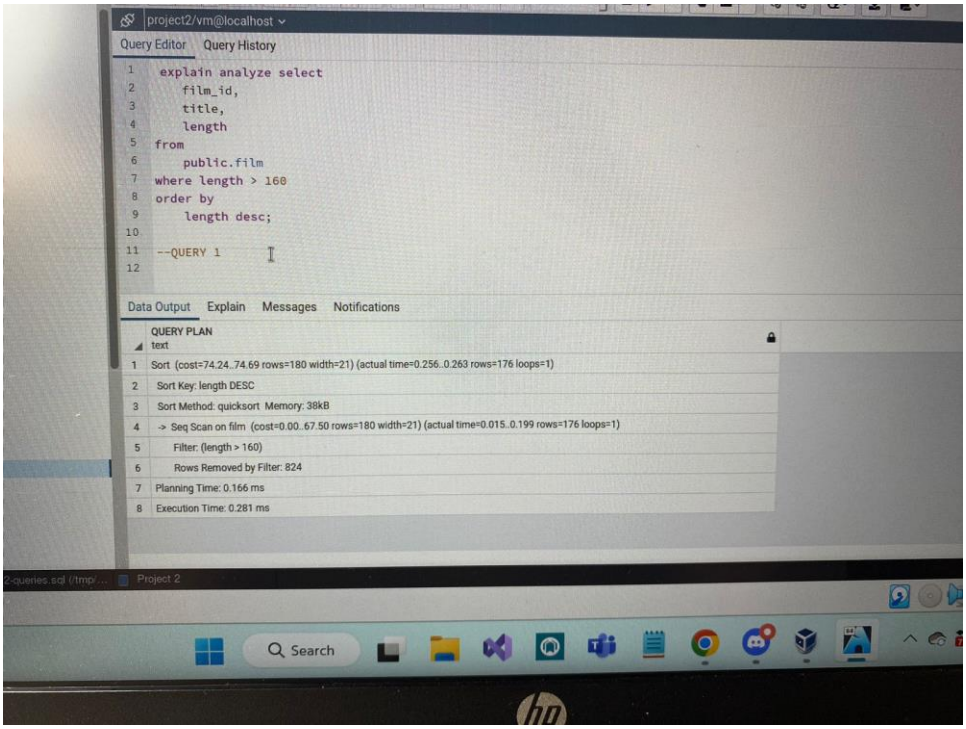
Query 1:

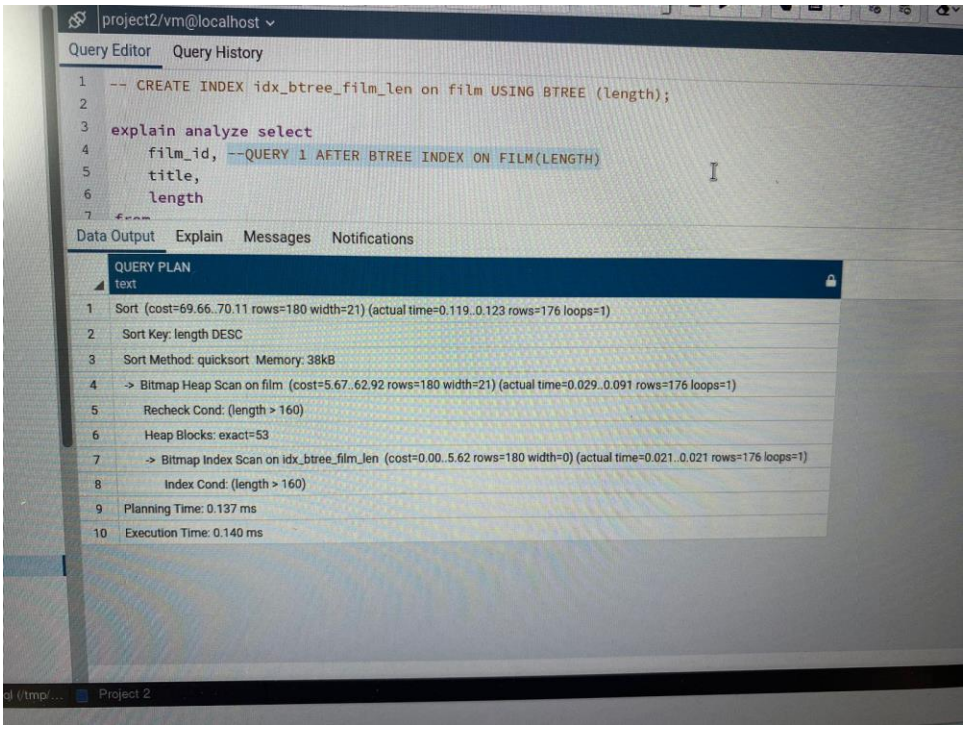
Before Query 1:

- **Highest Cost:** Seq scan on film
- **Slowest Runtime:** Seq scan on film
- **Largest Number of Rows:** sort and seq scan on film
- **Planning Time:** 0.166
- **Execution Time:** 0.281



After Query 1:

- **Highest Cost:** Bitmap Heap Scan on Film
- **Slowest Runtime:** Bitmap Heap Scan on Film
- **Largest Number of Rows:** Sort
- **Planning Time:** 0.137
- **Execution Time:** 0.140



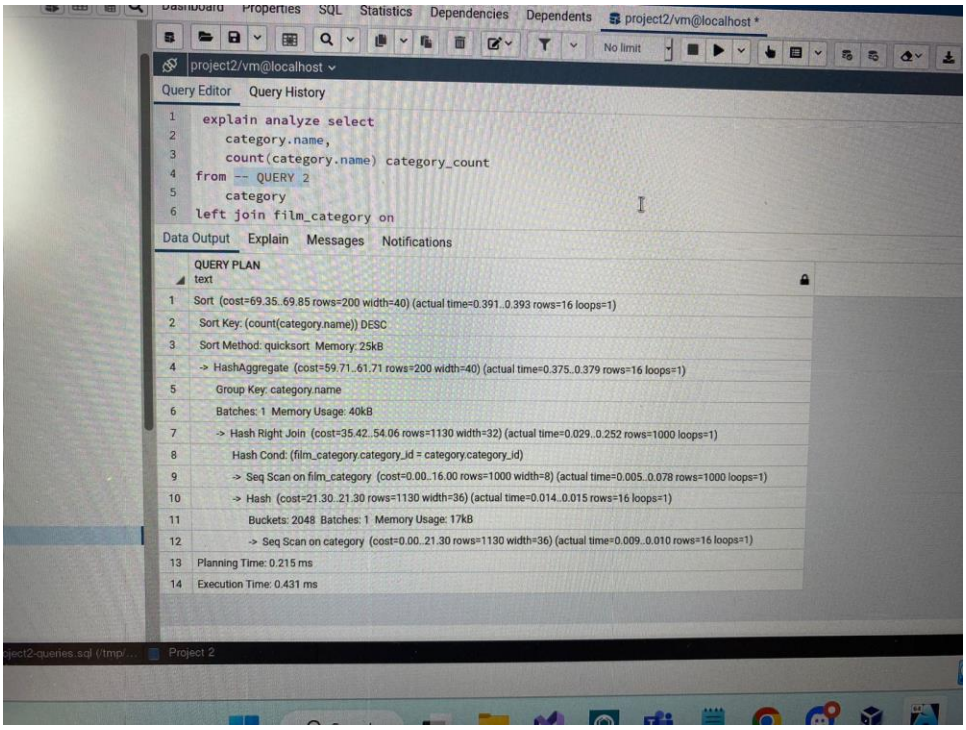
Query (1) Justification:

- Index On Attribute: Table-Name = film, Attribute-Name = length, we used a Btree index as the ‘Where’ clause contained a range.
- We did not change anything in the config file.
- The planner used our created index instead of using ‘seq scan’ on film, without the need of modifying any techniques in the config file which results in better planning, execution times.

<u>Query 2:</u>

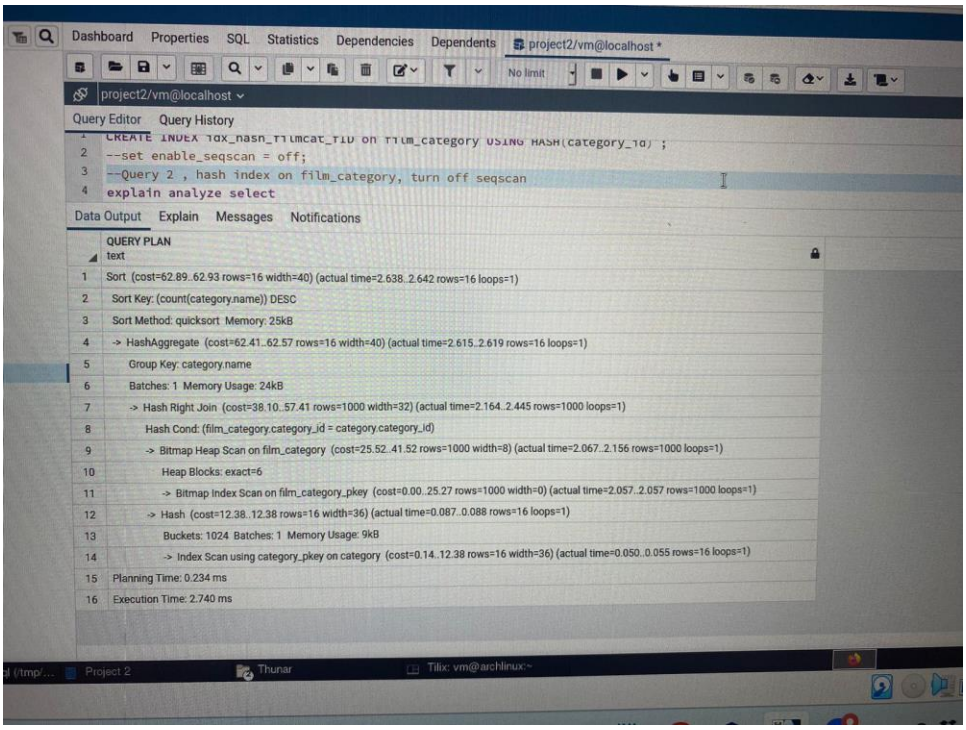
Before Query 2:

- **Highest Cost:** Seq scan on category
- **Slowest Runtime:** hash right join
- **Largest Number of Rows:** Seq scan on category and hash right join
- **Planning Time:** 0.215
- **Execution Time:** 0.431



After Query 2:

- **Highest Cost:** bitmap index scan on film_category_pkey
- **Slowest Runtime:** hash right join
- **Largest Number of Rows:** bitmap index scan on film_category_pkey and hash right join
- **Planning Time:** 0.234
- **Execution Time:** 2.740



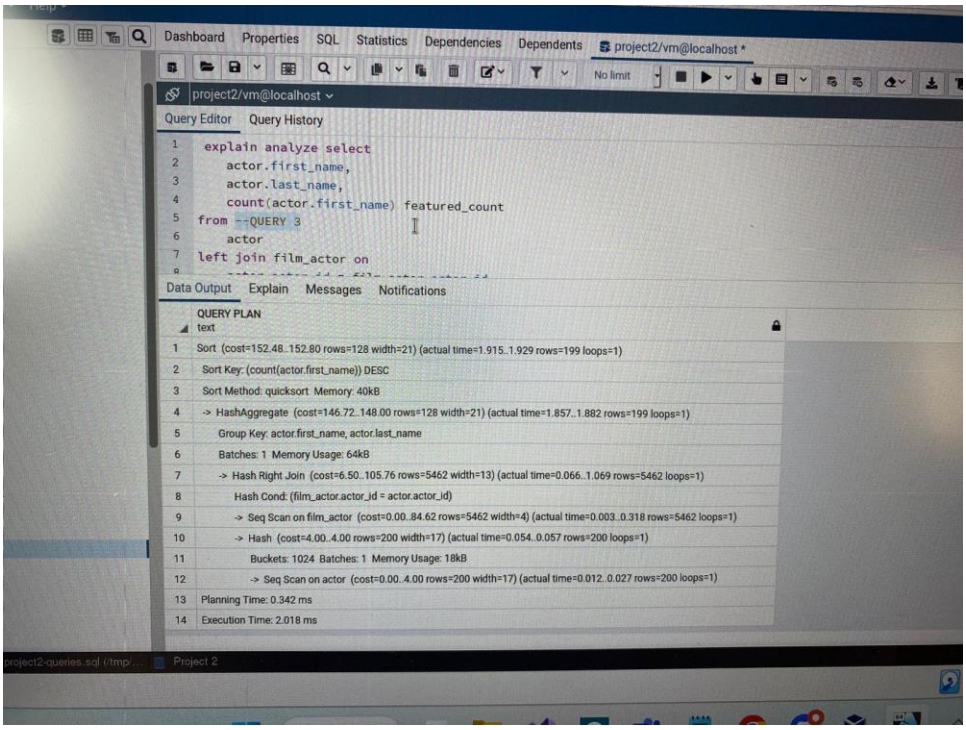
Query (2) Justification:

- Index On Attribute: Table-Name = film_category, Attribute-Name = category_id, we used a Hash index as we had a simple equality to join the two tables using the id's.
- We turned off the seq scan to make the planner use our index.
- The planner didn't use our index even after disabling the seq scan, as planner sees bitmap index scan on film category using film category primary key is a better option for optimizing the query than using our created index.

<u>Query 3:</u>

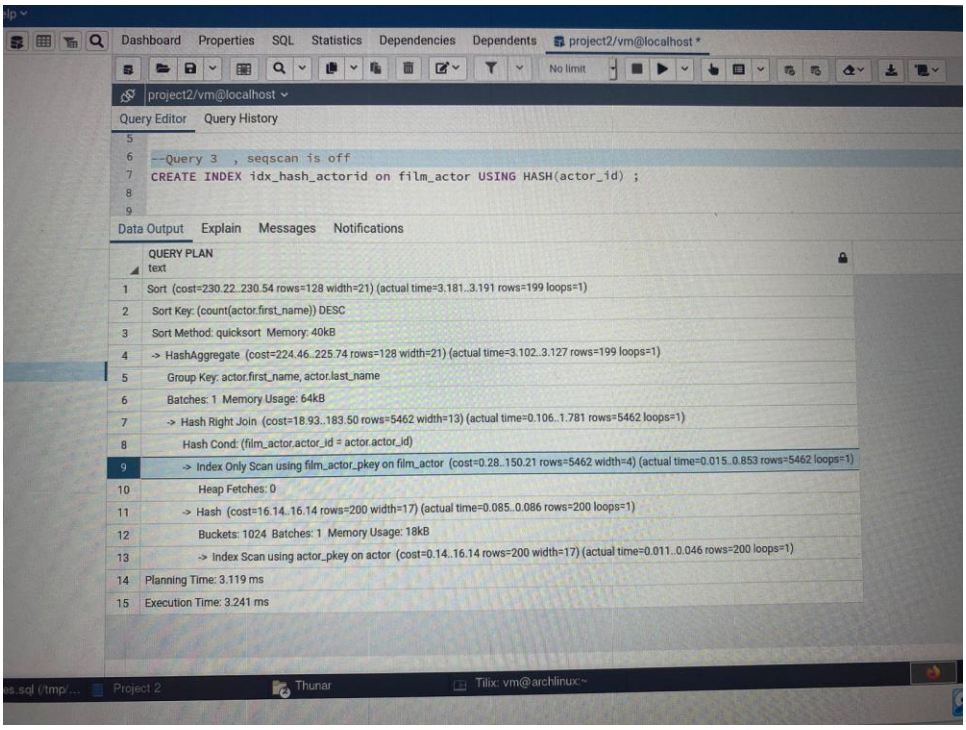
Before Query 3:

- **Highest Cost:** hash right join
- **Slowest Runtime:** seq scan on film actor
- **Largest Number of Rows:** hash right join and seq scan on film actor
- **Planning Time:** 0.342
- **Execution Time:** 2.018



After Query 3:

- **Highest Cost:** index only scan using film_actor_pkey
- **Slowest Runtime:** index only scan using film_actor_pkey
- **Largest Number of Rows:** hash right join
- **Planning Time:** 3.119
- **Execution Time:** 3.241



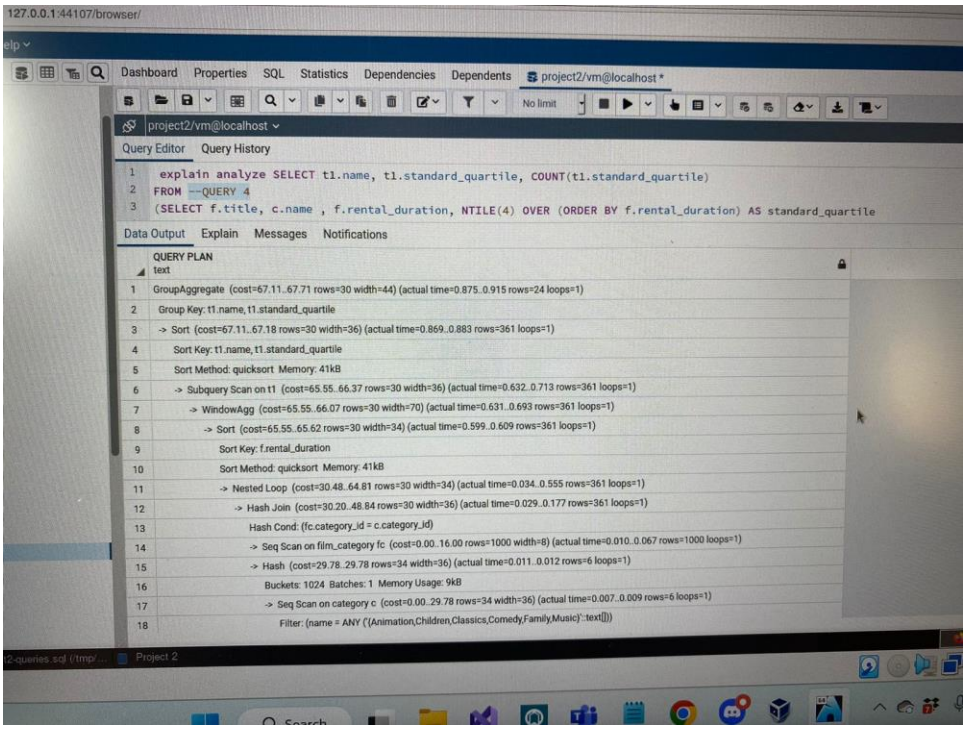
Query (3) Justification:

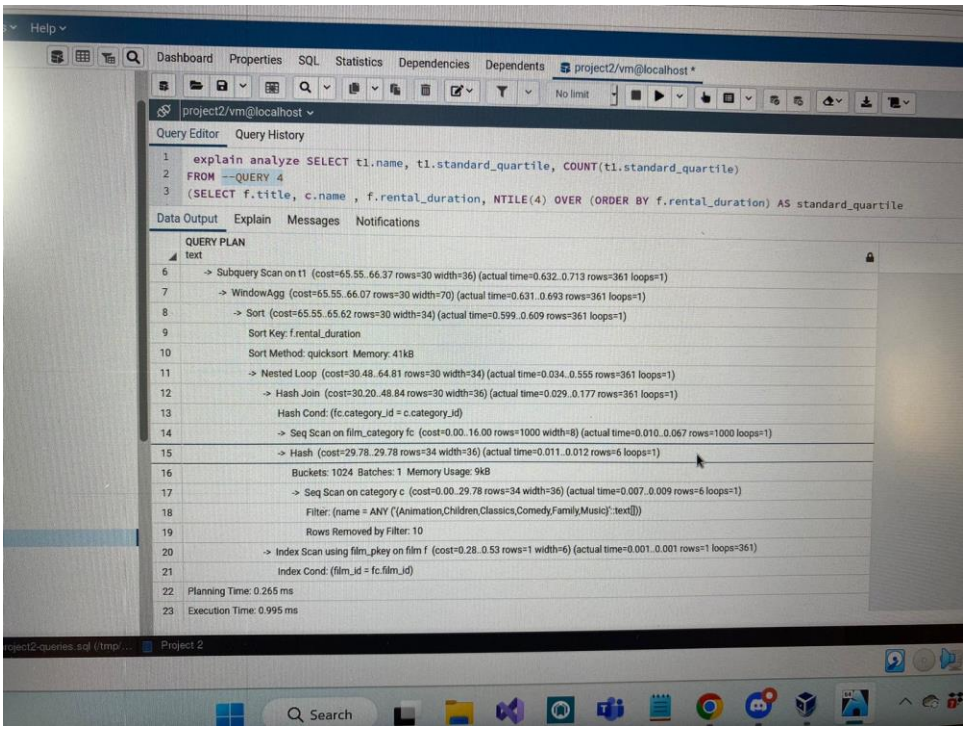
- Index On Attribute: Table-Name = film_actor, Attribute-Name = actor_id, we used a Hash index as we had a simple equality to join the two tables using the id's.
- We turned off the seq scan to make the planner use our index.
- The planner didn't use our index even after disabling the seq scan, as planner sees index only scan on film actor using film actor primary key is a better option for optimizing the query than using our created index.

Query 4:

Before Query 4:

- **Highest Cost:** Nested loop
- **Slowest Runtime:** Nested loop
- **Largest Number of Rows:** seq scan on film_category fc
- **Planning Time:** 0.265
- **Execution Time:** 0.995

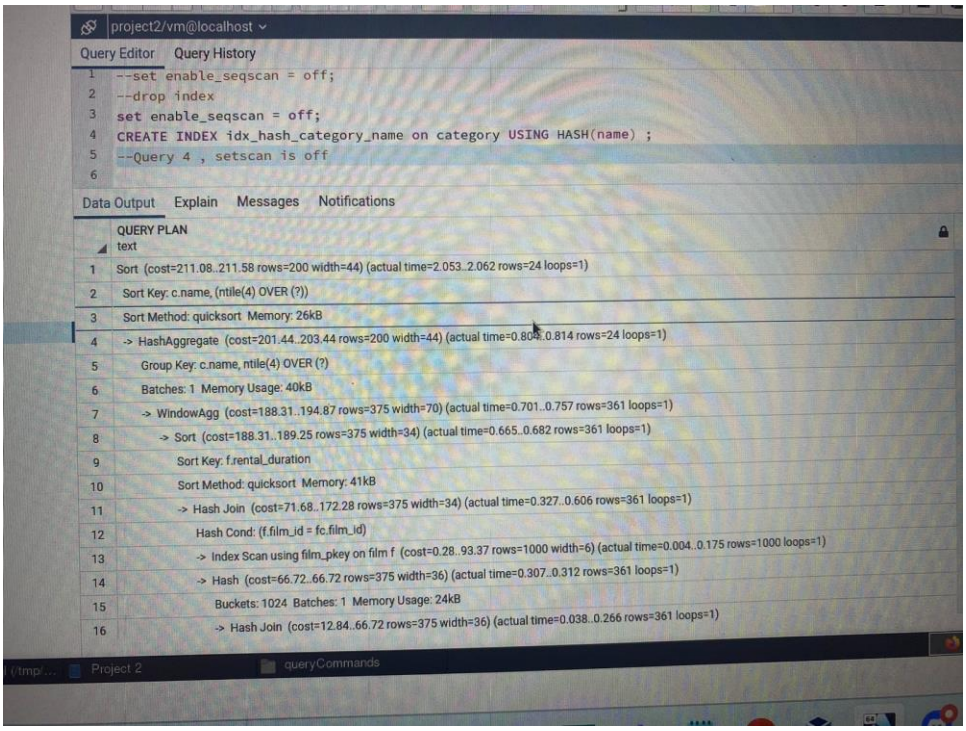


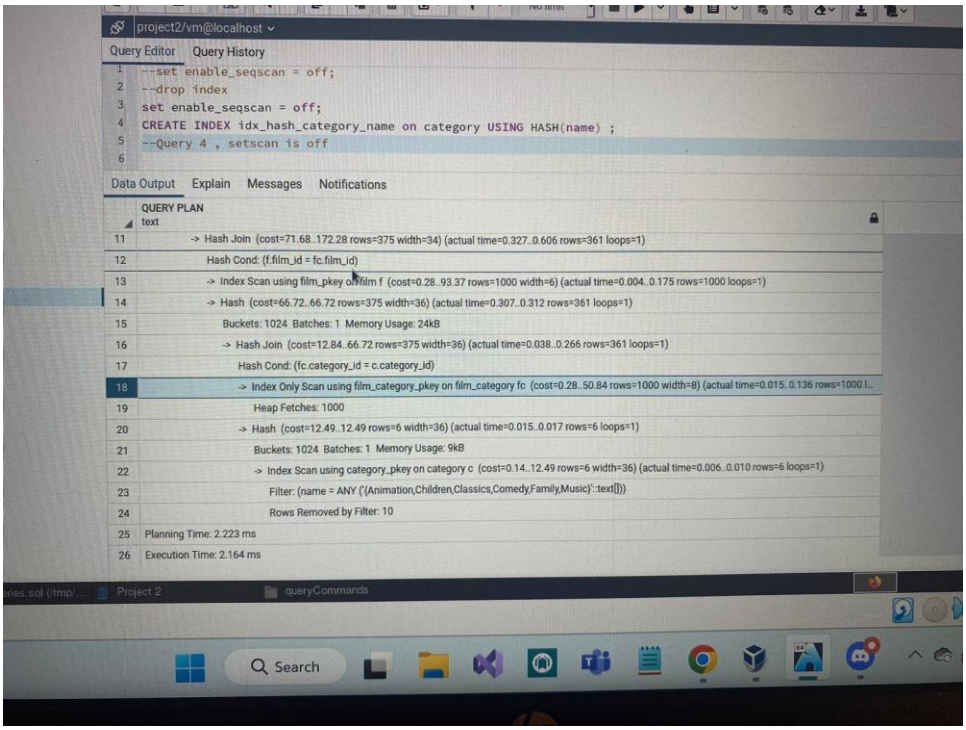


After Query 4:

- **Highest Cost:** index scan using film_pkey on film f
- **Slowest Runtime:** hash join

- **Largest Number of Rows:** index scan using film_pkey on film f
- **Planning Time:** 2.223
- **Execution Time:** 2.164





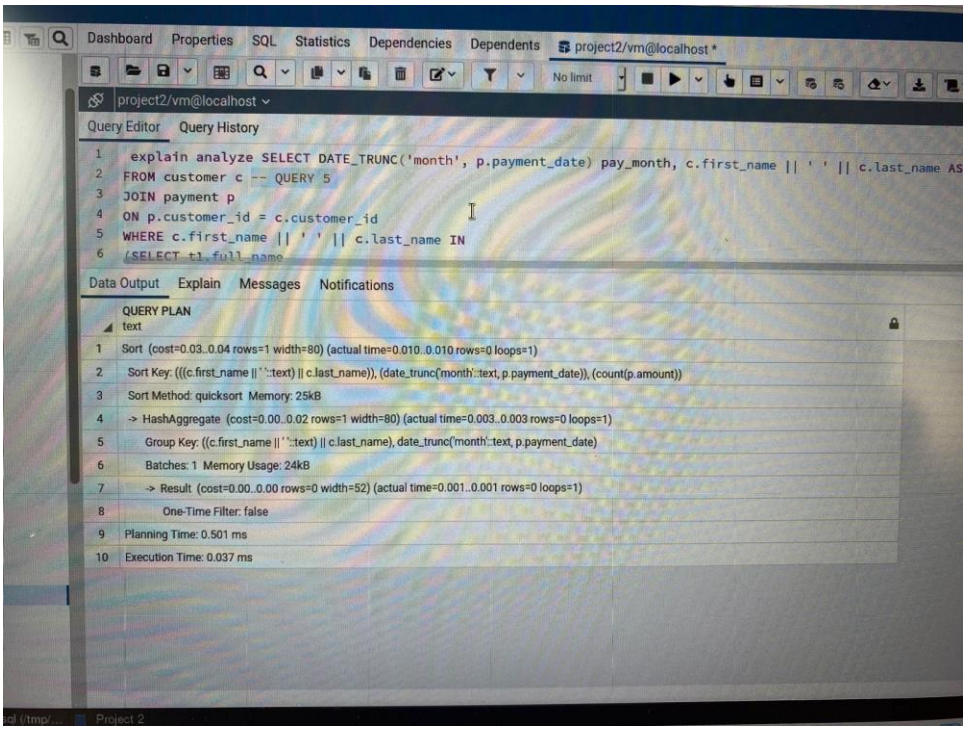
Query (4) Justification:

- Index On Attribute: Table-Name = category, Attribute-Name = name, we used a Hash index as we had a simple equality on name attribute in table category.
- We turned off the seq scan to make the planner use our index.
- The planner didn't use our index even after disabling the seq scan, as planner sees index scan on category using category primary key is a better option for optimizing the query than using our created index.

Query 5:

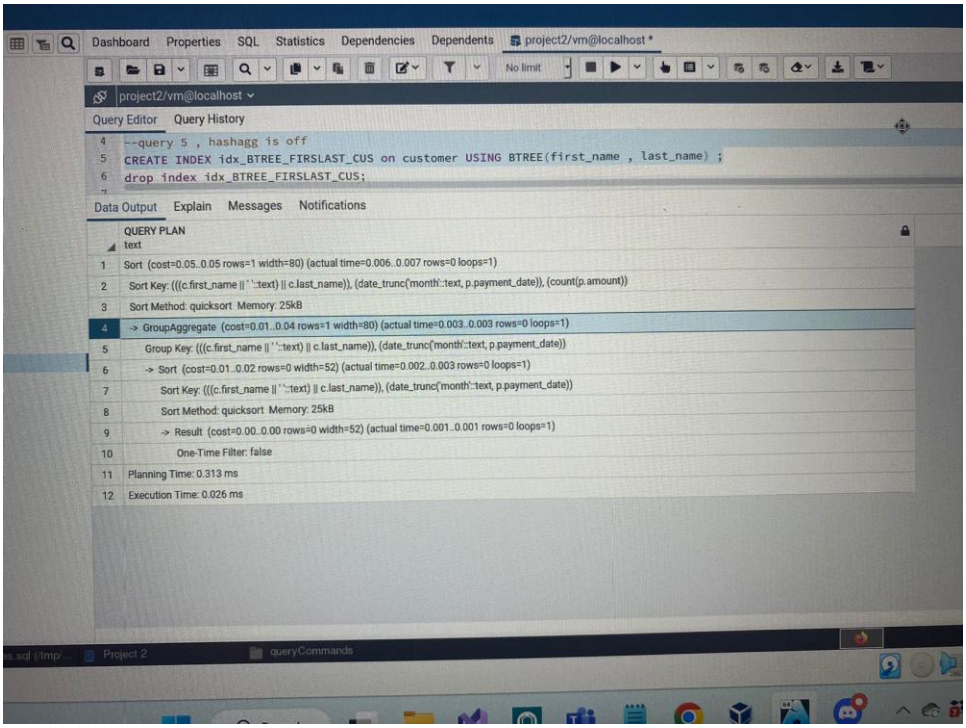
Before Query 5:

- **Highest Cost:** HashAggregate
- **Largest Number of Rows:** HashAggregate
- **Planning Time:** 0.501
- **Execution Time:** 0.037



After Query 5:

- **Highest Cost:** Group Aggregate
- **Slowest Runtime:** sort
- **Largest Number of Rows:** sort
- **Planning Time:** 0.313
- **Execution Time:** 0.026



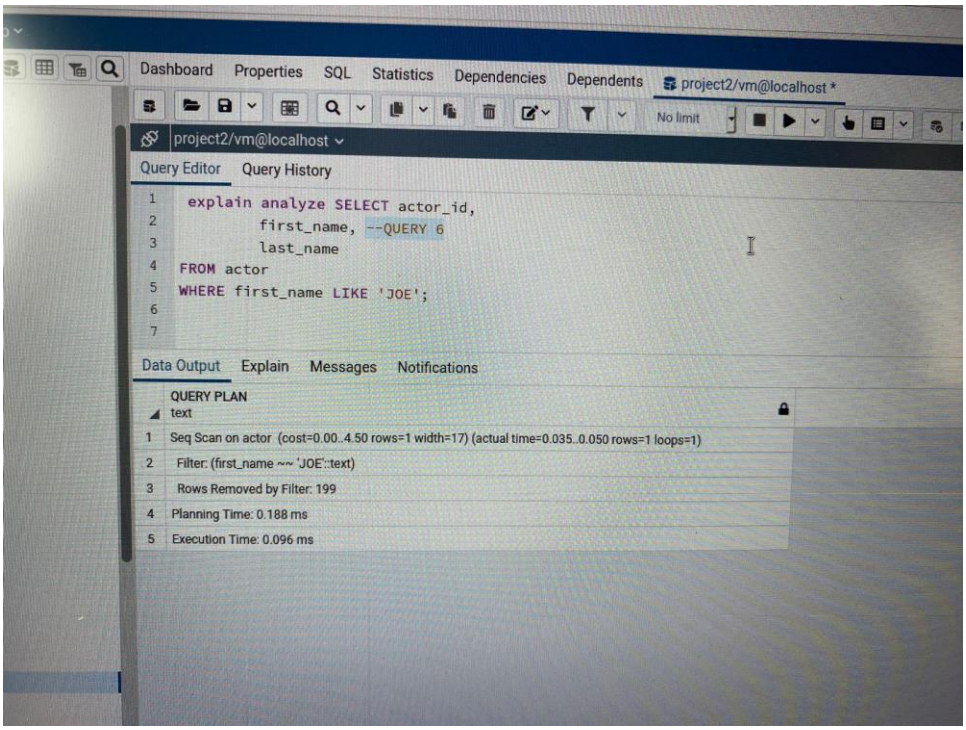
Query (5) Justification:

- Index On Attribute: Table-Name = customer, Attribute-Name = first_name, last_name , we used a Btree index as the where clause was searching through a large range of values.
- We turned off the hash_agg to make the planner use our index.
- The planner didn't use our index even after disabling the hash aggregate, as planner sees group aggregate a better option for optimizing the query than using our created index which results in better planning and execution times.

Query 6:

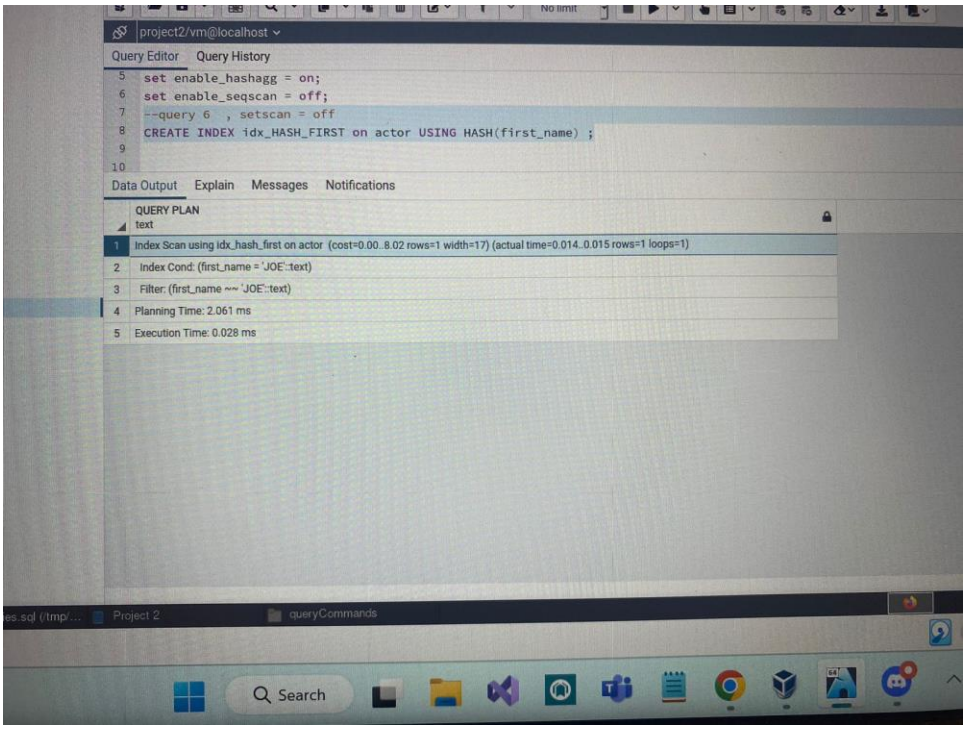
Before Query 6:

- **Highest Cost:** Seqscan
- **Largest Number of Rows:** Seqscan
- **Planning Time:** 0.188
- **Execution Time:** 0.096



After Query 6:

- **Highest Cost:** Index scan using idx_hash_first on actor
- **Slowest Runtime:** Index scan using idx_hash_first on actor
- **Largest Number of Rows:** Index scan using idx_hash_first on actor
- **Planning Time:** 2.061
- **Execution Time:** 0.028



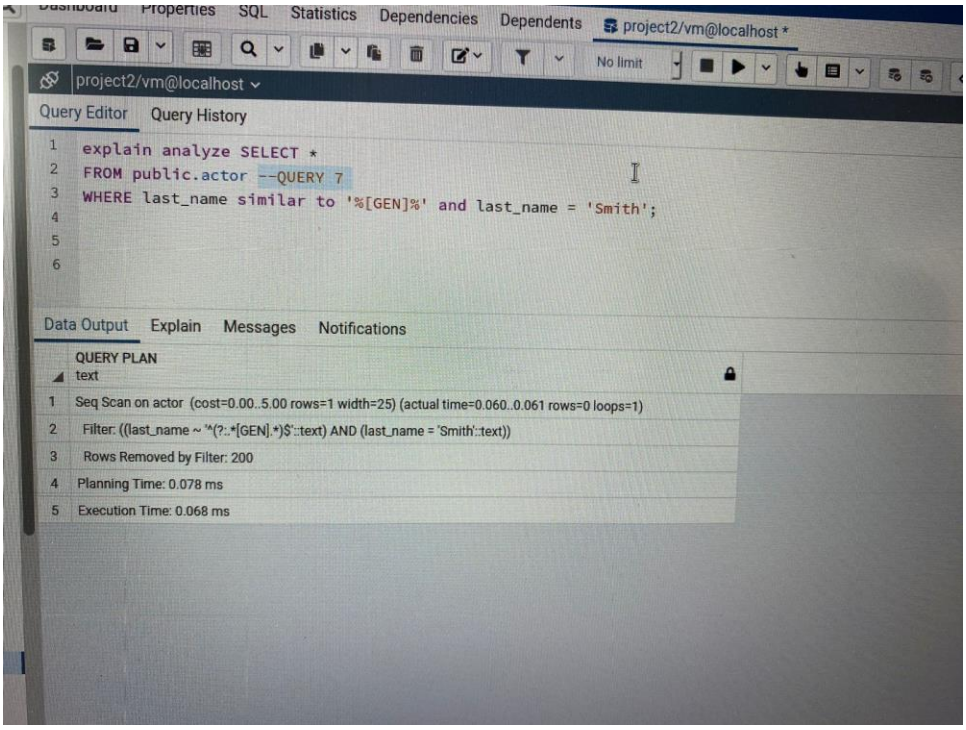
Query (6) Justification:

- Index On Attribute: Table-Name = actor, Attribute-Name = first_name, we used a hash index as the where clause was a simple equality on first_name.
- We turned off the seq_scan to make the planner use our index.
- The planner uses our index which results in higher costs and planning time, as the planner was already using the plan that results in the most optimized cost and planning time.

Query 7:

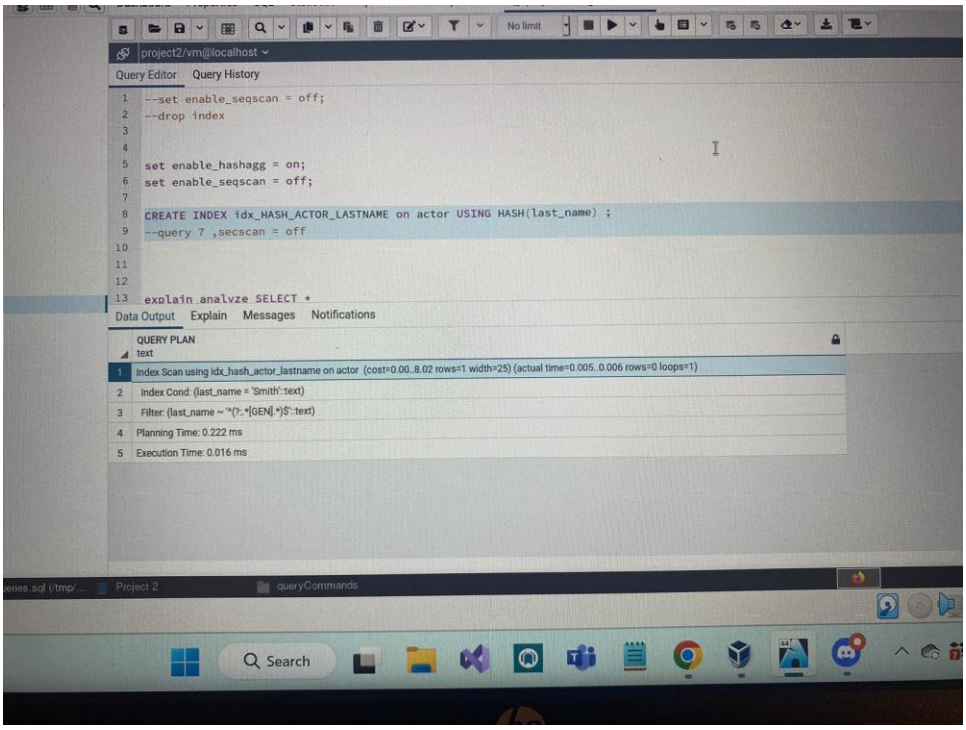
Before Query 7:

- **Highest Cost:** Seq scan on actor
- **Slowest Runtime:** Seq scan on actor
- **Largest Number of Rows:** Seq scan on actor
- **Planning Time:** 0.078
- **Execution Time:** 0.068



After Query 7:

- **Highest Cost:** Index scan using idx_hash_actor_lastname
- **Slowest Runtime:** Index scan using idx_hash_actor_lastname
- **Largest Number of Rows:** Index scan using idx_hash_actor_lastname
- **Planning Time:** 0.222
- **Execution Time:** 0.016



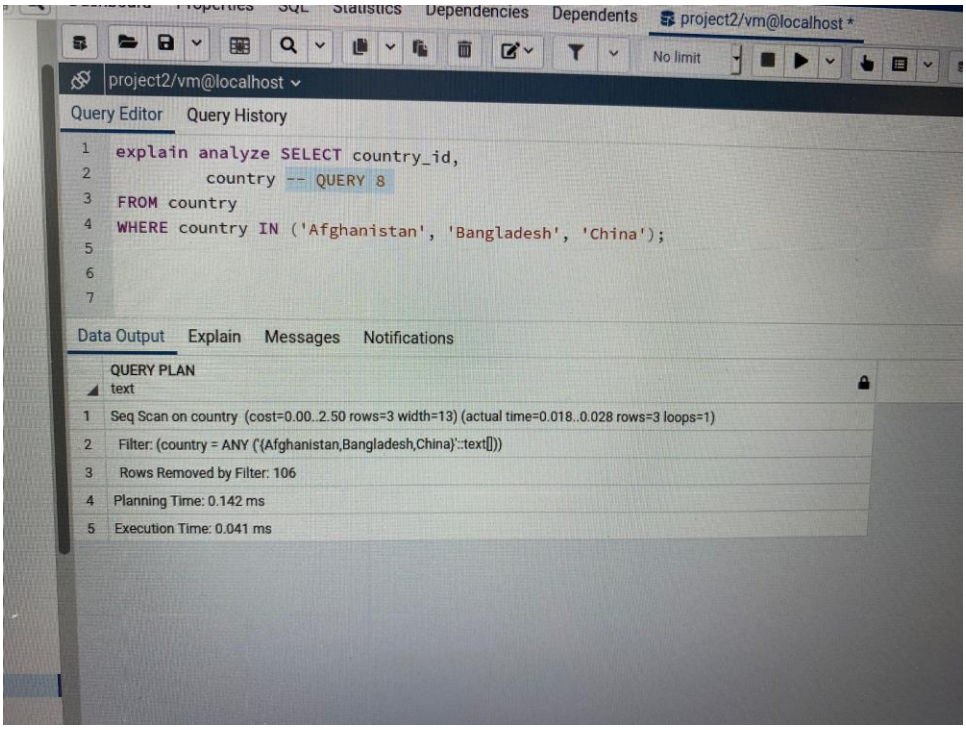
Query (7) Justification:

- Index On Attribute: Table-Name = actor, Attribute-Name = last_name we used a hash index as the where clause was a simple equality on last_name.
- We turned off the seq_scan to make the planner use our index.
- The planner uses our index which results in higher costs and planning time, as the planner was already using the plan that results in the most optimized cost and planning time.

Query 8:

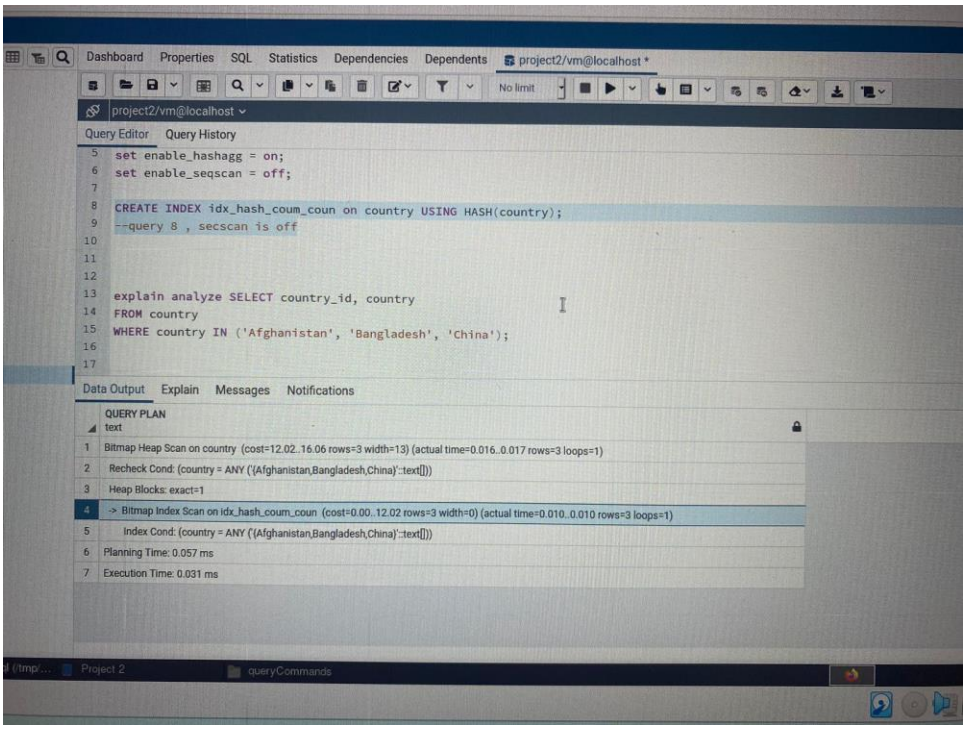
Before Query 8:

- **Highest Cost:** Seq scan on country
- **Slowest Runtime:** Seq scan on country
- **Largest Number of Rows:** Seq scan on country
- **Planning Time:** 0.142
- **Execution Time:** 0.041



After Query 8:

- **Highest Cost:** Bitmap Index scan on idx_hash_coum_coun
- **Slowest Runtime:** Bitmap Index scan on idx_hash_coum_coun
- **Largest Number of Rows:** bitmap heap scan on country
- **Planning Time:** 0.057
- **Execution Time:** 0.031



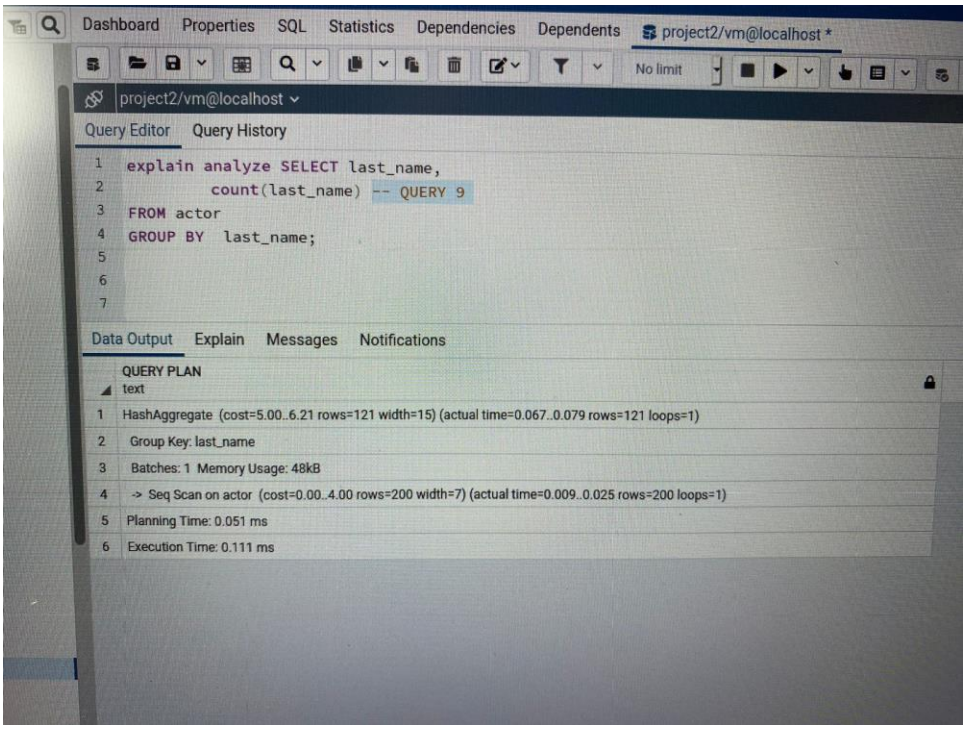
Query (8) Justification:

- Index On Attribute: Table-Name = country, Attribute-Name = country we used a hash index as the where clause was a simple equality on country.
- We turned off the seq_scan to make the planner use our index.
- The planner uses our index which results better planning and execution times.

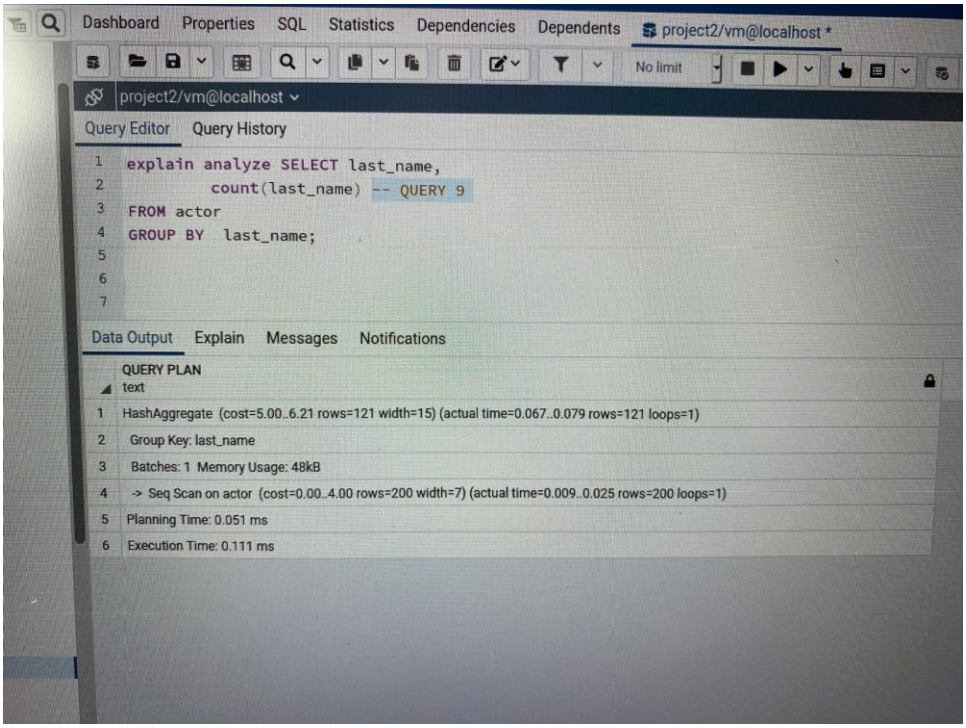
Query 9:

Before Query 9:

- **Highest Cost:** Seq scan on actor
- **Slowest Runtime:** Seq scan on actor
- **Largest Number of Rows:** Seq scan on actor
- **Planning Time:** 0.051
- **Execution Time:** 0.111



After Query 9:



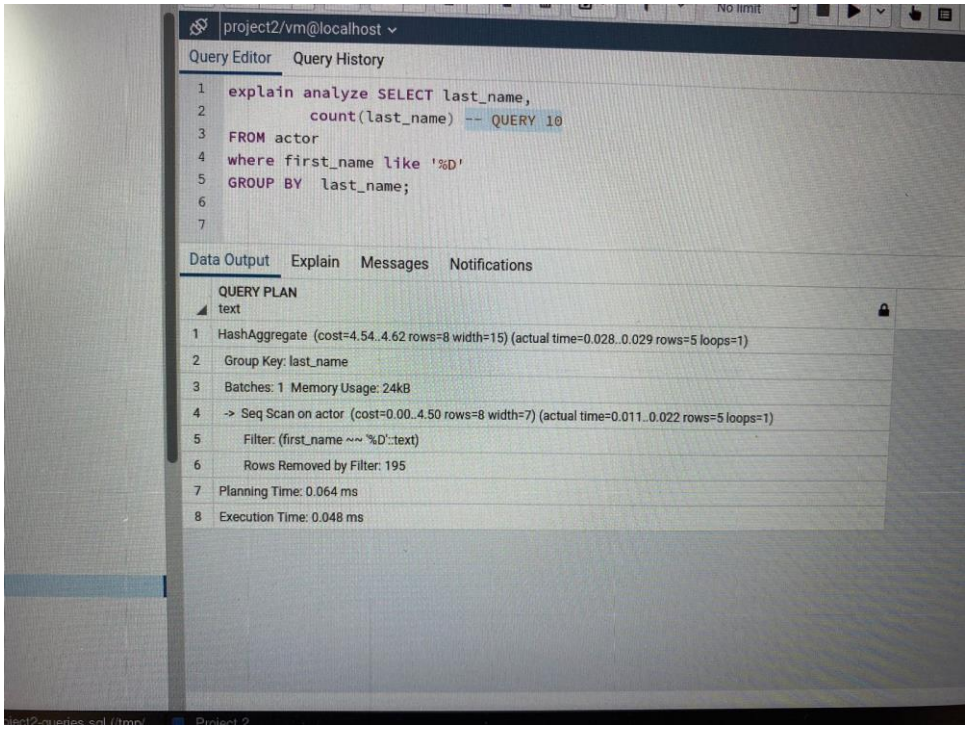
Query (9) Justification:

We did not create an index on this query as we found that by using different the planning and execution time got worse , moreover there was no clear attribute to apply an index on.

Query 10:

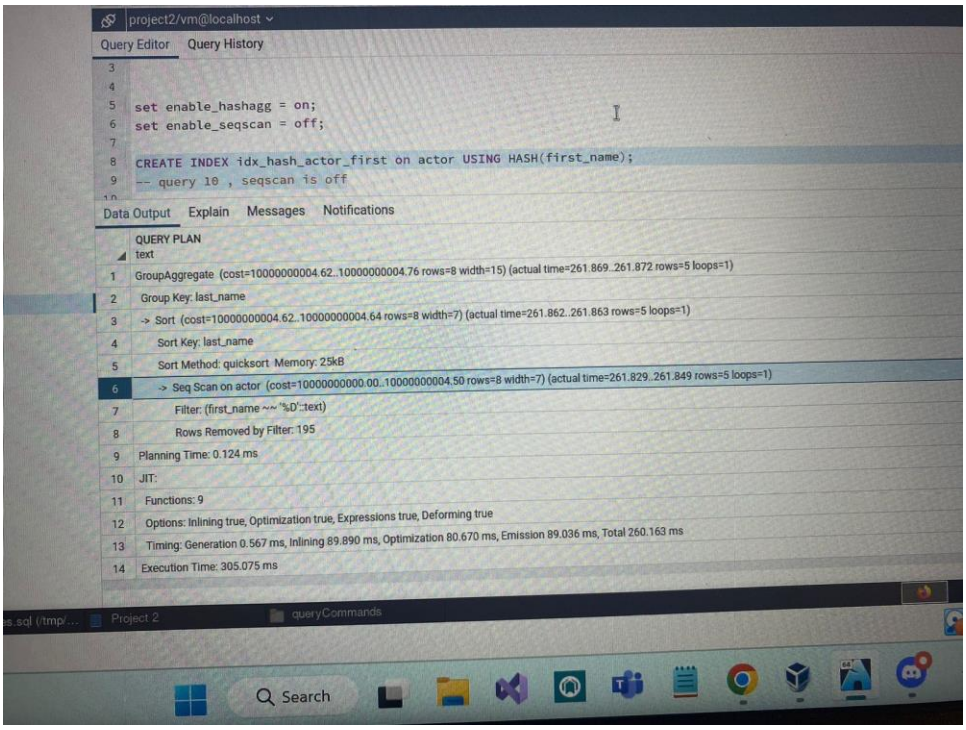
Before Query 10:

- **Highest Cost:** Seq scan on actor
- **Slowest Runtime:** Seq scan on actor
- **Largest Number of Rows:** hash aggregate and seq scan on actor
- **Planning Time:** 0.064
- **Execution Time:** 0.048



After Query 10:

- **Highest Cost:** sec scan on actor
- **Slowest Runtime:** seq scan on actor
- **Largest Number of Rows:** seq scan and group aggregate
- **Planning Time:** 0.124
- **Execution Time:** 305.075



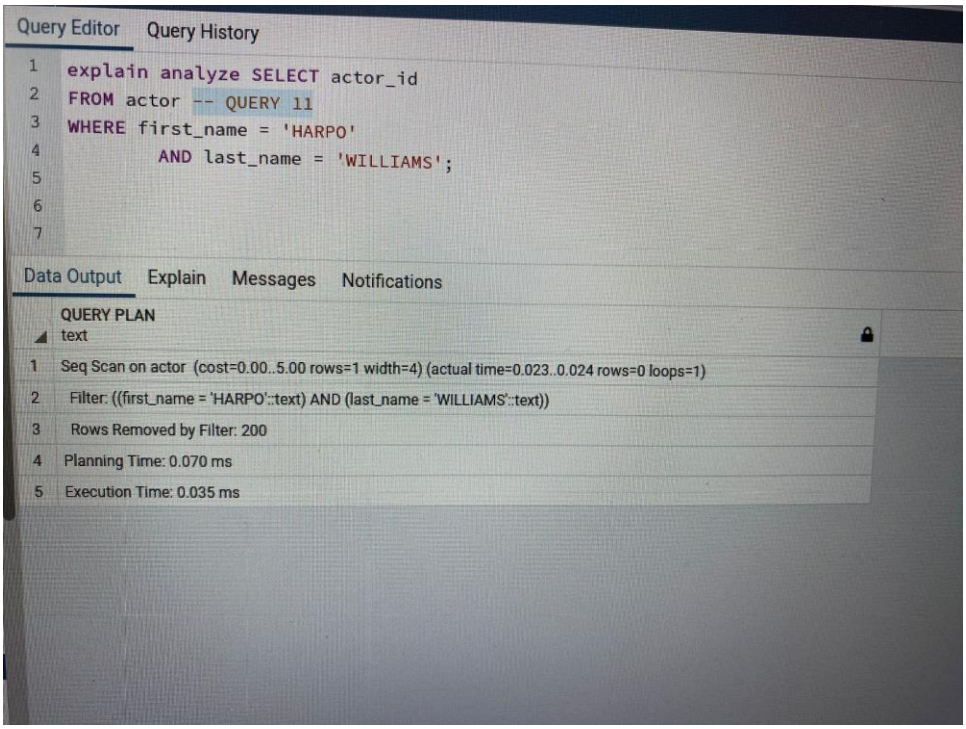
Query (10) Justification:

- Index On Attribute: Table-Name = actor, Attribute-Name = first_name, we used a Hash index as we were looking for first name that ends with “D” which is considered to be an equality where hash index is best suit for.
- We turned off the seq_scan to make the planner use our index.
- The planner is still using ‘seq scan’ on actor as he sees that using seq scan is the best option here, but he changed the plan a bit by using Group aggregate and sort instead of using hash aggregate.

Query 11:

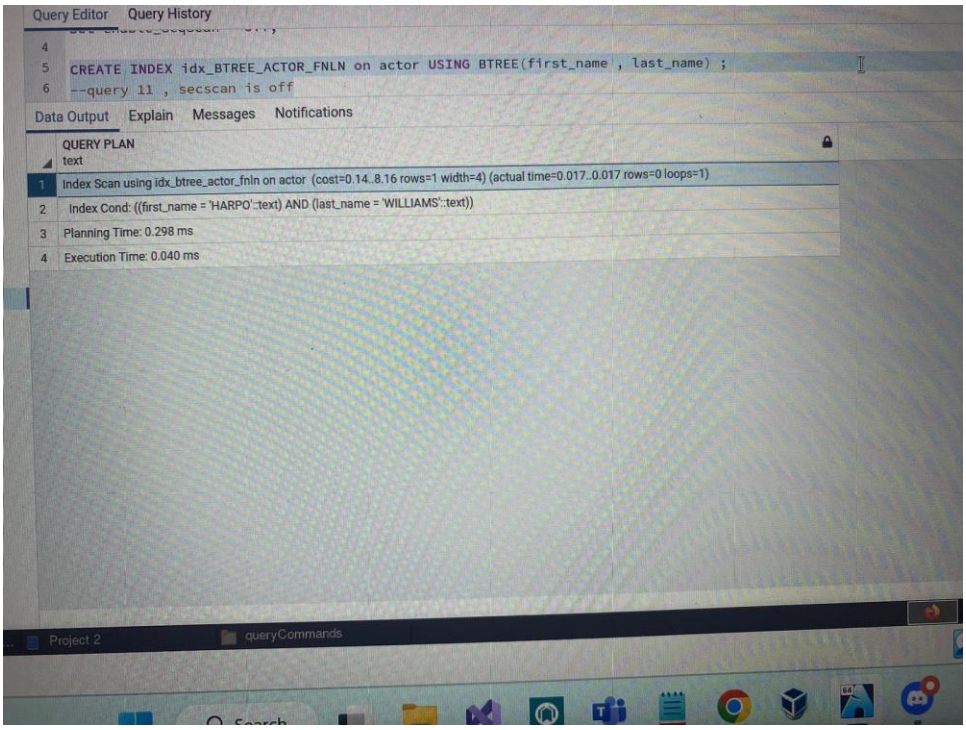
Before Query 11:

- **Highest Cost:** Seq scan on actor
- **Slowest Runtime:** Seq scan on actor
- **Largest Number of Rows:** seq scan on actor
- **Planning Time:** 0.070
- **Execution Time:** 0.035



After Query 11:

- **Highest Cost:** index scan on actor
- **Slowest Runtime:** index scan on actor
- **Largest Number of Rows:** index scan on actor
- **Planning Time:** 0.298
- **Execution Time:** 0.040



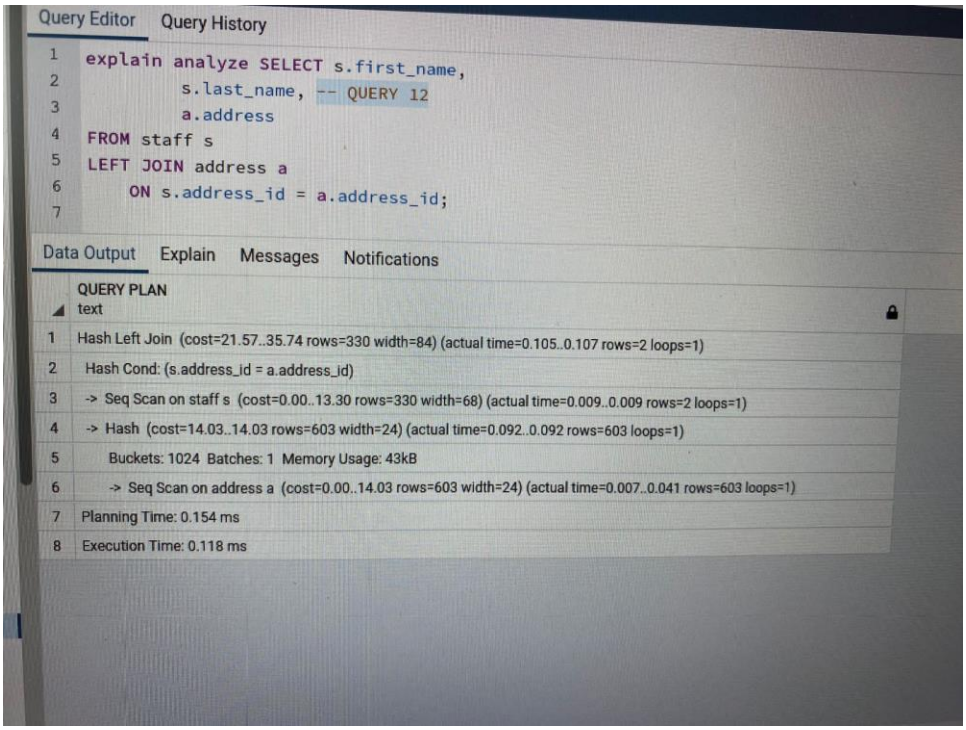
Query (11) Justification:

- Index On Attribute: Table-Name = actor, Attribute-Name = first_name and last_name, we used a Btree index as we're filtering our table based on a specific condition on firstname and lastname, so using a btree index will help us creating an index using 2 columns.
- We turned off the seq_scan to make the planner use our index.
- The planner uses our index which results in higher costs, planning and execution times, as the planner was already using the plan that results in the most optimized costs, planning and execution times.

Query 12:

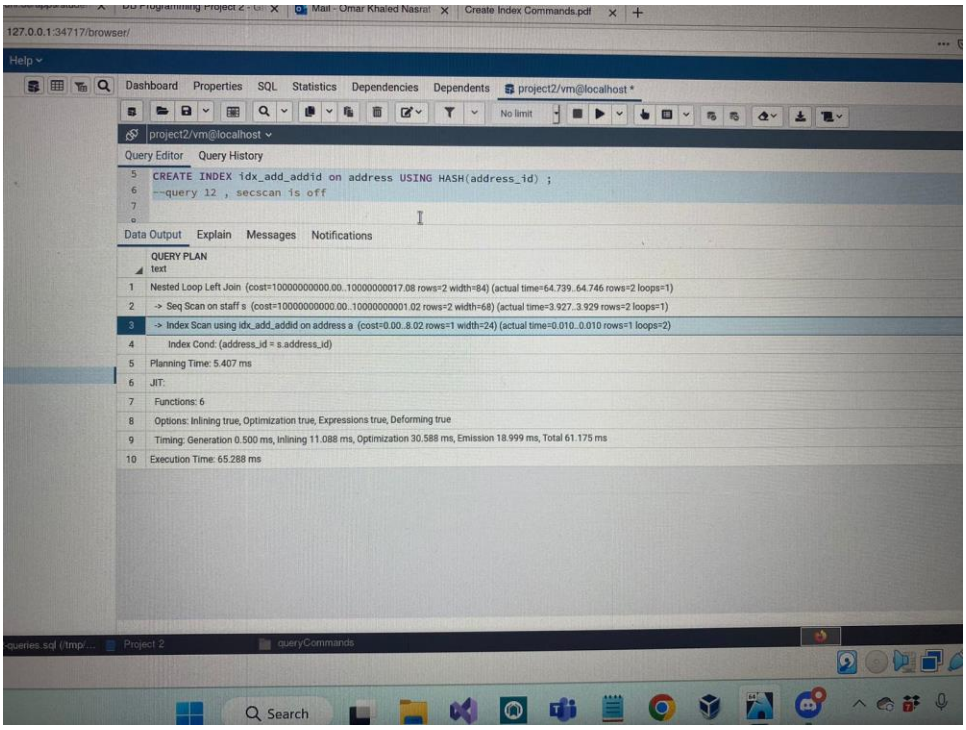
Before Query 12:

- **Highest Cost:** hash left join
- **Slowest Runtime:** Seq scan on address a
- **Largest Number of Rows:** hash and seq scan on address a
- **Planning Time:** 0.154
- **Execution Time:** 0.118



After Query 12:

- **Highest Cost:** Nested loop left join
- **Slowest Runtime:** Nested loop left join
- **Largest Number of Rows:** Nested loop left join and Seq Scan on staff s
- **Planning Time:** 5.407
- **Execution Time:** 65.288



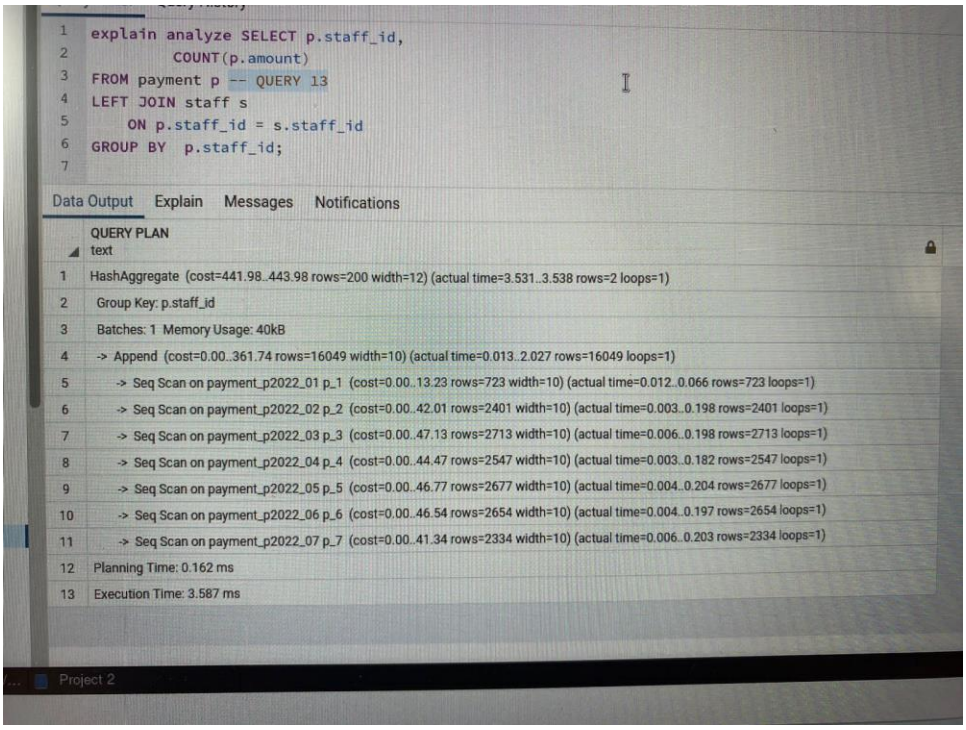
Query (12) Justification:

- Index On Attribute: Table-Name = address, Attribute-Name = address_id, we used hash index as we're creating an index on the foreign key so, we're filtering based on the equality of the IDs.
- We turned off the seq_scan to make the planner use our index.
- The planner uses our index which results in higher costs, planning and execution times, as the planner was already using the plan that results in the most optimized costs, planning and execution times.

Query 13:

Before Query 13:

- **Highest Cost:** Append
- **Slowest Runtime:** Append
- **Largest Number of Rows:** Append
- **Planning Time:** 0.162
- **Execution Time:** 3.587



```
1  explain analyze SELECT p.staff_id,
2           COUNT(p.amount)
3  FROM payment p -- QUERY 13
4  LEFT JOIN staff s
5      ON p.staff_id = s.staff_id
6  GROUP BY  p.staff_id;
7
```

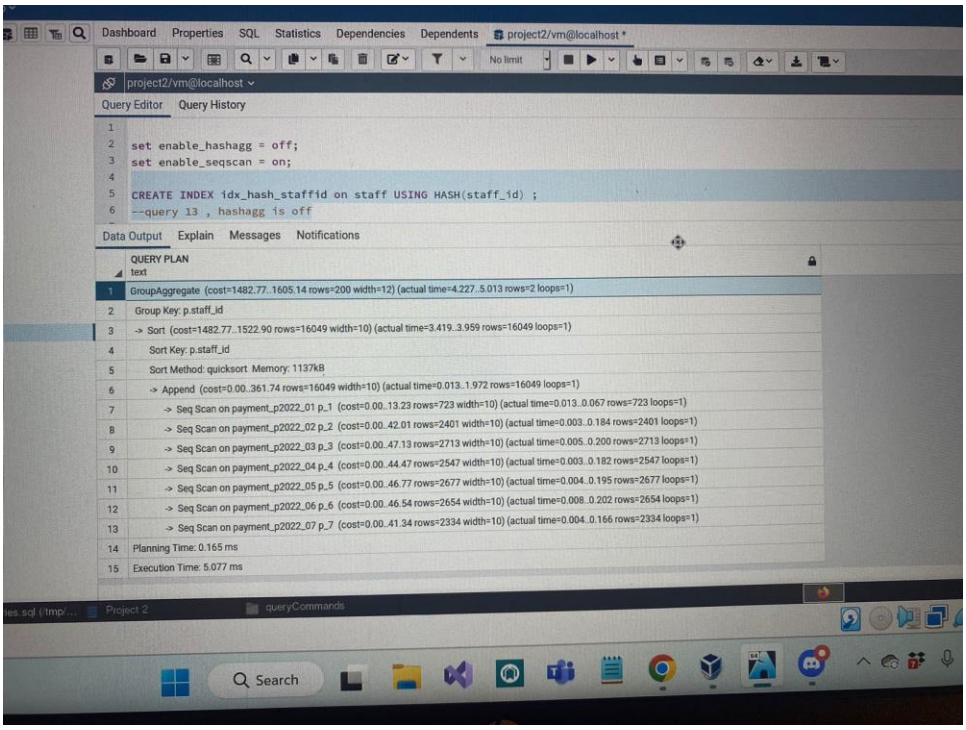
Data Output | Explain | Messages | Notifications

	QUERY PLAN text
1	HashAggregate (cost=441.98..443.98 rows=200 width=12) (actual time=3.531..3.538 rows=2 loops=1)
2	Group Key: p.staff_id
3	Batches: 1 Memory Usage: 40kB
4	-> Append (cost=0.00..361.74 rows=16049 width=10) (actual time=0.013..2.027 rows=16049 loops=1)
5	-> Seq Scan on payment_p2022_01 p_1 (cost=0.00..13.23 rows=723 width=10) (actual time=0.012..0.066 rows=723 loops=1)
6	-> Seq Scan on payment_p2022_02 p_2 (cost=0.00..42.01 rows=2401 width=10) (actual time=0.003..0.198 rows=2401 loops=1)
7	-> Seq Scan on payment_p2022_03 p_3 (cost=0.00..47.13 rows=2713 width=10) (actual time=0.006..0.198 rows=2713 loops=1)
8	-> Seq Scan on payment_p2022_04 p_4 (cost=0.00..44.47 rows=2547 width=10) (actual time=0.003..0.182 rows=2547 loops=1)
9	-> Seq Scan on payment_p2022_05 p_5 (cost=0.00..46.77 rows=2677 width=10) (actual time=0.004..0.204 rows=2677 loops=1)
10	-> Seq Scan on payment_p2022_06 p_6 (cost=0.00..46.54 rows=2654 width=10) (actual time=0.004..0.197 rows=2654 loops=1)
11	-> Seq Scan on payment_p2022_07 p_7 (cost=0.00..41.34 rows=2334 width=10) (actual time=0.006..0.203 rows=2334 loops=1)
12	Planning Time: 0.162 ms
13	Execution Time: 3.587 ms

Project 2

After Query 13:

- **Highest Cost:** Append
- **Slowest Runtime:** Append
- **Largest Number of Rows:** Sort and Append
- **Planning Time:** 0.165
- **Execution Time:** 5.077



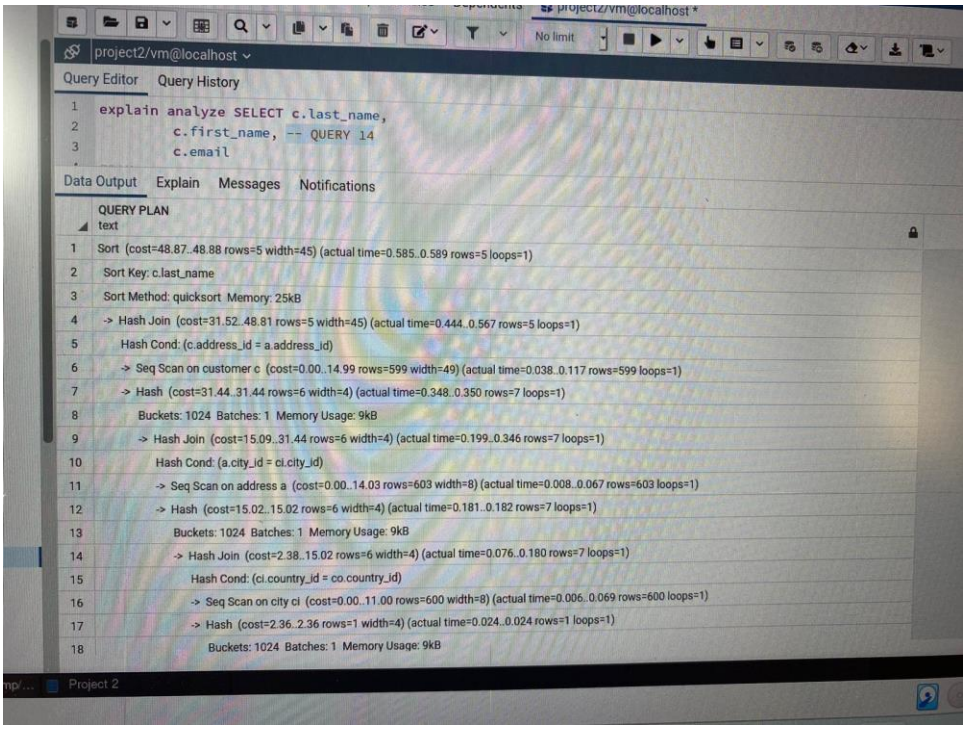
Query (13) Justification:

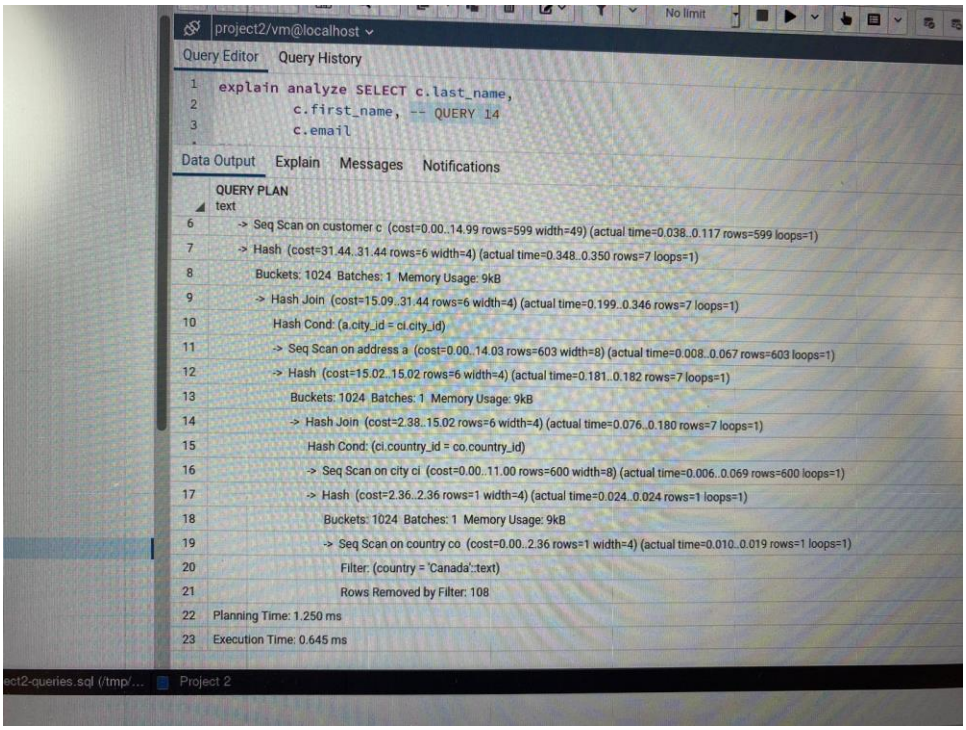
- Index On Attribute: Table-Name = staff, Attribute-Name = staff_id, we used hash index as we're creating an index on the foreign key so, we're filtering based on the equality of the IDs.
- We turned off the hash aggregate to make the planner use our index.
- The planner is still using 'seq scan' on payment , but he changed the plan a bit by using Group aggregate and sort instead of using hash aggregate, as the planner sees that using this plan is a better option than using the index we've created.

Query 14:

Before Query 14:

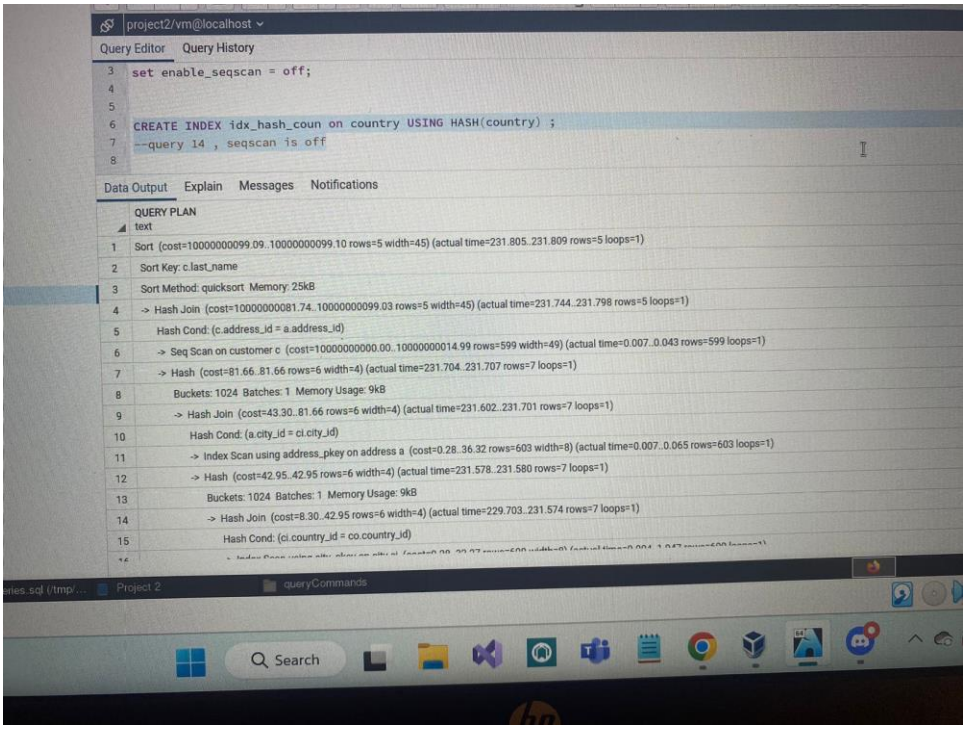
- **Highest Cost:** Hash join (cond. : c.address_id = a.address_id)
- **Slowest Runtime:** Hash join (cond. : a.city_id = ci.city_id)
- **Largest Number of Rows:** seq scan on address a
- **Planning Time:** 1.250
- **Execution Time:** 0.645

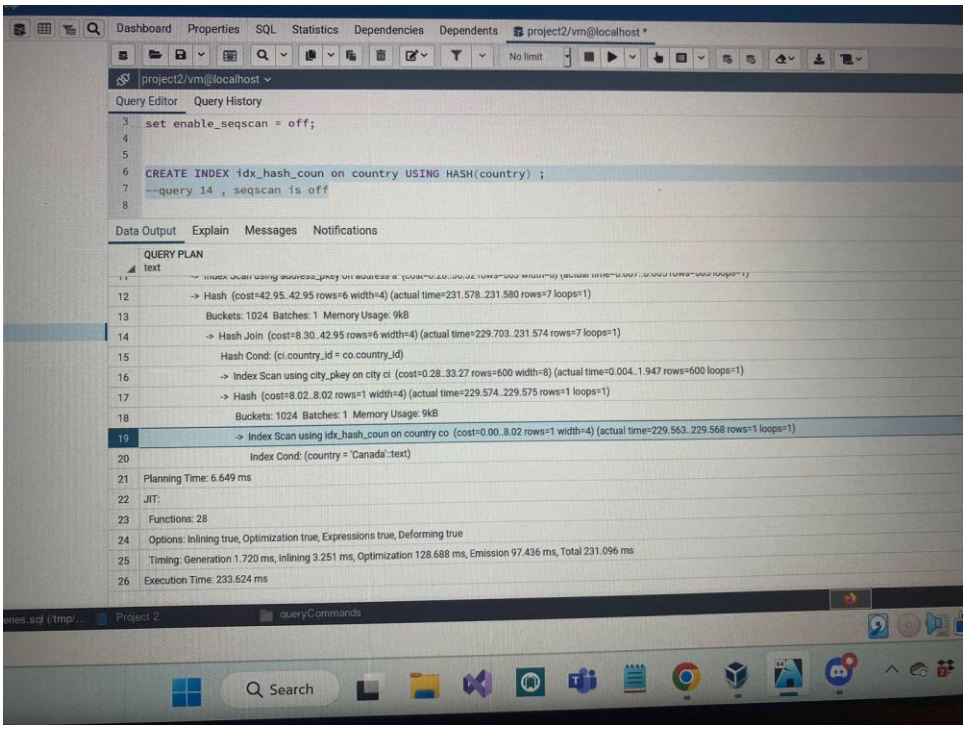




After Query 14:

- **Highest Cost:** Hash join (cond. : a.city_id = ci.city_id)
- **Slowest Runtime:** Hash join (cond. : ci.country_id = co.country_id)
- **Largest Number of Rows:** index scan using address_pkey on address a
- **Planning Time:** 6.649
- **Execution Time:** 223.624





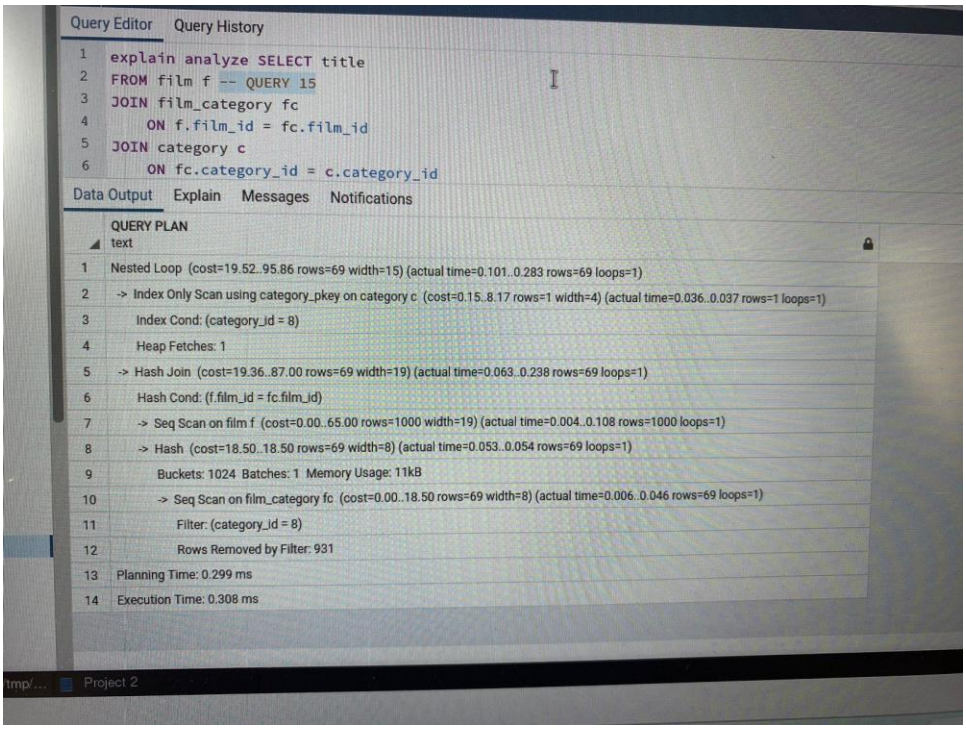
Query (14) Justification:

- Index On Attribute: Table-Name = country, Attribute-Name = country, we used a hash index as the where clause was a simple equality on country.
- We turned off the seq_scan to make the planner use our index.
- The planner uses our index which results in higher costs, planning and execution times, as the planner was already using the plan that results in the most optimized costs, planning and execution times.

Query 15:

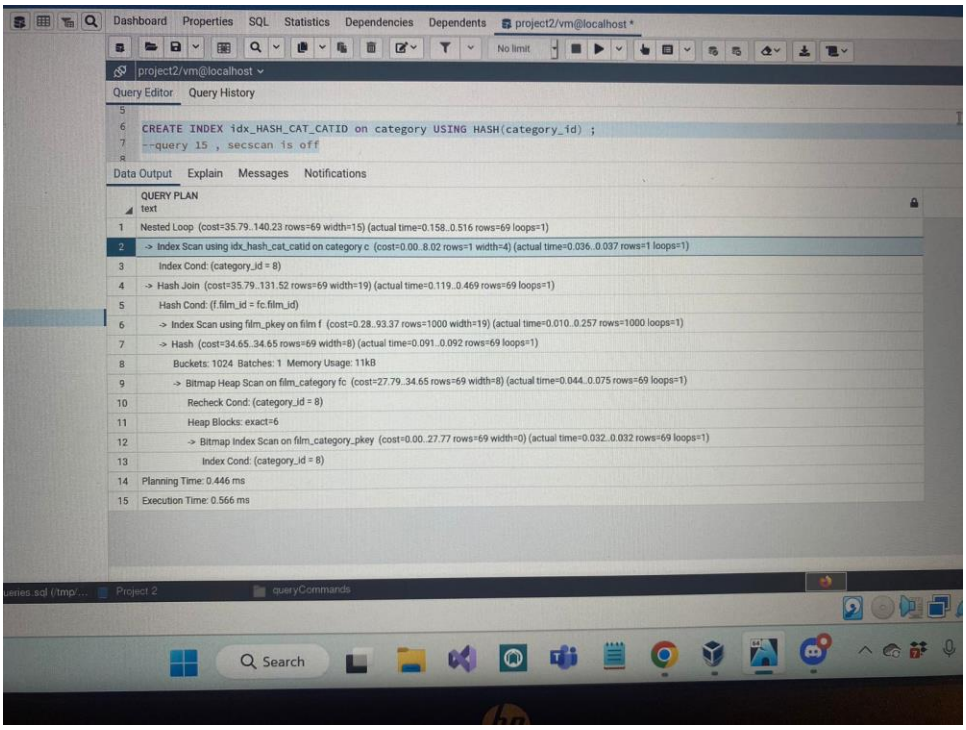
Before Query 15:

- **Highest Cost:** Nested Loop
- **Slowest Runtime:** Nested Loop
- **Largest Number of Rows:** seq scan on film f
- **Planning Time:** 0.299
- **Execution Time:** 0.308



After Query 15:

- **Highest Cost:** Nested Loop
- **Slowest Runtime:** Nested Loop
- **Largest Number of Rows:** Index scan using film_pkey on film f
- **Planning Time:** 0.446
- **Execution Time:** 0.566



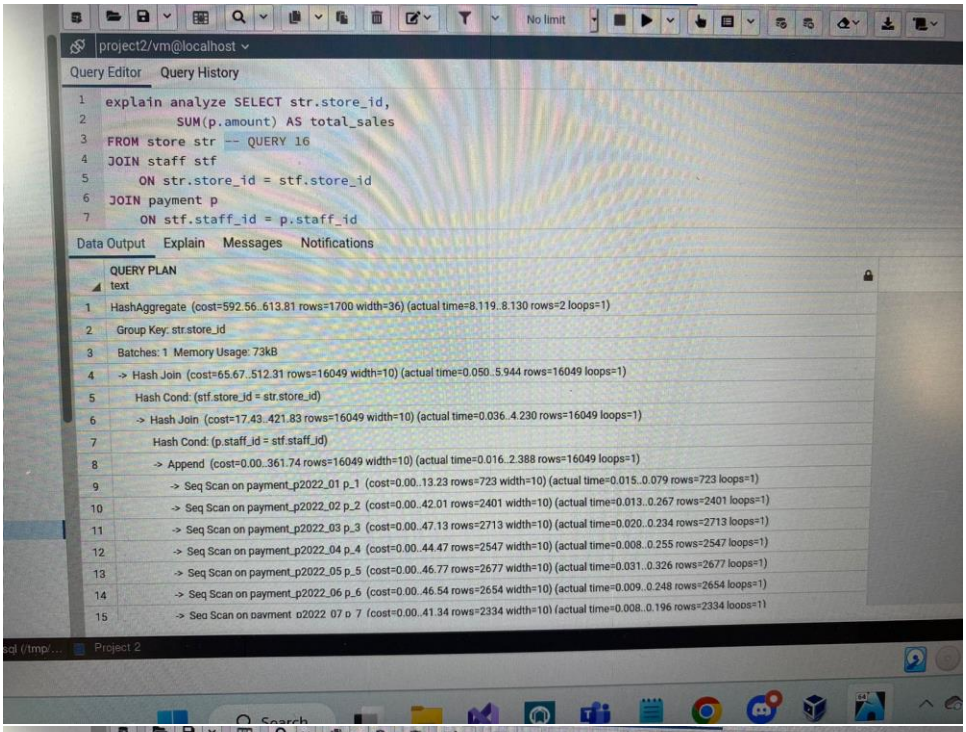
Query (15) Justification:

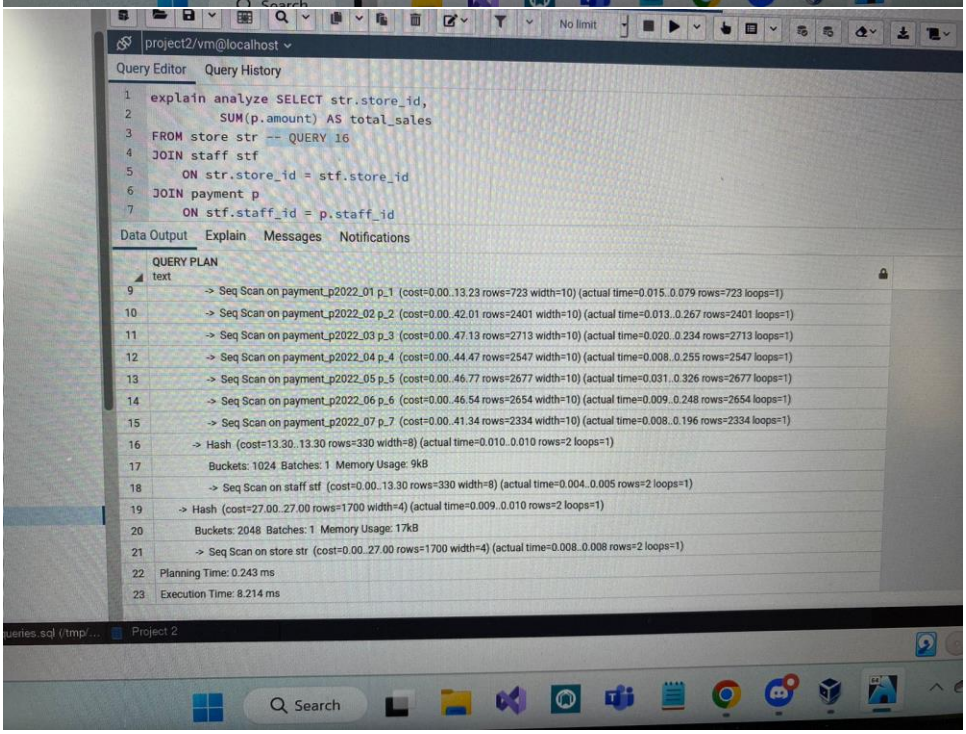
- Index On Attribute: Table-Name = category, Attribute-Name = category_id, we used hash index as we're creating an index on the foreign key so, we're filtering based on the equality of the IDs.
- We turned off the seq_scan to make the planner use our index.
- The planner uses our index which results in higher costs, planning and execution times, as the planner was already using the plan that results in the most optimized costs, planning and execution times.

Query 16:

Before Query 16:

- **Highest Cost:** Hash join (cond. : stf.store_id = str.store_id)
- **Slowest Runtime:** Hash join (cond. : stf.store_id = str.store_id)
- **Largest Number of Rows:** Hash join (cond. : stf.store_id = str.store_id) and Hash join (cond. : p.staff_id = stf.staff_id)
- **Planning Time:** 0.243
- **Execution Time:** 8.214

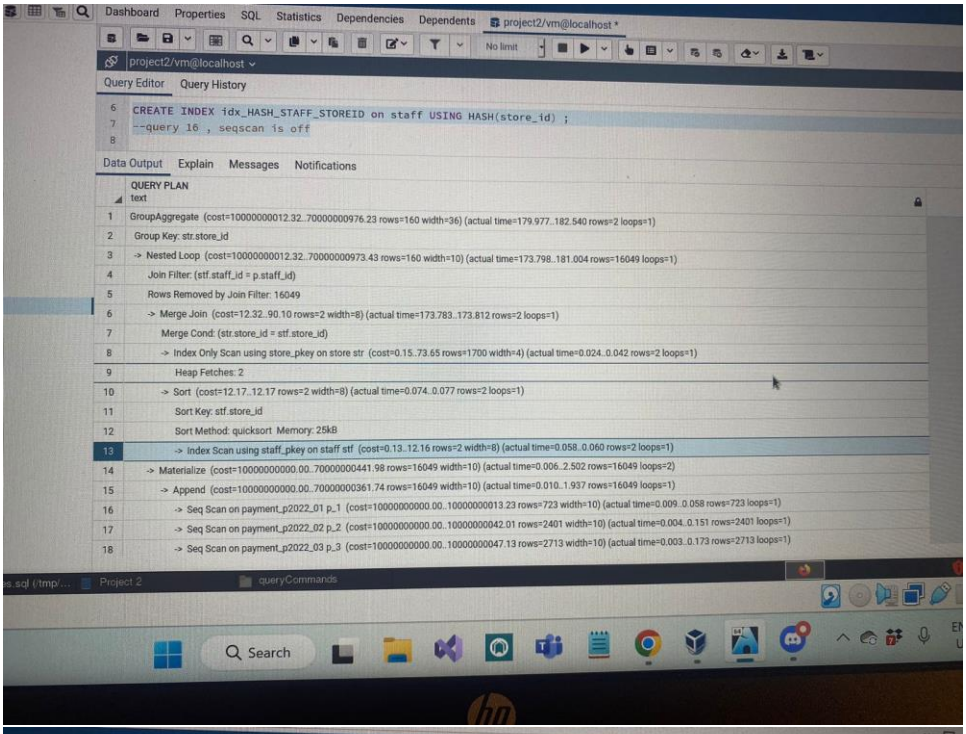


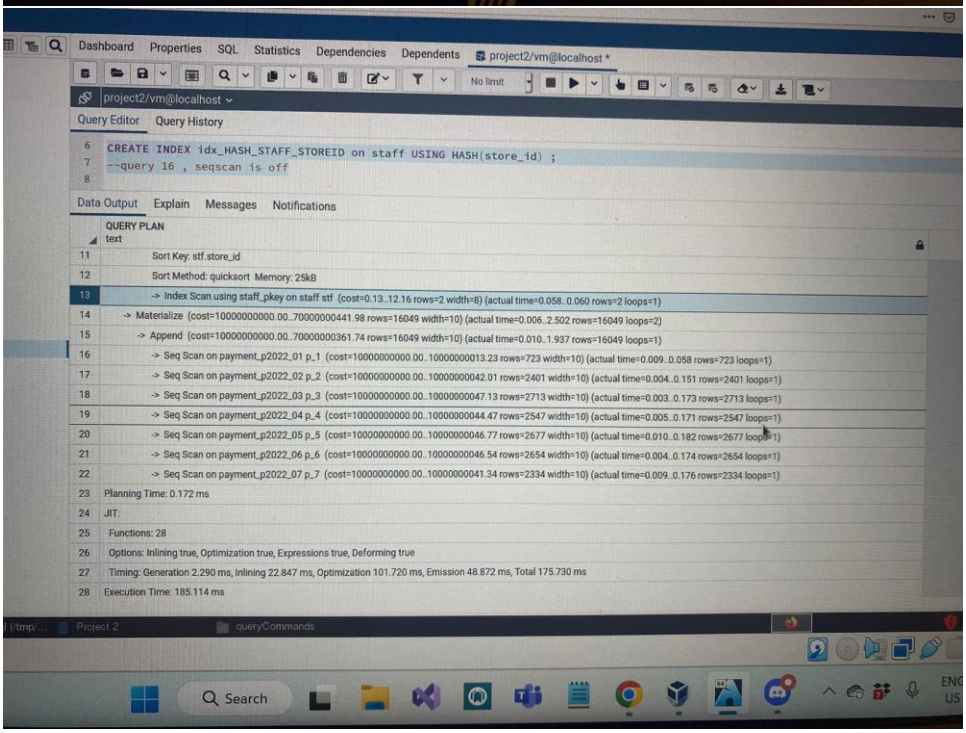


After Query 16:

- **Highest Cost:** materialize

- **Slowest Runtime:** Nested Loop
- **Largest Number of Rows:** materialize, append and Nested Loop
- **Planning Time:** 0.172
- **Execution Time:** 185.114





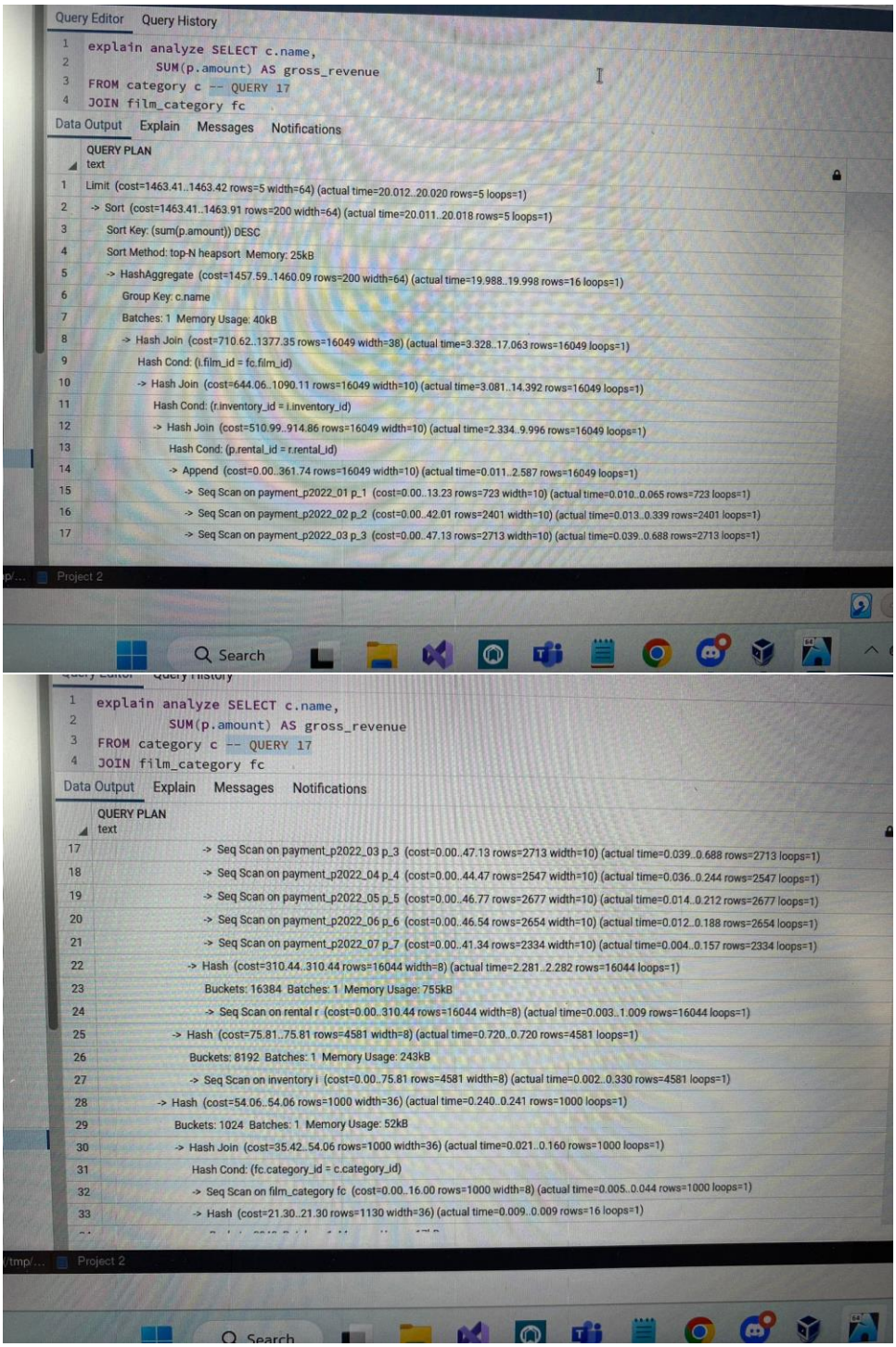
Query (16) Justification:

- Index On Attribute: Table-Name = staff, Attribute-Name = store_id, we used hash index as we're creating an index on the foreign key so, we're filtering based on the equality of the IDs.
- We turned off the seq_scan to make the planner use our index.
- The planner didn't use our index even after disabling the seq scan, as planner sees an index scan using the primary key is a better option for optimizing the query than our created index.

Query 17:

Before Query 17:

- **Highest Cost:** Hash Join (cond. : I.film_id = fc.film_id)
- **Slowest Runtime:** Hash Join (cond. : I.film_id = fc.film_id)
- **Largest Number of Rows:** Hash Join (cond. : I.film_id = fc.film_id), Hash Join (cond. : r.inventory_id = i.inventory_id , Hash Join (cond. : p.rental_id = r.rental_id) and append
- **Planning Time:** 0.504
- **Execution Time:** 20.071



Query Editor | Query History

```
1  explain analyze SELECT c.name,
2          SUM(p.amount) AS gross_revenue
3  FROM category c -- QUERY 17
4  JOIN film_category fc
```

Data Output | Explain | Messages | Notifications

```
QUERY PLAN
text
1  Limit  (cost=1463.41..1463.42 rows=5 width=64) (actual time=20.012..20.020 rows=5 loops=1)
2    -> Sort  (cost=1463.41..1463.91 rows=200 width=64) (actual time=20.011..20.018 rows=5 loops=1)
3      Sort Key: (sum(p.amount)) DESC
4      Sort Method: top-N heapsort  Memory: 25kB
5      -> HashAggregate  (cost=1457.59..1460.09 rows=200 width=64) (actual time=19.988..19.998 rows=16 loops=1)
6        Group Key: c.name
7        Batches: 1  Memory Usage: 40kB
8        -> Hash Join  (cost=710.62..1377.35 rows=16049 width=38) (actual time=3.328..17.063 rows=16049 loops=1)
9          Hash Cond: (i.film_id = fc.film_id)
10         -> Hash Join  (cost=644.06..1090.11 rows=16049 width=10) (actual time=3.081..14.392 rows=16049 loops=1)
11           Hash Cond: (r.inventory_id = i.inventory_id)
12           -> Hash Join  (cost=510.99..914.86 rows=16049 width=10) (actual time=2.334..9.996 rows=16049 loops=1)
13             Hash Cond: (p.rental_id = r.rental_id)
14             -> Append  (cost=0.00..361.74 rows=16049 width=10) (actual time=0.011..2.587 rows=16049 loops=1)
15               -> Seq Scan on payment_p2022_01 p_1  (cost=0.00..13.23 rows=723 width=10) (actual time=0.010..0.065 rows=723 loops=1)
16               -> Seq Scan on payment_p2022_02 p_2  (cost=0.00..42.01 rows=2401 width=10) (actual time=0.013..0.339 rows=2401 loops=1)
17               -> Seq Scan on payment_p2022_03 p_3  (cost=0.00..47.13 rows=2713 width=10) (actual time=0.039..0.688 rows=2713 loops=1)
```

Project 2

Search

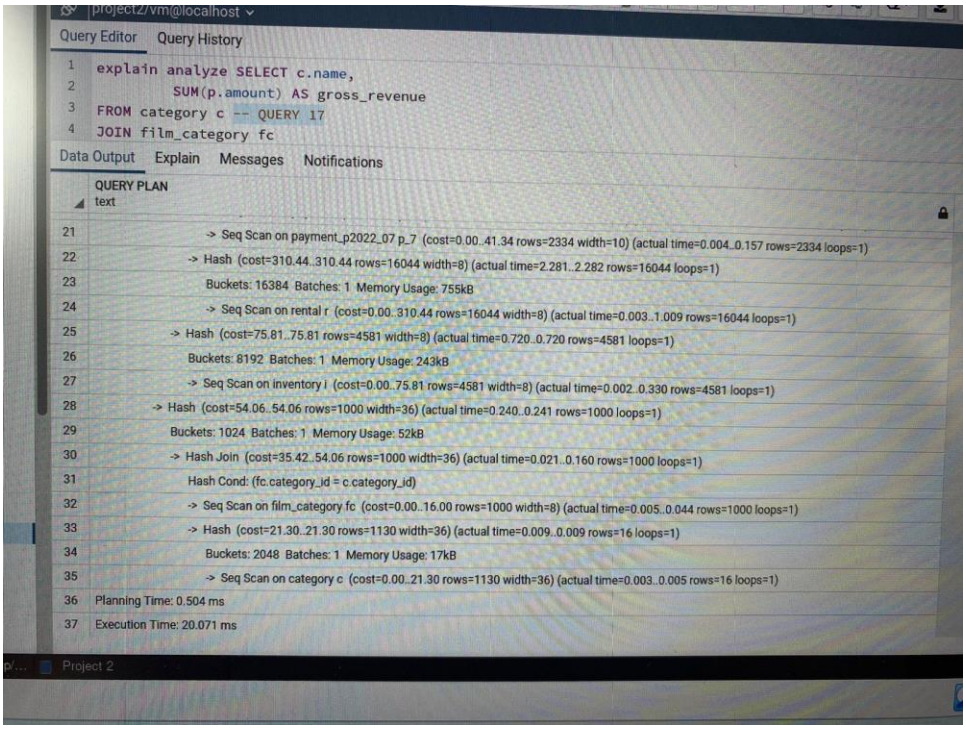

```
1  explain analyze SELECT c.name,
2          SUM(p.amount) AS gross_revenue
3  FROM category c -- QUERY 17
4  JOIN film_category fc
```

Data Output | Explain | Messages | Notifications

```
QUERY PLAN
text
17               -> Seq Scan on payment_p2022_03 p_3  (cost=0.00..47.13 rows=2713 width=10) (actual time=0.039..0.688 rows=2713 loops=1)
18               -> Seq Scan on payment_p2022_04 p_4  (cost=0.00..44.47 rows=2547 width=10) (actual time=0.036..0.244 rows=2547 loops=1)
19               -> Seq Scan on payment_p2022_05 p_5  (cost=0.00..46.77 rows=2677 width=10) (actual time=0.014..0.212 rows=2677 loops=1)
20               -> Seq Scan on payment_p2022_06 p_6  (cost=0.00..46.54 rows=2654 width=10) (actual time=0.012..0.188 rows=2654 loops=1)
21               -> Seq Scan on payment_p2022_07 p_7  (cost=0.00..41.34 rows=2334 width=10) (actual time=0.004..0.157 rows=2334 loops=1)
22             -> Hash  (cost=310.44..310.44 rows=16044 width=8) (actual time=2.281..2.282 rows=16044 loops=1)
23               Buckets: 16384  Batches: 1  Memory Usage: 755kB
24               -> Seq Scan on rental r  (cost=0.00..310.44 rows=16044 width=8) (actual time=0.003..1.009 rows=16044 loops=1)
25           -> Hash  (cost=75.81..75.81 rows=4581 width=8) (actual time=0.720..0.720 rows=4581 loops=1)
26             Buckets: 8192  Batches: 1  Memory Usage: 243kB
27             -> Seq Scan on inventory i  (cost=0.00..75.81 rows=4581 width=8) (actual time=0.002..0.330 rows=4581 loops=1)
28         -> Hash  (cost=54.06..54.06 rows=1000 width=36) (actual time=0.240..0.241 rows=1000 loops=1)
29           Buckets: 1024  Batches: 1  Memory Usage: 52kB
30           -> Hash Join  (cost=35.42..54.06 rows=1000 width=36) (actual time=0.021..0.160 rows=1000 loops=1)
31             Hash Cond: (fc.category_id = c.category_id)
32             -> Seq Scan on film_category fc  (cost=0.00..16.00 rows=1000 width=8) (actual time=0.005..0.044 rows=1000 loops=1)
33             -> Hash  (cost=21.30..21.30 rows=1130 width=36) (actual time=0.009..0.009 rows=16 loops=1)
```

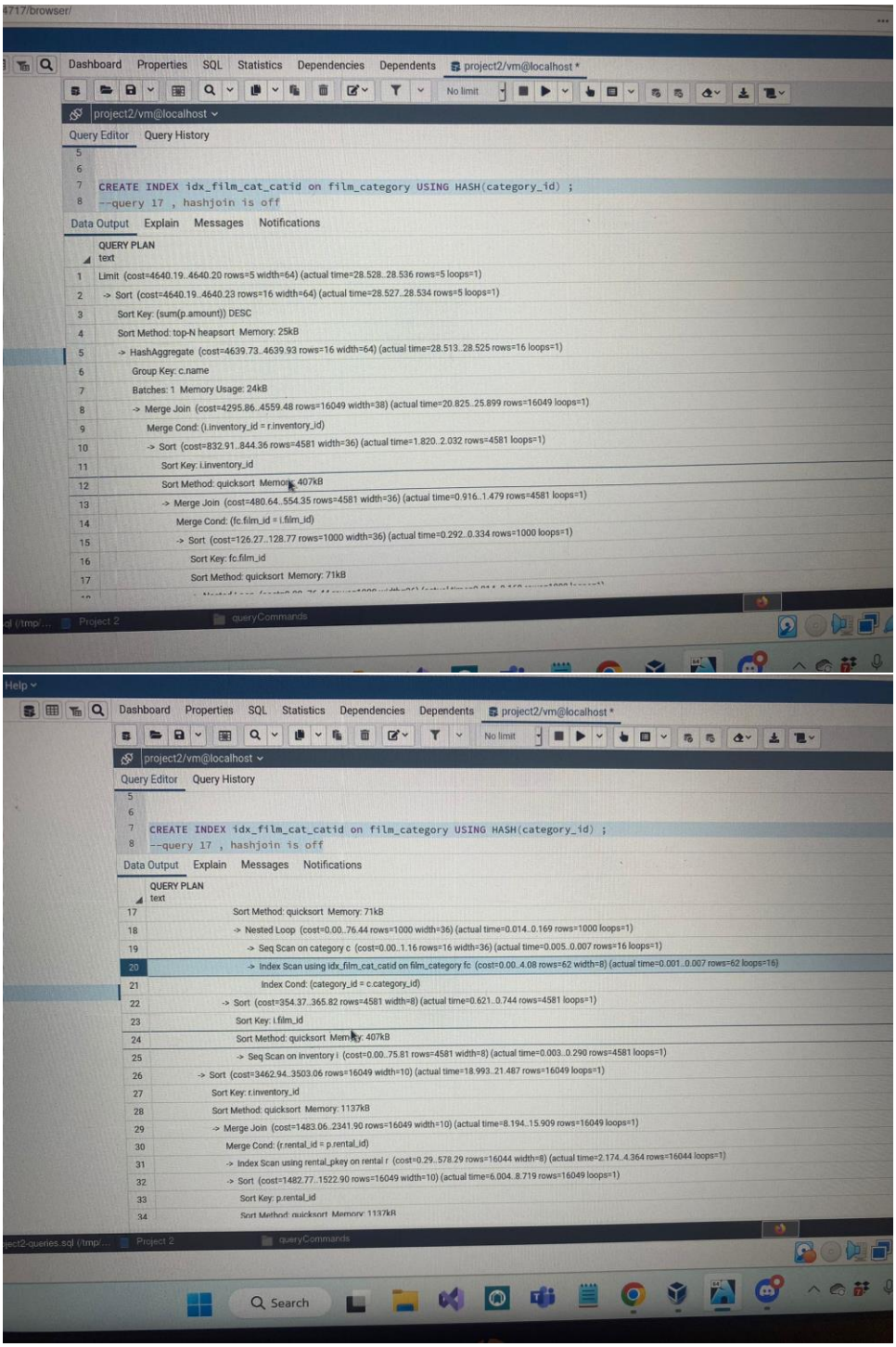
Project 2

Search

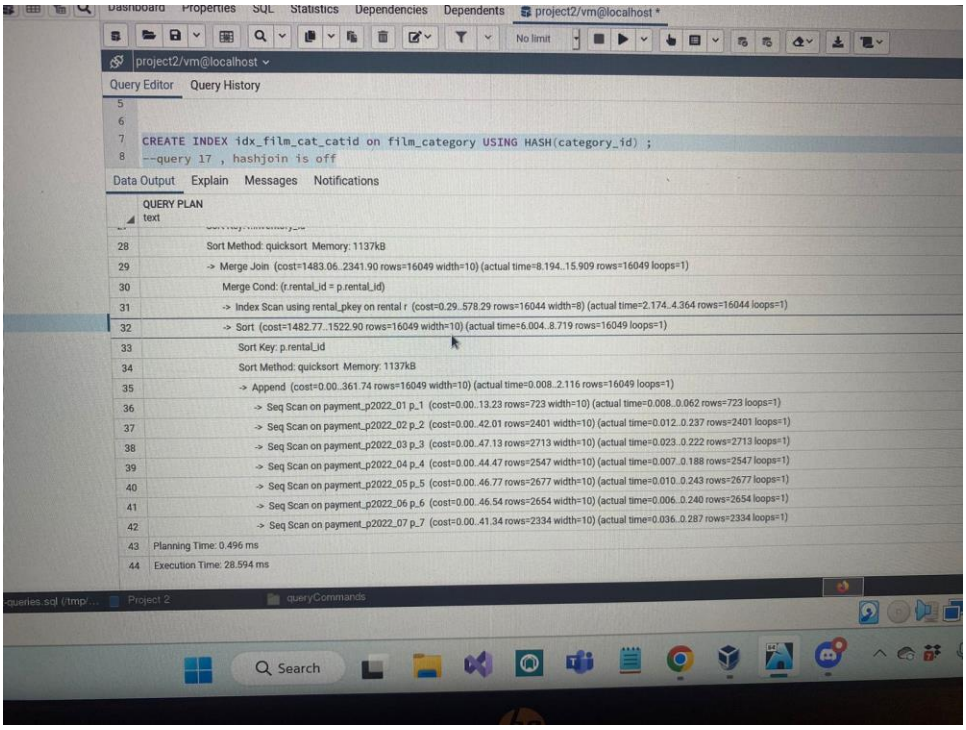


After Query 17:

- **Highest Cost:** Merge Join (cond. : r.rental_id = p.rental_id)
- **Slowest Runtime:** Merge Join (cond. : r.rental_id = p.rental_id)
- **Largest Number of Rows:** Merge Join (cond. : i.inventory_id=r.inventory_id), Merge Join (cond. : r.rental_id = p.rental_id) , Sort(sort key : r.inventory_id), Sort(sort key : p.rental_id)
- **Planning Time:** 0.496
- **Execution Time:** 28.594


Dashboard Properties SQL Statistics Dependencies Dependents project2/vm@localhost *
No limit
project2/vm@localhost
Query Editor Query History
CREATE INDEX idx_film_cat_catid on film_category USING HASH(category_id) ;
--query 17 , hashjoin is off
Data Output Explain Messages Notifications
QUERY PLAN
text
Limit (cost=4640.19..4640.20 rows=5 width=64) (actual time=28.528..28.536 rows=5 loops=1)
-> Sort (cost=4640.19..4640.23 rows=16 width=64) (actual time=28.527..28.534 rows=5 loops=1)
Sort Key: (sum(p.amount)) DESC
Sort Method: top-N heapsort Memory: 25kB
-> HashAggregate (cost=4639.73..4639.93 rows=16 width=64) (actual time=28.513..28.525 rows=16 loops=1)
Group Key: c.name
Batches: 1 Memory Usage: 24kB
-> Merge Join (cost=4295.86..4559.48 rows=16049 width=38) (actual time=20.825..25.899 rows=16049 loops=1)
Merge Cond: (i.inventory_id = r.inventory_id)
-> Sort (cost=832.91..844.36 rows=4581 width=36) (actual time=1.820..2.032 rows=4581 loops=1)
Sort Key: i.inventory_id
Sort Method: quicksort Memory: 407kB
-> Merge Join (cost=480.64..554.35 rows=4581 width=36) (actual time=0.916..1.479 rows=4581 loops=1)
Merge Cond: (fc.film_id = i.film_id)
-> Sort (cost=126.27..128.77 rows=1000 width=36) (actual time=0.292..0.334 rows=1000 loops=1)
Sort Key: fc.film_id
Sort Method: quicksort Memory: 71kB
queryCommands

Help
Dashboard Properties SQL Statistics Dependencies Dependents project2/vm@localhost *
No limit
project2/vm@localhost
Query Editor Query History
CREATE INDEX idx_film_cat_catid on film_category USING HASH(category_id) ;
--query 17 , hashjoin is off
Data Output Explain Messages Notifications
QUERY PLAN
text
Sort Method: quicksort Memory: 71kB
-> Nested Loop (cost=0.00..76.44 rows=1000 width=36) (actual time=0.014..0.169 rows=1000 loops=1)
-> Seq Scan on category c (cost=0.00..1.16 rows=16 width=36) (actual time=0.005..0.007 rows=16 loops=1)
-> Index Scan using idx_film_cat_catid on film_category fc (cost=0.00..4.08 rows=62 width=8) (actual time=0.001..0.007 rows=62 loops=16)
Index Cond: (category_id = c.category_id)
-> Sort (cost=354.37..365.82 rows=4581 width=8) (actual time=0.621..0.744 rows=4581 loops=1)
Sort Key: i.film_id
Sort Method: quicksort Memory: 407kB
-> Seq Scan on inventory i (cost=0.00..75.81 rows=4581 width=8) (actual time=0.003..0.290 rows=4581 loops=1)
-> Sort (cost=3462.94..3503.06 rows=16049 width=10) (actual time=18.993..21.487 rows=16049 loops=1)
Sort Key: r.inventory_id
Sort Method: quicksort Memory: 1137kB
-> Merge Join (cost=1483.06..2341.90 rows=16049 width=10) (actual time=8.194..15.909 rows=16049 loops=1)
Merge Cond: (r.rental_id = p.rental_id)
-> Index Scan using rental_pkey on rental r (cost=0.29..578.29 rows=16044 width=8) (actual time=2.174..4.364 rows=16044 loops=1)
-> Sort (cost=1482.77..1522.90 rows=16049 width=10) (actual time=6.004..8.719 rows=16049 loops=1)
Sort Key: p.rental_id
Sort Method: quicksort Memory: 1137kB
queryCommands
Search



Query (17) Justification:

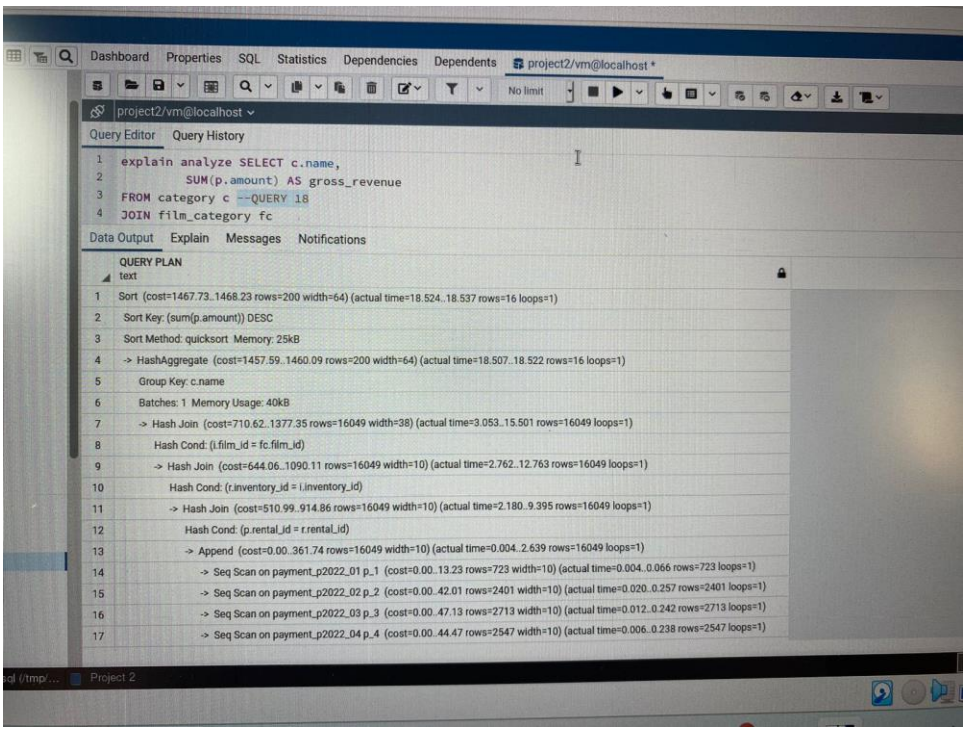
- Index On Attribute: Table-Name = category, Attribute-Name = category_id, we used hash index as we're creating an index on the foreign key so, we're filtering based on the equality of the IDs.
- We turned off the hash Join to make the planner use our index.
- The planner uses our index which results in higher costs, planning and execution times, as the planner was already using the plan that results in the most optimized costs, planning and execution times.

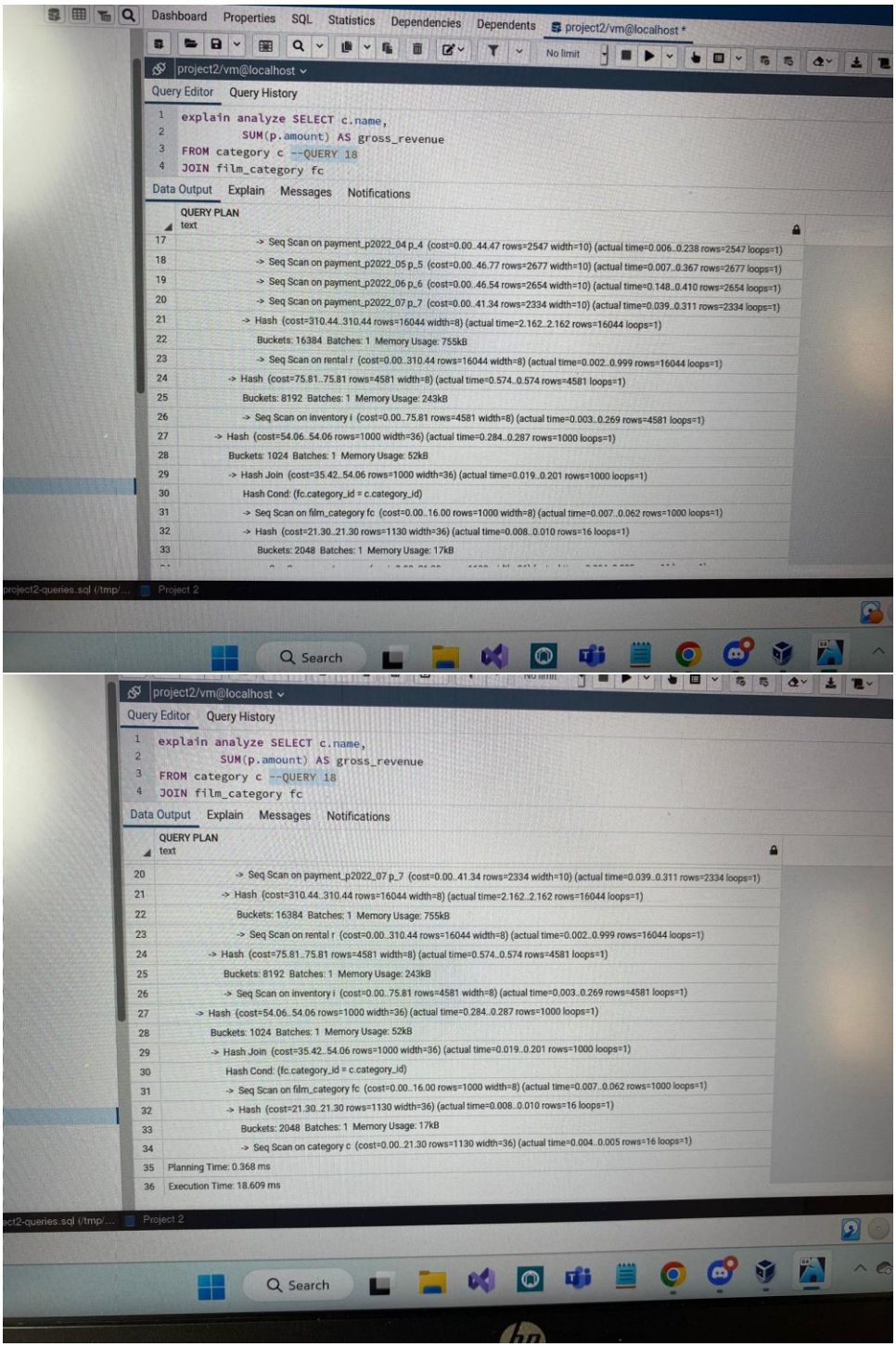
Query 18:

Before Query 18:

- **Highest Cost:** hash join (cond. : I.film_id = fc.film_id)
- **Slowest Runtime:** hash join (cond. : I.film_id = fc.film_id)
- **Largest Number of Rows:** hash join (cond. : I.film_id = fc.film_id) , hash join (cond. : r.inventory_id = i.inventory_id) , hash join (cond. : p.rental_id = r.rental_id) and append
- **Planning Time:** 0.368

- **Execution Time:** 18.609

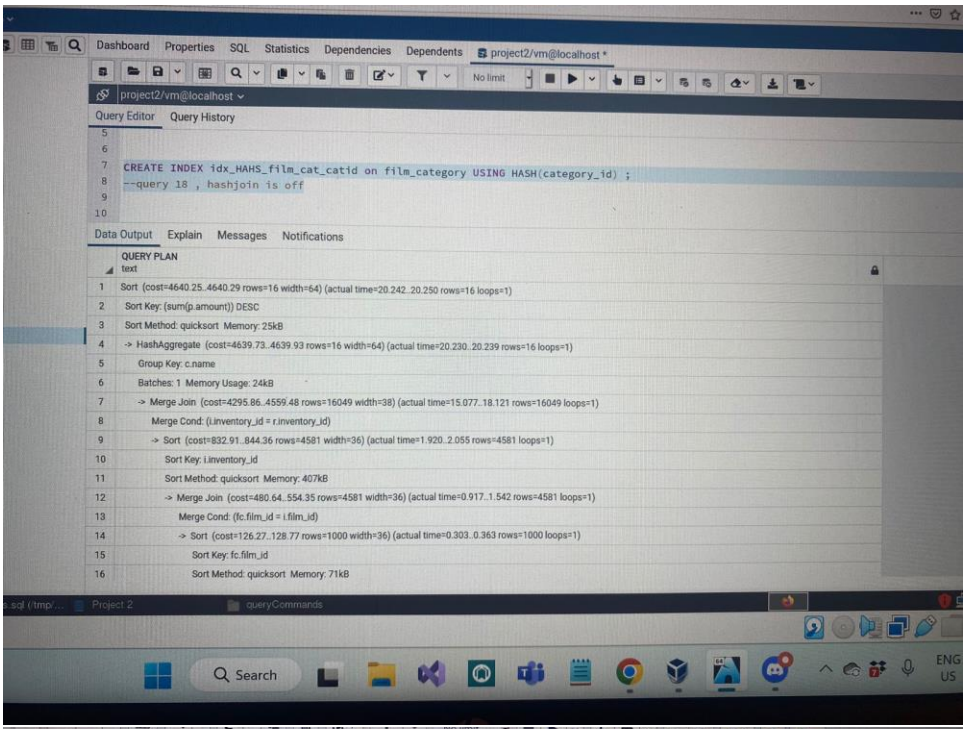


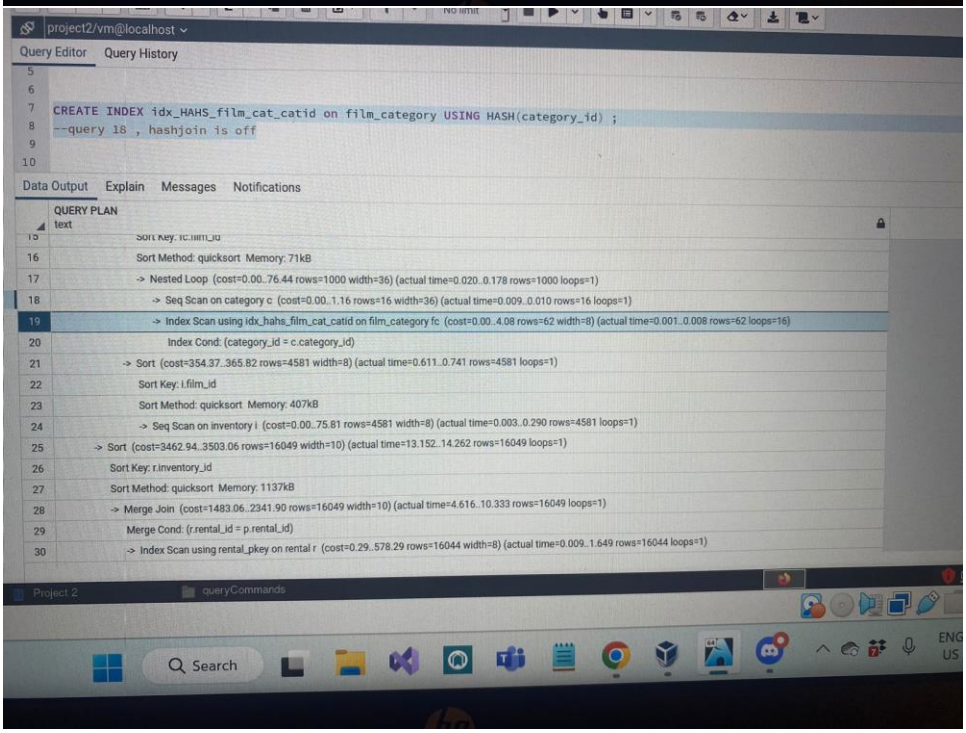


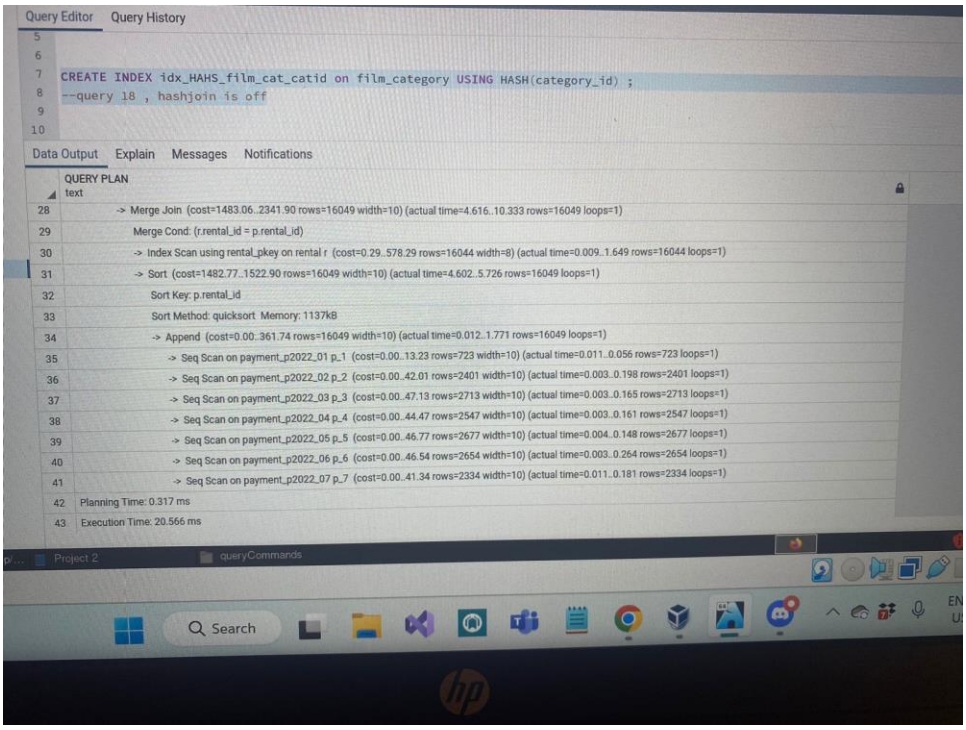
After Query 18:

- **Highest Cost:** Merge Join (cond. : r.rental_id = p.rental_id)
- **Slowest Runtime:** Merge Join (cond. : r.rental_id = p.rental_id)

- **Largest Number of Rows:** Merge Join (cond. : i.inventory_id=r.inventory_id), Merge Join (cond. : r.rental_id = p.rental_id) , Sort (sort key : p.rental_id) and Sort (sort key : r.inventory_id)
- **Planning Time:** 0.317
- **Execution Time:** 20.566


Dashboard Properties SQL Statistics Dependencies Dependents project2/vm@localhost *
project2/vm@localhost
Query Editor Query History
CREATE INDEX idx_HAHS_film_cat_catid on film_category USING HASH(category_id) ;
--query 18 , hashjoin is off
Data Output Explain Messages Notifications
QUERY PLAN
text
Sort (cost=4640.25..4640.29 rows=16 width=64) (actual time=20.242..20.250 rows=16 loops=1)
Sort Key: (sum(p.amount)) DESC
Sort Method: quicksort Memory: 25kB
-> HashAggregate (cost=4639.73..4639.93 rows=16 width=64) (actual time=20.230..20.239 rows=16 loops=1)
Group Key: c.name
Batches: 1 Memory Usage: 24kB
-> Merge Join (cost=4295.86..4559.48 rows=16049 width=38) (actual time=15.077..18.121 rows=16049 loops=1)
Merge Cond: (i.inventory_id = r.inventory_id)
-> Sort (cost=832.91..844.36 rows=4581 width=36) (actual time=1.920..2.055 rows=4581 loops=1)
Sort Key: i.inventory_id
Sort Method: quicksort Memory: 407kB
-> Merge Join (cost=480.64..554.35 rows=4581 width=36) (actual time=0.917..1.542 rows=4581 loops=1)
Merge Cond: (fc.film_id = i.film_id)
-> Sort (cost=126.27..128.77 rows=1000 width=36) (actual time=0.303..0.363 rows=1000 loops=1)
Sort Key: fc.film_id
Sort Method: quicksort Memory: 71kB
Project 2
queryCommands
Search
ENG
US


project2/vm@localhost
Query Editor Query History
CREATE INDEX idx_HAHS_film_cat_catid on film_category USING HASH(category_id) ;
--query 18 , hashjoin is off
Data Output Explain Messages Notifications
QUERY PLAN
text
Sort Method: quicksort Memory: 71kB
-> Nested Loop (cost=0.00..76.44 rows=1000 width=36) (actual time=0.020..0.178 rows=1000 loops=1)
-> Seq Scan on category c (cost=0.00..1.16 rows=16 width=36) (actual time=0.009..0.010 rows=16 loops=1)
-> Index Scan using idx_hahs_film_cat_catid on film_category fc (cost=0.00..4.08 rows=62 width=8) (actual time=0.001..0.008 rows=62 loops=16)
Index Cond: (category_id = c.category_id)
-> Sort (cost=354.37..365.82 rows=4581 width=8) (actual time=0.611..0.741 rows=4581 loops=1)
Sort Key: i.film_id
Sort Method: quicksort Memory: 407kB
-> Seq Scan on inventory i (cost=0.00..75.81 rows=4581 width=8) (actual time=0.003..0.290 rows=4581 loops=1)
-> Sort (cost=3462.94..3503.06 rows=16049 width=10) (actual time=13.152..14.262 rows=16049 loops=1)
Sort Key: r.inventory_id
Sort Method: quicksort Memory: 1137kB
-> Merge Join (cost=1483.06..2341.90 rows=16049 width=10) (actual time=4.616..10.333 rows=16049 loops=1)
Merge Cond: (r.rental_id = p.rental_id)
-> Index Scan using rental_pkey on rental r (cost=0.29..578.29 rows=16044 width=8) (actual time=0.009..1.649 rows=16044 loops=1)
Project 2
queryCommands
Search
ENG
US



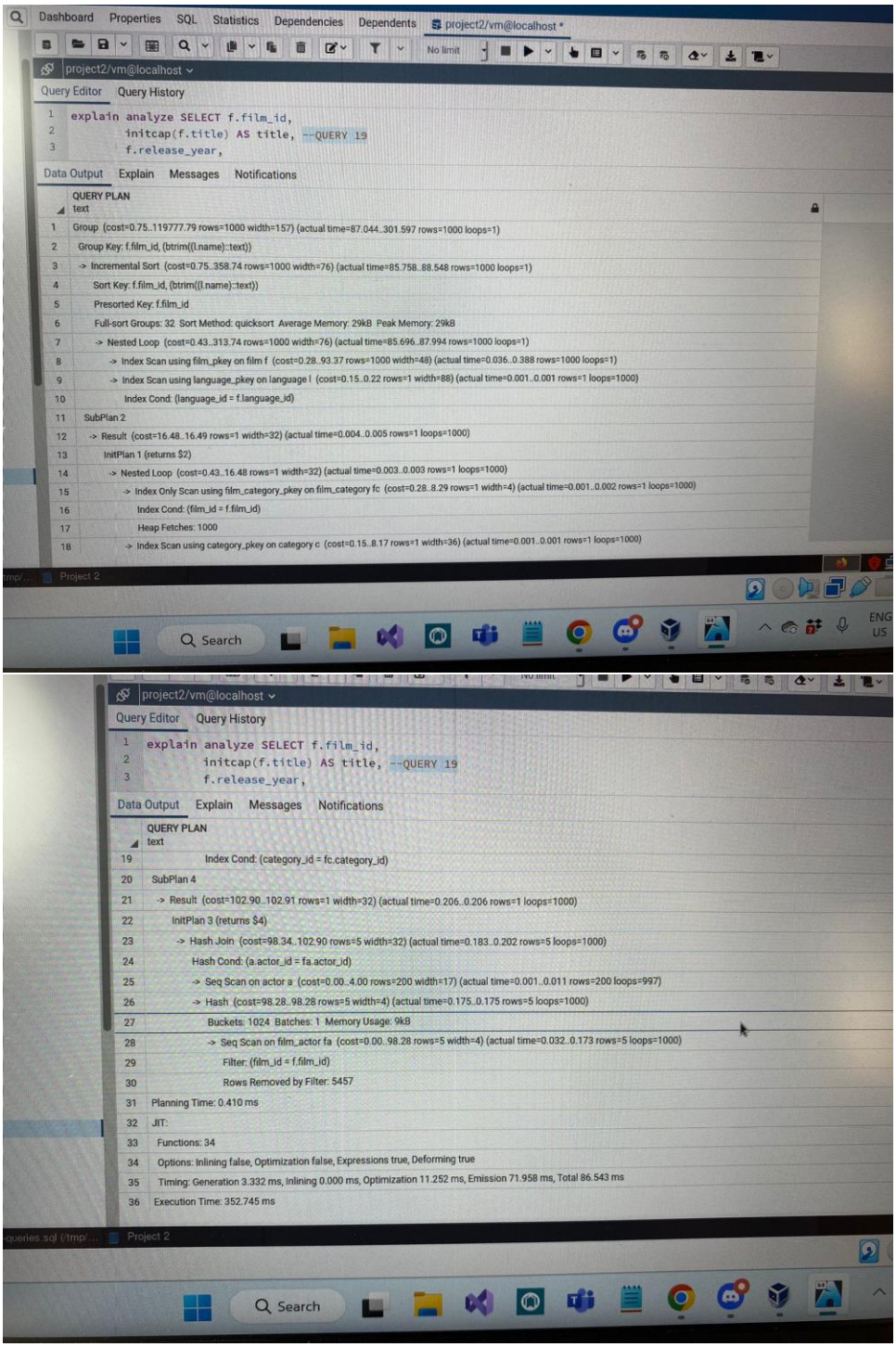
Query (18) Justification:

- Index On Attribute: Table-Name = category, Attribute-Name = category_id, we used hash index as we're creating an index on the foreign key so, we're filtering based on the equality of the IDs.
- We turned off the hash join to make the planner use our index.
- The planner uses our index which results in higher costs, planning and execution times, as the planner was already using the plan that results in the most optimized costs, planning and execution times.

Query 19:

Before Query 19:

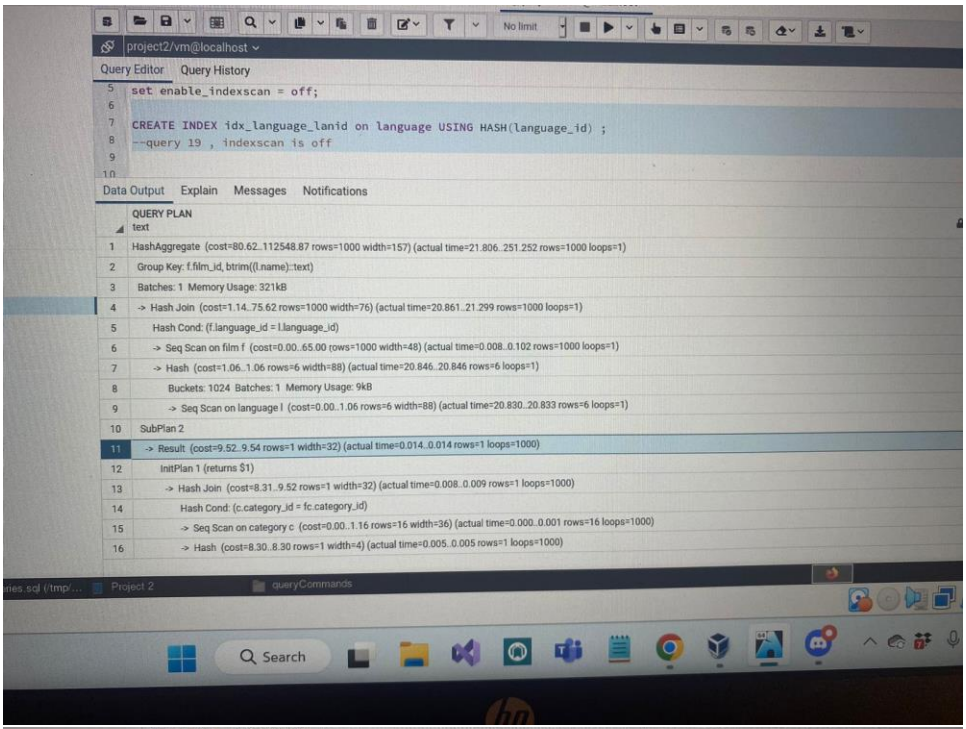
- **Highest Cost:** Group
- **Slowest Runtime:** Group
- **Largest Number of Rows:** Group
- **Planning Time:** 0.410
- **Execution Time:** 352.745

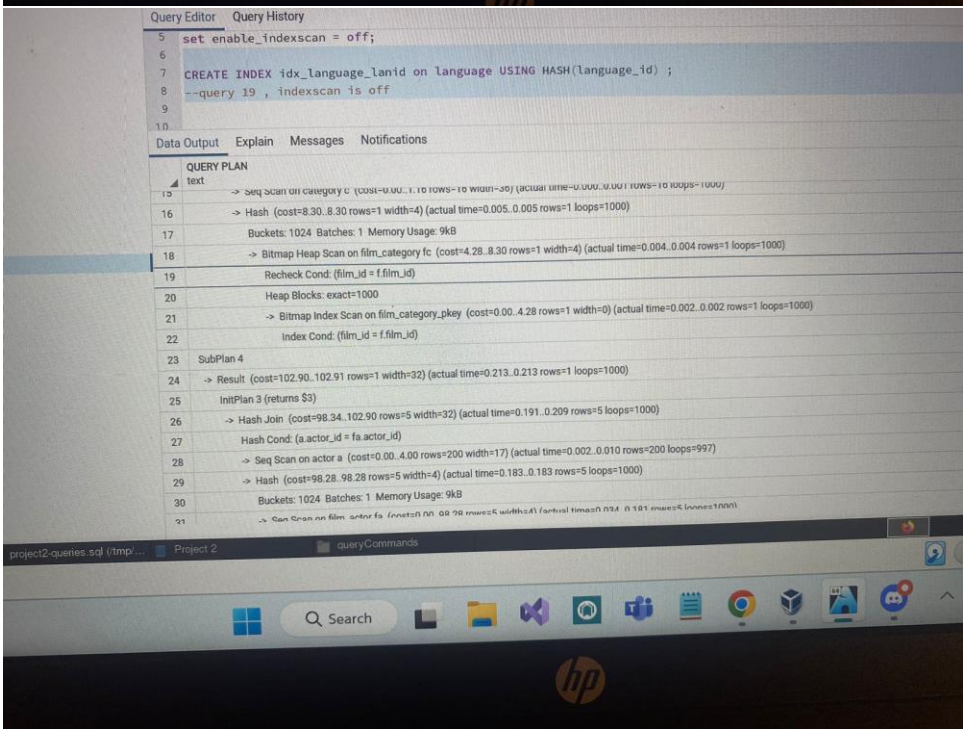


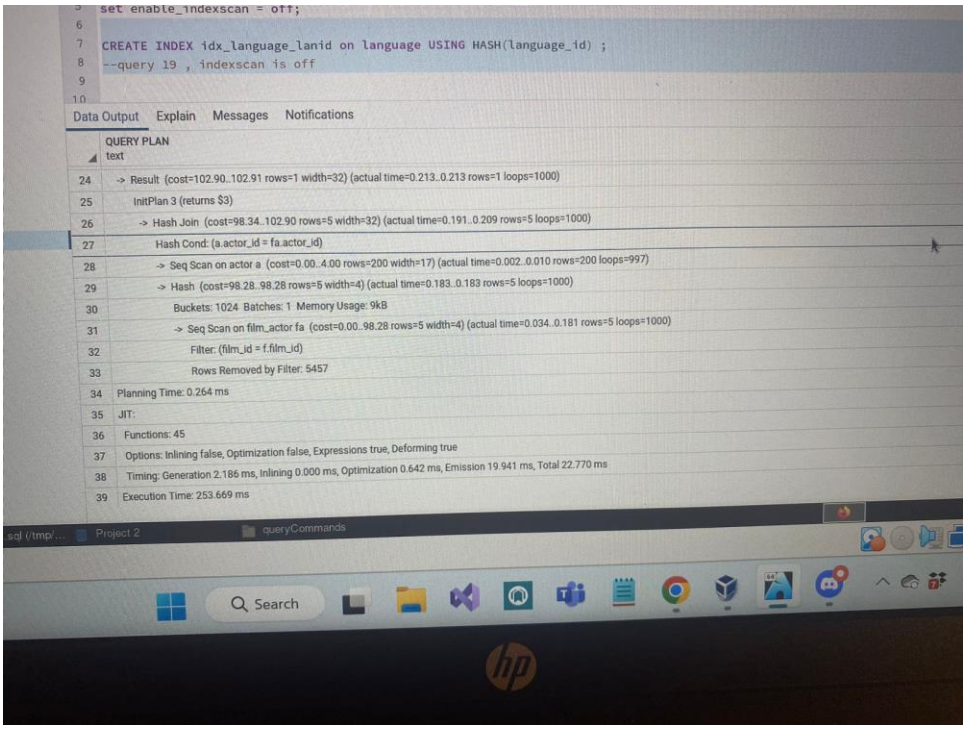
After Query 19:

- **Highest Cost:** Hash Aggregate
- **Slowest Runtime:** Hash Aggregate
- **Largest Number of Rows:** Hash Aggregate

- **Planning Time:** 0.264
- **Execution Time:** 253.669


project2/vm@localhost
Query Editor
Query History
set enable_indexscan = off;
CREATE INDEX idx_language_lanid on language USING HASH(language_id) ;
--query 19 , indexscan is off
Data Output
Explain
Messages
Notifications
QUERY PLAN
text
HashAggregate (cost=80.62..112548.87 rows=1000 width=157) (actual time=21.806..251.252 rows=1000 loops=1)
Group Key: f.film_id, btrim((l.name)::text)
Batches: 1 Memory Usage: 321kB
-> Hash Join (cost=1.14..75.62 rows=1000 width=76) (actual time=20.861..21.299 rows=1000 loops=1)
Hash Cond: (f.language_id = l.language_id)
-> Seq Scan on film f (cost=0.00..65.00 rows=1000 width=48) (actual time=0.008..0.102 rows=1000 loops=1)
-> Hash (cost=1.06..1.06 rows=6 width=88) (actual time=20.846..20.846 rows=6 loops=1)
Buckets: 1024 Batches: 1 Memory Usage: 9kB
-> Seq Scan on language l (cost=0.00..1.06 rows=6 width=88) (actual time=20.830..20.833 rows=6 loops=1)
SubPlan 2
-> Result (cost=9.52..9.54 rows=1 width=32) (actual time=0.014..0.014 rows=1 loops=1000)
InitPlan 1 (returns $1)
-> Hash Join (cost=8.31..9.52 rows=1 width=32) (actual time=0.008..0.009 rows=1 loops=1000)
Hash Cond: (c.category_id = fc.category_id)
-> Seq Scan on category c (cost=0.00..1.16 rows=16 width=36) (actual time=0.000..0.001 rows=16 loops=1000)
-> Hash (cost=8.30..8.30 rows=1 width=4) (actual time=0.005..0.005 rows=1 loops=1000)
Project 2
queryCommands
Search


Query Editor
Query History
set enable_indexscan = off;
CREATE INDEX idx_language_lanid on language USING HASH(language_id) ;
--query 19 , indexscan is off
Data Output
Explain
Messages
Notifications
QUERY PLAN
text
-> Hash (cost=8.30..8.30 rows=1 width=4) (actual time=0.005..0.005 rows=1 loops=1000)
Buckets: 1024 Batches: 1 Memory Usage: 9kB
-> Bitmap Heap Scan on film_category fc (cost=4.28..8.30 rows=1 width=4) (actual time=0.004..0.004 rows=1 loops=1000)
Recheck Cond: (film_id = f.film_id)
Heap Blocks: exact=1000
-> Bitmap Index Scan on film_category_pkey (cost=0.00..4.28 rows=1 width=0) (actual time=0.002..0.002 rows=1 loops=1000)
Index Cond: (film_id = f.film_id)
SubPlan 4
-> Result (cost=102.90..102.91 rows=1 width=32) (actual time=0.213..0.213 rows=1 loops=1000)
InitPlan 3 (returns $3)
-> Hash Join (cost=98.34..102.90 rows=5 width=32) (actual time=0.191..0.209 rows=5 loops=1000)
Hash Cond: (a.actor_id = fa.actor_id)
-> Seq Scan on actor a (cost=0.00..4.00 rows=200 width=17) (actual time=0.002..0.010 rows=200 loops=997)
-> Hash (cost=98.28..98.28 rows=5 width=4) (actual time=0.183..0.183 rows=5 loops=1000)
Buckets: 1024 Batches: 1 Memory Usage: 9kB
project2-queries.sql (/tmp/...
Project 2
queryCommands
Search



Query (19) Justification:

- Index On Attribute: Table-Name = language, Attribute-Name = language_id, we used hash index as we're creating an index on the foreign key so, we're filtering based on the equality of the IDs.
- We turned off the index scan to make the planner use our index.
- The planner didn't use our index even after disabling the index scan, as planner sees seq scan is a better option for optimizing the query than using our created index.