join dependent d on d.essn=e.ssn
where e.fname = d.dependent_name
and
e.sex = d.sex;

```

	QUERY PLAN text
1	Merge Join (cost=0.56..89.39 rows=22 width=42) (actual time=0.017..0.509 rows=600 loops=1)
2	Merge Cond: (e.ssn = d.essn)
3	Join Filter: ((e.fname = d.dependent_name) AND (e.sex = d.sex))
4	Rows Removed by Join Filter: 1
5	-> Index Scan using employee_pkey on employee e (cost=0.29..690.28 rows=16000 width=48) (actual time=0.006..0.102 rows=602 loops...
6	-> Index Only Scan using indexn4 on dependent d (cost=0.28..51.16 rows=601 width=27) (actual time=0.005..0.155 rows=601 loops=1)
7	Heap Fetches: 601
8	Planning Time: 2.302 ms
9	Execution Time: 0.552 ms



Query 5

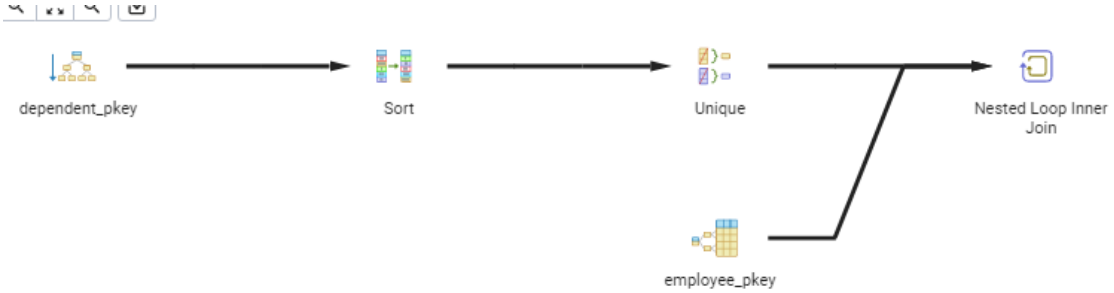
- **Original Query**
  - The original query without any index and its query plan and graph.
    - ➔ Cost: 1455.48
    - ➔ Execution time: 1.288 ms

```
SET enable_seqscan = off;
SET enable_async_append= on;
SET enable_bitmapscan= on;
SET enable_gathermerge=on;
SET enable_hashagg=off;
SET enable_hashjoin=off;
SET enable_incremental_sort=on;
SET enable_indexscan=on;
SET enable_indexonlyscan=on;
SET enable_material=off;
SET enable_memoize=on;
SET enable_mergejoin=off;
SET enable_nestloop =on;
SET enable_parallel_append=on;
SET enable_parallel_hash=off;
SET enable_partition_pruning=off;
SET enable_partitionwise_join =off;
SET enable_partitionwise_aggregate =off;
SET enable_sort=on;
SET enable_tidscan=on;


--drop index indexb5
--create index indexb5 on dependent  using btree(essn);

Explain analyze
select fname, lname
from employee
where exists ( select *
 from dependent
 where ssn=essn );
```

	QUERY PLAN text
1	Nested Loop (cost=79.19..1455.48 rows=601 width=42) (actual time=0.256..1.184 rows=601 loops=1)
2	-> Unique (cost=78.90..81.91 rows=601 width=4) (actual time=0.244..0.343 rows=601 loops=1)
3	-> Sort (cost=78.90..80.41 rows=601 width=4) (actual time=0.243..0.270 rows=601 loops=1)
4	Sort Key: dependent.essn
5	Sort Method: quicksort Memory: 53kB
6	-> Index Only Scan using dependent_pkey on dependent (cost=0.28..51.16 rows=601 width=4) (actual time=0.013..0.167 rows=601 loops=1)
7	Heap Fetches: 601
8	-> Index Scan using employee_pkey on employee (cost=0.29..2.36 rows=1 width=46) (actual time=0.001..0.001 rows=1 loops=601)
9	Index Cond: (ssn = dependent.essn)
10	Planning Time: 1.144 ms
11	Execution Time: 1.228 ms

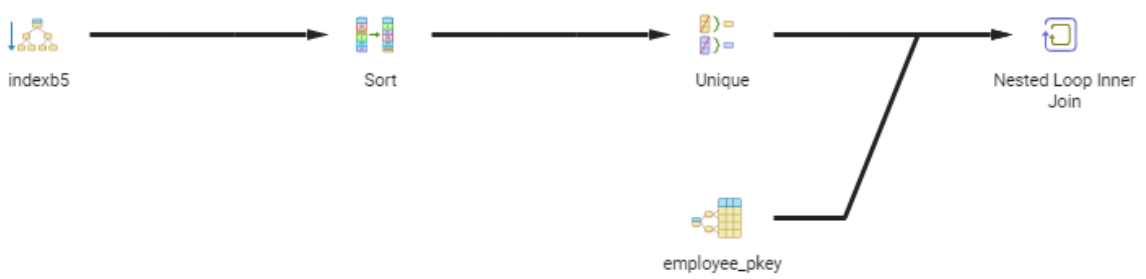


- My query with B+ trees indices only. Optimizing my original query using Btree optimizes my query because it searches with O(log n). So here is the optimized query and its query plan and graph.My performance was enhanced efficiently as we can see also from cost.
  - ➔ Cost: 1439.60
  - ➔ Execution time: 1.539 ms

Query Query History

```
1  SET enable_seqscan = off;
2  SET enable_async_append= on;
3  SET enable_bitmapscan= on;
4  SET enable_gathermerge=on;
5  SET enable_hashagg=off;
6  SET enable_hashjoin=off;
7  SET enable_incremental_sort=on;
8  SET enable_indexscan=on;
9  SET enable_indexonlyscan=on;
10 SET enable_material=off;
11 SET enable_memoize=on;
12 SET enable_mergejoin=off;
13 SET enable_nestloop =on;
14 SET enable_parallel_append=on;
15 SET enable_parallel_hash=off;
16 SET enable_partition_pruning=off;
17 SET enable_partitionwise_join =off;
18 SET enable_partitionwise_aggregate =off;
19 SET enable_sort=on;
20 SET enable_tidscan=on;
21
22
23 --drop index indexb5
24 --create index indexb5 on dependent  using btree(essn);
25
26 Explain analyze
27 select fname, lname
28 from employee
29 where exists ( select *
30  from dependent
31  where ssn=essn );
32
```

	QUERY PLAN text
1	Nested Loop (cost=63.31..1439.60 rows=601 width=42) (actual time=0.511..1.475 rows=601 loops=1)
2	-> Unique (cost=63.03..66.03 rows=601 width=4) (actual time=0.465..0.567 rows=601 loops=1)
3	-> Sort (cost=63.03..64.53 rows=601 width=4) (actual time=0.464..0.489 rows=601 loops=1)
4	Sort Key: dependent.essn
5	Sort Method: quicksort Memory: 53kB
6	-> Index Only Scan using indexb5 on dependent (cost=0.28..35.29 rows=601 width=4) (actual time=0.203..0.379 rows=601 loops=1)
7	Heap Fetches: 601
8	-> Index Scan using employee_pkey on employee (cost=0.29..2.36 rows=1 width=46) (actual time=0.001..0.001 rows=1 loops=601)
9	Index Cond: (ssn = dependent.essn)
10	Planning Time: 2.177 ms
11	Execution Time: 1.539 ms



- My query with hash indices only. Optimizing my original query optimizes my query because it searches with O(1). So here is the optimized query and its

query plan and graph.My performance was enhanced efficiently as we can see also from cost.

➔ Cost: 1420.29

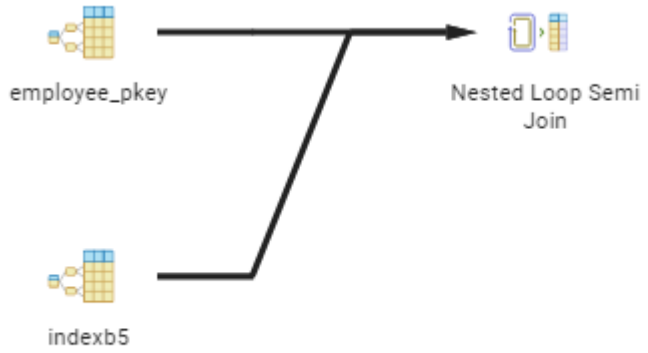
➔ Execution time: 9.961 ms

```
SET enable_seqscan = off;
SET enable_async_append= on;
SET enable_bitmapscan= on;
SET enable_gathermerge=on;
SET enable_hashagg=off;
SET enable_hashjoin=off;
SET enable_incremental_sort=on;
SET enable_indexscan=on;
SET enable_indexonlyscan=on;
SET enable_material=off;
SET enable_memoize=on;
SET enable_mergejoin=off;
SET enable_nestloop =on;
SET enable_parallel_append=on;
SET enable_parallel_hash=off;
SET enable_partition_pruning=off;
SET enable_partitionwise_join =off;
SET enable_partitionwise_aggregate =off;
SET enable_sort=on;
SET enable_tidscan=on;


--drop index indexb5
--create index indexb5 on dependent  using hash(essn);

Explain analyze
select fname, lname
from employee
where exists ( select *
 from dependent
 where ssn=essn );
```

	QUERY PLAN text
1	Nested Loop Semi Join (cost=0.29..1020.29 rows=601 width=42) (actual time=0.035..9.914 rows=601 loops=1)
2	-> Index Scan using employee_pkey on employee (cost=0.29..690.28 rows=16000 width=46) (actual time=0.022..2.370 rows=16000 loops...
3	-> Index Scan using indexb5 on dependent (cost=0.00..0.02 rows=1 width=4) (actual time=0.000..0.000 rows=0 loops=16000)
4	Index Cond: (essn = employee.ssn)
5	Planning Time: 0.166 ms
6	Execution Time: 9.961 ms



- My query with mixed indices only. Optimizing my original query using mixed indices and it does optimize my query . So here is the optimized query and its query plan and graph. Also i used btree and hashing for my mix indices optimization.My performance was enhanced efficiently as we can see also from cost.

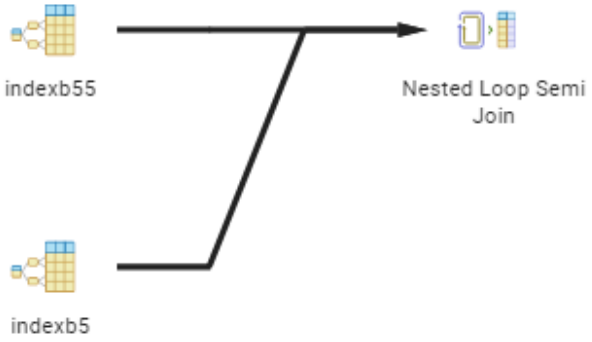
  ➔ Cost: 1020.29

  ➔ Execution time: 42.168 ms

Query | Query History

```
1  SET enable_seqscan = off;
2  SET enable_async_append= on;
3  SET enable_bitmapscan= on;
4  SET enable_gathermerge=on;
5  SET enable_hashagg=off;
6  SET enable_hashjoin=off;
7  SET enable_incremental_sort=on;
8  SET enable_indexscan=on;
9  SET enable_indexonlyscan=on;
10 SET enable_material=off;
11 SET enable_memoize=on;
12 SET enable_mergejoin=off;
13 SET enable_nestloop =on;
14 SET enable_parallel_append=on;
15 SET enable_parallel_hash=off;
16 SET enable_partition_pruning=off;
17 SET enable_partitionwise_join =off;
18 SET enable_partitionwise_aggregate =off;
19 SET enable_sort=on;
20 SET enable_tidscan=on;
21
22
23 --drop index indexb5
24 --drop index indexb55
25 --create index indexb5 on dependent using hash(essn);
26 --create index indexb55 on employee using btree(ssn);
27
28
29 Explain analyze
30 select fname, lname
31 from employee
32 where exists ( select *
33  from dependent
34  where ssn=essn );
35
```

	QUERY PLAN text
1	Nested Loop Semi Join (cost=0.29..1020.29 rows=601 width=42) (actual time=0.106..42.109 rows=601 loops=1)
2	-> Index Scan using indexb55 on employee (cost=0.29..690.28 rows=16000 width=46) (actual time=0.088..33.438 rows=16000 loops...
3	-> Index Scan using indexb5 on dependent (cost=0.00..0.02 rows=1 width=4) (actual time=0.000..0.000 rows=0 loops=16000)
4	Index Cond: (essn = employee.ssn)
5	Planning Time: 3.753 ms
6	Execution Time: 42.168 ms



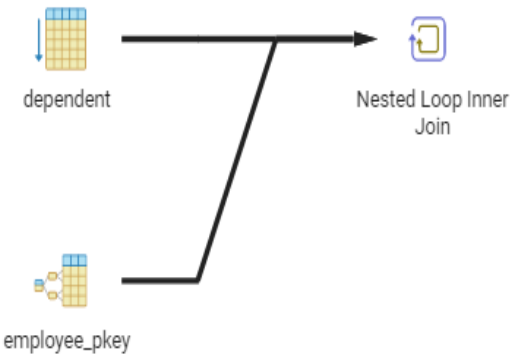
- My query using BRIN indices did not work because the query is not an analytical query and does not involve narrow-range search on columns with a huge range of values.So it wasn't an efficient way for optimizing my query to do better performance.

- **Alternative Query**
  - The alternative query to the original one without any index and its query plan and graph. This alternative query gives the same result as the original one and at the same time it reduces cost and so optimizes the query.My performance was enhanced efficiently as we can see also from cost.
    - ➔ Cost: 1430.81
    - ➔ Execution time: 0.775 ms

```
SET enable_seqscan = off;
SET enable_async_append= on;
SET enable_bitmapscan= on;
SET enable_gathermerge=on;
SET enable_hashagg=off;
SET enable_hashjoin=off;
SET enable_incremental_sort=on;
SET enable_indexscan=on;
SET enable_indexonlyscan=on;
SET enable_material=off;
SET enable_memoize=on;
SET enable_mergejoin=off;
SET enable_nestloop =on;
SET enable_parallel_append=on;
SET enable_parallel_hash=off;
SET enable_partition_pruning=off;
SET enable_partitionwise_join =off;
SET enable_partitionwise_aggregate =off;
SET enable_sort=on;
SET enable_tidscan=on;


Explain analyze
select fname, lname
from employee , dependent
where  ssn=essn;
```

QUERY PLAN text
Nested Loop (cost=0.29..1430.81 rows=601 width=42) (actual time=0.021..0.747 rows=601 loops=1)
-> Seq Scan on dependent d (cost=0.00..13.01 rows=601 width=4) (actual time=0.009..0.044 rows=601 loops=1)
-> Index Scan using employee_pkey on employee e (cost=0.29..2.36 rows=1 width=46) (actual time=0.001..0.001 rows=1 loops=6...
Index Cond: (ssn = d.essn)
Planning Time: 0.145 ms
Execution Time: 0.775 ms



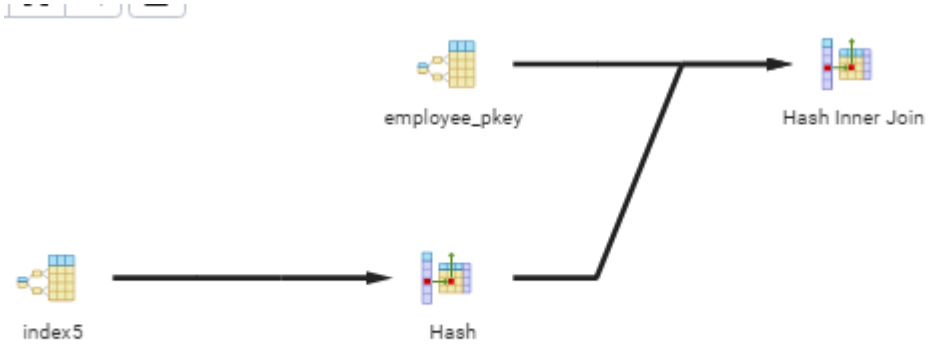
- My alternative query with B+ trees indices only. Optimizing my alternative query using Btree optimizes my query because it searches with O(log n). So here is the optimized query and its query plan and graph.My performance was enhanced efficiently as we can see also from from cost.
    - ➔ Cost: 799.10
    - ➔ Execution time: 3.279 ms

```
SET enable_seqscan = off;
SET enable_async_append= on;
SET enable_bitmapscan= off;
SET enable_gathermerge=off;
SET enable_hashagg=off;
SET enable_hashjoin=on;
SET enable_incremental_sort=off;
SET enable_indexscan=on;
SET enable_indexonlyscan=off;
SET enable_material=off;
SET enable_memoize=off;
SET enable_mergejoin=off;
SET enable_nestloop =on;
SET enable_parallel_append=off;
SET enable_parallel_hash=off;
SET enable_partition_pruning=off;
SET enable_partitionwise_join =off;
SET enable_partitionwise_aggregate =off;
SET enable_sort=off;
SET enable_tidscan=off;

--create index index5 on dependent using btree(essn);

Explain analyze
select fname, lname
from employee e
inner join dependent d on e.ssn=d.essn ;
```

	QUERY PLAN text
1	Hash Join (cost=43.09..799.10 rows=601 width=42) (actual time=0.223..3.245 rows=601 loops=1)
2	Hash Cond: (e.ssn = d.essn)
3	-> Index Scan using employee_pkey on employee e (cost=0.29..690.28 rows=16000 width=46) (actual time=0.014..1.660 rows=16000 loops...
4	-> Hash (cost=35.29..35.29 rows=601 width=4) (actual time=0.202..0.204 rows=601 loops=1)
5	Buckets: 1024 Batches: 1 Memory Usage: 30kB
6	-> Index Scan using index5 on dependent d (cost=0.28..35.29 rows=601 width=4) (actual time=0.060..0.141 rows=601 loops=1)
7	Planning Time: 0.223 ms
8	Execution Time: 3.279 ms



- My alternative query with hash indices only. Optimizing my alternative query with hashing optimizes my query because it searches with O(1). So here is the optimized query and its query plan and graph.My performance was enhanced efficiently as we can see also from cost.
    - ➔ Cost: 907.00
    - ➔ Execution time: 12.970 ms

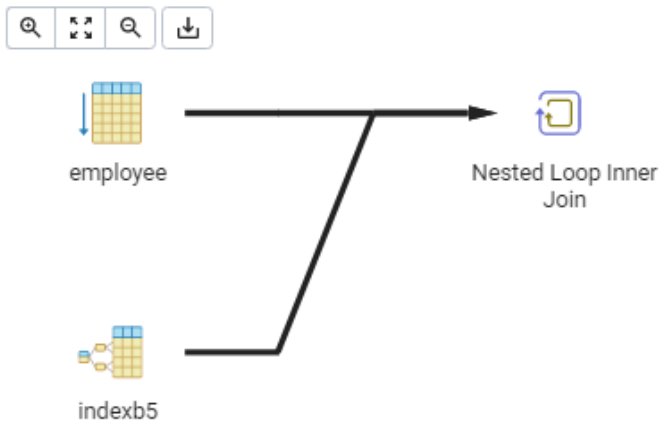
```
SET enable_seqscan = on;
SET enable_async_append= on;
SET enable_bitmapscan= off;
SET enable_gathermerge=off;
SET enable_hashagg=off;
SET enable_hashjoin=off;
SET enable_incremental_sort=off;
SET enable_indexscan=on;
SET enable_indexonlyscan=off;
SET enable_material=off;
SET enable_memoize=off;
SET enable_mergejoin=off;
SET enable_nestloop =on;
SET enable_parallel_append=off;
SET enable_parallel_hash=off;
SET enable_partition_pruning=off;
SET enable_partitionwise_join =off;
SET enable_partitionwise_aggregate =off;
SET enable_sort=off;
SET enable_tidscan=off;

--drop index indexb5
--create index indexb5 on dependent using hash(essn);


Explain analyze
select fname, lname
from employee e
inner join dependent d on e.ssn=d.essn ;

```

	QUERY PLAN text
1	Nested Loop (cost=0.00..907.00 rows=601 width=42) (actual time=0.028..12.930 rows=601 loops=1)
2	-> Seq Scan on employee e (cost=0.00..423.00 rows=16000 width=46) (actual time=0.014..1.441 rows=16000 loops=1)
3	-> Index Scan using indexb5 on dependent d (cost=0.00..0.02 rows=1 width=4) (actual time=0.000..0.000 rows=0 loops=160...
4	Index Cond: (essn = e.ssn)
5	Planning Time: 0.184 ms
6	Execution Time: 12.970 ms



- **My alternative query with mixed indices only.**

  Optimizing my alternative query using mixed indices and it does optimize my query . So here is the optimized query and its query plan and graph. Also i used btree and hashing for my mix indices optimization and it lowered my cost .My performance was enhanced efficiently as we can see also from cost.

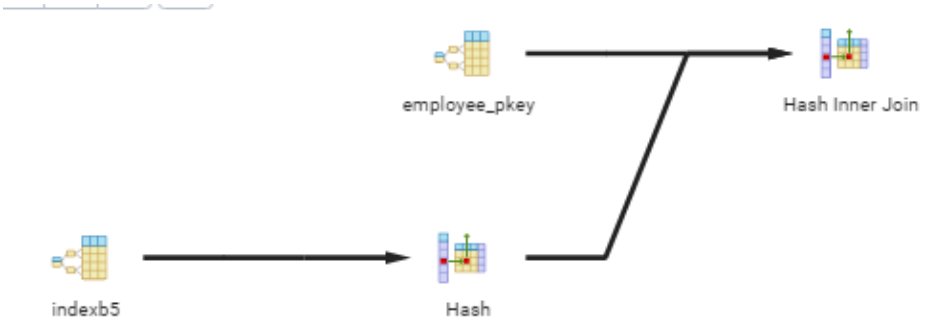
  - ➔ Cost: 799.10
  - ➔ Execution time : 3.821 ms

```
SET enable_seqscan = off;
SET enable_async_append= on;
SET enable_bitmapscan= off;
SET enable_gathermerge=off;
SET enable_hashagg=off;
SET enable_hashjoin=on;
SET enable_incremental_sort=off;
SET enable_indexscan=on;
SET enable_indexonlyscan=off;
SET enable_material=off;
SET enable_memoize=off;
SET enable_mergejoin=off;
SET enable_nestloop =on;
SET enable_parallel_append=off;
SET enable_parallel_hash=off;
SET enable_partition_pruning=off;
SET enable_partitionwise_join =off;
SET enable_partitionwise_aggregate =off;
SET enable_sort=off;
SET enable_tidscan=off;

--drop index indexb5
--create index indexb5 on dependent using btree(essn);
--create index indexb55 on dependent using hash(dependent_name)
Explain analyze
select fname, lname
from employee e
inner join dependent d on e.ssn=d.essn ;

```

	QUERY PLAN text
1	Hash Join (cost=43.09..799.10 rows=601 width=42) (actual time=0.236..3.768 rows=601 loops=1)
2	Hash Cond: (e.ssn = d.essn)
3	-> Index Scan using employee_pkey on employee e (cost=0.29..690.28 rows=16000 width=46) (actual time=0.019..2.025 rows=16000 loops...
4	-> Hash (cost=35.29..35.29 rows=601 width=4) (actual time=0.206..0.208 rows=601 loops=1)
5	Buckets: 1024 Batches: 1 Memory Usage: 30kB
6	-> Index Scan using indexb5 on dependent d (cost=0.28..35.29 rows=601 width=4) (actual time=0.055..0.143 rows=601 loops=1)
7	Planning Time: 2.479 ms
8	Execution Time: 3.821 ms



- My query using BRIN indices only did not work because the query is not an analytical query and does not involve narrow-range search on column with a huge range of values.So it wasn't an efficient way for optimizing my query to do better performance and did not lower my cost also.

Query 6

- **Original Query**
  - The original query without any index and its query plan and graph.
    - ➔ Cost: 2430.76
    - ➔ Execution time: 15.474 ms

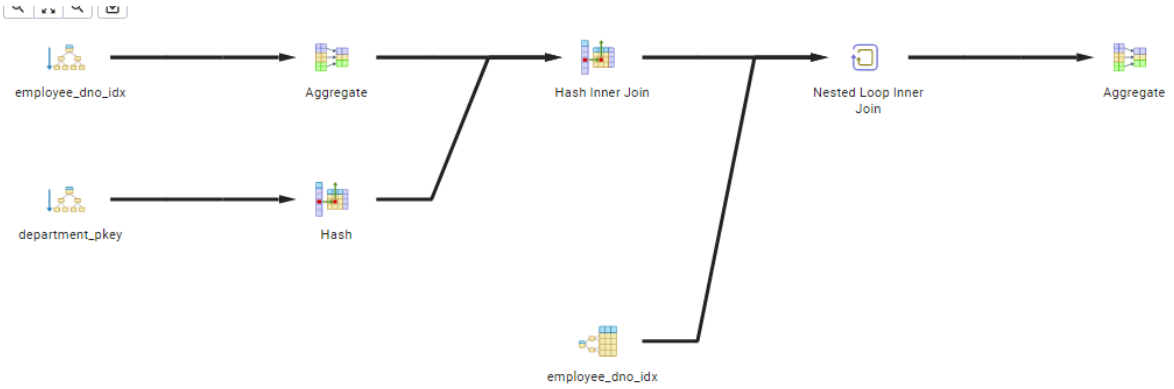
```
SET enable_seqscan = off;
SET enable_async_append= off;
SET enable_bitmapscan= on;
SET enable_gathermerge=on;
SET enable_hashagg=off;
SET enable_hashjoin=off;
SET enable_incremental_sort=on;
SET enable_indexscan=on;
SET enable_indexonlyscan=off;
SET enable_material=on;
SET enable_memoize=off;
SET enable_mergejoin=off;
SET enable_nestloop =on;
SET enable_parallel_append=off;
SET enable_parallel_hash=on;
SET enable_partition_pruning=on;
SET enable_partitionwise_join =on;
SET enable_partitionwise_aggregate =on;
SET enable_sort=off;
SET enable_tidscan=off;

Explain Analyze
select dnumber, count(*)
from department, employee
where dnumber=dno
and
salary > 40000
and
dno in (
 select dno
 from employee
 group by dno
 having count (*) > 5)
group by dnumber;

```

Data output | Messages | Notifications

	QUERY PLAN text
1	GroupAggregate (cost=0.72..2430.76 rows=150 width=12) (actual time=0.236..15.419 rows=150 loops=1)
2	Group Key: department.dnumber
3	-> Nested Loop (cost=0.72..2402.51 rows=5351 width=4) (actual time=0.132..14.262 rows=15854 loops=1)
4	-> Nested Loop (cost=0.43..1475.43 rows=99 width=8) (actual time=0.129..8.025 rows=150 loops=1)
5	-> GroupAggregate (cost=0.29..1442.53 rows=99 width=4) (actual time=0.123..7.728 rows=150 loops=1)
6	Group Key: employee_1.dno
7	Filter: (count(*) > 5)
8	Rows Removed by Filter: 146
9	-> Index Scan using employee_dno_idx on employee employee_1 (cost=0.29..1358.83 rows=16000 width=4) (actual time=0.046..5.613 rows=16000 loops=1)
10	-> Index Scan using department_pkey on department (cost=0.14..0.32 rows=1 width=4) (actual time=0.001..0.001 rows=1 loops=150)
11	Index Cond: (dnumber = employee_1.dno)
12	-> Index Scan using employee_dno_idx on employee (cost=0.29..8.82 rows=54 width=4) (actual time=0.002..0.035 rows=106 loops=150)
13	Index Cond: (dno = department.dnumber)
14	Filter: (salary > 40000)
15	Planning Time: 1.785 ms
16	Execution Time: 15.474 ms



- **My query with B+ trees indices only.**

  Optimizing my original query using Btree optimizes my query because it searches with O(log n). So here is the optimized query and its query plan and graph.My performance was enhanced efficiently as we can see also from cost.

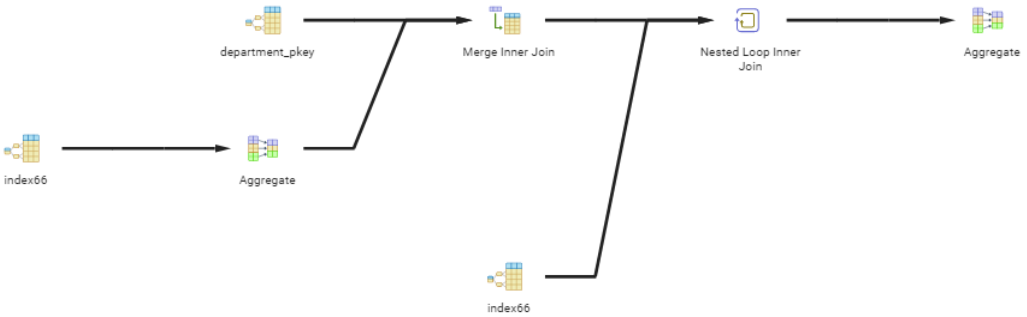
  ➔ Cost: 2415.86

  ➔ Execution time: 13.276 ms

Query | Query History

```
1  SET enable_seqscan = off;
2  SET enable_async_append= off;
3  SET enable_bitmapscan= on;
4  SET enable_gathermerge=on;
5  SET enable_hashagg=off;
6  SET enable_hashjoin=off;
7  SET enable_incremental_sort=on;
8  SET enable_indexscan=on;
9  SET enable_indexonlyscan=off;
10 SET enable_material=on;
11 SET enable_memoize=off;
12 SET enable_mergejoin=on;
13 SET enable_nestloop =on;
14 SET enable_parallel_append=off;
15 SET enable_parallel_hash=on;
16 SET enable_partition_pruning=on;
17 SET enable_partitionwise_join =on;
18 SET enable_partitionwise_aggregate =on;
19 SET enable_sort=off;
20 SET enable_tidscan=off;
21
22 --create index index66 on employee using btree(dno);
23 --drop index index66
24
25 Explain Analyze
26 select dnumber, count(*)
27 from department, employee
28 where dnumber=dno
29 and
30 salary > 40000
31 and
32 dno in (
33  select dno
34  from employee
35  group by dno
36  having count (*) > 5)
37 group by dnumber;
```

	QUERY PLAN text
1	GroupAggregate (cost=0.72..2415.86 rows=150 width=12) (actual time=0.217..13.220 rows=150 loops=1)
2	Group Key: department.dnumber
3	-> Nested Loop (cost=0.72..2387.60 rows=5351 width=4) (actual time=0.111..12.229 rows=15854 loops=1)
4	-> Merge Join (cost=0.43..1460.52 rows=99 width=8) (actual time=0.105..6.748 rows=150 loops=1)
5	Merge Cond: (department.dnumber = employee_1.dno)
6	-> Index Scan using department_pkey on department (cost=0.14..15.39 rows=150 width=4) (actual time=0.005..0.037 rows=150 loops=1)
7	-> GroupAggregate (cost=0.29..1442.53 rows=99 width=4) (actual time=0.097..6.649 rows=150 loops=1)
8	Group Key: employee_1.dno
9	Filter: (count(*) > 5)
10	-> Index Scan using index66 on employee employee_1 (cost=0.29..1358.83 rows=16000 width=4) (actual time=0.005..4.907 rows=15855 loops=1)
11	-> Index Scan using index66 on employee (cost=0.29..8.82 rows=54 width=4) (actual time=0.002..0.031 rows=106 loops=150)
12	Index Cond: (dno = department.dnumber)
13	Filter: (salary > 40000)
14	Planning Time: 0.900 ms
15	Execution Time: 13.276 ms



- **My query with hash indices only.**

  Optimizing my original query with a hash index optimizes my query because it searches with O(1). So here is the optimized query and its query plan and graph.My performance was enhanced efficiently as we can see also from cost.

  ➔ Cost: 2424.33

  ➔ Execution time: 26.129 ms

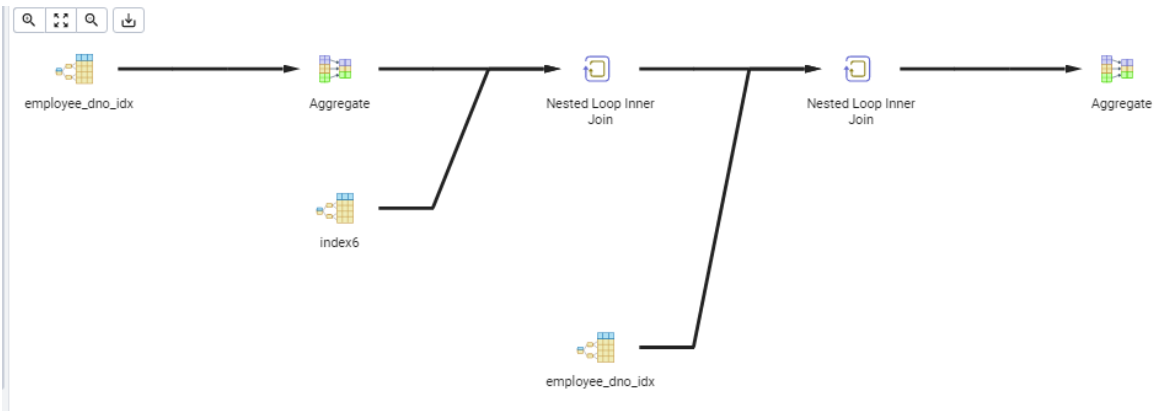
Query | Query History

```
SET enable_seqscan = off;
SET enable_async_append= off;
SET enable_bitmapscan= on;
SET enable_gathermerge=on;
SET enable_hashagg=off;
SET enable_hashjoin=off;
SET enable_incremental_sort=on;
SET enable_indexscan=on;
SET enable_indexonlyscan=off;
SET enable_material=on;
SET enable_memoize=off;
SET enable_mergejoin=on;
SET enable_nestloop =on;
SET enable_parallel_append=off;
SET enable_parallel_hash=on;
SET enable_partition_pruning=on;
SET enable_partitionwise_join =on;
SET enable_partitionwise_aggregate =on;
SET enable_sort=off;
SET enable_tidscan=off;


--create index index6 on department using hash(dnumber);

Explain Analyze
select dnumber, count(*)
from department, employee
where dnumber=dno
and
salary > 40000
and
dno in (
 select dno
 from employee
 group by dno
 having count (*) > 5)
group by dnumber;
```

	QUERY PLAN text
1	GroupAggregate (cost=0.57..2424.33 rows=150 width=12) (actual time=0.305..26.059 rows=150 loops=1)
2	Group Key: department.dnumber
3	-> Nested Loop (cost=0.57..2396.07 rows=5351 width=4) (actual time=0.211..24.171 rows=15854 loops=1)
4	-> Nested Loop (cost=0.29..1468.99 rows=99 width=8) (actual time=0.206..13.203 rows=150 loops=1)
5	-> GroupAggregate (cost=0.29..1442.53 rows=99 width=4) (actual time=0.197..12.785 rows=150 loops=1)
6	Group Key: employee_1.dno
7	Filter: (count(*) > 5)
8	Rows Removed by Filter: 146
9	-> Index Scan using employee_dno_idx on employee employee_1 (cost=0.29..1358.83 rows=16000 width=4) (actual time=0.010..9.059 rows=16000 loops=1)
10	-> Index Scan using index6 on department (cost=0.00..0.26 rows=1 width=4) (actual time=0.002..0.002 rows=1 loops=150)
11	Index Cond: (dnumber = employee_1.dno)
12	-> Index Scan using employee_dno_idx on employee (cost=0.29..8.82 rows=54 width=4) (actual time=0.003..0.064 rows=106 loops=150)
13	Index Cond: (dno = department.dnumber)
14	Filter: (salary > 40000)
15	Planning Time: 3.523 ms
16	Execution Time: 26.129 ms



- **My query with mixed indices only.**

  Optimizing my original query using mixed indices and it does optimize my query . So here is the optimized query and its query plan and graph. Also i used btree and hashing for my mix indices optimization.My performance was enhanced efficiently as we can see also from from cost.

  ➔ Cost: 2424.33
  ➔ Execution time: 19.937 ms

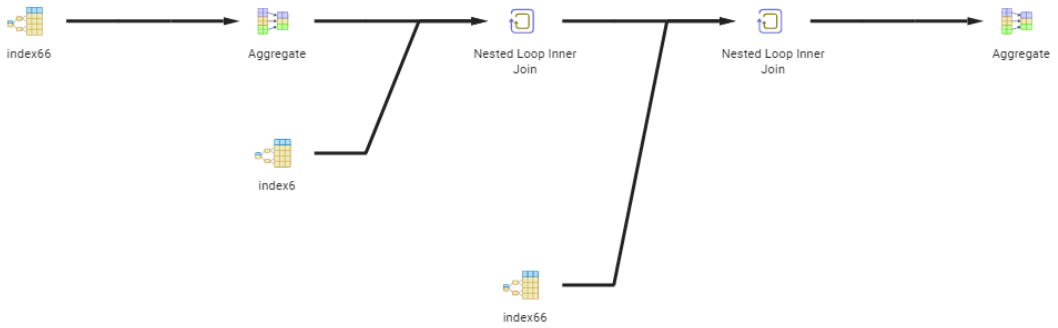
Query | Query History

```
SET enable_async_append= off;
SET enable_bitmapscan= on;
SET enable_gathermerge=on;
SET enable_hashagg=off;
SET enable_hashjoin=off;
SET enable_incremental_sort=on;
SET enable_indexscan=on;
SET enable_indexonlyscan=off;
SET enable_material=on;
SET enable_memoize=off;
SET enable_mergejoin=on;
SET enable_nestloop =on;
SET enable_parallel_append=off;
SET enable_parallel_hash=on;
SET enable_partition_pruning=on;
SET enable_partitionwise_join =on;
SET enable_partitionwise_aggregate =on;
SET enable_sort=off;
SET enable_tidscan=off;

--create index index6 on department using hash(dnumber);
--create index index66 on employee using btree(dno);
--drop index index6
Explain Analyze
select dnumber, count(*)
from department, employee
where dnumber=dno
and
salary > 40000
and
dno in (
 select dno
 from employee
 group by dno
 having count (*) > 5)
group by dnumber;

```

	QUERY PLAN text
1	GroupAggregate (cost=0.57..2424.33 rows=150 width=12) (actual time=0.264..19.877 rows=150 loops=1)
2	Group Key: department.dnumber
3	-> Nested Loop (cost=0.57..2396.07 rows=5351 width=4) (actual time=0.102..18.381 rows=15854 loops=1)
4	-> Nested Loop (cost=0.29..1468.99 rows=99 width=8) (actual time=0.093..10.011 rows=150 loops=1)
5	-> GroupAggregate (cost=0.29..1442.53 rows=99 width=4) (actual time=0.084..9.591 rows=150 loops=1)
6	Group Key: employee_1.dno
7	Filter: (count(*) > 5)
8	Rows Removed by Filter: 146
9	-> Index Scan using index66 on employee employee_1 (cost=0.29..1358.83 rows=16000 width=4) (actual time=0.006..7.172 rows=16000 loops=1)
10	-> Index Scan using index6 on department (cost=0.00..0.26 rows=1 width=4) (actual time=0.002..0.002 rows=1 loops=150)
11	Index Cond: (dnumber = employee_1.dno)
12	-> Index Scan using index66 on employee (cost=0.29..8.82 rows=54 width=4) (actual time=0.003..0.048 rows=106 loops=150)
13	Index Cond: (dno = department.dnumber)
14	Filter: (salary > 40000)
15	Planning Time: 4.076 ms
16	Execution Time: 19.937 ms



- **My query with Brin indices only.**

  Optimizing my original query using brin index optimizes my query because it searches with O(log r) where r is the number of ranges. Also brin index is very efficient only in two scenarios: analytical queries and queries involving narrow-range search on columns with a huge range of values so here we can see that we have ranges in this query . So here is the optimized query and its query plan and graph.

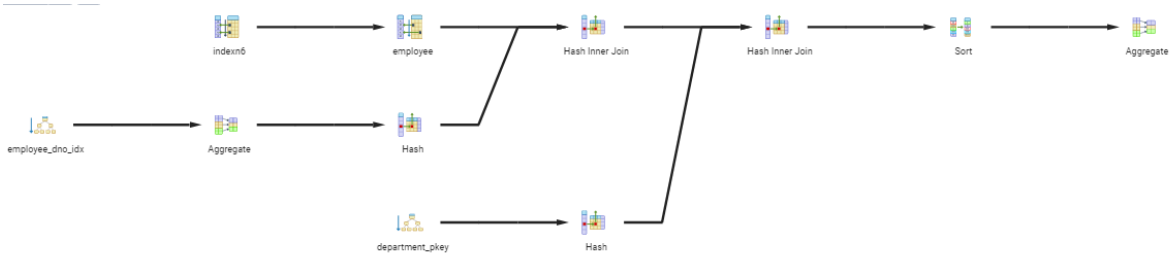
  ➔ Cost: 1320.43

  ➔ Execution time: 13.140 ms

```sql
SET enable_seqscan = off;
SET enable_async_append= off;
SET enable_bitmapscan= on;
SET enable_gathermerge=off;
SET enable_hashagg=off;
SET enable_hashjoin=on;
SET enable_incremental_sort=off;
SET enable_indexscan=on;
SET enable_indexonlyscan=on;
SET enable_material=off;
SET enable_memoize=off;
SET enable_mergejoin=off;
SET enable_nestloop =on;
SET enable_parallel_append=off;
SET enable_parallel_hash=off;
SET enable_partition_pruning=on;
SET enable_partitionwise_join =on;
SET enable_partitionwise_aggregate =on;
SET enable_sort=on;
SET enable_tidscan=on;
--create index indexn6 on employee using brin(dno, salary);
--drop index indexn6
explain analyze
select dnumber, count(*)
from department, employee
where dnumber=dno
and
salary > 40000
and
dno in (
 select dno
 from employee
 group by dno
 having count (*) > 5)
group by dnumber;

```

	QUERY PLAN text
1	GroupAggregate (cost=1278.80..1320.43 rows=150 width=12) (actual time=10.877..12.685 rows=150 loops=1)
2	Group Key: department.dnumber
3	-> Sort (cost=1278.80..1292.18 rows=5351 width=4) (actual time=10.844..11.438 rows=15854 loops=1)
4	Sort Key: department.dnumber
5	Sort Method: quicksort Memory: 1128kB
6	-> Hash Join (cost=427.58..947.42 rows=5351 width=4) (actual time=2.099..8.731 rows=15854 loops=1)
7	Hash Cond: (employee.dno = department.dnumber)
8	-> Hash Join (cost=410.31..915.91 rows=5351 width=8) (actual time=2.048..6.791 rows=15854 loops=1)
9	Hash Cond: (employee.dno = employee_1.dno)
10	-> Bitmap Heap Scan on employee (cost=16.10..479.10 rows=15999 width=4) (actual time=0.018..2.196 rows=16000 loops=1)
11	Recheck Cond: (salary > 40000)
12	Heap Blocks: lossy=263
13	-> Bitmap Index Scan on indexn6 (cost=0.00..12.10 rows=16000 width=0) (actual time=0.014..0.014 rows=2630 loops=1)
14	Index Cond: (salary > 40000)
15	-> Hash (cost=392.98..392.98 rows=99 width=4) (actual time=2.027..2.028 rows=150 loops=1)
16	Buckets: 1024 Batches: 1 Memory Usage: 14kB
17	-> GroupAggregate (cost=0.29..391.99 rows=99 width=4) (actual time=0.021..2.012 rows=150 loops=1)
18	Group Key: employee_1.dno
19	Filter: (count(*) > 5)
20	Rows Removed by Filter: 146
21	-> Index Only Scan using employee_dno_idx on employee employee_1 (cost=0.29..308.29 rows=16000 width=4) (actual time=0.007..0.987 rows=16000 loops=1)
22	Heap Fetches: 0
23	-> Hash (cost=15.39..15.39 rows=150 width=4) (actual time=0.047..0.047 rows=150 loops=1)
24	Buckets: 1024 Batches: 1 Memory Usage: 14kB
25	-> Index Only Scan using department_pkey on department (cost=0.14..15.39 rows=150 width=4) (actual time=0.009..0.031 rows=150 loops=1)
26	Heap Fetches: 150
27	Planning Time: 0.291 ms
28	Execution Time: 13.140 ms



- # Alternative Query
  - **The alternative query to the original one without any index and its query plan and graph.**

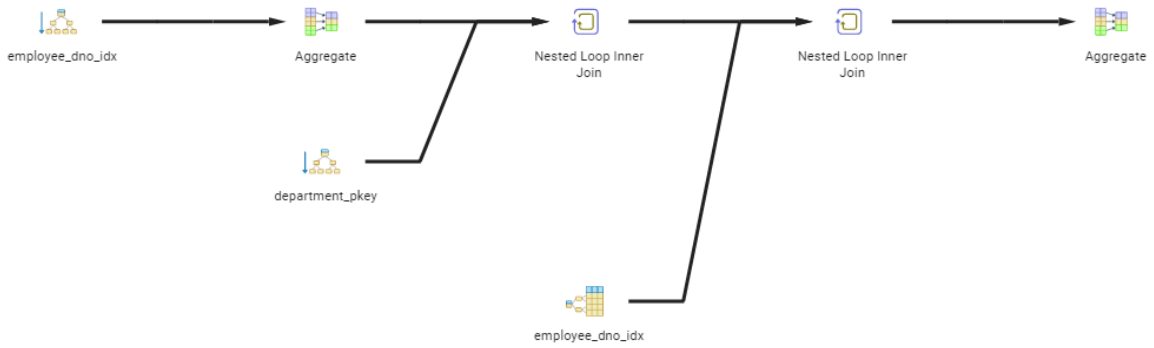
    This alternative query gives same result as the original one and at the same time it reduces cost and so optimizes my query and enhances its performance efficiently.

    ➔ Cost: 1380.22
    ➔ Execution time: 12.240 ms

```
SET enable_seqscan = off;
SET enable_async_append= off;
SET enable_bitmapscan= off;
SET enable_gathermerge=off;
SET enable_hashagg=off;
SET enable_hashjoin=on;
SET enable_incremental_sort=off;
SET enable_indexscan=on;
SET enable_indexonlyscan=on;
SET enable_material=off;
SET enable_memoize=off;
SET enable_mergejoin=off;
SET enable_nestloop =on;
SET enable_parallel_append=off;
SET enable_parallel_hash=off;
SET enable_partition_pruning=on;
SET enable_partitionwise_join =on;
SET enable_partitionwise_aggregate =on;
SET enable_sort=on;
SET enable_tidscan=on;

Explain Analyze
select dnumber, count(*)
from department d inner join employee e on
 d.dnumber=e.dno
and
salary > 40000
and
dno in (
 select dno
 from employee
 group by dno
 having count (*) > 5)
 group by dnumber;
```

	QUERY PLAN text
1	GroupAggregate (cost=0.72..1380.22 rows=150 width=12) (actual time=0.250..12.191 rows=150 loops=1)
2	Group Key: d.dnumber
3	-> Nested Loop (cost=0.72..1351.96 rows=5351 width=4) (actual time=0.158..10.979 rows=15854 loops=1)
4	-> Nested Loop (cost=0.43..424.89 rows=99 width=8) (actual time=0.145..2.772 rows=150 loops=1)
5	-> GroupAggregate (cost=0.29..391.99 rows=99 width=4) (actual time=0.024..2.305 rows=150 loops=1)
6	Group Key: employee.dno
7	Filter: (count(*) > 5)
8	Rows Removed by Filter: 146
9	-> Index Only Scan using employee_dno_idx on employee (cost=0.29..308.29 rows=16000 width=4) (actual time=0.008..1.149 rows=16000 loops=1)
10	Heap Fetches: 0
11	-> Index Only Scan using department_pkey on department d (cost=0.14..0.32 rows=1 width=4) (actual time=0.002..0.002 rows=1 loops=150)
12	Index Cond: (dnumber = employee.dno)
13	Heap Fetches: 150
14	-> Index Scan using employee_dno_idx on employee e (cost=0.29..8.82 rows=54 width=4) (actual time=0.002..0.048 rows=106 loops=150)
15	Index Cond: (dno = d.dnumber)
16	Filter: (salary > 40000)
17	Planning Time: 1.188 ms
18	Execution Time: 12.240 ms



- **My alternative query with B+ trees indices only.**

  Optimizing my alternative query using Btree optimizes my query because it searches with O(log n). Performance was enhanced efficiently and my cost was so much better. So here is the optimized query and its query plan and graph.

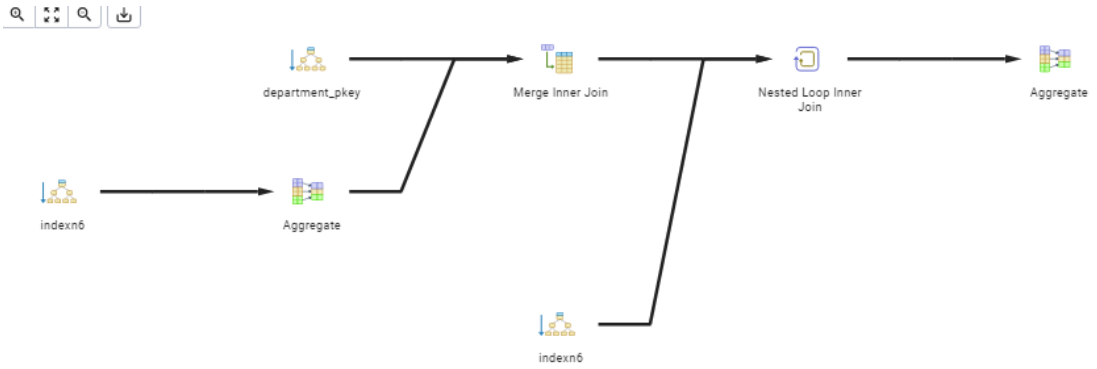
  ➔ Cost: 665.07

  ➔ Execution time: 5.493 ms

Query Query History

```
1  SET enable_seqscan = off;
2  SET enable_async_append= off;
3  SET enable_bitmapscan= off;
4  SET enable_gathermerge=off;
5  SET enable_hashagg=off;
6  SET enable_hashjoin=on;
7  SET enable_incremental_sort=off;
8  SET enable_indexscan=on;
9  SET enable_indexonlyscan=on;
10 SET enable_material=off;
11 SET enable_memoize=off;
12 SET enable_mergejoin=on;
13 SET enable_nestloop =on;
14 SET enable_parallel_append=off;
15 SET enable_parallel_hash=on;
16 SET enable_partition_pruning=on;
17 SET enable_partitionwise_join =off;
18 SET enable_partitionwise_aggregate =off;
19 SET enable_sort=on;
20 SET enable_tidscan=on;
21
22 --create index indexn6 on employee using btree (dno,salary);
23 --drop index indexn6
24 Explain Analyze
25 select dnumber, count(*)
26 from department d inner join employee e on
27  d.dnumber=e.dno
28 and
29 salary > 40000
30 and
31 dno in (
32  select dno
33  from employee
34  group by dno
35  having count (*) > 5)
36  group by dnumber;
```

	QUERY PLAN text
1	GroupAggregate (cost=0.72..665.07 rows=150 width=12) (actual time=0.083..5.425 rows=150 loops=1)
2	Group Key: d.dnumber
3	-> Nested Loop (cost=0.72..636.82 rows=5351 width=4) (actual time=0.048..4.393 rows=15854 loops=1)
4	-> Merge Join (cost=0.43..405.98 rows=99 width=8) (actual time=0.034..2.309 rows=150 loops=1)
5	Merge Cond: (d.dnumber = employee.dno)
6	-> Index Only Scan using department_pkey on department d (cost=0.14..15.39 rows=150 width=4) (actual time=0.006..0.045 rows=150 loops=1)
7	Heap Fetches: 150
8	-> GroupAggregate (cost=0.29..387.99 rows=99 width=4) (actual time=0.026..2.222 rows=150 loops=1)
9	Group Key: employee.dno
10	Filter: (count(*) > 5)
11	-> Index Only Scan using indexn6 on employee (cost=0.29..304.29 rows=16000 width=4) (actual time=0.009..1.241 rows=15855 loops=1)
12	Heap Fetches: 0
13	-> Index Only Scan using indexn6 on employee e (cost=0.29..1.79 rows=54 width=4) (actual time=0.002..0.008 rows=106 loops=150)
14	Index Cond: ((dno = d.dnumber) AND (salary > 40000))
15	Heap Fetches: 0
16	Planning Time: 2.943 ms
17	Execution Time: 5.493 ms



- **My alternative query with hash indices only**.

  Optimizing my alternative query with hashing optimizes my query because it searches with O(1). Hashing enhanced my performance efficiently here and lowered the cost. So here is the optimized query and its query plan and graph.

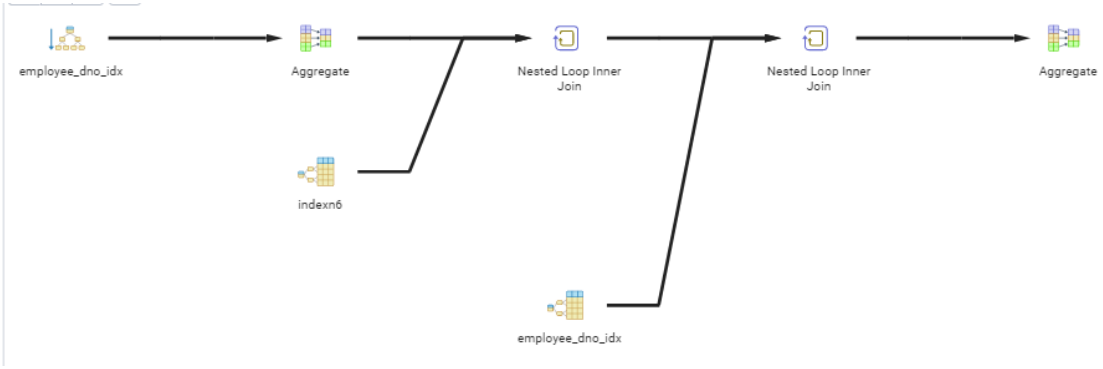
  ➔ Cost: 1373.79

  ➔ Execution time : 10.963

Query | Query History

```
1  SET enable_seqscan = off;
2  SET enable_async_append= off;
3  SET enable_bitmapscan= off;
4  SET enable_gathermerge=off;
5  SET enable_hashagg=off;
6  SET enable_hashjoin=on;
7  SET enable_incremental_sort=off;
8  SET enable_indexscan=on;
9  SET enable_indexonlyscan=on;
10 SET enable_material=off;
11 SET enable_memoize=off;
12 SET enable_mergejoin=off;
13 SET enable_nestloop =on;
14 SET enable_parallel_append=off;
15 SET enable_parallel_hash=off;
16 SET enable_partition_pruning=on;
17 SET enable_partitionwise_join =on;
18 SET enable_partitionwise_aggregate =on;
19 SET enable_sort=on;
20 SET enable_tidscan=on;
21
22 --create index indexn6 on department using hash (dnumber);
23 --drop index indexn6
24 Explain Analyze
25 select dnumber, count(*)
26 from department d inner join employee e on
27  d.dnumber=e.dno
28 and
29 salary > 40000
30 and
31 dno in (
32  select dno
33  from employee
34  group by dno
35  having count (*) > 5)
36  group by dnumber;
37
```

	QUERY PLAN text
1	GroupAggregate (cost=0.57..1373.79 rows=150 width=12) (actual time=0.634..10.915 rows=150 loops=1)
2	Group Key: d.dnumber
3	-> Nested Loop (cost=0.57..1345.53 rows=5351 width=4) (actual time=0.484..9.858 rows=15854 loops=1)
4	-> Nested Loop (cost=0.29..418.45 rows=99 width=8) (actual time=0.475..2.624 rows=150 loops=1)
5	-> GroupAggregate (cost=0.29..391.99 rows=99 width=4) (actual time=0.438..2.366 rows=150 loops=1)
6	Group Key: employee.dno
7	Filter: (count(*) > 5)
8	Rows Removed by Filter: 146
9	-> Index Only Scan using employee_dno_idx on employee (cost=0.29..308.29 rows=16000 width=4) (actual time=0.418..1.394 rows=16000 loops=1)
10	Heap Fetches: 0
11	-> Index Scan using indexn6 on department d (cost=0.00..0.26 rows=1 width=4) (actual time=0.001..0.001 rows=1 loops=150)
12	Index Cond: (dnumber = employee.dno)
13	-> Index Scan using employee_dno_idx on employee e (cost=0.29..8.82 rows=54 width=4) (actual time=0.001..0.042 rows=106 loops=150)
14	Index Cond: (dno = d.dnumber)
15	Filter: (salary > 40000)
16	Planning Time: 1.901 ms
17	Execution Time: 10.963 ms



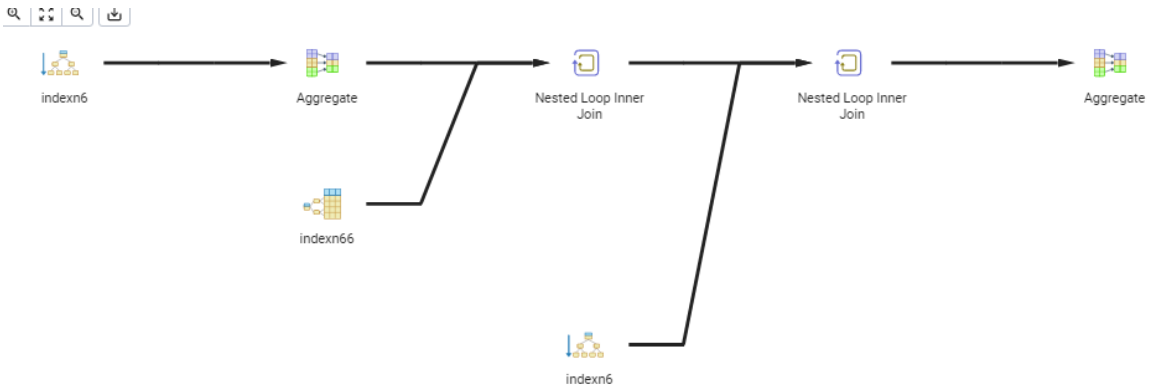
- **My alternative query with mixed indices only.**

  Optimizing my alternative query using mixed indices and it does optimize my query . So here is the optimized query and its query plan and graph. Also i used btree and hashing for my mix indices optimization and it lowered my cost .My performance was enhanced efficiently as we can see also from cost.

  ➔ Cost: 673.54
  ➔ Execution time: 7.060 ms

```sql
SET enable_seqscan = off;
SET enable_async_append= off;
SET enable_bitmapscan= off;
SET enable_gathermerge=off;
SET enable_hashagg=off;
SET enable_hashjoin=on;
SET enable_incremental_sort=off;
SET enable_indexscan=on;
SET enable_indexonlyscan=on;
SET enable_material=off;
SET enable_memoize=off;
SET enable_mergejoin=off;
SET enable_nestloop =on;
SET enable_parallel_append=off;
SET enable_parallel_hash=off;
SET enable_partition_pruning=on;
SET enable_partitionwise_join =on;
SET enable_partitionwise_aggregate =on;
SET enable_sort=on;
SET enable_tidscan=on;
--create index indexn6 on employee using btree(dno, salary);
--create index indexn66 on department using hash(dnumber);
--drop index indexn6
Explain Analyze
select dnumber, count(*)
from department d inner join employee e on
 d.dnumber=e.dno
and
salary > 40000
and
dno in (
 select dno
 from employee
 group by dno
 having count (*) > 5)
 group by dnumber;
```

	QUERY PLAN text
1	GroupAggregate (cost=0.57..673.54 rows=150 width=12) (actual time=0.904..6.975 rows=150 loops=1)
2	Group Key: d.dnumber
3	-> Nested Loop (cost=0.57..645.29 rows=5351 width=4) (actual time=0.867..5.839 rows=15854 loops=1)
4	-> Nested Loop (cost=0.29..414.45 rows=99 width=8) (actual time=0.858..3.627 rows=150 loops=1)
5	-> GroupAggregate (cost=0.29..387.99 rows=99 width=4) (actual time=0.842..3.424 rows=150 loops=1)
6	Group Key: employee.dno
7	Filter: (count(*) > 5)
8	Rows Removed by Filter: 146
9	-> Index Only Scan using indexn6 on employee (cost=0.29..304.29 rows=16000 width=4) (actual time=0.811..2.397 rows=16000 loops=1)
10	Heap Fetches: 0
11	-> Index Scan using indexn66 on department d (cost=0.00..0.26 rows=1 width=4) (actual time=0.001..0.001 rows=1 loops=150)
12	Index Cond: (dnumber = employee.dno)
13	-> Index Only Scan using indexn6 on employee e (cost=0.29..1.79 rows=54 width=4) (actual time=0.001..0.009 rows=106 loops=150)
14	Index Cond: ((dno = d.dnumber) AND (salary > 40000))
15	Heap Fetches: 0
16	Planning Time: 3.310 ms
17	Execution Time: 7.060 ms



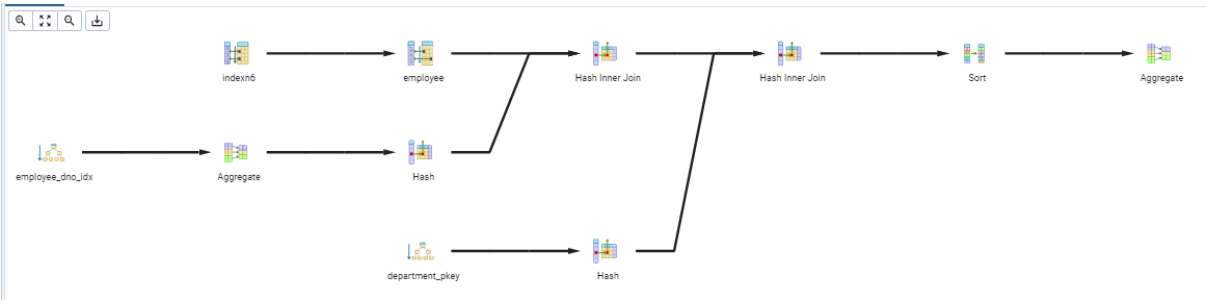
- **My alternative query with Brin indices only**.

  Optimizing my alternative query using brin index optimizes my query because it searches with O(log r) where r is the number of ranges. Also brin index is very efficient only in two scenarios: analytical queries and queries involving narrow-range search on column with a huge range of values so here we can see that we have ranges in this query .Hence, using brin here helps me enhance my query performance efficiently. So here is the optimized query and its query plan and graph.

  ➔ Cost: 1320.43
  ➔ Execution time: 16.902 ms

```
SET enable_seqscan = off;
SET enable_async_append= off;
SET enable_bitmapscan= on;
SET enable_gathermerge=off;
SET enable_hashagg=off;
SET enable_hashjoin=on;
SET enable_incremental_sort=off;
SET enable_indexscan=on;
SET enable_indexonlyscan=on;
SET enable_material=off;
SET enable_memoize=off;
SET enable_mergejoin=off;
SET enable_nestloop =on;
SET enable_parallel_append=off;
SET enable_parallel_hash=off;
SET enable_partition_pruning=on;
SET enable_partitionwise_join =on;
SET enable_partitionwise_aggregate =on;
SET enable_sort=on;
SET enable_tidscan=on;
--create index indexn6 on employee using brin(dno, salary);
--drop index indexn6
Explain Analyze
select dnumber, count(*)
from department d inner join employee e on
 d.dnumber=e.dno
and
salary > 40000
and
dno in (
 select dno
 from employee
 group by dno
 having count (*) > 5)
 group by dnumber;
```

	QUERY PLAN text
1	GroupAggregate (cost=1278.80..1320.43 rows=150 width=12) (actual time=14.642..16.838 rows=150 loops=1)
2	Group Key: d.dnumber
3	-> Sort (cost=1278.80..1292.18 rows=5351 width=4) (actual time=14.602..15.288 rows=15854 loops=1)
4	Sort Key: d.dnumber
5	Sort Method: quicksort Memory: 1128kB
6	-> Hash Join (cost=427.58..947.42 rows=5351 width=4) (actual time=2.266..11.761 rows=15854 loops=1)
7	Hash Cond: (e.dno = d.dnumber)
8	-> Hash Join (cost=410.31..915.91 rows=5351 width=8) (actual time=2.178..8.899 rows=15854 loops=1)
9	Hash Cond: (e.dno = employee.dno)
10	-> Bitmap Heap Scan on employee e (cost=16.10..479.10 rows=15999 width=4) (actual time=0.116..3.349 rows=16000 loops=1)
11	Recheck Cond: (salary > 40000)
12	Heap Blocks: lossy=263
13	-> Bitmap Index Scan on indexn6 (cost=0.00..12.10 rows=16000 width=0) (actual time=0.105..0.106 rows=2630 loops=1)
14	Index Cond: (salary > 40000)
15	-> Hash (cost=392.98..392.98 rows=99 width=4) (actual time=2.056..2.057 rows=150 loops=1)
16	Buckets: 1024 Batches: 1 Memory Usage: 14kB
17	-> GroupAggregate (cost=0.29..391.99 rows=99 width=4) (actual time=0.024..2.022 rows=150 loops=1)
18	Group Key: employee.dno
19	Filter: (count(*) > 5)
20	Rows Removed by Filter: 146
21	-> Index Only Scan using employee_dno_idx on employee (cost=0.29..308.29 rows=16000 width=4) (actual time=0.007..1.022 rows=16000 loops=1)
22	Heap Fetches: 0
23	-> Hash (cost=15.39..15.39 rows=150 width=4) (actual time=0.083..0.083 rows=150 loops=1)
24	Buckets: 1024 Batches: 1 Memory Usage: 14kB
25	-> Index Only Scan using department_pkey on department d (cost=0.14..15.39 rows=150 width=4) (actual time=0.011..0.060 rows=150 loops=1)
26	Heap Fetches: 150
27	Planning Time: 0.372 ms
28	Execution Time: 16.902 ms



Schema 3

Query 7

1)Query (7) without an index:

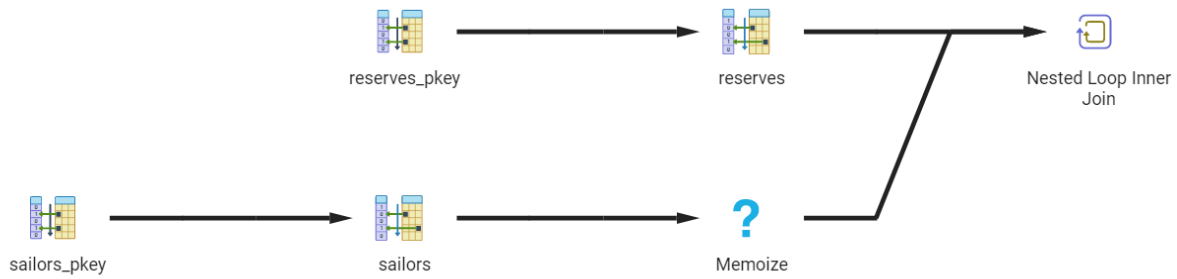
Cost = 5425.45 rows

Execution time = 6.609 msec

```
SET enable_seqscan = off;
SET enable_async_append= ON;
SET enable_bitmapscan= on;
SET enable_gathermerge=ON;
SET enable_hashagg=ON;
SET enable_hashjoin=ON;
SET enable_incremental_sort=ON;
SET enable_indexscan=off;
SET enable_indexonlyscan=off;
SET enable_material=ON;
SET enable_memoize=ON;
SET enable_mergejoin=ON;
SET enable_nestloop =ON;
SET enable_parallel_append=ON;
SET enable_parallel_hash=ON;
SET enable_partition_pruning=ON;
SET enable_partitionwise_join =ON;
SET enable_partitionwise_aggregate =ON;
SET enable_sort=ON;
SET enable_tidscan=ON;
```

```
Explain Analyze
select s.sname
from sailors s
where
s.sid in( select r.sid
from reserves r
where r.bid = 103 );
```

	QUERY PLAN text
1	Nested Loop (cost=707.56..5425.45 rows=1007 width=21) (actual time=0.500..6.489 rows=1008 loops=1)
2	-> Bitmap Heap Scan on reserves r (cost=707.04..909.63 rows=1007 width=4) (actual time=0.485..0.577 rows=1008 loops=...
3	Recheck Cond: (bid = 103)
4	Heap Blocks: exact=19
5	-> Bitmap Index Scan on reserves_pkey (cost=0.00..706.79 rows=1007 width=0) (actual time=0.479..0.479 rows=1008 loop...
6	Index Cond: (bid = 103)
7	-> Memoize (cost=0.52..4.53 rows=1 width=25) (actual time=0.006..0.006 rows=1 loops=1008)
8	Cache Key: r.sid
9	Cache Mode: logical
10	Hits: 0 Misses: 1008 Evictions: 0 Overflows: 0 Memory Usage: 126kB
11	-> Bitmap Heap Scan on sailors s (cost=0.51..4.52 rows=1 width=25) (actual time=0.002..0.002 rows=1 loops=1008)
12	Recheck Cond: (sid = r.sid)
13	Heap Blocks: exact=1008
14	-> Bitmap Index Scan on sailors_pkey (cost=0.00..0.51 rows=1 width=0) (actual time=0.001..0.001 rows=1 loops=1008)
15	Index Cond: (sid = r.sid)
16	Planning Time: 0.621 ms
17	Execution Time: 6.609 ms



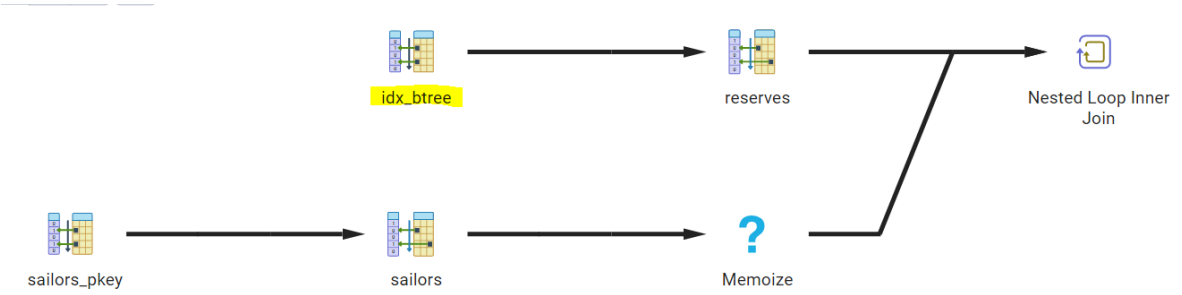
2)Query (7) with B+ trees indices only:

Cost = 4734.51 rows

Execution time = 5.609 msec

```
Create Index idx_btree on reserves using btree(bid)
```

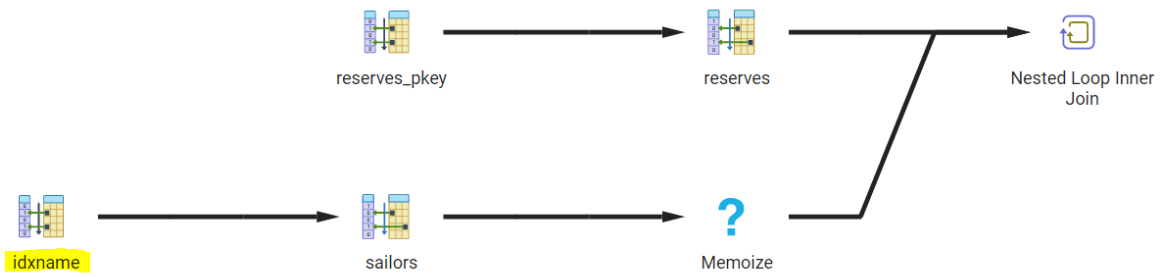
QUERY PLAN text
Nested Loop (cost=16.61..4734.51 rows=1007 width=21) (actual time=0.076..5.456 rows=1008 loops=1)
-> Bitmap Heap Scan on reserves r (cost=16.09..218.68 rows=1007 width=4) (actual time=0.061..0.141 rows=1008 loops=1)
Recheck Cond: (bid = 103)
Heap Blocks: exact=19
-> Bitmap Index Scan on idx_btree (cost=0.00..15.84 rows=1007 width=0) (actual time=0.054..0.054 rows=1008 loops=1)
Index Cond: (bid = 103)
-> Memoize (cost=0.52..4.53 rows=1 width=25) (actual time=0.005..0.005 rows=1 loops=1008)
Cache Key: r.sid
Cache Mode: logical
Hits: 0 Misses: 1008 Evictions: 0 Overflows: 0 Memory Usage: 126kB
-> Bitmap Heap Scan on sailors s (cost=0.51..4.52 rows=1 width=25) (actual time=0.001..0.001 rows=1 loops=1008)
Recheck Cond: (sid = r.sid)
Heap Blocks: exact=1008
-> Bitmap Index Scan on sailors_pkey (cost=0.00..0.51 rows=1 width=0) (actual time=0.001..0.001 rows=1 loops=1008)
Index Cond: (sid = r.sid)
Planning Time: 0.704 ms
Execution Time: 5.609 ms



3) Query (7) with hash indices only,
Cost = 5187.30 rows
Execution time = 9.828 msec

```
CREATE INDEX idxName ON sailors USING hash(sid)
```

QUERY PLAN text
Nested Loop (cost=707.32..5187.30 rows=1007 width=21) (actual time=0.445..9.666 rows=1008 loops=1)
-> Bitmap Heap Scan on reserves r (cost=707.04..909.63 rows=1007 width=4) (actual time=0.428..0.566 rows=1008 loops=1)
Recheck Cond: (bid = 103)
Heap Blocks: exact=19
-> Bitmap Index Scan on reserves_pkey (cost=0.00..706.79 rows=1007 width=0) (actual time=0.423..0.423 rows=1008 loops=1)
Index Cond: (bid = 103)
-> Memoize (cost=0.28..4.29 rows=1 width=25) (actual time=0.009..0.009 rows=1 loops=1008)
Cache Key: r.sid
Cache Mode: logical
Hits: 0 Misses: 1008 Evictions: 0 Overflows: 0 Memory Usage: 126kB
-> Bitmap Heap Scan on sailors s (cost=0.27..4.28 rows=1 width=25) (actual time=0.003..0.003 rows=1 loops=1008)
Recheck Cond: (sid = r.sid)
Heap Blocks: exact=1008
-> Bitmap Index Scan on idxname (cost=0.00..0.27 rows=1 width=0) (actual time=0.002..0.002 rows=1 loops=1008)
Index Cond: (sid = r.sid)
Planning Time: 0.410 ms
Execution Time: 9.828 ms



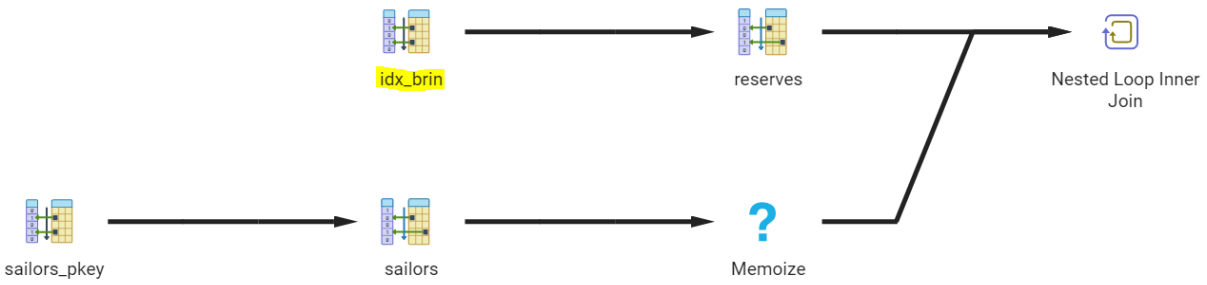
4) Query (7) with BRIN indices only,

Cost = 5155.64 rows

Execution time = 14.569 msec

```
create index idx_brin on reserves using brin(bid);
```

QUERY PLAN text
Nested Loop (cost=12.84..5155.64 rows=1007 width=21) (actual time=0.062..14.356 rows=1008 loops=1)
-> Bitmap Heap Scan on reserves r (cost=12.32..639.82 rows=1007 width=4) (actual time=0.052..4.303 rows=1008 loops=1)
Recheck Cond: (bid = 103)
Rows Removed by Index Recheck: 33992
Heap Blocks: lossy=190
-> Bitmap Index Scan on idx_brin (cost=0.00..12.06 rows=35000 width=0) (actual time=0.038..0.038 rows=1900 loops=1)
Index Cond: (bid = 103)
-> Memoize (cost=0.52..4.53 rows=1 width=25) (actual time=0.010..0.010 rows=1 loops=1008)
Cache Key: r.sid
Cache Mode: logical
Hits: 0 Misses: 1008 Evictions: 0 Overflows: 0 Memory Usage: 126kB
-> Bitmap Heap Scan on sailors s (cost=0.51..4.52 rows=1 width=25) (actual time=0.003..0.003 rows=1 loops=1008)
Recheck Cond: (sid = r.sid)
Heap Blocks: exact=1008
-> Bitmap Index Scan on sailors_pkey (cost=0.00..0.51 rows=1 width=0) (actual time=0.002..0.002 rows=1 loops=1008)
Index Cond: (sid = r.sid)
Planning Time: 0.518 ms
Execution Time: 14.569 ms



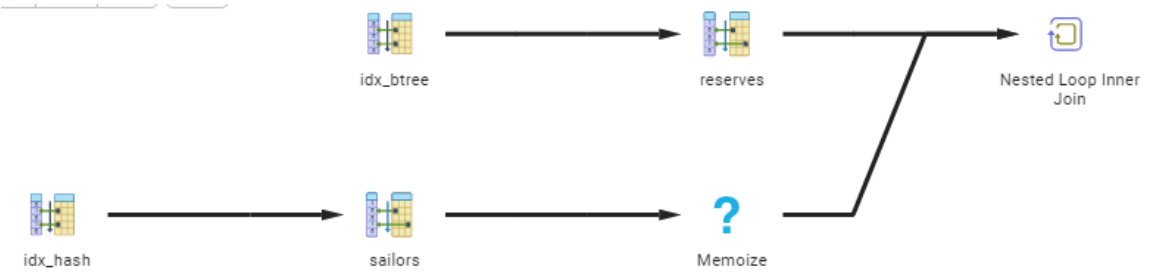
5) Query (7) with mixed indices.

Cost = 4496.35 rows

Execution time = 5.395 msec

```
CREATE INDEX idx_btree ON reserves USING btree(bid);
Create Index idx_hash on sailors using hash(sid);
```

QUERY PLAN text
Nested Loop (cost=16.37..4496.35 rows=1007 width=21) (actual time=0.060..5.268 rows=1008 loops=1)
-> Bitmap Heap Scan on reserves r (cost=16.09..218.68 rows=1007 width=4) (actual time=0.044..0.153 rows=1008 loop…
Recheck Cond: (bid = 103)
Heap Blocks: exact=19
-> Bitmap Index Scan on idx_btree (cost=0.00..15.84 rows=1007 width=0) (actual time=0.039..0.039 rows=1008 loops=1)
Index Cond: (bid = 103)
-> Memoize (cost=0.28..4.29 rows=1 width=25) (actual time=0.005..0.005 rows=1 loops=1008)
Cache Key: r.sid
Cache Mode: logical
Hits: 0 Misses: 1008 Evictions: 0 Overflows: 0 Memory Usage: 126kB
-> Bitmap Heap Scan on sailors s (cost=0.27..4.28 rows=1 width=25) (actual time=0.001..0.001 rows=1 loops=1008)
Recheck Cond: (sid = r.sid)
Heap Blocks: exact=1008
-> Bitmap Index Scan on idx_hash (cost=0.00..0.27 rows=1 width=0) (actual time=0.001..0.001 rows=1 loops=1008)
Index Cond: (sid = r.sid)
Planning Time: 0.671 ms
Execution Time: 5.395 ms



Optimized query (7):

Cost = 1876 rows

Execution time = 2.116 msec

```
29
30  --Optimized Query
31  Explain Analyze
32  select s.sname
33  from reserves r inner join sailors s
34  on r.sid=s.sid where r.bid=103;
35
```

Data output | Messages | Explain | Notifications

	QUERY PLAN text
1	Nested Loop (cost=0.58..1876.00 rows=1007 width=21) (actual time=0.045..2.064 rows=1008 loops=1)
2	-> Index Only Scan using reserves_pkey on reserves r (cost=0.29..716.86 rows=1007 width=4) (actual time=0.032..0.730 rows=1008 loops=1)
3	Index Cond: (bid = 103)
4	Heap Fetches: 0
5	-> Index Scan using sailors_pkey on sailors s (cost=0.29..1.15 rows=1 width=25) (actual time=0.001..0.001 rows=1 loops=1008)
6	Index Cond: (sid = r.sid)
7	Planning Time: 0.212 ms
8	Execution Time: 2.116 ms

2)Optimized query (7) after using Btree index :

Cost = 1377.82 rows

Execution time = 1.866 msec

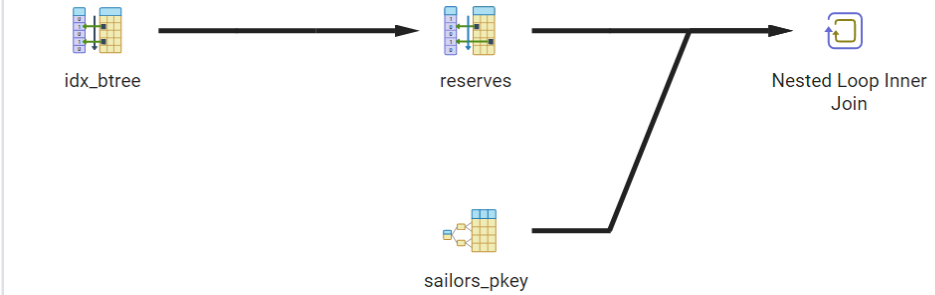
```
39
40  Create Index idx_btree on reserves using btree (bid);
```

Data output | Messages | Explain | Notifications

	QUERY PLAN text
1	Nested Loop (cost=16.38..1377.82 rows=1007 width=21) (actual time=0.103..1.802 rows=1008 loops=1)
2	-> Bitmap Heap Scan on reserves r (cost=16.09..218.68 rows=1007 width=4) (actual time=0.090..0.190 rows=1008 loops=1)
3	Recheck Cond: (bid = 103)
4	Heap Blocks: exact=19
5	-> Bitmap Index Scan on idx_btree (cost=0.00..15.84 rows=1007 width=0) (actual time=0.084..0.084 rows=1008 loops=1)
6	Index Cond: (bid = 103)
7	-> Index Scan using sailors_pkey on sailors s (cost=0.29..1.15 rows=1 width=25) (actual time=0.001..0.001 rows=1 loops=1008)
8	Index Cond: (sid = r.sid)
9	Planning Time: 0.638 ms
10	Execution Time: 1.866 ms

Data output | Messages | Explain | Notifications

Graphical | Analysis | Statistics



3) Optimized query (7) after using hash index :

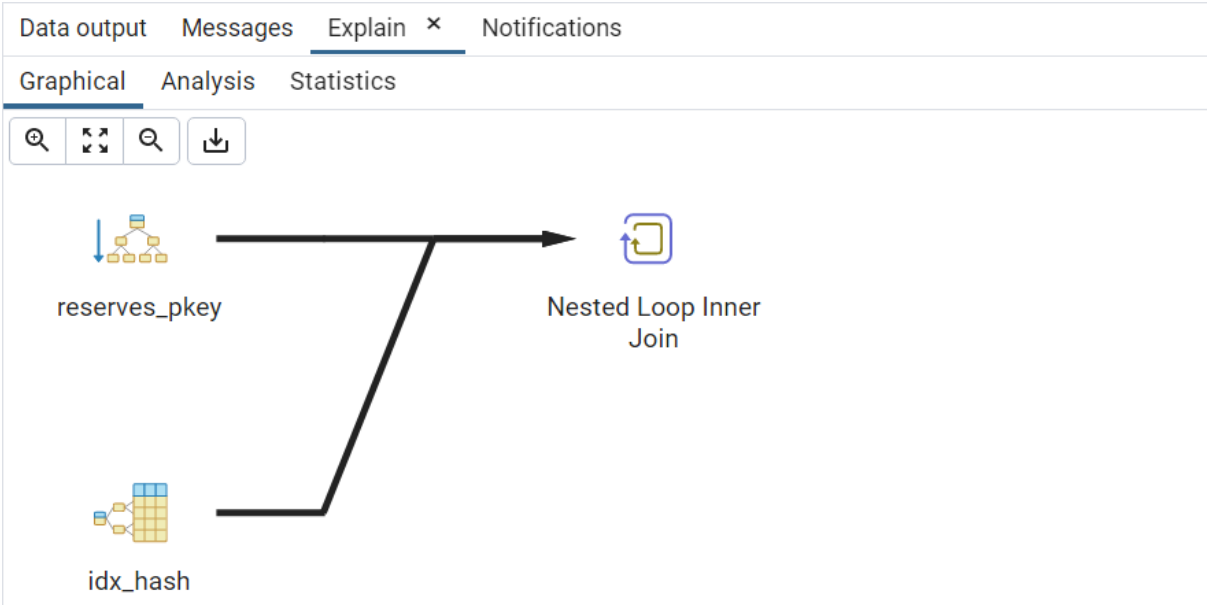
Cost = 16348 rows

Execution time = 2.144 msec

```
36
37  CREATE INDEX idx_hash ON sailors USING hash (sid);
```

Data output | Messages | Explain | Notifications

	QUERY PLAN text
1	Nested Loop (cost=0.29..1634.48 rows=1007 width=21) (actual time=0.029..2.094 rows=1008 loops=1)
2	-> Index Only Scan using reserves_pkey on reserves r (cost=0.29..716.86 rows=1007 width=4) (actual time=0.018..0.880 rows=1008 loops=1)
3	Index Cond: (bid = 103)
4	Heap Fetches: 0
5	-> Index Scan using idx_hash on sailors s (cost=0.00..0.91 rows=1 width=25) (actual time=0.001..0.001 rows=1 loops=1008)
6	Index Cond: (sid = r.sid)
7	Planning Time: 0.595 ms
8	Execution Time: 2.144 ms



4) Optimized Query (7) after brin index :
Cost = 1798.95 rows
Execution time=6.999ms

```
--Optimized Query
Explain Analyze
select s.sname
from reserves r inner join sailors s
on r.sid=s.sid where r.bid=103;

CREATE INDEX idx_brin ON reserves USING brin (bid);
```

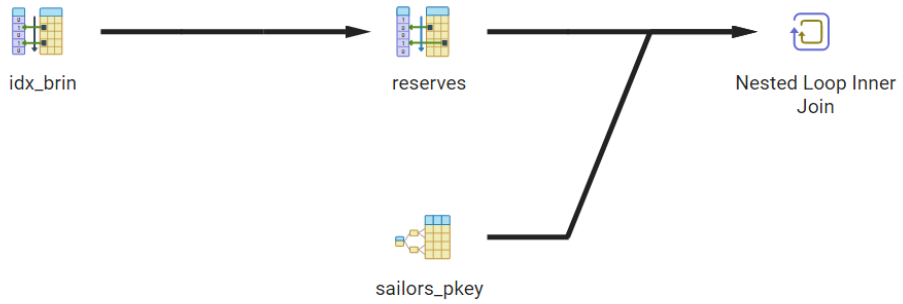
Data output | Messages | Explain | Notifications

	QUERY PLAN text
1	Nested Loop (cost=12.60..1798.95 rows=1007 width=21) (actual time=0.049..6.011 rows=1008 loops=1)
2	-> Bitmap Heap Scan on reserves r (cost=12.32..639.82 rows=1007 width=4) (actual time=0.037..3.925 rows=1008 loops=1)
3	Recheck Cond: (bid = 103)
4	Rows Removed by Index Recheck: 33992
5	Heap Blocks: lossy=190
6	-> Bitmap Index Scan on idx_brin (cost=0.00..12.06 rows=35000 width=0) (actual time=0.018..0.019 rows=1900 loops=1)
7	Index Cond: (bid = 103)
8	-> Index Scan using sailors_pkey on sailors s (cost=0.29..1.15 rows=1 width=25) (actual time=0.002..0.002 rows=1 loops=1008)
9	Index Cond: (sid = r.sid)
10	Planning Time: 0.186 ms

```
31  --Optimized Query
32  Explain Analyze
33  select s.sname
34  from reserves r inner join sailors s
35  on r.sid=s.sid where r.bid=103;
36
37  CREATE INDEX idx_brin ON reserves USING brin (bid);
```

Data output Messages Explain × Notifications

Graphical Analysis Statistics



5) Optimized Query (7) after using mixed indices :
Cost = 1136.30 rows
Execution time = 1.031 msec

