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

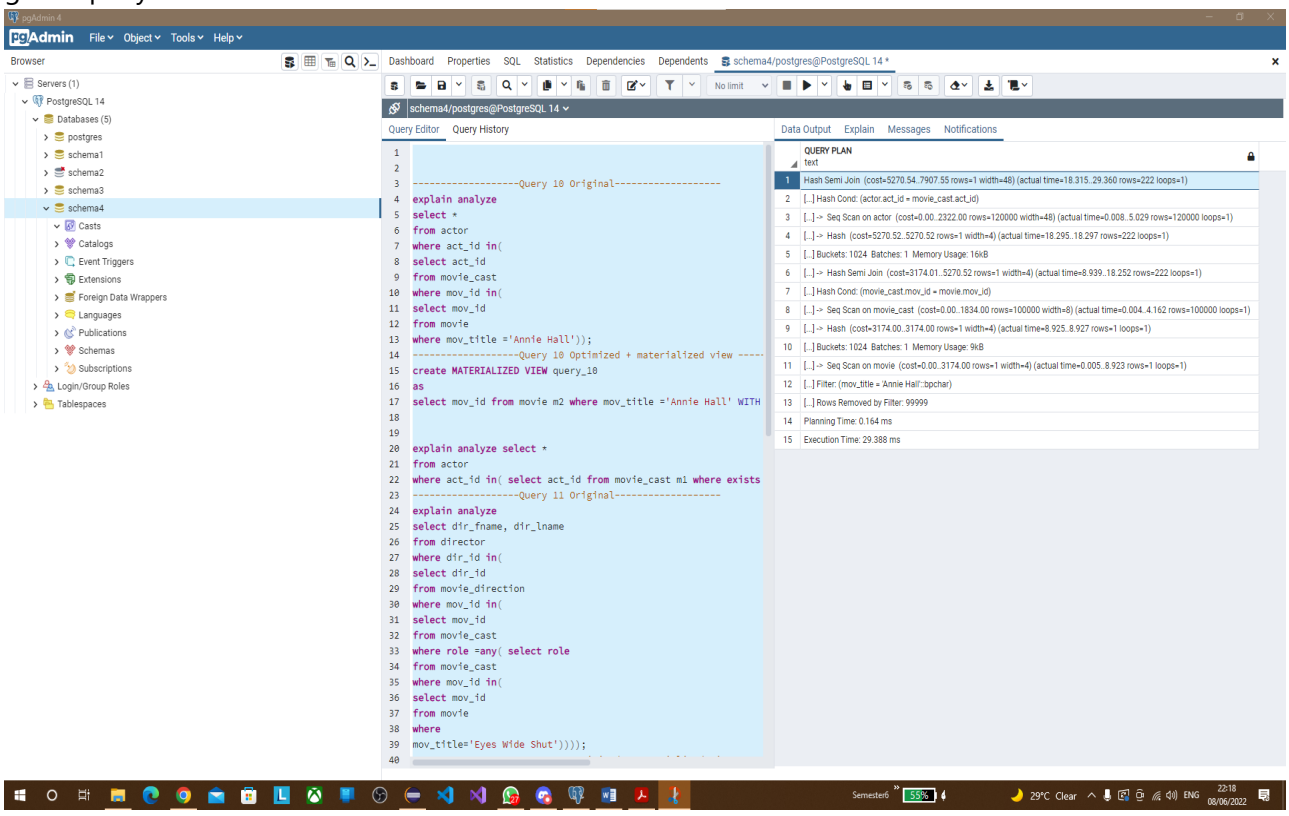
```
select *
from actor
where act_id in(
select act_id
from movie_cast
where mov_id in(
select mov_id
from movie
where mov_title ='Annie Hall'));
```

Result Set

- 222 Rows

Report

1. given query without an index :

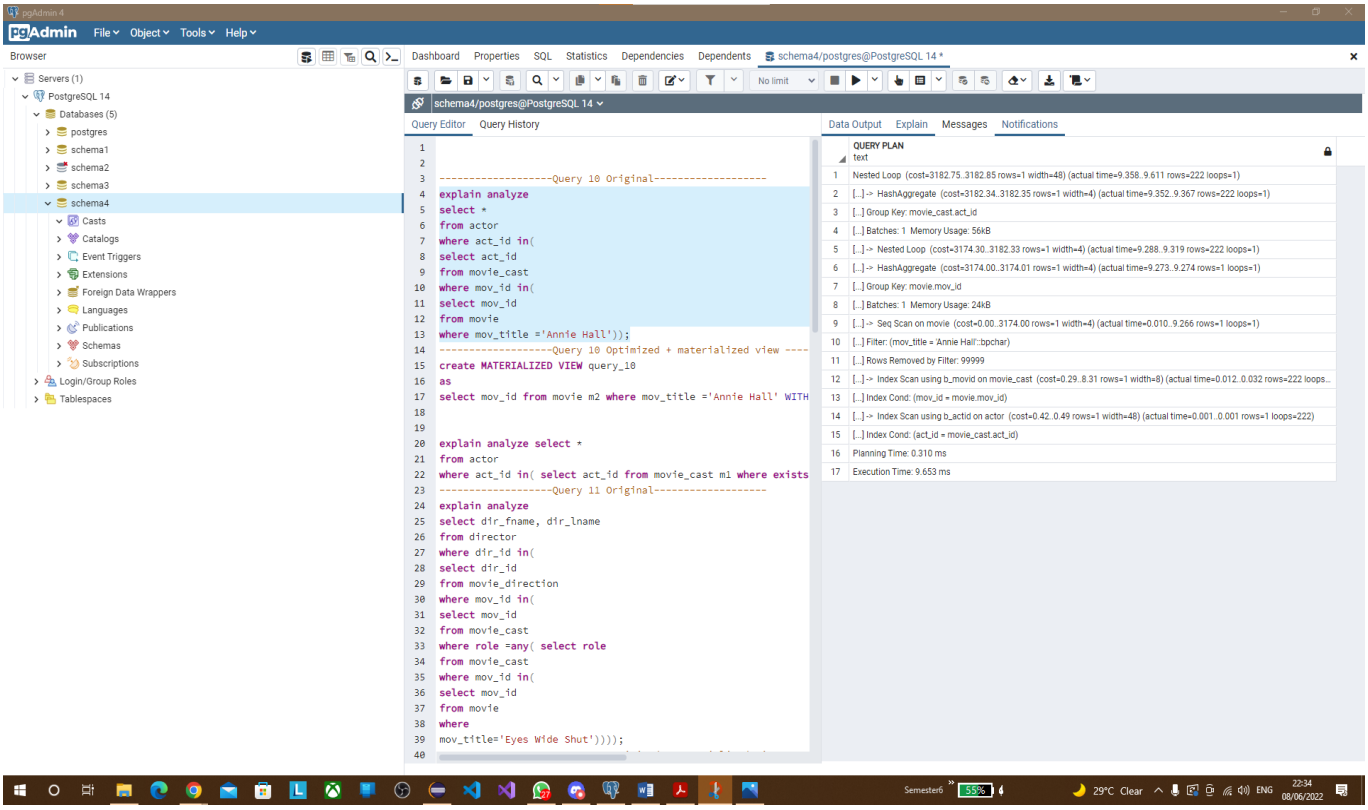


Explanation :

- Metrics :

Execution Time : 29.388 ms Total Expected Cost : 7907.55

2. given query with B+ trees indices only :



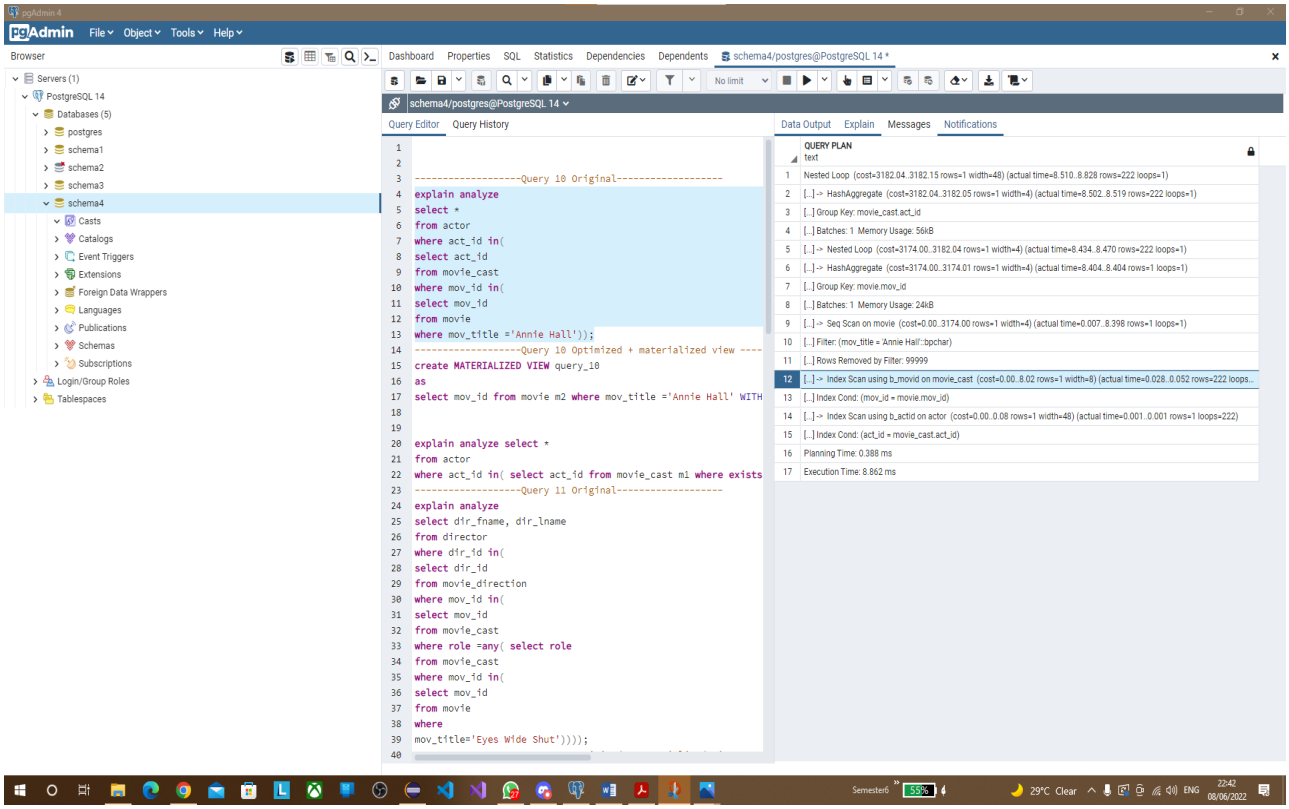
Explanation :

- 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 :



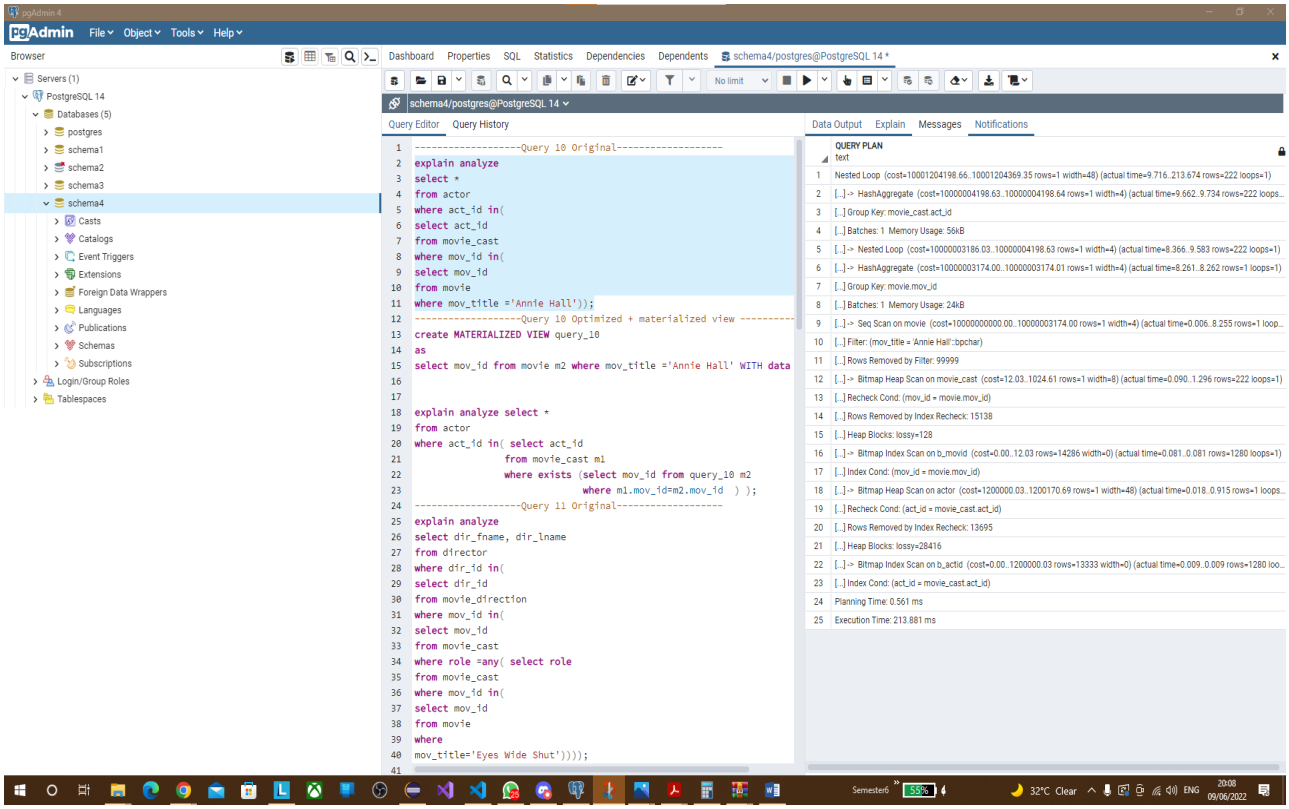
Explanation :

- 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 :




```
------------------Query 10 Original------------------
explain analyze
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'));
------------------Query 10 Optimized + materialized view ----------
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  ) );
------------------Query 11 Original------------------
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'))));
```

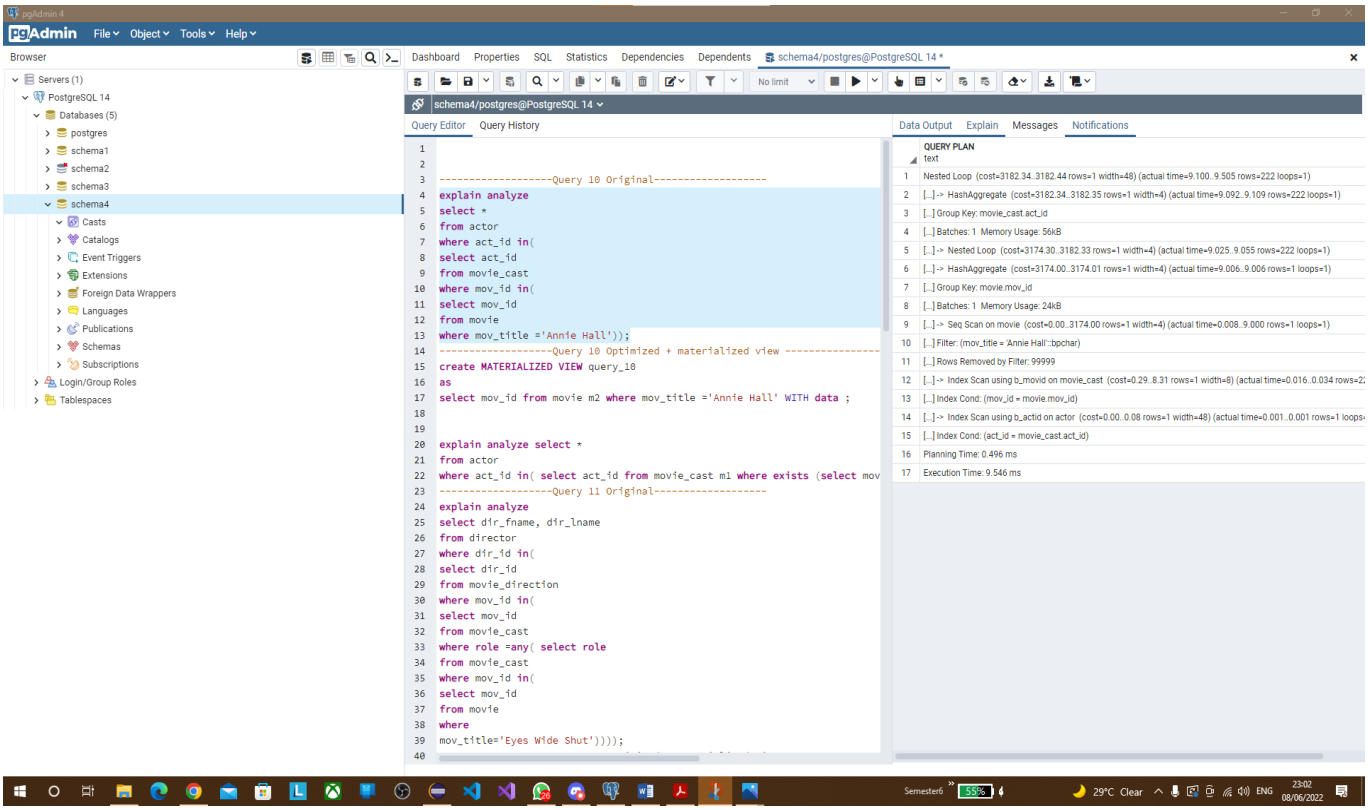
Explanation :

- 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) :



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

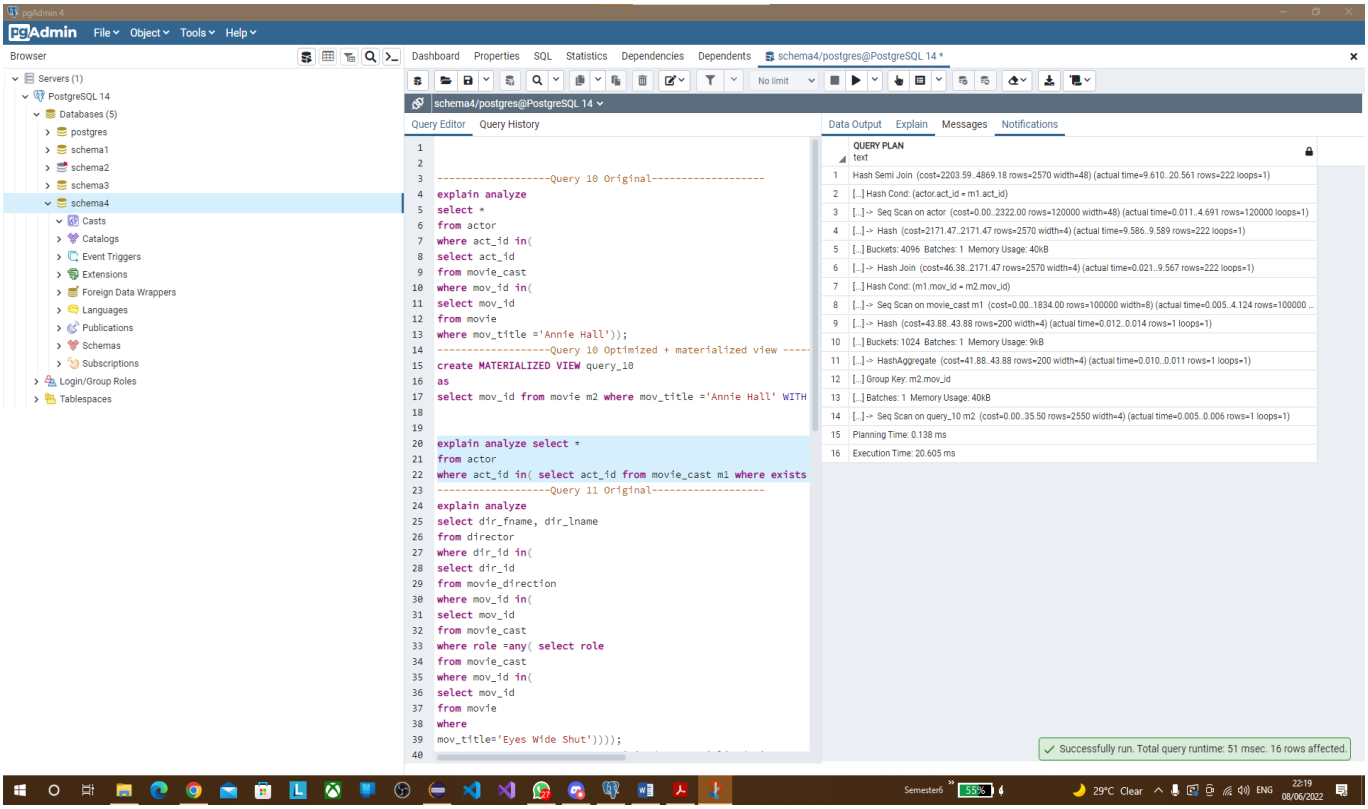
```
create MATERIALIZED VIEW query_10
as
select mov_id from movie m2 where mov_title ='Annie Hall' WITH data ;


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  ) );
```

Result Set

- 222 Rows

Report

1. Optimized Query without an index :



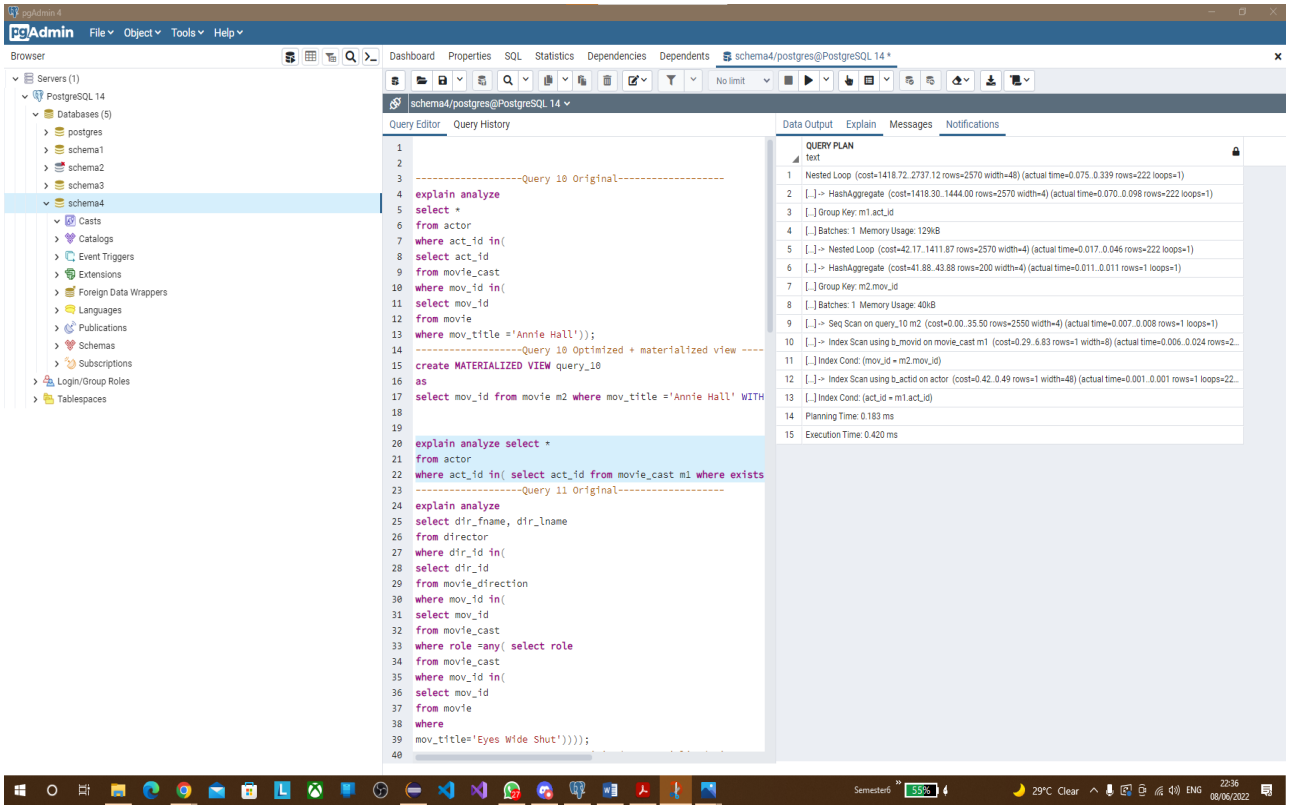
Explanation :

- 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 :




```
------------------Query 10 Original------------------
explain analyze
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'));
------------------Query 10 Optimized + materialized view ----
create MATERIALIZED VIEW query_10
as
select mov_id from movie m2 where mov_title ='Annie Hall' WITH


explain analyze select *
from actor
where act_id in( select act_id from movie_cast m1 where exists
------------------Query 11 Original------------------
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'))));
```

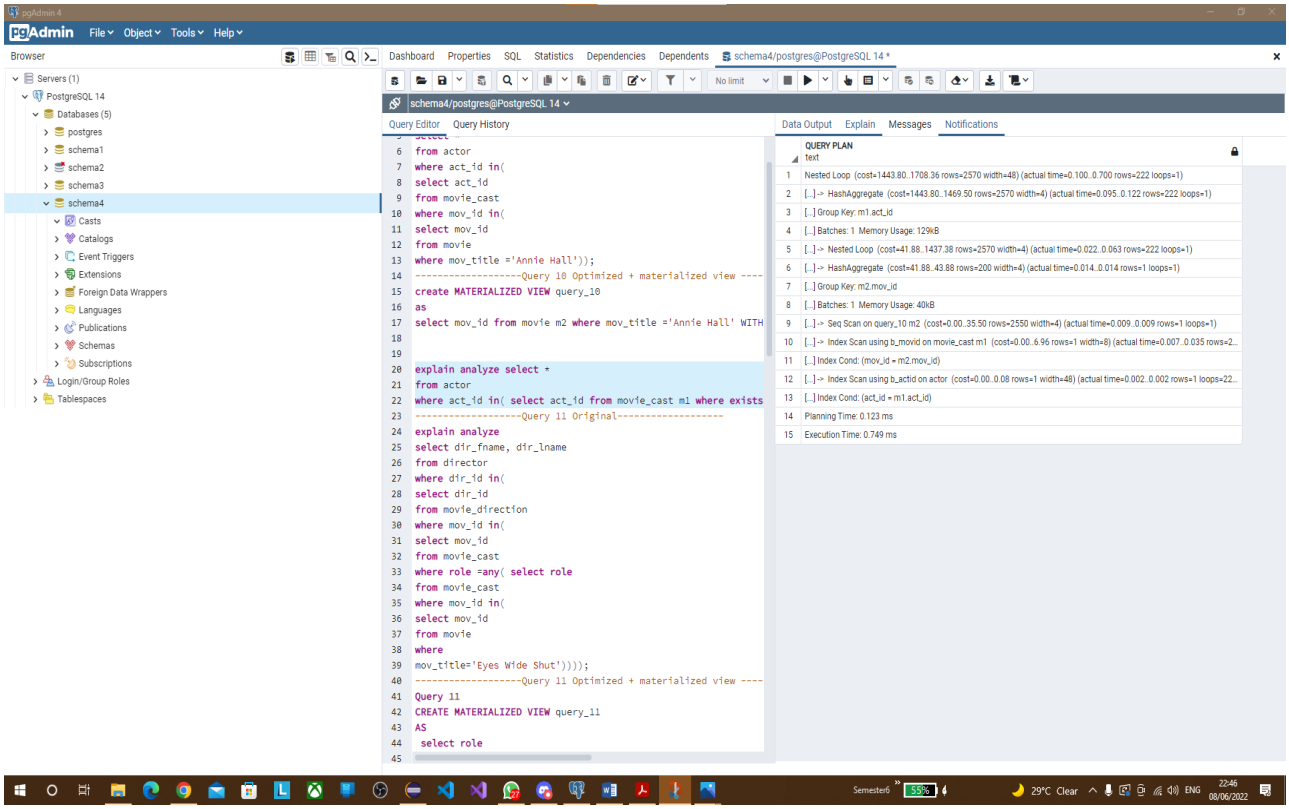
Explanation :

- 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 :



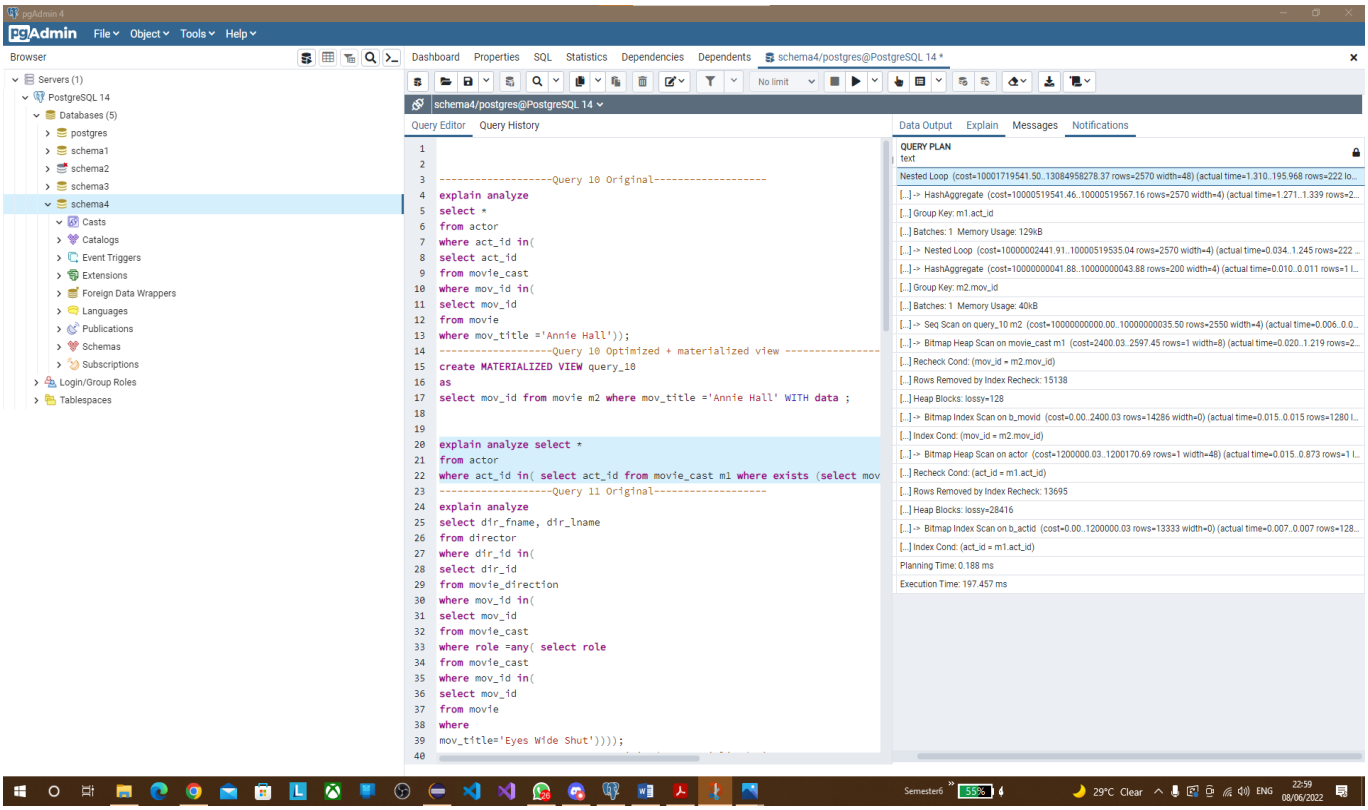
Explanation :

- 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 :



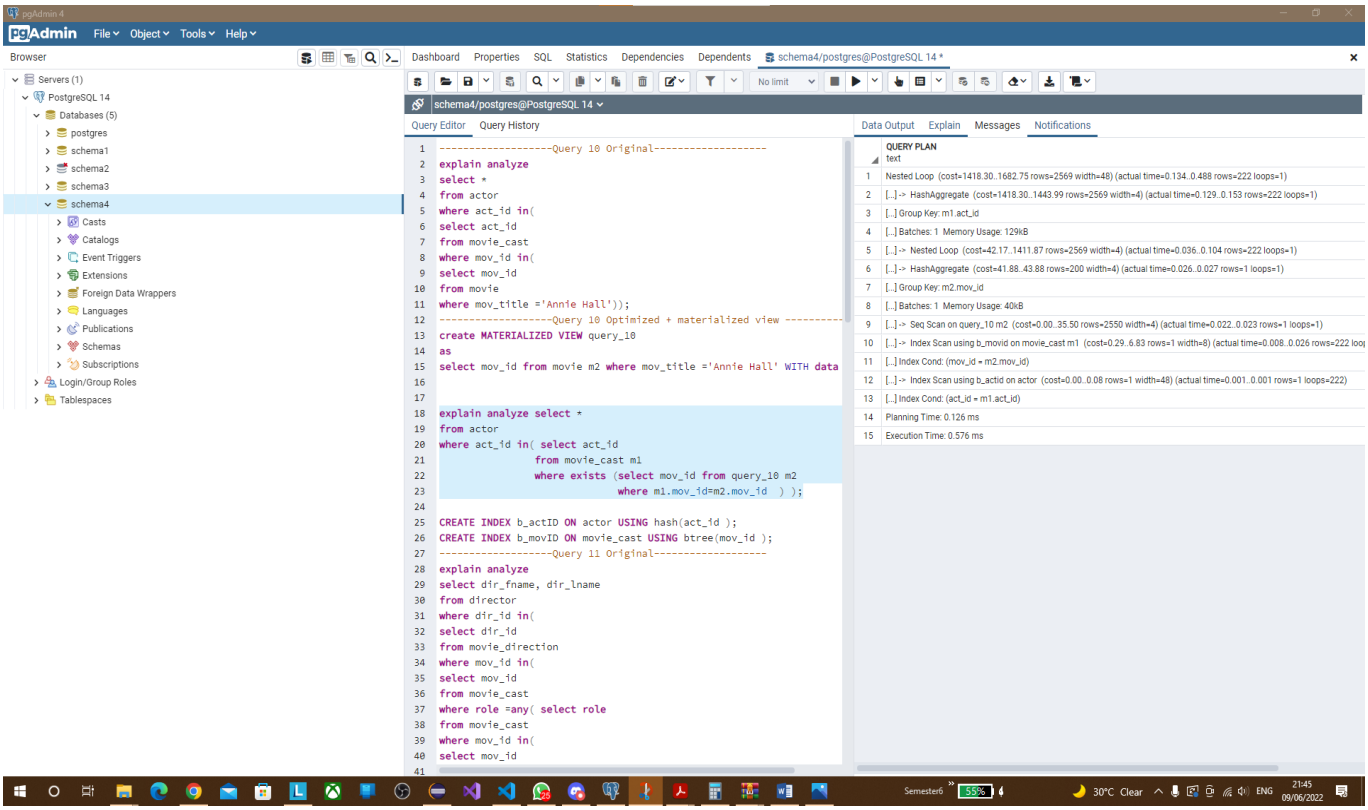
Explanation :

- 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) :




```
------------------Query 10 Original------------------
explain analyze
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'));
------------------Query 10 Optimized + materialized view ----------
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  ) );

CREATE INDEX b_actID ON actor USING hash(act_id );
CREATE INDEX b_movID ON movie_cast USING btree(mov_id );
------------------Query 11 Original------------------
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
```

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).