

Query 10

- List all the information of the actors who played a role in the movie 'Annie Hall'.

Note !

- all flags are set to default

Original Query

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

Result Set

- 222 Rows

Report

- given 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 query editor with the following SQL code:

```
1
2
3 -----Query 10 Original-----
4 explain analyze
5 select *
6 from actor
7 where act_id in(
8 select act_id
9 from movie_cast
10 where mov_id in(
11 select mov_id
12 from movie
13 where mov_title = 'Annie Hall'));
14 -----Query 10 Optimized + materialized view -----
15 create MATERIALIZED VIEW query_10
16 as
17 select mov_id from movie m2 where mov_title = 'Annie Hall' WITH
18
19
20 explain analyze select *
21 from actor
22 where act_id in( select act_id from movie_cast m1 where exists
23 -----Query 11 Original-----
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
39 mov_title='Eyes Wide Shut')));
40
```

The right pane shows the query plan for the first query. It includes a table with columns: QUERY PLAN, text, and Execution Time. The plan shows a Hash Semi Join operation with a cost of 5270.54 and an actual time of 18.315 ms. The plan also shows a Seq Scan on actor with a cost of 0.00 and an actual time of 0.008 ms. The plan also shows a Hash operation with a cost of 5270.52 and an actual time of 18.295 ms. The plan also shows a Hash Semi Join operation with a cost of 3174.01 and an actual time of 8.939 ms. The plan also shows a Hash operation with a cost of 3174.00 and an actual time of 8.925 ms. The plan also shows a Seq Scan on movie with a cost of 0.00 and an actual time of 0.004 ms. The plan also shows a Filter operation with a cost of 0.00 and an actual time of 0.005 ms. The plan also shows a Rows Removed by Filter operation with a cost of 0.00 and an actual time of 0.000 ms. The plan also shows a Planning Time of 0.164 ms and an Execution Time of 29.388 ms.

Explanation :

- Metrics :

Execution Time : 29.388 ms Total Expected Cost : 7907.55

2. given query with B+ trees indices only :

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 query editor with the following SQL code:

```

1
2
3
4 -----Query 10 Original-----
5
6 explain analyze
7 select *
8 from actor
9 where act_id in(
10 select act_id
11 from movie_cast
12 where mov_id in(
13 select mov_id
14 from movie
15 where mov_title = 'Annie Hall'));
16
17 -----Query 10 Optimized + materialized view -----
18
19 create MATERIALIZED VIEW query_10
20 as
21 select mov_id from movie m2 where mov_title = 'Annie Hall' WITH
22
23
24 explain analyze select *
25 from actor
26 where act_id in( select act_id from movie_cast m1 where exists
27
28 -----Query 11 Original-----
29
30 explain analyze
31 select dir_fname, dir_lname
32 from director
33 where dir_id in(
34 select dir_id
35 from movie_direction
36 where mov_id in(
37 select mov_id
38 from movie_cast
39 where role = any( select role
40 from movie_cast
41 where mov_id in(
42 select mov_id
43 from movie
44 where
45 mov_title='Eyes Wide Shut'))));
46
47
48

```

The right pane shows the query plan for the first query (Query 10). The plan is as follows:

Step	Operation	Cost	Rows	Width	Actual Time	Actual Rows	Actual Loops
1	Nested Loop	(cost=3182.75..3182.85 rows=1 width=48)	1	48	9.358	9,611	222
2	HashAggregate	(cost=3182.34..3182.35 rows=1 width=4)	1	4	9.352	9,367	222
3	Group Key	movie_cast act_id					
4	Batches	1 Memory Usage: 56kB					
5	Nested Loop	(cost=3174.30..3182.33 rows=1 width=4)	1	4	9.288	9,319	222
6	HashAggregate	(cost=3174.00..3174.01 rows=1 width=4)	1	4	9.273	9,274	222
7	Group Key	movie mov_id					
8	Batches	1 Memory Usage: 24kB					
9	Seq Scan on movie	(cost=0.00..3174.00 rows=1 width=4)	1	4	0.010	9,266	222
10	Filter	(mov_title = 'Annie Hall'::bpchar)					
11	Rows Removed by Filter	99999					
12	Index Scan using b_movid on movie_cast	(cost=0.29..8.31 rows=1 width=8)	1	8	0.032	222	222
13	Index Cond	(mov_id = movie_mov_id)					
14	Index Scan using b_actid on actor	(cost=0.42..0.49 rows=1 width=48)	1	48	0.001	222	222
15	Index Cond	(act_id = movie_cast_act_id)					
16	Planning Time	0.310 ms					
17	Execution Time	9.653 ms					

Explanation :

- Metrics :

Execution Time : 9.653 ms Total Expected Cost : 3182.85

- Index created on column act_id of table actor, column mov_id of table movie_cast, column mov_id of table movie.
- The Search is done with $O(\log n)$ for search using values of column act_id of table actor, column mov_id of table movie_cast, column mov_id of table movie so the performance is improved with respect to performance of the query without index (Total Expected Cost Was 7907.55 and became 3182.85).

