

Query 11

- Find the name of the director (first and last names) who directed a movie that casted a role for 'Eyes Wide Shut'. Empty result set not acceptable.

Note !

- all flags are set to default

Original Query

```
select dir_fname, dir_lname
from director
where dir_id in(
select dir_id
from movie_direction
where mov_id in(
select mov_id
from movie_cast
where role =any( select role
from movie_cast
where mov_id in(
select mov_id
from movie
where
mov_title='Eyes Wide Shut'))));
```

Result Set

- 9 Rows

Report

1. given query without an index :

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

```

1  select *
2  explain analyze
3  from actor
4  where act_id in(
5  select act_id
6  from movie_cast
7  where mov_id in(
8  select mov_id
9  from movie
10 where mov_title = 'Annie Hall')));
11
12 create MATERIALIZED VIEW query_10
13 as
14 select mov_id from movie m2 where mov_title = 'Annie Hall'
15
16
17 explain analyze select *
18 from actor
19 where act_id in( select act_id from movie_cast m1 where m1.dir_id =
20 (select dir_id from movie_direction where mov_id in(
21 select mov_id from movie_cast m2 where mov_title = 'Eyes Wide Shut'))));
22
23 explain analyze
24 select dir_fname, dir_lname
25 from director
26 where dir_id in(
27 select dir_id
28 from movie_direction
29 where mov_id in(
30 select mov_id
31 from movie_cast
32 where role = any( select role
33 from movie_cast
34 where mov_id in(
35 select mov_id
36 from movie
37 where mov_title = 'Eyes Wide Shut'))));
38
39
40

```

The execution plan shows a nested loop join with a cost of 5270.54 and an execution time of 113.955 ms. The plan includes details about the join filter, rows removed, and the cost of the subqueries.

Explanation :

- Metrics :

Execution Time : 113.955 ms Total Expected Cost : 7751.05

2. given query with B+ trees indices only :

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

```

20 where act_id in( select act_id from movie_cast m1 where m1.dir_id =
21 (select dir_id from movie_direction where mov_id in(
22 select mov_id from movie_cast m2 where mov_title = 'Eyes Wide Shut'))));
23
24 explain analyze
25 select dir_fname, dir_lname
26 from director
27 where dir_id in(
28 select dir_id
29 from movie_direction
30 where mov_id in(
31 select mov_id
32 from movie_cast
33 where role = any( select role
34 from movie_cast
35 where mov_id in(
36 select mov_id
37 from movie
38 where mov_title = 'Eyes Wide Shut'))));
39
40
41 create MATERIALIZED VIEW query_11
42 as
43 select role
44 from movie_cast m1
45 where exists (select mov_id from movie m2 where mov_title =
46
47
48
49 explain analyze
50 select dir_fname, dir_lname
51 from director
52 where dir_id in(
53 select dir_id
54 from movie_direction
55 where mov_id in(
56 select mov_id
57 from movie_cast
58 where role = any( select role from query_11));
59
60 explain analyze
61
62

```

The execution plan shows a nested loop join with a cost of 3285.90 and an execution time of 9.521 ms. The plan includes details about the join filter, rows removed, and the cost of the subqueries.

Explanation :

- Metrics :

Execution Time : 9.521 ms Total Expected Cost : 3285.99

- Index created on column dir_id of table director, column role of table movie_cast, column mov_id of table movie_cast.
- The Search is done with $O(\log n)$ for search using values of column dir_id of table director, column role of table movie_cast, column mov_id of table movie_cast so the performance is improved with respect to performance of the query without index (Total Expected Cost Was 7751.05 and became 3285.99).

3. given query with hash indices only :

The screenshot displays the PgAdmin interface with a SQL query editor and an execution plan. The query is as follows:

```

20 where act_id in( select act_id from movie_cast m1 where e
21
22 -----Query 11 Original-----
23 explain analyze
24 select dir_fname, dir_lname
25 from director
26 where dir_id in(
27 select dir_id
28 from movie_direction
29 where mov_id in(
30 select mov_id
31 from movie_cast
32 where role =any( select role
33 from movie_cast
34 where mov_id in(
35 select mov_id
36 from movie
37 mov_title='Eyes Wide Shut'))));
38 -----Query 11 Optimized + materialized view
39
40 CREATE MATERIALIZED VIEW query_11
41 AS
42 select role
43 from movie_cast m1
44 where exists (select mov_id from movie m2 where mov_title
45
46
47
48 explain analyze
49 select dir_fname, dir_lname
50 from director
51 where dir_id in(
52 select dir_id
53 from movie_direction
54 where mov_id in(
55 select mov_id
56 from movie_cast
57 where role =any( select role from query_11));
58 -----Query 12 Original-----
59 explain analyze
60

```

The execution plan on the right shows the following steps:

- Nested Loop (cost=3284.90..3285.00 rows=1 width=42) (actual time=9.530..9.553 rows=12 loops=1)
- HashAggregate (cost=3284.90..3284.91 rows=1 width=4) (actual time=9.524..9.527 rows=12 loops=1)
- Group Key: movie_direction.dir_id
- Batches: 1 Memory Usage: 24kB
- Hash Semi Join (cost=3182.15..3284.90 rows=1 width=4) (actual time=9.038..9.521 rows=12 loops=1)
- Hash Cond (movie_direction.mov_id = movie_cast.mov_id)
- Seq Scan on movie_direction (cost=0.00..87.00 rows=6000 width=8) (actual time=0.012..0.226 rows=6000 loops=1)
- Hash (cost=3182.13..3182.13 rows=1 width=4) (actual time=8.971..8.972 rows=200 loops=1)
- Buckets: 1024 Batches: 1 Memory Usage: 16kB
- Nested Loop (cost=3182.04..3182.13 rows=1 width=4) (actual time=8.836..8.952 rows=200 loops=1)
- HashAggregate (cost=3182.04..3182.05 rows=1 width=31) (actual time=8.827..8.828 rows=1 loops=1)
- Group Key: movie_cast_1.role
- Batches: 1 Memory Usage: 24kB
- Nested Loop (cost=3174.00..3182.04 rows=1 width=31) (actual time=8.823..8.825 rows=1 loops=1)
- HashAggregate (cost=3174.00..3174.01 rows=1 width=4) (actual time=8.803..8.804 rows=1 loops=1)
- Group Key: movie_mov_id
- Batches: 1 Memory Usage: 24kB
- Seq Scan on movie (cost=0.00..3174.00 rows=1 width=4) (actual time=0.081..8.796 rows=1 loops=1)
- Filter (mov_title = 'Eyes Wide Shut'::bpchar)
- Rows Removed by Filter: 99999
- Index Scan using b_movid on movie_cast movie_cast_1 (cost=0.00..8.02 rows=1 width=35) (actual time=0.017..0.018 rows=1 loops=1)
- Index Cond (mov_id = movie_mov_id)
- Index Scan using b_role on movie_cast (cost=0.00..0.07 rows=1 width=35) (actual time=0.008..0.115 rows=200 loops=1)
- Index Cond (role = movie_cast_1.role)
- Index Scan using b_dir_id on director (cost=0.00..0.08 rows=1 width=46) (actual time=0.002..0.002 rows=1 loops=12)
- Index Cond (dir_id = movie_direction.dir_id)
- Planning Time: 0.584 ms
- Execution Time: 9.636 ms

Explanation :

- Metrics :

Execution Time : 9.636 ms Total Expected Cost : 3285

- Index created on column dir_id of table director, column role of table movie_cast, column mov_id of table movie_cast.
- 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 7907.55 without index and 3182.85 with B+ tree and became 3285).

4. given query with BRIN indices only :

The top screenshot shows the PgAdmin 4 interface with the following SQL query in the Query Editor:

```

1 -----Query 10 Original-----
2 explain analyze
3 select *
4 from actor
5 where act_id in(
6 select act_id
7 from movie_cast
8 where mov_id in(
9 select mov_id
10 from movie
11 where mov_title = 'Annie Hall'));
12 -----Query 10 Optimized + materialized
13 create MATERIALIZED VIEW query_10
14 as
15 select mov_id from movie m2 where mov_title = 'Annie Hall';
16
17
18 explain analyze select *
19 from actor
20 where act_id in( select act_id from movie_cast m1 where
21 mov_id in(
22 -----Query 11 Original-----
23 select dir_fname, dir_lname
24 from director
25 where dir_id in(
26 select dir_id
27 from movie_direction
28 where mov_id in(
29 select mov_id
30 from movie_cast
31 where role = any( select role
32 from movie_cast
33 where mov_id in(
34 select mov_id
35 from movie
36 where
37 mov_title='Eyes Wide Shut')));
38 -----Query 11 Optimized + materialized
39
40 CREATE MATERIALIZED VIEW query_11
41 AS
42 select role
43 from movie_cast m1
44 where exists (select mov_id from movie m2 where mov_title = 'Eyes Wide Shut');
45
46

```

The bottom screenshot shows the PgAdmin 4 interface with the following SQL query in the Query Editor:

```

1 -----Query 11 Original-----
2 explain analyze
3 select *
4 from actor
5 where act_id in(
6 select act_id
7 from movie_cast
8 where mov_id in(
9 select mov_id
10 from movie
11 where mov_title = 'Annie Hall'));
12 -----Query 10 Optimized + materialized
13 create MATERIALIZED VIEW query_10
14 as
15 select mov_id from movie m2 where mov_title = 'Annie Hall';
16
17
18 explain analyze select *
19 from actor
20 where act_id in( select act_id from movie_cast m1 where
21 mov_id in(
22 -----Query 11 Original-----
23 select dir_fname, dir_lname
24 from director
25 where dir_id in(
26 select dir_id
27 from movie_direction
28 where mov_id in(
29 select mov_id
30 from movie_cast
31 where role = any( select role
32 from movie_cast
33 where mov_id in(
34 select mov_id
35 from movie
36 where
37 mov_title='Eyes Wide Shut')));
38 -----Query 11 Optimized + materialized
39
40 CREATE MATERIALIZED VIEW query_11
41 AS
42 select role
43 from movie_cast m1
44 where exists (select mov_id from movie m2 where mov_title = 'Eyes Wide Shut');
45
46

```

The execution plans for both queries are shown on the right side of the interface. The top execution plan for 'query_10' shows a Nested Loop join with a HashAggregate and a Seq Scan on movie_direction. The bottom execution plan for 'query_11' shows a Nested Loop join with a HashAggregate and a Seq Scan on movie_direction.

Explanation :

- Metrics :

Execution Time : 27.554 ms Total Expected Cost : 20001264616.30

- Index created on column dir_id of table director, column role of table movie_cast, column mov_id of table movie_cast.

- 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:

```

--Query 10 Optimized + materialized view
create MATERIALIZED VIEW query_10
as
select mov_id from movie m2 where mov_title = 'Annie Hall'

--Query 11 Original
explain analyze select *
from actor
where act_id in( select act_id from movie_cast m1 where mov_id = m2.mov_id )

--Query 11 Optimized + materialized view
explain analyze
select dir_fname, dir_lname
from director
where dir_id in(
select dir_id
from movie_direction
where mov_id in(
select mov_id
from movie_cast
where role = any( select role
from movie_cast
where mov_id in(
select mov_id
from movie
where
mov_title='Eyes Wide Shut'
)))));

CREATE INDEX b_dir_id ON director USING hash(dir_id);
CREATE INDEX b_role ON movie_cast USING btree(role);
CREATE INDEX b_movID ON movie_cast USING btree(mov_id);

--Query 11 Optimized + materialized view
AS
select role
from movie_cast m1
where exists (select mov_id from movie m2
where mov_title='Eyes Wide Shut' and m1.mov_id=m2.mov_id)

```

The execution plan shows a nested loop join with various index scans and aggregates. The execution time is 8.907 ms and the total expected cost is 3285.72.

Explanation :

- Metrics :

Execution Time 8.907 ms Total Expected Cost : 3285.72

- As shown in screenshot Index created on column dir_id of table director(hash), column role of table movie_cast(btree), column mov_id of table movie_cast(btree).
- 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 director will be in O(1) and with values of column mov_id of table movie_cast or column role of table movie_cast will be in O(Log n).

Optimized Query

```

CREATE MATERIALIZED VIEW query_11
AS
select role
from movie_cast m1
where exists (select mov_id from movie m2
where mov_title='Eyes Wide Shut' and m1.mov_id=m2.mov_id)

```

```
explain analyze
select dir_fname, dir_lname
from director
where dir_id in(
select dir_id
from movie_direction
where mov_id in(
select mov_id
from movie_cast
where role =any( select role from query 11)));
```

Result Set

- 9 Rows

Report

1. Optimized Query without an index :

Browsers

- 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

Dashboard Properties SQL Statistics Dependencies Dependents schema4/postgres@PostgreSQL 14

Query Editor Query History

```

29 select mov_id
30 from movie_cast
31 where role =any( select role
32 from movie_cast
33 where mov_id in(
34 select mov_id
35 from movie
36 where
37 mov_title='Eyes Wide Shut')));
38 -----Query 11 Optimized + materialized view
39
40 CREATE MATERIALIZED VIEW query_11
41 AS
42 select role
43 from movie_cast m1
44 where exists (select mov_id from movie m2 where mov_title
45
46
47
48 explain analyze
49 select dir_fname, dir_lname
50 from director
51 where dir_id in(
52 select dir_id
53 from movie_direction
54 where mov_id in(
55 select mov_id
56 from movie_cast
57 where role =any( select role from query_11)));
58 -----Query 12 Original-----
59 explain analyze
60 select mov_title
61 from movie
62 where mov_id in (
63 select mov_id
64 from movie_direction
65 where dir_id=
66 (select dir_id
67 from director
68 where dir_fname='Woddy'
69

```

Data Output Explain Messages Notifications

QUERY PLAN

- Hash Semi Join (cost=2238.56..2372.03 rows=32 width=42) (actual time=12.606..13.078 rows=12 loops=1)
- [.] Hash Cond: (director.dir_id = movie.direction_dir_id)
- [.] Seq Scan on director (cost=0.00..117.00 rows=6000 width=46) (actual time=0.008..0.232 rows=6000 loops=1)
- [.] Hash (cost=2238.16..2238.16 rows=32 width=4) (actual time=12.547..12.549 rows=12 loops=1)
- [.] Buckets: 1024 Batches: 1 Memory Usage: 9kB
- [.] Hash Semi Join (cost=2190.22..2238.16 rows=32 width=4) (actual time=12.018..12.544 rows=12 loops=1)
- [.] Hash Cond: (movie.direction_mov_id = movie.cast.mov_id)
- [.] Seq Scan on movie_cast (cost=0.00..87.00 rows=6000 width=8) (actual time=0.004..0.231 rows=6000 loops=1)
- [.] Hash (cost=2123.55..2123.55 rows=533 width=4) (actual time=11.960..11.961 rows=200 loops=1)
- [.] Buckets: 1024 Batches: 1 Memory Usage: 16kB
- [.] Hash Join (cost=2113.21..2123.55 rows=533 width=4) (actual time=0.081..11.925 rows=200 loops=1)
- [.] Hash Cond: (movie.cast.role = query_11.role)
- [.] Seq Scan on movie_cast (cost=0.00..1834.00 rows=100000 width=35) (actual time=0.005..4.034 rows=100000 loops=1)
- [.] Hash (cost=18.63..18.63 rows=200 width=124) (actual time=0.010..0.010 rows=1 loops=1)
- [.] Buckets: 1024 Batches: 1 Memory Usage: 9kB
- [.] HashAggregate (cost=16.63..18.63 rows=200 width=124) (actual time=0.008..0.008 rows=1 loops=1)
- [.] Group Key: query_11.role
- [.] Batches: 1 Memory Usage: 40kB
- [.] Seq Scan on query_11 (cost=0.00..15.30 rows=530 width=124) (actual time=0.004..0.004 rows=1 loops=1)
- Planning Time: 0.136 ms
- Execution Time: 13.139 ms

Explanation :

- Metrics :

Execution Time : 13.139 ms Total Expected Cost : 2372.03

- Reason : Since selecting roles of movie Eyes wide shut is needed as sub query so I created materialized view storing those values with name of the view Query 11 and used it inside my query. So, the performance is improved with respect to performance of original query (Total Expected Cost Was 7751.05 and became 2372.03).

2. Optimized Query with B+ trees indices only :

The screenshot shows the PgAdmin interface with the following content:

Query Editor:

```

34 select mov_id
35 from movie
36 where
37 mov_title='Eyes Wide Shut');
38 -----Query 11 Optimized + materialized view
39
40 CREATE MATERIALIZED VIEW query_11
41 AS
42 select role
43 from movie_cast m1
44 where exists (select mov_id from movie m2 where mov_title
45
46
47
48 explain analyze
49 select dir_fname, dir_lname
50 from director
51 where dir_id in(
52 select dir_id
53 from movie_direction
54 where mov_id in(
55 select mov_id
56 from movie_cast
57 where role =any( select role from query_11));
58 -----Query 12 Original-----
59 explain analyze
60 select mov_title
61 from movie
62 where mov_id in (
63 select mov_id
64 from movie_direction
65 where dir_id=
66 (select dir_id
67 from director
68 where dir_fname='Woody'
69 and
70 dir_lname='Allen'));
71 -----Query 12 Optimized + materialized view
72 CREATE MATERIALIZED VIEW query_12
73 AS
74

```

QUERY PLAN:

```

1 Nested Loop (cost=1626.60..1638.28 rows=32 width=42) (actual time=0.770..0.793 rows=12 loops=1)
2  [...] HashAggregate (cost=1626.31..1626.63 rows=32 width=4) (actual time=0.764..0.766 rows=12 loops=1)
3  [...] Group Key: movie_direction.dir_id
4  [...] Batches: 1 Memory Usage: 24kB
5  [...] Hash Semi Join (cost=1518.29..1626.23 rows=32 width=4) (actual time=0.198..0.762 rows=12 loops=1)
6  [...] Hash Cond (movie_direction.mov_id = movie_cast.mov_id)
7  [...] Seq Scan on movie_direction (cost=0.00..87.00 rows=6000 width=8) (actual time=0.007..0.231 rows=6000 loops=1)
8  [...] Hash (cost=1511.62..1511.62 rows=533 width=4) (actual time=0.138..0.139 rows=200 loops=1)
9  [...] Buckets: 1024 Batches: 1 Memory Usage: 16kB
10  [...] Nested Loop (cost=17.04..1511.62 rows=533 width=4) (actual time=0.024..0.121 rows=200 loops=1)
11  [...] HashAggregate (cost=16.63..18.63 rows=200 width=124) (actual time=0.007..0.007 rows=1 loops=1)
12  [...] Group Key: query_11.role
13  [...] Batches: 1 Memory Usage: 40kB
14  [...] Seq Scan on query_11 (cost=0.00..15.30 rows=530 width=124) (actual time=0.004..0.004 rows=1 loops=1)
15  [...] Index Scan using b_role on movie_cast (cost=0.42..7.46 rows=1 width=35) (actual time=0.015..0.102 rows=200 loops=1)
16  [...] Index Cond: (role = query_11.role)
17  [...] Index Scan using b_dir_id on director (cost=0.28..0.35 rows=1 width=46) (actual time=0.002..0.002 rows=1 loops=12)
18  [...] Index Cond: (dir_id = movie_direction.dir_id)
19 Planning Time: 0.210 ms
20 Execution Time: 0.827 ms

```

Explanation :

- Metrics :

Execution Time : 0.827 ms Total Expected Cost : 1638.28

- Index created on column dir_id of table director, column role of table movie_cast, column mov_id of table movie_cast.
- The Search is done with $O(\log n)$ for search using values of column dir_id of table director, column role of table movie_cast, column mov_id of table movie_cast so the performance is improved with respect to performance of the query without index (Total Expected Cost Was 2372.03 and became 1638.28).

3. Optimized Query with hash indices only :

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

```

39
40 CREATE MATERIALIZED VIEW query_11
41 AS
42 select role
43 from movie_cast m1
44 where exists (select mov_id from movie m2 where mov_title = m1.role)
45
46
47
48 explain analyze
49 select dir_fname, dir_lname
50 from director
51 where dir_id in (
52 select dir_id
53 from movie_direction
54 where mov_id in (
55 select mov_id
56 from movie_cast
57 where role = any(select role from query_11)))
58 -----Query 12 Original-----
59 explain analyze
60 select mov_title
61 from movie
62 where mov_id in (
63 select mov_id
64 from movie_direction
65 where dir_id =
66 (select dir_id
67 from director
68 where dir_fname='Woody'
69 and
70 dir_lname='Allen')));
71 -----Query 12 Optimized + materialized view
72 CREATE MATERIALIZED VIEW query_12
73 AS
74 select dir_id from director where dir_fname='Woody' and
75 WITH DATA;
76
77
78
79

```

The query plan window shows the following execution plan:

```

QUERY PLAN
1  Nested Loop  (cost=1526.81..1530.08 rows=32 width=42) (actual time=0.735..0.762 rows=12 loops=1)
2    [.]> HashAggregate  (cost=1526.81..1527.13 rows=32 width=4) (actual time=0.730..0.732 rows=12 loops=1)
3      [.] Group Key: movie_direction.dir_id
4      [.] Batches: 1  Memory Usage: 24kB
5      [.]> Hash Semi Join  (cost=1418.79..1526.73 rows=32 width=4) (actual time=0.237..0.728 rows=12 loops=1)
6        [.]> Hash Cond  (movie_direction.mov_id = movie_cast.mov_id)
7          [.]> Seq Scan on movie_direction  (cost=0.00..87.00 rows=6000 width=8) (actual time=0.012..0.231 rows=6000 loops=1)
8          [.]> Hash  (cost=1412.13..1412.13 rows=533 width=4) (actual time=0.173..0.174 rows=200 loops=1)
9            Buckets: 1024 Batches: 1  Memory Usage: 16kB
10           [.]> Nested Loop  (cost=16.63..1412.13 rows=533 width=4) (actual time=0.018..0.156 rows=200 loops=1)
11             [.]> HashAggregate  (cost=16.63..18.63 rows=200 width=124) (actual time=0.008..0.009 rows=1 loops=1)
12               [.] Group Key: query_11.role
13               [.] Batches: 1  Memory Usage: 40kB
14               [.]> Seq Scan on query_11  (cost=0.00..15.30 rows=530 width=124) (actual time=0.004..0.005 rows=1 loops=1)
15             [.]> Index Scan using b_role on movie_cast  (cost=0.00..6.96 rows=1 width=35) (actual time=0.009..0.137 rows=200 loops=1)
16               Index Cond: (role = query_11.role)
17             [.]> Index Scan using b_dir_id on director  (cost=0.00..0.08 rows=1 width=46) (actual time=0.002..0.002 rows=1 loops=12)
18               Index Cond: (dir_id = movie_direction.dir_id)
19 Planning Time: 0.177 ms
20 Execution Time: 0.837 ms

```

Explanation :

- Metrics :

Execution Time : 0.837 ms Total Expected Cost : 1530.08

- Index created on column dir_id of table director, column role of table movie_cast, column mov_id of table movie_cast.
- The Search is done with 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 2372.03 without index and 1638.28 with B+ tree and became 1530.08).

4. Optimized Query with BRIN indices only :

The screenshot shows the pgAdmin 4 interface with a PostgreSQL 14 database. The left sidebar shows the server structure, including databases (postgres, schema1, schema2, schema3, schema4) and schema4's contents (casts, catalogs, event triggers, extensions, foreign data wrappers, languages, publications, schemas, subscriptions, login/group roles, tablespaces). The main window displays a SQL query editor with the following SQL code:

```

39
40 CREATE MATERIALIZED VIEW query_11
41 AS
42 select role
43 from movie_cast m1
44 where exists (select mov_id from movie m2
45 where mov_title='Eyes Wide Shut' and m1.mov_id=m2.mov_id)
46
47
48 explain analyze
49 select dir_fname, dir_lname
50 from director
51 where dir_id in(
52 select dir_id
53 from movie_direction
54 where mov_id in(
55 select mov_id
56 from movie_cast
57 where role = any( select role from query_11));
58 -----Query 12 Original-----
59
60 explain analyze
61 select mov_title
62 from movie
63 where mov_id in (
64 select mov_id
65 from movie_direction
66 where dir_id=
67 (select dir_id
68 from director
69 where dir_fname='Woody'
70 and
71 dir_lname='Allen'));
72 -----Query 12 Optimized + materialized view -
73 CREATE MATERIALIZED VIEW query_12
74 AS
75 select dir_id from director where dir_fname='Woody' and di
76 WITH DATA;
77
78
79

```

The right pane shows the query plan for the original query (Query 12 Original). The plan includes a nested loop join, hash aggregate, and various scans. The execution time is 15.497 ms and the total expected cost is 20002699998.13.

Explanation :

- Metrics :

Execution Time : 15.497 ms Total Expected Cost : 20002699998.13

- Index created on column `dir_id` of table `director`, column `role` of table `movie_cast`, column `mov_id` of table `movie_cast`.
 - 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 a query editor and a query plan. The query is as follows:

```

31 where role =any( select role
32 from movie_cast
33 where mov_id in(
34 select mov_id
35 from movie
36 where
37 mov_title='Eyes Wide Shut')));
38
39 CREATE INDEX b_dir_id ON director USING hash(dir_id );
40 CREATE INDEX b_role ON movie_cast USING btree(role );
41 CREATE INDEX b_movID ON movie_cast USING btree(mov_id );
42 -----Query 11 Optimized + materialized view -----
43
44 CREATE MATERIALIZED VIEW query_11
45 AS
46 select role
47 from movie_cast m1
48 where exists (select mov_id from movie m2
49 where mov_title='Eyes Wide Shut' and m1.mov_id=m2.mov_id)
50
51
52
53 explain analyze
54 select dir_fname, dir_lname
55 from director
56 where dir_id in(
57 select dir_id
58 from movie_direction
59 where mov_id in(
60 select mov_id
61 from movie_cast
62 where role =any( select role from query_11));
63 -----Query 12 Original-----
64 explain analyze
65 select mov_title
66 from movie
67 where mov_id in (
68 select mov_id
69 from movie_direction
70 where dir_id=
71

```

The query plan on the right shows the following steps:

- Nested Loop (cost=1626.31..1629.58 rows=32 width=42) (actual time=0.702..0.725 rows=12 loops=1)
- [.]> HashAggregate (cost=1626.31..1626.63 rows=32 width=4) (actual time=0.698..0.700 rows=12 loops=1)
- [.] Group Key: movie_direction.dir_id
- [.] Batches: 1 Memory Usage: 24kB
- [.]> Hash Semi Join (cost=1518.29..1626.23 rows=32 width=4) (actual time=0.208..0.696 rows=12 loops=1)
- [.] Hash Cond: (movie_direction.mov_id = movie_cast.mov_id)
- [.]> Seq Scan on movie_direction (cost=0.00..87.00 rows=6000 width=8) (actual time=0.007..0.224 rows=5000 loops=1)
- [.]> Hash (cost=1511.62..1511.62 rows=533 width=4) (actual time=0.147..0.148 rows=200 loops=1)
- [.] Buckets: 1024 Batches: 1 Memory Usage: 16kB
- [.]> Nested Loop (cost=17.04..1511.62 rows=533 width=4) (actual time=0.022..0.130 rows=200 loops=1)
- [.]> HashAggregate (cost=16.63..18.63 rows=200 width=124) (actual time=0.008..0.008 rows=1 loops=1)
- [.] Group Key: query_11.role
- [.] Batches: 1 Memory Usage: 40kB
- [.]> Seq Scan on query_11 (cost=0.00..15.30 rows=530 width=124) (actual time=0.004..0.004 rows=1 loops=1)
- [.]> Index Scan using b_role on movie_cast (cost=0.42..7.46 rows=1 width=35) (actual time=0.013..0.111 rows=200 loops=1)
- [.] Index Cond: (role = query_11.role)
- [.]> Index Scan using b_dir_id on director (cost=0.00..0.08 rows=1 width=46) (actual time=0.002..0.002 rows=1 loops=12)
- [.] Index Cond: (dir_id = movie_direction.dir_id)
- Planning Time: 0.204 ms
- Execution Time: 0.758 ms

Explanation :

- Metrics :

Execution Time : 0.758 ms Total Expected Cost : 1629.58

- As shown in screenshot Index created on column dir_id of table director(hash), column role of table movie_cast(btree), column mov_id of table movie_cast(btree).
- 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 director will be in $O(1)$ and with values of column role of table movie_cast will be in $O(\log n)$.