```
CREATE INDEX idx_hash ON sailors USING hash (sid);
Create Index idx_btree on reserves using btree (bid);
```

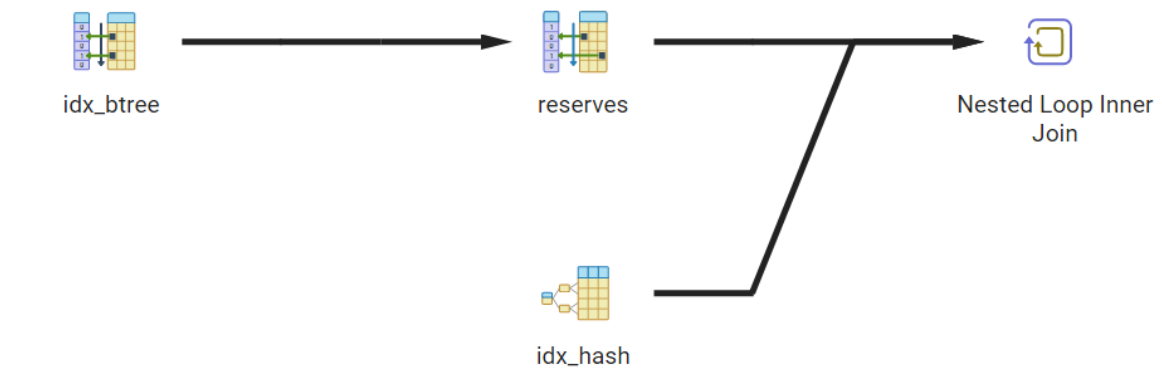
output Messages Explain × Notifications

QUERY PLAN text
Nested Loop (cost=16.09..1136.30 rows=1007 width=21) (actual time=0.034..0.983 rows=1008 loops=1)
-> Bitmap Heap Scan on reserves r (cost=16.09..218.68 rows=1007 width=4) (actual time=0.026..0.095 rows=1008 loops=1)
Recheck Cond: (bid = 103)
Heap Blocks: exact=19
-> Bitmap Index Scan on idx_btree (cost=0.00..15.84 rows=1007 width=0) (actual time=0.021..0.021 rows=1008 loops=1)
Index Cond: (bid = 103)
-> Index Scan using idx_hash on sailors s (cost=0.00..0.91 rows=1 width=25) (actual time=0.001..0.001 rows=1 loops=1008)
Index Cond: (sid = r.sid)
Planning Time: 0.440 ms
Execution Time: 1.031 ms

```
37  CREATE INDEX idx_hash ON sailors USING hash (sid);
38  Create Index idx_btree on reserves using btree (bid);
```

Data output | Messages | Explain | Notifications

Graphical | Analysis | Statistics



Query 8

Query (8) before optimization :

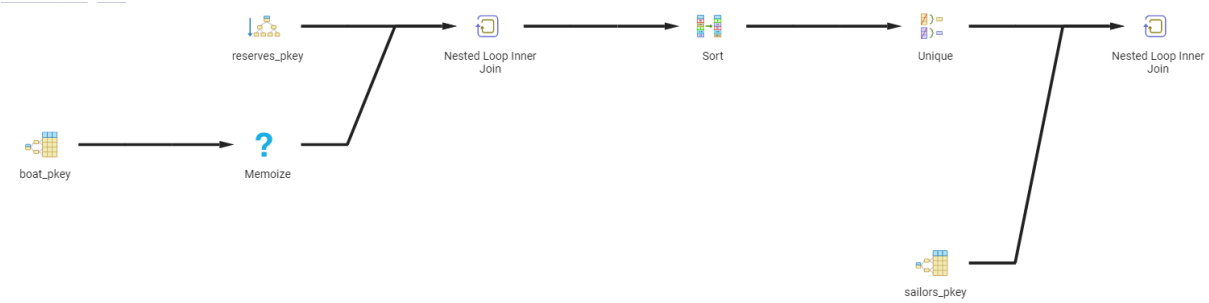
Cost = 10352.69 rows

Execution time = 35.257 msec

```
SET enable_seqscan = OFF;
SET enable_async_append= on;
SET enable_bitmapscan= on;
SET enable_gathermerge=on;
SET enable_hashagg=off;
SET enable_hashjoin=off;
SET enable_incremental_sort=on;
SET enable_indexscan=on;
SET enable_indexonlyscan=On;
SET enable_material=on;
SET enable_memoize=on;
SET enable_mergejoin=off;
SET enable_nestloop =on;
```

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

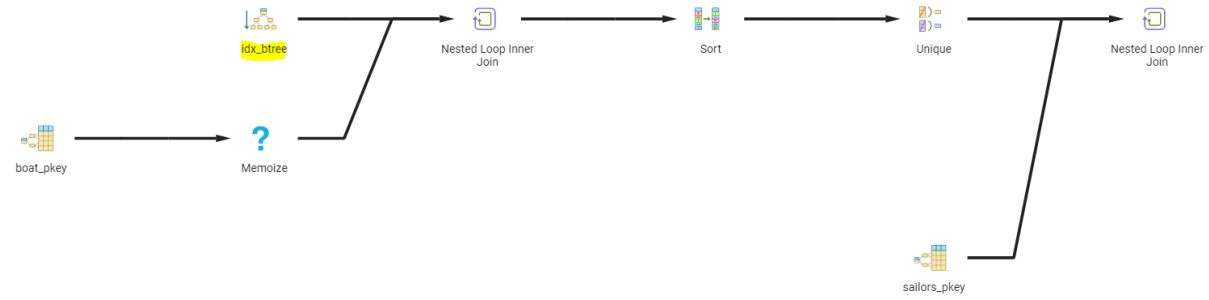
QUERY PLAN text
Nested Loop (cost=3989.38..10352.69 rows=17488 width=21) (actual time=18.003..34.626 rows=10608 loops=1)
-> Unique (cost=3989.09..4076.53 rows=17488 width=4) (actual time=17.970..20.547 rows=10608 loops=1)
-> Sort (cost=3989.09..4032.81 rows=17488 width=4) (actual time=17.969..18.744 rows=18480 loops=1)
Sort Key: r.sid
Sort Method: quicksort Memory: 1635kB
-> Nested Loop (cost=0.58..2756.70 rows=17488 width=4) (actual time=0.035..15.586 rows=18480 loops=1)
-> Index Only Scan using reserves_pkey on reserves r (cost=0.29..969.29 rows=35000 width=8) (actual time=0.017..3.044 rows=35000 loops=1)
Heap Fetches: 0
-> Memoize (cost=0.29..0.31 rows=1 width=4) (actual time=0.000..0.000 rows=1 loops=35000)
Cache Key: r.bid
Cache Mode: logical
Hits: 32000 Misses: 3000 Evictions: 0 Overflows: 0 Memory Usage: 252kB
-> Index Scan using boat_pkey on boat b (cost=0.28..0.30 rows=1 width=4) (actual time=0.001..0.001 rows=0 loops=3000)
Index Cond: (bid = r.bid)
Filter: (color = 'red'::bpchar)
Rows Removed by Filter: 1
-> Index Scan using sailors_pkey on sailors s (cost=0.29..0.35 rows=1 width=25) (actual time=0.001..0.001 rows=1 loops=10608)
Index Cond: (sid = r.sid)
Planning Time: 0.326 ms
Execution Time: 35.257 ms



2) Query (8) after using Btree index :
Cost = 10300.69 rows
Execution time = 31.906 msec

```
create index idx_btree on reserves using btree(sid,bid);
```

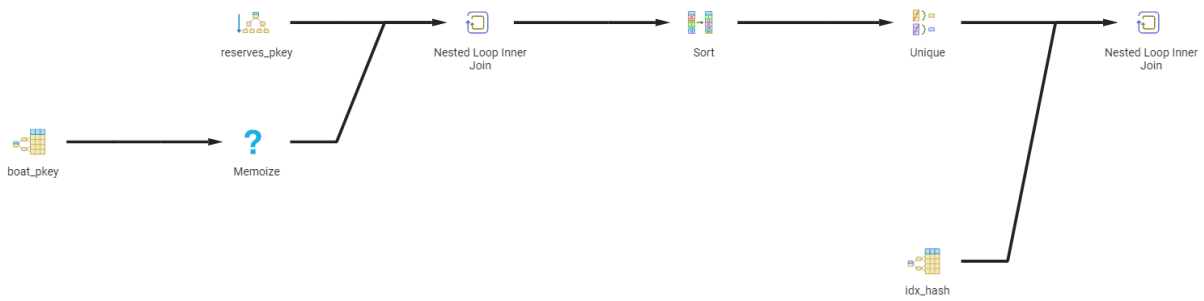
QUERY PLAN text
Nested Loop (cost=3937.38..10300.69 rows=17488 width=21) (actual time=18.318..31.055 rows=10608 loops=1)
-> Unique (cost=3937.09..4024.53 rows=17488 width=4) (actual time=18.299..20.350 rows=10608 loops=1)
-> Sort (cost=3937.09..3980.81 rows=17488 width=4) (actual time=18.299..18.857 rows=18480 loops=1)
Sort Key: r.sid
Sort Method: quicksort Memory: 1635kB
-> Nested Loop (cost=0.58..2704.70 rows=17488 width=4) (actual time=0.097..16.502 rows=18480 loops=1)
-> Index Only Scan using idx_btree on reserves r (cost=0.29..917.29 rows=35000 width=8) (actual time=0.079..3.726 rows=35000 loops=1)
Heap Fetches: 0
-> Memoize (cost=0.29..0.31 rows=1 width=4) (actual time=0.000..0.000 rows=1 loops=35000)
Cache Key: r.bid
Cache Mode: logical
Hits: 32000 Misses: 3000 Evictions: 0 Overflows: 0 Memory Usage: 252kB
-> Index Scan using boat_pkey on boat b (cost=0.28..0.30 rows=1 width=4) (actual time=0.001..0.001 rows=0 loops=3000)
Index Cond: (bid = r.bid)
Filter: (color = 'red'::bpchar)
Rows Removed by Filter: 1
-> Index Scan using sailors_pkey on sailors s (cost=0.29..0.35 rows=1 width=25) (actual time=0.001..0.001 rows=1 loops=10608)
Index Cond: (sid = r.sid)
Planning Time: 0.670 ms
Execution Time: 31.906 ms



3)Query (8) with hash indices only
Cost = 5369.94 rows
Execution time = 39.011 msec

```
create index idx_hash on sailors using hash (sid);
```

QUERY PLAN text
Nested Loop (cost=3989.09..5369.94 rows=17488 width=21) (actual time=20.297..38.675 rows=10608 loops=1)
-> Unique (cost=3989.09..4076.53 rows=17488 width=4) (actual time=20.280..23.451 rows=10608 loops=1)
-> Sort (cost=3989.09..4032.81 rows=17488 width=4) (actual time=20.279..21.239 rows=18480 loops=1)
Sort Key: r.sid
Sort Method: quicksort Memory: 1635kB
-> Nested Loop (cost=0.58..2756.70 rows=17488 width=4) (actual time=0.024..18.308 rows=18480 loops=1)
-> Index Only Scan using reserves_pkey on reserves r (cost=0.29..969.29 rows=35000 width=8) (actual time=0.012..3.567 rows=35000 loops=1)
Heap Fetches: 0
-> Memoize (cost=0.29..0.31 rows=1 width=4) (actual time=0.000..0.000 rows=1 loops=35000)
Cache Key: r.bid
Cache Mode: logical
Hits: 32000 Misses: 3000 Evictions: 0 Overflows: 0 Memory Usage: 252kB
-> Index Scan using boat_pkey on boat b (cost=0.28..0.30 rows=1 width=4) (actual time=0.001..0.001 rows=0 loops=3000)
Index Cond: (bid = r.bid)
Filter: (color = 'red'::bpchar)
Rows Removed by Filter: 1
-> Index Scan using idx_hash on sailors s (cost=0.00..0.07 rows=1 width=25) (actual time=0.001..0.001 rows=1 loops=10608)
Index Cond: (sid = r.sid)
Planning Time: 0.226 ms
Execution Time: 39.011 ms

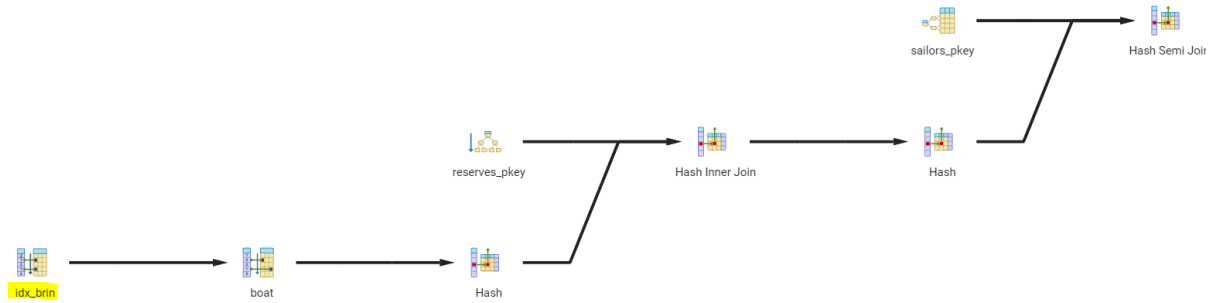


4) Query (8) with BRIN indices only,
Cost = 2346.84 rows
Execution time = 14.579 msec

```
SET enable_seqscan = OFF;
SET enable_bitmapscan= on;
SET enable_hashagg=on;
SET enable_hashjoin=on;
SET enable_incremental_sort=on;
SET enable_indexscan=On; --
SET enable_indexonlyscan=On; --
SET enable_memoize=on;
SET enable_mergejoin=on; --
SET enable_nestloop =on;
SET enable_sort=on;
SET enable_tidscan=on;
```

```
create index idx_brin on boat using brin(color);
```

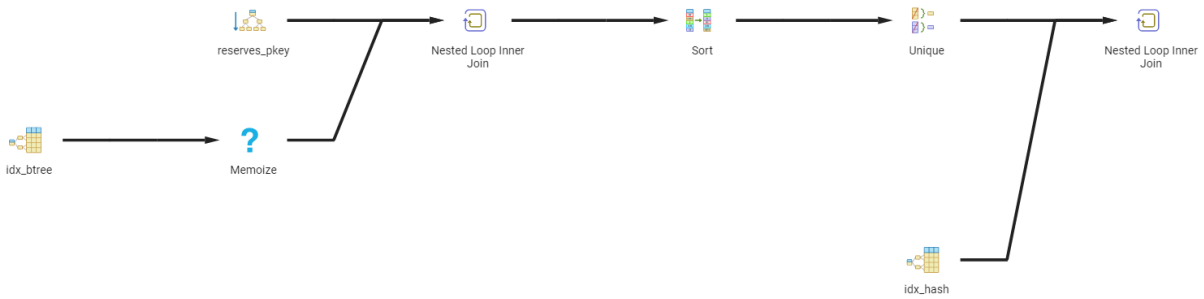
QUERY PLAN text
Hash Semi Join (cost=1373.83..2346.84 rows=17488 width=21) (actual time=7.763..14.154 rows=10608 loops=1)
Hash Cond: (s.sid = r.sid)
-> Index Scan using sailors_pkey on sailors s (cost=0.29..663.29 rows=19000 width=25) (actual time=0.004..2.858 rows=19000 loops=1)
-> Hash (cost=1154.94..1154.94 rows=17488 width=4) (actual time=7.681..7.687 rows=18480 loops=1)
Buckets: 32768 Batches: 1 Memory Usage: 906kB
-> Hash Join (cost=93.93..1154.94 rows=17488 width=4) (actual time=0.500..5.694 rows=18480 loops=1)
Hash Cond: (r.bid = b.bid)
-> Index Only Scan using reserves_pkey on reserves r (cost=0.29..969.29 rows=35000 width=8) (actual time=0.008..2.268 rows=35000 loops=1)
Heap Fetches: 0
-> Hash (cost=74.91..74.91 rows=1499 width=4) (actual time=0.486..0.489 rows=1499 loops=1)
Buckets: 2048 Batches: 1 Memory Usage: 69kB
-> Bitmap Heap Scan on boat b (cost=12.41..74.91 rows=1499 width=4) (actual time=0.062..0.367 rows=1499 loops=1)
Recheck Cond: (color = 'red'::bpchar)
Rows Removed by Index Recheck: 1501
Heap Blocks: lossy=25
-> Bitmap Index Scan on idx_brin (cost=0.00..12.03 rows=3000 width=0) (actual time=0.021..0.021 rows=250 loops=1)
Index Cond: (color = 'red'::bpchar)
Planning Time: 0.378 ms
Execution Time: 14.579 ms



5) Query (8) with mixed indices
Cost = 5369.94 rows
Execution time = 30.882 msec

```
create index idx_btree on boat using btree(bid);
create index idx_hash on sailors using hash (sid);
```

QUERY PLAN text
Nested Loop (cost=3989.09..5369.94 rows=17488 width=21) (actual time=17.847..30.184 rows=10608 loops=1)
-> Unique (cost=3989.09..4076.53 rows=17488 width=4) (actual time=17.810..19.953 rows=10608 loops=1)
-> Sort (cost=3989.09..4032.81 rows=17488 width=4) (actual time=17.809..18.416 rows=18480 loops=1)
Sort Key: r.sid
Sort Method: quicksort Memory: 1635kB
-> Nested Loop (cost=0.58..2756.70 rows=17488 width=4) (actual time=0.036..15.564 rows=18480 loops=1)
-> Index Only Scan using reserves_pkey on reserves r (cost=0.29..969.29 rows=35000 width=8) (actual time=0.010..2.865 rows=35000 loops=1)
Heap Fetches: 0
-> Memoize (cost=0.29..0.31 rows=1 width=4) (actual time=0.000..0.000 rows=1 loops=35000)
Cache Key: r.bid
Cache Mode: logical
Hits: 32000 Misses: 3000 Evictions: 0 Overflows: 0 Memory Usage: 252kB
-> Index Scan using idx_btree on boat b (cost=0.28..0.30 rows=1 width=4) (actual time=0.001..0.001 rows=0 loops=3000)
Index Cond: (bid = r.bid)
Filter: (color = 'red'::bpchar)
Rows Removed by Filter: 1
-> Index Scan using idx_hash on sailors s (cost=0.00..0.07 rows=1 width=25) (actual time=0.001..0.001 rows=1 loops=10608)
Index Cond: (sid = r.sid)
Planning Time: 0.620 ms
Execution Time: 30.882 ms



Query (8) after optimization :

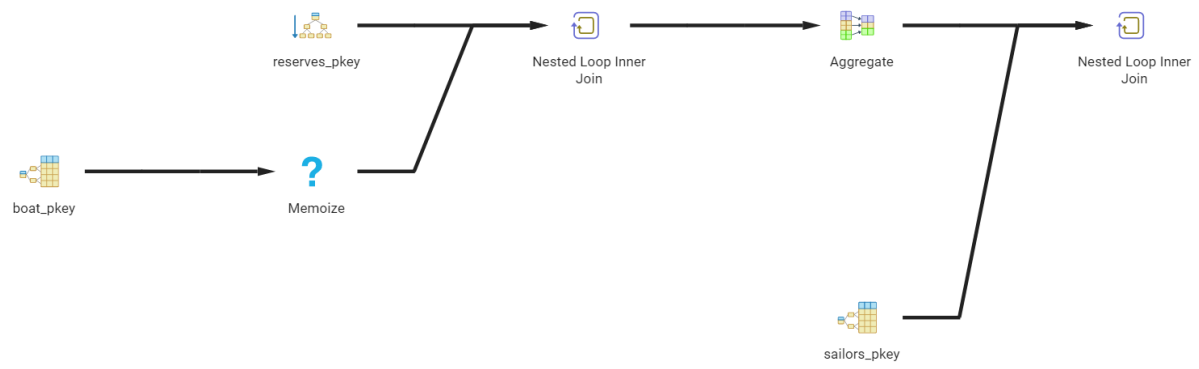
Cost = 9251.46 rows

Execution time = 36.594 msec

```
SET enable_seqscan = OFF;
SET enable_async_append= on;
SET enable_bitmapscan= on;
SET enable_gathermerge=on;
SET enable_hashagg=on;
SET enable_hashjoin=off;
SET enable_incremental_sort=on;
SET enable_indexscan=on;
SET enable_indexonlyscan=On;
SET enable_material=on;
SET enable_memoize=on;
SET enable_mergejoin=off;
SET enable_nestloop =on;
```

```
Explain Analyze
select s.sname
from sailors s
where exists (
select 1 from reserves r where exists
    (select 1 from boat b where b.color='red'
        and r.bid=b.bid) and s.sid=r.sid)
```

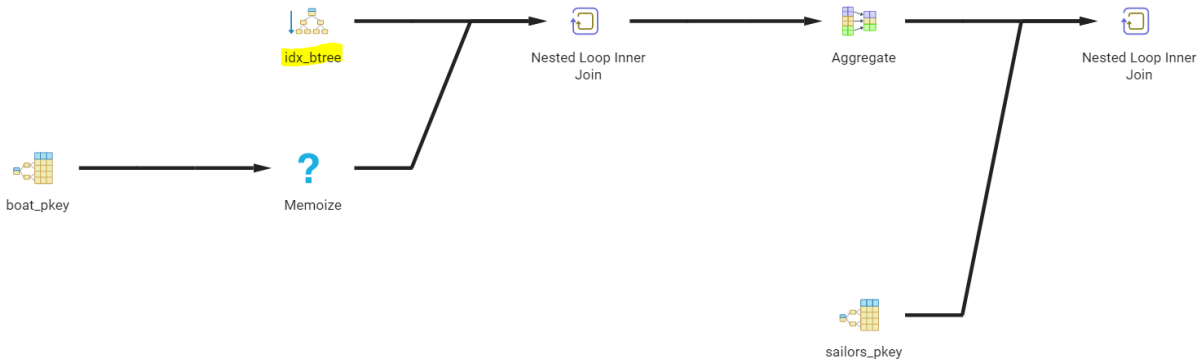
QUERY PLAN text
Nested Loop (cost=2800.71..9251.46 rows=17488 width=21) (actual time=18.366..36.190 rows=10608 loops=1)
-> HashAggregate (cost=2800.42..2975.30 rows=17488 width=4) (actual time=18.342..20.526 rows=10608 loops=1)
Group Key: r.sid
Batches: 1 Memory Usage: 1297kB
-> Nested Loop (cost=0.58..2756.70 rows=17488 width=4) (actual time=0.460..15.137 rows=18480 loops=1)
-> Index Only Scan using reserves_pkey on reserves r (cost=0.29..969.29 rows=35000 width=8) (actual time=0.433..3.142 rows=35000 loops=1)
Heap Fetches: 0
-> Memoize (cost=0.29..0.31 rows=1 width=4) (actual time=0.000..0.000 rows=1 loops=35000)
Cache Key: r.bid
Cache Mode: logical
Hits: 32000 Misses: 3000 Evictions: 0 Overflows: 0 Memory Usage: 252kB
-> Index Scan using boat_pkey on boat b (cost=0.28..0.30 rows=1 width=4) (actual time=0.001..0.001 rows=0 loops=3000)
Index Cond: (bid = r.bid)
Filter: (color = 'red'::bpchar)
Rows Removed by Filter: 1
-> Index Scan using sailors_pkey on sailors s (cost=0.29..0.35 rows=1 width=25) (actual time=0.001..0.001 rows=1 loops=10608)
Index Cond: (sid = r.sid)
Planning Time: 1.955 ms
Execution Time: 36.594 ms



2) Query (8) after optimization with B+ trees indices only,
Cost = 9199.46 rows
Execution time = 46.382 msec

```
create index idx_btree on reserves using btree(sid,bid);
```

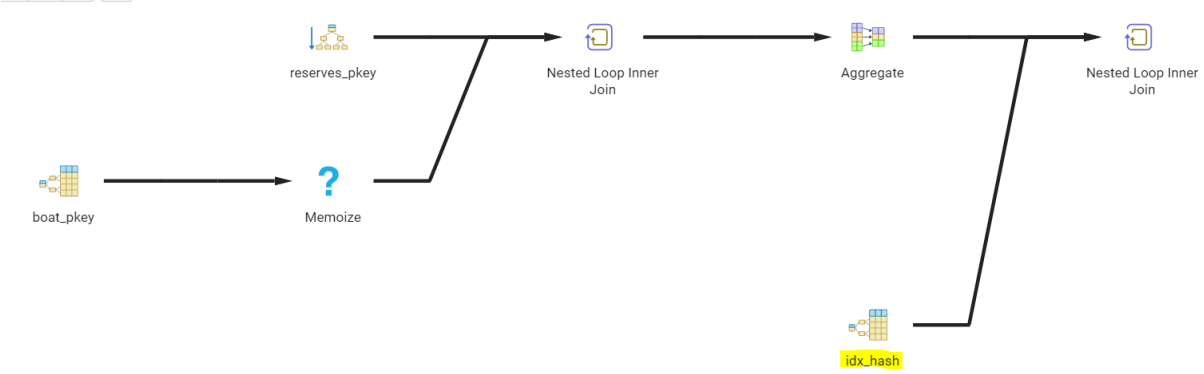
QUERY PLAN text
Nested Loop (cost=2748.71..9199.46 rows=17488 width=21) (actual time=21.381..45.966 rows=10608 loops=1)
-> HashAggregate (cost=2748.42..2923.30 rows=17488 width=4) (actual time=21.351..23.717 rows=10608 loops=1)
Group Key: r.sid
Batches: 1 Memory Usage: 1297kB
-> Nested Loop (cost=0.58..2704.70 rows=17488 width=4) (actual time=0.192..17.661 rows=18480 loops=1)
-> Index Only Scan using idx_btree on reserves r (cost=0.29..917.29 rows=35000 width=8) (actual time=0.178..4.091 rows=35000 loops=1)
Heap Fetches: 0
-> Memoize (cost=0.29..0.31 rows=1 width=4) (actual time=0.000..0.000 rows=1 loops=35000)
Cache Key: r.bid
Cache Mode: logical
Hits: 32000 Misses: 3000 Evictions: 0 Overflows: 0 Memory Usage: 252kB
-> Index Scan using boat_pkey on boat b (cost=0.28..0.30 rows=1 width=4) (actual time=0.001..0.001 rows=0 loops=3000)
Index Cond: (bid = r.bid)
Filter: (color = 'red'::bpchar)
Rows Removed by Filter: 1
-> Index Scan using sailors_pkey on sailors s (cost=0.29..0.35 rows=1 width=25) (actual time=0.002..0.002 rows=1 loops=10608)
Index Cond: (sid = r.sid)
Planning Time: 0.567 ms
Execution Time: 46.382 ms



3) Query (8) after optimization with hash indices only,
Cost = 4268.72 rows
Execution time = 30.151 msec

```
create index idx_hash on sailors using hash (sid);
```

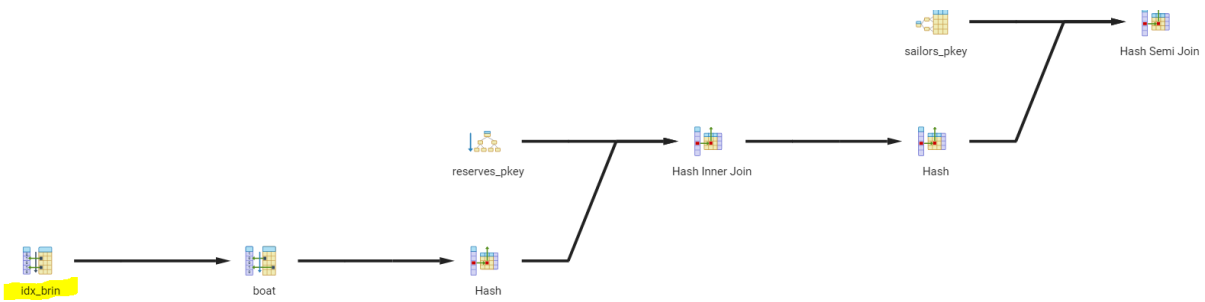
QUERY PLAN text
Nested Loop (cost=2800.42..4268.72 rows=17488 width=21) (actual time=19.100..29.489 rows=10608 loops=1)
-> HashAggregate (cost=2800.42..2975.30 rows=17488 width=4) (actual time=19.080..20.387 rows=10608 loops=1)
Group Key: r.sid
Batches: 1 Memory Usage: 1297kB
-> Nested Loop (cost=0.58..2756.70 rows=17488 width=4) (actual time=0.074..15.727 rows=18480 loops=1)
-> Index Only Scan using reserves_pkey on reserves r (cost=0.29..969.29 rows=35000 width=8) (actual time=0.062..2.881 rows=35000 loops=1)
Heap Fetches: 0
-> Memoize (cost=0.29..0.31 rows=1 width=4) (actual time=0.000..0.000 rows=1 loops=35000)
Cache Key: r.bid
Cache Mode: logical
Hits: 32000 Misses: 3000 Evictions: 0 Overflows: 0 Memory Usage: 252kB
-> Index Scan using boat_pkey on boat b (cost=0.28..0.30 rows=1 width=4) (actual time=0.001..0.001 rows=0 loops=3000)
Index Cond: (bid = r.bid)
Filter: (color = 'red'::bpchar)
Rows Removed by Filter: 1
-> Index Scan using idx_hash on sailors s (cost=0.00..0.07 rows=1 width=25) (actual time=0.001..0.001 rows=1 loops=10608)
Index Cond: (sid = r.sid)
Planning Time: 0.403 ms
Execution Time: 30.151 ms



4) Query (8) after optimization with BRIN indices only,
Cost = 2346.84 rows
Execution time = 12.897 msec

```
create index idx_brin on boat using brin(color);
```

QUERY PLAN text
Hash Semi Join (cost=1373.83..2346.84 rows=17488 width=21) (actual time=8.259..12.454 rows=10608 loops=1)
Hash Cond: (s.sid = r.sid)
-> Index Scan using sailors_pkey on sailors s (cost=0.29..663.29 rows=19000 width=25) (actual time=0.004..1.761 rows=19000 loops=1)
-> Hash (cost=1154.94..1154.94 rows=17488 width=4) (actual time=8.170..8.172 rows=18480 loops=1)
Buckets: 32768 Batches: 1 Memory Usage: 906kB
-> Hash Join (cost=93.93..1154.94 rows=17488 width=4) (actual time=0.483..5.936 rows=18480 loops=1)
Hash Cond: (r.bid = b.bid)
-> Index Only Scan using reserves_pkey on reserves r (cost=0.29..969.29 rows=35000 width=8) (actual time=0.015..2.392 rows=35000 loops=1)
Heap Fetches: 0
-> Hash (cost=74.91..74.91 rows=1499 width=4) (actual time=0.461..0.462 rows=1499 loops=1)
Buckets: 2048 Batches: 1 Memory Usage: 69kB
-> Bitmap Heap Scan on boat b (cost=12.41..74.91 rows=1499 width=4) (actual time=0.026..0.320 rows=1499 loops=1)
Recheck Cond: (color = 'red'::bpchar)
Rows Removed by Index Recheck: 1501
Heap Blocks: lossy=25
-> Bitmap Index Scan on idx_brin (cost=0.00..12.03 rows=3000 width=0) (actual time=0.020..0.021 rows=250 loops=1)
Index Cond: (color = 'red'::bpchar)
Planning Time: 0.306 ms
Execution Time: 12.897 ms



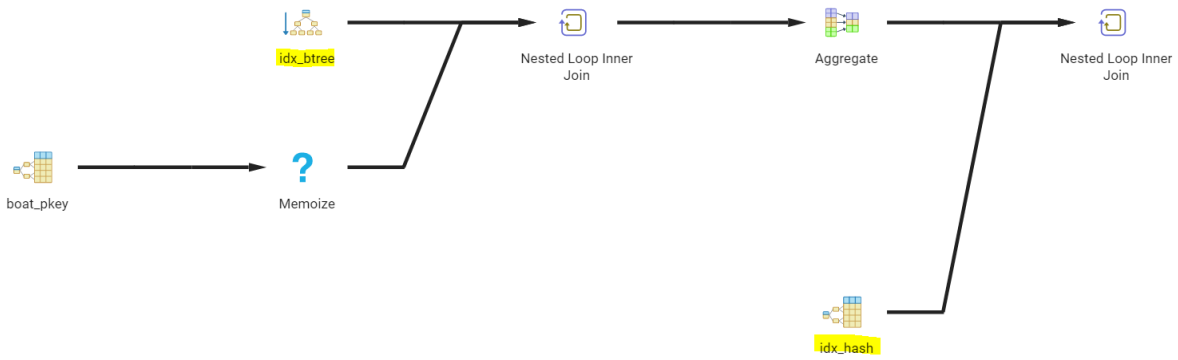
5) Query (8) after optimization with mixed indices.

Cost = 4216.72 rows

Execution time = 36.912 msec

```
create index idx_btree on reserves using btree(sid,bid);
create index idx_hash on sailors using hash (sid);
```

QUERY PLAN text
Nested Loop (cost=2748.42..4216.72 rows=17488 width=21) (actual time=21.268..36.481 rows=10608 loops=1)
-> HashAggregate (cost=2748.42..2923.30 rows=17488 width=4) (actual time=21.246..23.532 rows=10608 loops=1)
Group Key: r.sid
Batches: 1 Memory Usage: 1297kB
-> Nested Loop (cost=0.58..2704.70 rows=17488 width=4) (actual time=0.051..16.732 rows=18480 loops=1)
-> Index Only Scan using idx_btree on reserves r (cost=0.29..917.29 rows=35000 width=8) (actual time=0.025..3.120 rows=35000 loops=1)
Heap Fetches: 0
-> Memoize (cost=0.29..0.31 rows=1 width=4) (actual time=0.000..0.000 rows=1 loops=35000)
Cache Key: r.bid
Cache Mode: logical
Hits: 32000 Misses: 3000 Evictions: 0 Overflows: 0 Memory Usage: 252kB
-> Index Scan using boat_pkey on boat b (cost=0.28..0.30 rows=1 width=4) (actual time=0.001..0.001 rows=0 loops=3000)
Index Cond: (bid = r.bid)
Filter: (color = 'red'::bpchar)
Rows Removed by Filter: 1
-> Index Scan using idx_hash on sailors s (cost=0.00..0.07 rows=1 width=25) (actual time=0.001..0.001 rows=1 loops=10608)
Index Cond: (sid = r.sid)
Planning Time: 0.498 ms
Execution Time: 36.912 ms



Query 9

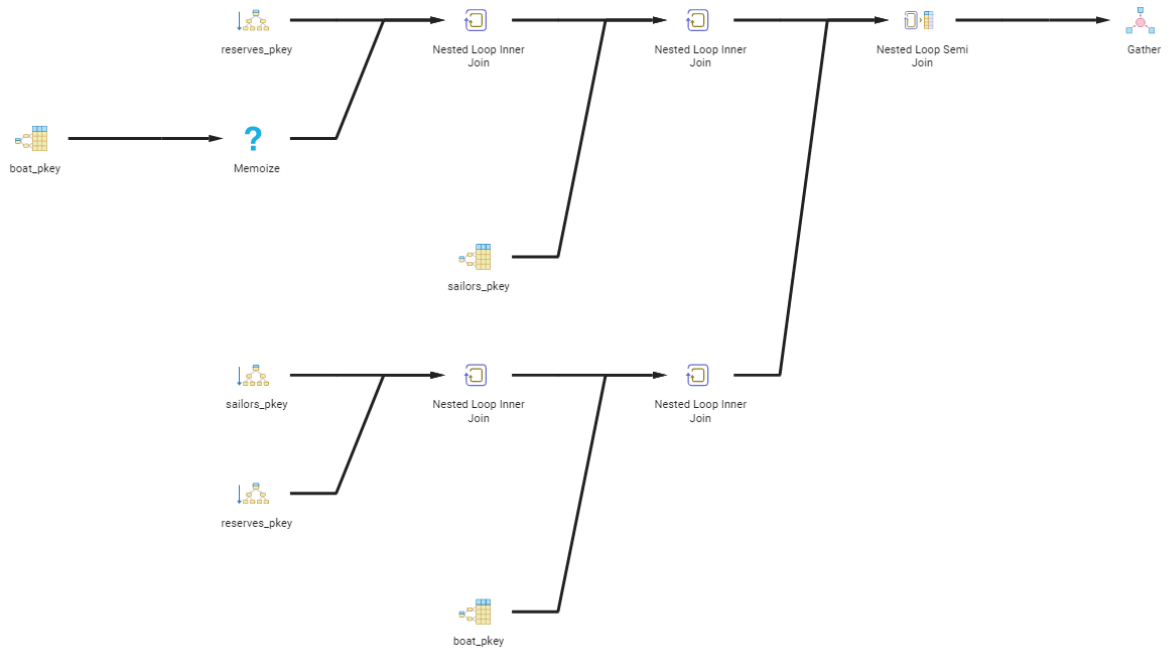
1) Query (9) without an index:
Cost = 18093.55 rows
Execution time = 205.865 msec

```
SET enable_seqscan = OFF;
SET enable_async_append= on;
SET enable_bitmapscan= on;
SET enable_gathermerge=on;
SET enable_hashagg=on;
SET enable_hashjoin=off;
SET enable_incremental_sort=on;
SET enable_indexscan=on;
SET enable_indexonlyscan=On;
SET enable_material=on;
SET enable_memoize=on;
SET enable_mergejoin=off;
SET enable_nestloop =on;
```

```
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 PLAN text
Gather (cost=1001.73..18093.55 rows=16119 width=21) (actual time=139.559..205.387 rows=1038 loops=1)
Workers Planned: 1
Workers Launched: 1
-> Nested Loop Semi Join (cost=1.73..15481.65 rows=9482 width=21) (actual time=122.314..173.028 rows=519 loops=2)
-> Nested Loop (cost=0.87..5264.60 rows=10287 width=29) (actual time=0.199..40.713 rows=9240 loops=2)
-> Nested Loop (cost=0.58..1876.69 rows=10287 width=4) (actual time=0.181..21.383 rows=9240 loops=2)
-> Parallel Index Only Scan using reserves_pkey on reserves r (cost=0.29..825.17 rows=20588 width=8) (actual time=0.148..2.963 rows=17500 loo…
Heap Fetches: 0
-> Memoize (cost=0.29..0.31 rows=1 width=4) (actual time=0.001..0.001 rows=1 loops=35000)
Cache Key: r.bid
Cache Mode: logical
Hits: 13224 Misses: 3000 Evictions: 0 Overflows: 0 Memory Usage: 252kB
Worker 0: Hits: 15776 Misses: 3000 Evictions: 0 Overflows: 0 Memory Usage: 252kB
-> Index Scan using boat_pkey on boat b (cost=0.28..0.30 rows=1 width=4) (actual time=0.002..0.002 rows=0 loops=6000)
Index Cond: (bid = r.bid)
Filter: (color = 'red'::bpchar)
Rows Removed by Filter: 1
-> Index Scan using sailors_pkey on sailors s (cost=0.29..0.33 rows=1 width=25) (actual time=0.002..0.002 rows=1 loops=18480)
Index Cond: (sid = r.sid)
-> Nested Loop (cost=0.86..0.98 rows=1 width=8) (actual time=0.014..0.014 rows=0 loops=18480)
-> Nested Loop (cost=0.58..0.68 rows=1 width=12) (actual time=0.003..0.004 rows=4 loops=18480)

Join Filter: (s2.sid = r2.sid)
-> Index Only Scan using sailors_pkey on sailors s2 (cost=0.29..0.32 rows=1 width=4) (actual time=0.001..0.001 rows=1 loops=18480)
Index Cond: (sid = s.sid)
Heap Fetches: 0
-> Index Only Scan using reserves_pkey on reserves r2 (cost=0.29..0.34 rows=2 width=8) (actual time=0.001..0.002 rows=4 loops=18480)
Index Cond: (sid = r.sid)
Heap Fetches: 0
-> Index Scan using boat_pkey on boat b2 (cost=0.28..0.30 rows=1 width=4) (actual time=0.003..0.003 rows=0 loops=64686)
Index Cond: (bid = r2.bid)
Filter: (color = 'green'::bpchar)
Rows Removed by Filter: 1
Planning Time: 1.422 ms
Execution Time: 205.865 ms



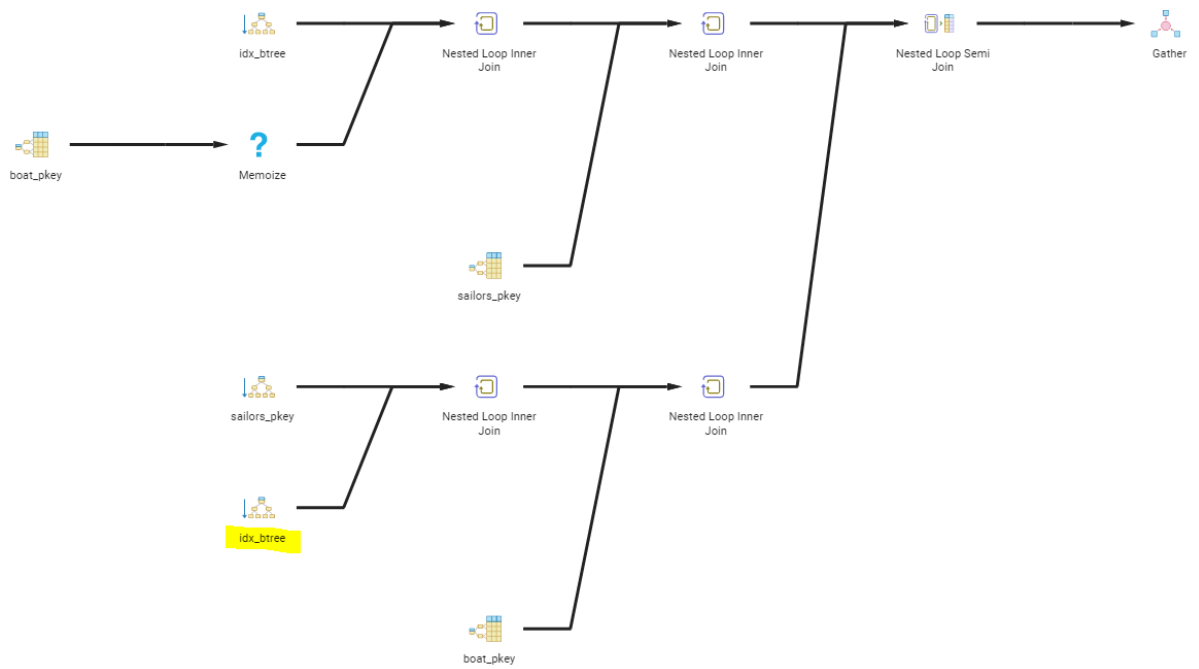
2)Query (9) with B+ trees indices only,

Cost = 18026.27 rows

Execution time = 208.786 msec

```
create index idx_btree on reserves using btree(sid,bid);
```

QUERY PLAN text
Gather (cost=1001.73..18026.27 rows=16119 width=21) (actual time=142.302..208.649 rows=1038 loops=1)
Workers Planned: 1
Workers Launched: 1
-> Nested Loop Semi Join (cost=1.73..15414.37 rows=9482 width=21) (actual time=120.409..172.735 rows=519 loops=2)
-> Nested Loop (cost=0.87..5212.60 rows=10287 width=29) (actual time=0.248..41.423 rows=9240 loops=2)
-> Nested Loop (cost=0.58..1824.69 rows=10287 width=4) (actual time=0.228..21.114 rows=9240 loops=2)
-> Parallel Index Only Scan using idx_btree on reserves r (cost=0.29..773.17 rows=20588 width=8) (actual time=0.192..3.540 rows=17500 loops=2)
Heap Fetches: 0
-> Memoize (cost=0.29..0.31 rows=1 width=4) (actual time=0.001..0.001 rows=1 loops=35000)
Cache Key: r.bid
Cache Mode: logical
Hits: 11640 Misses: 3000 Evictions: 0 Overflows: 0 Memory Usage: 252kB
Worker 0: Hits: 17360 Misses: 3000 Evictions: 0 Overflows: 0 Memory Usage: 252kB
-> Index Scan using boat_pkey on boat b (cost=0.28..0.30 rows=1 width=4) (actual time=0.002..0.002 rows=0 loops=6000)
Index Cond: (bid = r.bid)
Filter: (color = 'red'::bpchar)
Rows Removed by Filter: 1
-> Index Scan using sailors_pkey on sailors s (cost=0.29..0.33 rows=1 width=25) (actual time=0.002..0.002 rows=1 loops=18480)
Index Cond: (sid = r.sid)
-> Nested Loop (cost=0.86..0.98 rows=1 width=8) (actual time=0.014..0.014 rows=0 loops=18480)
-> Nested Loop (cost=0.58..0.68 rows=1 width=12) (actual time=0.003..0.005 rows=4 loops=18480)
Join Filter: (s2.sid = r2.sid)
-> Index Only Scan using sailors_pkey on sailors s2 (cost=0.29..0.32 rows=1 width=4) (actual time=0.001..0.001 rows=1 loops=18480)
Index Cond: (sid = s.sid)
Heap Fetches: 0
-> Index Only Scan using idx_btree on reserves r2 (cost=0.29..0.34 rows=2 width=8) (actual time=0.001..0.002 rows=4 loops=18480)
Index Cond: (sid = r.sid)
Heap Fetches: 0
-> Index Scan using boat_pkey on boat b2 (cost=0.28..0.30 rows=1 width=4) (actual time=0.002..0.002 rows=0 loops=64686)
Index Cond: (bid = r2.bid)
Filter: (color = 'green'::bpchar)
Rows Removed by Filter: 1
Planning Time: 1.085 ms
Execution Time: 208.786 ms

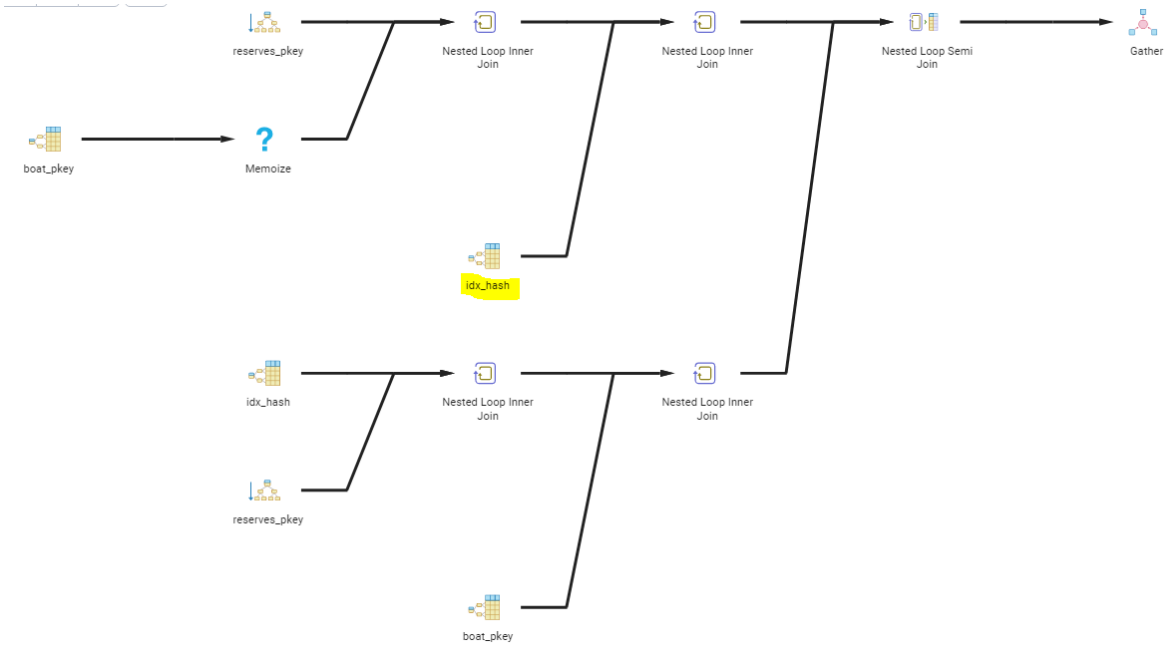


3)Query (9) with hash indices only,

Cost =12501.54 rows
Execution time = 160.648 msec

```
create index idx_hash on sailors using hash (sid);
```

QUERY PLAN text
Gather (cost=1001.15..12501.54 rows=16119 width=21) (actual time=104.576..160.235 rows=1038 loops=1)
Workers Planned: 1
Workers Launched: 1
-> Nested Loop Semi Join (cost=1.15..9889.64 rows=9482 width=21) (actual time=87.014..129.006 rows=519 loops=2)
-> Nested Loop (cost=0.58..2321.19 rows=10287 width=29) (actual time=0.204..33.818 rows=9240 loops=2)
-> Nested Loop (cost=0.58..1876.69 rows=10287 width=4) (actual time=0.185..19.048 rows=9240 loops=2)
-> Parallel Index Only Scan using reserves_pkey on reserves r (cost=0.29..825.17 rows=20588 width=8) (actual time=0.154..2.533 rows=17500 loo…
Heap Fetches: 0
-> Memoize (cost=0.29..0.31 rows=1 width=4) (actual time=0.001..0.001 rows=1 loops=35000)
Cache Key: r.bid
Cache Mode: logical
Hits: 9864 Misses: 3000 Evictions: 0 Overflows: 0 Memory Usage: 252kB
Worker 0: Hits: 19136 Misses: 3000 Evictions: 0 Overflows: 0 Memory Usage: 252kB
-> Index Scan using boat_pkey on boat b (cost=0.28..0.30 rows=1 width=4) (actual time=0.002..0.002 rows=0 loops=6000)
Index Cond: (bid = r.bid)
Filter: (color = 'red'::bpchar)
Rows Removed by Filter: 1
-> Index Scan using idx_hash on sailors s (cost=0.00..0.04 rows=1 width=25) (actual time=0.001..0.001 rows=1 loops=18480)
Index Cond: (sid = r.sid)
-> Nested Loop (cost=0.57..0.73 rows=1 width=8) (actual time=0.010..0.010 rows=0 loops=18480)
-> Nested Loop (cost=0.29..0.42 rows=1 width=12) (actual time=0.002..0.003 rows=4 loops=18480)
Join Filter: ((s.sid = s2.sid) AND (s2.sid = r2.sid))
-> Index Scan using idx_hash on sailors s2 (cost=0.00..0.04 rows=1 width=4) (actual time=0.001..0.001 rows=1 loops=18480)
Index Cond: (sid = r.sid)
-> Index Only Scan using reserves_pkey on reserves r2 (cost=0.29..0.35 rows=2 width=8) (actual time=0.001..0.002 rows=4 loops=18480)
Index Cond: (sid = s.sid)
Heap Fetches: 0
-> Index Scan using boat_pkey on boat b2 (cost=0.28..0.30 rows=1 width=4) (actual time=0.002..0.002 rows=0 loops=64686)
Index Cond: (bid = r2.bid)
Filter: (color = 'green'::bpchar)
Rows Removed by Filter: 1
Planning Time: 1.490 ms
Execution Time: 160.648 ms

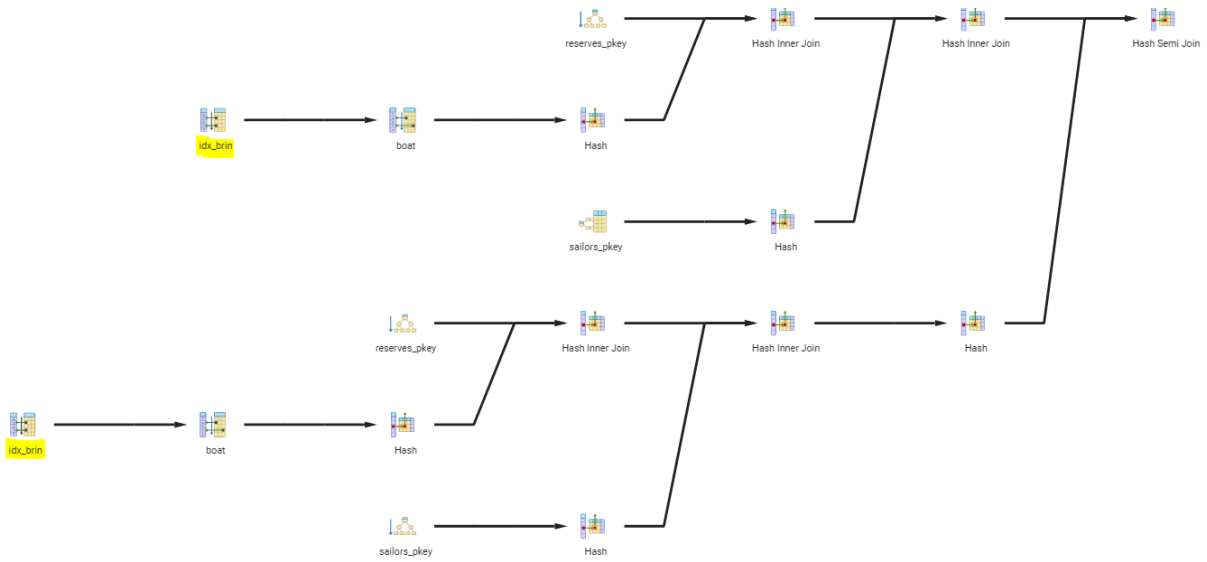


4) Query (9) with BRIN indices only,
Cost = 4485.50 rows
Execution time = 40.315 msec

```
SET enable_seqscan = OFF;
SET enable_bitmapscan= on;
SET enable_hashagg=on;
SET enable_hashjoin=on;
SET enable_incremental_sort=on;
SET enable_indexscan=On; --
SET enable_indexonlyscan=On; --
SET enable_memoize=on;
SET enable_mergejoin=on; --
SET enable_nestloop =on;
SET enable_sort=on;
SET enable_tidscan=on;
```

```
create index idx_brin on boat using brin(color);
```

QUERY PLAN text
Hash Semi Join (cost=3153.36..4485.50 rows=16119 width=21) (actual time=28.127..39.506 rows=1038 loops=1)
Hash Cond: (s.sid = s2.sid)
-> Hash Join (cost=994.72..2101.65 rows=17488 width=29) (actual time=6.173..20.087 rows=18480 loops=1)
Hash Cond: (r.sid = s.sid)
-> Hash Join (cost=93.93..1154.94 rows=17488 width=4) (actual time=0.468..9.717 rows=18480 loops=1)
Hash Cond: (r.bid = b.bid)
-> Index Only Scan using reserves_pkey on reserves r (cost=0.29..969.29 rows=35000 width=8) (actual time=0.008..4.075 rows=35000 loops=1)
Heap Fetches: 0
-> Hash (cost=74.91..74.91 rows=1499 width=4) (actual time=0.451..0.453 rows=1499 loops=1)
Buckets: 2048 Batches: 1 Memory Usage: 69kB
-> Bitmap Heap Scan on boat b (cost=12.41..74.91 rows=1499 width=4) (actual time=0.032..0.319 rows=1499 loops=1)
Recheck Cond: (color = 'red'::bpchar)
Rows Removed by Index Recheck: 1501
Heap Blocks: lossy=25
-> Bitmap Index Scan on idx_brin (cost=0.00..12.03 rows=3000 width=0) (actual time=0.025..0.025 rows=250 loops=1)
Index Cond: (color = 'red'::bpchar)
-> Hash (cost=663.29..663.29 rows=19000 width=25) (actual time=5.599..5.599 rows=19000 loops=1)
Buckets: 32768 Batches: 1 Memory Usage: 1370kB
-> Index Scan using sailors_pkey on sailors s (cost=0.29..663.29 rows=19000 width=25) (actual time=0.007..2.918 rows=19000 loops=1)
-> Hash (cost=1939.74..1939.74 rows=17512 width=8) (actual time=16.581..16.584 rows=16520 loops=1)
Buckets: 32768 Batches: 1 Memory Usage: 902kB
-> Hash Join (cost=832.75..1939.74 rows=17512 width=8) (actual time=4.826..13.558 rows=16520 loops=1)
Hash Cond: (r2.sid = s2.sid)
-> Hash Join (cost=93.96..1154.97 rows=17512 width=4) (actual time=1.443..6.590 rows=16520 loops=1)
Hash Cond: (r2.bid = b2.bid)
-> Index Only Scan using reserves_pkey on reserves r2 (cost=0.29..969.29 rows=35000 width=8) (actual time=0.013..2.671 rows=35000 loops=1)
Heap Fetches: 0
-> Hash (cost=74.91..74.91 rows=1501 width=4) (actual time=0.441..0.442 rows=1501 loops=1)
Buckets: 2048 Batches: 1 Memory Usage: 69kB
-> Bitmap Heap Scan on boat b2 (cost=12.41..74.91 rows=1501 width=4) (actual time=0.170..0.320 rows=1501 loops=1)
Recheck Cond: (color = 'green'::bpchar)
Rows Removed by Index Recheck: 1499
Heap Blocks: lossy=25
-> Bitmap Index Scan on idx_brin (cost=0.00..12.03 rows=3000 width=0) (actual time=0.046..0.046 rows=250 loops=1)
Index Cond: (color = 'green'::bpchar)
-> Hash (cost=501.29..501.29 rows=19000 width=4) (actual time=3.309..3.309 rows=19000 loops=1)
Buckets: 32768 Batches: 1 Memory Usage: 924kB
-> Index Only Scan using sailors_pkey on sailors s2 (cost=0.29..501.29 rows=19000 width=4) (actual time=0.011..1.309 rows=19000 loops=1)
Heap Fetches: 0
Planning Time: 1.293 ms
Execution Time: 40.315 ms

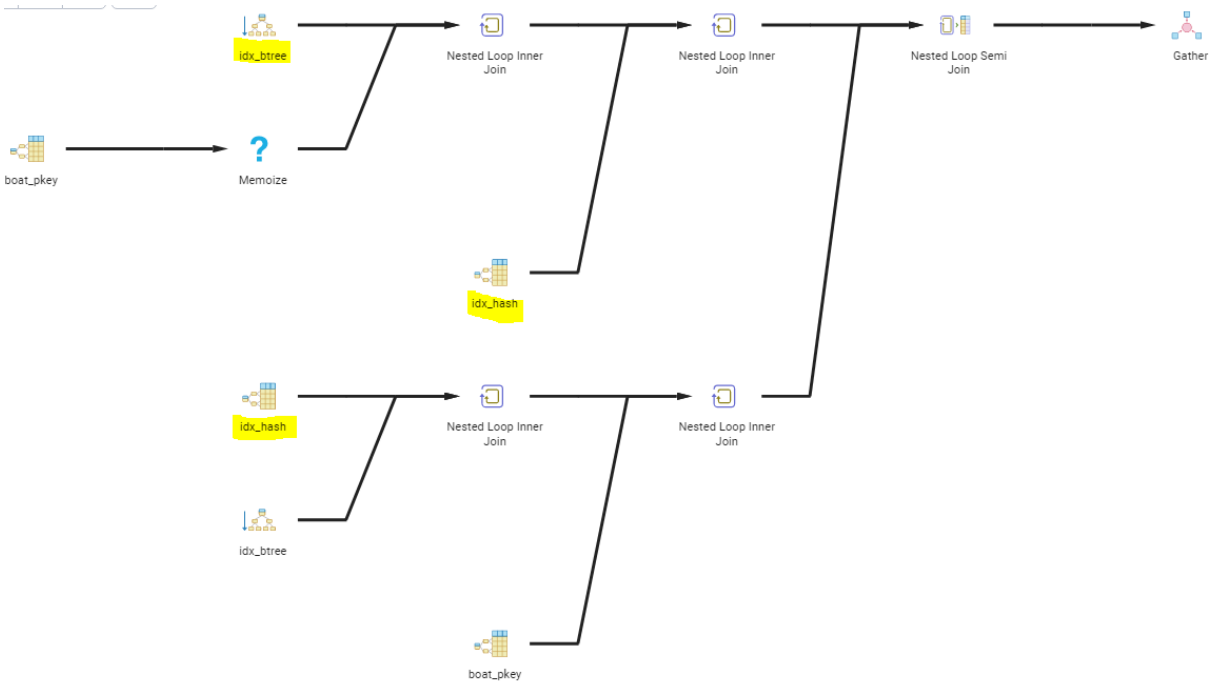


5) Query (9) with mixed indices.
Cost = 12421.39 rows
Execution time = 224.528 msec

```
create index idx_btree on reserves using btree(sid,bid);
create index idx_hash on sailors using hash (sid);
```

QUERY PLAN text
Gather (cost=1001.15..12421.39 rows=16119 width=21) (actual time=156.420..224.345 rows=1038 loops=1)
Workers Planned: 1
Workers Launched: 1
-> Nested Loop Semi Join (cost=1.15..9809.49 rows=9482 width=21) (actual time=105.954..162.525 rows=519 loops=2)
-> Nested Loop (cost=0.58..2269.19 rows=10287 width=29) (actual time=0.255..41.847 rows=9240 loops=2)
-> Nested Loop (cost=0.58..1824.69 rows=10287 width=4) (actual time=0.240..23.344 rows=9240 loops=2)
-> Parallel Index Only Scan using idx_btree on reserves r (cost=0.29..773.17 rows=20588 width=8) (actual time=0.211..3.907 rows=17500 loops=2)
Heap Fetches: 0
-> Memoize (cost=0.29..0.31 rows=1 width=4) (actual time=0.001..0.001 rows=1 loops=35000)
Cache Key: r.bid
Cache Mode: logical
Hits: 11310 Misses: 2964 Evictions: 0 Overflows: 0 Memory Usage: 249kB
Worker 0: Hits: 17788 Misses: 2938 Evictions: 0 Overflows: 0 Memory Usage: 246kB
-> Index Scan using boat_pkey on boat b (cost=0.28..0.30 rows=1 width=4) (actual time=0.002..0.002 rows=0 loops=5902)
Index Cond: (bid = r.bid)
Filter: (color = 'red'::bpchar)
Rows Removed by Filter: 1
-> Index Scan using idx_hash on sailors s (cost=0.00..0.04 rows=1 width=25) (actual time=0.001..0.001 rows=1 loops=18480)
Index Cond: (sid = r.sid)
-> Nested Loop (cost=0.57..0.72 rows=1 width=8) (actual time=0.013..0.013 rows=0 loops=18480)
-> Nested Loop (cost=0.29..0.42 rows=1 width=12) (actual time=0.003..0.004 rows=4 loops=18480)

```
Join Filter: ((s.sid = s2.sid) AND (s2.sid = r2.sid))
-> Index Scan using idx_hash on sailors s2 (cost=0.00..0.04 rows=1 width=4) (actual time=0.001..0.001 rows=1 loops=18480)
Index Cond: (sid = r.sid)
-> Index Only Scan using idx_btree on reserves r2 (cost=0.29..0.35 rows=2 width=8) (actual time=0.002..0.002 rows=4 loops=18480)
Index Cond: (sid = s.sid)
Heap Fetches: 0
-> Index Scan using boat_pkey on boat b2 (cost=0.28..0.30 rows=1 width=4) (actual time=0.002..0.002 rows=0 loops=64686)
Index Cond: (bid = r2.bid)
Filter: (color = 'green'::bpchar)
Rows Removed by Filter: 1
Planning Time: 1.887 ms
Execution Time: 224.528 ms
```



Query (9) after optimization:

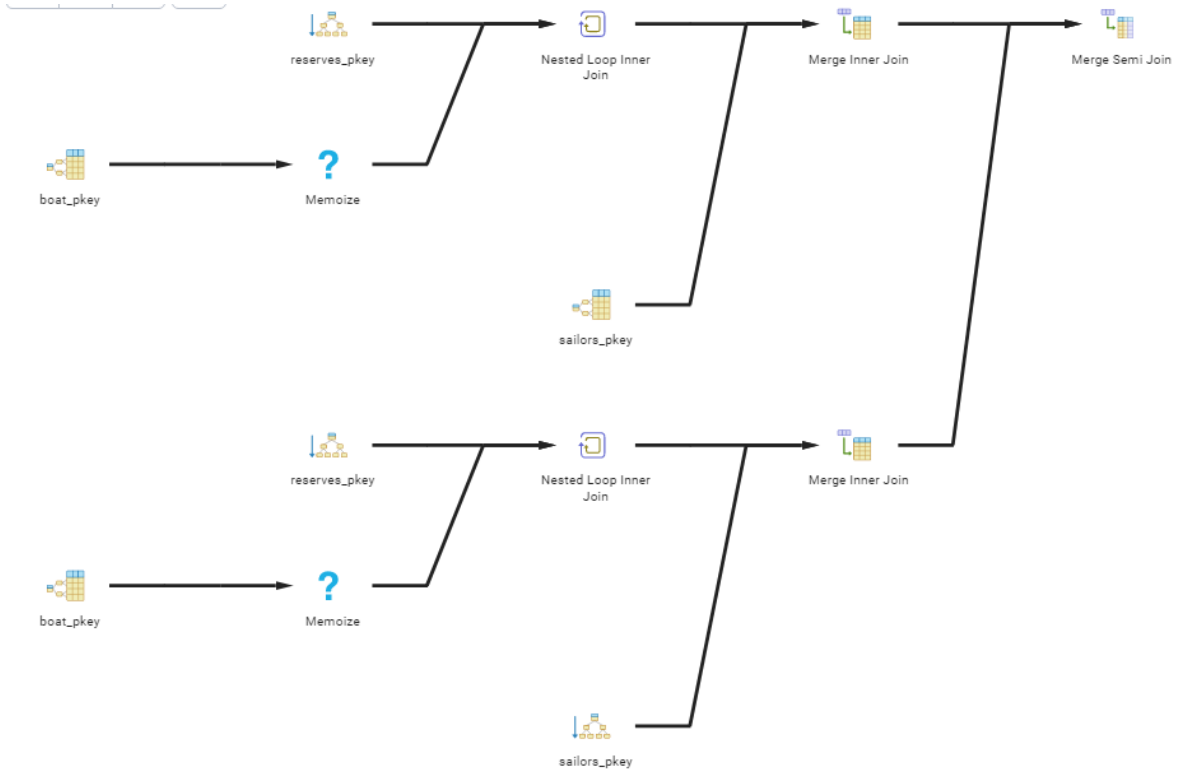
1) Query (9) after optimization without an index,
   Cost = 7459.16 rows
   Execution time = 44.039 msec

```
SET enable_seqscan = OFF;
SET enable_async_append= on;
SET enable_bitmapscan= on;
SET enable_gathermerge=on;
SET enable_hashagg=on;
SET enable_hashjoin=off;
SET enable_incremental_sort=on;
SET enable_indexscan=on;
SET enable_indexonlyscan=On;
SET enable_material=on;
SET enable_memoize=on;
SET enable_mergejoin=on;
SET enable_nestloop =on;
```

```
Explain Analyze
select s.sname
from sailors s
inner join reserves r on s.sid=r.sid
inner join boat b on b.bid=r.bid
where 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 PLAN text
Merge Semi Join (cost=1.74..7459.16 rows=16119 width=21) (actual time=11.224..43.856 rows=1038 loops=1)
Merge Cond: (s.sid = s2.sid)
-> Merge Join (cost=0.87..3686.09 rows=17488 width=29) (actual time=0.023..22.138 rows=17486 loops=1)
Merge Cond: (r.sid = s.sid)
-> Nested Loop (cost=0.58..2756.70 rows=17488 width=4) (actual time=0.016..16.372 rows=17486 loops=1)
-> Index Only Scan using reserves_pkey on reserves r (cost=0.29..969.29 rows=35000 width=8) (actual time=0.007..2.690 rows=34006 loops=1)
Heap Fetches: 0
-> Memoize (cost=0.29..0.31 rows=1 width=4) (actual time=0.000..0.000 rows=1 loops=34006)
Cache Key: r.bid
Cache Mode: logical
Hits: 31006 Misses: 3000 Evictions: 0 Overflows: 0 Memory Usage: 252kB
-> Index Scan using boat_pkey on boat b (cost=0.28..0.30 rows=1 width=4) (actual time=0.001..0.001 rows=0 loops=3000)
Index Cond: (bid = r.bid)
Filter: (color = 'red'::bpchar)
Rows Removed by Filter: 1
-> Index Scan using sailors_pkey on sailors s (cost=0.29..663.29 rows=19000 width=25) (actual time=0.005..2.442 rows=18006 loops=1)
-> Merge Join (cost=0.87..3524.39 rows=17512 width=8) (actual time=4.290..20.131 rows=16520 loops=1)
Merge Cond: (r2.sid = s2.sid)
-> Nested Loop (cost=0.58..2756.70 rows=17512 width=4) (actual time=4.149..16.257 rows=16520 loops=1)
-> Index Only Scan using reserves_pkey on reserves r2 (cost=0.29..969.29 rows=35000 width=8) (actual time=0.005..2.661 rows=35000 loops=1)
Heap Fetches: 0

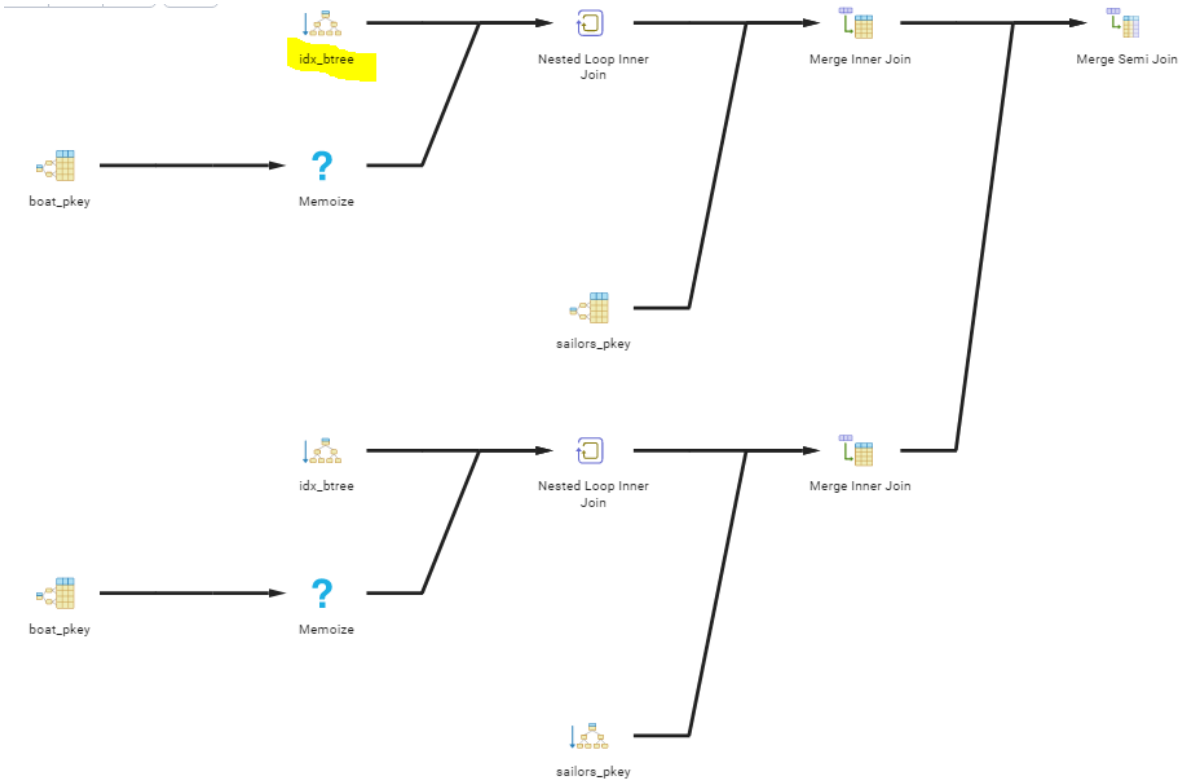
-> Memoize (cost=0.29..0.31 rows=1 width=4) (actual time=0.000..0.000 rows=0 loops=35000)
Cache Key: r2.bid
Cache Mode: logical
Hits: 32000 Misses: 3000 Evictions: 0 Overflows: 0 Memory Usage: 252kB
-> Index Scan using boat_pkey on boat b2 (cost=0.28..0.30 rows=1 width=4) (actual time=0.001..0.001 rows=1 loops=3000)
Index Cond: (bid = r2.bid)
Filter: (color = 'green'::bpchar)
Rows Removed by Filter: 0
-> Index Only Scan using sailors_pkey on sailors s2 (cost=0.29..501.29 rows=19000 width=4) (actual time=0.016..1.283 rows=18005 loops=1)
Heap Fetches: 0
Planning Time: 0.843 ms
Execution Time: 44.039 ms



2) Query (9) after optimization with B+ trees indices only,
Cost = 7355.16 rows
Execution time = 41.408 msec

```
create index idx_btree on reserves using btree(sid,bid);
```

QUERY PLAN text
Merge Semi Join (cost=1.74..7355.16 rows=16119 width=21) (actual time=11.352..41.329 rows=1038 loops=1)
Merge Cond: (s.sid = s2.sid)
-> Merge Join (cost=0.87..3634.09 rows=17488 width=29) (actual time=0.018..21.114 rows=17486 loops=1)
Merge Cond: (r.sid = s.sid)
-> Nested Loop (cost=0.58..2704.70 rows=17488 width=4) (actual time=0.013..15.827 rows=17486 loops=1)
-> Index Only Scan using idx_btree on reserves r (cost=0.29..917.29 rows=35000 width=8) (actual time=0.005..2.823 rows=34006 loops=1)
Heap Fetches: 0
-> Memoize (cost=0.29..0.31 rows=1 width=4) (actual time=0.000..0.000 rows=1 loops=34006)
Cache Key: r.bid
Cache Mode: logical
Hits: 31006 Misses: 3000 Evictions: 0 Overflows: 0 Memory Usage: 252kB
-> Index Scan using boat_pkey on boat b (cost=0.28..0.30 rows=1 width=4) (actual time=0.001..0.001 rows=0 loops=3000)
Index Cond: (bid = r.bid)
Filter: (color = 'red'::bpchar)
Rows Removed by Filter: 1
-> Index Scan using sailors_pkey on sailors s (cost=0.29..663.29 rows=19000 width=25) (actual time=0.004..2.116 rows=18006 loops=1)
-> Merge Join (cost=0.87..3472.39 rows=17512 width=8) (actual time=4.252..18.845 rows=16520 loops=1)
Merge Cond: (r2.sid = s2.sid)
-> Nested Loop (cost=0.58..2704.70 rows=17512 width=4) (actual time=4.115..15.223 rows=16520 loops=1)
-> Index Only Scan using idx_btree on reserves r2 (cost=0.29..917.29 rows=35000 width=8) (actual time=0.004..3.070 rows=35000 loops=1)
Heap Fetches: 0
-> Memoize (cost=0.29..0.31 rows=1 width=4) (actual time=0.000..0.000 rows=0 loops=35000)
-> Memoize (cost=0.29..0.31 rows=1 width=4) (actual time=0.000..0.000 rows=0 loops=35000)
Cache Key: r2.bid
Cache Mode: logical
Hits: 32000 Misses: 3000 Evictions: 0 Overflows: 0 Memory Usage: 252kB
-> Index Scan using boat_pkey on boat b2 (cost=0.28..0.30 rows=1 width=4) (actual time=0.001..0.001 rows=1 loops=3000)
Index Cond: (bid = r2.bid)
Filter: (color = 'green'::bpchar)
Rows Removed by Filter: 0
-> Index Only Scan using sailors_pkey on sailors s2 (cost=0.29..501.29 rows=19000 width=4) (actual time=0.006..1.183 rows=18005 loops=1)
Heap Fetches: 0
Planning Time: 1.249 ms
Execution Time: 41.408 ms

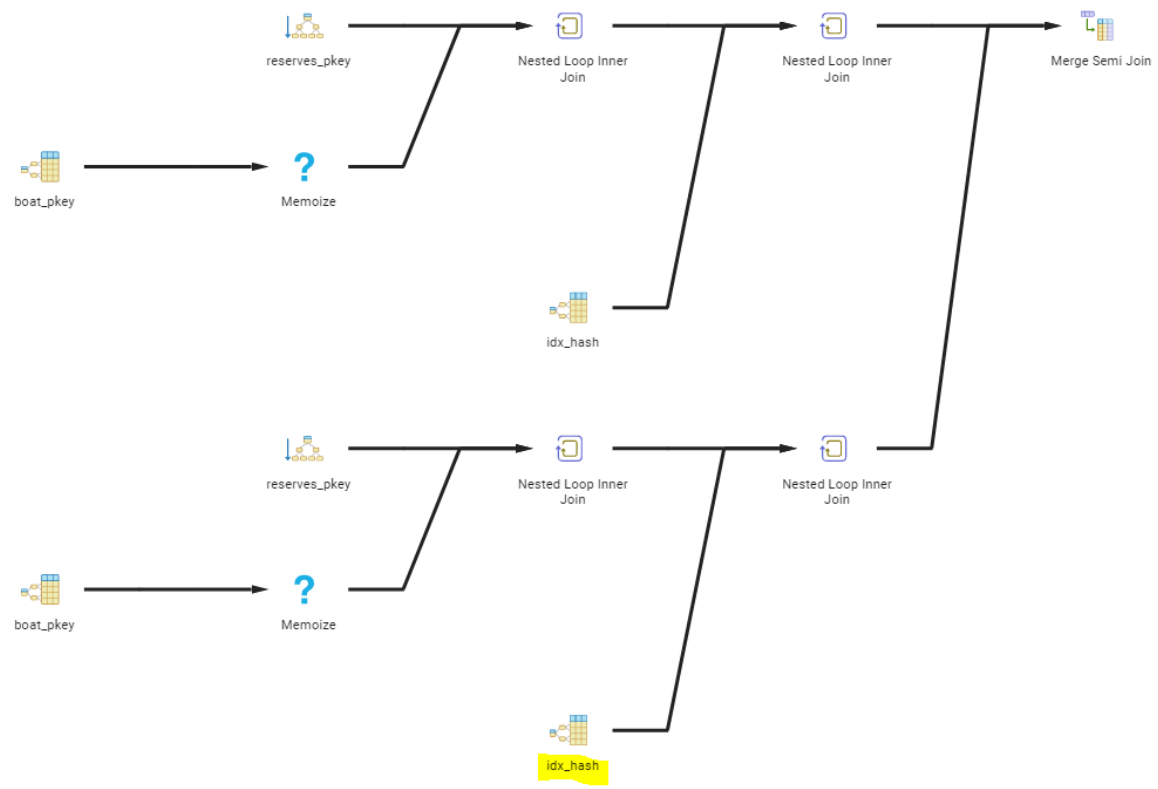


3)Query (9) after optimization with hash indices only,
Cost = 7274.50 rows
Execution time= 80.874 msec

```
create index idx_hash on sailors using hash (sid);
```

QUERY PLAN text
Merge Semi Join (cost=1.16..7274.50 rows=16119 width=21) (actual time=23.671..80.763 rows=1038 loops=1)
Merge Cond: (s.sid = s2.sid)
-> Nested Loop (cost=0.58..3512.39 rows=17488 width=29) (actual time=0.023..38.452 rows=17486 loops=1)
-> Nested Loop (cost=0.58..2756.70 rows=17488 width=4) (actual time=0.017..20.724 rows=17486 loops=1)
-> Index Only Scan using reserves_pkey on reserves r (cost=0.29..969.29 rows=35000 width=8) (actual time=0.007..3.326 rows=34006 loops=1)
Heap Fetches: 0
-> Memoize (cost=0.29..0.31 rows=1 width=4) (actual time=0.000..0.000 rows=1 loops=34006)
Cache Key: r.bid
Cache Mode: logical
Hits: 31006 Misses: 3000 Evictions: 0 Overflows: 0 Memory Usage: 252kB
-> Index Scan using boat_pkey on boat b (cost=0.28..0.30 rows=1 width=4) (actual time=0.001..0.001 rows=0 loops=3000)
Index Cond: (bid = r.bid)
Filter: (color = 'red'::bpchar)
Rows Removed by Filter: 1
-> Index Scan using idx_hash on sailors s (cost=0.00..0.04 rows=1 width=25) (actual time=0.001..0.001 rows=1 loops=17486)
Index Cond: (sid = r.sid)
-> Nested Loop (cost=0.58..3513.43 rows=17512 width=8) (actual time=7.057..40.385 rows=16520 loops=1)
-> Nested Loop (cost=0.58..2756.70 rows=17512 width=4) (actual time=7.043..22.795 rows=16520 loops=1)
-> Index Only Scan using reserves_pkey on reserves r2 (cost=0.29..969.29 rows=35000 width=8) (actual time=0.005..3.983 rows=35000 loops=1)
Heap Fetches: 0
-> Memoize (cost=0.29..0.31 rows=1 width=4) (actual time=0.000..0.000 rows=0 loops=35000)

Cache Key: r2.bid
Cache Mode: logical
Hits: 32000 Misses: 3000 Evictions: 0 Overflows: 0 Memory Usage: 252kB
-> Index Scan using boat_pkey on boat b2 (cost=0.28..0.30 rows=1 width=4) (actual time=0.002..0.002 rows=1 loops=3000)
Index Cond: (bid = r2.bid)
Filter: (color = 'green'::bpchar)
Rows Removed by Filter: 0
-> Index Scan using idx_hash on sailors s2 (cost=0.00..0.04 rows=1 width=4) (actual time=0.001..0.001 rows=1 loops=16520)
Index Cond: (sid = r2.sid)
Planning Time: 1.724 ms
Execution Time: 80.874 ms

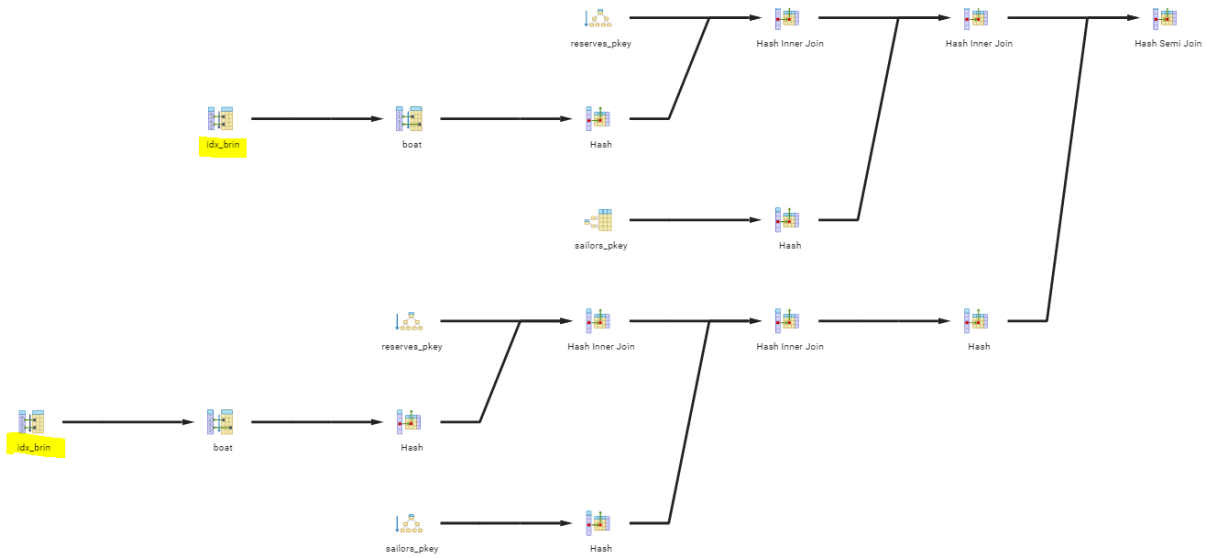


4) Query (9) after optimization with BRIN indices only,
Cost = 4485.50 rows
Execution time = 35.221 msec

```
SET enable_seqscan = OFF;
SET enable_bitmapscan= on;
SET enable_hashagg=on;
SET enable_hashjoin=on;
SET enable_incremental_sort=on;
SET enable_indexscan=On; --
SET enable_indexonlyscan=On; --
SET enable_memoize=on;
SET enable_mergejoin=on; --
SET enable_nestloop =on;
SET enable_sort=on;
SET enable_tidscan=on;

create index idx_brin on boat using brin(color);
```

QUERY PLAN text
Hash Semi Join (cost=3153.36..4485.50 rows=16119 width=21) (actual time=25.974..34.062 rows=1038 loops=1)
Hash Cond: (s.sid = s2.sid)
-> Hash Join (cost=994.72..2101.65 rows=17488 width=29) (actual time=5.690..15.610 rows=18480 loops=1)
Hash Cond: (r.sid = s.sid)
-> Hash Join (cost=93.93..1154.94 rows=17488 width=4) (actual time=0.497..7.071 rows=18480 loops=1)
Hash Cond: (r.bid = b.bid)
-> Index Only Scan using reserves_pkey on reserves r (cost=0.29..969.29 rows=35000 width=8) (actual time=0.009..2.783 rows=35000 loops=1)
Heap Fetches: 0
-> Hash (cost=74.91..74.91 rows=1499 width=4) (actual time=0.480..0.482 rows=1499 loops=1)
Buckets: 2048 Batches: 1 Memory Usage: 69kB
-> Bitmap Heap Scan on boat b (cost=12.41..74.91 rows=1499 width=4) (actual time=0.023..0.326 rows=1499 loops=1)
Recheck Cond: (color = 'red'::bpchar)
Rows Removed by Index Recheck: 1501
Heap Blocks: lossy=25
-> Bitmap Index Scan on idx_brin (cost=0.00..12.03 rows=3000 width=0) (actual time=0.018..0.018 rows=250 loops=1)
Index Cond: (color = 'red'::bpchar)
-> Hash (cost=663.29..663.29 rows=19000 width=25) (actual time=5.109..5.110 rows=19000 loops=1)
Buckets: 32768 Batches: 1 Memory Usage: 1370kB
-> Index Scan using sailors_pkey on sailors s (cost=0.29..663.29 rows=19000 width=25) (actual time=0.007..2.579 rows=19000 loops=1)
-> Hash (cost=1939.74..1939.74 rows=17512 width=8) (actual time=16.478..16.483 rows=16520 loops=1)
Buckets: 32768 Batches: 1 Memory Usage: 902kB
-> Hash Join (cost=832.75..1939.74 rows=17512 width=8) (actual time=4.869..13.754 rows=16520 loops=1)
Hash Cond: (r2.sid = s2.sid)
-> Hash Join (cost=93.96..1154.97 rows=17512 width=4) (actual time=1.510..6.929 rows=16520 loops=1)
Hash Cond: (r2.bid = b2.bid)
-> Index Only Scan using reserves_pkey on reserves r2 (cost=0.29..969.29 rows=35000 width=8) (actual time=0.011..2.858 rows=35000 loops=1)
Heap Fetches: 0
-> Hash (cost=74.91..74.91 rows=1501 width=4) (actual time=0.515..0.517 rows=1501 loops=1)
Buckets: 2048 Batches: 1 Memory Usage: 69kB
-> Bitmap Heap Scan on boat b2 (cost=12.41..74.91 rows=1501 width=4) (actual time=0.165..0.318 rows=1501 loops=1)
Recheck Cond: (color = 'green'::bpchar)
Rows Removed by Index Recheck: 1499
Heap Blocks: lossy=25
-> Bitmap Index Scan on idx_brin (cost=0.00..12.03 rows=3000 width=0) (actual time=0.043..0.043 rows=250 loops=1)
Index Cond: (color = 'green'::bpchar)
-> Hash (cost=501.29..501.29 rows=19000 width=4) (actual time=3.271..3.272 rows=19000 loops=1)
Buckets: 32768 Batches: 1 Memory Usage: 924kB
-> Index Only Scan using sailors_pkey on sailors s2 (cost=0.29..501.29 rows=19000 width=4) (actual time=0.023..1.238 rows=19000 loops=1)
Heap Fetches: 0
Planning Time: 0.930 ms
Execution Time: 35.221 ms

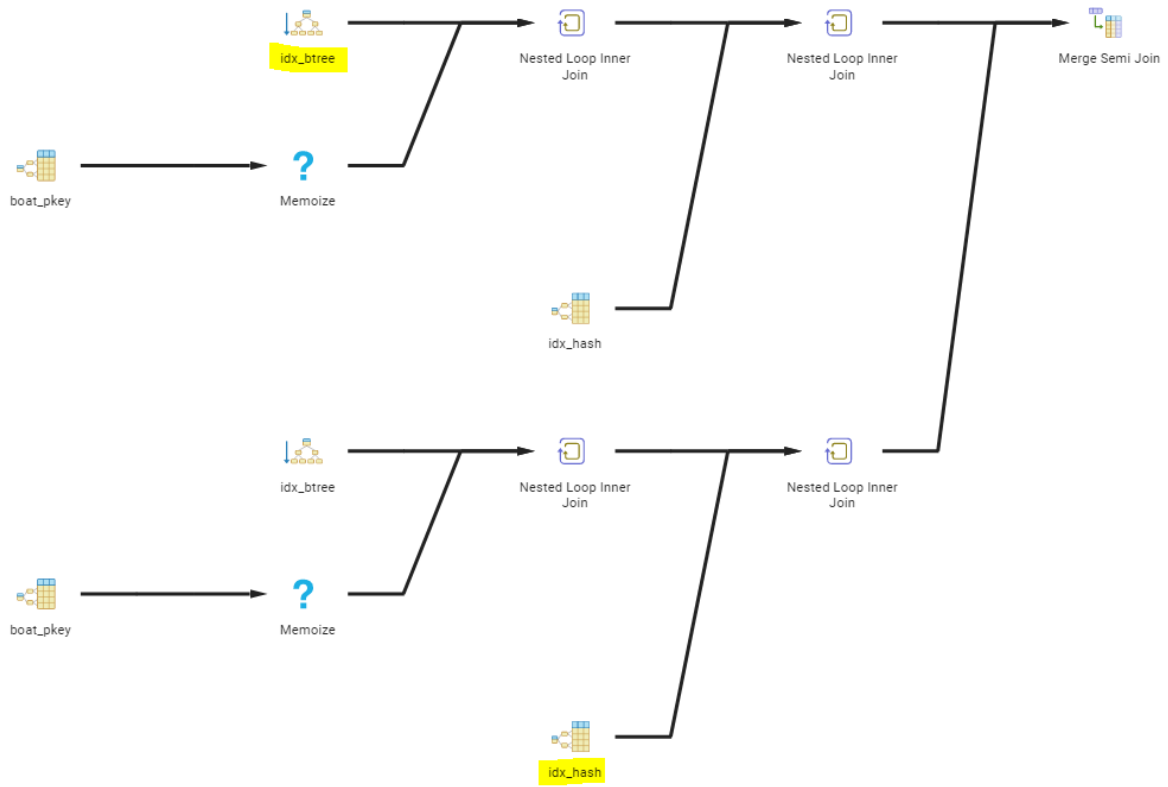


5) Query (9) after optimization with mixed indices
Cost = 7170.50 rows
Execution time = 102.212 msec