3. given 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 content:

```

1
2
3 -----Query 10 Original-----
4 explain analyze
5 select *
6 from actor
7 where act_id in(
8 select act_id
9 from movie_cast
10 where mov_id in(
11 select mov_id
12 from movie
13 where mov_title = 'Annie Hall'));
14 -----Query 10 Optimized + materialized view -----
15 create MATERIALIZED VIEW query_10
16 as
17 select mov_id from movie m2 where mov_title = 'Annie Hall' WITH
18
19
20 explain analyze select *
21 from actor
22 where act_id in( select act_id from movie_cast m1 where exists
23 -----Query 11 Original-----
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
39 mov_title='Eyes Wide Shut'))));
40

```

The right pane shows the query plan for the optimized query. The plan is as follows:

Step	Operation	Cost	Rows	Width	Actual Time	Actual Rows	Actual Loops
1	Nested Loop	(cost=3182.04..3182.15 rows=1 width=48)	1	48	8.510	8.828	rows=222 loops=1
2	HashAggregate	(cost=3182.04..3182.05 rows=1 width=4)	1	4	8.502	8.519	rows=222 loops=1
3	Group Key	movie_cast.act_id					
4	Batches	1 Memory Usage: 59kB					
5	Nested Loop	(cost=3174.00..3182.04 rows=1 width=4)	1	4	8.434	8.470	rows=222 loops=1
6	HashAggregate	(cost=3174.00..3174.01 rows=1 width=4)	1	4	8.404	8.404	rows=1 loops=1
7	Group Key	movie.mov_id					
8	Batches	1 Memory Usage: 24kB					
9	Seq Scan on movie	(cost=0.00..3174.00 rows=1 width=4)	1	4	0.007	8.398	rows=1 loops=1
10	Filter	(mov_title = 'Annie Hall'::bpchar)					
11	Rows Removed by Filter	99999					
12	Index Scan using b_movid on movie_cast	(cost=0.00..8.02 rows=1 width=8)	1	8	0.028	0.052	rows=222 loops=1
13	Index Cond	(mov_id = movie.mov_id)					
14	Index Scan using b_actid on actor	(cost=0.00..0.08 rows=1 width=48)	1	48	0.001	0.001	rows=1 loops=222
15	Index Cond	(act_id = movie_cast.act_id)					
16	Planning Time	0.388 ms					
17	Execution Time	8.862 ms					

Explanation :

- Metrics :

Execution Time : 8.862 ms Total Expected Cost : 3182.15

- Index created on column act_id of table actor, 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 3182.15).

4. given query with BRIN indices only :

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 query editor with the following SQL code:

```

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 view -----
13 create MATERIALIZED VIEW query_10
14 as
15 select mov_id from movie m2 where mov_title = 'Annie Hall' WITH data
16
17
18 explain analyze select *
19 from actor
20 where act_id in( select act_id
21                  from movie_cast m1
22                  where exists (select mov_id from query_10 m2
23                              where m1.mov_id=m2.mov_id ));
24 -----Query 11 Original-----
25
26 select dir_fname, dir_lname
27 from director
28 where dir_id in(
29 select dir_id
30 from movie_direction
31 where mov_id in(
32 select mov_id
33 from movie_cast
34 where role =any( select role
35                  from movie_cast
36                  where mov_id in(
37 select mov_id
38 from movie
39 where
40 mov_title='Eyes Wide Shut'))));
41

```

The right pane shows the query plan for the first query (Query 10). The plan is as follows:

```

QUERY PLAN
text
1  Nested Loop (cost=10001204198.66..10001204369.35 rows=1 width=48) (actual time=9.716..213.674 rows=222 loops=1)
2    [.]> HashAggregate (cost=10000004198.63..10000004198.64 rows=1 width=4) (actual time=9.662..9.734 rows=222 loops=1)
3      [.] Group Key: movie_cast.act_id
4      [.] Batches: 1 Memory Usage: 56kB
5    [.]> Nested Loop (cost=10000003186.03..10000004198.63 rows=1 width=4) (actual time=8.366..9.583 rows=222 loops=1)
6      [.]> HashAggregate (cost=10000003174.00..10000003174.01 rows=1 width=4) (actual time=8.261..8.262 rows=1 loops=1)
7        [.] Group Key: movie.mov_id
8        [.] Batches: 1 Memory Usage: 24kB
9      [.]> Seq Scan on movie (cost=10000000000.00..10000003174.00 rows=1 width=4) (actual time=0.006..8.255 rows=1 loops=1)
10        Filter: (mov_title = 'Annie Hall'::bpchar)
11        Rows Removed by Filter: 999999
12      [.]> Bitmap Heap Scan on movie_cast (cost=12.03..1024.61 rows=1 width=8) (actual time=0.090..1.296 rows=222 loops=1)
13        Recheck Cond: (mov_id = movie.mov_id)
14        Rows Removed by Index Recheck: 15138
15        Heap Blocks: losty=128
16      [.]> Bitmap Index Scan on b_movid (cost=0.00..12.03 rows=14286 width=0) (actual time=0.081..0.081 rows=1 loops=1)
17        Index Cond: (mov_id = movie_mov_id)
18      [.]> Bitmap Heap Scan on actor (cost=1200000.03..1200170.69 rows=1 width=48) (actual time=0.018..0.915 rows=1 loops=1)
19        Recheck Cond: (act_id = movie_cast.act_id)
20        Rows Removed by Index Recheck: 13695
21        Heap Blocks: losty=28416
22      [.]> Bitmap Index Scan on b_actid (cost=0.00..1200000.03 rows=13333 width=0) (actual time=0.009..0.009 rows=1280 loops=1)
23        Index Cond: (act_id = movie_cast.act_id)
24  Planning Time: 0.561 ms
25  Execution Time: 213.881 ms

```

Explanation :

- Metrics :

Execution Time : 213.881 ms Total Expected Cost : 10001204369.35

- Index created on column act_id of table actor, 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 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 the 'Query Editor' tab active. The left sidebar shows the database structure, including 'PostgreSQL 14', 'Databases (5)', and 'schema4'. The main editor displays two queries, 'Query 10' and 'Query 11', both marked as 'Original'. The 'Query 10' query is highlighted in blue and includes an 'explain analyze' statement. The 'Query 11' query is also highlighted in blue and includes an 'explain analyze' statement. The right pane shows the 'Query Plan' for 'Query 10', which is a Nested Loop. The plan details include costs, row counts, and execution times for various steps, such as 'Nested Loop (cost=3182.34..3182.44 rows=1 width=48)' and 'HashAggregate (cost=3174.00..3174.01 rows=1 width=4)'. The overall execution time for 'Query 10' is 9.546 ms.

Explanation :

- Metrics :

Execution Time : 9.546 ms Total Expected Cost : 3182.44

- Index created on column act_id of table actor (Hash Index), column mov_id of table movie_cast (B+Tree), column mov_id of table movie (B+Tree).
- Now the Performance is better than with B+ index but performance with Hash only is better as any search with values of column act_id of table actor will be in O(1) and with values of column mov_id of table movie_cast or column mov_id of table movie will be in O(Log n).

Optimized Query

```
create MATERIALIZED VIEW query_10
as
select mov_id from movie m2 where mov_title = 'Annie Hall' WITH data ;

explain analyze select *
from actor
where act_id in( select act_id
                  from movie_cast m1
                  where exists (select mov_id from query_10 m2
                                where m1.mov_id=m2.mov_id ) );
```

- 222 Rows

1. Optimized Query without an index :

Explanation :

- Metrics :

Execution Time : 20.605 ms Total Expected Cost : 4869.18

- Reason : Since selecting movie ids of any movie with title Annie Hall is needed as sub query so i created materialized view storing those values with name of the view Query 10 and used it inside my query.In addition, Because the loops ends faster I used exists operator instead of the In operator. Finally, the performance is improved with respect to performance of original query (Total Expected Cost Was 7907.55 and became 4869.18).

2. Optimized Query with B+ trees indices only :

The screenshot shows the pgAdmin 4 interface with a PostgreSQL query editor. The left sidebar shows the database structure, including schemas and tables. The main window displays the 'Query Editor' with two queries. The first query, 'Query 10 Original', is a nested loop join. The second query, 'Query 10 Optimized + materialized view', is a nested loop join with a materialized view. The execution plan for 'Query 10 Optimized' is shown on the right, detailing the nested loop join and the use of B+ tree indices on act_id and mov_id.

```

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

```

The execution plan for 'Query 10 Optimized' is shown on the right, detailing the nested loop join and the use of B+ tree indices on act_id and mov_id.

QUERY PLAN	cost	actual time	loops
1 Nested Loop (cost=1418.72..2737.12 rows=2570 width=48) (actual time=0.075..0.339 rows=222 loops=1)			
2 [...] HashAggregate (cost=1418.30..1444.00 rows=2570 width=4) (actual time=0.070..0.098 rows=222 loops=1)			
3 [...] Group Key: m1.act_id			
4 [...] Batches: 1 Memory Usage: 129kB			
5 [...] Nested Loop (cost=42.17..1411.87 rows=2570 width=4) (actual time=0.017..0.045 rows=222 loops=1)			
6 [...] HashAggregate (cost=41.88..43.88 rows=200 width=4) (actual time=0.011..0.011 rows=1 loops=1)			
7 [...] Group Key: m2.mov_id			
8 [...] Batches: 1 Memory Usage: 40kB			
9 [...] Seq Scan on query_10 m2 (cost=0.00..35.50 rows=2550 width=4) (actual time=0.007..0.008 rows=1 loops=1)			
10 [...] Index Scan using b_movid on movie_cast m1 (cost=0.29..6.83 rows=1 width=8) (actual time=0.006..0.024 rows=2..)			
11 [...] Index Cond: (mov_id = m2.mov_id)			
12 [...] Index Scan using b_actid on actor (cost=0.42..0.49 rows=1 width=48) (actual time=0.001..0.001 rows=1 loops=22..)			
13 [...] Index Cond: (act_id = m1.act_id)			
14 Planning Time: 0.183 ms			
15 Execution Time: 0.420 ms			

Explanation :

- Metrics :

Execution Time : 0.420 ms Total Expected Cost : 2737.12

- Index created on column act_id of table actor, column mov_id of table movie_cast.
- The Search is done with $O(\log n)$ for search using values of column act_id of table actor, 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 4869.18 and became 2737.12).

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 in the 'Query Editor' and its execution plan in the 'Data Output' pane.

Query Editor:

```

1 select
2 from actor
3 where act_id in(
4 select act_id
5 from movie_cast
6 where mov_id in(
7 select mov_id
8 from movie
9 where mov_title = 'Annie Hall'));
10 -----Query 10 Optimized + materialized view -----
11 create MATERIALIZED VIEW query_10
12 as
13 select mov_id from movie m2 where mov_title = 'Annie Hall' WITH
14
15
16
17
18
19
20 explain analyze select *
21 from actor
22 where act_id in( select act_id from movie_cast m1 where exists
23 -----Query 11 Original-----
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
39 mov_title='Eyes Wide Shut'))));
40 -----Query 11 Optimized + materialized view -----
41 Query 11
42 CREATE MATERIALIZED VIEW query_11
43 AS
44 select role
45

```

Data Output (Query Plan):

Step	Plan	Cost	Rows	Width	Actual Time	Actual Rows	Loops
1	Nested Loop	1443.80	1708.36	2570	width=48	(actual time=0.100..0.700 rows=222 loops=1)	
2	[.]> HashAggregate	(cost=1443.80..1469.50 rows=2570 width=4)	(actual time=0.095..0.122 rows=222 loops=1)				
3	[.] Group Key: m1.act_id						
4	[.] Batches: 1 Memory Usage: 129kB						
5	[.]> Nested Loop	(cost=41.88..1437.38 rows=2570 width=4)	(actual time=0.022..0.063 rows=222 loops=1)				
6	[.]> HashAggregate	(cost=41.88..43.88 rows=200 width=4)	(actual time=0.014..0.014 rows=1 loops=1)				
7	[.] Group Key: m2.mov_id						
8	[.] Batches: 1 Memory Usage: 40kB						
9	[.]> Seq Scan on query_10 m2	(cost=0.00..35.50 rows=2550 width=4)	(actual time=0.009..0.009 rows=1 loops=1)				
10	[.]> Index Scan using b_movid on movie_cast m1	(cost=0.00..6.96 rows=1 width=8)	(actual time=0.007..0.035 rows=2..)				
11	[.] Index Cond: (mov_id = m2.mov_id)						
12	[.]> Index Scan using b_actid on actor	(cost=0.00..0.08 rows=1 width=48)	(actual time=0.002..0.002 rows=1 loops=22..)				
13	[.] Index Cond: (act_id = m1.act_id)						
14	Planning Time: 0.123 ms						
15	Execution Time: 0.749 ms						

Explanation :

- Metrics :

Execution Time : 0.749 ms Total Expected Cost : 1708.36

- Index created on column act_id of table actor, column mov_id of table movie_cast, column mov_id of table movie.
- 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 4869.18 without index and 2737.12 with B+ tree and became 1708.36).

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 hierarchy, including 'schema4'. The main window displays a query editor with the following SQL:

```

1
2
3 -----Query 10 Original-----
4 explain analyze
5 select *
6 from actor
7 where act_id in(
8 select act_id
9 from movie_cast
10 where mov_id in(
11 select mov_id
12 from movie
13 where mov_title = 'Annie Hall');
14 -----Query 10 Optimized + materialized view -----
15 create MATERIALIZED VIEW query_10
16 as
17 select mov_id from movie m2 where mov_title = 'Annie Hall' WITH data ;
18
19
20 explain analyze select *
21 from actor
22 where act_id in( select act_id from movie_cast m1 where exists (select mov
23 -----Query 11 Original-----
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
39 mov_title='Eyes Wide Shut'))));
40

```

The right pane shows the 'QUERY PLAN' for the original query (Query 10). The plan is as follows:

```

QUERY PLAN
text
Nested Loop (cost=10001719541.50..13084958278.37 rows=2570 width=48) (actual time=1.310..195.968 rows=222 lo...
[.]> HashAggregate (cost=10000519541.46..10000519567.16 rows=2570 width=4) (actual time=1.271..1.339 rows=2...
[.] Group Key: m1.act_id
[.] Batches: 1 Memory Usage: 129kB
[.]> Nested Loop (cost=10000002441.91..10000519535.04 rows=2570 width=4) (actual time=0.034..1.245 rows=222...
[.]> HashAggregate (cost=10000000041.88..10000000043.88 rows=200 width=4) (actual time=0.010..0.011 rows=1 L...
[.] Group Key: m2.mov_id
[.] Batches: 1 Memory Usage: 40kB
[.]> Seq Scan on query_10 m2 (cost=10000000000.00..10000000035.50 rows=2550 width=4) (actual time=0.006..0.0...
[.]> Bitmap Heap Scan on movie_cast m1 (cost=2400.03..2997.45 rows=1 width=8) (actual time=0.020..1.219 rows=2...
[.] Recheck Cond: (mov_id = m2.mov_id)
[.] Rows Removed by Index Recheck: 15138
[.] Heap Blocks: lossy=128
[.]> Bitmap Index Scan on b_movid (cost=0.00..2400.03 rows=14286 width=0) (actual time=0.015..0.015 rows=1280 L...
[.] Index Cond: (mov_id = m2.mov_id)
[.]> Bitmap Heap Scan on actor (cost=1200000.03..1200170.69 rows=1 width=48) (actual time=0.015..0.873 rows=1 L...
[.] Recheck Cond: (act_id = m1.act_id)
[.] Rows Removed by Index Recheck: 13695
[.] Heap Blocks: lossy=28416
[.]> Bitmap Index Scan on b_actid (cost=0.00..1200000.03 rows=13333 width=0) (actual time=0.007..0.007 rows=128...
[.] Index Cond: (act_id = m1.act_id)
Planning Time: 0.188 ms
Execution Time: 197.457 ms

```

Explanation :

- Metrics :

Execution Time : 197.457 ms Total Expected Cost : 13084958278.37

- Index created on column act_id of table actor, 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 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 PostgreSQL database connection. The left sidebar displays the database structure, including schemas and tables. The main window shows a SQL query editor with the following content:

```

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 view -----
13 create MATERIALIZED VIEW query_10
14 as
15 select mov_id from movie m2 where mov_title = 'Annie Hall' WITH data
16
17
18 explain analyze select *
19 from actor
20 where act_id in( select act_id
21                  from movie_cast m1
22                  where exists (select mov_id from query_10 m2
23                               where m1.mov_id=m2.mov_id ) );
24
25 CREATE INDEX b_actID ON actor USING hash(act_id );
26 CREATE INDEX b_movID ON movie_cast USING btree(mov_id );
27 -----Query 11 Original-----
28 explain analyze
29 select dir_fname, dir_lname
30 from director
31 where dir_id in(
32 select dir_id
33 from movie_direction
34 where mov_id in(
35 select mov_id
36 from movie_cast
37 where role =any( select role
38                  from movie_cast
39                  where mov_id in(
40 select mov_id
41

```

The right pane shows the query plan for the executed query. The plan includes a nested loop join, a hash aggregate, and a seq scan on the materialized view. The execution time is 0.576 ms.

Explanation :

- Metrics :

Execution Time : 0.576 ms Total Expected Cost : 1682.75

- Index created on column act_id of table actor (Hash Index), column mov_id of table movie_cast (B+ Tree).
- Now the Performance is better than with B+ index but performance with Hash only is slightly better as any search with values of column act_id like of table actor will be in O(1) and with values of column mov_id of table movie_cast will be in O(Log n).