

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 :

The screenshot shows the pgAdmin 4 interface. The left pane shows the database structure with 'schema4' selected. The central pane shows the following SQL query:

```

63 select mov_id
64 from movie_cast
65 where role = any( select role from query_11));
66 -----Query 12 Original-----
67 explain analyze
68 select mov_title
69 from movie
70 where mov_id in (
71 select mov_id
72 from movie_direction
73 where dir_id=
74 (select dir_id
75 from director
76 where dir_fname='Woddy'
77 and
78 dir_lname='Allen'));
79 -----Query 12 Optimized + materialized view -----
80 CREATE MATERIALIZED VIEW query_12
81 AS
82 select dir_id from director
83 where dir_fname='Woddy' and dir_lname='Allen'
84 WITH DATA;
85
86
87
88 explain analyze select mov_title
89 from movie
90 where mov_id in (select m1.mov_id from movie_direction m1
91 where exists (select * from query_12
92 m2 where m1.dir_id=m2.dir_id ))
93
94
95
96
97
98
99
100
101
102
103

```

The right pane shows the query plan:

```

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=1)
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=351)
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=351)
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 :

The screenshot shows the pgAdmin 4 interface with the same query as before, but with additional index creation statements at the bottom:

```

96 CREATE INDEX b_mov_id ON movie USING btree(mov_id);
97 CREATE INDEX b_dir_id ON movie_direction USING btree(dir_id);
98 CREATE INDEX b_dir_fname ON director USING btree(dir_fname);
99 CREATE INDEX b_dir_lname ON director USING btree(dir_lname);
100
101
102
103

```

The right pane shows the updated query plan:

```

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_fname on director (cost=0.28..8.30 rows=1 width=4) (actual time=0.014..0.015 rows=1 loops=1)
4   [.] Index Cond: (dir_fname = 'Woddy'::bpchar)
5   [.] Filter: (dir_lname = '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=351)
10    [.] Index Cond: (dir_id = $0)
11    [.] Index Scan using b_mov_id on movie (cost=0.29..8.31 rows=1 width=55) (actual time=0.001..0.001 rows=1 loops=1)
12    [.] Index Cond: (mov_id = movie_direction.mov_id)
13    Planning Time: 0.158 ms
14    Execution Time: 0.543 ms

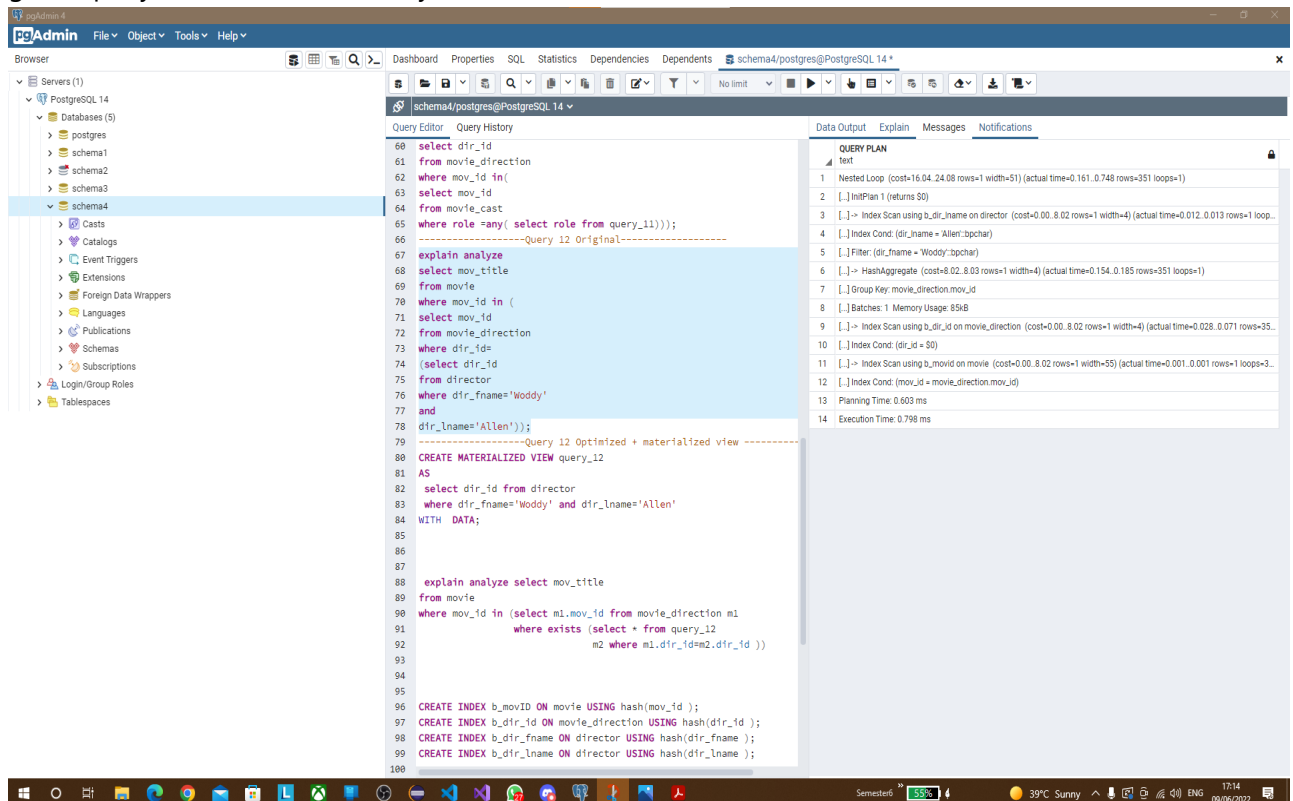
```

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 ($\text{dir_lname} = \text{'Allen'}$) 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 :

**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 :

The screenshot shows the pgAdmin 4 interface with a query editor and a query plan. The query is as follows:

```

64 from movie_cast
65 where role = any( select role from query_11));
66 -----Query 12 Original-----
67 explain analyze
68 select mov_title
69 from movie
70 where mov_id in (
71 select mov_id
72 from movie_direction
73 where dir_id=
74 (select dir_id
75 from director
76 where dir_fname='Woddy'
77 and
78 dir_lname='Allen'));
79 -----Query 12 Optimized + materialized view -----
80 CREATE MATERIALIZED VIEW query_12
81 AS
82 select dir_id from director
83 where dir_fname='Woddy' and dir_lname='Allen'
84 WITH DATA;
85
86
87
88 explain analyze select mov_title
89 from movie
90 where mov_id in (select m1.mov_id from movie_direction m1
91 where exists (select * from query_12
92 m2 where m1.dir_id=m2.dir_id ))
93
94
95
96 CREATE INDEX b_movID ON movie USING brin(mov_id );
97 CREATE INDEX b_dir_id ON movie_direction USING brin(dir_id );
98
99
100
101
102
103
104

```

The query plan on the right shows the execution details:

Step	Operation	Cost	Actual Time	Actual Rows	Loops
1	Nested Loop	(cost=10000002275.07..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..)		
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=1)		
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) :

The screenshot shows the pgAdmin 4 interface with a PostgreSQL query editor. The query editor contains the following SQL code:

```

64 from movie_cast
65 where role = any( select role from query_11));
66
67
68 -----Query 12 Original-----
69
70 explain analyze
71 select mov_title
72 from movie
73 where mov_id in (
74 select mov_id
75 from movie_direction
76 where dir_id =
77 (select dir_id
78 from director
79 where dir_fname='Woddy'
80 and
81 dir_lname='Allen'));
82
83 -----Query 12 Optimized + materialized view -----
84
85 CREATE MATERIALIZED VIEW query_12
86 AS
87 select dir_id from director
88 where dir_fname='Woddy' and dir_lname='Allen'
89 WITH DATA;
90
91
92 explain analyze select mov_title
93 from movie
94 where mov_id in (select m1.mov_id from movie_direction m1
95 where exists (select * from query_12
96 m2 where m1.dir_id=m2.dir_id ))
97
98
99 CREATE INDEX b_movID ON movie USING hash(mov_id );
100 CREATE INDEX b_dir_id ON movie_direction USING btree(dir_id );
101 CREATE INDEX b_dir_fname ON director USING hash(dir_fname );
102 CREATE INDEX b_dir_lname ON director USING hash(dir_lname );
103
104

```

The execution plan on the right shows the following steps:

- 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=1)
- 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=387 loops=1)
- 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=387)
- 12 [.] Index Cond: (mov_id = movie_direction.mov_id)
- 13 Planning Time: 0.141 ms
- 14 Execution Time: 0.687 ms

A green message box at the bottom right states: "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 :

The screenshot shows the pgAdmin 4 interface with a PostgreSQL 14 database. The left sidebar shows the database structure, including schemas and tables. The main window displays a SQL query editor with the following code:

```

62 where mov_id in (
63 select mov_id
64 from movie_cast
65 where role = any( select role from query_11));
66 -----Query 12 Original-----
67
68 explain analyze
69 select mov_title
70 from movie
71 where mov_id in (
72 select mov_id
73 from movie_direction
74 where dir_id=
75 (select dir_id
76 from director
77 where dir_fname='Woddy'
78 and
79 dir_lname='Allen'));
80 -----Query 12 Optimized + materialized view -----
81
82 CREATE MATERIALIZED VIEW query_12
83 AS
84 select dir_id from director
85 where dir_fname='Woddy' and dir_lname='Allen'
86 WITH DATA;
87
88
89 explain analyze select mov_title
90 from movie
91 where mov_id in (select m1.mov_id from movie_direction m1
92 where exists (select * from query_12
93 m2 where m1.dir_id=m2.dir_id ))
94
95
96
97
98
99
100
101
102
103

```

The right pane shows the query plan for the optimized query. The plan includes a Hash Semi Join, Seq Scan on movie, Hash, Seq Scan on movie_direction m1, Hash, Seq Scan on query_12 m2, and Planning Time. The execution time is 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 :

The screenshot shows the pgAdmin 4 interface with a PostgreSQL query editor. The query editor contains the following SQL code:

```

63 select mov_id
64 from movie_cast
65 where role = any( select role from query_11 ));
66 -----Query 12 Original-----
67 explain analyze
68 select mov_title
69 from movie
70 where mov_id in (
71 select mov_id
72 from movie_direction
73 where dir_id=
74 (select dir_id
75 from director
76 where dir_fname='Woddy'
77 and
78 dir_lname='Allen'));
79 -----Query 12 Optimized + materialized view -----
80 CREATE MATERIALIZED VIEW query_12
81 AS
82 select dir_id from director
83 where dir_fname='Woddy' and dir_lname='Allen'
84 WITH DATA;
85
86
87
88 explain analyze select mov_title
89 from movie
90 where mov_id in (select m1.mov_id from movie_direction m1
91 where exists (select * from query_12
92 m2 where m1.dir_id=m2.dir_id ))
93
94
95
96 CREATE INDEX b_movID ON movie USING btree(mov_id );
97 CREATE INDEX b_dir_id ON movie_direction USING btree(dir_id );
98 CREATE INDEX b_dir_fname ON director USING btree(dir_fname );
99 CREATE INDEX b_dir_lname ON director USING btree(dir_lname );
100
101
102
103

```

The execution plan on the right shows the following steps:

- Nested Loop (cost=9.63..11.13 rows=1 width=51) (actual time=0.115..0.509 rows=351 loops=1)
- [.]> HashAggregate (cost=9.33..9.34 rows=1 width=4) (actual time=0.111..0.136 rows=351 loops=1)
- [.] Group Key: m1.mov_id
- [.] Batches: 1 Memory Usage: 85kB
- [.]> Nested Loop (cost=1.29..9.33 rows=1 width=4) (actual time=0.016..0.063 rows=351 loops=1)
- [.]> HashAggregate (cost=1.01..1.02 rows=1 width=4) (actual time=0.010..0.010 rows=1 loops=1)
- [.] Group Key: m2.dir_id
- [.] Batches: 1 Memory Usage: 24kB
- [.]> 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)
- [.]> 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=351 loops=1)
- [.] Index Cond: (dir_id = m2.dir_id)
- [.]> 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)
- [.] Index Cond: (mov_id = m1.mov_id)
- Planning Time: 0.142 ms
- 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 :

The screenshot shows the PgAdmin 4 interface with a PostgreSQL database. The left sidebar shows the database structure, including schemas and tables. The main window displays a SQL query editor with the following code:

```

60 select dir_id
61 from movie_direction
62 where mov_id in(
63 select mov_id
64 from movie_cast
65 where role =any( select role from query_11));
66 -----Query 12 Original-----
67 explain analyze
68 select mov_title
69 from movie
70 where mov_id in (
71 select mov_id
72 from movie_direction
73 where dir_id=
74 (select dir_id
75 from director
76 where dir_fname='Woddy'
77 and
78 dir_lname='Allen'));
79 -----Query 12 Optimized + materialized view -----
80 CREATE MATERIALIZED VIEW query_12
81 AS
82 select dir_id from director
83 where dir_fname='Woddy' and dir_lname='Allen'
84 WITH DATA;
85
86
87
88 explain analyze select mov_title
89 from movie
90 where mov_id in (select m1.mov_id from movie_direction m1
91 where exists (select * from query_12
92 m2 where m1.dir_id=m2.dir_id ))
93
94
95
96 CREATE INDEX b_mov_id ON movie USING hash(mov_id );
97 CREATE INDEX b_dir_id ON movie_direction USING hash(dir_id );
98 CREATE INDEX b_dir_fname ON director USING hash(dir_fname );
99 CREATE INDEX b_dir_lname ON director USING hash(dir_lname );
100

```

The right pane shows the query plan, which includes a nested loop join with hash aggregates, indicating the use of hash indices. The execution time is 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 :

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

- Left Panel (Browser):** Shows the database structure for PostgreSQL 14, including schemas (schema1, schema2, schema3, schema4) and tables (movie, movie_direction, director).
- Query Editor:** Contains the following SQL code:


```

64 from movie_cast
65 where role = any( select role from query_11));
66 -----Query 12 Original-----
67 explain analyze
68 select mov_title
69 from movie
70 where mov_id in (
71 select mov_id
72 from movie_direction
73 where dir_id=
74 (select dir_id
75 from director
76 where dir_fname='Woddy'
77 and
78 dir_lname='Allen'));
79 -----Query 12 Optimized + materialized view -----
80 CREATE MATERIALIZED VIEW query_12
81 AS
82 select dir_id from director
83 where dir_fname='Woddy' and dir_lname='Allen'
84 WITH DATA;
85
86
87
88 explain analyze select mov_title
89 from movie
90 where mov_id in (select m1.mov_id from movie_direction m1
91 where exists (select * from query_12
92 m2 where m1.dir_id=m2.dir_id ))
93
94
95
96 CREATE INDEX b_mov_id ON movie USING brin(mov_id );
97 CREATE INDEX b_dir_id ON movie_direction USING brin(dir_id );
98
99
100
101
102
103
104

```
- Query Plan:** Shows the execution plan for the query, including a nested loop, hash aggregate, and seq scan on the movie_direction table. The plan includes details such as cost, rows, width, and actual time.

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) :

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

- Left Panel (Browser):** Shows the database structure including Servers (1), Databases (5), and Schemas (4). The 'schema4' database is selected.
- Query Editor:** Contains the following SQL code:


```

64 from movie_cast
65 where role = any( select role from query_11));
66
67 -----Query 12 Original-----
68
69 explain analyze
70 select mov_title
71 from movie
72 where mov_id in (
73   select mov_id
74   from movie_direction
75   where dir_id=
76     (select dir_id
77      from director
78      where dir_fname='Woddy'
79      and
80      dir_lname='Allen'));
81 -----Query 12 Optimized + materialized view-----
82
83 CREATE MATERIALIZED VIEW query_12
84 AS
85   select dir_id from director
86   where dir_fname='Woddy' and dir_lname='Allen'
87 WITH DATA;
88
89
90 explain analyze select mov_title
91 from movie
92 where mov_id in (select m1.mov_id from movie_direction m1
93                  where exists (select * from query_12
94                               m2 where m1.dir_id=m2.dir_id ))
95
96
97 CREATE INDEX b_movID ON movie USING hash(mov_id );
98 CREATE INDEX b_dir_id ON movie_direction USING btree(dir_id );
99 CREATE INDEX b_dir_fname ON director USING hash(dir_fname );
100 CREATE INDEX b_dir_lname ON director USING hash(dir_lname );
101
102
103
104

```
- Query Plan:** Shows the execution plan for the query. The plan includes the following steps:
 - 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=387 loops=1)
 - 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 $O(1)$.