```
create index idx_btree on reserves using btree(sid,bid);
create index idx_hash on sailors using hash (sid);
```

QUERY PLAN text
Merge Semi Join (cost=1.16..7170.50 rows=16119 width=21) (actual time=21.719..102.069 rows=1038 loops=1)
Merge Cond: (s.sid = s2.sid)
-> Nested Loop (cost=0.58..3460.39 rows=17488 width=29) (actual time=0.022..50.970 rows=17486 loops=1)
-> Nested Loop (cost=0.58..2704.70 rows=17488 width=4) (actual time=0.012..25.823 rows=17486 loops=1)
-> Index Only Scan using idx_btree on reserves r (cost=0.29..917.29 rows=35000 width=8) (actual time=0.005..4.562 rows=34006 loops=1)
Heap Fetches: 0
-> Memoize (cost=0.29..0.31 rows=1 width=4) (actual time=0.000..0.000 rows=1 loops=34006)
Cache Key: r.bid
Cache Mode: logical
Hits: 31006 Misses: 3000 Evictions: 0 Overflows: 0 Memory Usage: 252kB
-> Index Scan using boat_pkey on boat b (cost=0.28..0.30 rows=1 width=4) (actual time=0.001..0.001 rows=0 loops=3000)
Index Cond: (bid = r.bid)
Filter: (color = 'red'::bpchar)
Rows Removed by Filter: 1
-> Index Scan using idx_hash on sailors s (cost=0.00..0.04 rows=1 width=25) (actual time=0.001..0.001 rows=1 loops=17486)
Index Cond: (sid = r.sid)
-> Nested Loop (cost=0.58..3461.43 rows=17512 width=8) (actual time=4.738..48.508 rows=16520 loops=1)
-> Nested Loop (cost=0.58..2704.70 rows=17512 width=4) (actual time=4.730..25.707 rows=16520 loops=1)
-> Index Only Scan using idx_btree on reserves r2 (cost=0.29..917.29 rows=35000 width=8) (actual time=0.004..4.868 rows=35000 loops=1)
Heap Fetches: 0
-> Memoize (cost=0.29..0.31 rows=1 width=4) (actual time=0.000..0.000 rows=0 loops=35000)

Cache Key: r2.bid
Cache Mode: logical
Hits: 32000 Misses: 3000 Evictions: 0 Overflows: 0 Memory Usage: 252kB
-> Index Scan using boat_pkey on boat b2 (cost=0.28..0.30 rows=1 width=4) (actual time=0.001..0.001 rows=1 loops=3000)
Index Cond: (bid = r2.bid)
Filter: (color = 'green'::bpchar)
Rows Removed by Filter: 0
-> Index Scan using idx_hash on sailors s2 (cost=0.00..0.04 rows=1 width=4) (actual time=0.001..0.001 rows=1 loops=16520)
Index Cond: (sid = r2.sid)
Planning Time: 1.288 ms
Execution Time: 102.212 ms



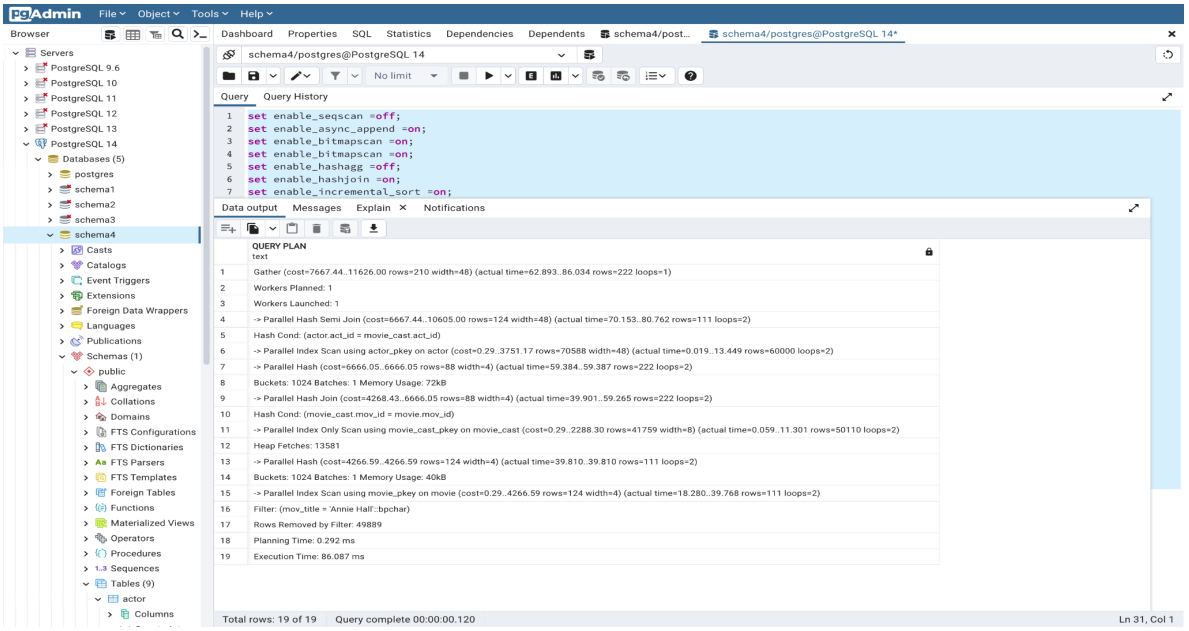
Schema 4

Query10

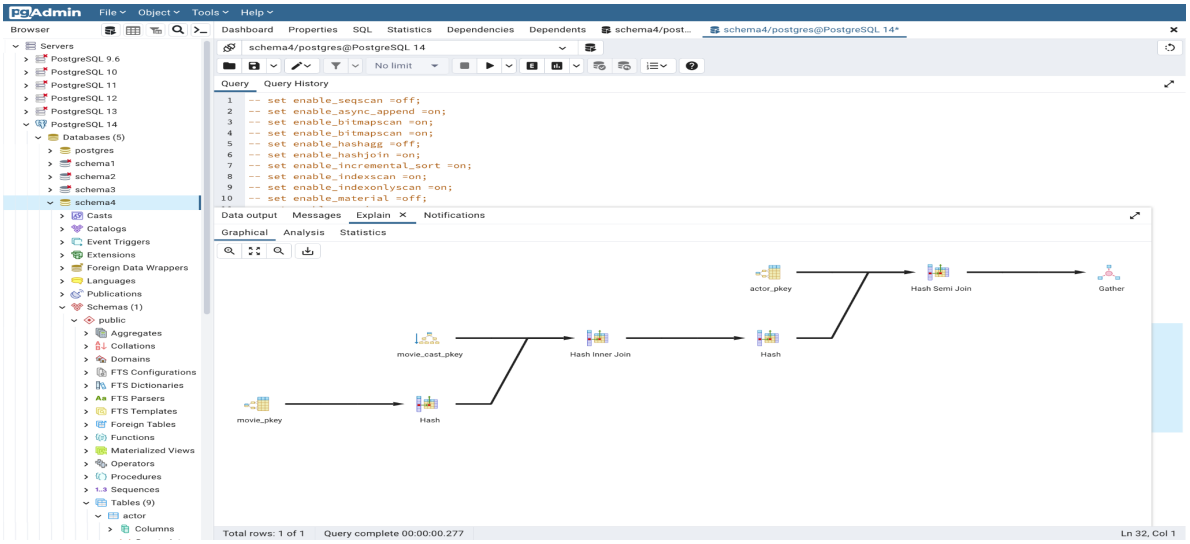
For this query, I adjusted the flags in order to find the best cost so that it could be reflected on the optimization process.

At first the execution time=86.087ms and the cost was 11626.00 and we'll try to reduce the this cost by applying several indices techniques which are:

1)Btree
2)Hash
3)Brin
4) Mixed
5)Optimised Query
6)Btree on optimised
7)Hash on optimised
8)Brin on optimised
9)Mixed on optimised



And this is the physical plan for the query:

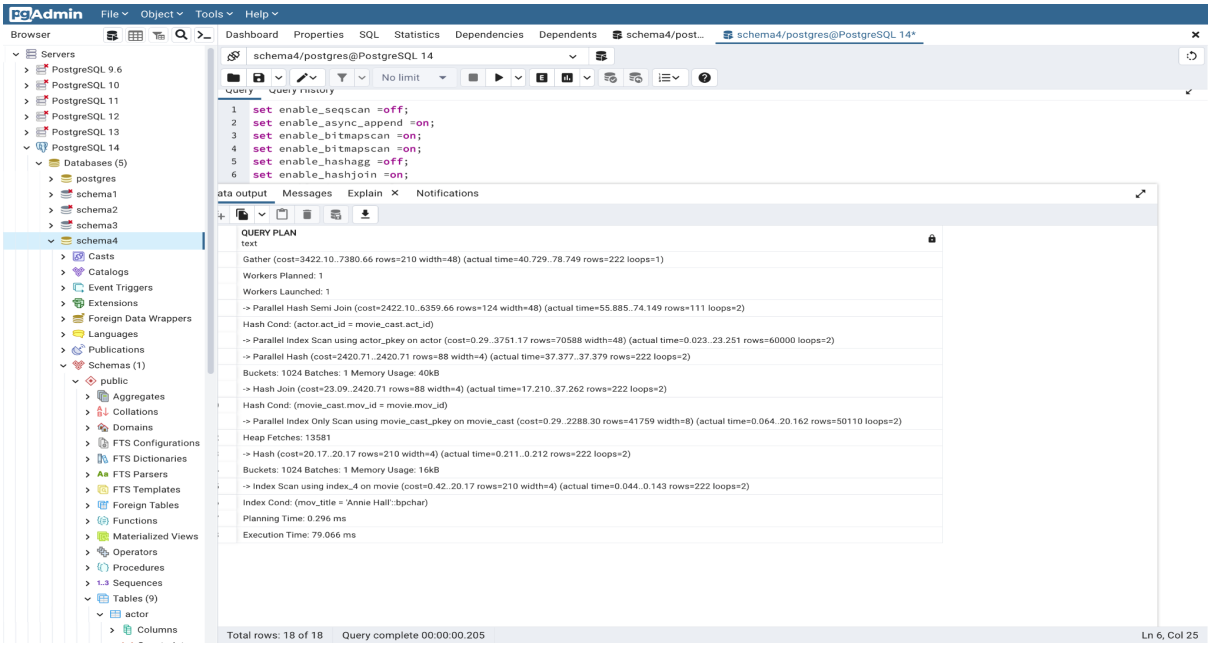


1)<u>Btree indexing:</u>

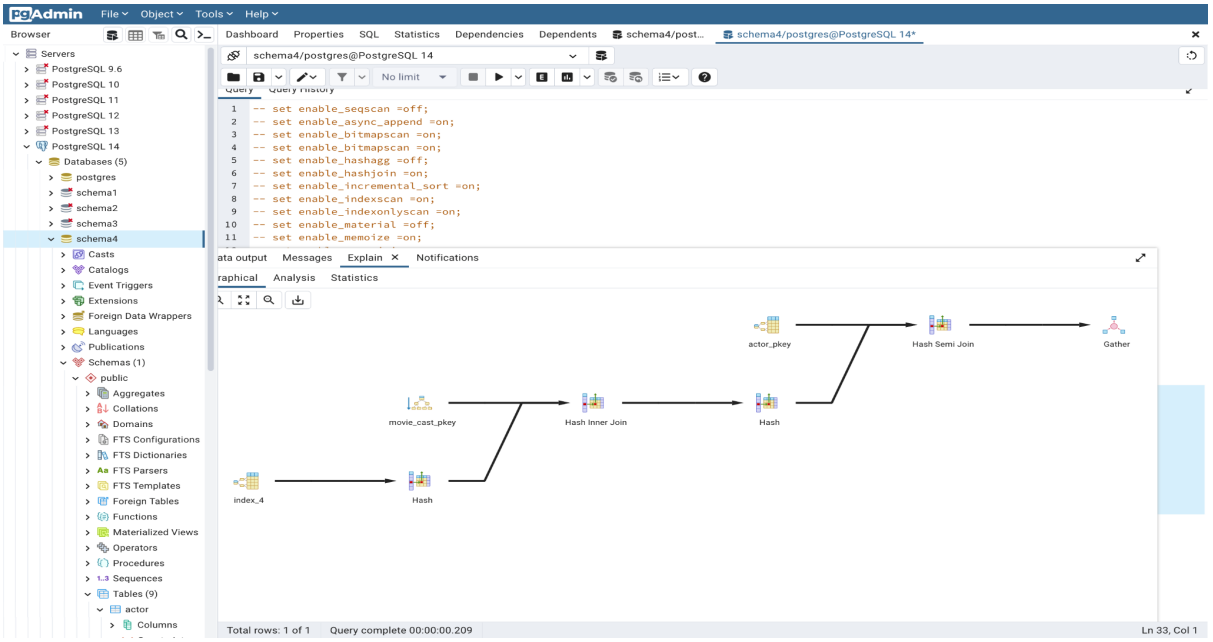
After applying Btree index the cost and execution time has decreased rapidly showing great performance such that:

*execution time became 79.066 and the cost=7380.66

It was made on table movie and column mov_title.



Btree indexing physical plan:

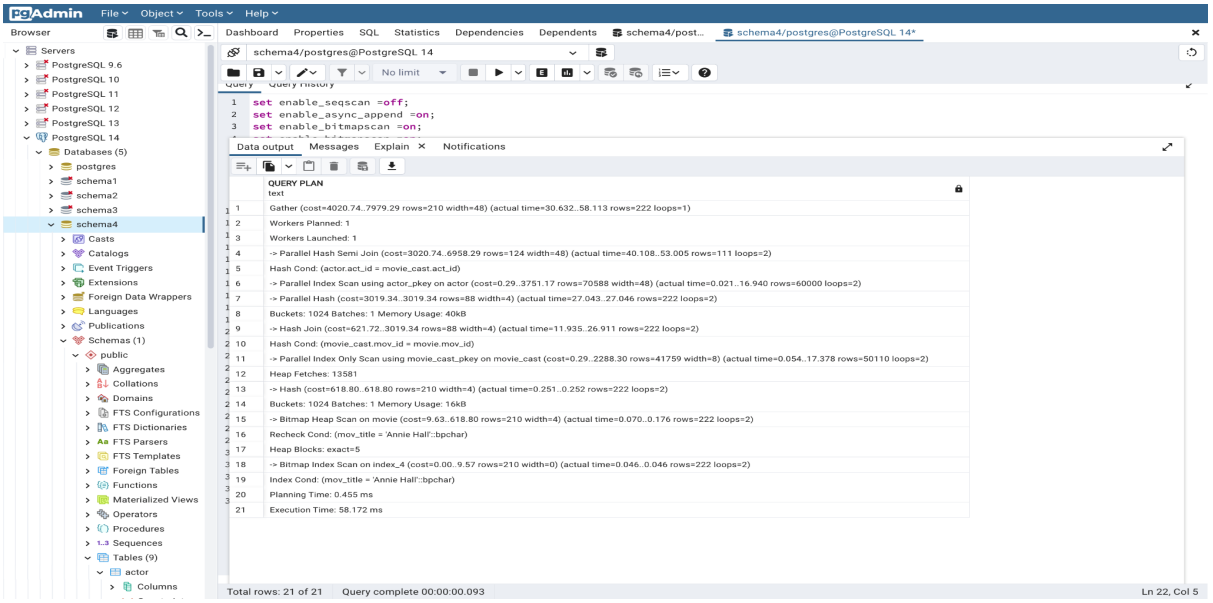


2)<u>Hash indexing:</u>

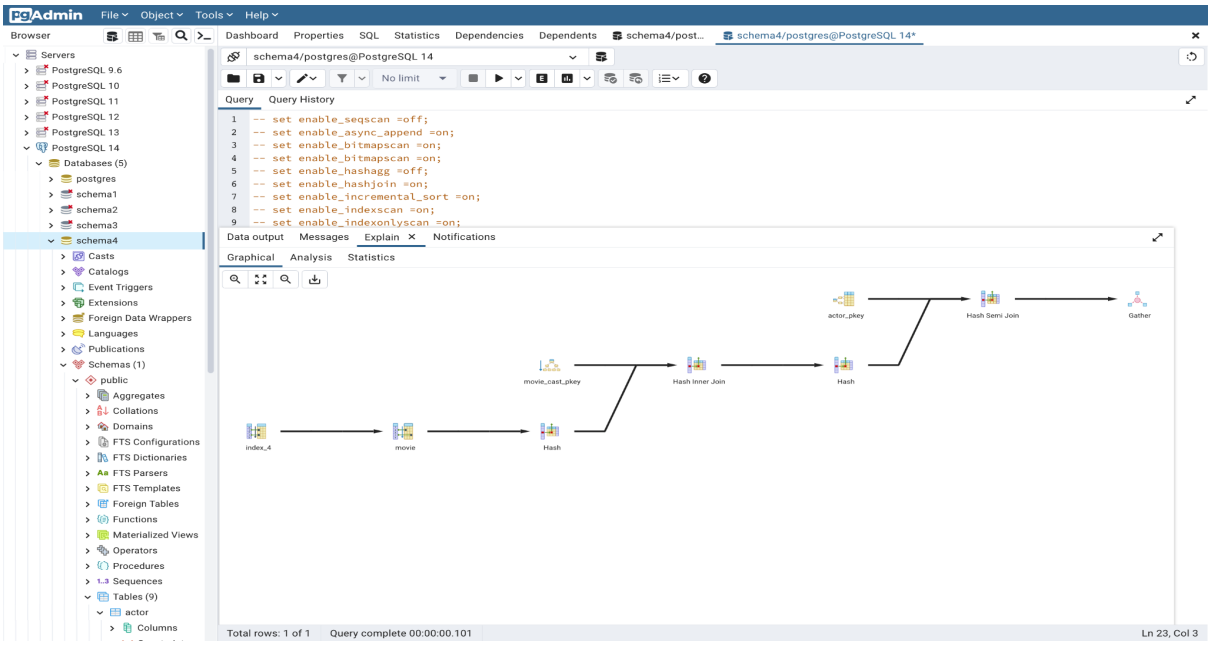
For the hash indexing it reduced the cost and execution time too but not as much as
the Btree as:

*execution time=58.172ms and the cost got reduced to=7979.29

It was made on table movie and column mov_title



Hash index physical plan:

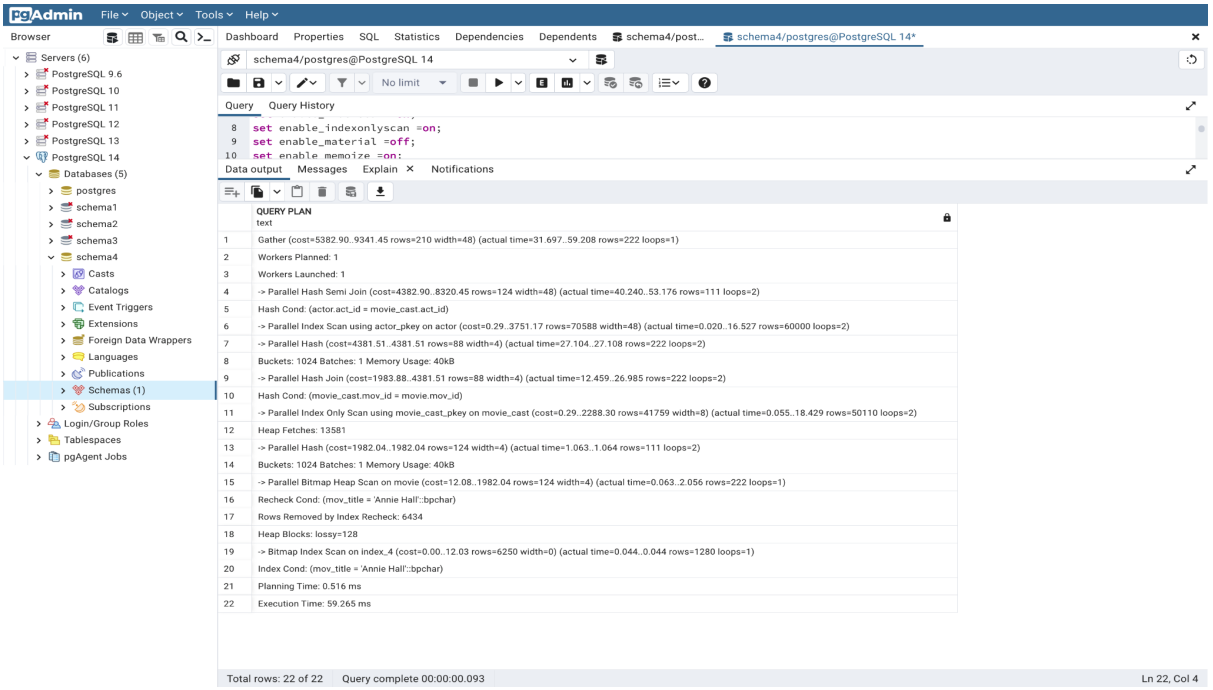


3)<u>Brin indexing:</u>

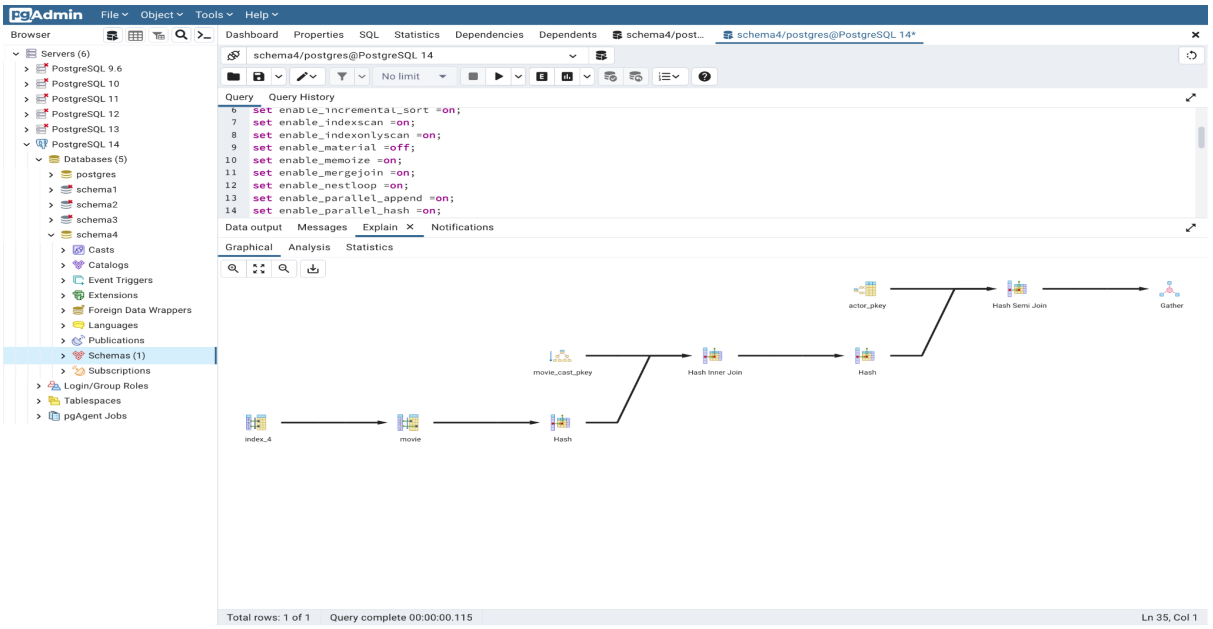
As for Brin the cost and execution time too did get reduced to :

*execution time= 59.265 and with a cost=9341.45

It was made on a table movie and column mov_title.



Brin indexing physical plan:

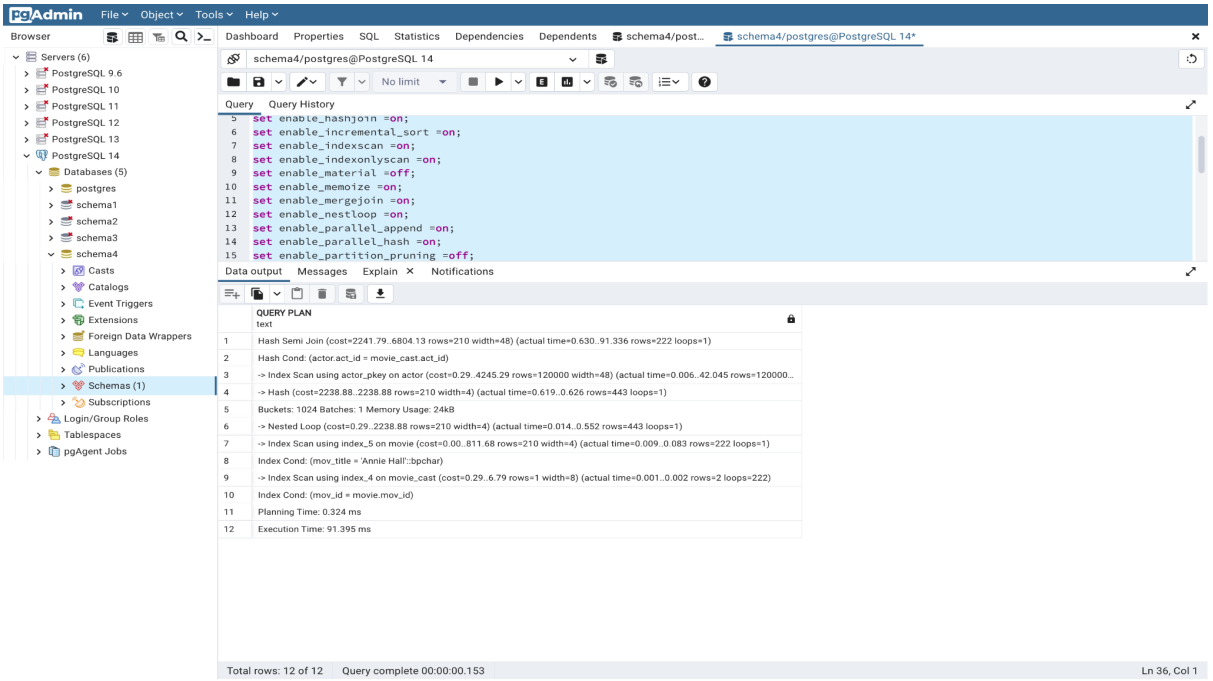


4)<u>Mixed indexing:</u>

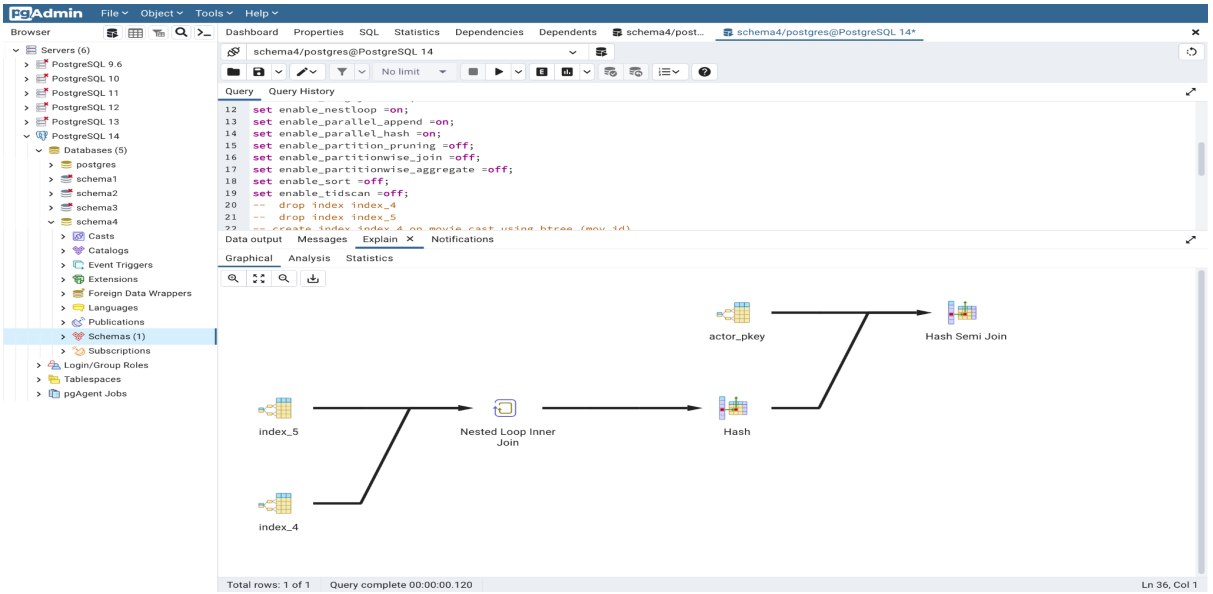
As for mixed the cost and execution time too did get reduced to :

*execution time= 91.395 and with a cost=6804.13

It was made on a table movie using hash index and column mov_title and using movie_cast and column mov_id.



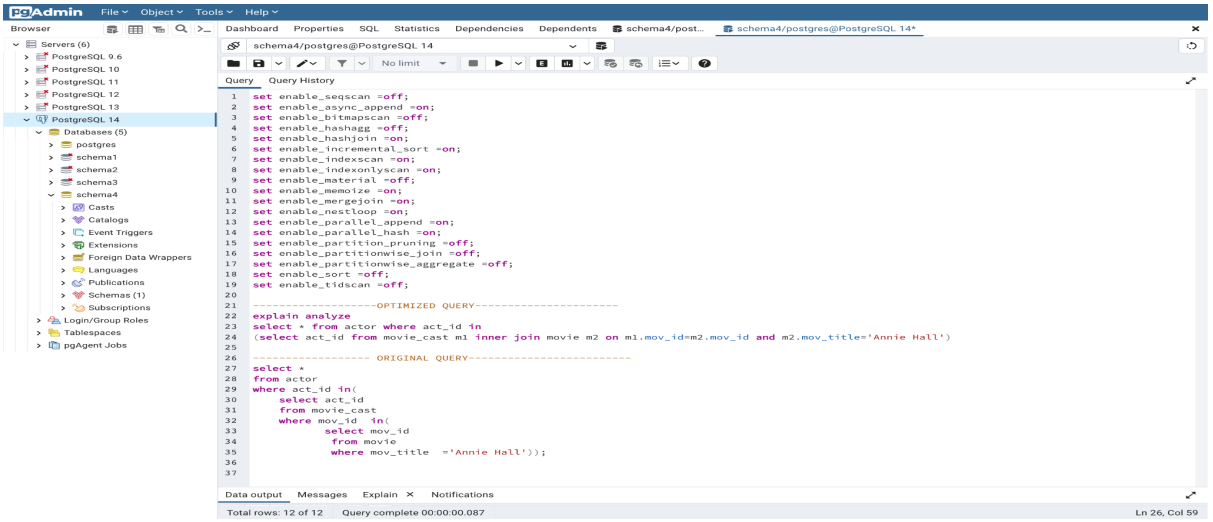
Mixed index physical plan:



5)Optimised Query:

For this part, we tried to optimise the query such that the optimised one should give the same result but in lesser cost and execution time. And after optimising it, it did reduce both with a better performance.

This is the optimised vs the original query both written down:



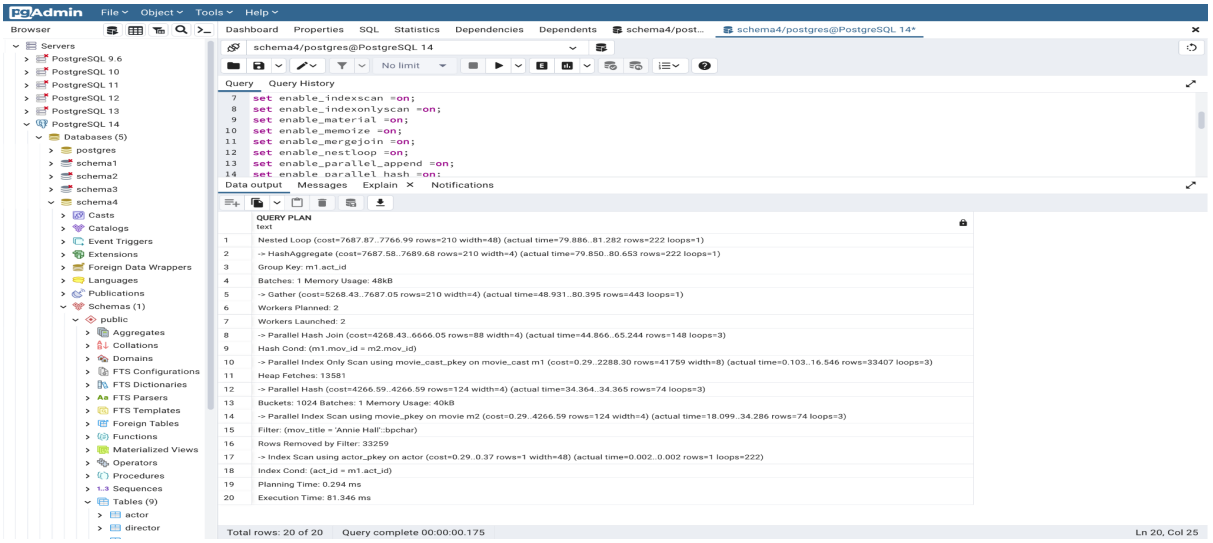
```
set enable_seqscan =off;
set enable_async_append =on;
set enable_bitmapscan =off;
set enable_hashagg =off;
set enable_hashjoin =on;
set enable_incremental_sort =on;
set enable_indexscan =on;
set enable_indexonlyscan =on;
set enable_material =off;
set enable_memoize =on;
set enable_mergejoin =on;
set enable_nestloop =on;
set enable_parallel_append =on;
set enable_parallel_hash =on;
set enable_partition_pruning =off;
set enable_partitionwise_join =off;
set enable_partitionwise_aggregate =off;
set enable_sort =off;
set enable_tidscan =off;

--------------------OPTIMIZED QUERY---------------------
explain analyze
select * from actor where act_id in
(select act_id from movie_cast m1 inner join movie m2 on m1.mov_id=m2.mov_id and m2.mov_title='Annie Hall')

------------------ ORIGINAL QUERY-------------------------
select *
from actor
where act_id in(
    select act_id
    from movie_cast
    where mov_id  in(
            select mov_id
             from movie
             where mov_title  ='Annie Hall'));
```

The costs of the optimised one:

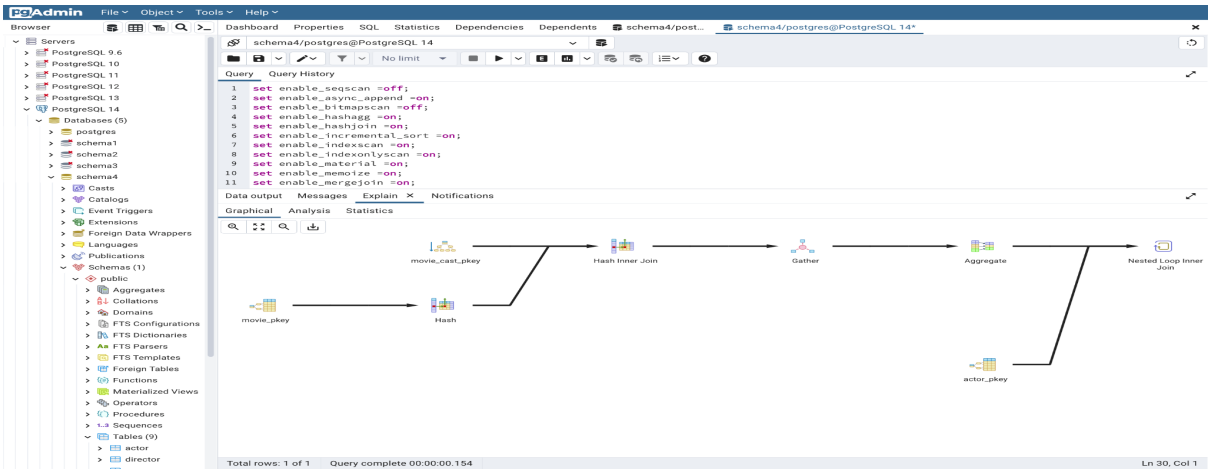
*execution time=81.346 and the cost=7766.99



```
set enable_indexscan =on;
set enable_indexonlyscan =on;
set enable_material =on;
set enable_memoize =on;
set enable_mergejoin =on;
set enable_nestloop =on;
set enable_parallel_append =on;
set enable_parallel_hash =on;
```

QUERY PLAN text
1 Nested Loop (cost=7687.87..7766.99 rows=210 width=48) (actual time=79.886..81.282 rows=222 loops=1)
2 -> HashAggregate (cost=7687.58..7689.68 rows=210 width=4) (actual time=79.850..80.653 rows=222 loops=1)
3 Group Key: m1.act_id
4 Batches: 1 Memory Usage: 48kB
5 -> Gather (cost=5268.43..7687.05 rows=210 width=4) (actual time=48.931..80.395 rows=443 loops=1)
6 Workers Planned: 2
7 Workers Launched: 2
8 -> Parallel Hash Join (cost=4268.43..6666.05 rows=88 width=4) (actual time=44.866..65.244 rows=148 loops=3)
9 Hash Cond: (m1.mov_id = m2.mov_id)
10 -> Parallel Index Only Scan using movie_cast_pkey on movie_cast m1 (cost=0.29..2288.30 rows=41759 width=8) (actual time=0.103..16.546 rows=33407 loops=3)
11 Heap Fetches: 13581
12 -> Parallel Hash (cost=4266.59..4266.59 rows=124 width=4) (actual time=34.364..34.365 rows=74 loops=3)
13 Buckets: 1024 Batches: 1 Memory Usage: 40kB
14 -> Parallel Index Scan using movie_pkey on movie m2 (cost=0.29..4266.59 rows=124 width=4) (actual time=18.099..34.286 rows=74 loops=3)
15 Filter: (mov_title = 'Annie Hall'::bpchar)
16 Rows Removed by Filter: 33259
17 -> Index Scan using actor_pkey on actor (cost=0.29..0.37 rows=1 width=48) (actual time=0.002..0.002 rows=1 loops=222)
18 Index Cond: (act_id = m1.act_id)
19 Planning Time: 0.294 ms
20 Execution Time: 81.346 ms

And the physical plan of the optimised query:

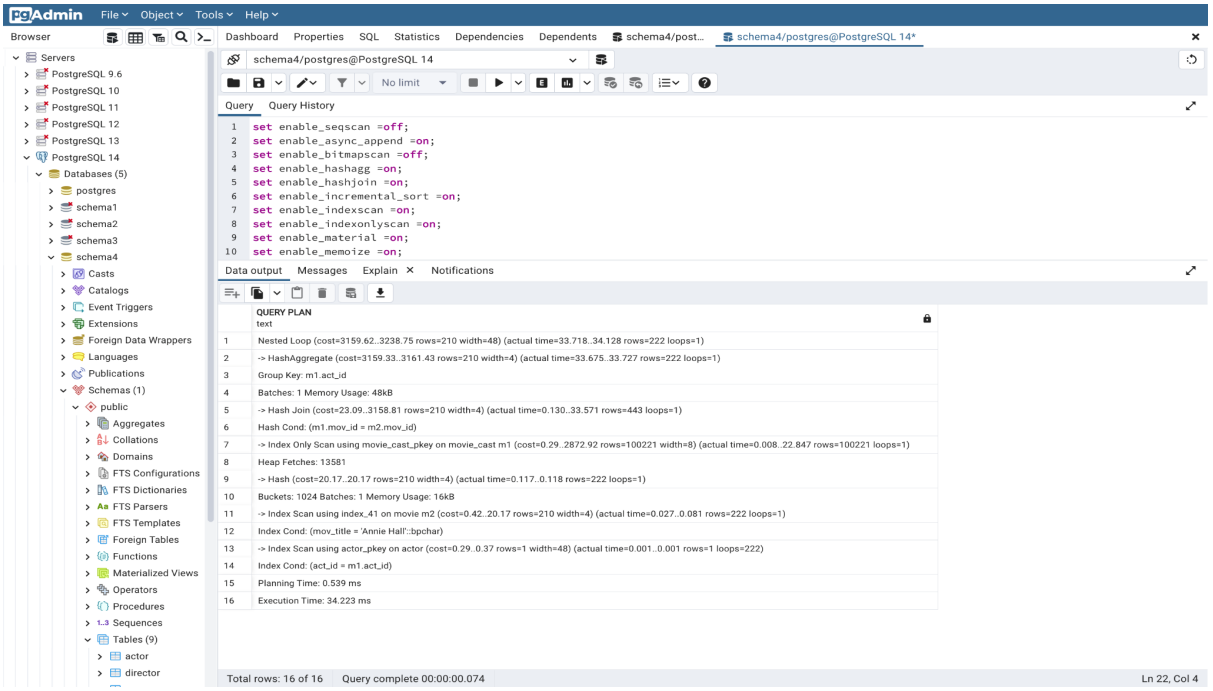


6)Btree Optimised Query:

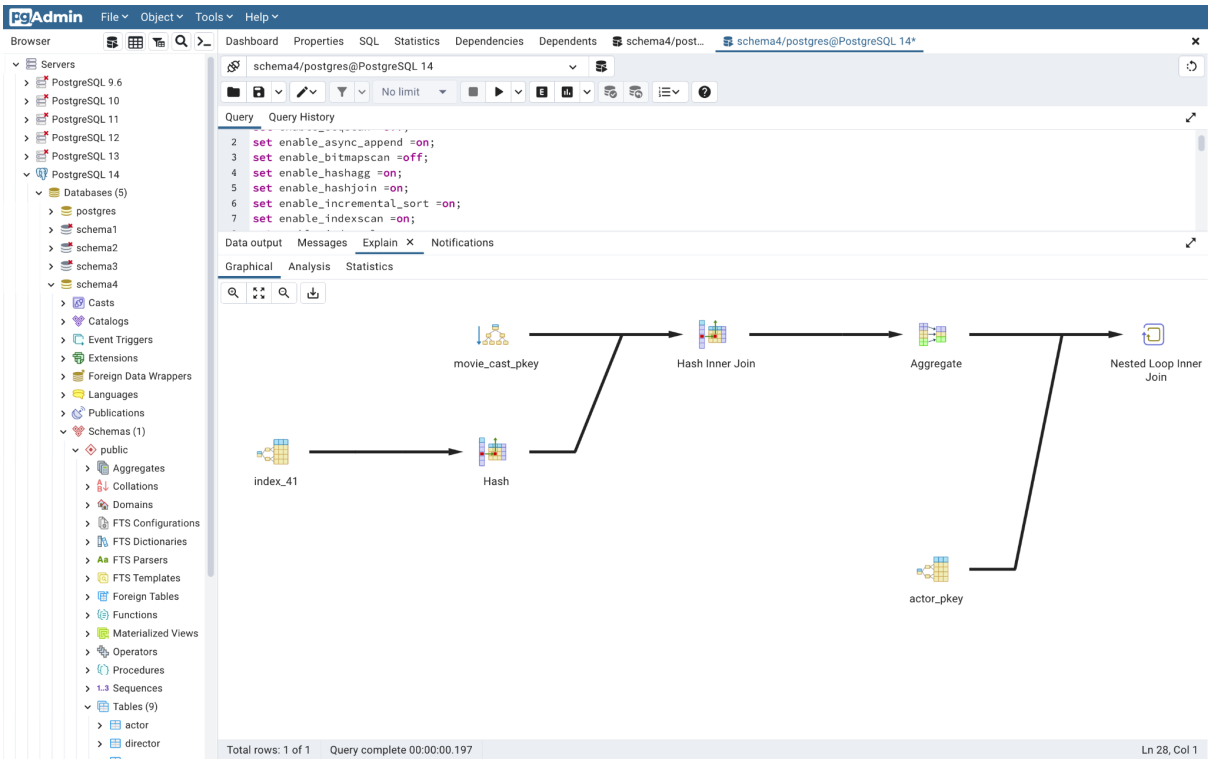
After applying Btree index on the optimised one the execution time and cost got reduced even more:

*execution time=34.223 and the cost=3238.75

Applied on table movie and column mov_title.



Optimised Btree physical plan:

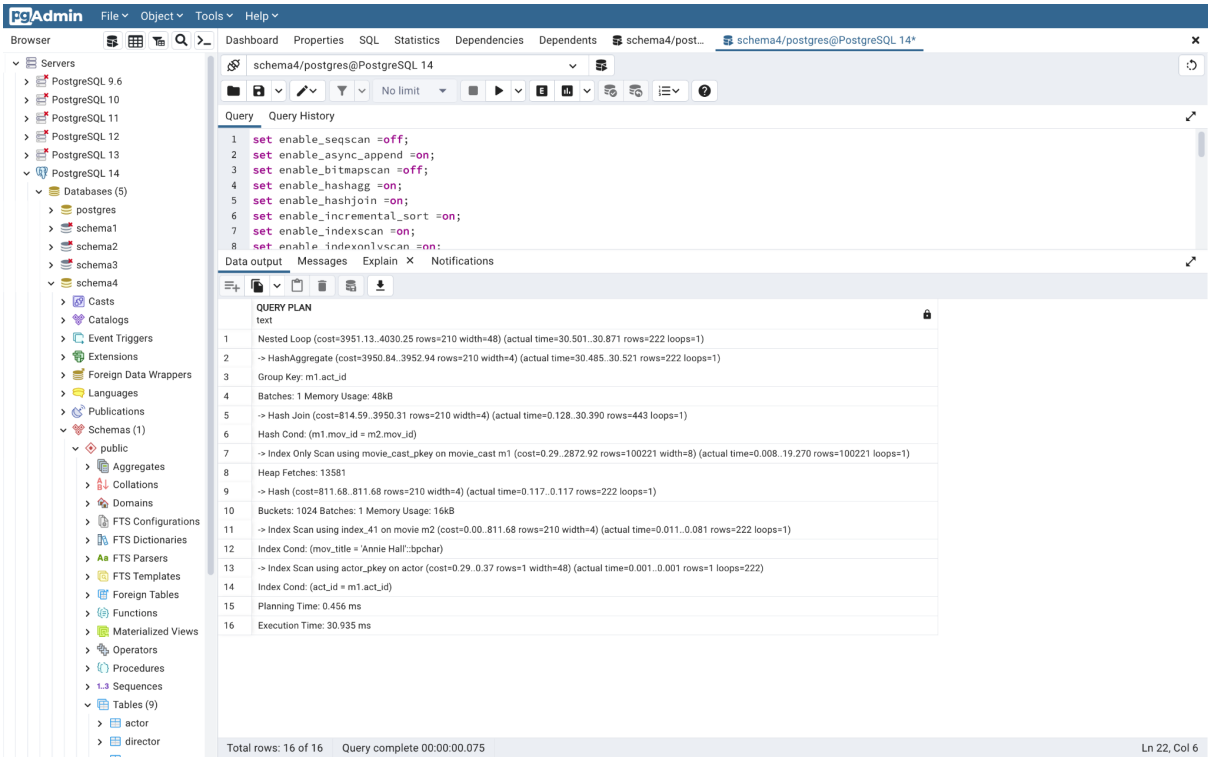


7)Hash Optimised Query:

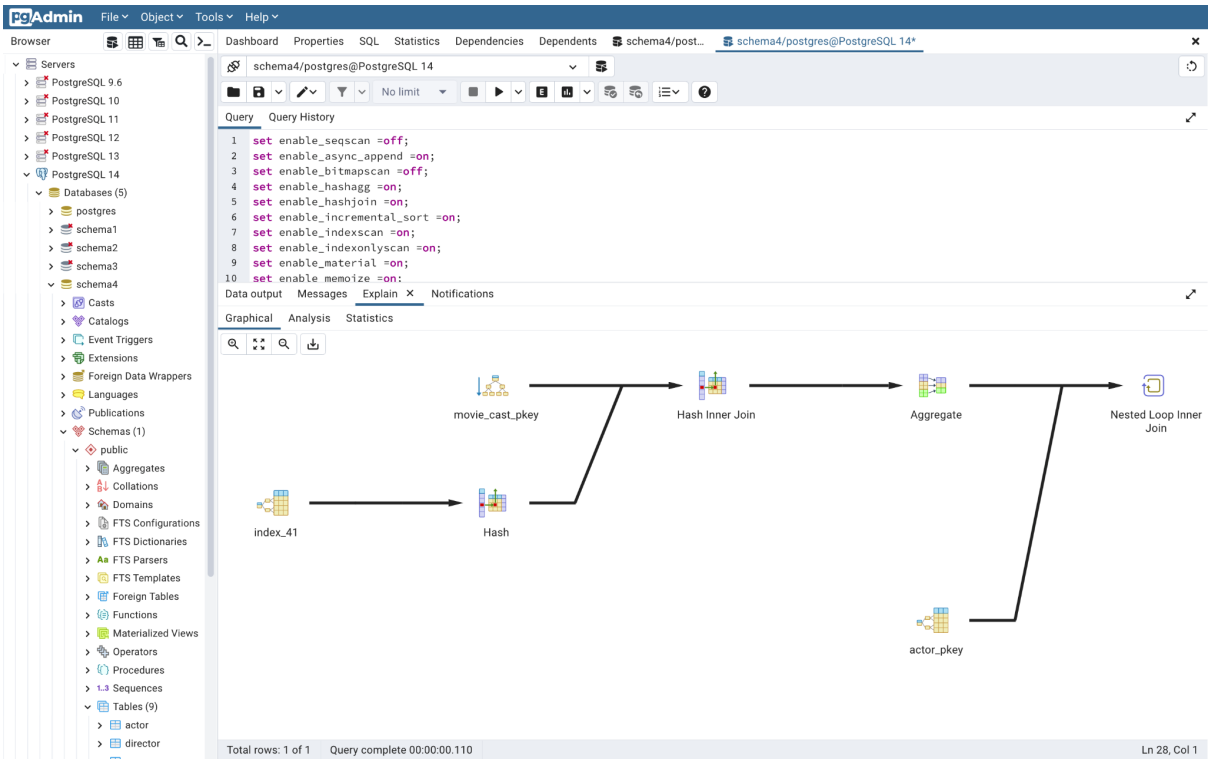
After applying hash index on the optimised one the execution time and cost got reduced too:

*execution time=30.935 and the cost=4030.25

Applied on table movie and column mov_title.



Optimised hash physical plan:

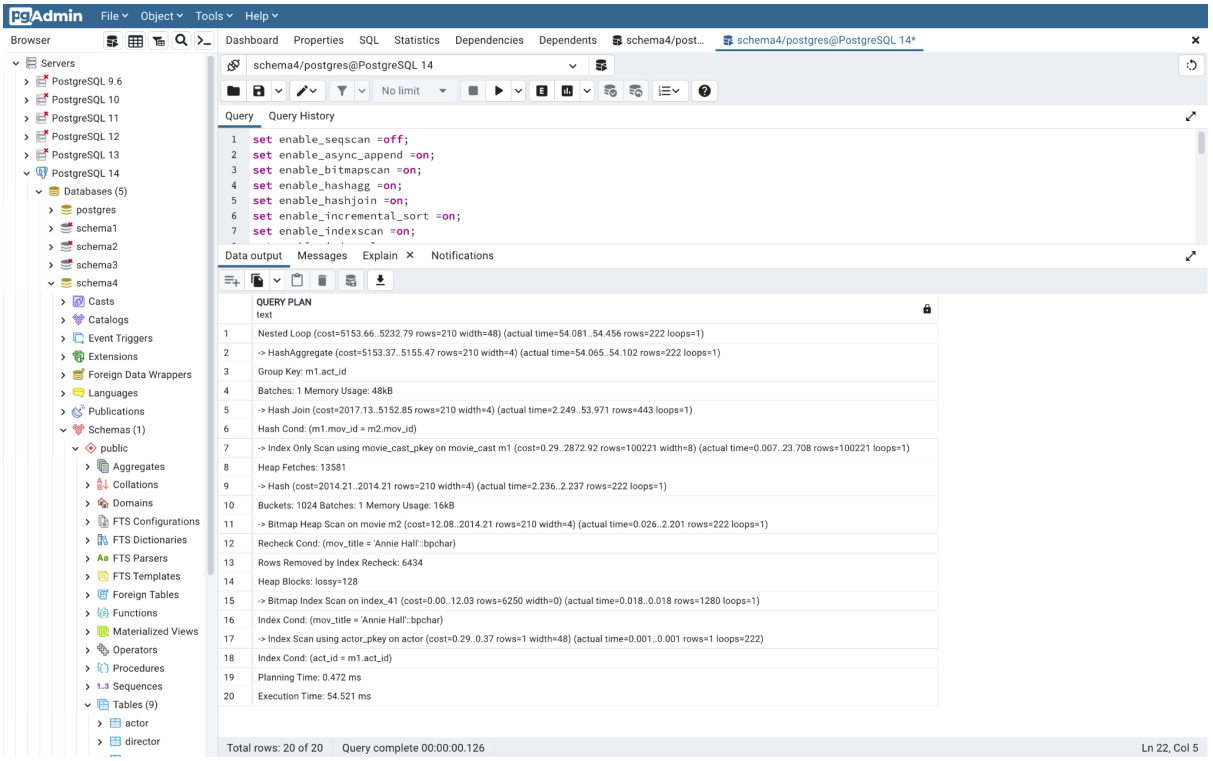


8)Brin Optimised Query:

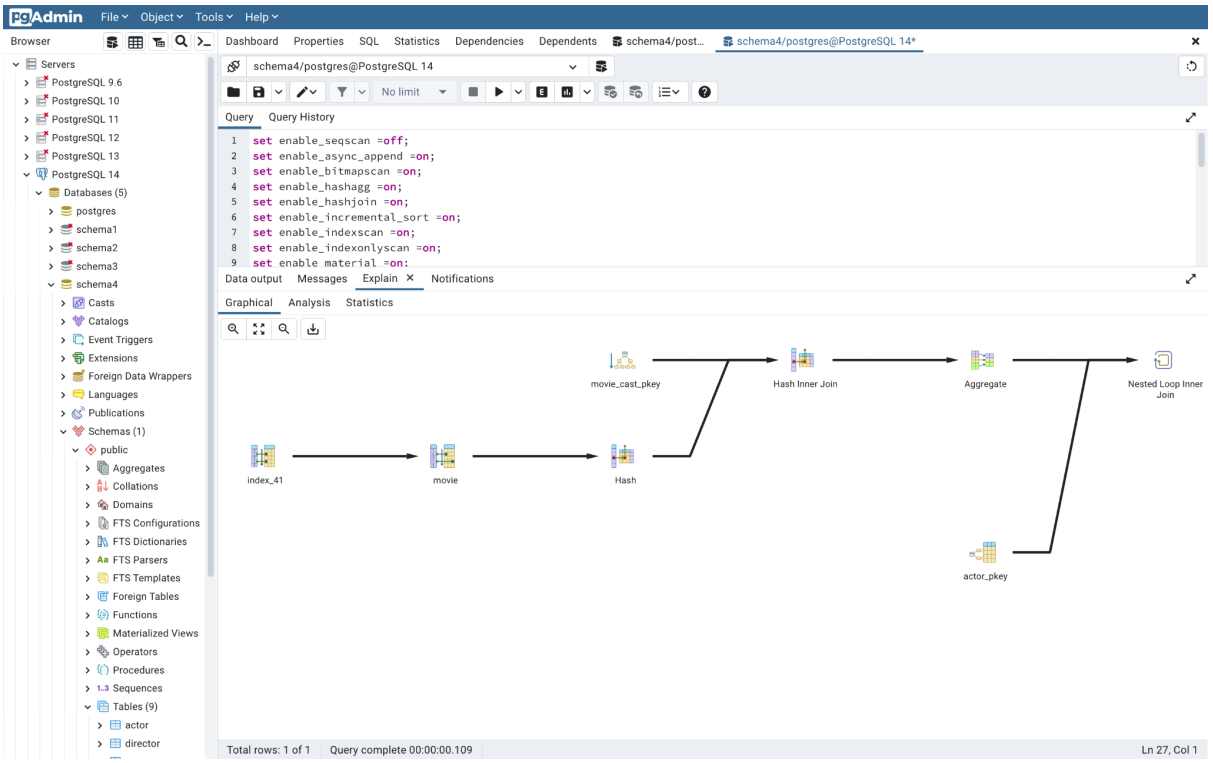
After applying Brin index on the optimised one the execution time and cost got reduced:

*execution time=54.521ms and the cost=5232.79

Applied on table movie and column mov_title.



Optimised Brin physical plan:

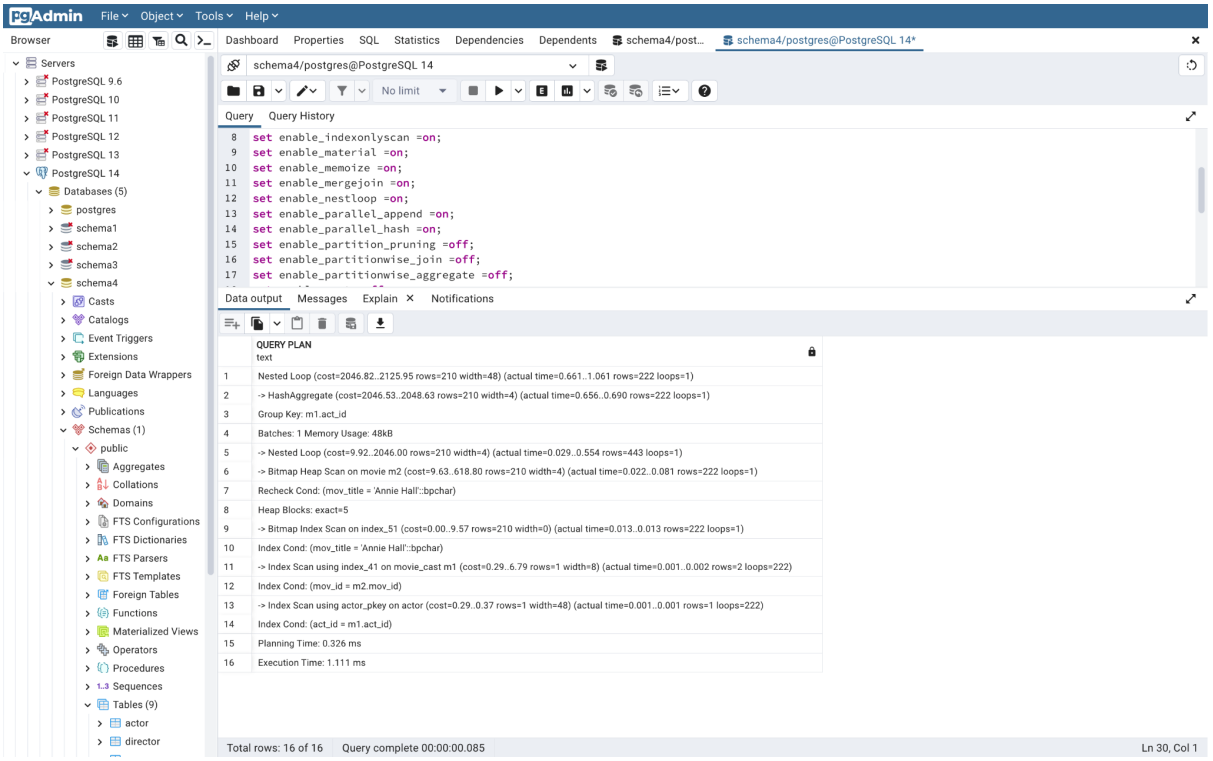


9)<u>Mixed Optimised Query:</u>

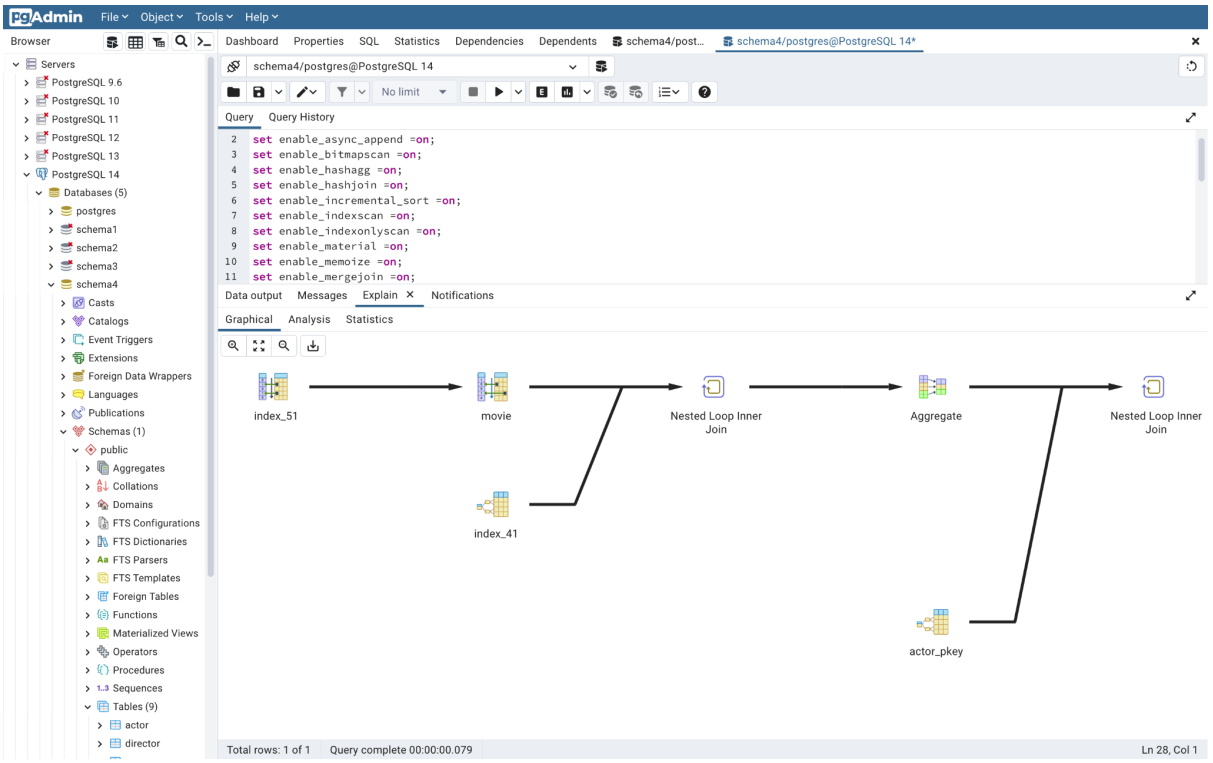
After applying Mixed index on the optimised one the execution time and cost got reduced:

*execution time=1.111ms and the cost=2125.95

Applied on table movie_cast using Btree with column mov_id and table movie suing hash with column mov_title



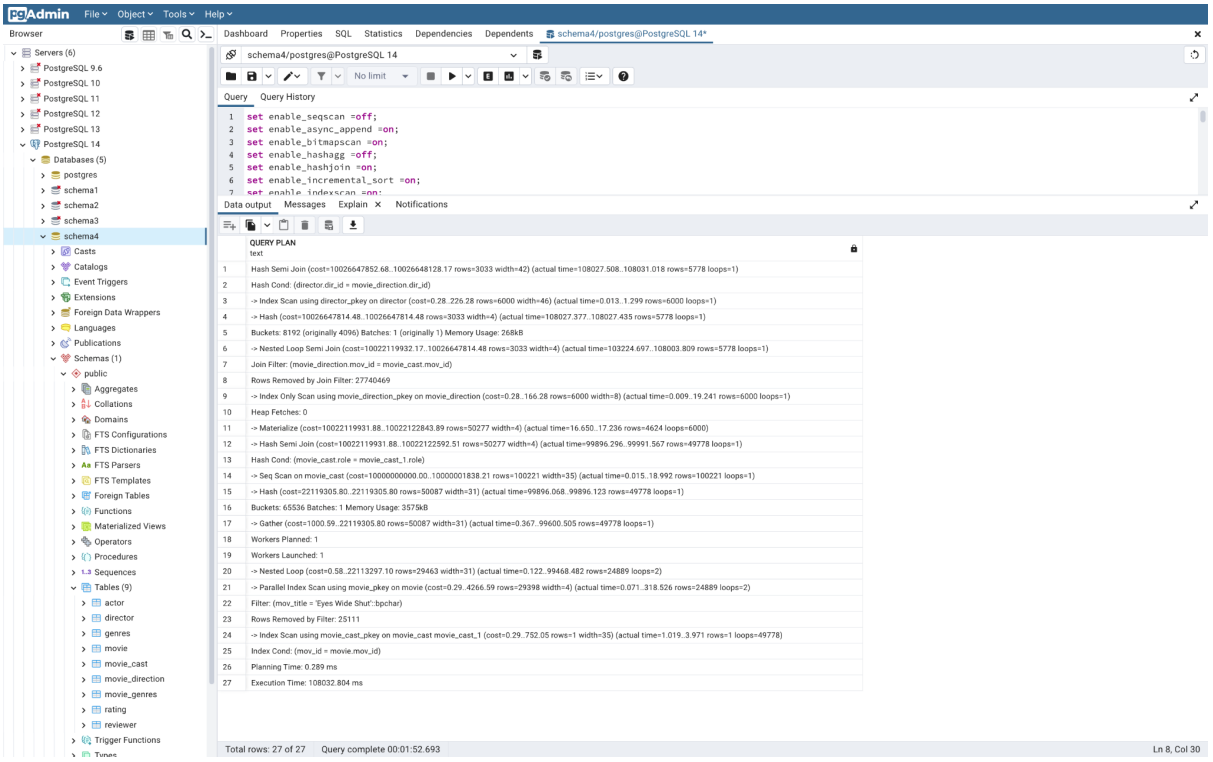
Optimised mixed physical plan:



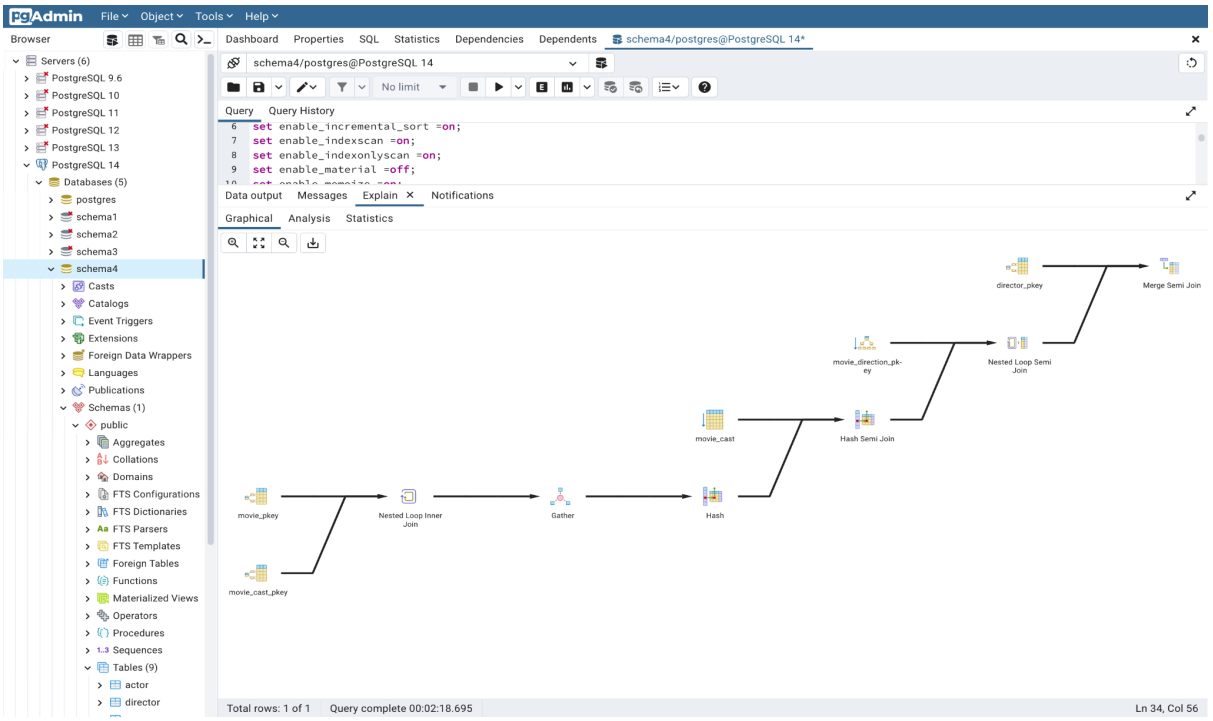
Query11

For this query, I adjusted the flags in order to find the best cost so that it could be reflected on the optimization process and the flags enabled/disabled are not like the other queries.

At first the execution time=108032.804ms and the cost was 10026648128.17 and we'll do the same as the other queries.



And the physical plan was:

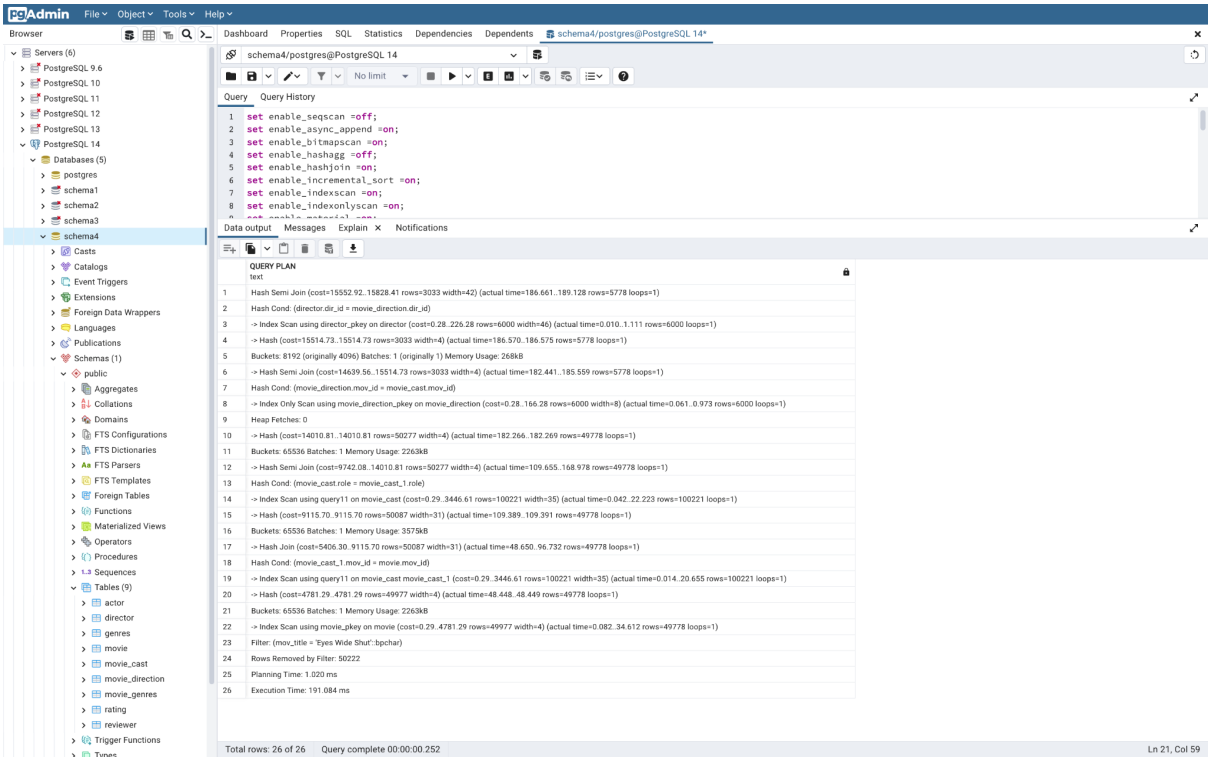


1)Btree indexing:

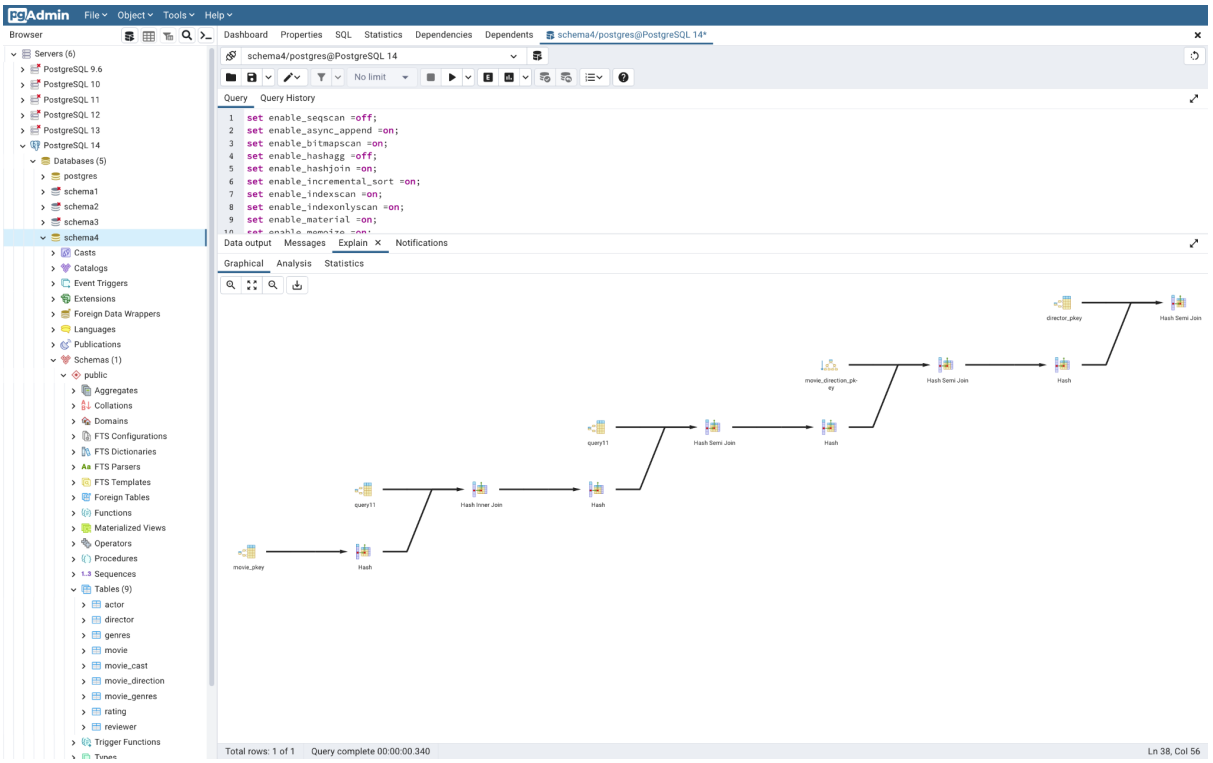
After applying Btree index the cost and execution time has decreased rapidly showing great performance such that:

*execution time became 191.084ms and the cost=15828.41

It was applied on table movie_cast and column mov_id.



Btree physical plan:

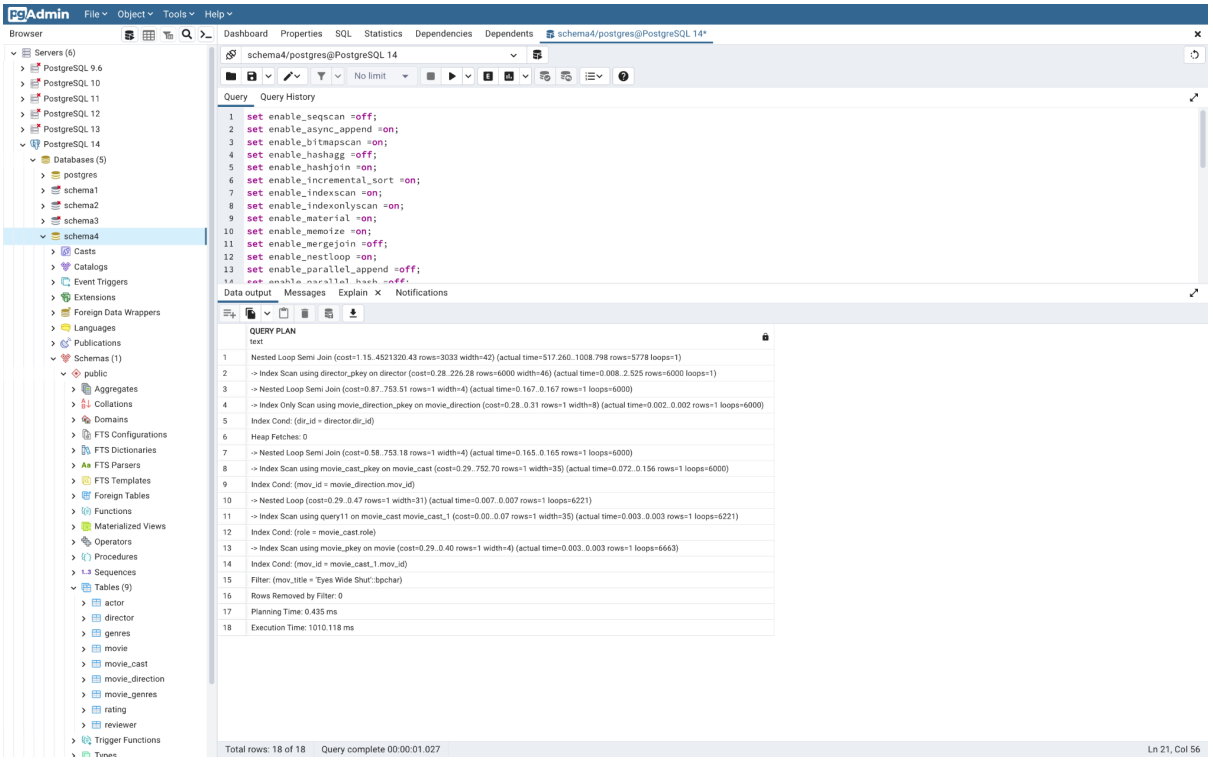


2)Hash indexing:

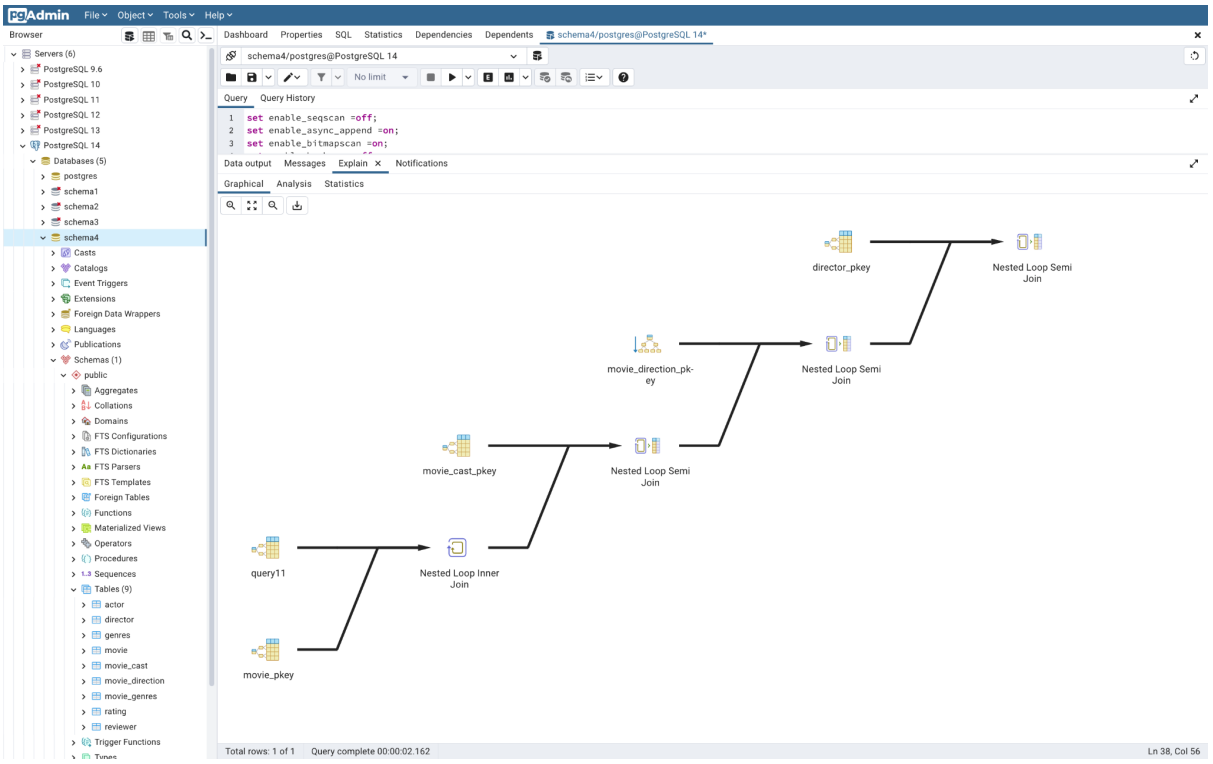
For the hash indexing it reduced the cost and execution time too but not as much as
the Btree as:

*execution time=1010.118ms and the cost got reduced to=4521320.43

It was made on table movie_cast and column role.



Hash physical plan:

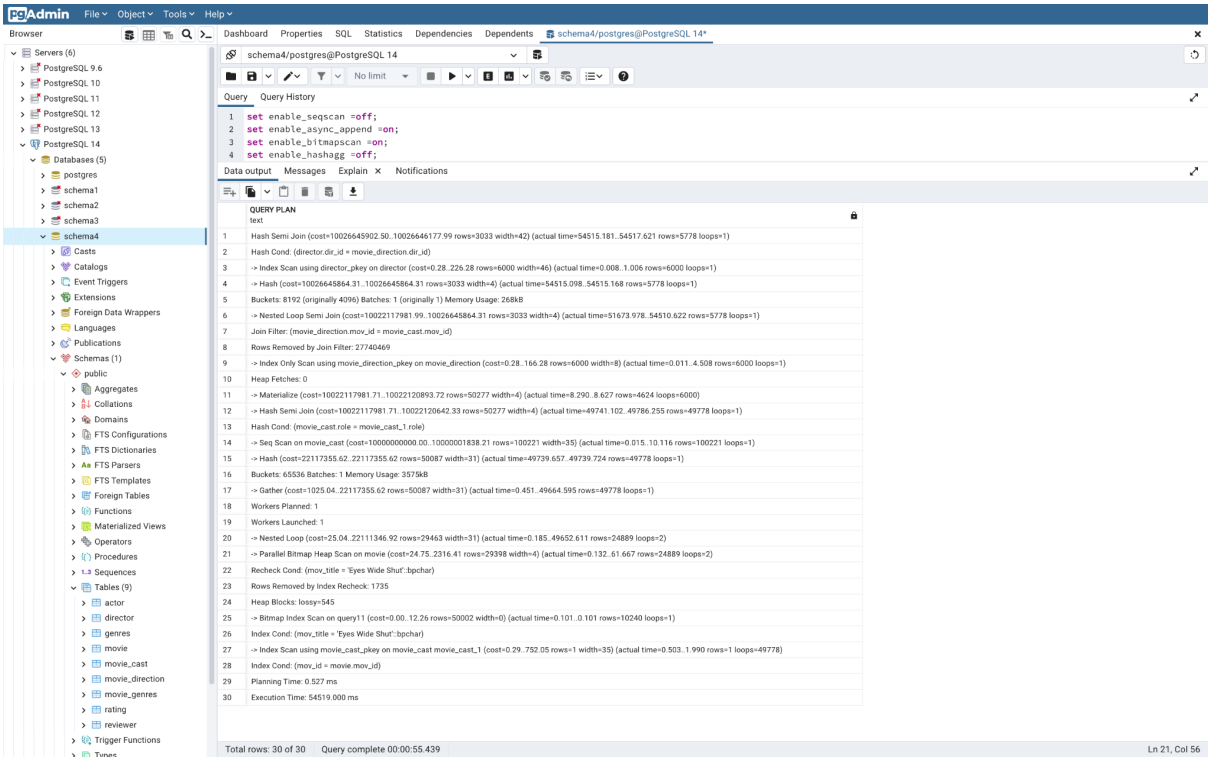


3)Brin indexing:

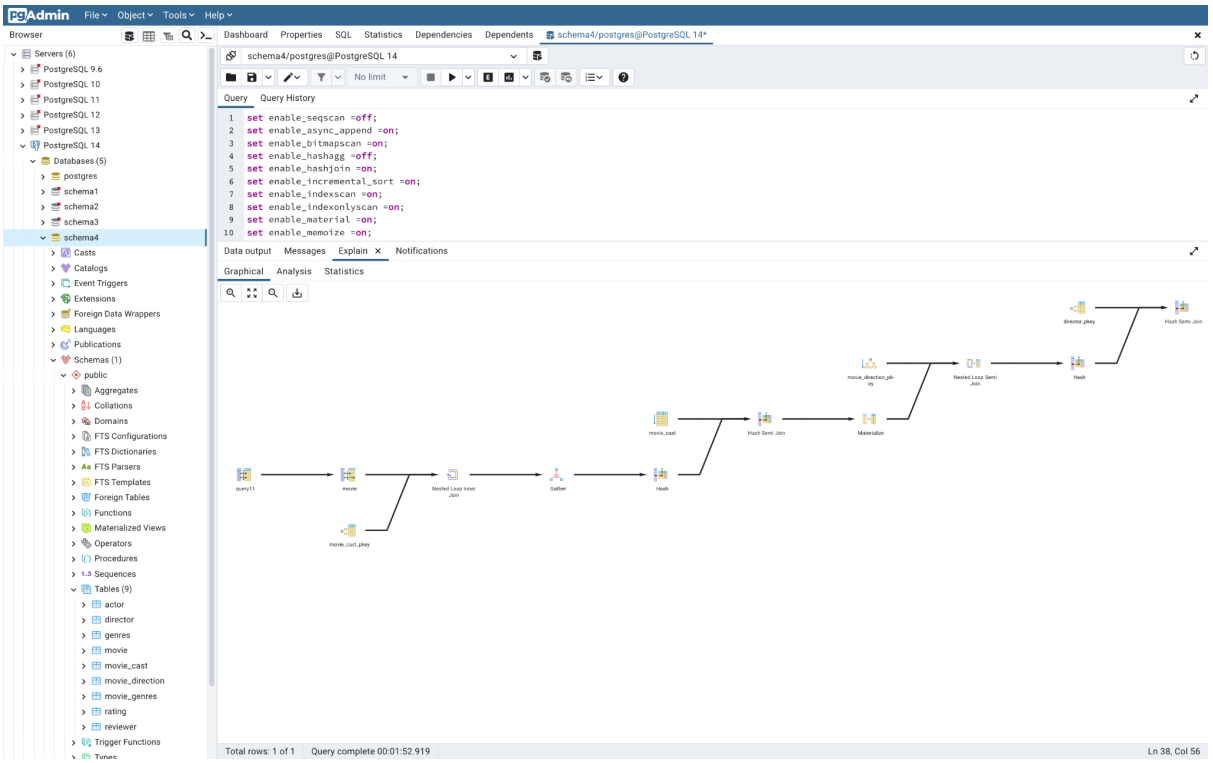
As for Brin the cost and execution time too did get reduced to :

*execution time= 54519.00ms and with a cost=10026646177.99

It was made on a table movie and column mov_title.



Brin physical plan:

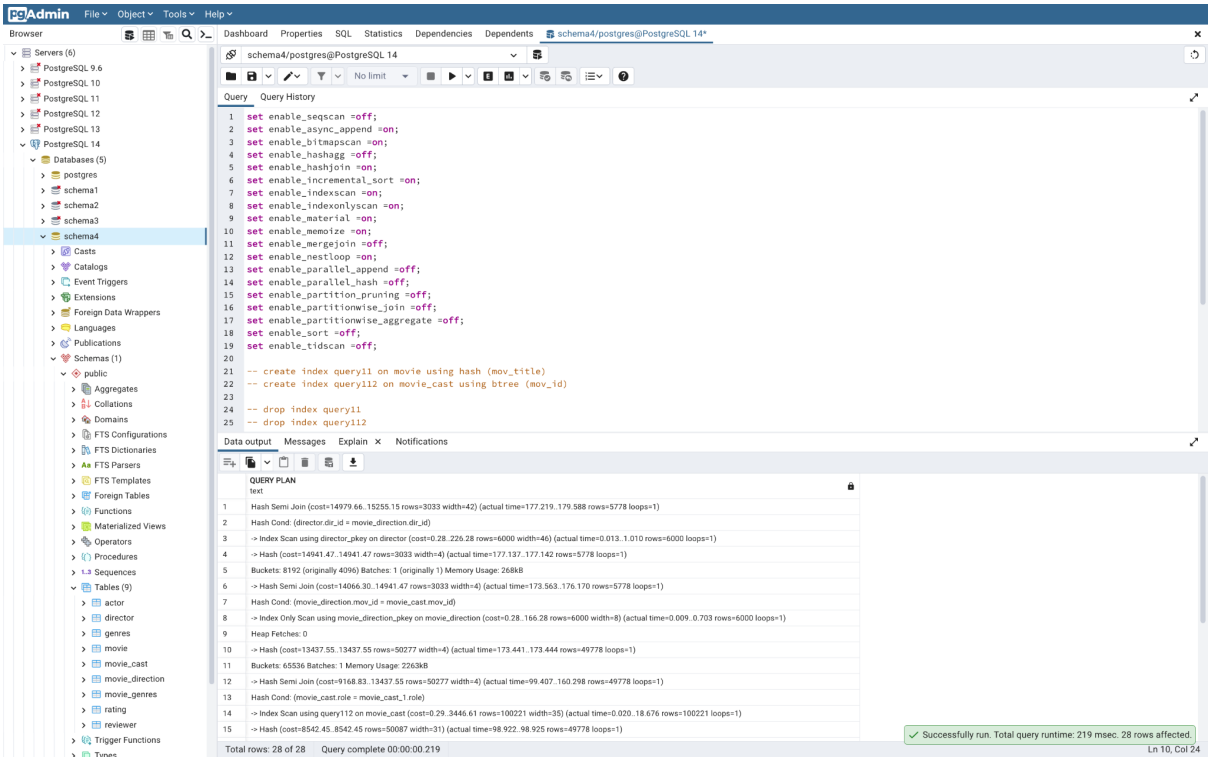


4)<u>Mixed indexing:</u>

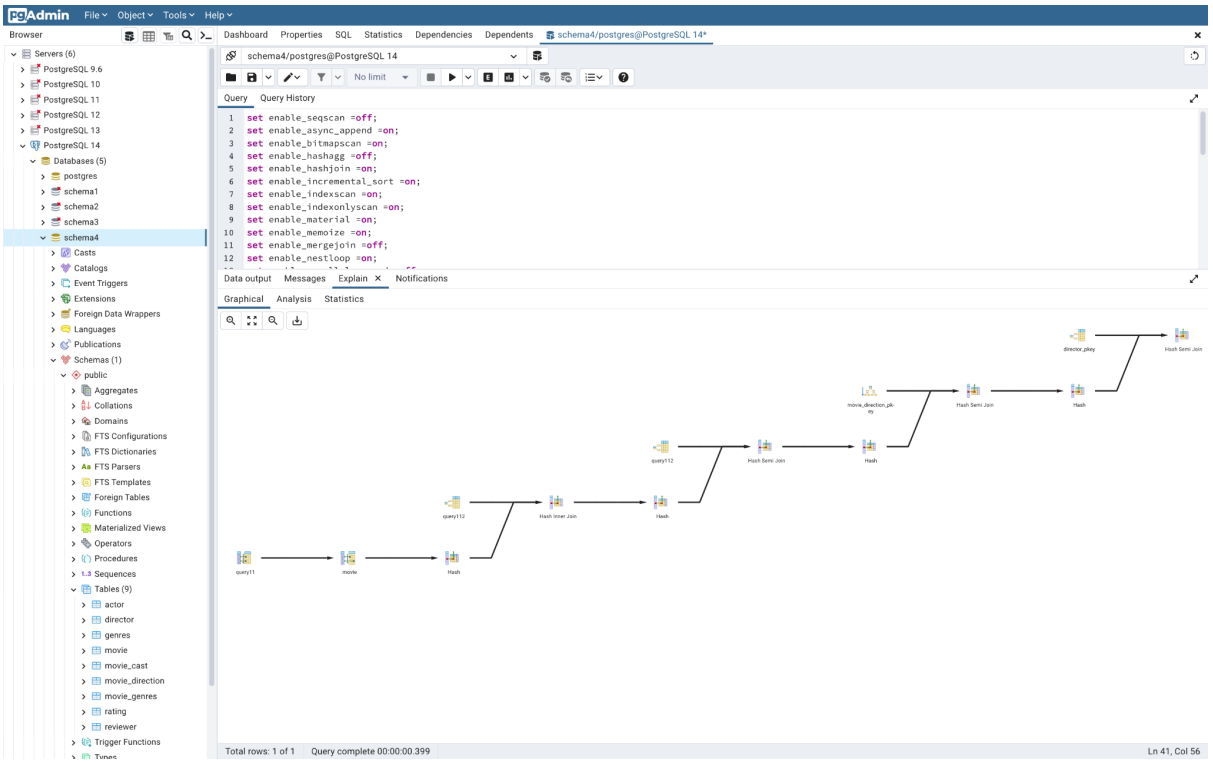
As for mixed the cost and execution time too did get reduced to :

* with a cost=15255.15

It was made on a table movie using hash index and column mov_title and using movie_cast and column mov_id.



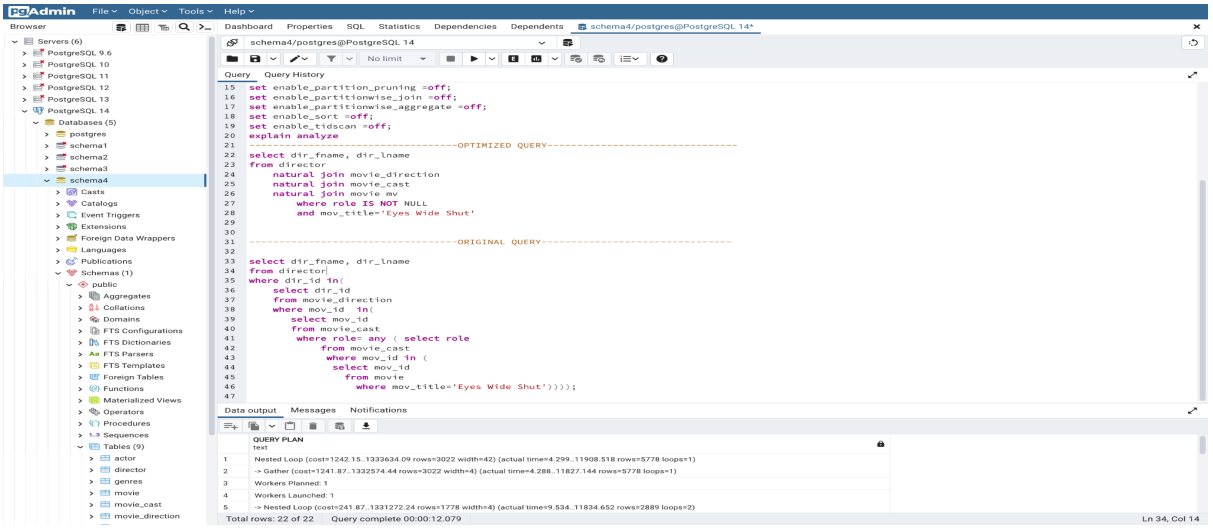
Mixed physical plan:



5)Optimised Query:

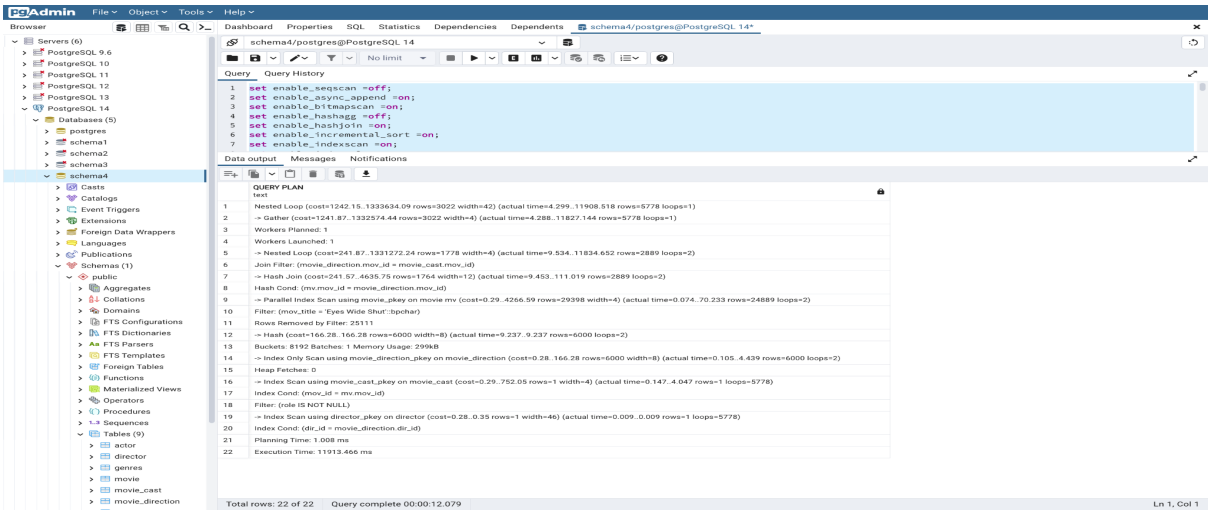
For this part, we tried to optimise the query such that the optimised one should give the same result but in lesser cost and execution time. And after optimising it, it did reduce both with a better performance.

This is the optimised vs the original query both written down:

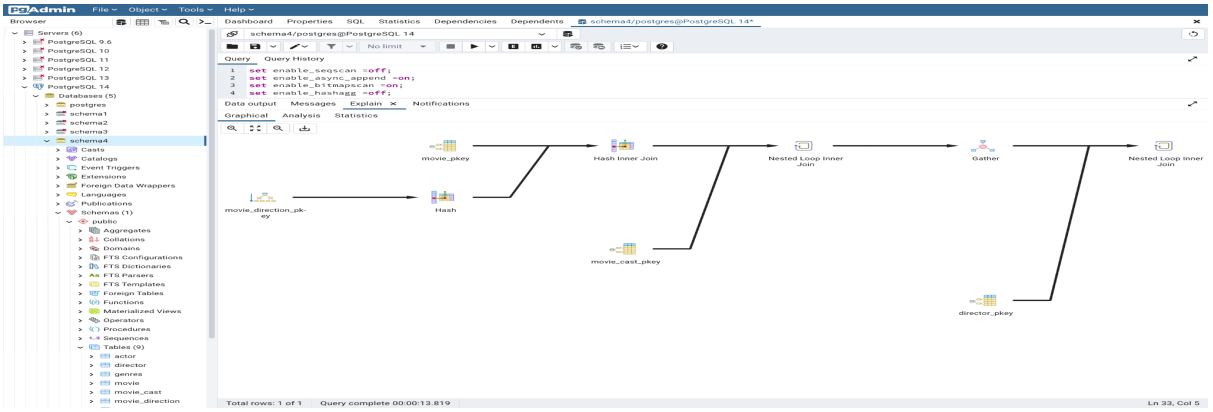


The costs of the optimised one:

*execution time=81.346 and the cost=7766.99



Optimised physical plan:

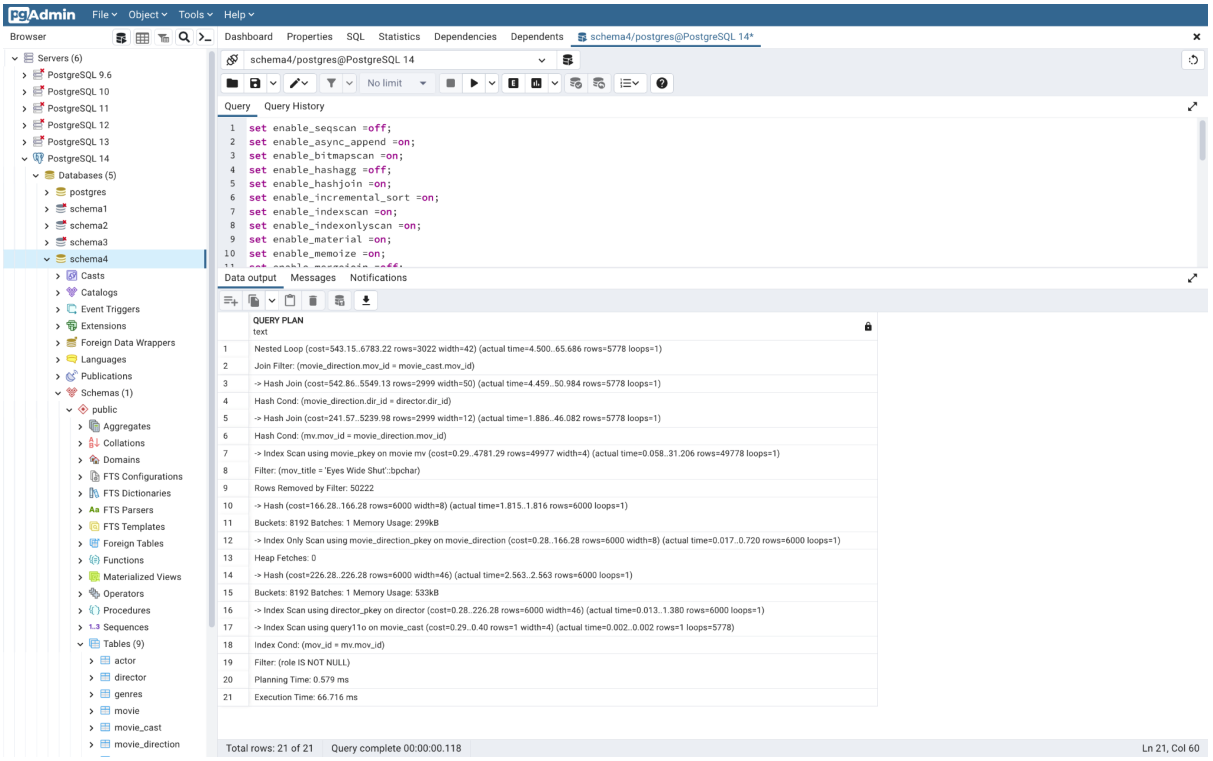


6)Btree Optimised Query:

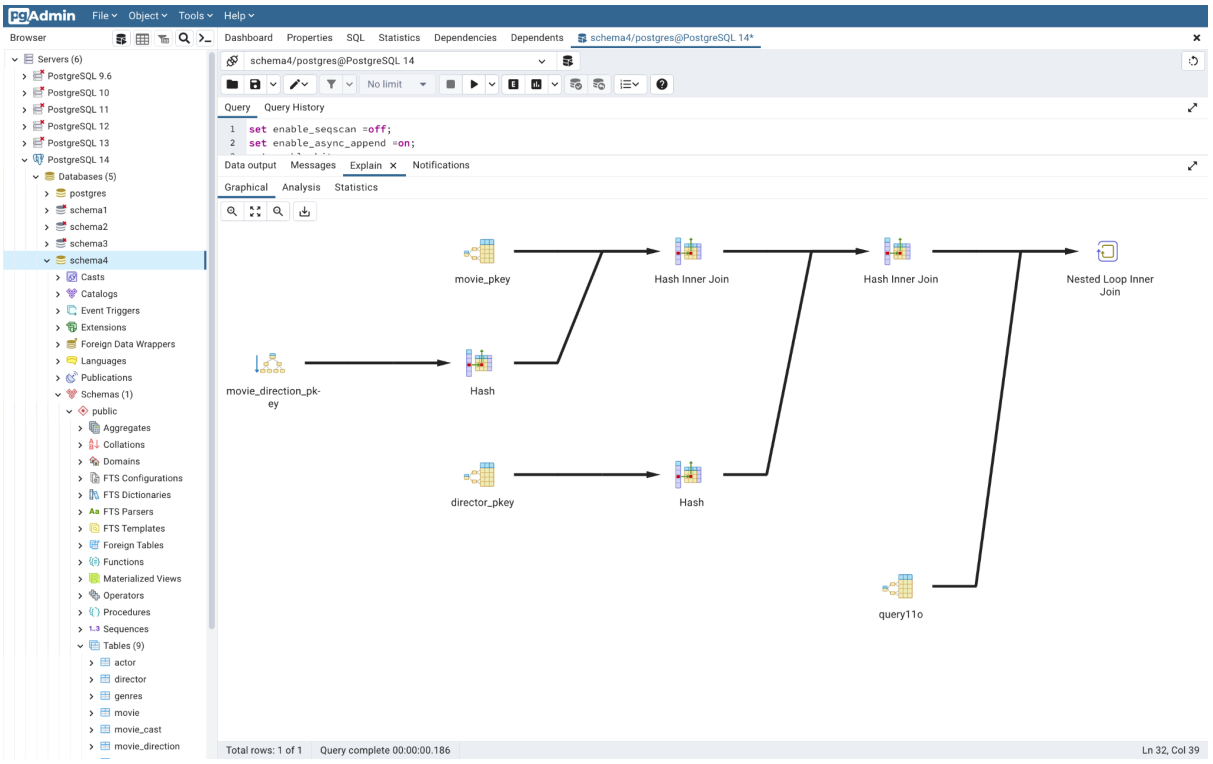
After applying Btree index on the optimised one the execution time and cost got reduced even more:

*execution time=66.716ms and the cost=6783.22

Applied on table movie_cast and column mov_id.



Optimised Btree physical plan:

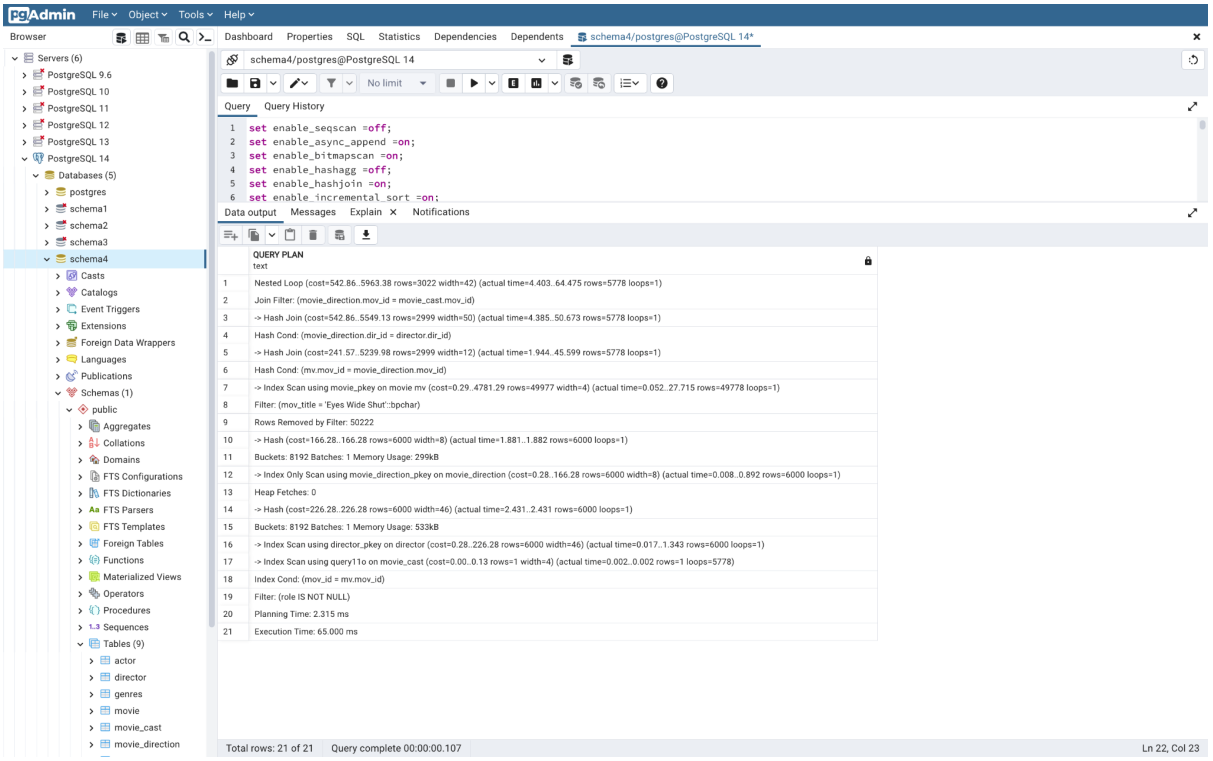


7)Hash Optimised Query:

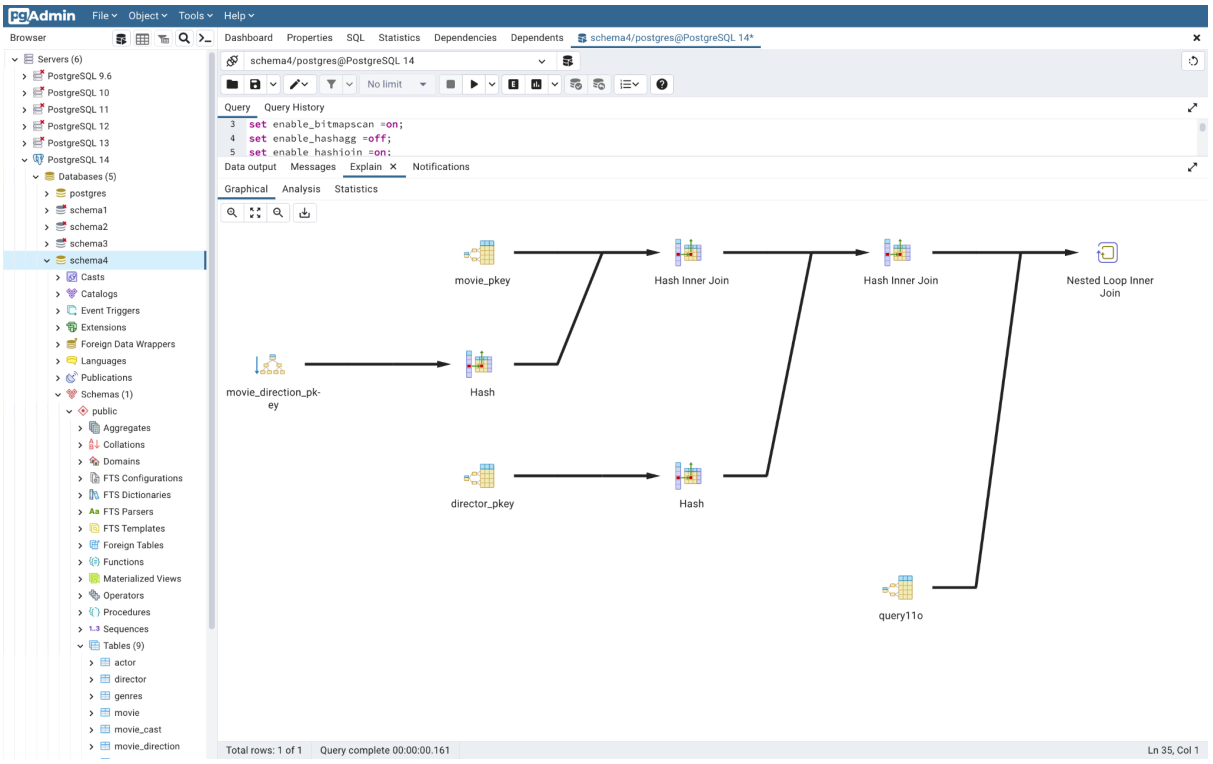
After applying hash index on the optimised one the execution time and cost got reduced too:

*execution time=65.00ms and the cost=5963.38

Applied on table movie_cast and column mov_id.



Optimised hash physical plan:

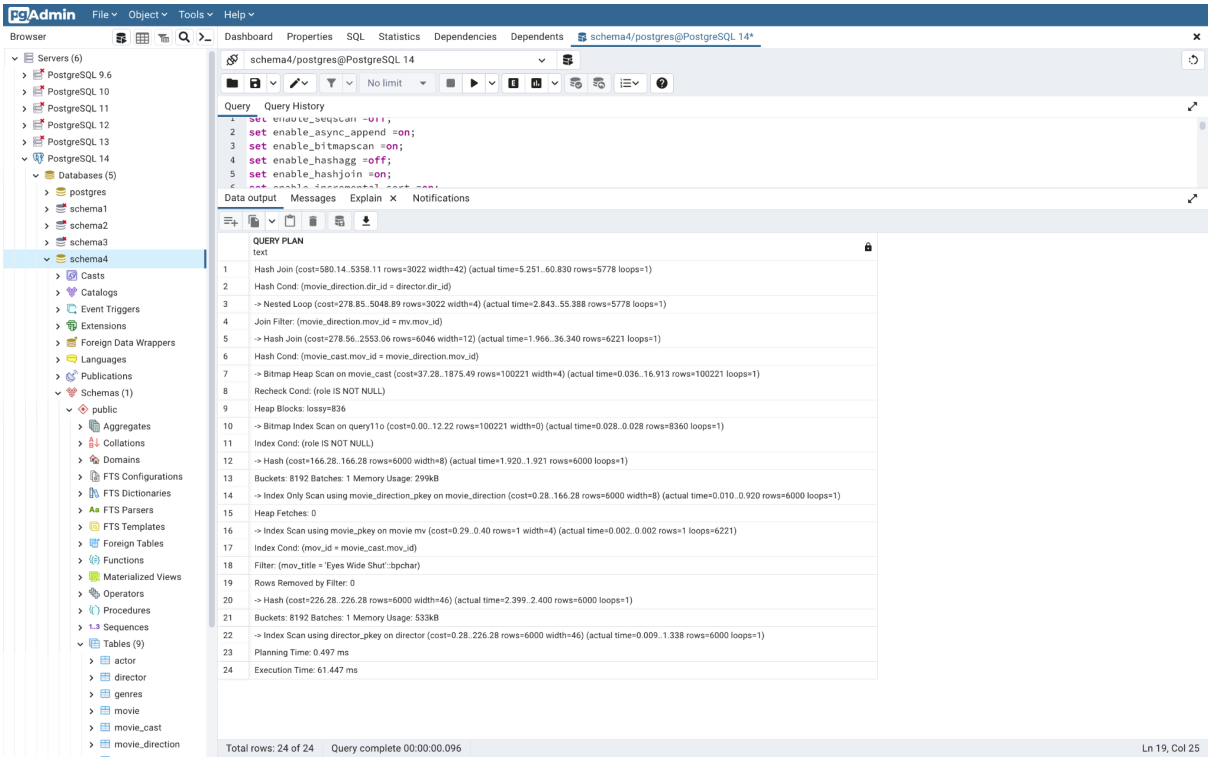


8)Brin Optimised Query:

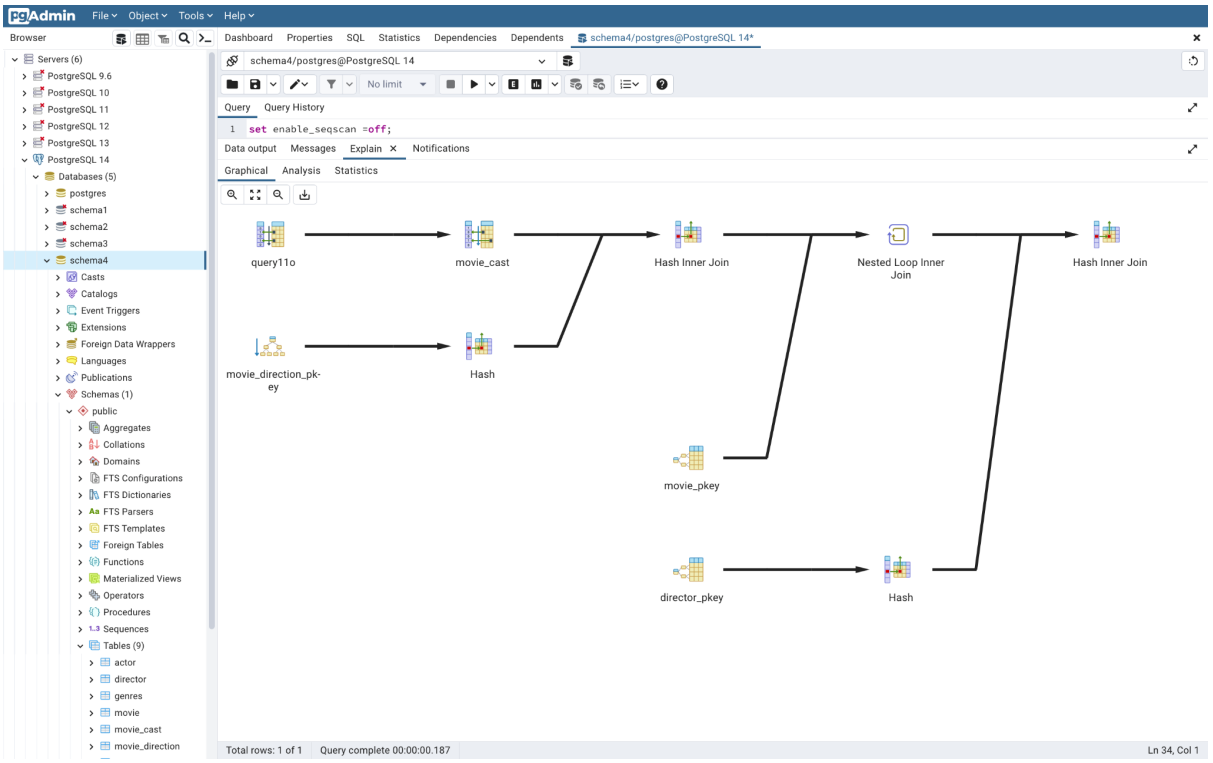
After applying Brin index on the optimised one the execution time and cost got reduced:

*execution time=61.447ms and the cost=5358.11

Applied on table movie_cast and column role.



Optimised Brin physical plan:

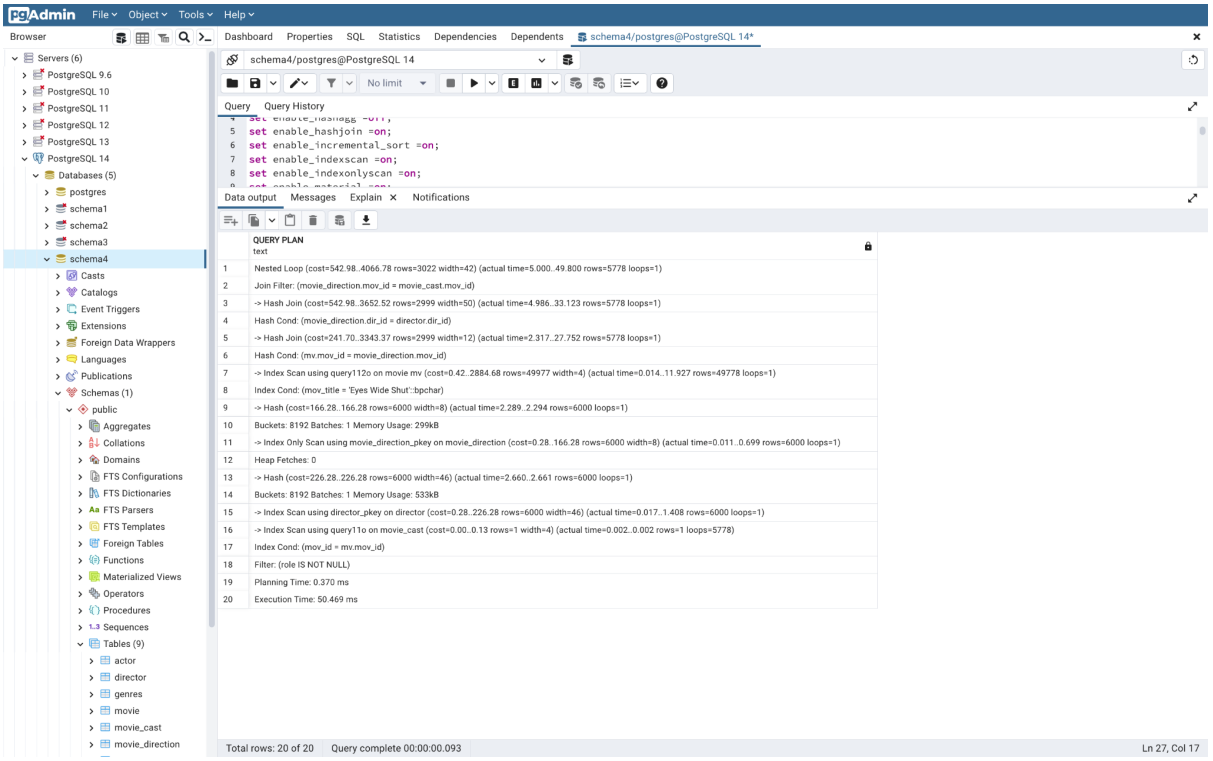


9)Mixed Optimised Query:

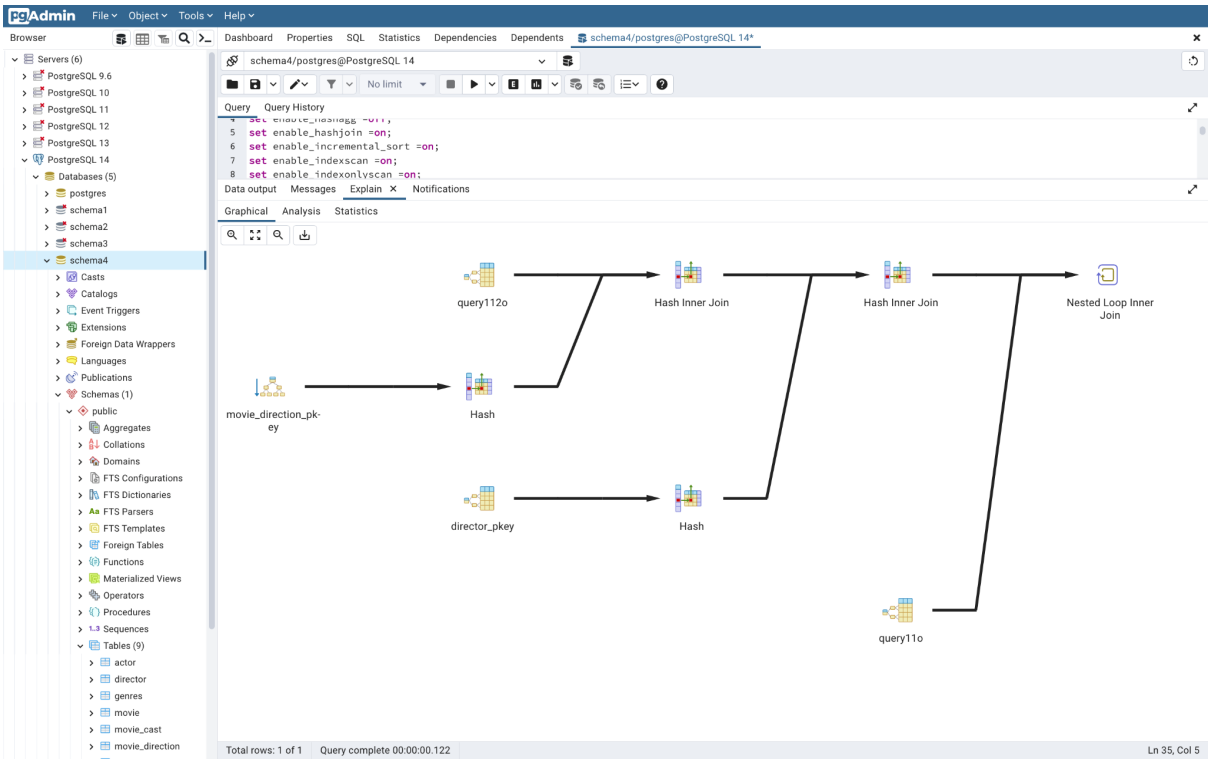
After applying Mixed index on the optimised one the execution time and cost got reduced:

*execution time=50.469ms and the cost=4066.78

Applied on table movie_cast using Hash with column mov_id and table movie using Btree with column mov_title.



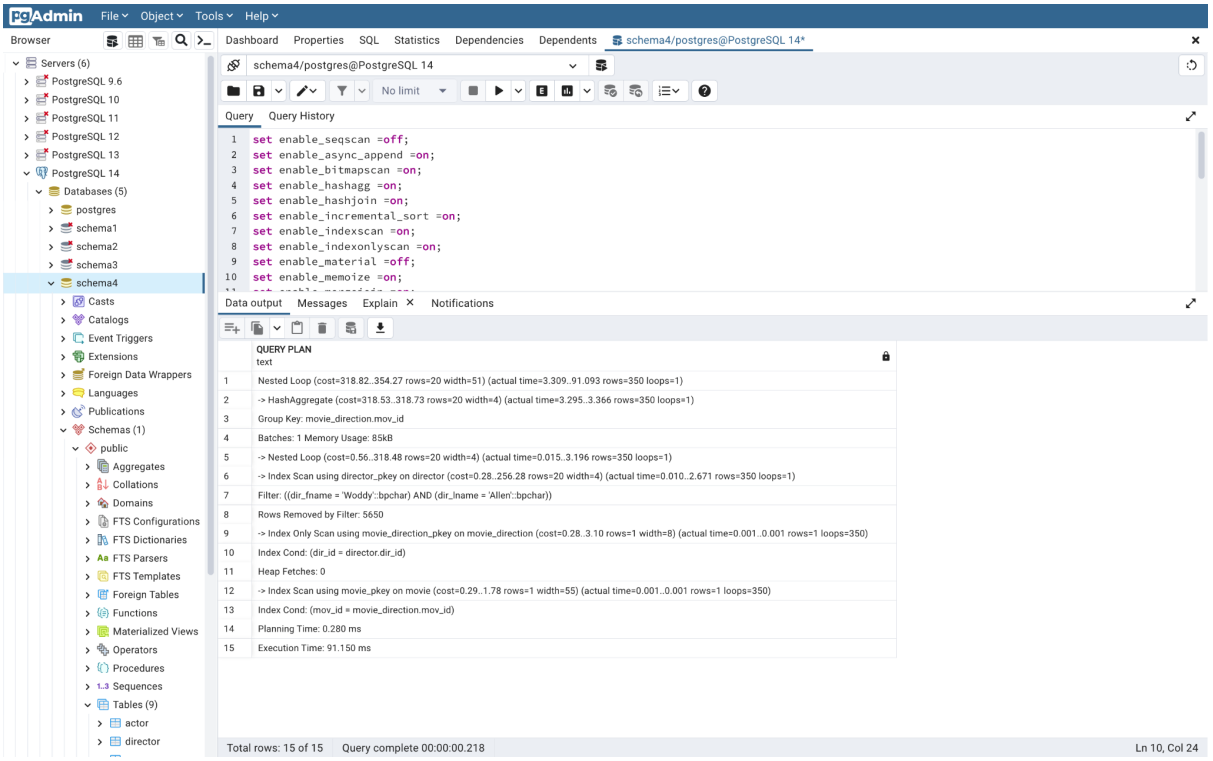
Optimised mixed physical plan:



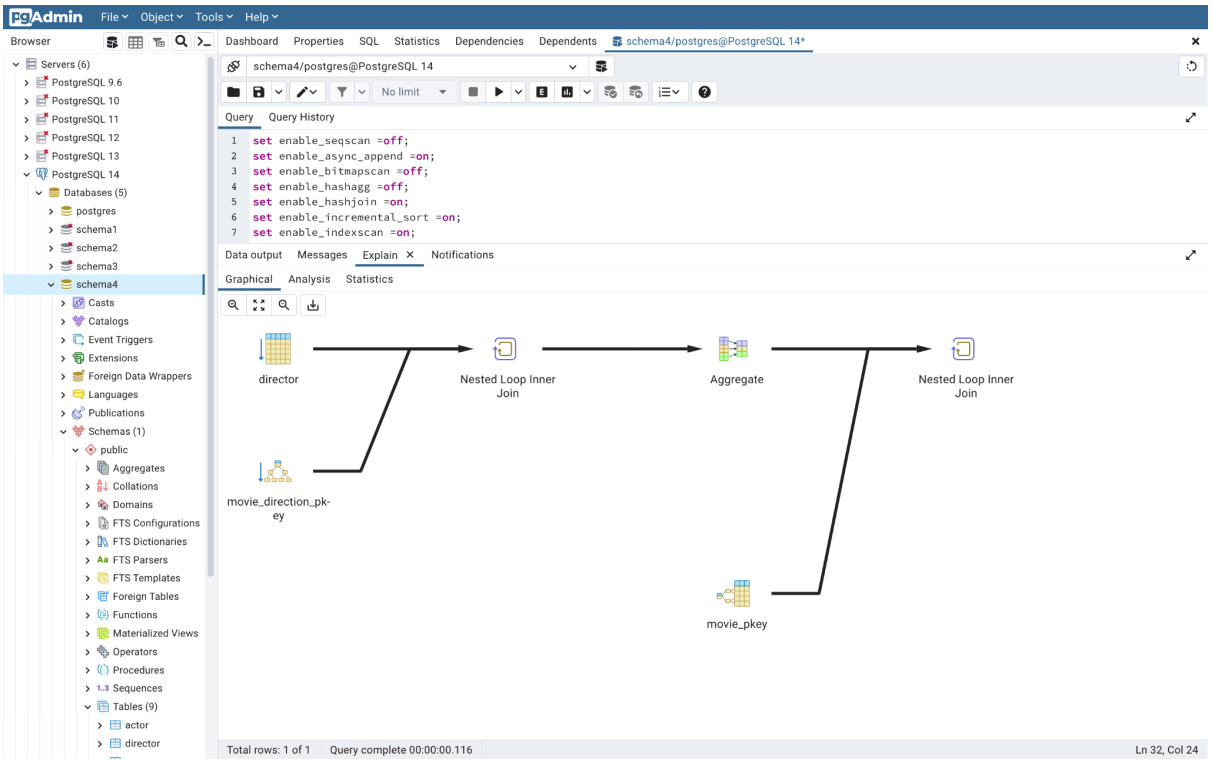
Query12

For this query, I adjusted the flags in order to find the best cost so that it could be reflected on the optimization process and the flags enabled/disabled are not like the other queries.

At first the execution time=91.150ms and the cost was 354.27 and we'll do the same as the other queries.



And the physical plan:

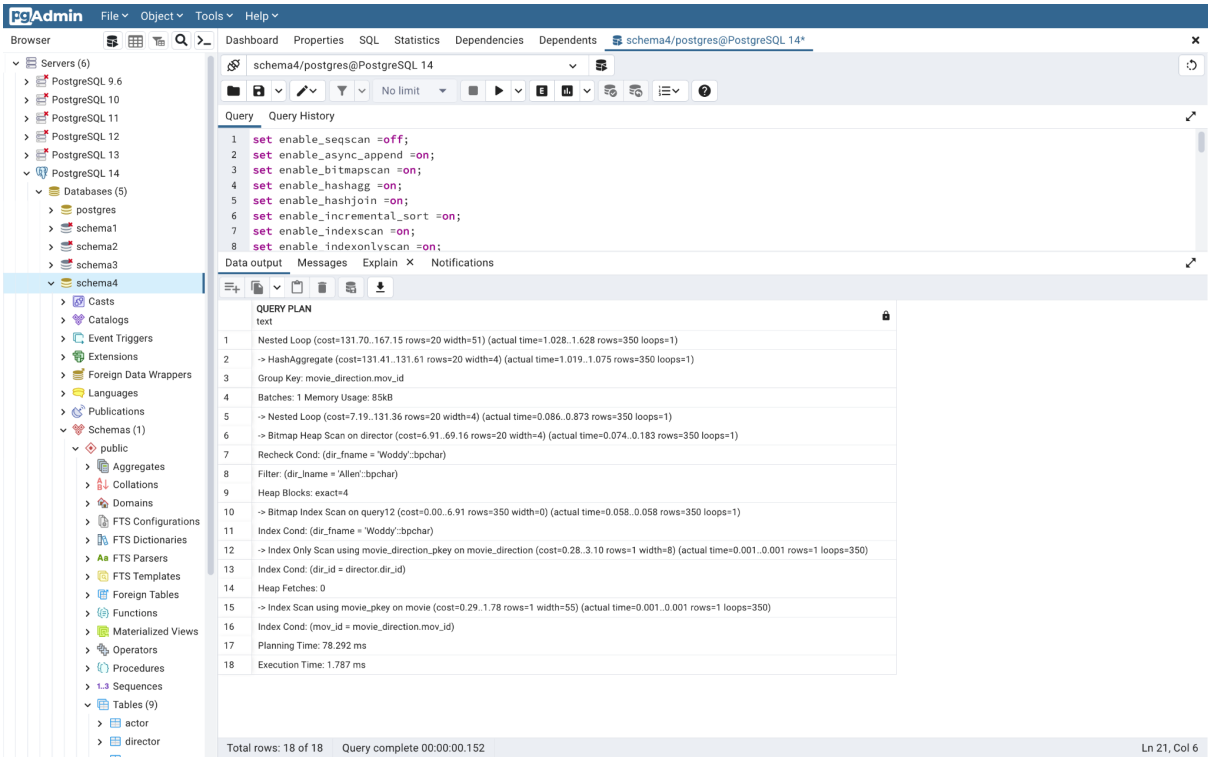


1)<u>Btree indexing:</u>

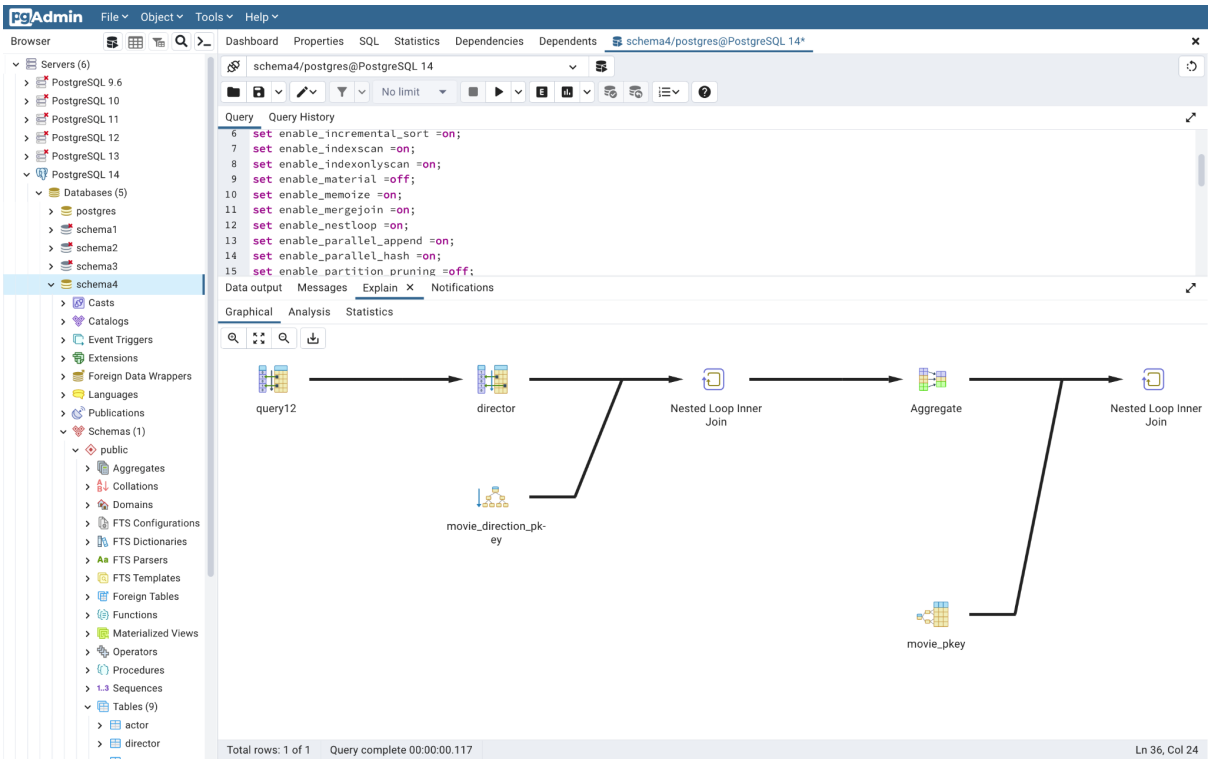
After applying Btree index the cost and execution time has decreased rapidly showing great performance such that:

*execution time became 1.787ms and the cost=167.15

It was applied to the table director and column dir_fname.



Btree physical plan:

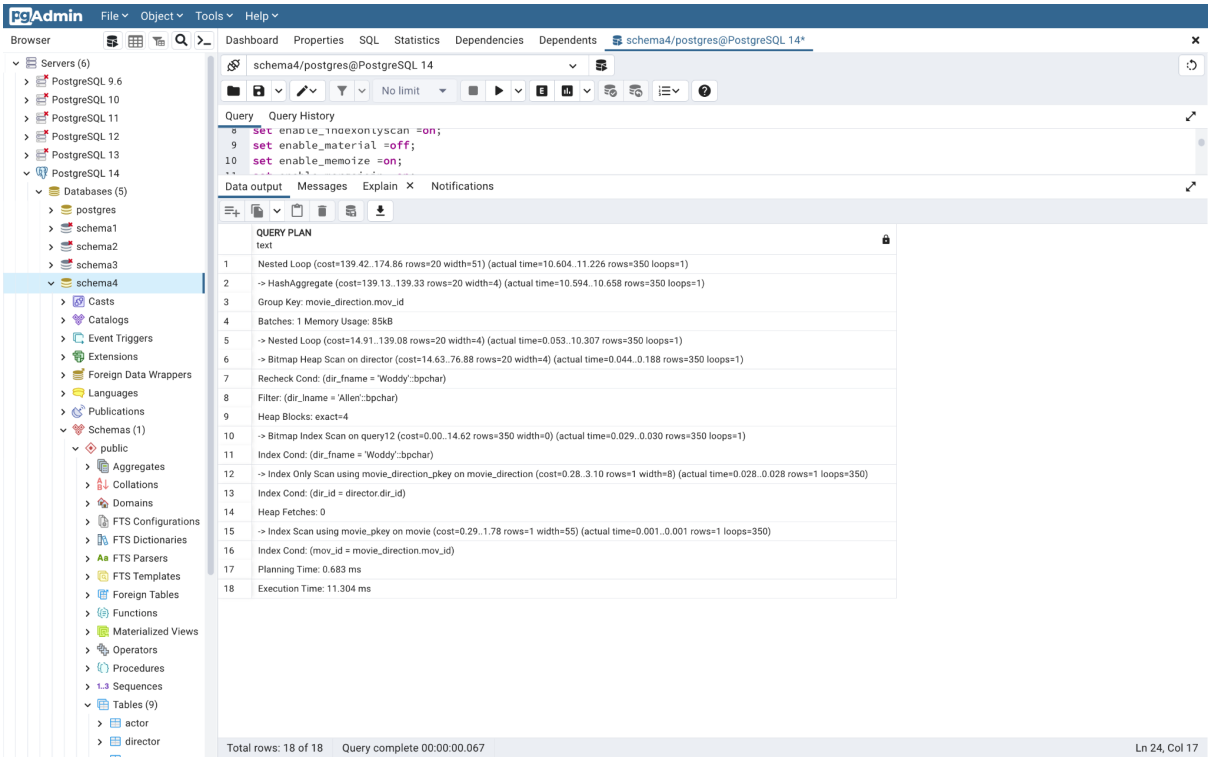


2)<u>Hash indexing:</u>

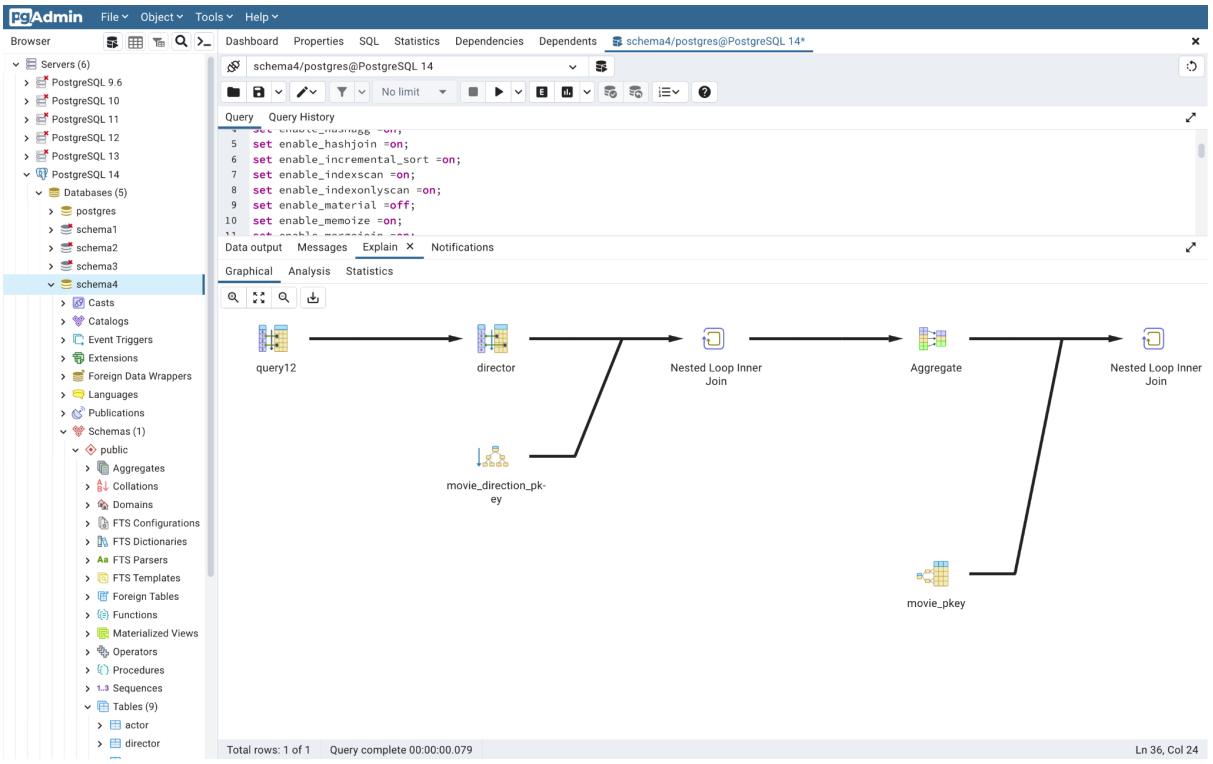
For the hash indexing it reduced the cost and execution time too but not as much as
the Btree as:

*execution time=111.304ms and the cost got reduced to=174.86

It was made on table director and column dir_fname.



Hash physical plan:

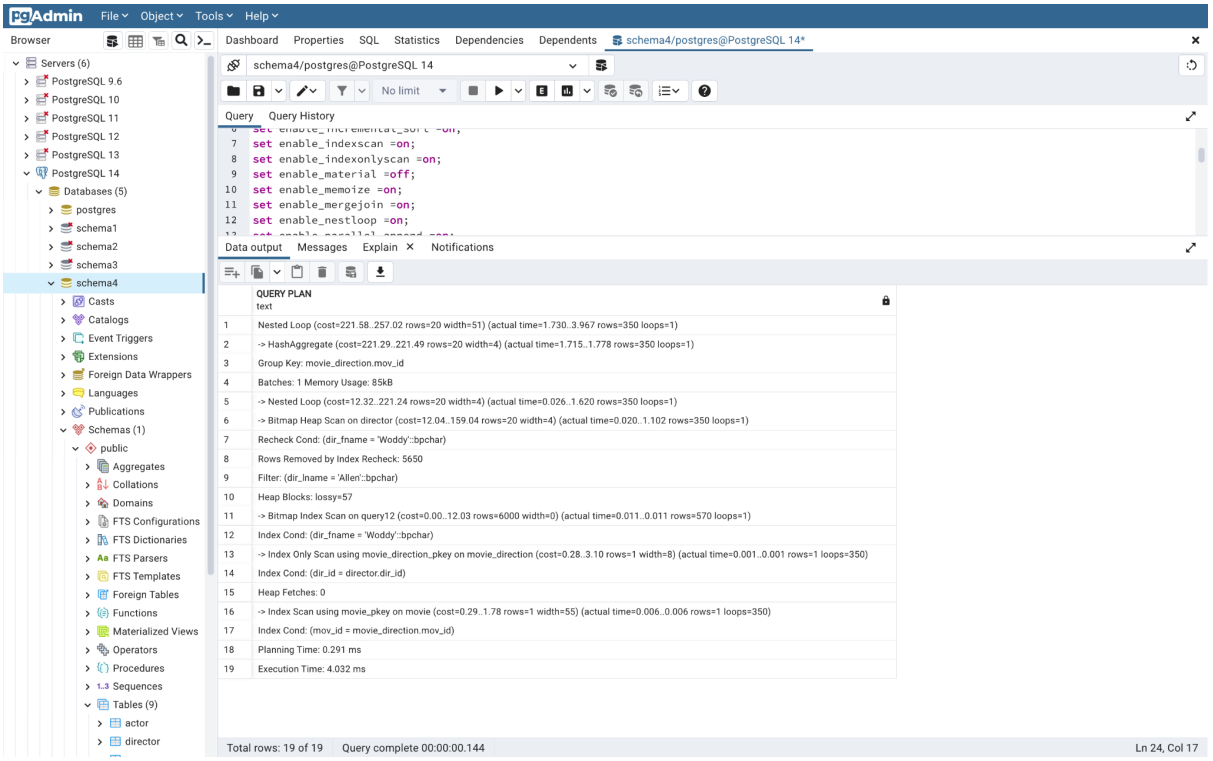


3)<u>Brin indexing:</u>

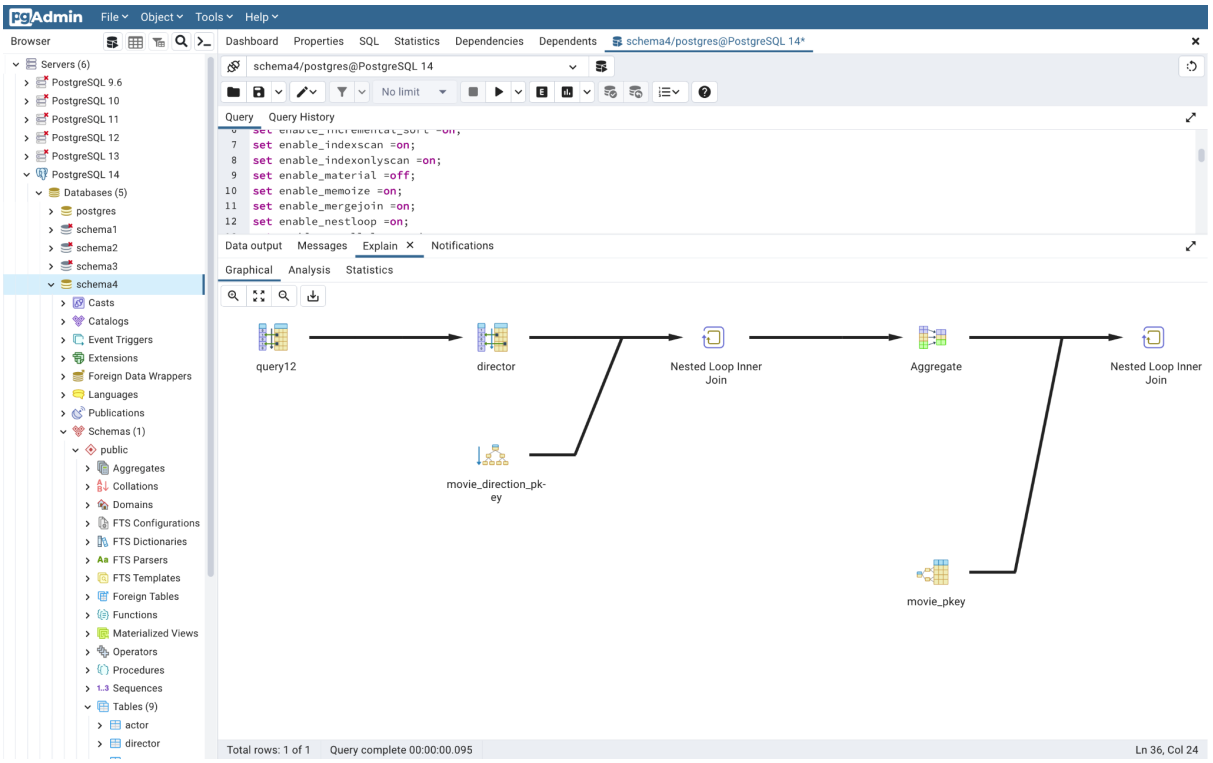
As for Brin the cost and execution time too did get reduced to :

*execution time= 4.032ms and with a cost=257.02

It was made on a table director and column dir_fname.



Brin physical plan:

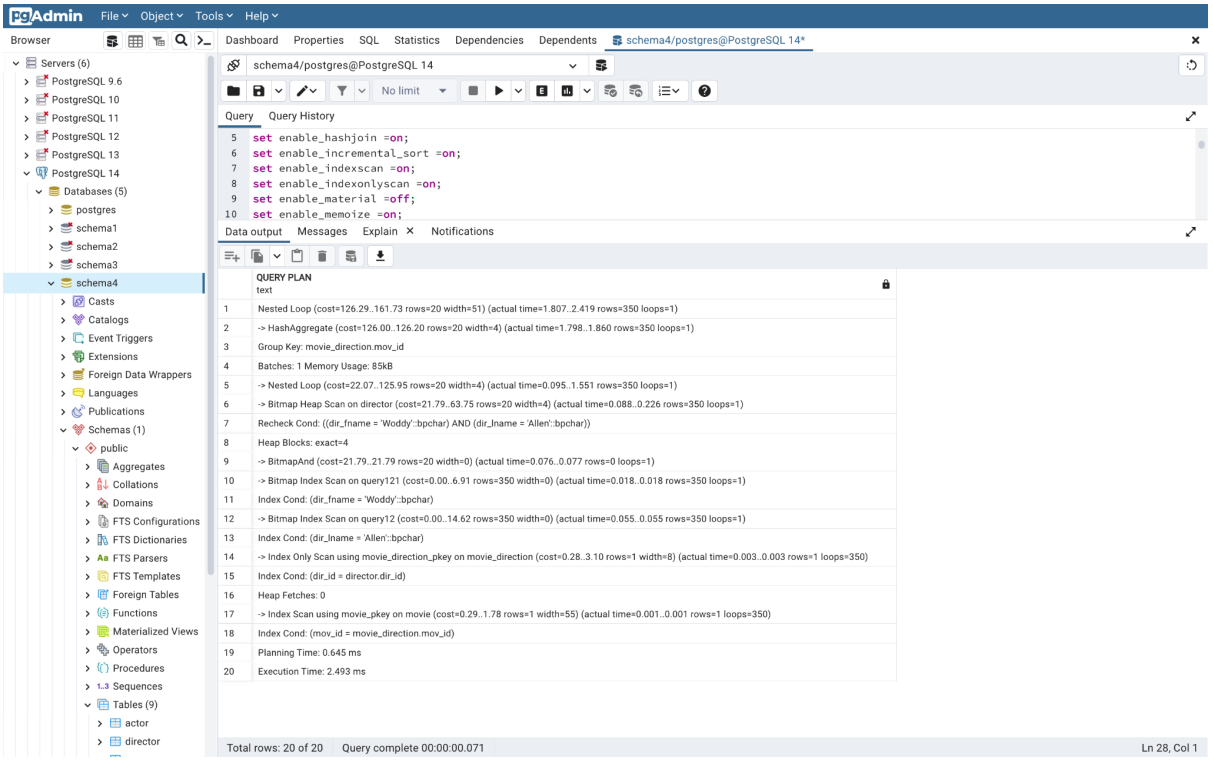


4)<u>Mixed indexing:</u>

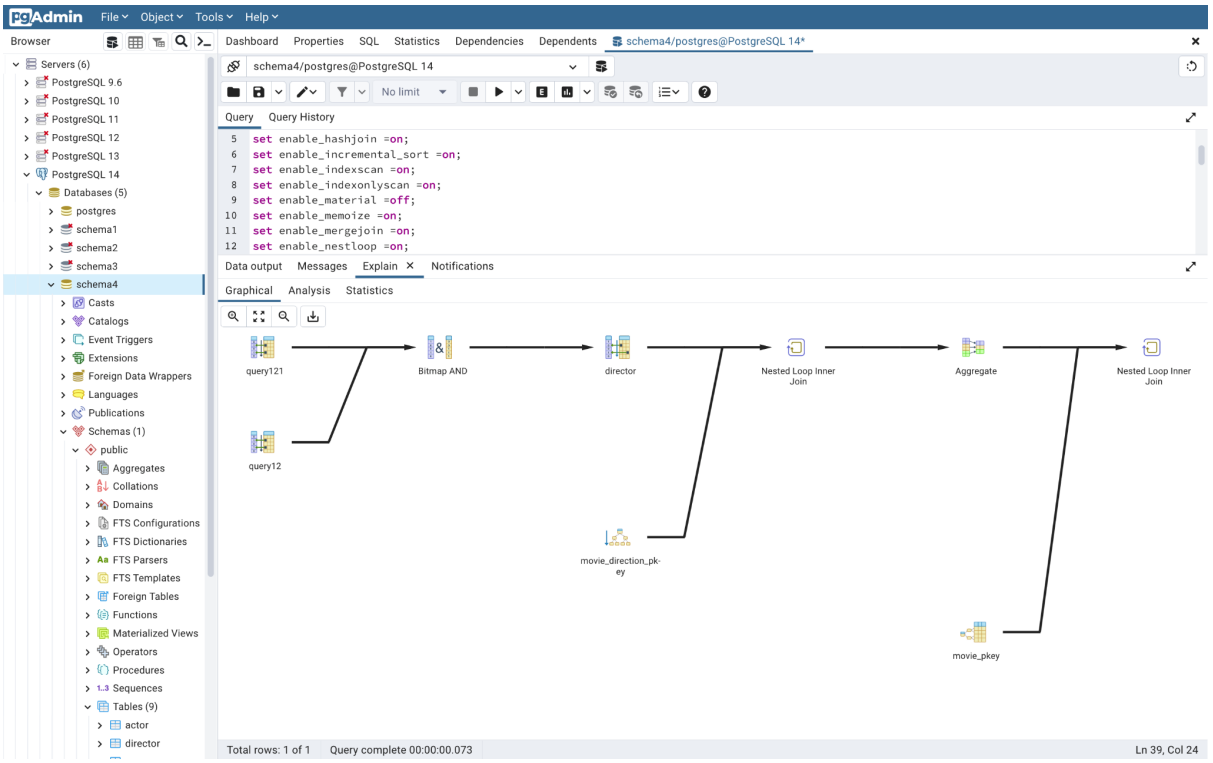
As for mixed the cost and execution time too did get reduced to :

*execution time=2.493ms with a cost=161.73

It was made on a table director using hash index and column dir_lname and using director and column dir_fname.



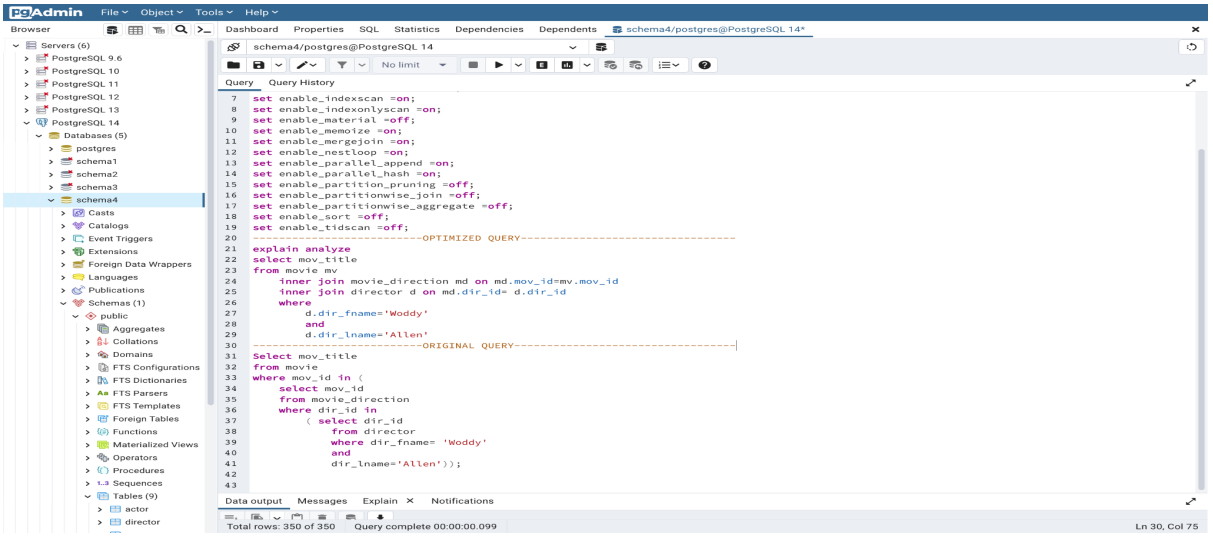
Mixed physical plan:



5)Optimised Query:

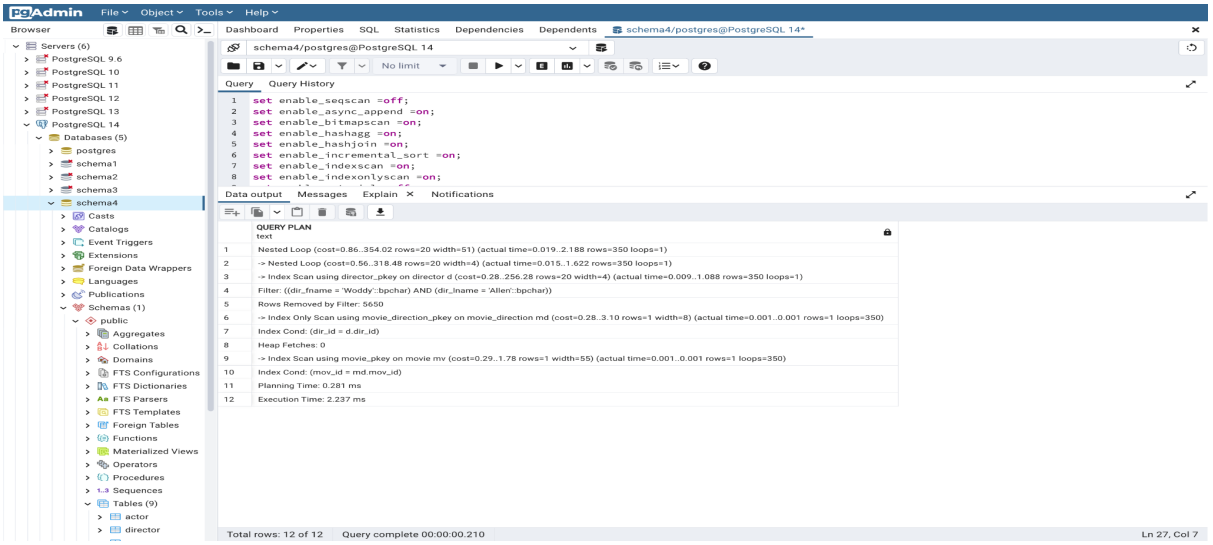
For this part, we tried to optimise the query such that the optimised one should give the same result but in lesser cost and execution time. And after optimising it, it did reduce both with a better performance.

This is the optimised vs the original query both written down:

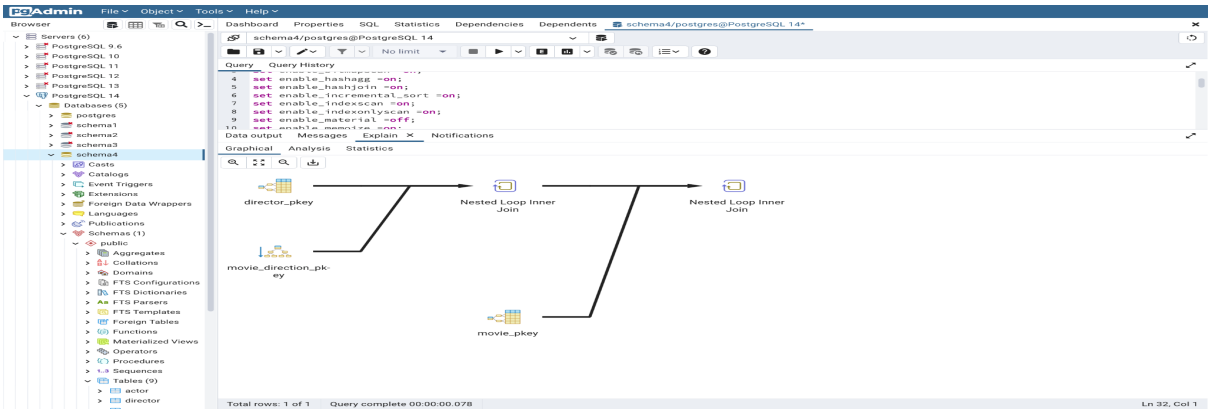


The costs of the optimised one:

*execution time=2.237ms and the cost=354.02



Optimised physical plan:

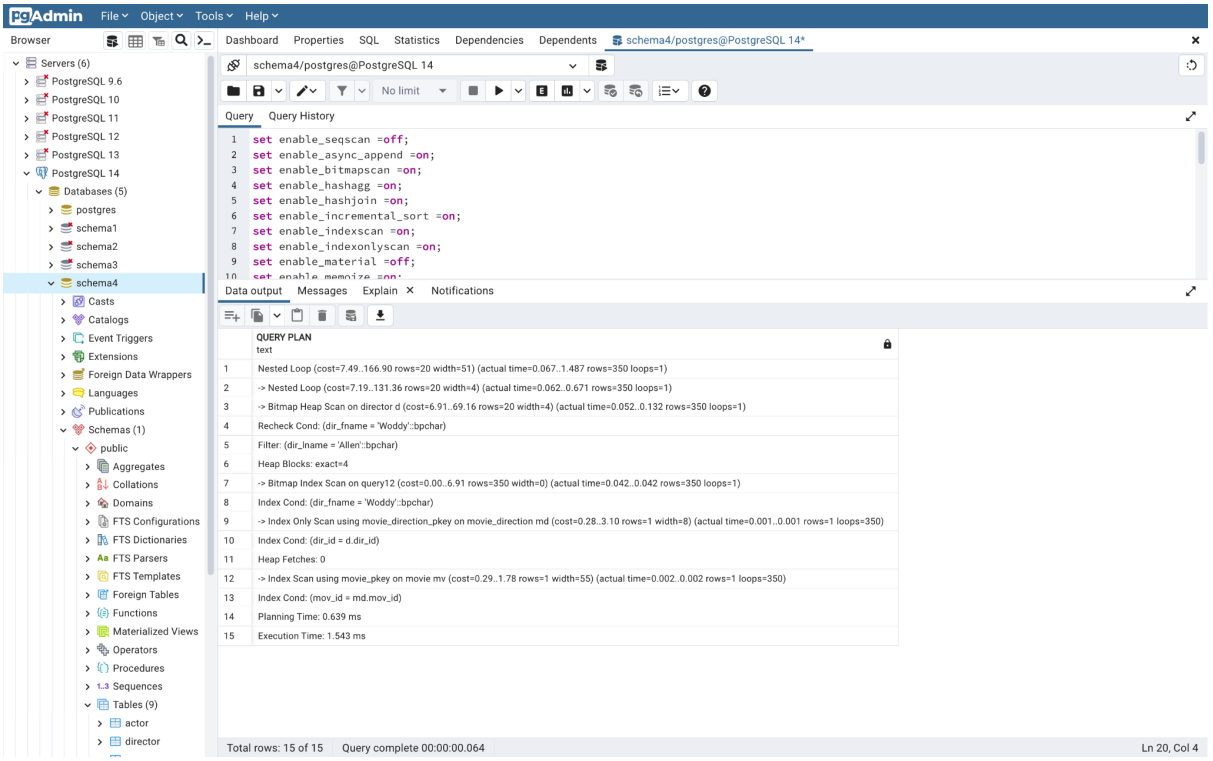


6)Btree Optimised Query:

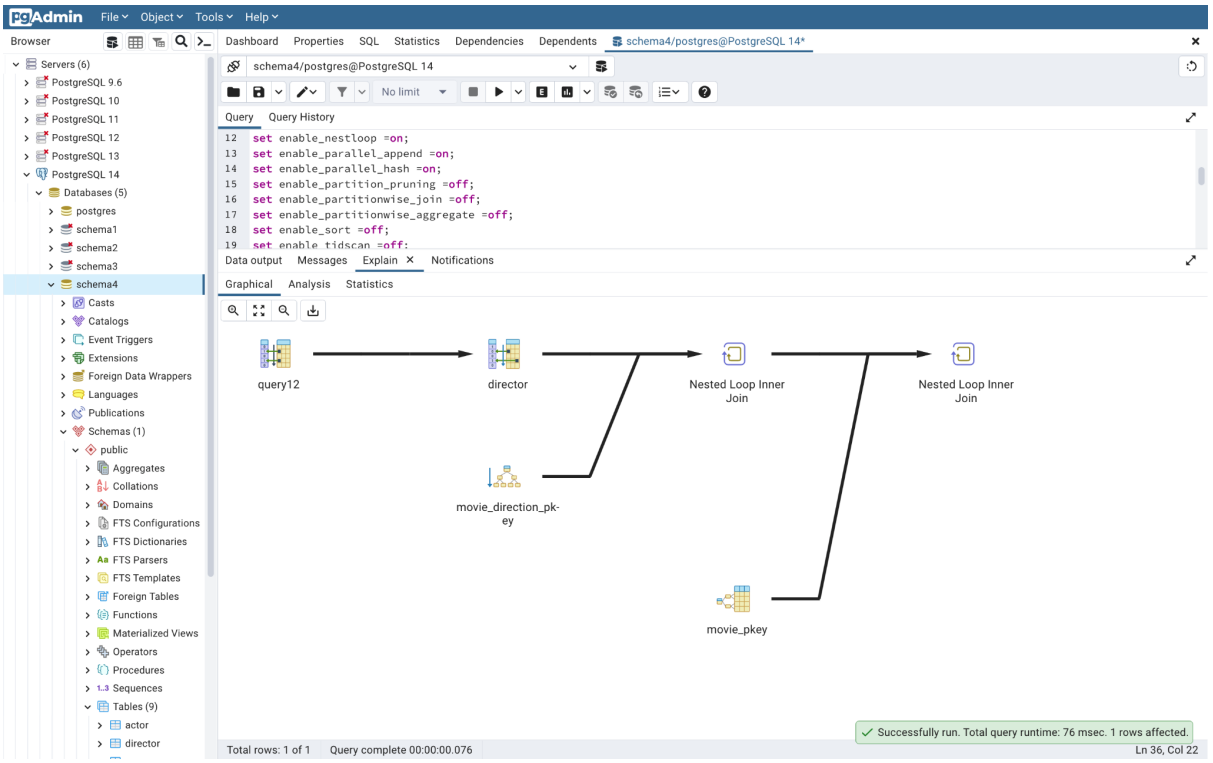
After applying Btree index on the optimised one the execution time and cost got reduced even more:

*execution time=1.543ms and the cost=166.90

Applied on table director and column dir_fname.



Optimised Btree physical plan:

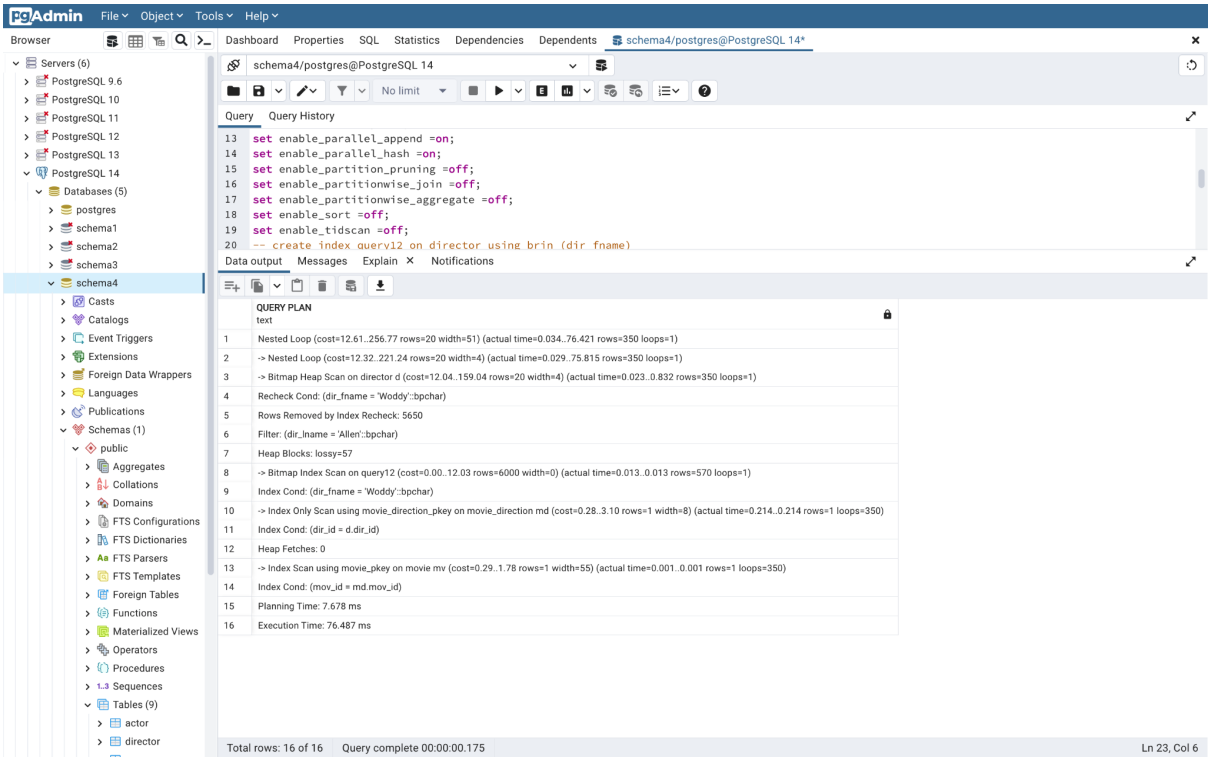


7)Hash Optimised Query:

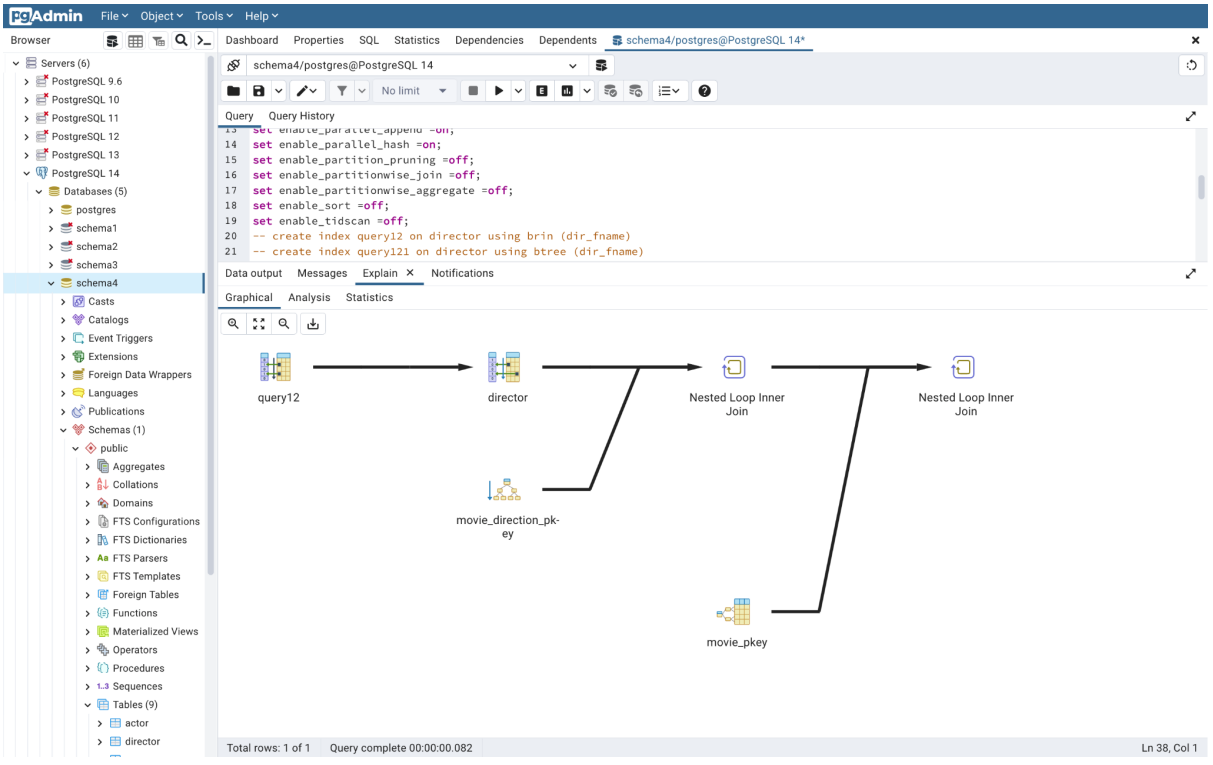
After applying hash index on the optimised one the execution time and cost got reduced too:

*execution time=76.487ms and the cost=256.77

Applied on table director and column dir_fname.



Optimised hash physical plan:

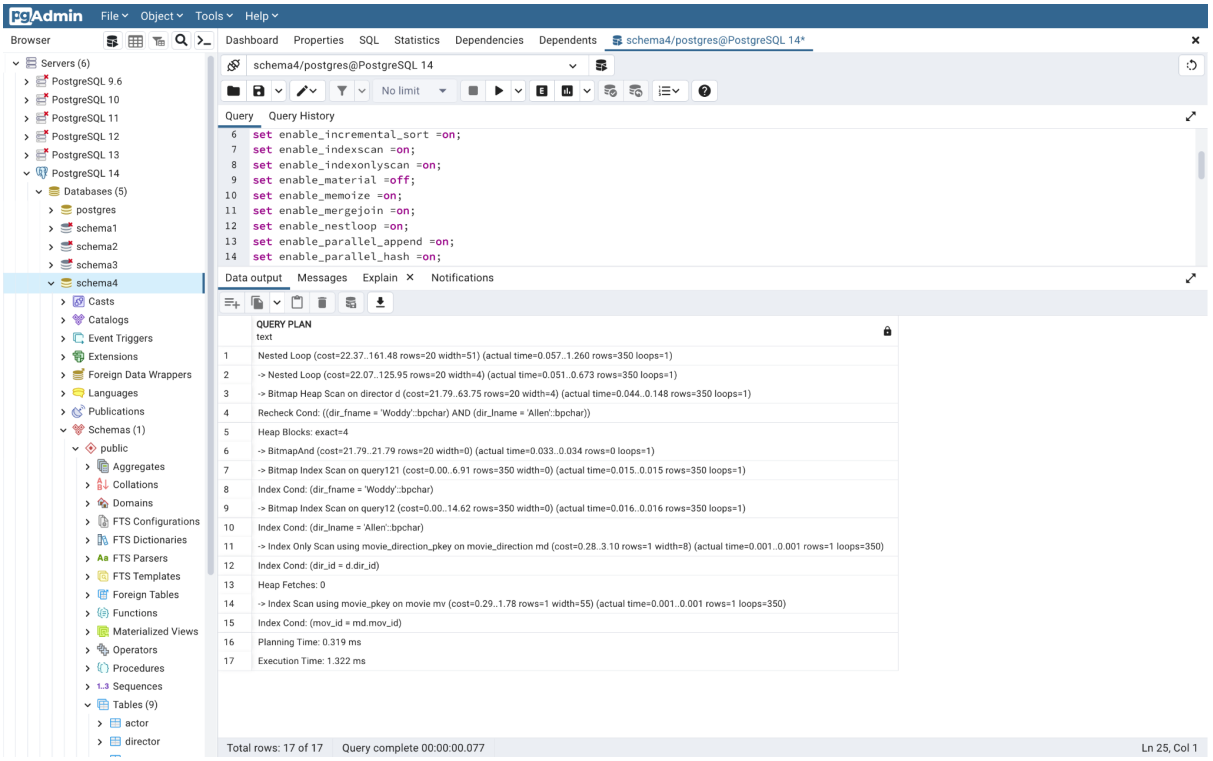


9)Mixed Optimised Query:

After applying Mixed index on the optimised one the execution time and cost got reduced:

*execution time=1.322ms and the cost=161.48

Applied on table director using Hash with column dir_lname and table director using Btree with column dir_fname.



Optimised mixed physical plan: