Query 12

- Find the titles of all movies directed by the director whose first and last name are Woddy Allen.

Note !

- all flags are set to default

Original Query

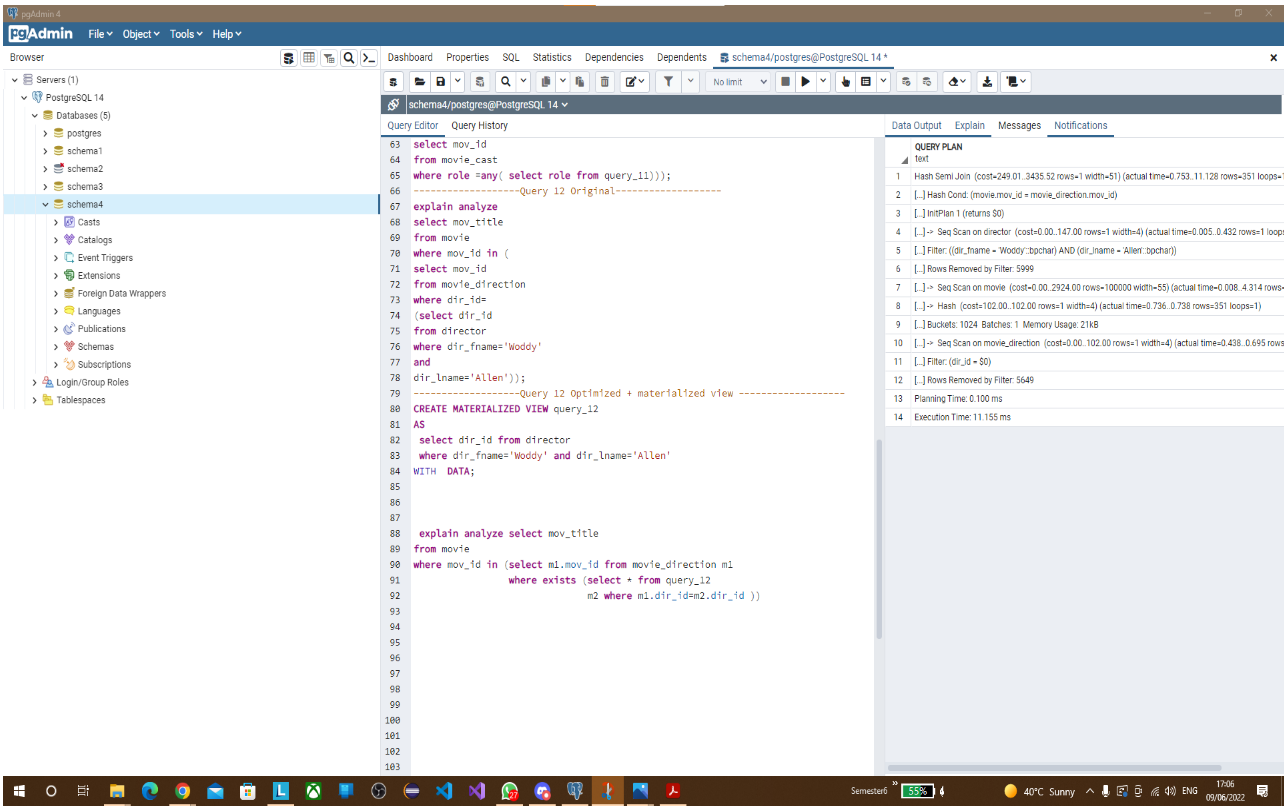
```
select mov_title
from movie
where mov_id in (
select mov_id
from movie_direction
where dir_id=
(select dir_id
from director
where dir_fname='Woddy'
and
dir_lname='Allen'));
```

Result Set

- 387 Rows

Report

1. given query without an index :



```sql
select mov_id
from movie_cast
where role =any( select role from query_11)));
-------------------Query 12 Original-------------------
explain analyze
select mov_title
from movie
where mov_id in (
select mov_id
from movie_direction
where dir_id=
(select dir_id
from director
where dir_fname='Woddy'
and
dir_lname='Allen'));
-------------------Query 12 Optimized + materialized view -------------------
CREATE MATERIALIZED VIEW query_12
AS
 select dir_id from director
 where dir_fname='Woddy' and dir_lname='Allen'
WITH  DATA;



 explain analyze select mov_title
from movie
where mov_id in (select m1.mov_id from movie_direction m1
                where exists (select * from query_12
                             m2 where m1.dir_id=m2.dir_id ))
```

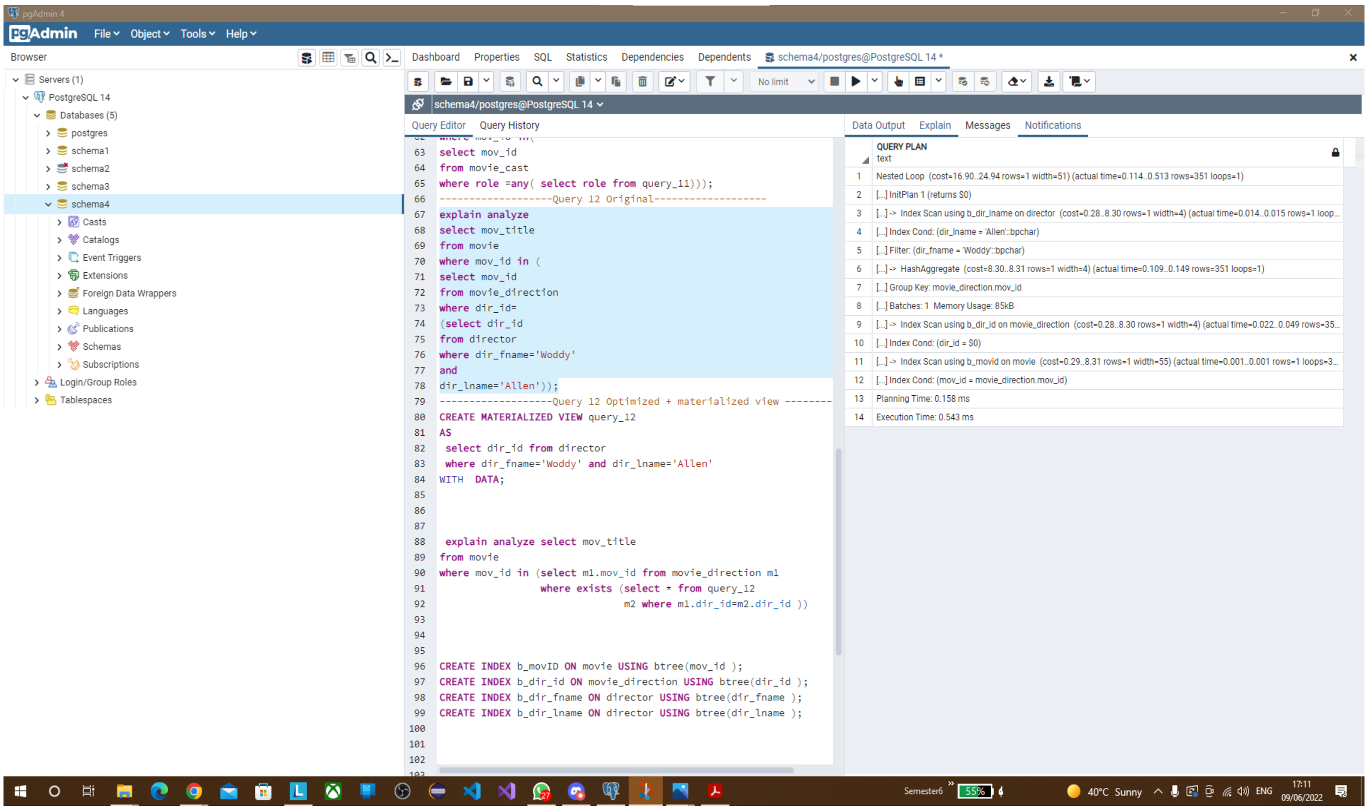
```
QUERY PLAN
text
1   Hash Semi Join  (cost=249.01..3435.52 rows=1 width=51) (actual time=0.753..11.128 rows=351 loops=1
2   [...] Hash Cond: (movie.mov_id = movie_direction.mov_id)
3   [...] InitPlan 1 (returns $0)
4   [...] ->  Seq Scan on director  (cost=0.00..147.00 rows=1 width=4) (actual time=0.005..0.432 rows=1 loops
5   [...] Filter: ((dir_fname = 'Woddy'::bpchar) AND (dir_lname = 'Allen'::bpchar))
6   [...] Rows Removed by Filter: 5999
7   [...] ->  Seq Scan on movie  (cost=0.00..2924.00 rows=100000 width=55) (actual time=0.008..4.314 rows=
8   [...] ->  Hash  (cost=102.00..102.00 rows=1 width=4) (actual time=0.736..0.738 rows=351 loops=1)
9   [...] Buckets: 1024  Batches: 1  Memory Usage: 21kB
10  [...] ->  Seq Scan on movie_direction  (cost=0.00..102.00 rows=1 width=4) (actual time=0.438..0.695 rows
11  [...] Filter: (dir_id = $0)
12  [...] Rows Removed by Filter: 5649
13  Planning Time: 0.100 ms
14  Execution Time: 11.155 ms
```

Explanation :

- Metrics :

Execution Time : 11.155 ms   Total Expected Cost : 3435.52

2. given query with B+ trees indices only :



```sql
select mov_id
from movie_cast
where role =any( select role from query_11)));
-------------------Query 12 Original-------------------
explain analyze
select mov_title
from movie
where mov_id in (
select mov_id
from movie_direction
where dir_id=
(select dir_id
from director
where dir_fname='Woddy'
and
dir_lname='Allen'));
-------------------Query 12 Optimized + materialized view --------
CREATE MATERIALIZED VIEW query_12
AS
 select dir_id from director
 where dir_fname='Woddy' and dir_lname='Allen'
WITH  DATA;



 explain analyze select mov_title
from movie
where mov_id in (select m1.mov_id from movie_direction m1
                where exists (select * from query_12
                             m2 where m1.dir_id=m2.dir_id ))



CREATE INDEX b_movID ON movie USING btree(mov_id );
CREATE INDEX b_dir_id ON movie_direction USING btree(dir_id );
CREATE INDEX b_dir_fname ON director USING btree(dir_fname );
CREATE INDEX b_dir_lname ON director USING btree(dir_lname );
```

```
QUERY PLAN
text
1   Nested Loop  (cost=16.90..24.94 rows=1 width=51) (actual time=0.114..0.513 rows=351 loops=1)
2   [...] InitPlan 1 (returns $0)
3   [...] ->  Index Scan using b_dir_lname on director  (cost=0.28..8.30 rows=1 width=4) (actual time=0.014..0.015 rows=1 loop...
4   [...] Index Cond: (dir_lname = 'Allen'::bpchar)
5   [...] Filter: (dir_fname = 'Woddy'::bpchar)
6   [...] ->  HashAggregate  (cost=8.30..8.31 rows=1 width=4) (actual time=0.109..0.149 rows=351 loops=1)
7   [...] Group Key: movie_direction.mov_id
8   [...] Batches: 1  Memory Usage: 85kB
9   [...] ->  Index Scan using b_dir_id on movie_direction  (cost=0.28..8.30 rows=1 width=4) (actual time=0.022..0.049 rows=35...
10  [...] Index Cond: (dir_id = $0)
11  [...] ->  Index Scan using b_movid on movie  (cost=0.29..8.31 rows=1 width=55) (actual time=0.001..0.001 rows=1 loops=3...
12  [...] Index Cond: (mov_id = movie_direction.mov_id)
13  Planning Time: 0.158 ms
14  Execution Time: 0.543 ms
```

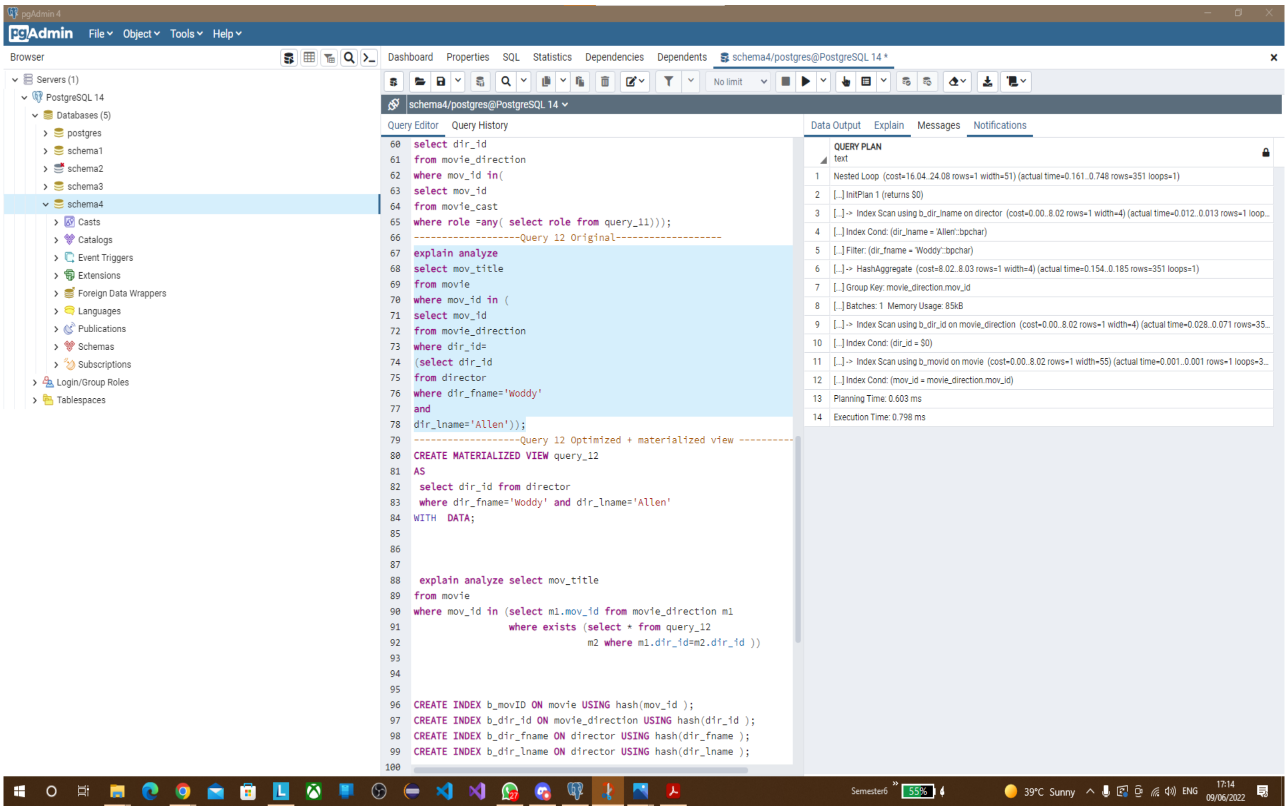
Explanation :

- Metrics :

  Execution Time : 0.543 ms   Total Expected Cost : 24.94

- Indices created as shown on above screenshot.
- The Search is done with O(Log n) for (dir_lname='Allen') for example or any search using columns index created on (Total Expected Cost Was 3435.52 and became 24.94).

3. given query with hash indices only :



```
select dir_id
from movie_direction
where mov_id in(
select mov_id
from movie_cast
where role =any( select role from query_11)));
------------------Query 12 Original------------------
explain analyze
select mov_title
from movie
where mov_id in (
select mov_id
from movie_direction
where dir_id=
(select dir_id
from director
where dir_fname='Woddy'
and
dir_lname='Allen'));
------------------Query 12 Optimized + materialized view ---------
CREATE MATERIALIZED VIEW query_12
AS
 select dir_id from director
 where dir_fname='Woddy' and dir_lname='Allen'
WITH  DATA;



 explain analyze select mov_title
from movie
where mov_id in (select m1.mov_id from movie_direction m1
                 where exists (select * from query_12
                               m2 where m1.dir_id=m2.dir_id ))



CREATE INDEX b_movID ON movie USING hash(mov_id );
CREATE INDEX b_dir_id ON movie_direction USING hash(dir_id );
CREATE INDEX b_dir_fname ON director USING hash(dir_fname );
CREATE INDEX b_dir_lname ON director USING hash(dir_lname );
```

```
QUERY PLAN
text
1  Nested Loop  (cost=16.04..24.08 rows=1 width=51) (actual time=0.161..0.748 rows=351 loops=1)
2  [...] InitPlan 1 (returns $0)
3  [...] ->  Index Scan using b_dir_lname on director  (cost=0.00..8.02 rows=1 width=4) (actual time=0.012..0.013 rows=1 loop...
4  [...] Index Cond: (dir_lname = 'Allen'::bpchar)
5  [...] Filter: (dir_fname = 'Woddy'::bpchar)
6  [...] ->  HashAggregate  (cost=8.02..8.03 rows=1 width=4) (actual time=0.154..0.185 rows=351 loops=1)
7  [...] Group Key: movie_direction.mov_id
8  [...] Batches: 1  Memory Usage: 85kB
9  [...] ->  Index Scan using b_dir_id on movie_direction  (cost=0.00..8.02 rows=1 width=4) (actual time=0.028..0.071 rows=35...
10 [...] Index Cond: (dir_id = $0)
11 [...] ->  Index Scan using b_movid on movie  (cost=0.00..8.02 rows=1 width=55) (actual time=0.001..0.001 rows=1 loops=3...
12 [...] Index Cond: (mov_id = movie_direction.mov_id)
13 Planning Time: 0.603 ms
14 Execution Time: 0.798 ms
```

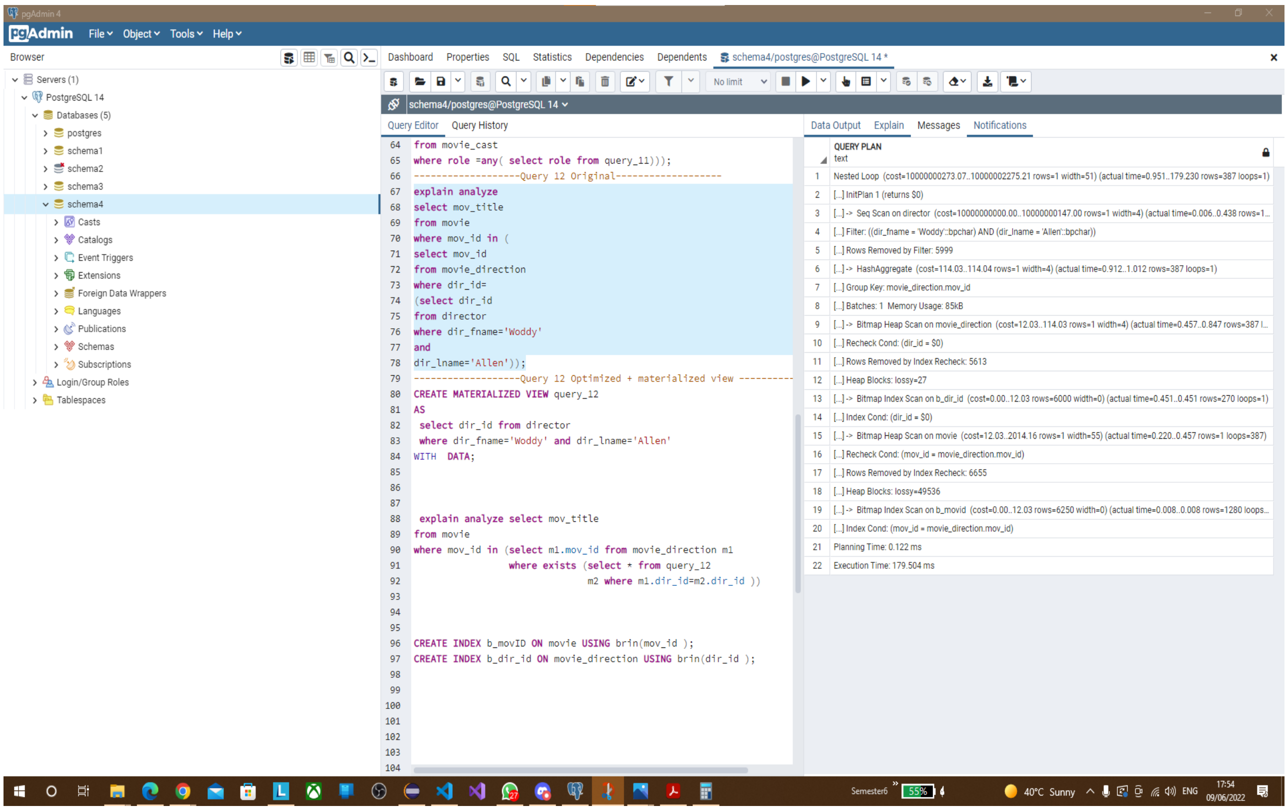
Explanation :

- Metrics :

  Execution Time : 0.798 ms   Total Expected Cost : 24.08

- Indices created as shown on above screenshot.
- The Search is done with O(1) when we search for values of any column I created index on it so the performance slightly improved with respect to performance of the query with B+ Tree index and the increase in performance is small due to the small number of rows(Total Expected Cost Was 3435.52 without index and 24.94 with B+ tree and became 24.08).

4. given query with BRIN indices only :



```
from movie_cast
where role =any( select role from query_11)));
-------------------Query 12 Original-------------------
explain analyze
select mov_title
from movie
where mov_id in (
select mov_id
from movie_direction
where dir_id=
(select dir_id
from director
where dir_fname='Woddy'
and
dir_lname='Allen'));
-------------------Query 12 Optimized + materialized view ---------
CREATE MATERIALIZED VIEW query_12
AS
 select dir_id from director
 where dir_fname='Woddy' and dir_lname='Allen'
WITH  DATA;



 explain analyze select mov_title
from movie
where mov_id in (select m1.mov_id from movie_direction m1
                 where exists (select * from query_12
                               m2 where m1.dir_id=m2.dir_id ))



CREATE INDEX b_movID ON movie USING brin(mov_id );
CREATE INDEX b_dir_id ON movie_direction USING brin(dir_id );
```

```
QUERY PLAN
text
1  Nested Loop  (cost=10000000273.07..10000002275.21 rows=1 width=51) (actual time=0.951..179.230 rows=387 loops=1)
2  [...] InitPlan 1 (returns $0)
3  [...] ->  Seq Scan on director  (cost=10000000000.00..10000000147.00 rows=1 width=4) (actual time=0.006..0.438 rows=1...
4  [...] Filter: ((dir_fname = 'Woddy'::bpchar) AND (dir_lname = 'Allen'::bpchar))
5  [...] Rows Removed by Filter: 5999
6  [...] ->  HashAggregate  (cost=114.03..114.04 rows=1 width=4) (actual time=0.912..1.012 rows=387 loops=1)
7  [...] Group Key: movie_direction.mov_id
8  [...] Batches: 1  Memory Usage: 85kB
9  [...] ->  Bitmap Heap Scan on movie_direction  (cost=12.03..114.03 rows=1 width=4) (actual time=0.457..0.847 rows=387 l...
10 [...] Recheck Cond: (dir_id = $0)
11 [...] Rows Removed by Index Recheck: 5613
12 [...] Heap Blocks: lossy=27
13 [...] ->  Bitmap Index Scan on b_dir_id  (cost=0.00..12.03 rows=6000 width=0) (actual time=0.451..0.451 rows=270 loops=1)
14 [...] Index Cond: (dir_id = $0)
15 [...] ->  Bitmap Heap Scan on movie  (cost=12.03..2014.16 rows=1 width=55) (actual time=0.220..0.457 rows=1 loops=387)
16 [...] Recheck Cond: (mov_id = movie_direction.mov_id)
17 [...] Rows Removed by Index Recheck: 6655
18 [...] Heap Blocks: lossy=49536
19 [...] ->  Bitmap Index Scan on b_movid  (cost=0.00..12.03 rows=6250 width=0) (actual time=0.008..0.008 rows=1280 loops...
20 [...] Index Cond: (mov_id = movie_direction.mov_id)
21 Planning Time: 0.122 ms
22 Execution Time: 179.504 ms
```

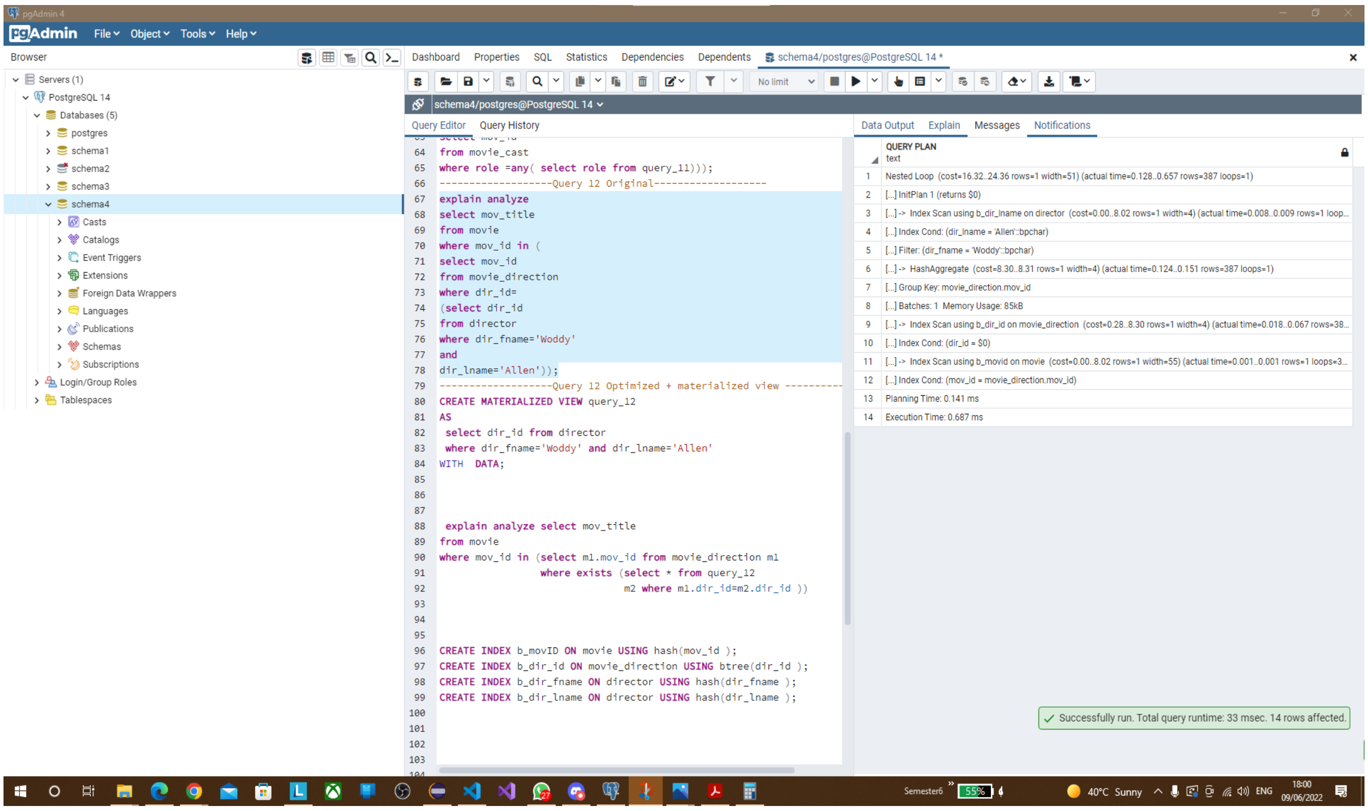
Explanation :

- Metrics :

  Execution Time : 179.504 ms   Total Expected Cost : 10000002275.21

- Indices created as shown on above screenshot.
- Here the BRIN was not used in the original Query Plan settings because using brin in our case here is not efficient as all my query searches is not searching in small range within the range of the column "not low selectivity query" so I set flag seqscan=off .
- The Execution Time and Expected Cost became the Worst of all as a result of turning seqscan flag off.

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



```
from movie_cast
where role =any( select role from query_11)));
-------------------Query 12 Original-------------------
explain analyze
select mov_title
from movie
where mov_id in (
select mov_id
from movie_direction
where dir_id=
(select dir_id
from director
where dir_fname='Woddy'
and
dir_lname='Allen'));
-------------------Query 12 Optimized + materialized view ----------
CREATE MATERIALIZED VIEW query_12
AS
 select dir_id from director
 where dir_fname='Woddy' and dir_lname='Allen'
WITH  DATA;



 explain analyze select mov_title
from movie
where mov_id in (select m1.mov_id from movie_direction m1
                 where exists (select * from query_12
                              m2 where m1.dir_id=m2.dir_id ))



CREATE INDEX b_movID ON movie USING hash(mov_id );
CREATE INDEX b_dir_id ON movie_direction USING btree(dir_id );
CREATE INDEX b_dir_fname ON director USING hash(dir_fname );
CREATE INDEX b_dir_lname ON director USING hash(dir_lname );
```

```
   QUERY PLAN
   text
1  Nested Loop  (cost=16.32..24.36 rows=1 width=51) (actual time=0.128..0.657 rows=387 loops=1)
2  [...] InitPlan 1 (returns $0)
3  [...] ->  Index Scan using b_dir_lname on director  (cost=0.00..8.02 rows=1 width=4) (actual time=0.008..0.009 rows=1 loop...
4  [...] Index Cond: (dir_lname = 'Allen'::bpchar)
5  [...] Filter: (dir_fname = 'Woddy'::bpchar)
6  [...] ->  HashAggregate  (cost=8.30..8.31 rows=1 width=4) (actual time=0.124..0.151 rows=387 loops=1)
7  [...] Group Key: movie_direction.mov_id
8  [...] Batches: 1  Memory Usage: 85kB
9  [...] ->  Index Scan using b_dir_id on movie_direction  (cost=0.28..8.30 rows=1 width=4) (actual time=0.018..0.067 rows=38...
10 [...] Index Cond: (dir_id = $0)
11 [...] ->  Index Scan using b_movid on movie  (cost=0.00..8.02 rows=1 width=55) (actual time=0.001..0.001 rows=1 loops=3...
12 [...] Index Cond: (mov_id = movie_direction.mov_id)
13 Planning Time: 0.141 ms
14 Execution Time: 0.687 ms
```

Successfully run. Total query runtime: 33 msec. 14 rows affected.

Explanation :

- Metrics :

  Execution Time 0.687 ms **Total Expected Cost : 24.36**

- Indices created as shown on above screenshot.
- Now the Performance is better than with B+ index but performance with Hash only is better as any search with values of column dir_id of table movie_direction will be in O(Log n) and with values of column mov_id of table movie or dir_fname,dir_lname of table director will be in O(1).

Optimized Query

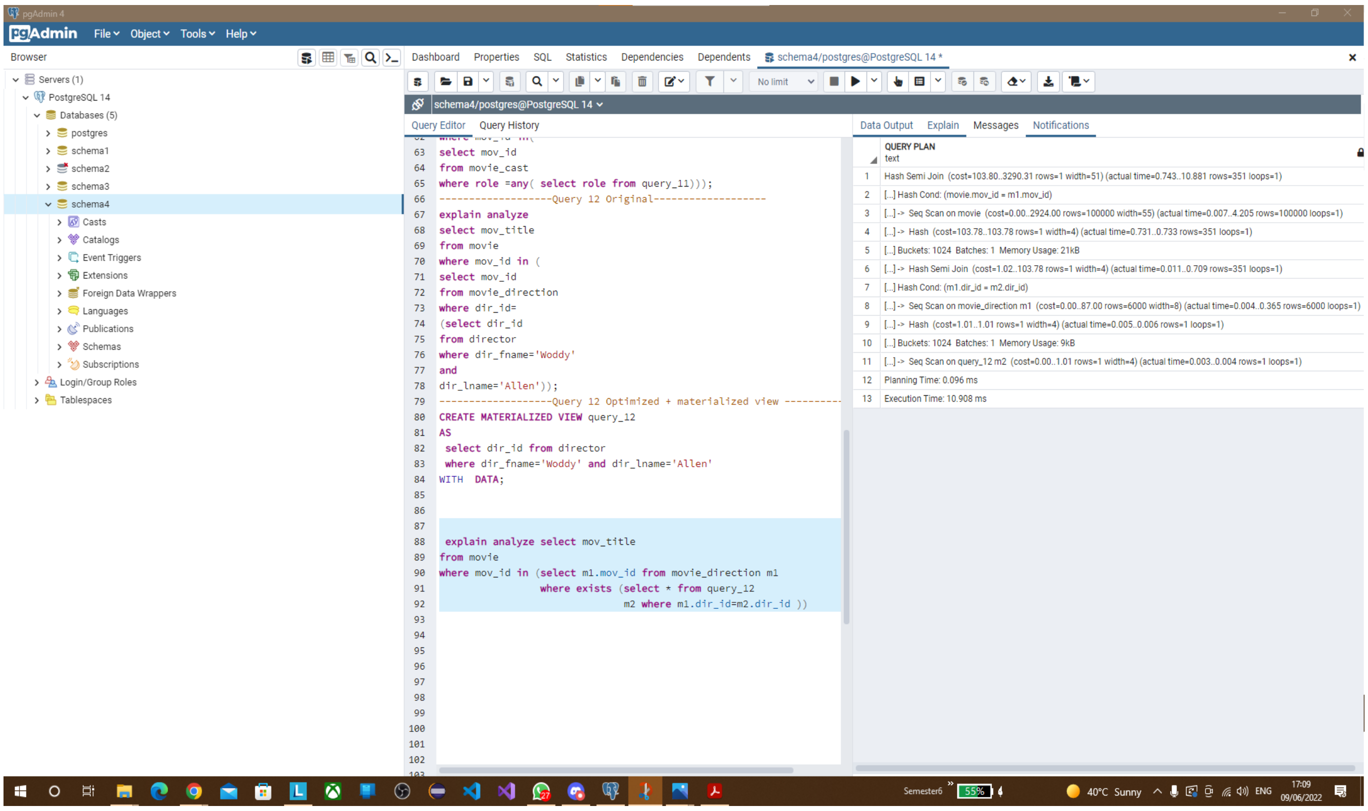
```
CREATE MATERIALIZED VIEW query_12
AS
 select dir_id from director
 where dir_fname='Woddy' and dir_lname='Allen'
WITH  DATA;



 explain analyze select mov_title
from movie
where mov_id in (select m1.mov_id from movie_direction m1
                 where exists (select * from query_12
                              m2 where m1.dir_id=m2.dir_id ))
```

Result Set

- 387 Rows

Report

1. Optimized Query without an index :



```sql
select mov_id
from movie_cast
where role =any( select role from query_11)));
-------------------Query 12 Original-------------------
explain analyze
select mov_title
from movie
where mov_id in (
select mov_id
from movie_direction
where dir_id=
(select dir_id
from director
where dir_fname='Woddy'
and
dir_lname='Allen'));
-------------------Query 12 Optimized + materialized view ---------
CREATE MATERIALIZED VIEW query_12
AS
 select dir_id from director
 where dir_fname='Woddy' and dir_lname='Allen'
WITH  DATA;



 explain analyze select mov_title
from movie
where mov_id in (select m1.mov_id from movie_direction m1
                 where exists (select * from query_12
                               m2 where m1.dir_id=m2.dir_id ))
```

```
QUERY PLAN
text
1  Hash Semi Join  (cost=103.80..3290.31 rows=1 width=51) (actual time=0.743..10.881 rows=351 loops=1)
2  [...] Hash Cond: (movie.mov_id = m1.mov_id)
3  [...] ->  Seq Scan on movie  (cost=0.00..2924.00 rows=100000 width=55) (actual time=0.007..4.205 rows=100000 loops=1)
4  [...] ->  Hash  (cost=103.78..103.78 rows=1 width=4) (actual time=0.731..0.733 rows=351 loops=1)
5  [...] Buckets: 1024  Batches: 1  Memory Usage: 21kB
6  [...] ->  Hash Semi Join  (cost=1.02..103.78 rows=1 width=4) (actual time=0.011..0.709 rows=351 loops=1)
7  [...] Hash Cond: (m1.dir_id = m2.dir_id)
8  [...] ->  Seq Scan on movie_direction m1  (cost=0.00..87.00 rows=6000 width=8) (actual time=0.004..0.365 rows=6000 loops=1)
9  [...] ->  Hash  (cost=1.01..1.01 rows=1 width=4) (actual time=0.005..0.006 rows=1 loops=1)
10 [...] Buckets: 1024  Batches: 1  Memory Usage: 9kB
11 [...] ->  Seq Scan on query_12 m2  (cost=0.00..1.01 rows=1 width=4) (actual time=0.003..0.004 rows=1 loops=1)
12 Planning Time: 0.096 ms
13 Execution Time: 10.908 ms
```

Explanation :

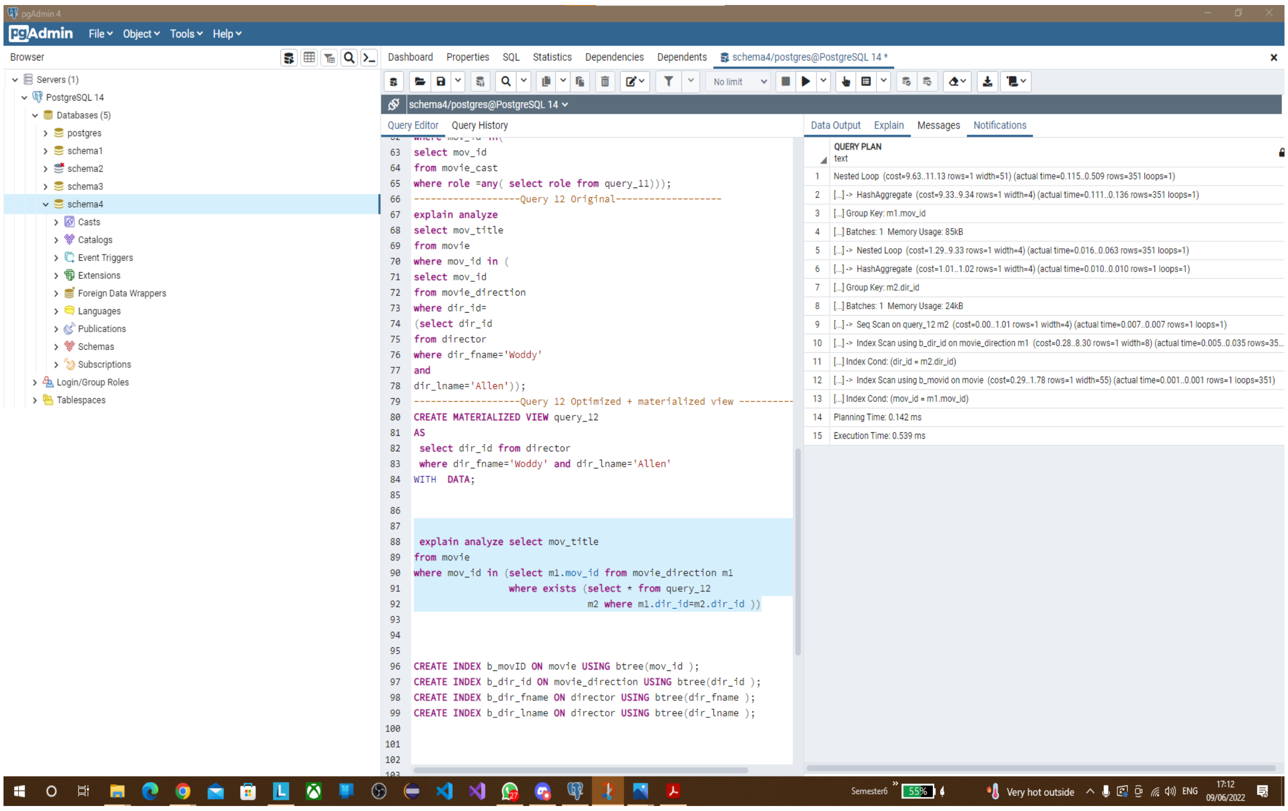
- Metrics :

Execution Time : 10.908 ms	Total Expected Cost : 3290.31

- Reason :

Since selecting dir_id of director whose first name is Woddy and last name is Allen is needed as sub query so I created materialized view storing those values with name of Query 12 and used it inside my query.In addition, used exists instead of = because more than one director can have same first and last name.Finally, The performance is improved with respect to performance of original query after using materialized view Query 12 (Total Expected Cost Was 3435.52 and became 3290.31).

2. Optimized Query with B+ trees indices only :



```sql
where mov_id in(
select mov_id
from movie_cast
where role =any( select role from query_11)));
------------------Query 12 Original------------------
explain analyze
select mov_title
from movie
where mov_id in (
select mov_id
from movie_direction
where dir_id=
(select dir_id
from director
where dir_fname='Woddy'
and
dir_lname='Allen'));
------------------Query 12 Optimized + materialized view ---------
CREATE MATERIALIZED VIEW query_12
AS
 select dir_id from director
 where dir_fname='Woddy' and dir_lname='Allen'
WITH  DATA;



 explain analyze select mov_title
from movie
where mov_id in (select m1.mov_id from movie_direction m1
                where exists (select * from query_12
                              m2 where m1.dir_id=m2.dir_id ))



CREATE INDEX b_movID ON movie USING btree(mov_id );
CREATE INDEX b_dir_id ON movie_direction USING btree(dir_id );
CREATE INDEX b_dir_fname ON director USING btree(dir_fname );
CREATE INDEX b_dir_lname ON director USING btree(dir_lname );
```

```
QUERY PLAN
text
1   Nested Loop  (cost=9.63..11.13 rows=1 width=51) (actual time=0.115..0.509 rows=351 loops=1)
2   [...] ->  HashAggregate  (cost=9.33..9.34 rows=1 width=4) (actual time=0.111..0.136 rows=351 loops=1)
3   [...] Group Key: m1.mov_id
4   [...] Batches: 1  Memory Usage: 85kB
5   [...] ->  Nested Loop  (cost=1.29..9.33 rows=1 width=4) (actual time=0.016..0.063 rows=351 loops=1)
6   [...] ->  HashAggregate  (cost=1.01..1.02 rows=1 width=4) (actual time=0.010..0.010 rows=1 loops=1)
7   [...] Group Key: m2.dir_id
8   [...] Batches: 1  Memory Usage: 24kB
9   [...] ->  Seq Scan on query_12 m2  (cost=0.00..1.01 rows=1 width=4) (actual time=0.007..0.007 rows=1 loops=1)
10  [...] ->  Index Scan using b_dir_id on movie_direction m1  (cost=0.28..8.30 rows=1 width=8) (actual time=0.005..0.035 rows=35...
11  [...] Index Cond: (dir_id = m2.dir_id)
12  [...] ->  Index Scan using b_movid on movie  (cost=0.29..1.78 rows=1 width=55) (actual time=0.001..0.001 rows=1 loops=351)
13  [...] Index Cond: (mov_id = m1.mov_id)
14  Planning Time: 0.142 ms
15  Execution Time: 0.539 ms
```

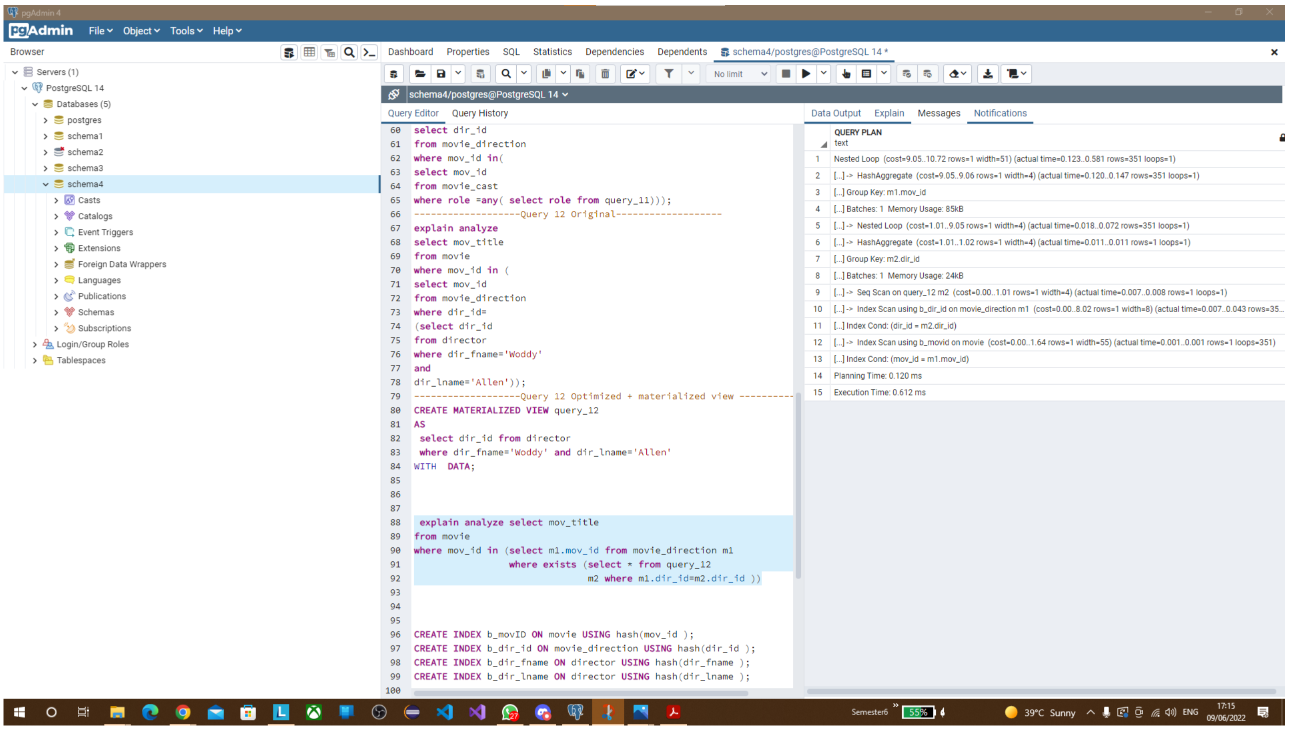
Explanation :

- Metrics :

  Execution Time : 0.539 ms   Total Expected Cost : 11.13

- Indices created as shown on above screenshot.
- The Search is done with O(Log n) using columns index created on, so the performance is improved with respect to performance of the query without index (Total Expected Cost Was 3290.31 and became 11.13).

3. Optimized Query with hash indices only :



```
select dir_id
from movie_direction
where mov_id in(
select mov_id
from movie_cast
where role =any( select role from query_11)));
-------------------Query 12 Original-------------------
explain analyze
select mov_title
from movie
where mov_id in (
select mov_id
from movie_direction
where dir_id=
(select dir_id
from director
where dir_fname='Woddy'
and
dir_lname='Allen'));
-------------------Query 12 Optimized + materialized view ---------
CREATE MATERIALIZED VIEW query_12
AS
 select dir_id from director
 where dir_fname='Woddy' and dir_lname='Allen'
WITH  DATA;



 explain analyze select mov_title
from movie
where mov_id in (select m1.mov_id from movie_direction m1
                 where exists (select * from query_12
                               m2 where m1.dir_id=m2.dir_id ))



CREATE INDEX b_movID ON movie USING hash(mov_id );
CREATE INDEX b_dir_id ON movie_direction USING hash(dir_id );
CREATE INDEX b_dir_fname ON director USING hash(dir_fname );
CREATE INDEX b_dir_lname ON director USING hash(dir_lname );
```

```
QUERY PLAN
text
1   Nested Loop  (cost=9.05..10.72 rows=1 width=51) (actual time=0.123..0.581 rows=351 loops=1)
2   [...] ->  HashAggregate  (cost=9.05..9.06 rows=1 width=4) (actual time=0.120..0.147 rows=351 loops=1)
3   [...] Group Key: m1.mov_id
4   [...] Batches: 1  Memory Usage: 85kB
5   [...] ->  Nested Loop  (cost=1.01..9.05 rows=1 width=4) (actual time=0.018..0.072 rows=351 loops=1)
6   [...] ->  HashAggregate  (cost=1.01..1.02 rows=1 width=4) (actual time=0.011..0.011 rows=1 loops=1)
7   [...] Group Key: m2.dir_id
8   [...] Batches: 1  Memory Usage: 24kB
9   [...] ->  Seq Scan on query_12 m2  (cost=0.00..1.01 rows=1 width=4) (actual time=0.007..0.008 rows=1 loops=1)
10  [...] ->  Index Scan using b_dir_id on movie_direction m1  (cost=0.00..8.02 rows=1 width=8) (actual time=0.007..0.043 rows=35...
11  [...] Index Cond: (dir_id = m2.dir_id)
12  [...] ->  Index Scan using b_movid on movie  (cost=0.00..1.64 rows=1 width=55) (actual time=0.001..0.001 rows=1 loops=351)
13  [...] Index Cond: (mov_id = m1.mov_id)
14  Planning Time: 0.120 ms
15  Execution Time: 0.612 ms
```

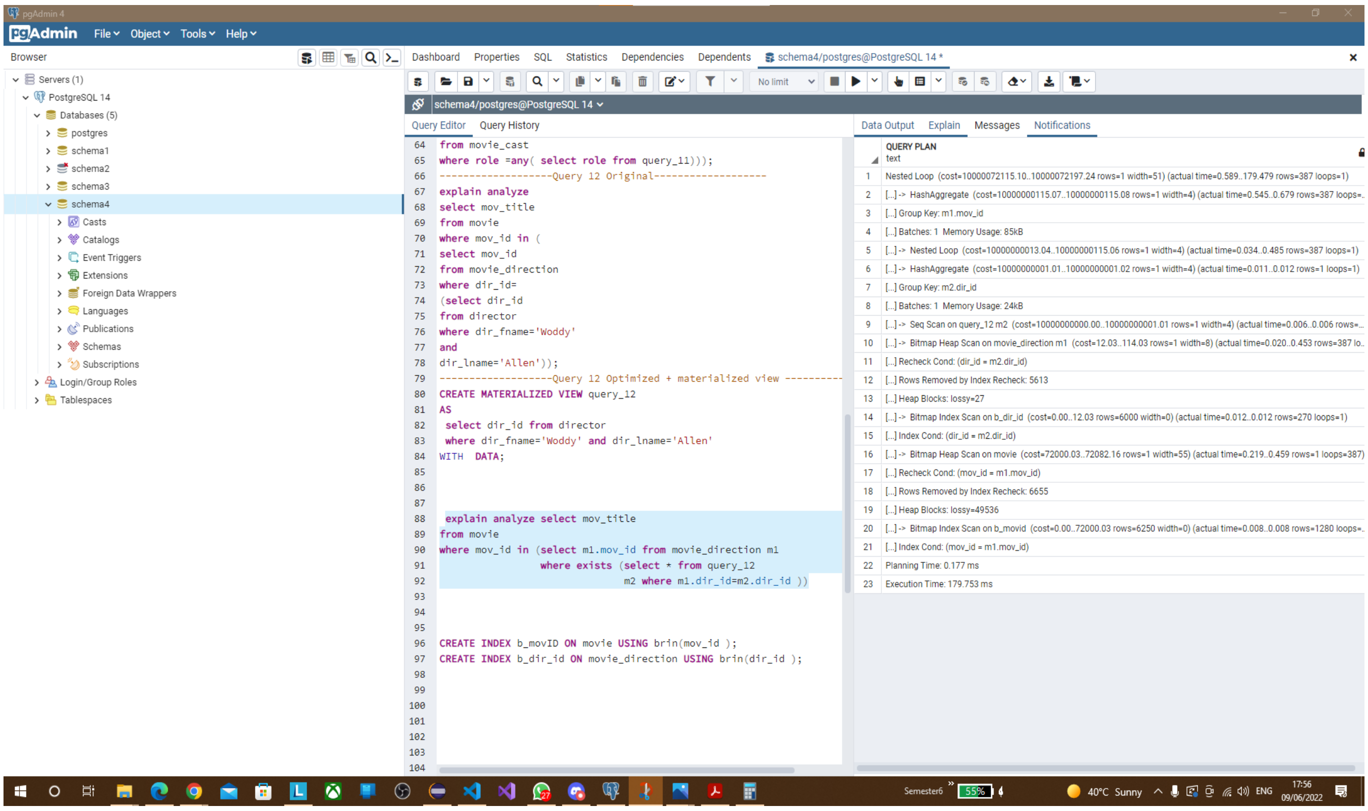
Explanation :

- Metrics :

  Execution Time : 0.612 ms   Total Expected Cost : 10.72

- Indices created as shown on above screenshot.
- The Search is done in O(1) when we search for values of any column I created hash index on it so the performance slightly improved with respect to performance of the query with B+ Tree index and the increase in performance is small due to the small number of rows(Total Expected Cost Was 3290.31 without index and 11.13 with B+ tree and became 10.72).

4. Optimized Query with BRIN indices only :



```sql
from movie_cast
where role =any( select role from query_11)));
-------------------Query 12 Original-------------------
explain analyze
select mov_title
from movie
where mov_id in (
select mov_id
from movie_direction
where dir_id=
(select dir_id
from director
where dir_fname='Woddy'
and
dir_lname='Allen'));
-------------------Query 12 Optimized + materialized view ---------
CREATE MATERIALIZED VIEW query_12
AS
 select dir_id from director
 where dir_fname='Woddy' and dir_lname='Allen'
WITH  DATA;



 explain analyze select mov_title
from movie
where mov_id in (select m1.mov_id from movie_direction m1
                 where exists (select * from query_12
                               m2 where m1.dir_id=m2.dir_id ))



CREATE INDEX b_movID ON movie USING brin(mov_id );
CREATE INDEX b_dir_id ON movie_direction USING brin(dir_id );
```

```
QUERY PLAN
text
1   Nested Loop  (cost=10000072115.10..10000072197.24 rows=1 width=51) (actual time=0.589..179.479 rows=387 loops=1)
2   [...] ->  HashAggregate  (cost=10000000115.07..10000000115.08 rows=1 width=4) (actual time=0.545..0.679 rows=387 loops=..
3   [...] Group Key: m1.mov_id
4   [...] Batches: 1  Memory Usage: 85kB
5   [...] ->  Nested Loop  (cost=10000000013.04..10000000115.06 rows=1 width=4) (actual time=0.034..0.485 rows=387 loops=1)
6   [...] ->  HashAggregate  (cost=10000000001.01..10000000001.02 rows=1 width=4) (actual time=0.011..0.012 rows=1 loops=1)
7   [...] Group Key: m2.dir_id
8   [...] Batches: 1  Memory Usage: 24kB
9   [...] ->  Seq Scan on query_12 m2  (cost=10000000000.00..10000000001.01 rows=1 width=4) (actual time=0.006..0.006 rows=...
10  [...] ->  Bitmap Heap Scan on movie_direction m1  (cost=12.03..114.03 rows=1 width=8) (actual time=0.020..0.453 rows=387 lo..
11  [...] Recheck Cond: (dir_id = m2.dir_id)
12  [...] Rows Removed by Index Recheck: 5613
13  [...] Heap Blocks: lossy=27
14  [...] ->  Bitmap Index Scan on b_dir_id  (cost=0.00..12.03 rows=6000 width=0) (actual time=0.012..0.012 rows=270 loops=1)
15  [...] Index Cond: (dir_id = m2.dir_id)
16  [...] ->  Bitmap Heap Scan on movie  (cost=72000.03..72082.16 rows=1 width=55) (actual time=0.219..0.459 rows=1 loops=387)
17  [...] Recheck Cond: (mov_id = m1.mov_id)
18  [...] Rows Removed by Index Recheck: 6655
19  [...] Heap Blocks: lossy=49536
20  [...] ->  Bitmap Index Scan on b_movid  (cost=0.00..72000.03 rows=6250 width=0) (actual time=0.008..0.008 rows=1280 loops=..
21  [...] Index Cond: (mov_id = m1.mov_id)
22  Planning Time: 0.177 ms
23  Execution Time: 179.753 ms
```

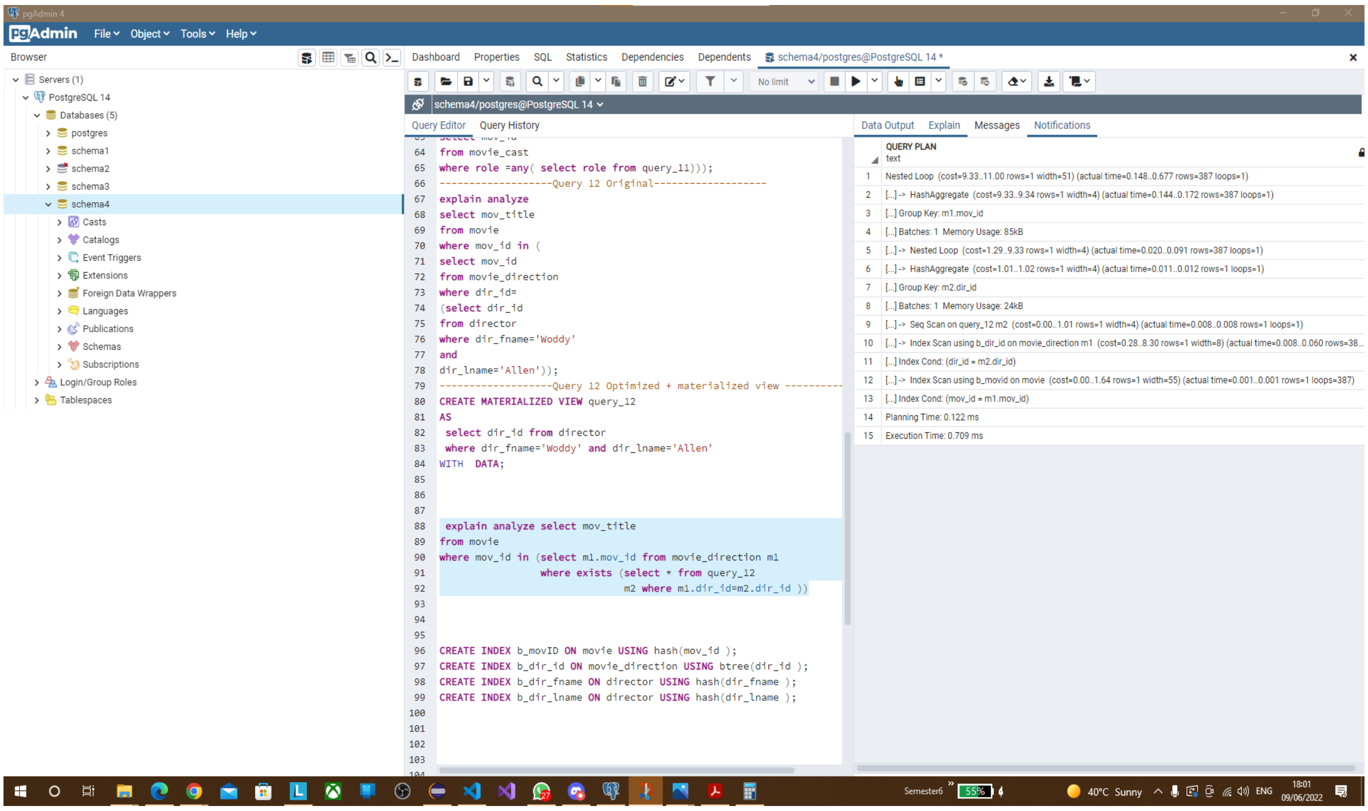
Explanation :

- Metrics :

  Execution Time : 179.753 ms   Total Expected Cost : 100000072197.24

- Indices created as shown on above screenshot.
- Here the BRIN was not used in the original Query Plan settings because using brin in our case here is not efficient as all my query searches is not searching in small range within the range of the column "not low selectivity query" so I set flag seqscan=off.
- The Execution Time and Expected Cost became the Worst of all as a result of turning seqscan flag off.

5. Optimized Query with mixed indices (any mix of your choice) :



```sql
from movie_cast
where role =any( select role from query_11)));
-------------------Query 12 Original-------------------
explain analyze
select mov_title
from movie
where mov_id in (
select mov_id
from movie_direction
where dir_id=
(select dir_id
from director
where dir_fname='Woddy'
and
dir_lname='Allen'));
-------------------Query 12 Optimized + materialized view ----------
CREATE MATERIALIZED VIEW query_12
AS
 select dir_id from director
 where dir_fname='Woddy' and dir_lname='Allen'
WITH  DATA;



 explain analyze select mov_title
from movie
where mov_id in (select m1.mov_id from movie_direction m1
                 where exists (select * from query_12
                               m2 where m1.dir_id=m2.dir_id ))



CREATE INDEX b_movID ON movie USING hash(mov_id );
CREATE INDEX b_dir_id ON movie_direction USING btree(dir_id );
CREATE INDEX b_dir_fname ON director USING hash(dir_fname );
CREATE INDEX b_dir_lname ON director USING hash(dir_lname );
```

```
   QUERY PLAN
   text
1  Nested Loop  (cost=9.33..11.00 rows=1 width=51) (actual time=0.148..0.677 rows=387 loops=1)
2  [...] ->  HashAggregate  (cost=9.33..9.34 rows=1 width=4) (actual time=0.144..0.172 rows=387 loops=1)
3  [...] Group Key: m1.mov_id
4  [...] Batches: 1  Memory Usage: 85kB
5  [...] ->  Nested Loop  (cost=1.29..9.33 rows=1 width=4) (actual time=0.020..0.091 rows=387 loops=1)
6  [...] ->  HashAggregate  (cost=1.01..1.02 rows=1 width=4) (actual time=0.011..0.012 rows=1 loops=1)
7  [...] Group Key: m2.dir_id
8  [...] Batches: 1  Memory Usage: 24kB
9  [...] ->  Seq Scan on query_12 m2  (cost=0.00..1.01 rows=1 width=4) (actual time=0.008..0.008 rows=1 loops=1)
10 [...] ->  Index Scan using b_dir_id on movie_direction m1  (cost=0.28..8.30 rows=1 width=8) (actual time=0.008..0.060 rows=38...
11 [...] Index Cond: (dir_id = m2.dir_id)
12 [...] ->  Index Scan using b_movid on movie  (cost=0.00..1.64 rows=1 width=55) (actual time=0.001..0.001 rows=1 loops=387)
13 [...] Index Cond: (mov_id = m1.mov_id)
14 Planning Time: 0.122 ms
15 Execution Time: 0.709 ms
```

Explanation :

- Metrics :

  Execution Time : 0.709 ms   Total Expected Cost : 11.00

- Indices created as shown on above screenshot.
- Now the Performance is slightly better than with B+ index but performance with Hash only is slightly better as any search with values of column dir_id of table movie_direction will be in O(Log n) and with values of column mov_id of table movie will be in O(1).