QUERY 10

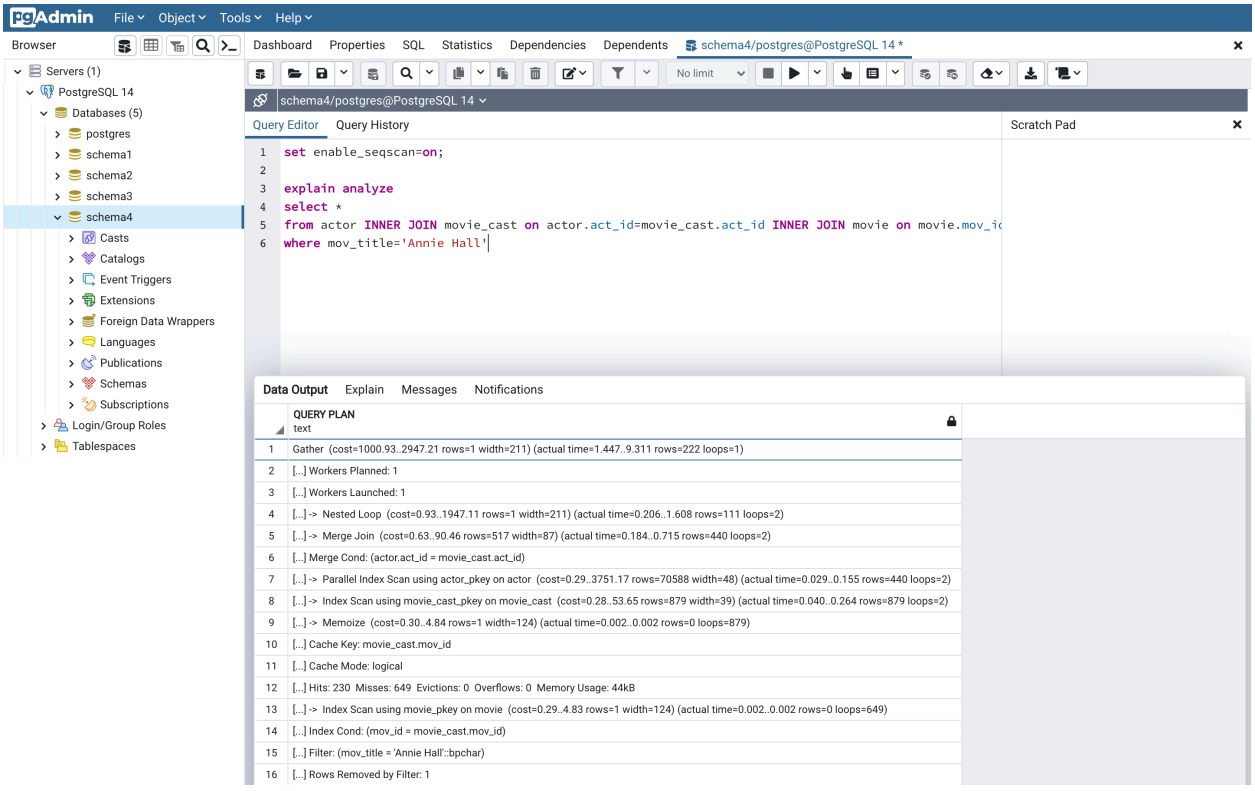
QUERY 10 AFTER OPTIMIZATION IS

```
select *
from actor INNER JOIN movie_cast on actor.act_id=movie_cast.act_id INNER JOIN movie on
movie.mov_id = movie_cast.mov_id
where mov_title='Annie Hall'
```

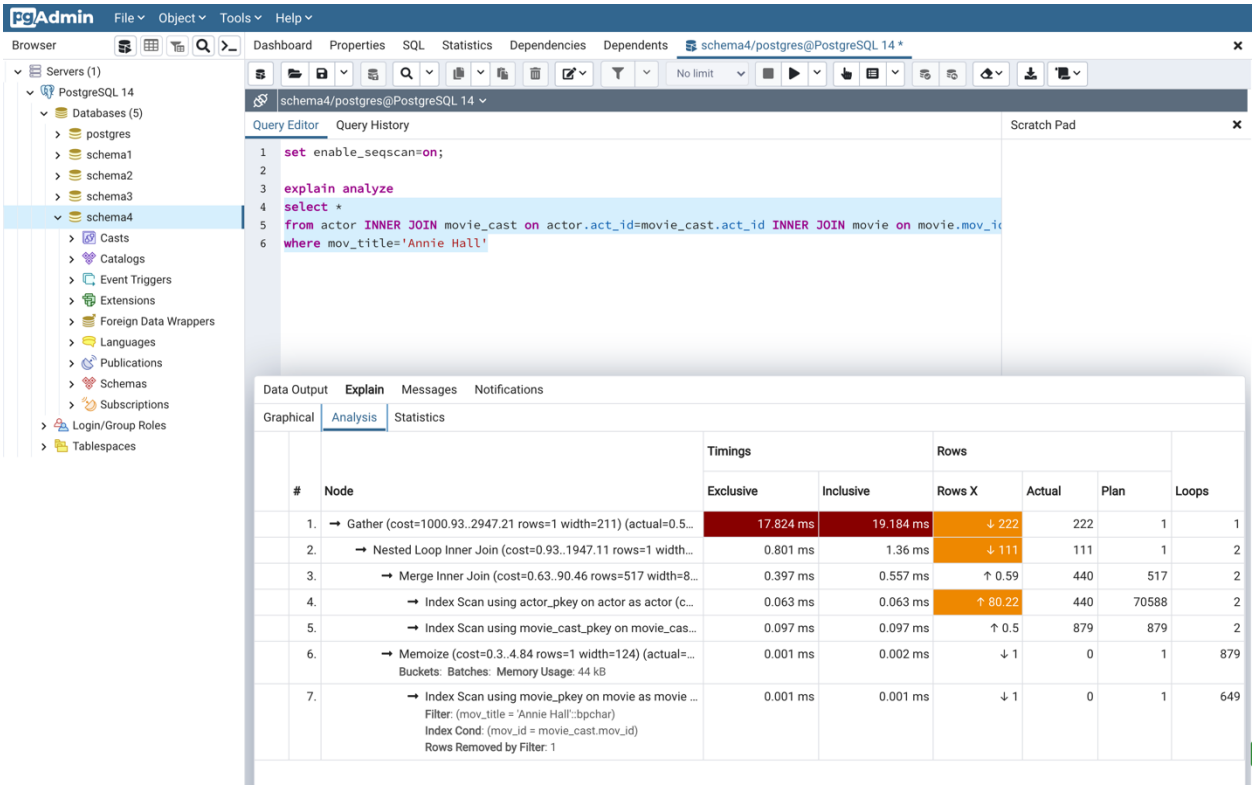
This is optimized because INNER JOIN are much faster than subqueries so we replaced every IN with a join

Query 10 with no index

Planning and execution time :



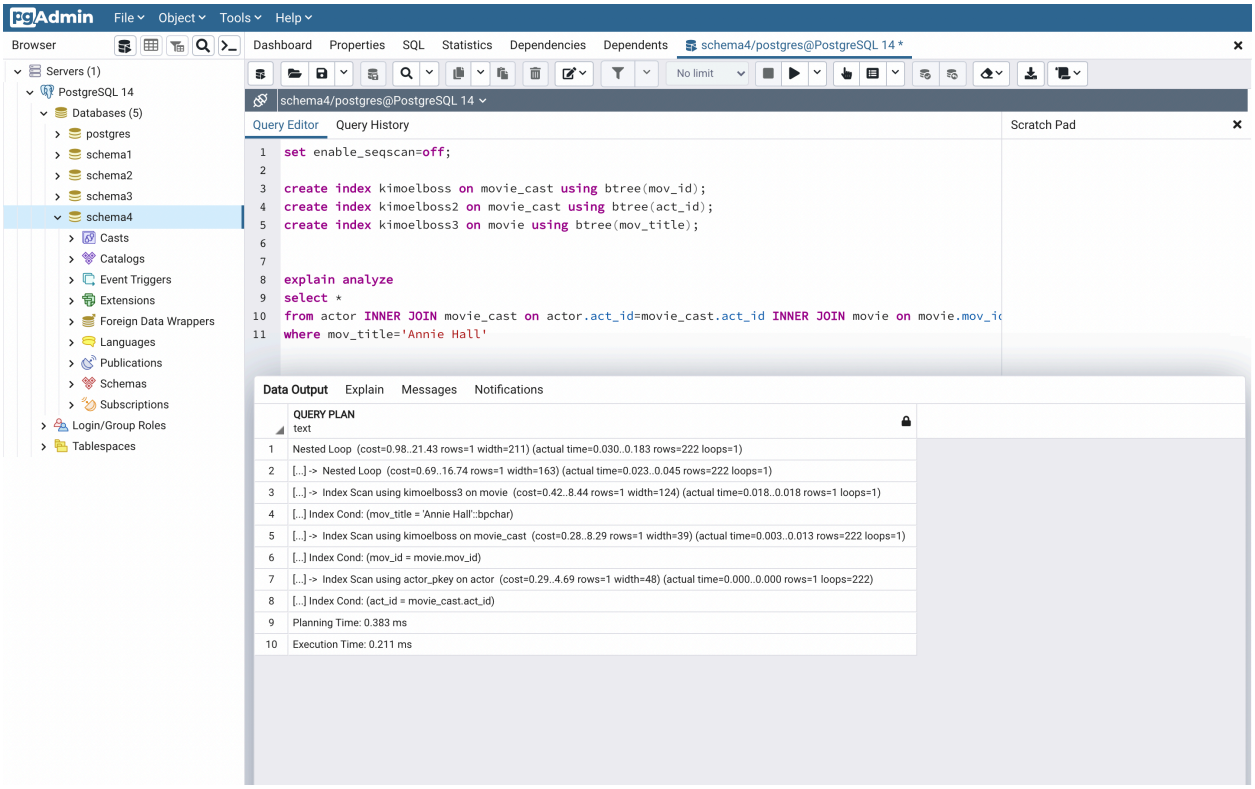
Execution Plan and costs :



Query 10 using B+ index on : movie_cast (mov_id) , movie_cast (act_id) movie (mov_title)

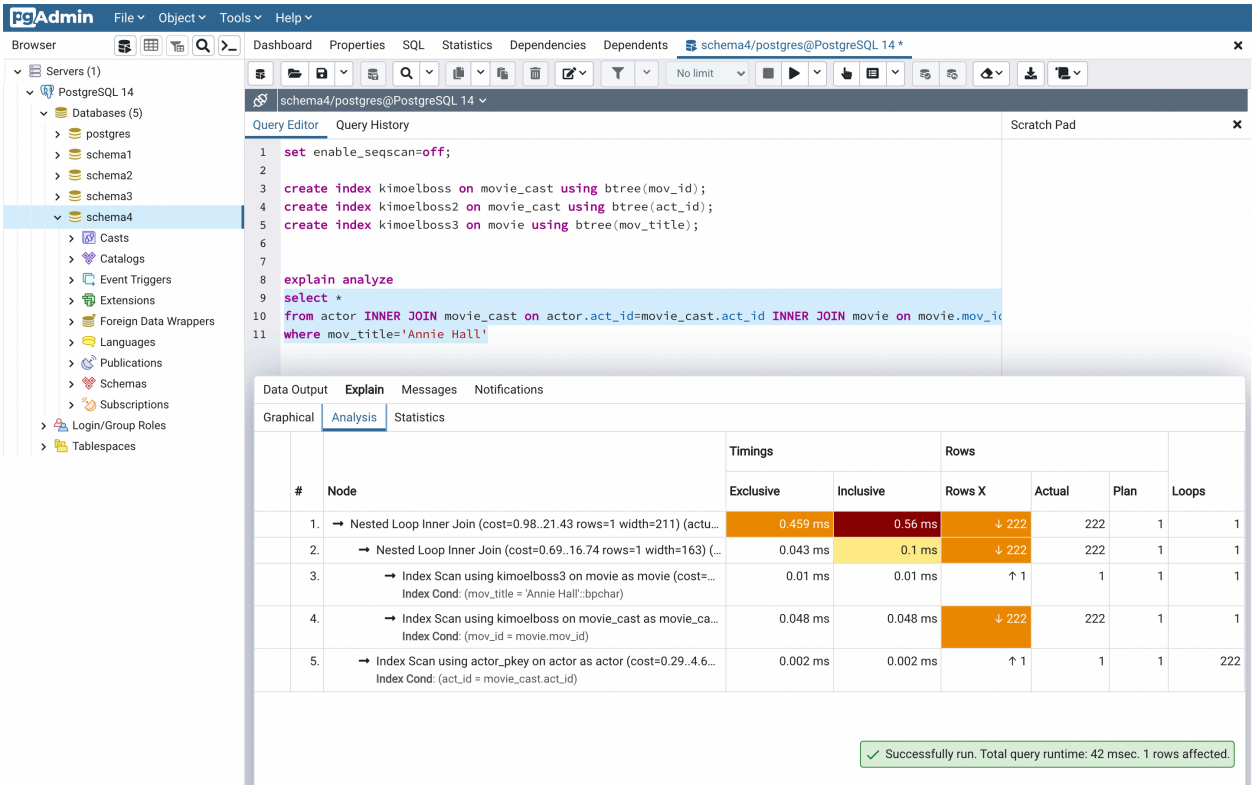
Disabled flags: seqscan

Planning and execution time :



Here the B+ tree optimized the query as we put it on the columns that we search on like act_id and mov_id together as well as the mov_title as B+ tree search in O(logn) time so it is better than seqscan

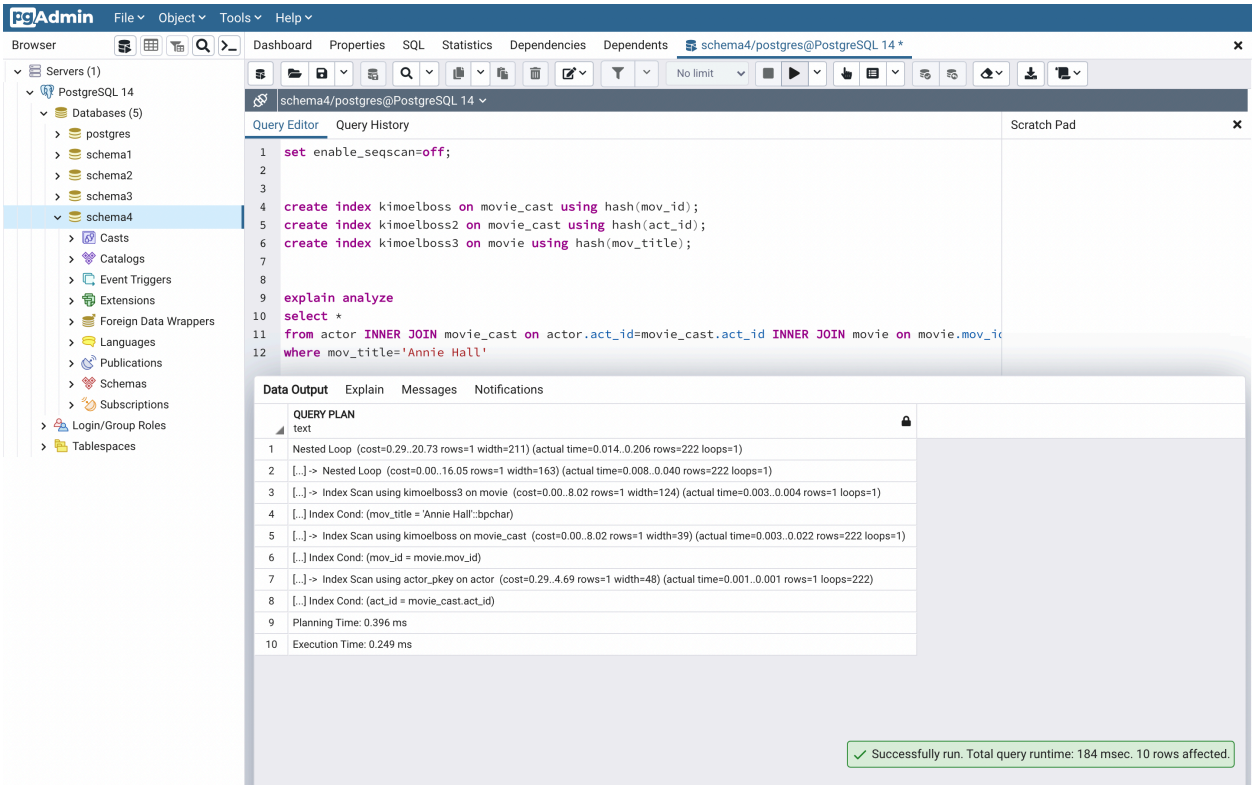
Execution Plan and costs :



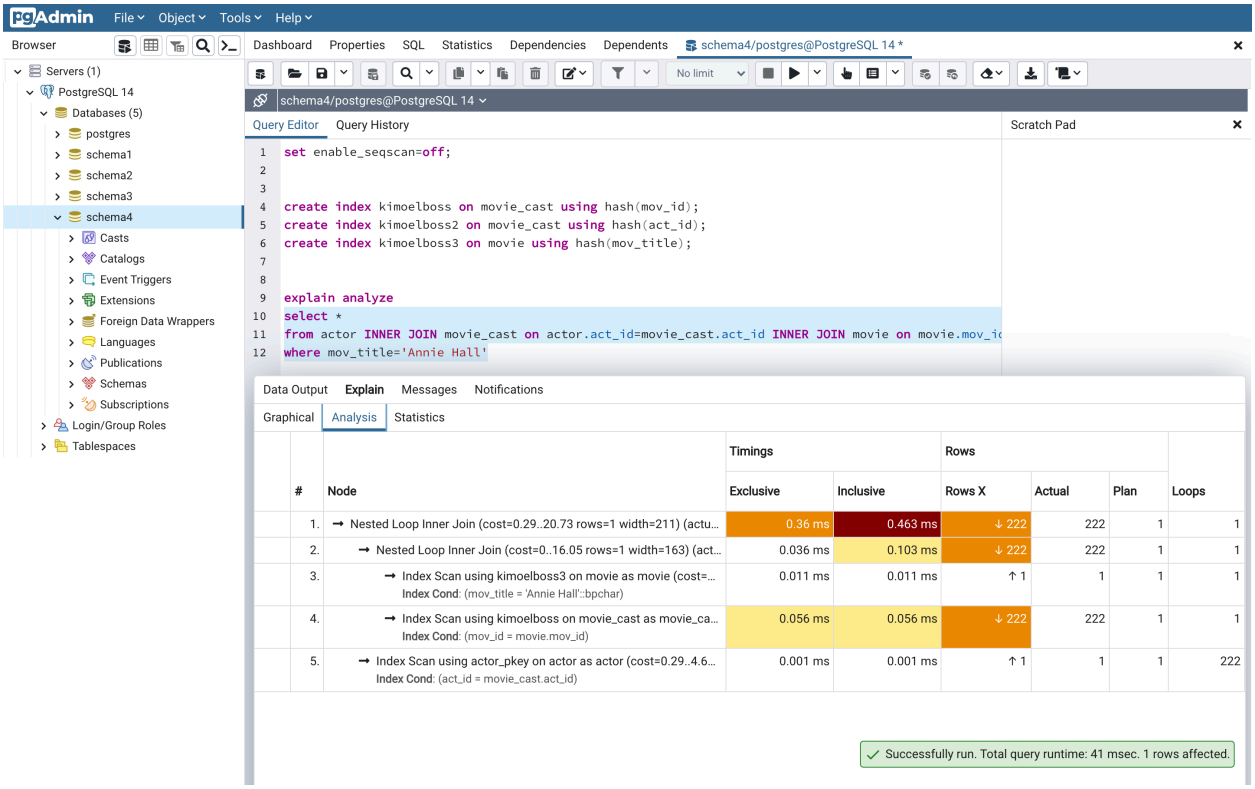
Query 10 using Hash index on : movie_cast (mov_id), , movie_cast(act_id)), movie (mov_title)

Seq Scan off

Planning and execution time :



Execution plan and costs:

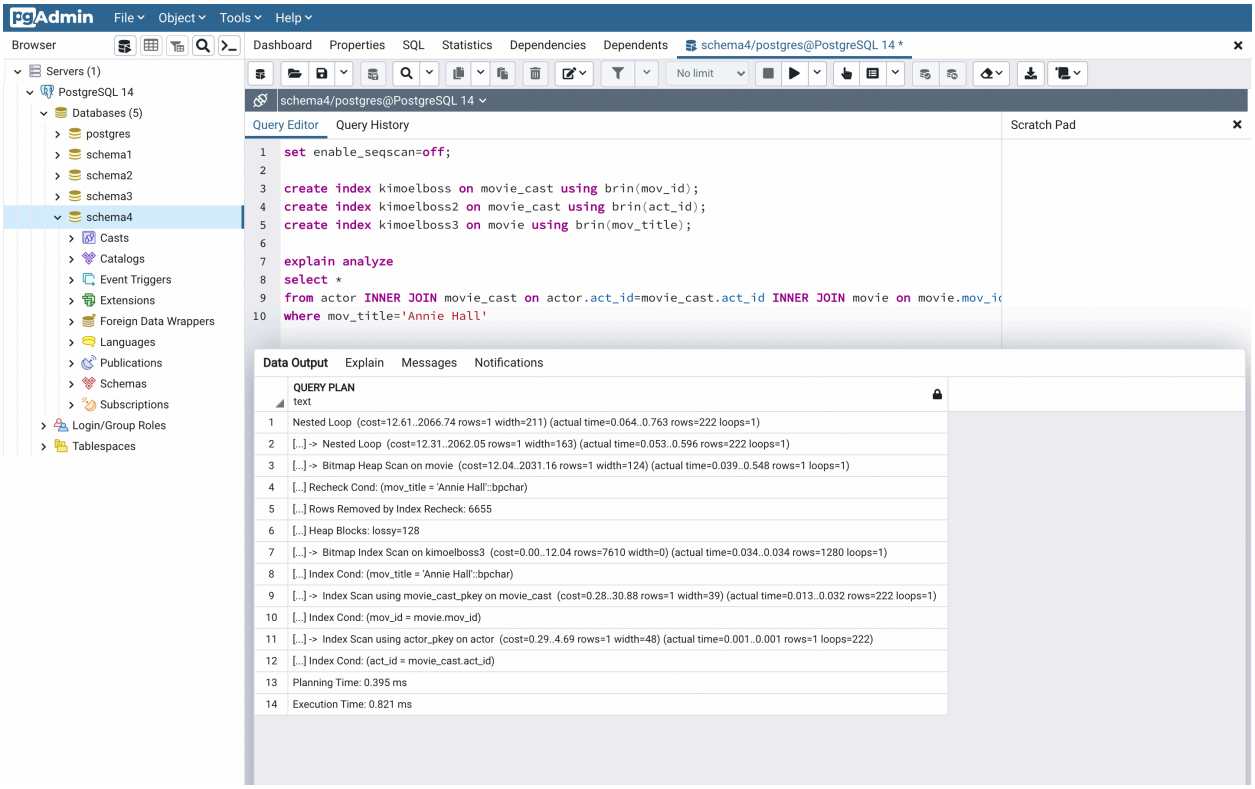


Here the hash index increased performance as it search in O(1) time so it helped in searching from mov_title=”Annie Hall” and mov_id and act_id

Query 10 using BRIN index on : movie_cast (mov_id), , movie_cast(act_id)), movie (mov_title)

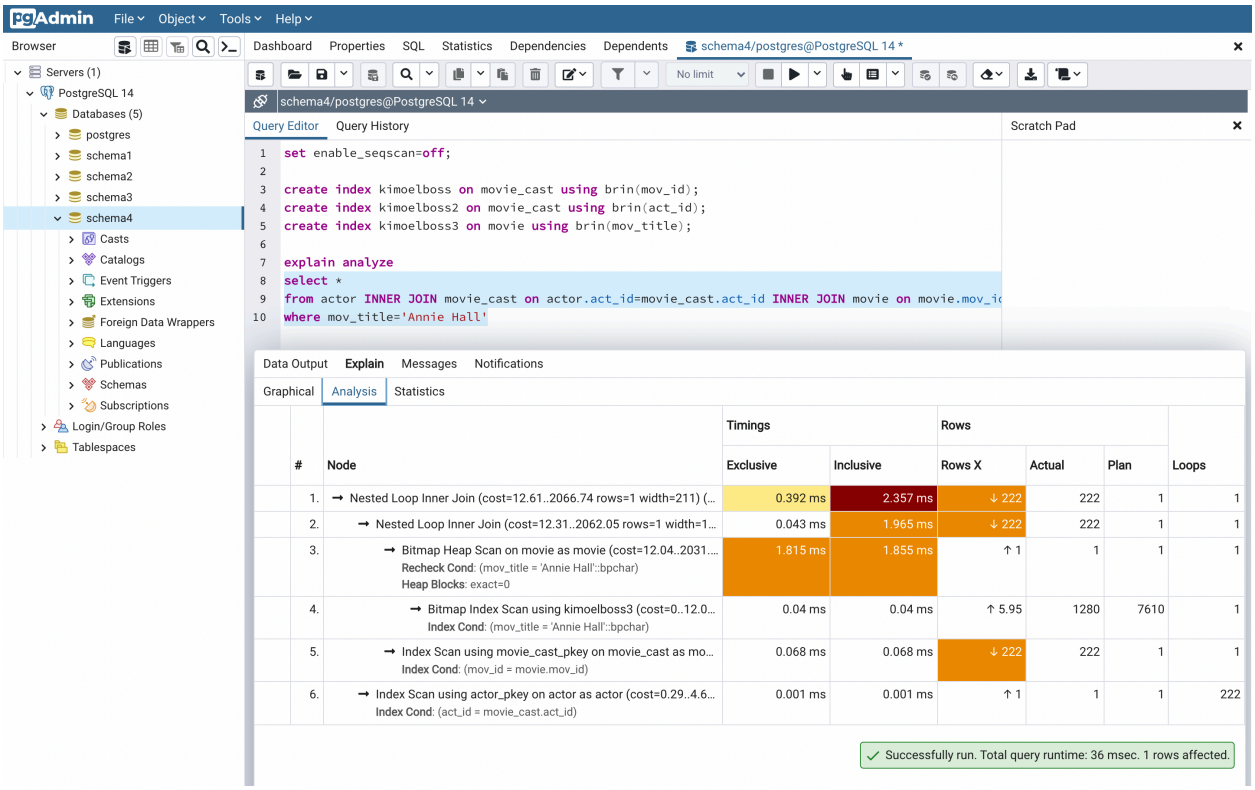
Disabled flags: seqscan

Planning and execution time :



Here the BRIN index decreased the cost but not much (from 3207 to 2067). BRIN is used in small range queries in big ranges so it does not help that much in exact value queries which was mov_title, we wanted it to be an exact value so BRIN wasn't the most efficient index

Execution plan and costs:

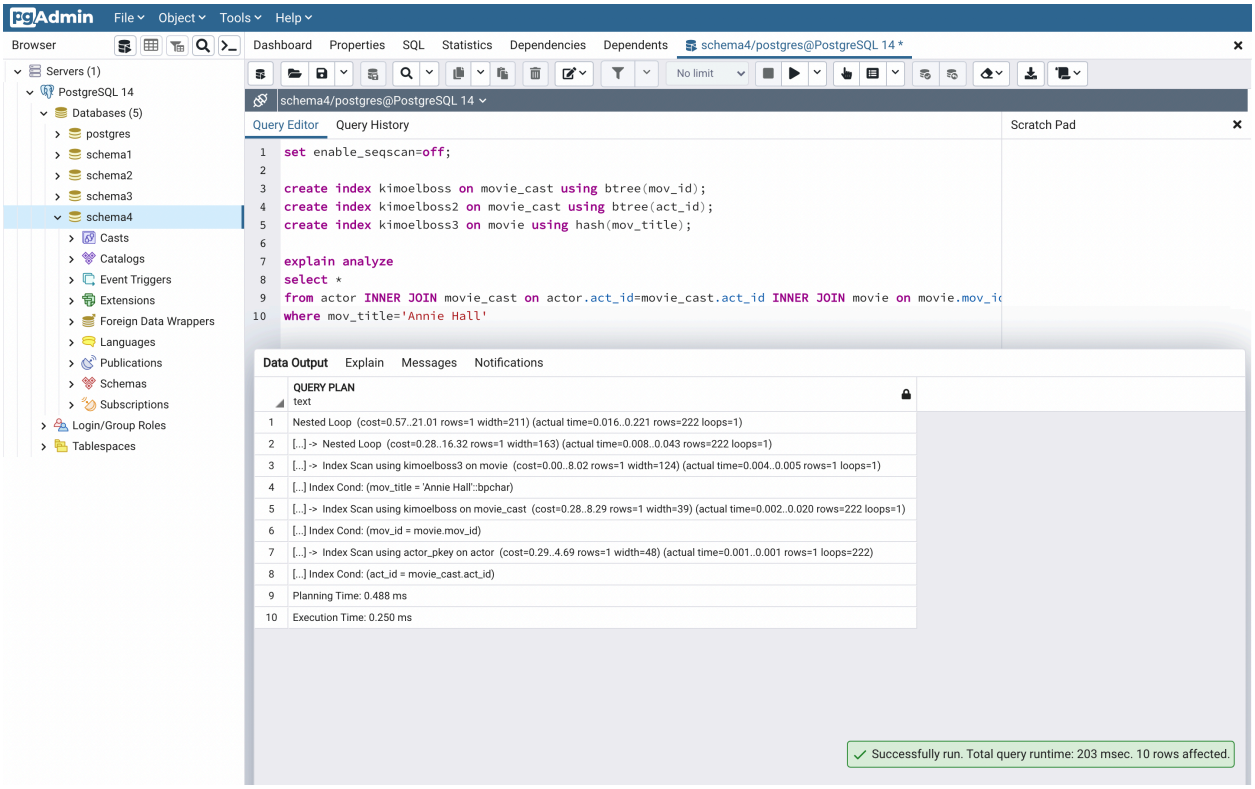


Query 10 using MIX index on : Btree-> movie_cast (mov_id), , movie_cast(act_id)),

Hash -> movie (mov_title)

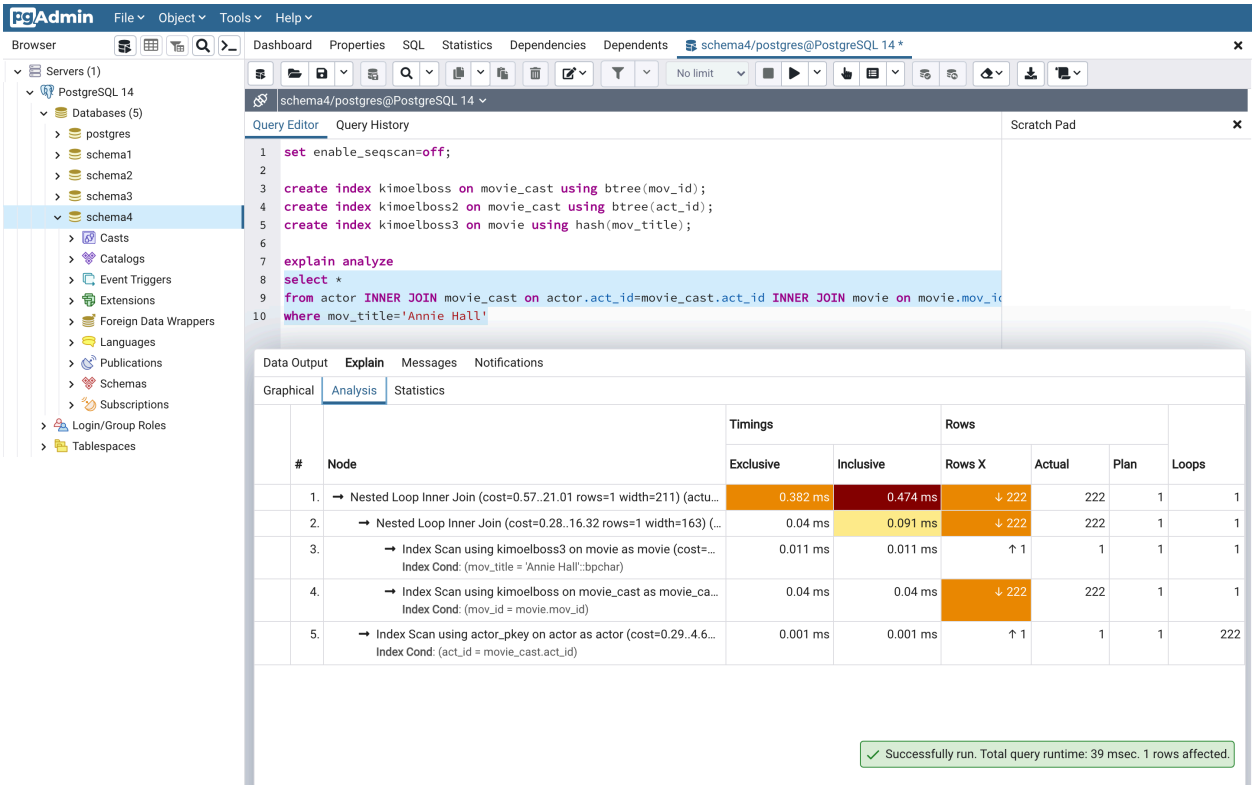
Disabled flags: seqscan

Planning and execution time :



Here the hash index increased performance and less cost as it search in O(1) time so it helped in searching from mov_title=”Annie Hall” and B+ tree optimized the searched for the ids of movie and actor in O(logn) so overall much better performance

Execution plan and costs:



QUERY 11

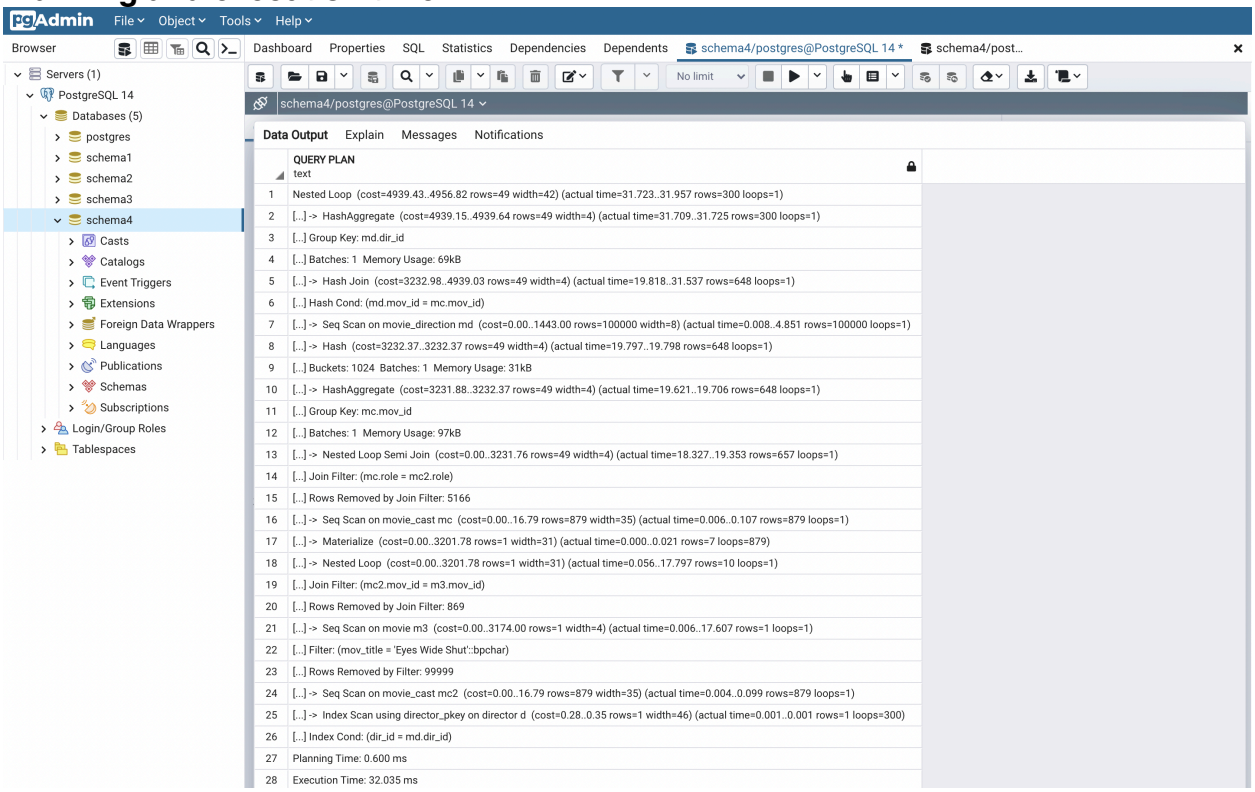
QUERY 11 AFTER OPTIMIZATION IS

```
select dir_fname, dir_lname
from director d
where exists(
        select dir_id
        from movie_direction md
        where md.dir_id=d.dir_id and exists(
                        select mov_id
                        from movie_cast mc
                        where mc.mov_id=md.mov_id and mc.role =any( select role
                                                                    from movie_cast mc2
                                                                    where exists(
                                                                            select mov_id
                                                                            from movie m3
                                                                            where
                                mov_title='Eyes Wide Shut' and m3.mov_id=mc2.mov_id ))));
```

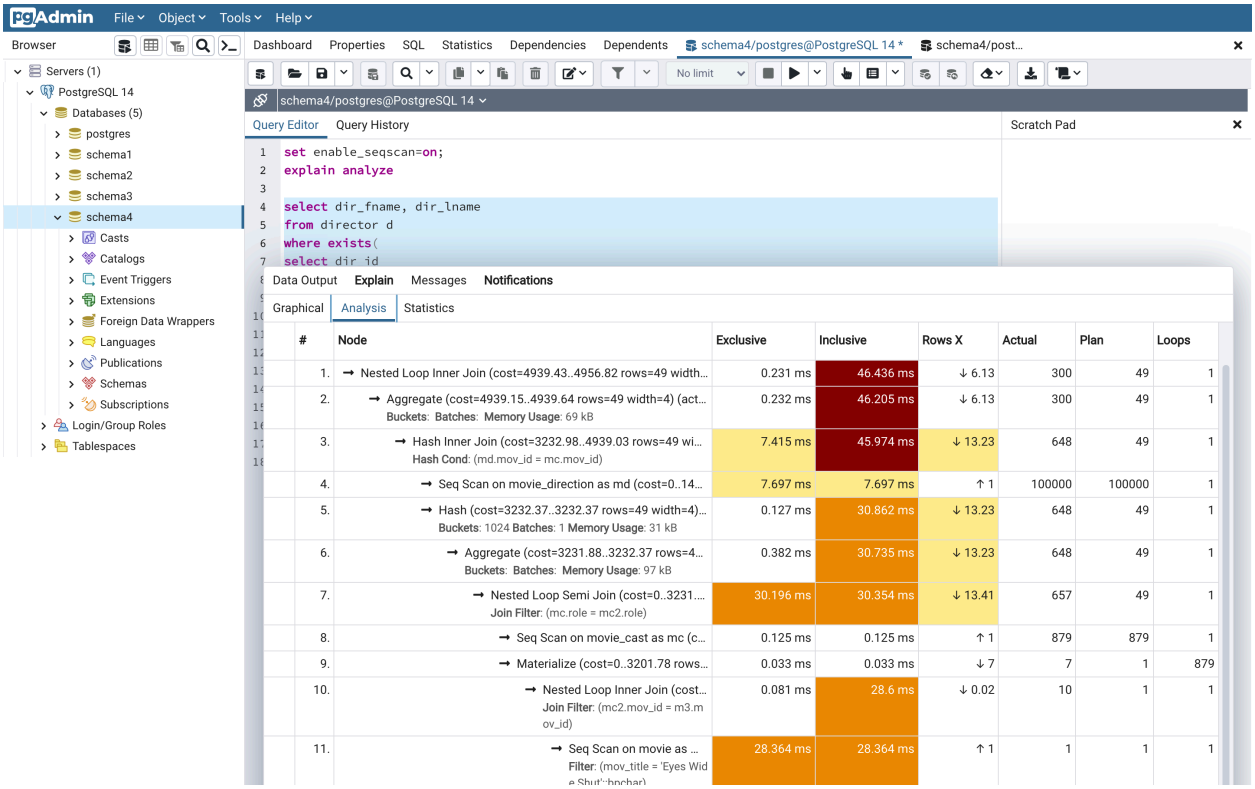
This query doesn't have a more optimized alternative. This is more of an equivalent alternative query that produces the same exact result set.

Query 11 with no index

Planning and execution time :



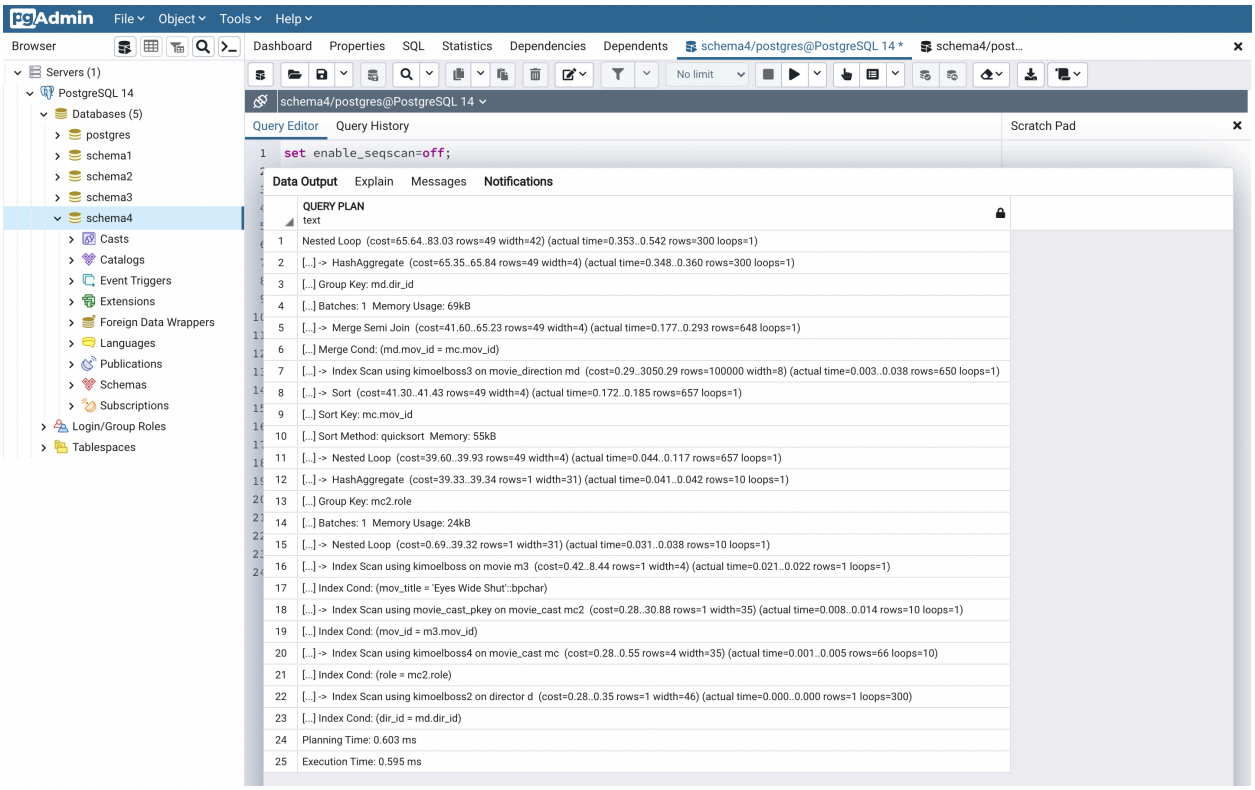
Execution plan and costs:



Query 11 using B+ indices on movie (mov_title) , director (dir_id), movie_direction (mov_id) , movie_cast (role)

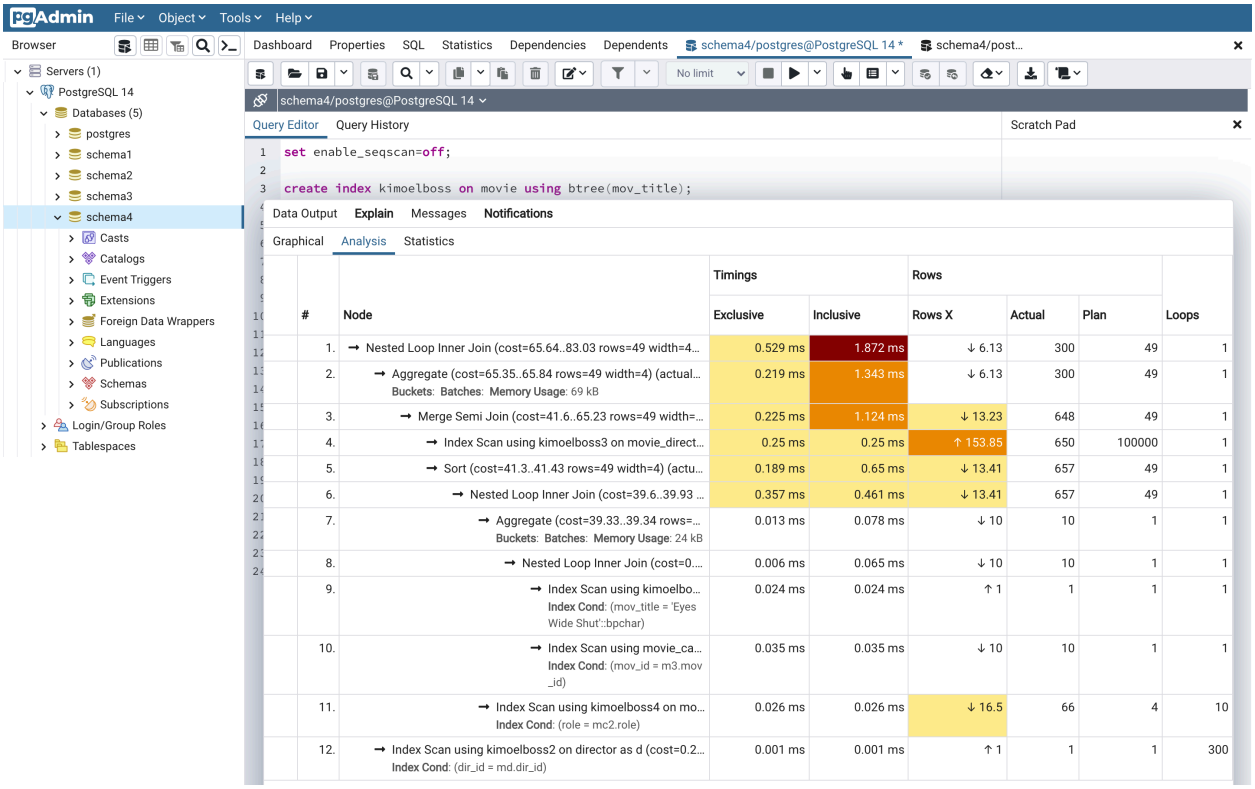
Disabled flags: seqscan

Planning and execution time :



Here the B+ tree optimized the query as we put it on the columns that we search on like dir_id and mov_id and the mov_title and the role as B+ tree search in O(logn) time so it is better than seqscan

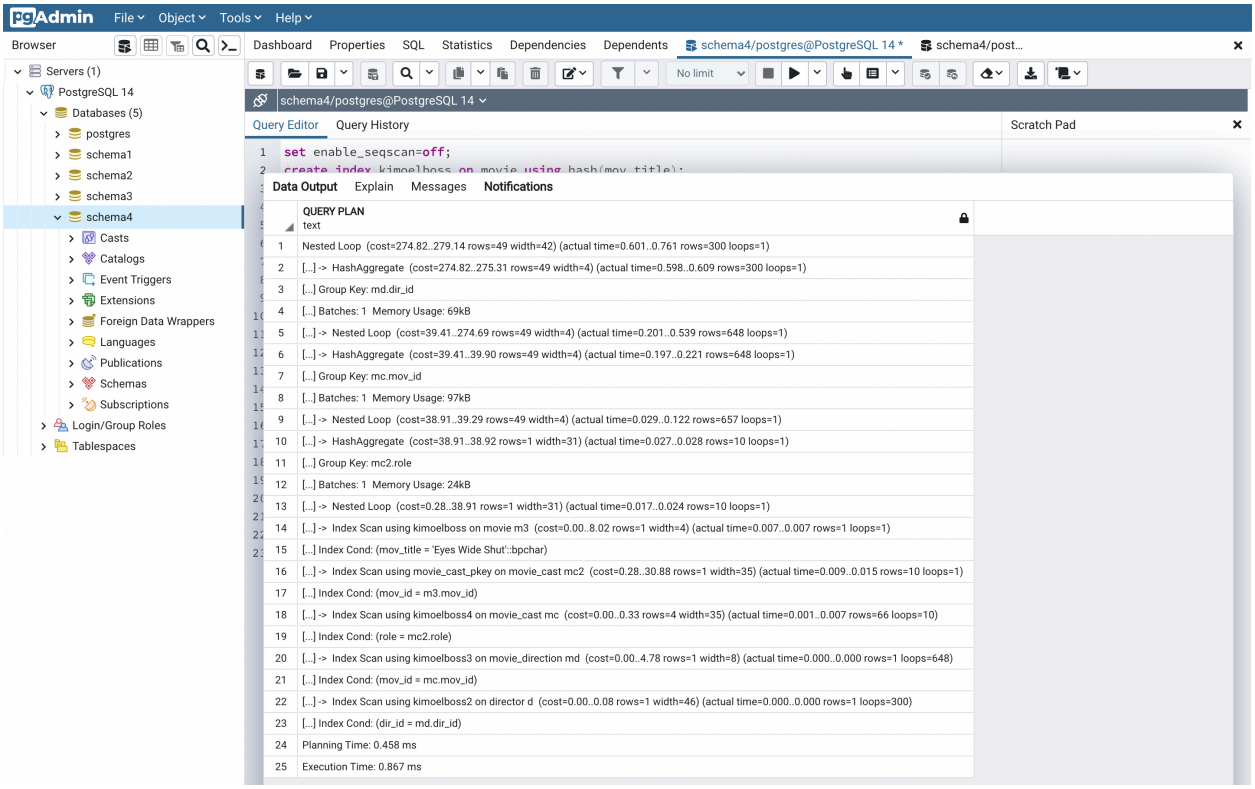
Execution plan and costs:



Query 11 using Hash index on movie (mov_title) , director (dir_id), movie_direction (mov_id) , movie_cast (role)

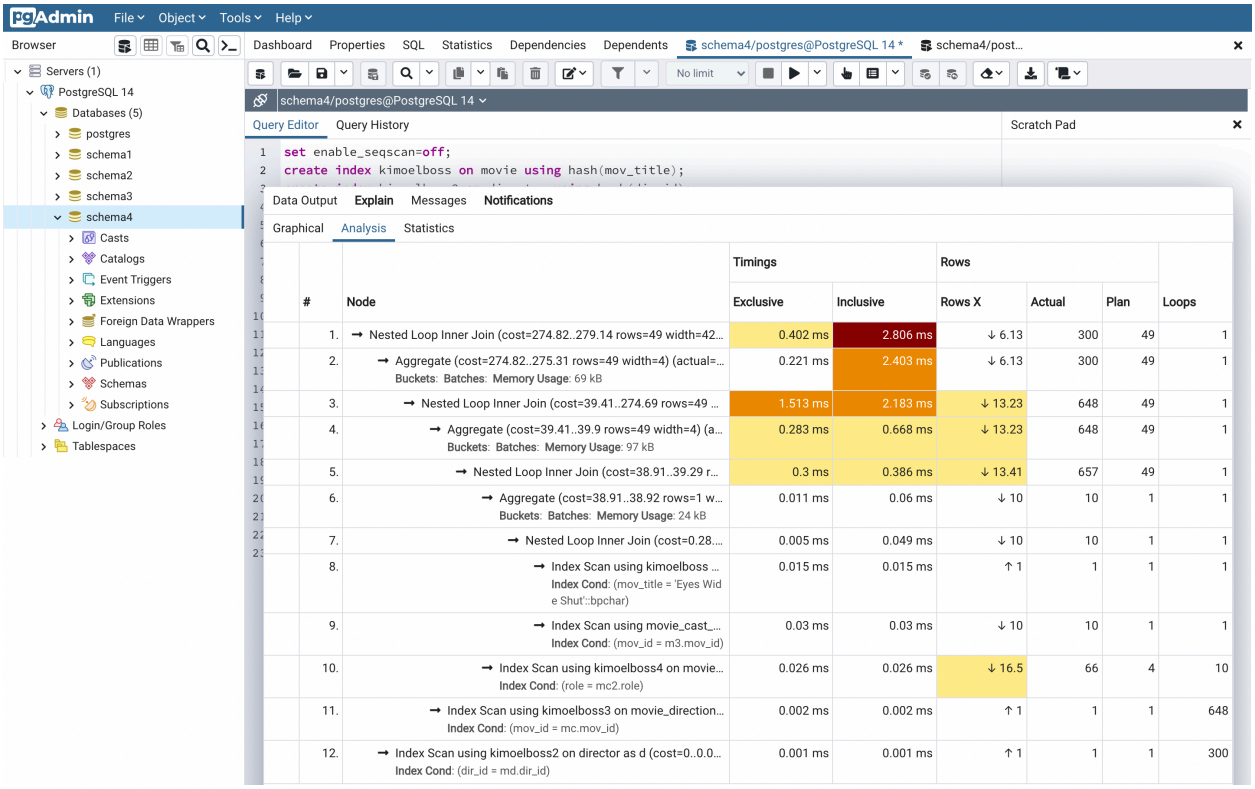
Disabled flags: seqscan

Planning and execution time :



Here the Hash tree optimized the query as we put it on the columns that we search on like dir_id and mov_id and the mov_title and the role as Hash takes O(1)

Execution plan and costs:

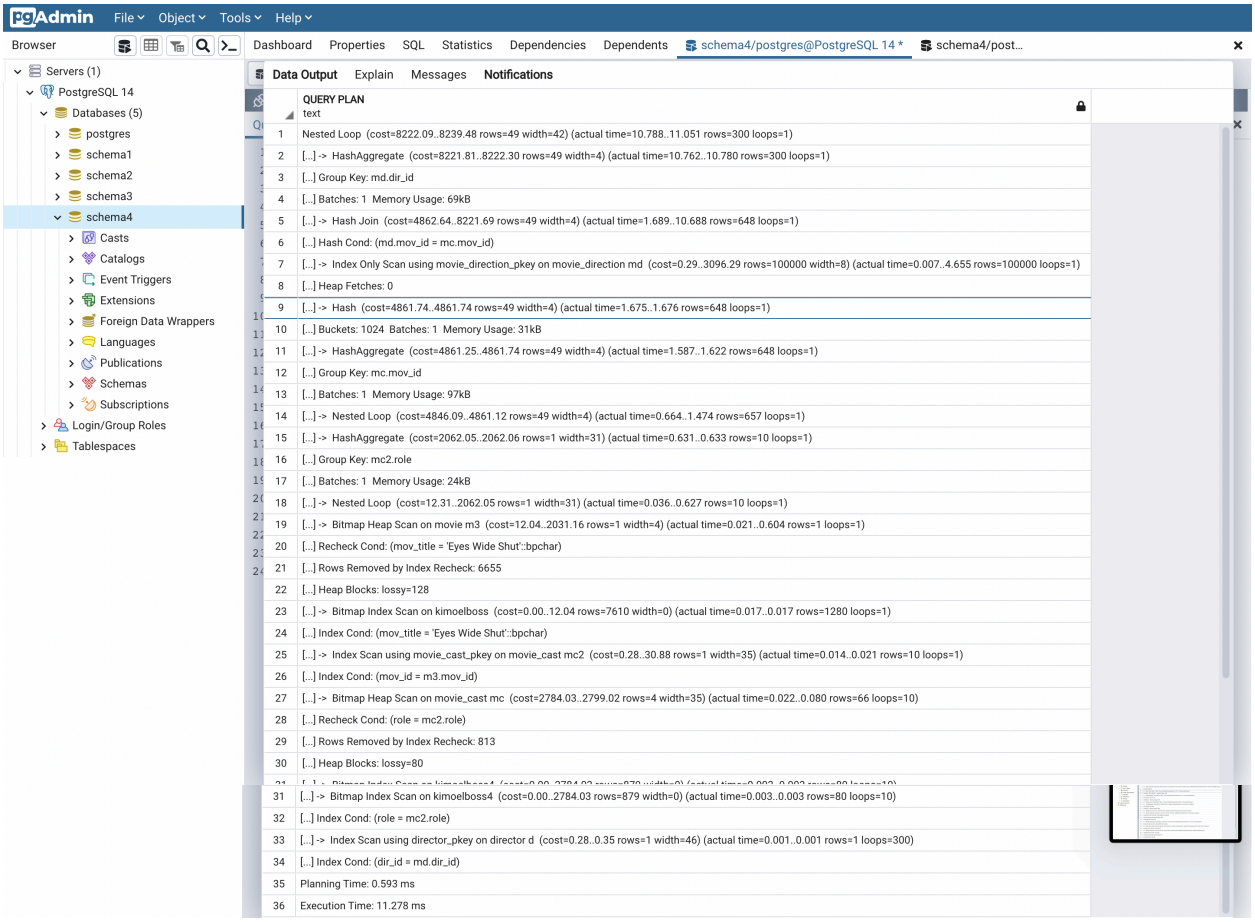


Query 11 using BRIN index on movie (mov_title) , director (dir_id), movie_direction (mov_id) , movie_cast (role)

Disabled flags: seqscan

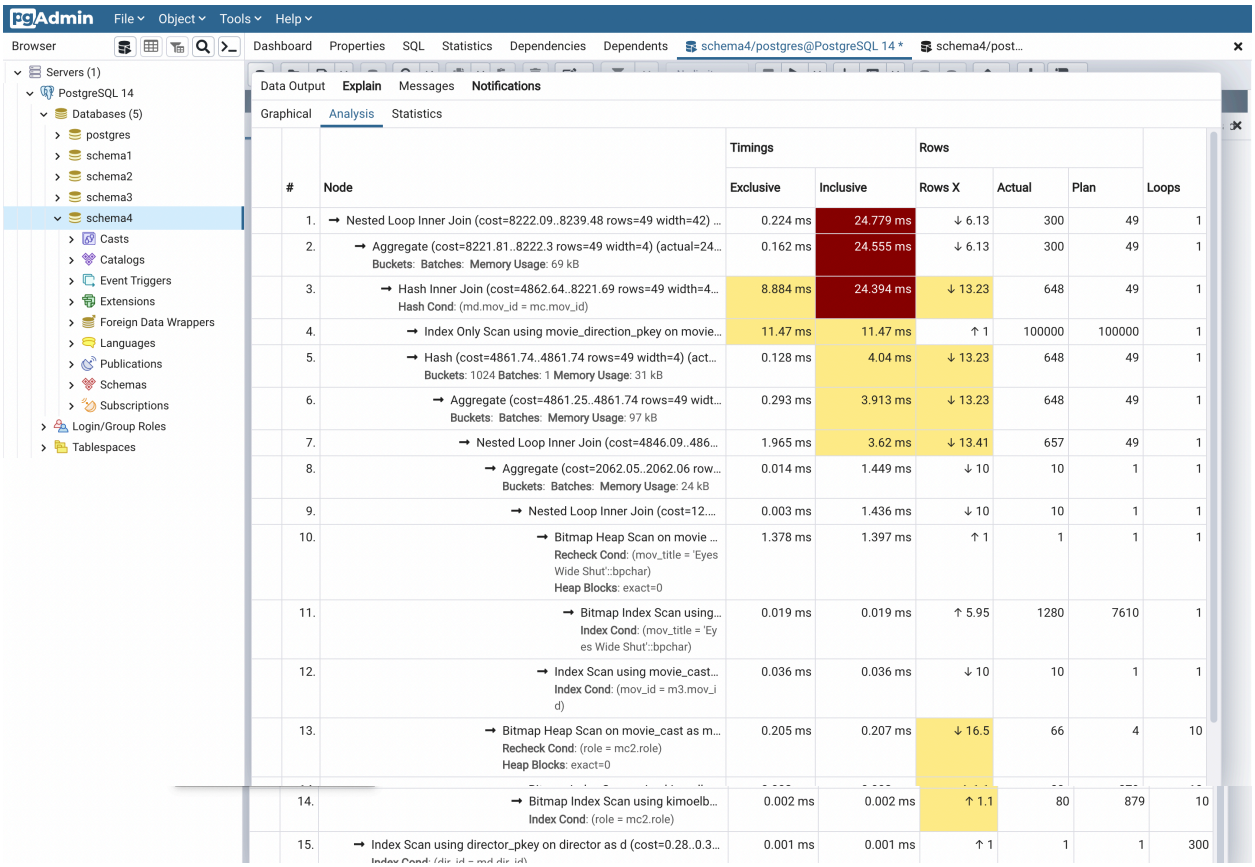
PRIMARY KEY ON

Planning and execution time :


pgAdmin
File
Object
Tools
Help
Browser
Dashboard
Properties
SQL
Statistics
Dependencies
Dependents
schema4/postgres@PostgreSQL 14 *
schema4/post...
Servers (1)
PostgreSQL 14
Databases (5)
postgres
schema1
schema2
schema3
schema4
Casts
Catalogs
Event Triggers
Extensions
Foreign Data Wrappers
Languages
Publications
Schemas
Subscriptions
Login/Group Roles
Tablespaces
Data Output
Explain
Messages
Notifications
QUERY PLAN
text
1 Nested Loop (cost=8222.09..8239.48 rows=49 width=42) (actual time=10.788..11.051 rows=300 loops=1)
2 [...] -> HashAggregate (cost=8221.81..8222.30 rows=49 width=4) (actual time=10.762..10.780 rows=300 loops=1)
3 [...] Group Key: md.dir_id
4 [...] Batches: 1 Memory Usage: 69kB
5 [...] -> Hash Join (cost=4862.64..8221.69 rows=49 width=4) (actual time=1.689..10.688 rows=648 loops=1)
6 [...] Hash Cond: (md.mov_id = mc.mov_id)
7 [...] -> Index Only Scan using movie_direction_pkey on movie_direction md (cost=0.29..3096.29 rows=100000 width=8) (actual time=0.007..4.655 rows=100000 loops=1)
8 [...] Heap Fetches: 0
9 [...] -> Hash (cost=4861.74..4861.74 rows=49 width=4) (actual time=1.675..1.676 rows=648 loops=1)
10 [...] Buckets: 1024 Batches: 1 Memory Usage: 31kB
11 [...] -> HashAggregate (cost=4861.25..4861.74 rows=49 width=4) (actual time=1.587..1.622 rows=648 loops=1)
12 [...] Group Key: mc.mov_id
13 [...] Batches: 1 Memory Usage: 97kB
14 [...] -> Nested Loop (cost=4846.09..4861.12 rows=49 width=4) (actual time=0.664..1.474 rows=657 loops=1)
15 [...] -> HashAggregate (cost=2062.05..2062.06 rows=1 width=31) (actual time=0.631..0.633 rows=10 loops=1)
16 [...] Group Key: mc2.role
17 [...] Batches: 1 Memory Usage: 24kB
18 [...] -> Nested Loop (cost=12.31..2062.05 rows=1 width=31) (actual time=0.036..0.627 rows=10 loops=1)
19 [...] -> Bitmap Heap Scan on movie m3 (cost=12.04..2031.16 rows=1 width=4) (actual time=0.021..0.604 rows=1 loops=1)
20 [...] Recheck Cond: (mov_title = 'Eyes Wide Shut'::bpchar)
21 [...] Rows Removed by Index Recheck: 6655
22 [...] Heap Blocks: lossy=128
23 [...] -> Bitmap Index Scan on kimoelboss (cost=0.00..12.04 rows=7610 width=0) (actual time=0.017..0.017 rows=1280 loops=1)
24 [...] Index Cond: (mov_title = 'Eyes Wide Shut'::bpchar)
25 [...] -> Index Scan using movie_cast_pkey on movie_cast mc2 (cost=0.28..30.88 rows=1 width=35) (actual time=0.014..0.021 rows=10 loops=1)
26 [...] Index Cond: (mov_id = m3.mov_id)
27 [...] -> Bitmap Heap Scan on movie_cast mc (cost=2784.03..2799.02 rows=4 width=35) (actual time=0.022..0.080 rows=66 loops=10)
28 [...] Recheck Cond: (role = mc2.role)
29 [...] Rows Removed by Index Recheck: 813
30 [...] Heap Blocks: lossy=80
31 [...] -> Bitmap Index Scan on kimoelboss4 (cost=0.00..2784.03 rows=879 width=0) (actual time=0.003..0.003 rows=80 loops=10)
32 [...] Index Cond: (role = mc2.role)
33 [...] -> Index Scan using director_pkey on director d (cost=0.28..0.35 rows=1 width=46) (actual time=0.001..0.001 rows=1 loops=300)
34 [...] Index Cond: (dir_id = md.dir_id)
35 Planning Time: 0.593 ms
36 Execution Time: 11.278 ms

Here the BRIN increased the cost significantly as it is most suitable in small range queries in big ranges. However that wasnot the case and the performance is much bad now

Execution plan and costs:

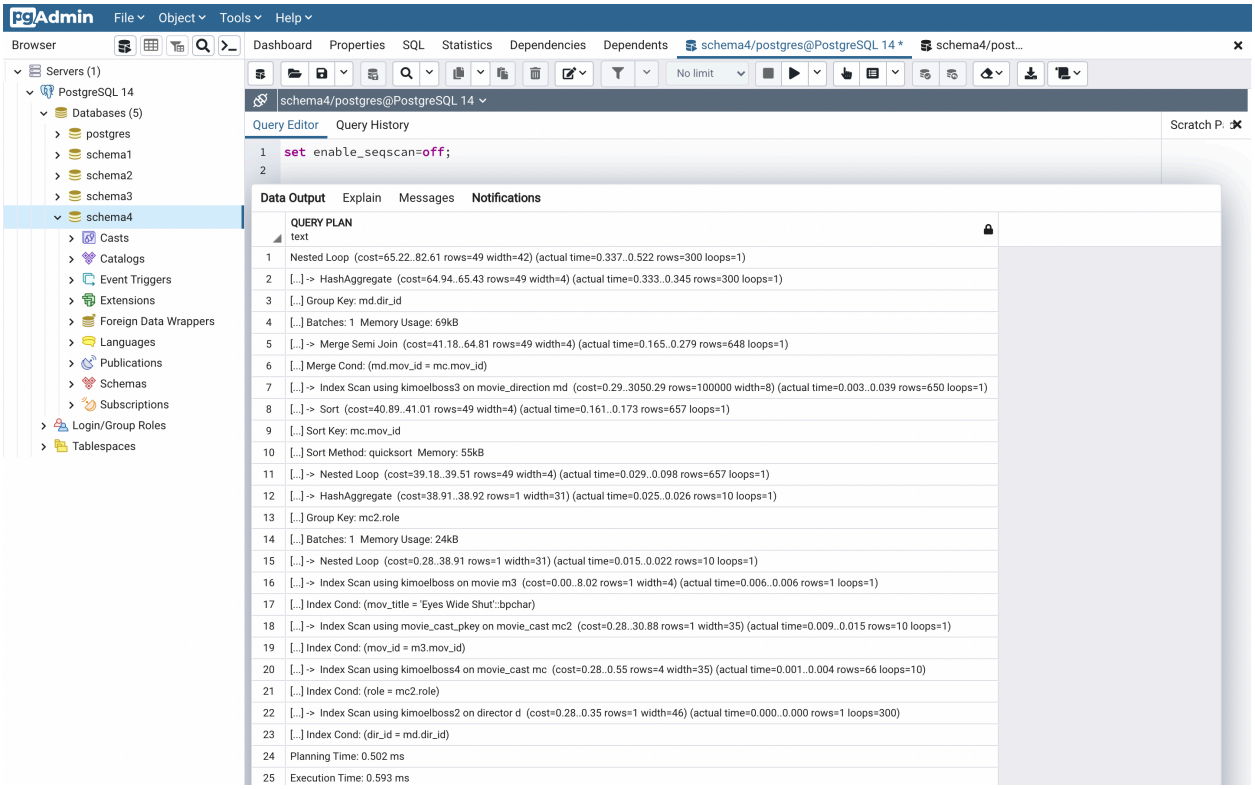


#	Node	Timings		Rows			
		Exclusive	Inclusive	Rows X	Actual	Plan	Loops
1.	→ Nested Loop Inner Join (cost=8222.09..8239.48 rows=49 width=42) ...	0.224 ms	24.779 ms	↓ 6.13	300	49	1
2.	→ Aggregate (cost=8221.81..8222.3 rows=49 width=4) (actual=24... Buckets: Batches: Memory Usage: 69 kB	0.162 ms	24.555 ms	↓ 6.13	300	49	1
3.	→ Hash Inner Join (cost=4862.64..8221.69 rows=49 width=4... Hash Cond: (md.mov_id = mc.mov_id)	8.884 ms	24.394 ms	↓ 13.23	648	49	1
4.	→ Index Only Scan using movie_direction_pkey on movie...	11.47 ms	11.47 ms	↑ 1	100000	100000	1
5.	→ Hash (cost=4861.74..4861.74 rows=49 width=4) (act... Buckets: 1024 Batches: 1 Memory Usage: 31 kB	0.128 ms	4.04 ms	↓ 13.23	648	49	1
6.	→ Aggregate (cost=4861.25..4861.74 rows=49 widt... Buckets: Batches: Memory Usage: 97 kB	0.293 ms	3.913 ms	↓ 13.23	648	49	1
7.	→ Nested Loop Inner Join (cost=4846.09..486...	1.965 ms	3.62 ms	↓ 13.41	657	49	1
8.	→ Aggregate (cost=2062.05..2062.06 row... Buckets: Batches: Memory Usage: 24 kB	0.014 ms	1.449 ms	↓ 10	10	1	1
9.	→ Nested Loop Inner Join (cost=12....	0.003 ms	1.436 ms	↓ 10	10	1	1
10.	→ Bitmap Heap Scan on movie ... Recheck Cond: (mov_title = 'Eyes Wide Shut'::bpchar) Heap Blocks: exact=0	1.378 ms	1.397 ms	↑ 1	1	1	1
11.	→ Bitmap Index Scan using... Index Cond: (mov_title = 'Eyes Wide Shut'::bpchar)	0.019 ms	0.019 ms	↑ 5.95	1280	7610	1
12.	→ Index Scan using movie_cast... Index Cond: (mov_id = m3.mov_id)	0.036 ms	0.036 ms	↓ 10	10	1	1
13.	→ Bitmap Heap Scan on movie_cast as m... Recheck Cond: (role = mc2.role) Heap Blocks: exact=0	0.205 ms	0.207 ms	↓ 16.5	66	4	10
14.	→ Bitmap Index Scan using kimoelb... Index Cond: (role = mc2.role)	0.002 ms	0.002 ms	↑ 1.1	80	879	10
15.	→ Index Scan using director_pkey on director as d (cost=0.28..0.3...	0.001 ms	0.001 ms	↑ 1	1	1	300

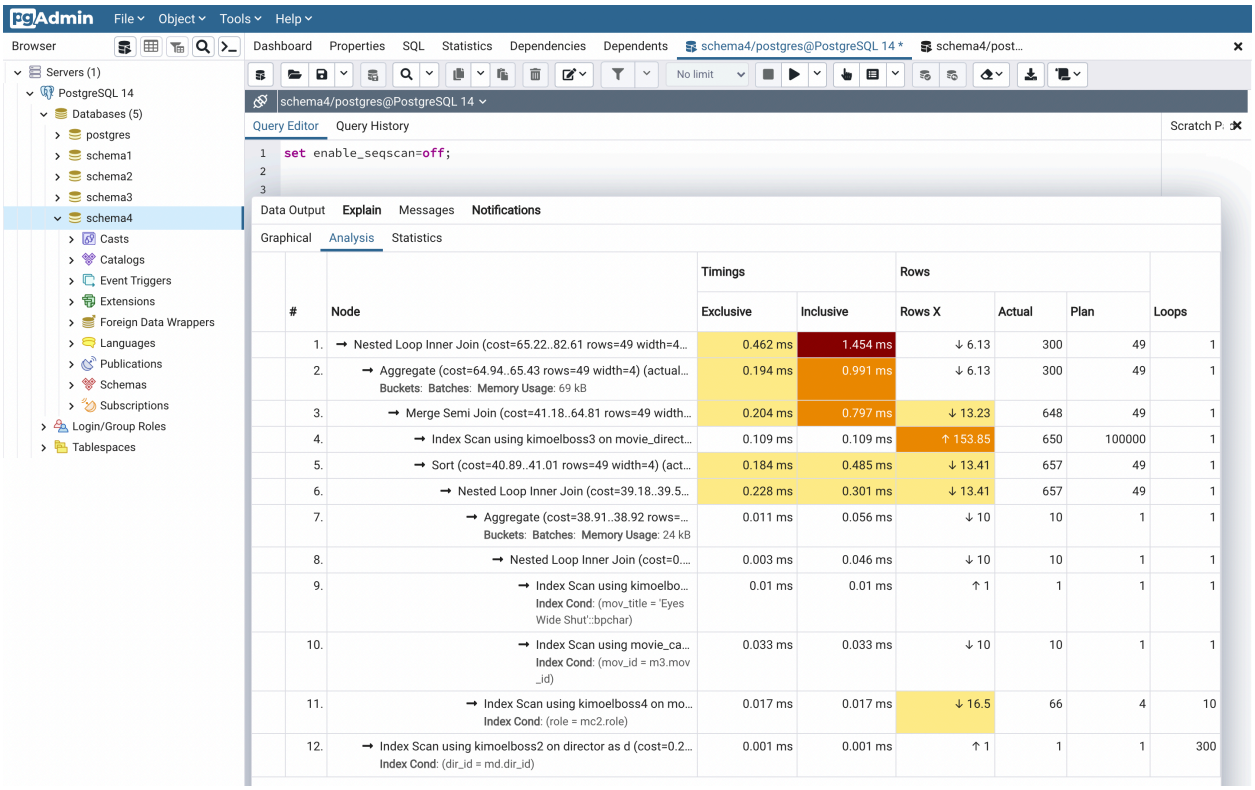
Query 11 using Hash index on movie (mov_title) , and B+ tree on director (dir_id), movie_direction (mov_id) , movie_cast (role)

Disabled flags: seqscan

Planning and execution time :



Execution plan and costs:



Both B+ tree and hash increased performance as the search in O(logn) and O(1) time

QUERY 12

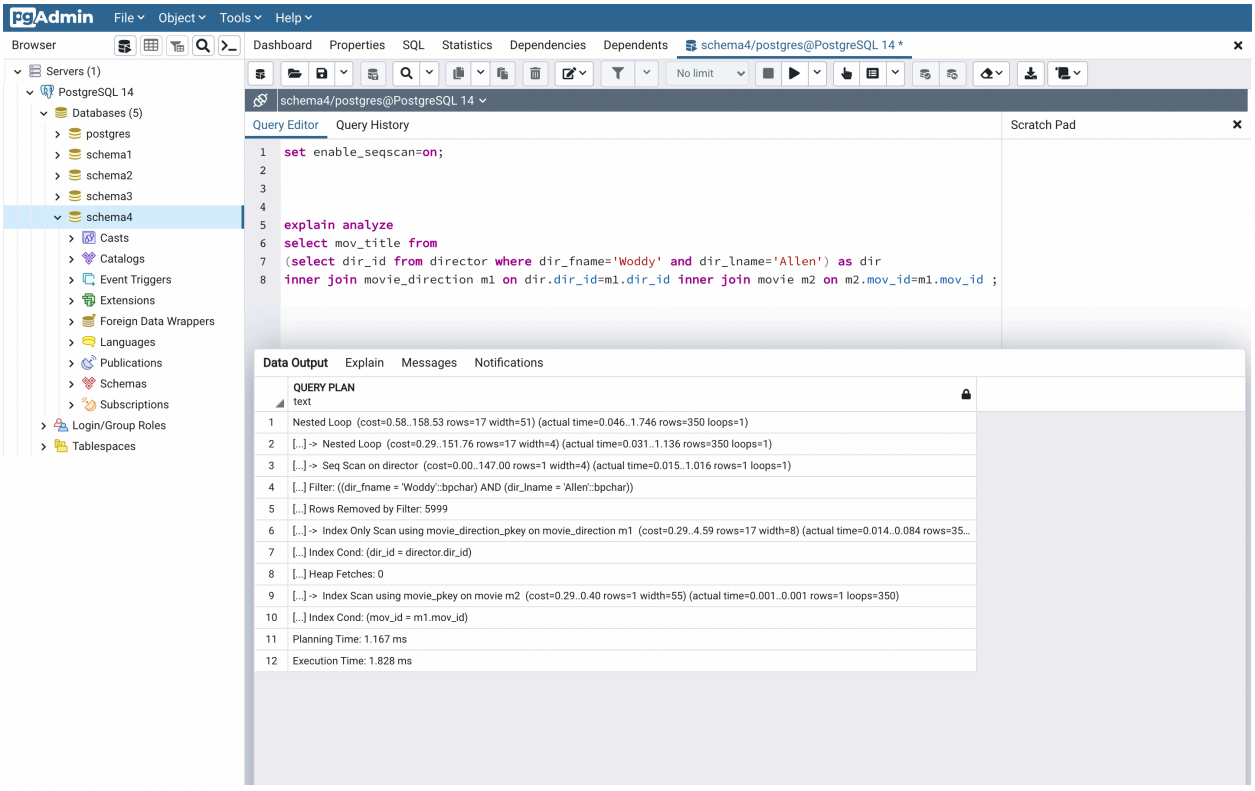
QUERY 12 AFTER OPTIMIZATION IS

```
select mov_title from
(select dir_id from director where dir_fname='Woddy' and dir_lname='Allen') as dir
inner join movie_direction m1 on dir.dir_id=m1.dir_id inner join movie m2 on
m2.mov_id=m1.mov_id ;
```

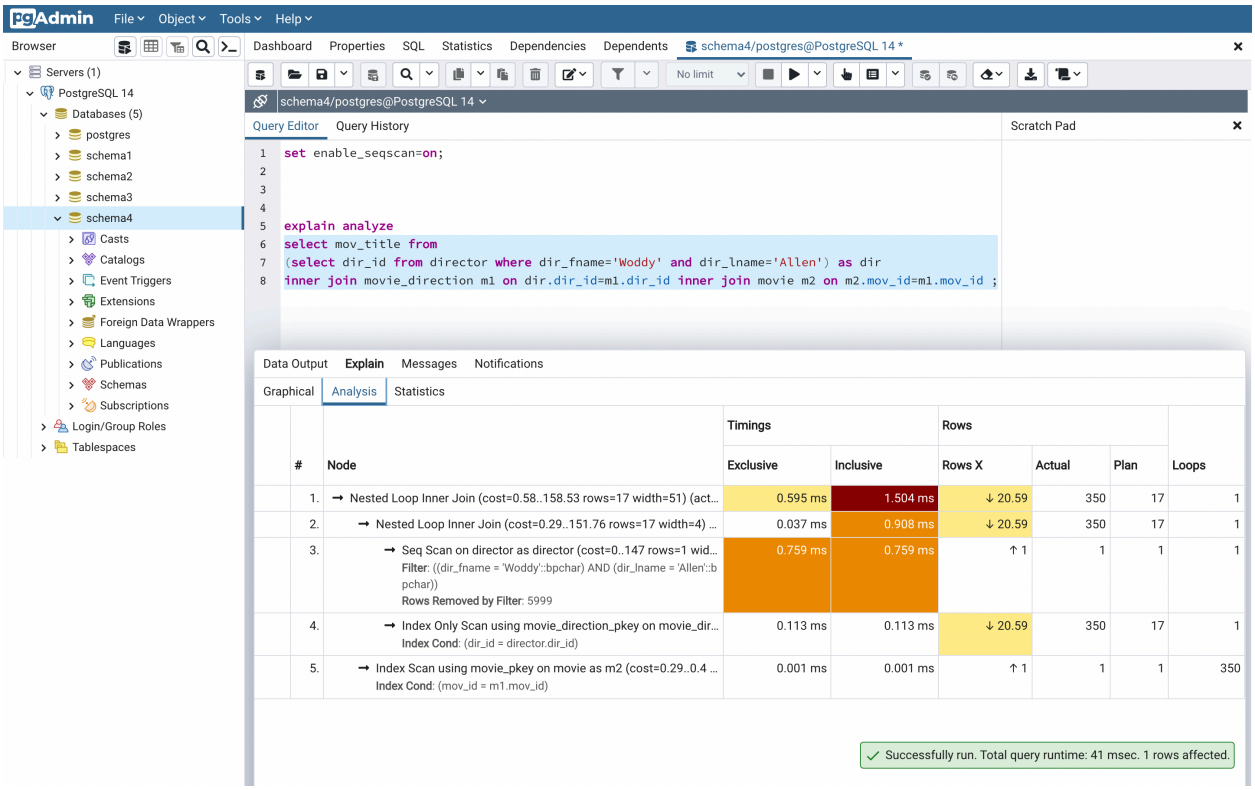
This is optimized due to the push down slection and the use of inner join instead of IN making it more optimized

Query 12 with no index

Planning and execution time :



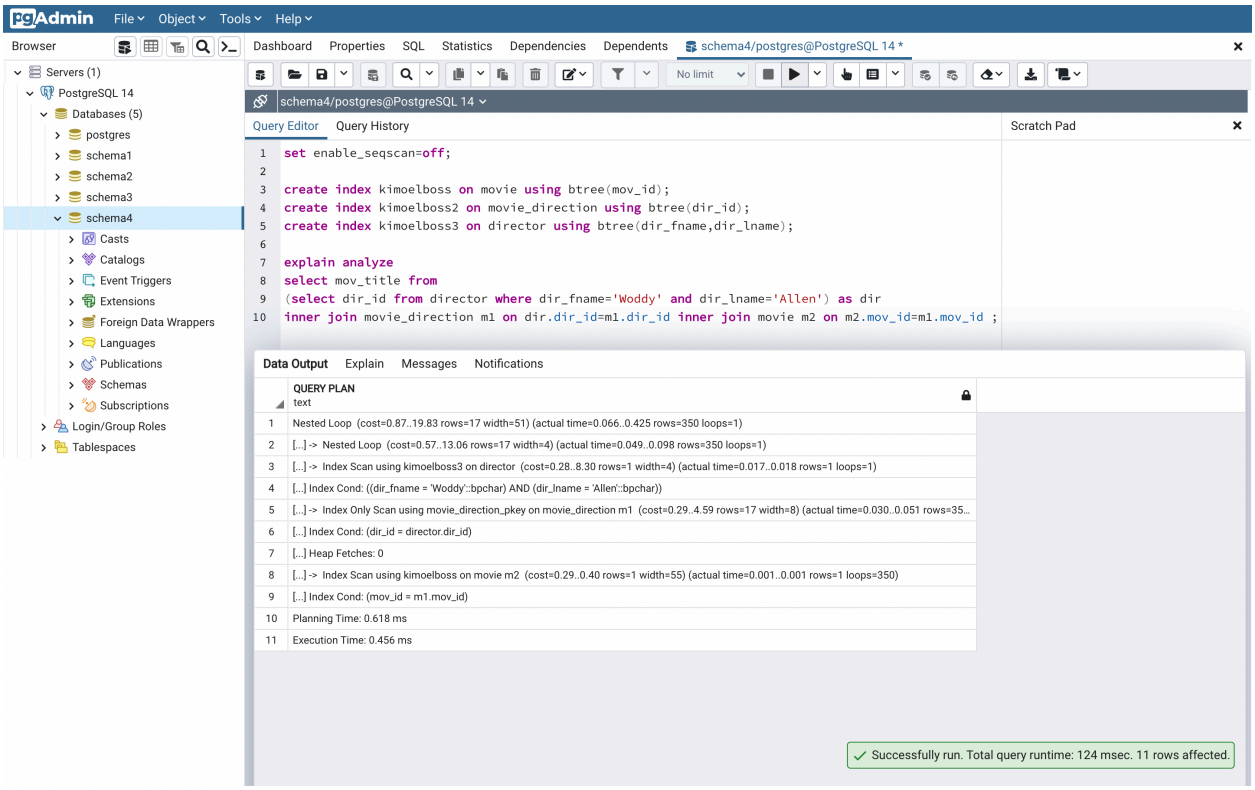
Execution plan and costs:



Query 12 using B+ indices on movie (mov_id) , director (dir_fname,dir_lname), movie_direction (dir_id)

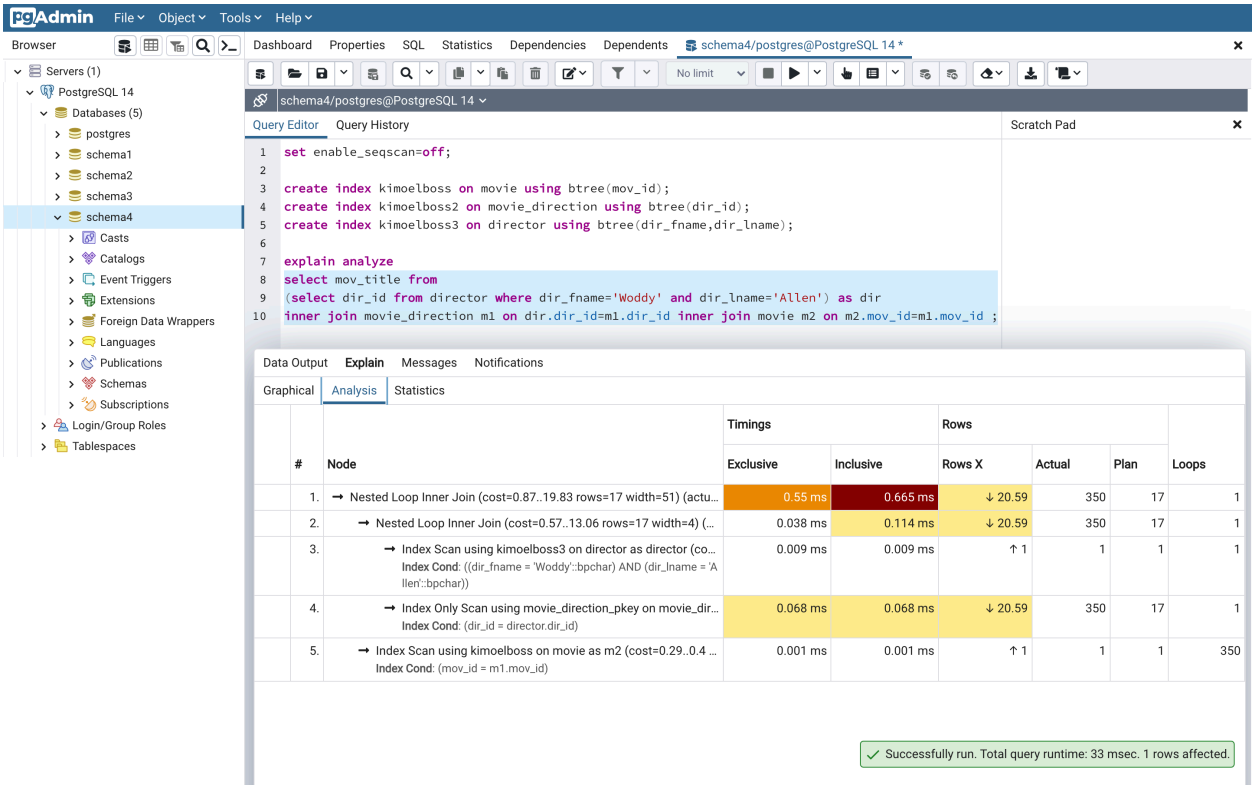
Disabled flags: seqscan

Planning and execution time :



Here the B+ tree optimized the query as we put it on the columns that we search on like dir_fname and dir_lname TOGETHER and mov_id and the dir_id and the role as B+ tree search in O(logn) time so it is better than seqscan

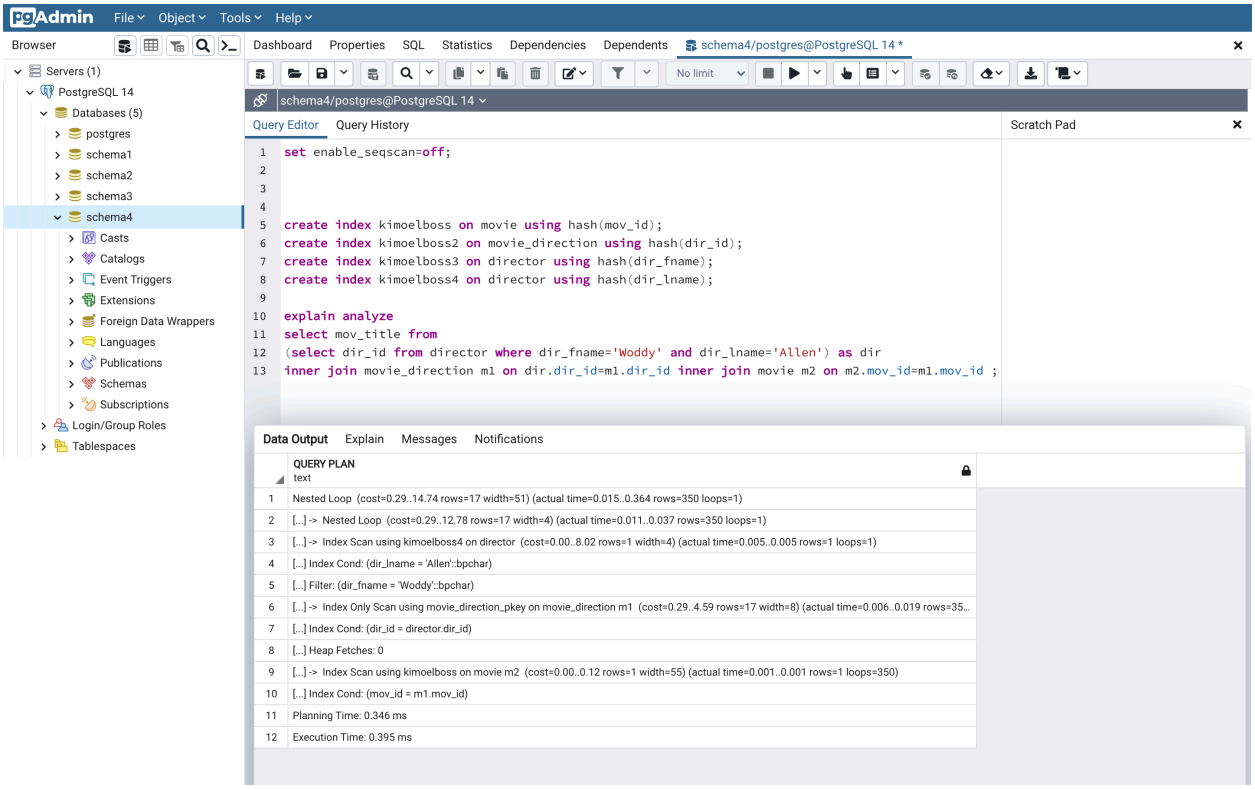
Execution plan and costs:



Query 12 using Hash indices on movie (mov_id) , director (dir_fname), director dir_lname), movie_direction (dir_id)

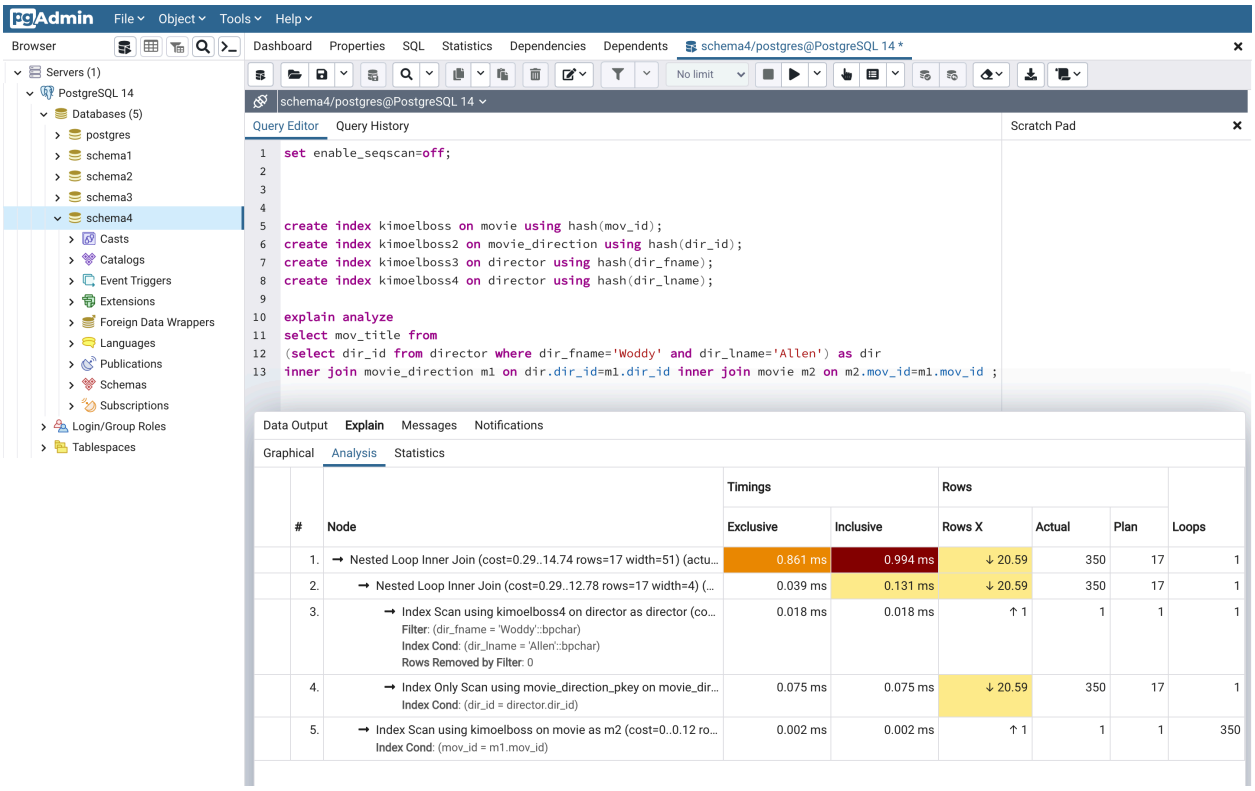
Disabled flags: seqscan

Planning and execution time :



Here the Hash tree optimized the query as we put it on the columns that we search on like dir_id and mov_id and the dir_fname and lname and the role as Hash takes O(1)

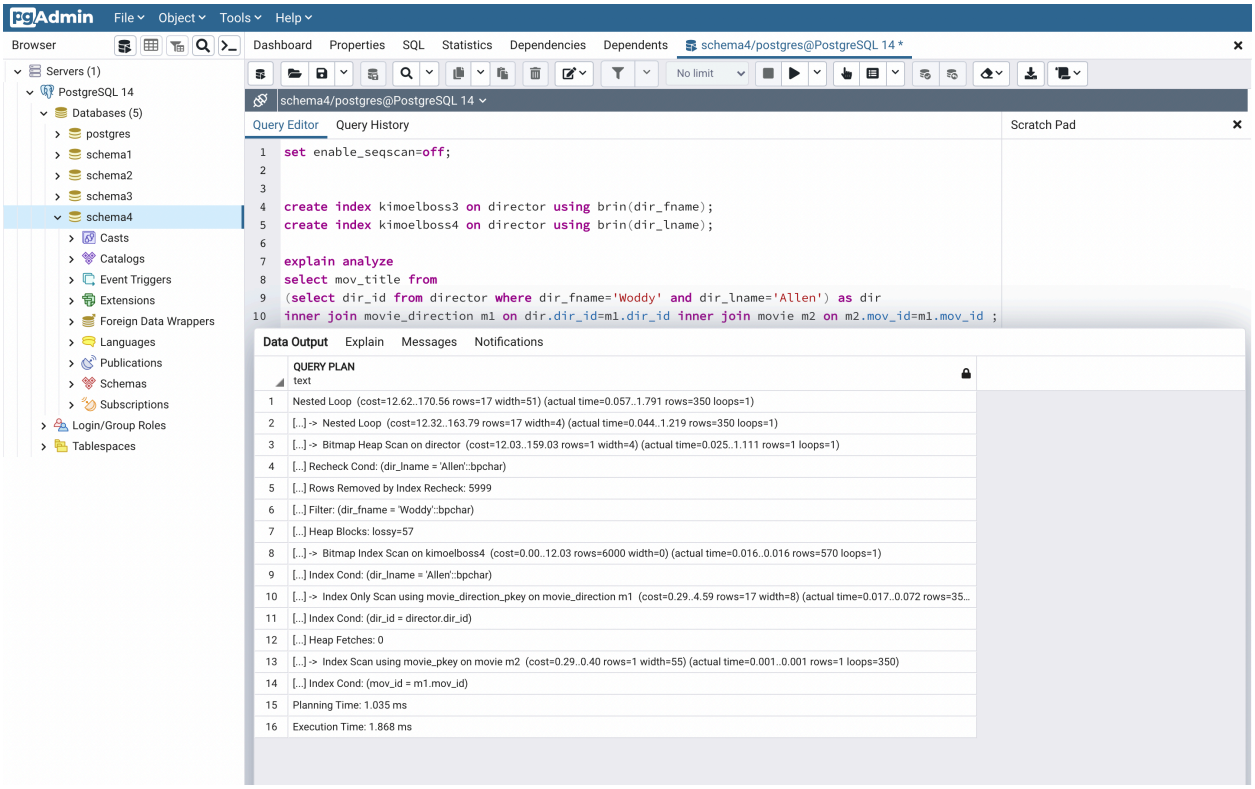
Execution plan and costs:



Query 12 using BRIN indices on director (dir_fname), director dir_lname),

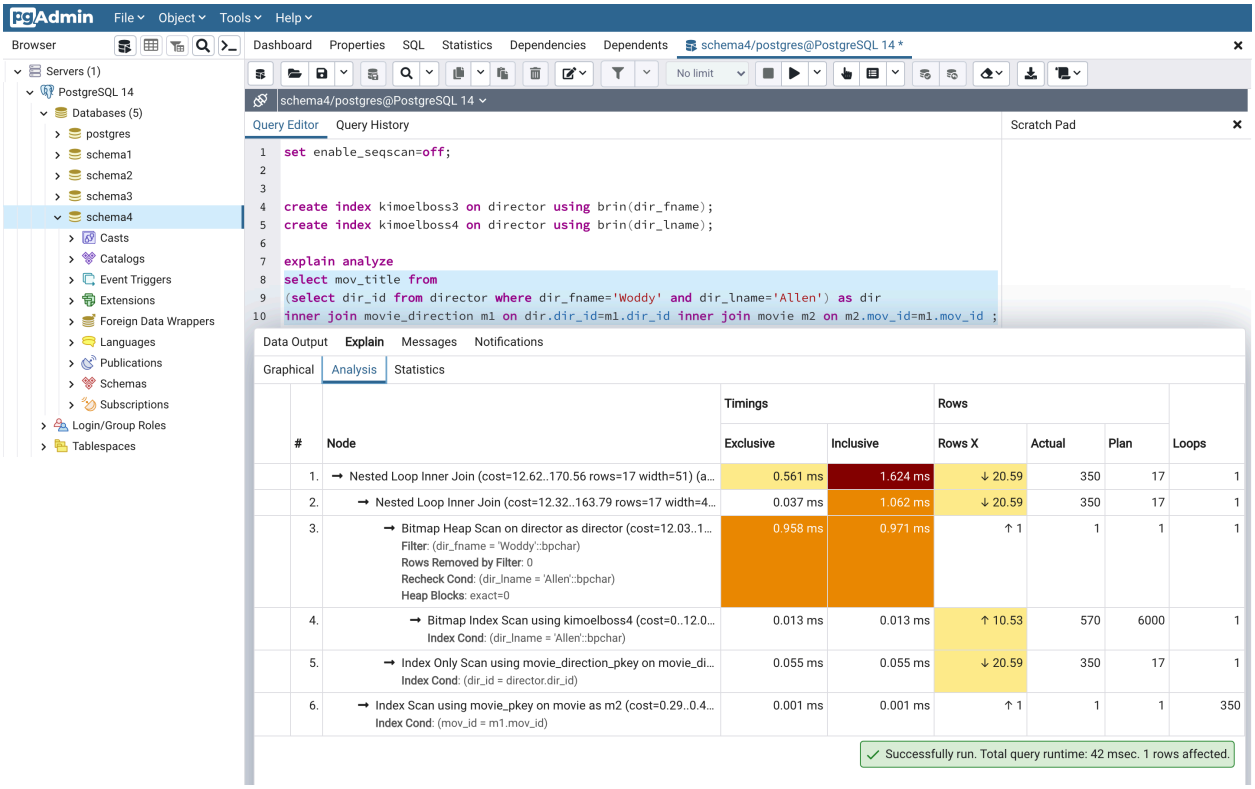
Disabled flags: seqscan

Planning and execution time :



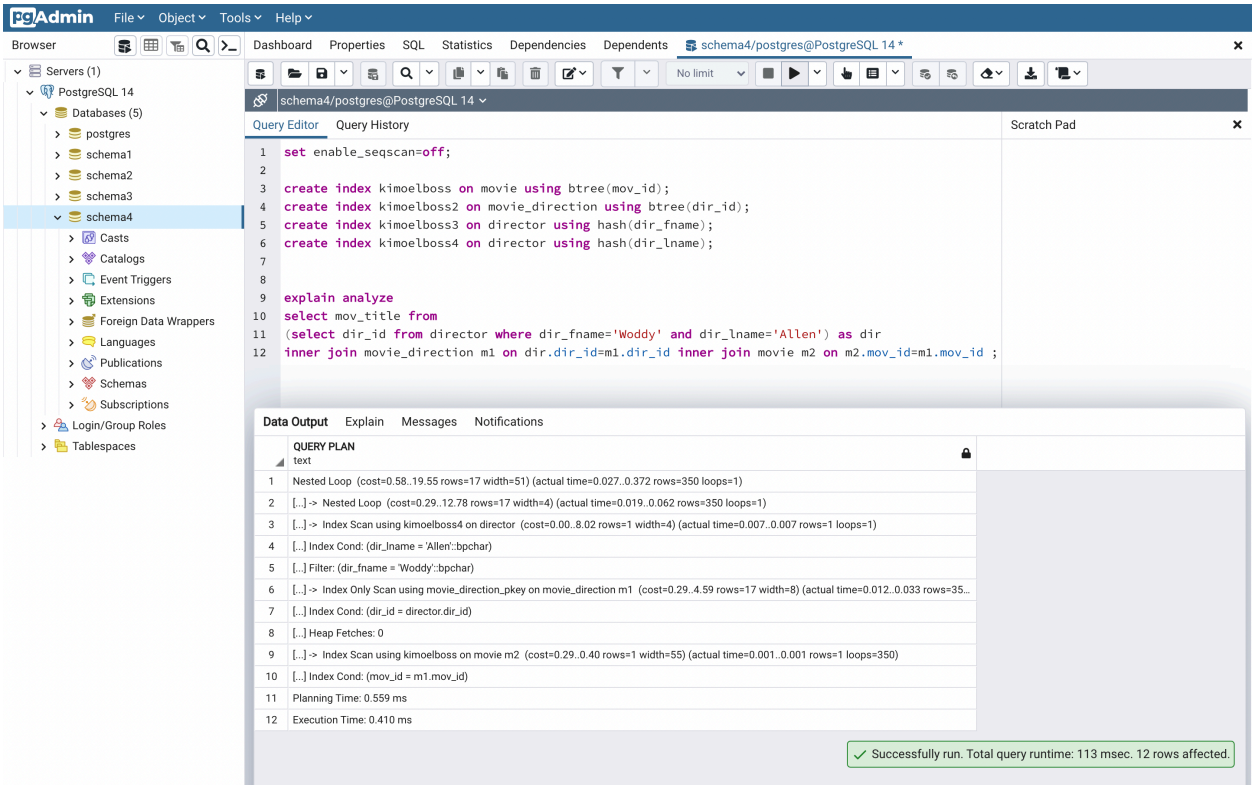
Here the BRIN increased the cost as it is most suitable in small range queries in big ranges. However that wasnot the case and the performance is bad now from 293 to 305. We used then on director first name and last name

Execution plan and costs:



Query 12 using Hash indices on director (dir_fname), director dir_lname) and Btree on movie (mov_id) , movie_direction (dir_id)

Disabled flags: seqscan
Planning and execution time :



Execution plan and costs: