

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

A screenshot of a PostgreSQL Explain Analyze output. The query is:

```
1 explain analyze select
2   film_id,
3   title,
4   length
5   from
6     public.film
7   where length > 160
8   order by
9     length desc;
10
11 --QUERY 1
12
```

The Explain output shows the following steps:

- 1 Sort (cost=74.24..74.69 rows=180 width=21) (actual time=0.256..0.263 rows=176 loops=1)
- 2 Sort Key: length DESC
- 3 Sort Method: quicksort Memory: 38kB
- 4 → Seq Scan on film (cost=0.00..67.50 rows=180 width=21) (actual time=0.015..0.199 rows=176 loops=1)
- 5 Filter: (length > 160)
- 6 Rows Removed by Filter: 824
- 7 Planning Time: 0.166 ms
- 8 Execution Time: 0.281 ms

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

```

project2/vm@localhost ~
Query Editor Query History
1 -- CREATE INDEX idx_btree_film_len ON film USING BTREE (length);
2
3 explain analyze select
4   film_id, --QUERY 1 AFTER BTREE INDEX ON FILM(LENGTH)
5   title,
6   length
7 from
Data Output Explain Messages Notifications
QUERY PLAN
text
1 Sort (cost=69.66..70.11 rows=180 width=21) (actual time=0.119..0.123 rows=176 loops=1)
2   Sort Key: length DESC
3   Sort Method: quicksort Memory: 38kB
4     > Bitmap Heap Scan on film (cost=5.67..62.92 rows=180 width=21) (actual time=0.029..0.091 rows=176 loops=1)
5       Recheck Cond: (length > 160)
6       Heap Blocks: exact=53
7         > Bitmap Index Scan on idx_btree_film_len (cost=0.00..5.62 rows=180 width=0) (actual time=0.021..0.021 rows=176 loops=1)
8           Index Cond: (length > 160)
9   Planning Time: 0.137 ms
10  Execution Time: 0.140 ms

```

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.

Query 2:

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

The screenshot shows a PostgreSQL Explain Analyze output. The query is:

```
explain analyze select
    category.name,
    count(category.name) category_count
from -- QUERY 2
    category
left join film_category on
```

The Explain output shows the following plan:

```
1  Sort (cost=69.35..69.85 rows=200 width=40) (actual time=0.391..0.393 rows=16 loops=1)
2  Sort Key: (count(category.name)) DESC
3  Sort Method: quicksort Memory: 25kB
4  -> HashAggregate (cost=59.71..61.71 rows=200 width=40) (actual time=0.375..0.379 rows=16 loops=1)
5    Group Key category.name
6    Batches: 1 Memory Usage: 40kB
7    -> Hash Right Join (cost=35.42..54.06 rows=1130 width=32) (actual time=0.029..0.252 rows=1000 loops=1)
8      Hash Cond: (film_category.category_id = category.category_id)
9      -> Seq Scan on film_category (cost=0.00..16.00 rows=1000 width=8) (actual time=0.005..0.078 rows=1000 loops=1)
10     -> Hash (cost=21.30..21.30 rows=1130 width=36) (actual time=0.014..0.015 rows=16 loops=1)
11       Buckets: 2048 Batches: 1 Memory Usage: 17kB
12       -> Seq Scan on category (cost=0.00..21.30 rows=1130 width=36) (actual time=0.009..0.010 rows=16 loops=1)
13 Planning Time: 0.215 ms
14 Execution Time: 0.431 ms
```

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

```

1 CREATE INDEX idx_hash_filmcat_id ON film_category USING HASH(category_id);
2 --set enable_seqscan = off;
3 --Query 2 , hash index on film_category, turn off seqscan
4 explain analyze select

```

QUERY PLAN

```

1 Sort (cost=62.89..62.93 rows=16 width=40) (actual time=2.638..2.642 rows=16 loops=1)
2  Sort Key: (count(category.name)) DESC
3  Sort Method: quicksort Memory: 25kB
4   > HashAggregate (cost=62.41..62.57 rows=16 width=40) (actual time=2.615..2.619 rows=16 loops=1)
5    Group Key: category.name
6    Batches: 1 Memory Usage: 24kB
7   > Hash Right Join (cost=38.10..57.41 rows=1000 width=32) (actual time=2.164..2.445 rows=1000 loops=1)
8     Hash Cond: (film_category.category_id = category.category_id)
9     > Bitmap Heap Scan on film_category (cost=25.52..41.52 rows=1000 width=8) (actual time=2.067..2.156 rows=1000 loops=1)
10    Heap Blocks: exact=6
11   > Bitmap Index Scan on film_category_pkey (cost=0.00..25.27 rows=1000 width=0) (actual time=2.057..2.057 rows=1000 loops=1)
12   > Hash (cost=12.38..12.38 rows=16 width=36) (actual time=0.087..0.088 rows=16 loops=1)
13     Buckets: 1024 Batches: 1 Memory Usage: 9kB
14     > Index Scan using category_pkey on category (cost=0.14..12.38 rows=16 width=36) (actual time=0.050..0.055 rows=16 loops=1)
15 Planning Time: 0.234 ms
16 Execution Time: 2.740 ms

```

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.

Query 3:

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

The screenshot shows the pgAdmin III interface with the following details:

- Project:** project2/vm@localhost
- Query Editor:** Contains the SQL query:

```
1  explain analyze select
2    actor.first_name,
3    actor.last_name,
4    count(actor.first_name) featured_count
5  from -- QUERY 3
6    actor
7  left join film_actor on
```
- Data Output:** Shows the query plan in text format:

```
QUERY PLAN
text
1  Sort  (cost=152.48..152.80 rows=128 width=21) (actual time=1.915..1.929 rows=199 loops=1)
2  Sort Key: (count(actor.first_name)) DESC
3  Sort Method: quicksort Memory: 40kB
4  -> HashAggregate  (cost=146.72..148.00 rows=128 width=21) (actual time=1.857..1.882 rows=199 loops=1)
5    Group Key: actor.first_name, actor.last_name
6    Batches: 1 Memory Usage: 64kB
7    -> Hash Right Join  (cost=6.50..105.76 rows=5462 width=13) (actual time=0.066..1.069 rows=5462 loops=1)
8      Hash Cond: (film_actor.actor_id = actor.actor_id)
9      -> Seq Scan on film_actor  (cost=0.00..84.62 rows=5462 width=4) (actual time=0.003..0.318 rows=5462 loops=1)
10     -> Hash  (cost=4.00..4.00 rows=200 width=17) (actual time=0.054..0.057 rows=200 loops=1)
11       Buckets: 1024 Batches: 1 Memory Usage: 18kB
12       -> Seq Scan on actor  (cost=0.00..4.00 rows=200 width=17) (actual time=0.012..0.027 rows=200 loops=1)
13 Planning Time: 0.342 ms
14 Execution Time: 2.018 ms
```

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

```

5
6 --Query 3 , seqscan is off
7 CREATE INDEX idx_hash_actorid on film_actor USING HASH(actor_id) ;
8
9
Data Output Explain Messages Notifications
QUERY PLAN
text
1 Sort (cost=230.22..230.54 rows=128 width=21) (actual time=3.181..3.191 rows=199 loops=1)
2   Sort Key: (count(actor.first_name)) DESC
3   Sort Method: quicksort Memory: 40kB
4     > HashAggregate (cost=224.46..225.74 rows=128 width=21) (actual time=3.102..3.127 rows=199 loops=1)
5       Group Key: actor.first_name, actor.last_name
6       Batches: 1 Memory Usage: 64kB
7         > Hash Right Join (cost=18.93..183.50 rows=5462 width=13) (actual time=0.106..1.781 rows=5462 loops=1)
8           Hash Cond: (film_actor.actor_id = actor.actor_id)
9             > Index Only Scan using film_actor_pkey on film_actor (cost=0.28..150.21 rows=5462 width=4) (actual time=0.015..0.853 rows=5462 loops=1)
10            Heap Fetches: 0
11           > Hash (cost=16.14..16.14 rows=200 width=17) (actual time=0.085..0.086 rows=200 loops=1)
12             Buckets: 1024 Batches: 1 Memory Usage: 18kB
13               > Index Scan using actor_pkey on actor (cost=0.14..16.14 rows=200 width=17) (actual time=0.011..0.046 rows=200 loops=1)
14 Planning Time: 3.119 ms
15 Execution Time: 3.241 ms

```

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

127.0.0.1:44107/browser/

elp ▾

Dashboard Properties SQL Statistics Dependencies Dependents project2/vm@localhost *

project2/vm@localhost ▾

Query Editor Query History

```

1 explain analyze SELECT t1.name, t1.standard_quartile, COUNT(t1.standard_quartile)
2 FROM --QUERY 4
3 (SELECT f.title, c.name , f.rental_duration, NTILE(4) OVER (ORDER BY f.rental_duration) AS standard_quartile

```

Data Output Explain Messages Notifications

QUERY PLAN

text

```

1 GroupAggregate (cost=67.11..67.71 rows=30 width=44) (actual time=0.875..0.915 rows=24 loops=1)
2   Group Key: t1.name, t1.standard_quartile
3     -> Sort (cost=67.11..67.18 rows=30 width=36) (actual time=0.869..0.883 rows=361 loops=1)
4       Sort Key: t1.name, t1.standard_quartile
5       Sort Method: quicksort Memory: 41kB
6       -> Subquery Scan on t1 (cost=65.55..66.37 rows=30 width=36) (actual time=0.632..0.713 rows=361 loops=1)
7         -> WindowAgg (cost=65.55..66.07 rows=30 width=70) (actual time=0.631..0.693 rows=361 loops=1)
8           -> Sort (cost=65.55..65.62 rows=30 width=34) (actual time=0.599..0.609 rows=361 loops=1)
9             Sort Key: f.rental_duration
10            Sort Method: quicksort Memory: 41kB
11            -> Nested Loop (cost=30.48..64.81 rows=30 width=34) (actual time=0.034..0.555 rows=361 loops=1)
12              -> Hash Join (cost=30.20..48.84 rows=30 width=36) (actual time=0.029..0.177 rows=361 loops=1)
13                Hash Cond: (fc.category_id = c.category_id)
14                  -> Seq Scan on film_category fc (cost=0.00..16.00 rows=1000 width=8) (actual time=0.010..0.067 rows=1000 loops=1)
15                  -> Hash (cost=29.78..29.78 rows=34 width=36) (actual time=0.011..0.012 rows=6 loops=1)
16                    Buckets: 1024 Batches: 1 Memory Usage: 9kB
17                    -> Seq Scan on category c (cost=0.00..29.78 rows=34 width=36) (actual time=0.007..0.009 rows=6 loops=1)
18                      Filter: (name = ANY ('Animation,Children,Classics,Comedy,Family,Music')::text[])
19
20
21
22 Planning Time: 0.265 ms
23 Execution Time: 0.995 ms

```

2-queries.sql (tmp... Project 2

Help ▾

Dashboard Properties SQL Statistics Dependencies Dependents project2/vm@localhost *

project2/vm@localhost ▾

Query Editor Query History

```

1 explain analyze SELECT t1.name, t1.standard_quartile, COUNT(t1.standard_quartile)
2 FROM --QUERY 4
3 (SELECT f.title, c.name , f.rental_duration, NTILE(4) OVER (ORDER BY f.rental_duration) AS standard_quartile

```

Data Output Explain Messages Notifications

QUERY PLAN

text

```

6   -> Subquery Scan on t1 (cost=65.55..66.37 rows=30 width=36) (actual time=0.632..0.713 rows=361 loops=1)
7     -> WindowAgg (cost=65.55..66.07 rows=30 width=70) (actual time=0.631..0.693 rows=361 loops=1)
8       -> Sort (cost=65.55..65.62 rows=30 width=34) (actual time=0.599..0.609 rows=361 loops=1)
9         Sort Key: f.rental_duration
10        Sort Method: quicksort Memory: 41kB
11        -> Nested Loop (cost=30.48..64.81 rows=30 width=34) (actual time=0.034..0.555 rows=361 loops=1)
12          -> Hash Join (cost=30.20..48.84 rows=30 width=36) (actual time=0.029..0.177 rows=361 loops=1)
13            Hash Cond: (fc.category_id = c.category_id)
14              -> Seq Scan on film_category fc (cost=0.00..16.00 rows=1000 width=8) (actual time=0.010..0.067 rows=1000 loops=1)
15              -> Hash (cost=29.78..29.78 rows=34 width=36) (actual time=0.011..0.012 rows=6 loops=1)
16                Buckets: 1024 Batches: 1 Memory Usage: 9kB
17                -> Seq Scan on category c (cost=0.00..29.78 rows=34 width=36) (actual time=0.007..0.009 rows=6 loops=1)
18                  Filter: (name = ANY ('Animation,Children,Classics,Comedy,Family,Music')::text[])
19
20
21
22 Planning Time: 0.265 ms
23 Execution Time: 0.995 ms

```

project2queries.sql (tmp... Project 2

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

```

project2/vm@localhost ~
Query Editor Query History
1 --set enable_seqscan = off;
2 --drop index
3 set enable_seqscan = off;
4 CREATE INDEX idx_hash_category_name ON category USING HASH(name) ;
5 --Query 4 , setscan is off
6

Data Output Explain Messages Notifications
QUERY PLAN
text
1 Sort (cost=211.08..211.58 rows=200 width=44) (actual time=2.053..2.062 rows=24 loops=1)
2 Sort Key: c.name, (ntile(4) OVER (?))
3 Sort Method: quicksort Memory: 26kB
4 -> HashAggregate (cost=201.44..203.44 rows=200 width=44) (actual time=0.804..0.814 rows=24 loops=1)
5   Group Key: c.name, ntile(4) OVER (?)
6   Batches: 1 Memory Usage: 40kB
7     -> WindowAgg (cost=188.31..194.87 rows=375 width=70) (actual time=0.701..0.757 rows=361 loops=1)
8       -> Sort (cost=188.31..189.25 rows=375 width=34) (actual time=0.665..0.682 rows=361 loops=1)
9         Sort Key: f.rental_duration
10        Sort Method: quicksort Memory: 41kB
11        -> Hash Join (cost=71.68..172.28 rows=375 width=34) (actual time=0.327..0.606 rows=361 loops=1)
12          Hash Cond: (ffilm_id = fc.film_id)
13            -> Index Scan using film_pkey on film f (cost=0.28..93.37 rows=1000 width=6) (actual time=0.004..0.175 rows=1000 loops=1)
14            -> Hash (cost=66.72..66.72 rows=375 width=36) (actual time=0.307..0.312 rows=361 loops=1)
15              Buckets: 1024 Batches: 1 Memory Usage: 24kB
16                -> Hash Join (cost=12.84..66.72 rows=375 width=36) (actual time=0.038..0.266 rows=361 loops=1)
17                  Hash Cond: (fc.category_id = c.category_id)
18                  -> Index Only Scan using film_category_pkey on film_category fc (cost=0.28..50.84 rows=1000 width=8) (actual time=0.015..0.136 rows=1000 loops=1)
19                    Heap Fetches: 1000
20                    -> Hash (cost=12.49..12.49 rows=6 width=36) (actual time=0.015..0.017 rows=6 loops=1)
21                      Buckets: 1024 Batches: 1 Memory Usage: 9kB
22                        -> Index Scan using category_pkey on category c (cost=0.14..12.49 rows=6 width=36) (actual time=0.006..0.010 rows=6 loops=1)
23                          Filter: (name = ANY ('{Animation,Children,Classics,Comedy,Family,Musical}'::text[]))
24                          Rows Removed by Filter: 10
25 Planning Time: 2.223 ms
26 Execution Time: 2.164 ms

```

```

project2/vm@localhost ~
Query Editor Query History
1 --set enable_seqscan = off;
2 --drop index
3 set enable_seqscan = off;
4 CREATE INDEX idx_hash_category_name ON category USING HASH(name) ;
5 --Query 4 , setscan is off
6

Data Output Explain Messages Notifications
QUERY PLAN
text
11 -> Hash Join (cost=71.68..172.28 rows=375 width=34) (actual time=0.327..0.606 rows=361 loops=1)
12   Hash Cond: (ffilm_id = fc.film_id)
13     -> Index Scan using film_pkey on film f (cost=0.28..93.37 rows=1000 width=6) (actual time=0.004..0.175 rows=1000 loops=1)
14     -> Hash (cost=66.72..66.72 rows=375 width=36) (actual time=0.307..0.312 rows=361 loops=1)
15       Buckets: 1024 Batches: 1 Memory Usage: 24kB
16         -> Hash Join (cost=12.84..66.72 rows=375 width=36) (actual time=0.038..0.266 rows=361 loops=1)
17           Hash Cond: (fc.category_id = c.category_id)
18           -> Index Only Scan using film_category_pkey on film_category fc (cost=0.28..50.84 rows=1000 width=8) (actual time=0.015..0.136 rows=1000 loops=1)
19             Heap Fetches: 1000
20             -> Hash (cost=12.49..12.49 rows=6 width=36) (actual time=0.015..0.017 rows=6 loops=1)
21               Buckets: 1024 Batches: 1 Memory Usage: 9kB
22                 -> Index Scan using category_pkey on category c (cost=0.14..12.49 rows=6 width=36) (actual time=0.006..0.010 rows=6 loops=1)
23                   Filter: (name = ANY ('{Animation,Children,Classics,Comedy,Family,Musical}'::text[]))
24                   Rows Removed by Filter: 10
25 Planning Time: 2.223 ms
26 Execution Time: 2.164 ms

```

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

```
1  explain analyze SELECT DATE_TRUNC('month', p.payment_date) pay_month, c.first_name || ' ' || c.last_name AS
2  FROM customer c -- QUERY 5
3  JOIN payment p
4  ON p.customer_id = c.customer_id
5  WHERE c.first_name || ' ' || c.last_name IN
6  (SELECT t1.full_name
7
8
9
10 Planning Time: 0.501 ms
11 Execution Time: 0.037 ms
```

After Query 5:

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

The screenshot shows a PostgreSQL query editor interface. The top menu includes Dashboard, Properties, SQL, Statistics, Dependencies, Dependents, and a connection tab labeled "project2/vm@localhost". Below the menu is a toolbar with various icons. The main area is divided into sections: Query Editor, Query History, Data Output, Explain, Messages, and Notifications. The "Explain" section is active and displays the query plan for the current query. The query itself is:

```

4 --query 5 , hashagg is off
5 CREATE INDEX idx_BTREE_FIRSLAST_CUS ON customer USING BTREE(first_name , last_name) ;
6 drop index idx_BTREE_FIRSLAST_CUS;
7

```

The query plan (highlighted in blue) shows the following steps:

- Sort (cost=0.05..0.05 rows=1 width=80) (actual time=0.006..0.007 rows=0 loops=1)
- Sort Key: ((c.first_name || '-'|| c.last_name), (date_trunc('month', p.payment_date)), (count(p.amount)))
- Sort Method: quicksort Memory: 25kB
- GroupAggregate (cost=0.01..0.04 rows=1 width=80) (actual time=0.003..0.003 rows=0 loops=1)
- Group Key: (((c.first_name || '-'|| c.last_name), (date_trunc('month', p.payment_date)))
- Sort (cost=0.01..0.02 rows=0 width=52) (actual time=0.002..0.003 rows=0 loops=1)
- Sort Key: ((c.first_name || '-'|| c.last_name), (date_trunc('month', p.payment_date)))
- Sort Method: quicksort Memory: 25kB
- Result (cost=0..0.00 rows=0 width=52) (actual time=0.001..0.001 rows=0 loops=1)
- One-Time Filter: false
- Planning Time: 0.313 ms
- Execution Time: 0.026 ms

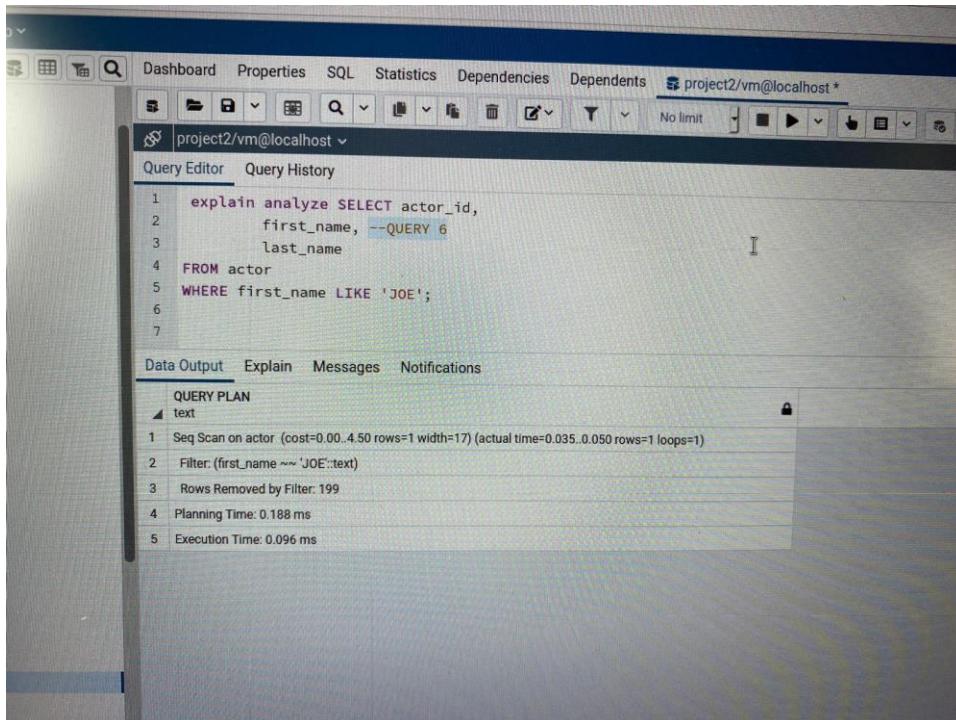
Query (5) Justification:

- 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



The screenshot shows the pgAdmin interface with the following details:

- Project:** project2/vm@localhost
- Query Editor:** Contains the SQL query:

```
1 explain analyze SELECT actor_id,
2         first_name, --QUERY 6
3         last_name
4     FROM actor
5     WHERE first_name LIKE 'JOE';
```
- Data Output:** Shows the **QUERY PLAN** tab selected, displaying the execution plan:
 - Seq Scan on actor (cost=0.00..4.50 rows=1 width=17) (actual time=0.035..0.050 rows=1 loops=1)
 - Filter: (first_name ~~ 'JOE':text)
 - Rows Removed by Filter: 199
- Messages:** Planning Time: 0.188 ms, Execution Time: 0.096 ms.

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

```
project2/vm@localhost ~
Query Editor Query History
5 set enable_hashagg = on;
6 set enable_seqscan = off;
7 --query 6 , setscan = off
8 CREATE INDEX idx_HASH_FIRST on actor USING HASH(first_name) ;
9
10
Data Output Explain Messages Notifications
QUERY PLAN
text
1 index Scan using idx_hash_first on actor (cost=0.00..8.02 rows=1 width=17) (actual time=0.014..0.015 rows=1 loops=1)
2 Index Cond: (first_name = 'JOE':text)
3 Filter: (first_name ~~ 'JOE':text)
4 Planning Time: 2.061 ms
5 Execution Time: 0.028 ms
```

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

The screenshot shows a PostgreSQL Explain Analyze output. The query is:

```
1 explain analyze SELECT *
2 FROM public.actor --QUERY 7
3 WHERE last_name similar to '%[GEN]%' and last_name = 'Smith';
4
5
6
```

The Data Output section shows the query plan:

```
1 Seq Scan on actor (cost=0.00..5.00 rows=1 width=25) (actual time=0.060..0.061 rows=0 loops=1)
2   Filter: ((last_name ~ '^([GEN].*)$'::text) AND (last_name = 'Smith'::text))
3   Rows Removed by Filter: 200
4 Planning Time: 0.078 ms
5 Execution Time: 0.068 ms
```

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

```

1 --set enable_seqscan = off;
2 --drop index
3
4
5 set enable_hashagg = on;
6 set enable_seqscan = off;
7
8 CREATE INDEX idx_HASH_ACTOR_LASTNAME on actor USING HASH(last_name);
9 --query 7 ,seqscan = off
10
11
12
13 explain_analyze SELECT *

```

The screenshot shows a PostgreSQL query editor window titled 'project2/vm@localhost'. It contains a script named 'queryCommands' with several SQL statements. The 13th statement is an 'explain_analyze' command for a 'SELECT *' query. The resulting explain plan is displayed in a table:

	text
1	Index Scan using idx_hash_actor_lastname on actor (cost=0.00..8.02 rows=1 width=25) (actual time=0.005..0.006 rows=1 loops=1)
2	Index Cond: (last_name ~ 'Smith')
3	Filter: (last_name ~ '^(?![GEN].*)\$')
4	Planning Time: 0.222 ms
5	Execution Time: 0.016 ms

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

The screenshot shows a PostgreSQL query editor window titled "project2/vm@localhost". The "Query Editor" tab is selected. The query text is:

```
1 explain analyze SELECT country_id,
2         country -- QUERY 8
3 FROM country
4 WHERE country IN ('Afghanistan', 'Bangladesh', 'China');
5
6
7
```

Below the query text, there are tabs for "Data Output", "Explain", "Messages", and "Notifications". The "Explain" tab is selected, showing the "QUERY PLAN" in "text" format:

```
1 Seq Scan on country (cost=0.00..2.50 rows=3 width=13) (actual time=0.018..0.028 rows=3 loops=1)
2   Filter: (country = ANY ('{Afghanistan,Bangladesh,China}'::text[]))
3 Rows Removed by Filter: 106
4 Planning Time: 0.142 ms
5 Execution Time: 0.041 ms
```

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

The screenshot shows a PostgreSQL query editor interface. The top bar has tabs for Dashboard, Properties, SQL, Statistics, Dependencies, Dependents, and project2/vm@localhost*. Below the tabs is a toolbar with various icons. The main area is a Query Editor with the following SQL code:

```

5 set enable_hashagg = on;
6 set enable_seqscan = off;
7
8 CREATE INDEX idx_hash_coun ON country USING HASH(country);
9 --query 8 , seqscan is off
10
11
12
13 explain analyze SELECT country_id, country
14 FROM country
15 WHERE country IN ('Afghanistan', 'Bangladesh', 'China');
16
17

```

Below the code, there are tabs for Data Output, Explain, Messages, and Notifications. The Explain tab is selected, showing the following query plan:

```

QUERY PLAN
text
1 Bitmap Heap Scan on country (cost=12.02..16.06 rows=3 width=13) (actual time=0.016..0.017 rows=3 loops=1)
2  Recheck Cond: (country = ANY ('{Afghanistan,Bangladesh,China}'::text[]))
3  Heap Blocks: exact=1
4 -> Bitmap Index Scan on idx_hash_coun (cost=0.00..12.02 rows=3 width=0) (actual time=0.010..0.010 rows=3 loops=1)
5   Index Cond: (country = ANY ('{Afghanistan,Bangladesh,China}'::text[]))
6   Planning Time: 0.057 ms
7   Execution Time: 0.031 ms

```

The Explain output shows that a bitmap index scan was chosen over a bitmap heap scan due to the use of the hash index.

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

The screenshot shows a database interface with a toolbar at the top and a main window below. The main window has tabs for 'Query Editor' and 'Query History', with 'Query Editor' selected. The code area contains the following SQL:

```
1 explain analyze SELECT last_name,
2      count(last_name) -- QUERY 9
3 FROM actor
4 GROUP BY last_name;
5
6
7
```

Below the code, there are tabs for 'Data Output', 'Explain', 'Messages', and 'Notifications'. The 'Explain' tab is selected, showing a 'QUERY PLAN' section with the following details:

```
1 HashAggregate (cost=5.00..6.21 rows=121 width=15) (actual time=0.067..0.079 rows=121 loops=1)
2   Group Key: last_name
3   Batches: 1 Memory Usage: 48kB
4     -> Seq Scan on actor (cost=0.00..4.00 rows=200 width=7) (actual time=0.009..0.025 rows=200 loops=1)
5 Planning Time: 0.051 ms
6 Execution Time: 0.111 ms
```

After Query 9:

This screenshot is identical to the one above, showing the same database interface and Explain Analyze results for Query 9. The code and query plan are the same, indicating that the query has already been executed.

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

The screenshot shows a PostgreSQL Explain Analyze output. The query is:

```
1 explain analyze SELECT last_name,
2      count(last_name) -- QUERY 10
3  FROM actor
4  where first_name like '%D'
5  GROUP BY last_name;
```

The Explain output shows a HashAggregate plan:

```
1 HashAggregate (cost=4.54..4.62 rows=8 width=15) (actual time=0.028..0.029 rows=5 loops=1)
2   Group Key: last_name
3   Batches: 1 Memory Usage: 24kB
4   -> Seq Scan on actor (cost=0.00..4.50 rows=8 width=7) (actual time=0.011..0.022 rows=5 loops=1)
5     Filter: (first_name ~~ '%D':text)
6     Rows Removed by Filter: 195
7   Planning Time: 0.064 ms
8   Execution Time: 0.048 ms
```

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

The screenshot shows a PostgreSQL query editor window titled 'project2/vm@localhost'. The 'Query Editor' tab is active, displaying the following SQL code:

```

3
4
5 set enable_hashagg = on;
6 set enable_seqscan = off;
7
8 CREATE INDEX idx_hash_actor_first ON actor USING HASH(first_name);
9 -- query 10 , seqscan is off
10

```

Below the code, the 'Data Output' tab is selected, showing the 'QUERY PLAN' for the query. The plan details the execution steps:

- GroupAggregate (cost=10000000004.62..10000000004.76 rows=8 width=15) (actual time=261.869..261.872 rows=5 loops=1)
 - Group Key: last_name
 - Sort (cost=10000000004.62..1000000004.64 rows=8 width=7) (actual time=261.862..261.863 rows=5 loops=1)
 - Sort Key: last_name
 - Sort Method: quicksort Memory: 25kB
 - Seq Scan on actor (cost=10000000000.00..10000000004.50 rows=8 width=7) (actual time=261.829..261.849 rows=5 loops=1)
 - Filter: (first_name ~~ '%D:text')
 - Rows Removed by Filter: 195
- Planning Time: 0.124 ms
- JIT:
 - Functions: 9
 - Options: Inlining true, Optimization true, Expressions true, Deforming true
 - Timing: Generation 0.567 ms, Inlining 89.890 ms, Optimization 80.670 ms, Emission 89.036 ms, Total 260.163 ms
- Execution Time: 305.075 ms

The operating system taskbar at the bottom shows various application icons.

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

The screenshot shows a PostgreSQL Query Editor interface. The top bar has tabs for "Query Editor" and "Query History". The main area displays the following SQL code:

```
1 explain analyze SELECT actor_id
2 FROM actor -- QUERY 11
3 WHERE first_name = 'HARPO'
4     AND last_name = 'WILLIAMS';
5
6
7
```

Below the code, there are tabs for "Data Output", "Explain", "Messages", and "Notifications". The "Explain" tab is selected, showing the "QUERY PLAN" with the "text" option expanded. The plan details the execution steps:

- Seq Scan on actor (cost=0.00..5.00 rows=1 width=4) (actual time=0.023..0.024 rows=0 loops=1)
- Filter: ((first_name = 'HARPO'::text) AND (last_name = 'WILLIAMS'::text))
- Rows Removed by Filter: 200
- Planning Time: 0.070 ms
- Execution Time: 0.035 ms

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

The screenshot shows a PostgreSQL Query Editor window. At the top, there are tabs for 'Query Editor' and 'Query History'. Below the tabs, the query text is displayed:

```
4
5 CREATE INDEX idx_BTREE_ACTOR_FNLN ON actor USING BTREE(first_name, last_name);
6 --query 11 , seqscan is off
```

Below the query text, there are tabs for 'Data Output', 'Explain', 'Messages', and 'Notifications'. The 'Explain' tab is selected, showing the following execution plan:

```
1 index Scan using idx_btreet_actor_fnln on actor (cost=0.14..8.16 rows=1 width=4) (actual time=0.017..0.017 rows=0 loops=1)
2   Index Cond: ((first_name = 'HARPO'-text) AND (last_name = 'WILLIAMS'-text))
3 Planning Time: 0.298 ms
4 Execution Time: 0.040 ms
```

The bottom of the window shows the operating system's taskbar with various application icons.

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

The screenshot shows a PostgreSQL Query Editor window. The top bar has tabs for "Query Editor" and "Query History". The main area displays the following SQL query:

```
1  explain analyze SELECT s.first_name,
2      s.last_name, -- QUERY 12
3      a.address
4  FROM staff s
5  LEFT JOIN address a
6    ON s.address_id = a.address_id;
```

Below the query, there are tabs for "Data Output", "Explain", "Messages", and "Notifications". The "Explain" tab is selected, showing the query plan in text format:

```
1 Hash Left Join (cost=21.57..35.74 rows=330 width=84) (actual time=0.105..0.107 rows=2 loops=1)
2   Hash Cond: (s.address_id = a.address_id)
3     -> Seq Scan on staff s (cost=0.00..13.30 rows=330 width=68) (actual time=0.009..0.009 rows=2 loops=1)
4     -> Hash (cost=14.03..14.03 rows=603 width=24) (actual time=0.092..0.092 rows=603 loops=1)
5       Buckets: 1024 Batches: 1 Memory Usage: 43kB
6         -> Seq Scan on address a (cost=0.00..14.03 rows=603 width=24) (actual time=0.007..0.041 rows=603 loops=1)
7 Planning Time: 0.154 ms
8 Execution Time: 0.118 ms
```

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

```

CREATE INDEX idx_add_addid on address USING HASH(address_id);
query 12 , seqscan is off
Nested Loop Left Join (cost=10000000000.00..10000000017.08 rows=2 width=84) (actual time=64.739..64.746 rows=2 loops=1)
  -> Seq Scan on staff s (cost=10000000000.00..1000000001.02 rows=2 width=68) (actual time=3.927..3.929 rows=2 loops=1)
  -> Index Scan using idx_add_addid on address a (cost=0.00..8.02 rows=1 width=24) (actual time=0.010..0.010 rows=1 loops=2)
    Index Cond: (address_id = s.address_id)
Planning Time: 5.407 ms
JIT:
  Functions: 6
  Options: Inlining true, Optimization true, Expressions true, Deforming true
  Timing: Generation 0.500 ms, Inlining 11.088 ms, Optimization 30.588 ms, Emission 18.999 ms, Total 61.175 ms
Execution Time: 65.288 ms

```

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
```

After Query 13:

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

```

1
2 set enable_hashagg = off;
3 set enable_seqscan = on;
4
5 CREATE INDEX idx_hash_staffid ON staff USING HASH(staff_id) ;
6 --query 13 , hashagg is off

```

QUERY PLAN

```

1 GroupAggregate (cost=1482.77..1605.14 rows=200 width=12) (actual time=4.227..5.013 rows=2 loops=1)
  2  Group Key: p.staff_id
  3   -> Sort (cost=1482.77..1522.90 rows=16049 width=10) (actual time=3.419..3.959 rows=16049 loops=1)
      Sort Key: p.staff_id
      Sort Method: quicksort Memory: 1137kB
  4     Append (cost=0.00..361.74 rows=16049 width=10) (actual time=0.013..1.972 rows=16049 loops=1)
      5       -> Seq Scan on payment_p2022_01 p_1 (cost=0.00..13.23 rows=723 width=10) (actual time=0.002..0.067 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..0.184 rows=2401 loops=1)
      7       -> Seq Scan on payment_p2022_03 p_3 (cost=0.00..47.13 rows=2713 width=10) (actual time=0.005..0.200 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..0.195 rows=2677 loops=1)
     10      -> Seq Scan on payment_p2022_06 p_6 (cost=0.00..46.54 rows=2654 width=10) (actual time=0.008..0.202 rows=2654 loops=1)
     11      -> Seq Scan on payment_p2022_07 p_7 (cost=0.00..41.34 rows=2334 width=10) (actual time=0.004..0.166 rows=2334 loops=1)
  12
  13
  14 Planning Time: 0.165 ms
  15 Execution Time: 5.077 ms

```

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:

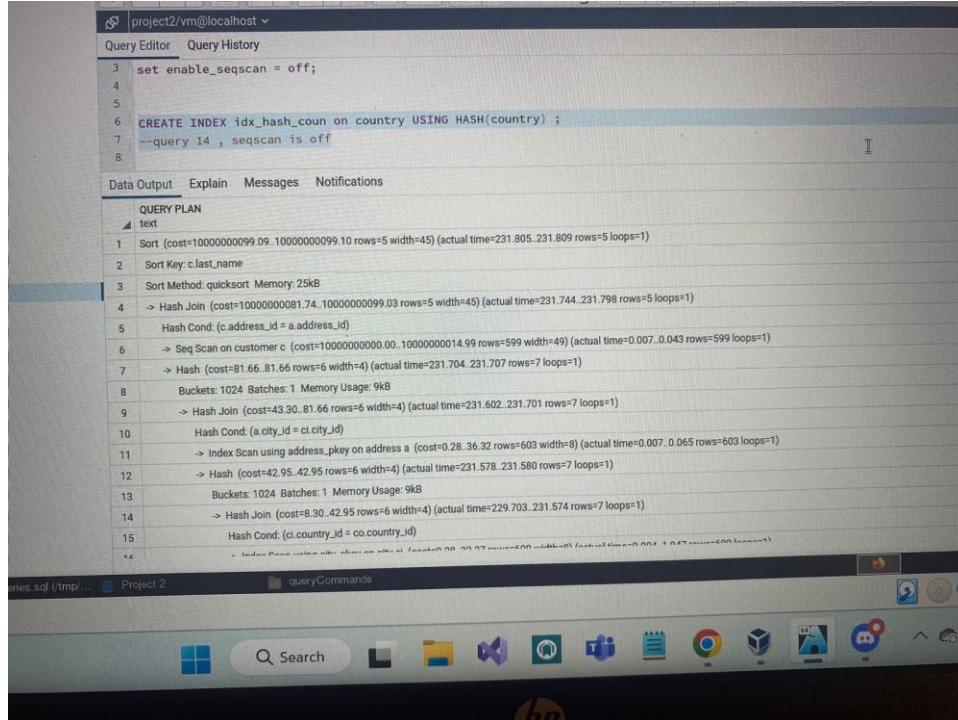
- **Highest Cost:** Hash join (cond. : c.address_id = a.address_id)
- **Slowest Runtime:** Hash join (cond. : a.city_id = ci.city_id)
- **Largest Number of Rows:** seq scan on address a
- **Planning Time:** 1.250
- **Execution Time:** 0.645

```
project2/vm@localhost * No limit
project2/vm@localhost
Query Editor Query History
1 explain analyze SELECT c.last_name,
2   c.first_name, -- QUERY 14
3   c.email
Data Output Explain Messages Notifications
QUERY PLAN
text
1 Sort (cost=48.87..48.88 rows=5 width=45) (actual time=0.585..0.589 rows=5 loops=1)
2 Sort Key: c.last_name
3 Sort Method: quicksort Memory: 25kB
4 -> Hash Join (cost=31.52..48.81 rows=5 width=45) (actual time=0.444..0.567 rows=5 loops=1)
5 Hash Cond: (c.address_id = a.address_id)
6   -> Seq Scan on customer c (cost=0.00..14.99 rows=599 width=49) (actual time=0.038..0.117 rows=599 loops=1)
7   -> Hash (cost=31.44..31.44 rows=6 width=4) (actual time=0.348..0.350 rows=7 loops=1)
8     Buckets: 1024 Batches: 1 Memory Usage: 9kB
9     -> Hash Join (cost=15.09..31.44 rows=6 width=4) (actual time=0.199..0.346 rows=7 loops=1)
10    Hash Cond: (a.city_id = ci.city_id)
11    -> Seq Scan on address a (cost=0.00..14.03 rows=603 width=8) (actual time=0.008..0.067 rows=603 loops=1)
12    -> Hash (cost=15.02..15.02 rows=6 width=4) (actual time=0.181..0.182 rows=7 loops=1)
13      Buckets: 1024 Batches: 1 Memory Usage: 9kB
14      -> Hash Join (cost=2.38..15.02 rows=6 width=4) (actual time=0.076..0.180 rows=7 loops=1)
15      Hash Cond: (ci.country_id = co.country_id)
16        -> Seq Scan on city ci (cost=0.00..11.00 rows=600 width=8) (actual time=0.006..0.069 rows=600 loops=1)
17        -> Hash (cost=2.36..2.36 rows=1 width=4) (actual time=0.024..0.024 rows=1 loops=1)
18          Buckets: 1024 Batches: 1 Memory Usage: 9kB
Project 2
```

```
project2/vm@localhost * No limit
project2/vm@localhost
Query Editor Query History
1 explain analyze SELECT c.last_name,
2   c.first_name, -- QUERY 14
3   c.email
Data Output Explain Messages Notifications
QUERY PLAN
text
6   -> Seq Scan on customer c (cost=0.00..14.99 rows=599 width=49) (actual time=0.038..0.117 rows=599 loops=1)
7   -> Hash (cost=31.44..31.44 rows=6 width=4) (actual time=0.348..0.350 rows=7 loops=1)
8     Buckets: 1024 Batches: 1 Memory Usage: 9kB
9     -> Hash Join (cost=15.09..31.44 rows=6 width=4) (actual time=0.199..0.346 rows=7 loops=1)
10    Hash Cond: (a.city_id = ci.city_id)
11    -> Seq Scan on address a (cost=0.00..14.03 rows=603 width=8) (actual time=0.008..0.067 rows=603 loops=1)
12    -> Hash (cost=15.02..15.02 rows=6 width=4) (actual time=0.181..0.182 rows=7 loops=1)
13      Buckets: 1024 Batches: 1 Memory Usage: 9kB
14      -> Hash Join (cost=2.38..15.02 rows=6 width=4) (actual time=0.076..0.180 rows=7 loops=1)
15      Hash Cond: (ci.country_id = co.country_id)
16        -> Seq Scan on city ci (cost=0.00..11.00 rows=600 width=8) (actual time=0.006..0.069 rows=600 loops=1)
17        -> Hash (cost=2.36..2.36 rows=1 width=4) (actual time=0.024..0.024 rows=1 loops=1)
18          Buckets: 1024 Batches: 1 Memory Usage: 9kB
19          -> Seq Scan on country co (cost=0.00..2.36 rows=1 width=4) (actual time=0.010..0.019 rows=1 loops=1)
20          Filter: (country = 'Canada':text)
21          Rows Removed by Filter: 108
22 Planning Time: 1.250 ms
23 Execution Time: 0.645 ms
ect2-queries.sql (/tmp/... Project 2
```

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



The screenshot shows a MySQL Workbench interface with a query editor window. The query being run is:

```

3 set enable_seqscan = off;
4
5
6 CREATE INDEX idx_hash_coun ON country USING HASH(country) ;
7 --query 14 , seqscan is off
8

```

The results section displays the execution plan (QUERY PLAN) in text format:

```

Data Output Explain Messages Notifications
QUERY PLAN
text
1 Sort [cost=10000000099.09..10000000099.10 rows=5 width=45] (actual time=231.805..231.809 rows=5 loops=1)
2 Sort Key c.last_name
3 Sort Method: quicksort Memory: 25kB
4   > Hash Join [cost=10000000081.74..10000000099.03 rows=5 width=45] (actual time=231.744..231.798 rows=5 loops=1)
5     Hash Cond: (c.address_id = a.address_id)
6       > Seq Scan on customer c [cost=10000000000.00..10000000014.99 rows=599 width=49] (actual time=0.007..0.043 rows=599 loops=1)
7         > Hash [cost=81.66..81.66 rows=5 width=4] (actual time=231.704..231.707 rows=7 loops=1)
8           Buckets: 1024 Batches: 1 Memory Usage: 9kB
9             > Hash Join [cost=43.30..81.66 rows=6 width=4] (actual time=231.602..231.701 rows=7 loops=1)
10            Hash Cond: (a.city_id = ci.city_id)
11              > Index Scan using address_pkey on address a [cost=0.28..36.32 rows=603 width=8] (actual time=0.007..0.065 rows=603 loops=1)
12                > Hash [cost=42.95..42.95 rows=6 width=4] (actual time=231.578..231.580 rows=7 loops=1)
13                  Buckets: 1024 Batches: 1 Memory Usage: 9kB
14                    > Hash Join [cost=8.30..42.95 rows=6 width=4] (actual time=229.703..231.574 rows=7 loops=1)
15                      Hash Cond: (ci.country_id = co.country_id)
16

```

The interface includes tabs for Data Output, Explain, Messages, and Notifications. The Explain tab is currently selected, showing the detailed execution plan. The status bar at the bottom indicates the file is 'queries.sql' and the project is 'Project 2'.

The screenshot shows a PostgreSQL query editor interface. The query being run is:

```

3 set enable_seqscan = off;
4
5
6 CREATE INDEX idx_hash_coun ON country USING HASH(country);
7 --query 14 , seqscan is off
8

```

The "Data Output" tab is selected, showing the execution plan:

```

QUERY PLAN
text
11 > Hash (cost=42.95..42.95 rows=6 width=4) (actual time=231.578..231.580 rows=7 loops=1)
12   Buckets: 1024 Batches: 1 Memory Usage: 9kB
13
14   Hash Join (cost=8.30..42.95 rows=6 width=4) (actual time=229.703..231.574 rows=7 loops=1)
15     Hash Cond: (ci.country_id = co.country_id)
16       > Index Scan using city_pkey on city ci (cost=0.28..33.27 rows=600 width=8) (actual time=0.004..1.947 rows=600 loops=1)
17       > Hash (cost=8.02..8.02 rows=1 width=4) (actual time=229.574..229.575 rows=1 loops=1)
18         Buckets: 1024 Batches: 1 Memory Usage: 9kB
19       > Index Scan using idx_hash_coun on country co (cost=0.00..8.02 rows=1 width=4) (actual time=229.563..229.568 rows=1 loops=1)
20         Index Cond: (country = 'Canada'.text)
21 Planning Time: 6.649 ms
22 JIT:
23   Functions: 28
24   Options: Inlining true, Optimization true, Expressions true, Deforming true
25   Timing: Generation 1.720 ms, Inlining 3.251 ms, Optimization 128.688 ms, Emission 97.436 ms, Total 231.096 ms
26 Execution Time: 233.624 ms

```

The "Planning Time" is 6.649 ms, and the "Execution Time" is 233.624 ms.

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

The screenshot shows a PostgreSQL Query Editor window. The top menu bar includes 'Query Editor' and 'Query History'. Below the menu is a code editor containing the following SQL query:

```
1 explain analyze SELECT title
2 FROM film f -- QUERY 15
3 JOIN film_category fc
4   ON f.film_id = fc.film_id
5 JOIN category c
6   ON fc.category_id = c.category_id
```

Below the code editor are tabs for 'Data Output', 'Explain', 'Messages', and 'Notifications'. A large text area displays the 'QUERY PLAN' for the query, starting with 'Nested Loop' and detailing various stages of the execution plan.

```
1 Nested Loop (cost=19.52..95.86 rows=69 width=15) (actual time=0.101..0.283 rows=69 loops=1)
2   -> Index Only Scan using category_pkey on category c (cost=0.15..8.17 rows=1 width=4) (actual time=0.036..0.037 rows=1 loops=1)
3     Index Cond: (category_id = 8)
4   Heap Fetches: 1
5   -> Hash Join (cost=19.36..87.00 rows=69 width=19) (actual time=0.063..0.238 rows=69 loops=1)
6     Hash Cond: (f.film_id = fc.film_id)
7     -> Seq Scan on film f (cost=0.00..65.00 rows=1000 width=19) (actual time=0.004..0.108 rows=1000 loops=1)
8     -> Hash (cost=18.50..18.50 rows=69 width=8) (actual time=0.053..0.054 rows=69 loops=1)
9       Buckets: 1024 Batches: 1 Memory Usage: 11kB
10      -> Seq Scan on film_category fc (cost=0.00..18.50 rows=69 width=8) (actual time=0.006..0.046 rows=69 loops=1)
11      Filter: (category_id = 8)
12      Rows Removed by Filter: 931
13 Planning Time: 0.299 ms
14 Execution Time: 0.308 ms
```

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

```

5
6 CREATE INDEX idx_HASH_CAT_CATID on category USING HASH(category_id) ;
7 --query 15 , seqscan is off
8
9
10
11
12
13
14
15

```

QUERY PLAN
text

- 1 Nested Loop (cost=35.79..140.23 rows=69 width=15) (actual time=0.158..0.516 rows=69 loops=1)
 - 2 -> Index Scan using idx_hash_cat_catid on category c (cost=0.00..8.02 rows=1 width=4) (actual time=0.036..0.037 rows=1 loops=1)
 - 3 Index Cond: (category_id = 8)
 - 4 -> Hash Join (cost=35.79..131.52 rows=69 width=19) (actual time=0.119..0.469 rows=69 loops=1)
 - 5 Hash Cond: (f.film_id = fc.film_id)
 - 6 -> Index Scan using film_pkey on film f (cost=0.28..93.37 rows=1000 width=19) (actual time=0.010..0.257 rows=1000 loops=1)
 - 7 -> Hash (cost=34.65..34.65 rows=69 width=8) (actual time=0.091..0.092 rows=69 loops=1)
 - 8 Buckets: 1024 Batches: 1 Memory Usage: 11kB
 - 9 -> Bitmap Heap Scan on film_category fc (cost=27.79..34.65 rows=69 width=8) (actual time=0.044..0.075 rows=69 loops=1)
 - 10 Recheck Cond: (category_id = 8)
 - 11 Heap Block: exact=6
 - 12 -> Bitmap Index Scan on film_category_pkey (cost=0.00..27.77 rows=69 width=0) (actual time=0.032..0.032 rows=69 loops=1)
 - 13 Index Cond: (category_id = 8)

Planning Time: 0.446 ms
Execution Time: 0.566 ms

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

```

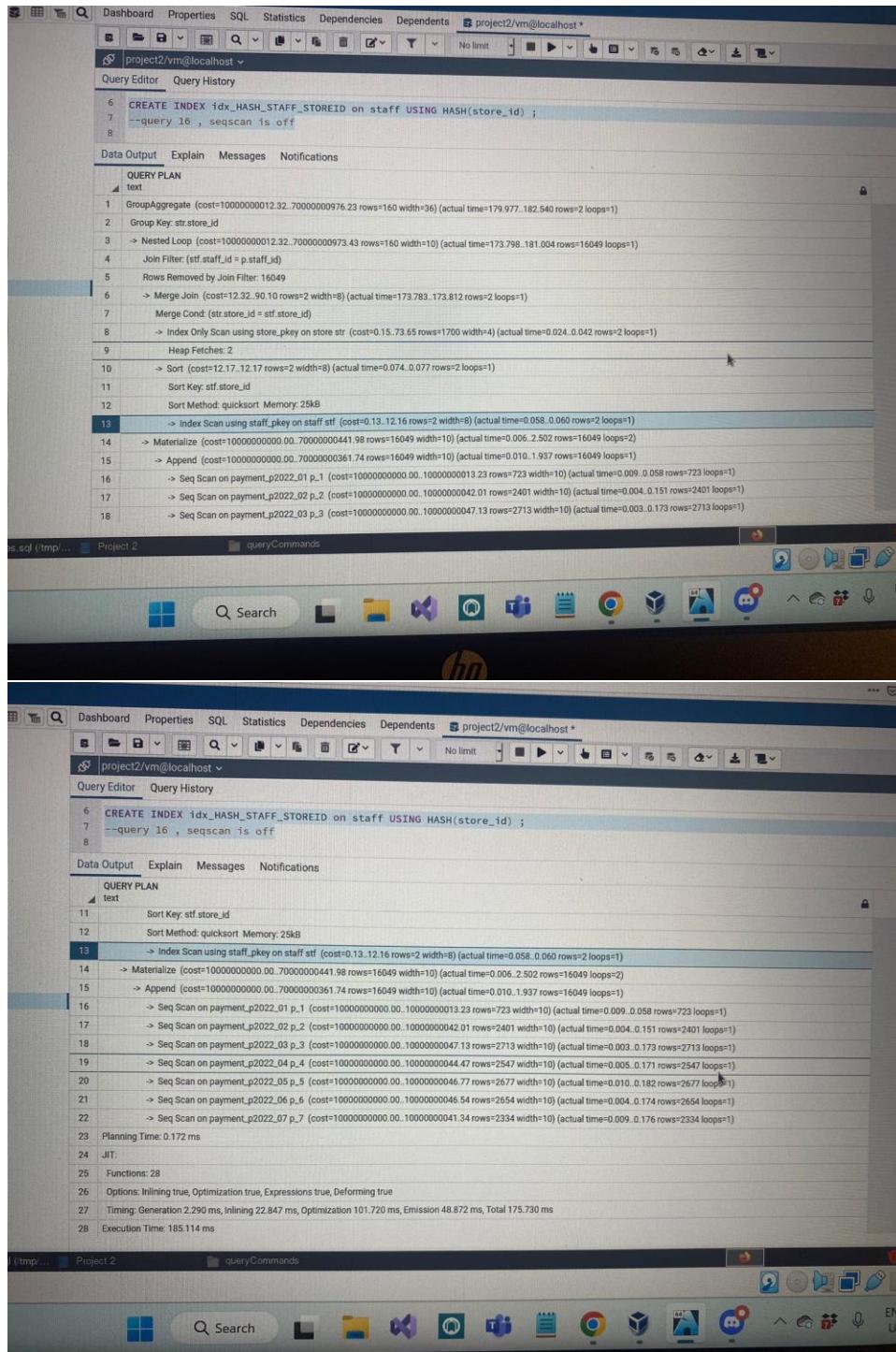
project2/vm@localhost
Query Editor Query History
1  explain analyze SELECT str.store_id,
2      SUM(p.amount) AS total_sales
3  FROM store str -- QUERY 16
4  JOIN staff stf
5    ON str.store_id = stf.store_id
6  JOIN payment p
7    ON stf.staff_id = p.staff_id
Data Output Explain Messages Notifications
QUERY PLAN
text
1 HashAggregate (cost=592.56..613.81 rows=1700 width=36) (actual time=8.119..8.130 rows=2 loops=1)
  Key: str.store_id
  Batches: 1 Memory Usage: 73kB
  -> Hash Join (cost=65.67..512.31 rows=16049 width=10) (actual time=0.050..5.944 rows=16049 loops=1)
    Hash Cond: (stf.store_id = str.store_id)
    -> Hash Join (cost=17.43..421.83 rows=16049 width=10) (actual time=0.036..4.230 rows=16049 loops=1)
      Hash Cond: (p.staff_id = stf.staff_id)
      -> Append (cost=0.00..361.74 rows=16049 width=10) (actual time=0.016..2.388 rows=16049 loops=1)
        -> Seq Scan on payment_p2022_01 p_1 (cost=0.00..13.23 rows=723 width=10) (actual time=0.015..0.079 rows=723 loops=1)
        -> Seq Scan on payment_p2022_02 p_2 (cost=0.00..42.01 rows=2401 width=10) (actual time=0.013..0.267 rows=2401 loops=1)
        -> Seq Scan on payment_p2022_03 p_3 (cost=0.00..47.13 rows=2713 width=10) (actual time=0.020..0.234 rows=2713 loops=1)
        -> Seq Scan on payment_p2022_04 p_4 (cost=0.00..44.47 rows=2547 width=10) (actual time=0.008..0.255 rows=2547 loops=1)
        -> Seq Scan on payment_p2022_05 p_5 (cost=0.00..46.77 rows=2677 width=10) (actual time=0.031..0.326 rows=2677 loops=1)
        -> Seq Scan on payment_p2022_06 p_6 (cost=0.00..46.54 rows=2654 width=10) (actual time=0.009..0.248 rows=2654 loops=1)
        -> Seq Scan on payment_p2022_07 p_7 (cost=0.00..41.34 rows=2334 width=10) (actual time=0.008..0.196 rows=2334 loops=1)
      -> Seq Scan on staff stf (cost=0.00..13.30 rows=330 width=8) (actual time=0.010..0.010 rows=2 loops=1)
      Buckets: 1024 Batches: 1 Memory Usage: 9kB
      -> Hash (cost=27.00..27.00 rows=1700 width=4) (actual time=0.009..0.010 rows=2 loops=1)
      Buckets: 2048 Batches: 1 Memory Usage: 17kB
      -> Seq Scan on store str (cost=0.00..27.00 rows=1700 width=4) (actual time=0.008..0.008 rows=2 loops=1)
Planning Time: 0.243 ms
Execution Time: 8.214 ms

```

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



The screenshot shows the Oracle SQL Developer interface with three windows side-by-side. Each window displays the same SQL command and its execution plan.

```

6 CREATE INDEX idx_HASH_STAFF_STOREID on staff USING HASH(store_id) ;
7 --query 16 , seqscan is off
8

```

QUERY PLAN

```

1 GroupAggregate (cost=100000000012.32..70000000976.23 rows=160 width=36) (actual time=179.977..182.540 rows=2 loops=1)
  Group Key: stf.store_id
  2 Nested Loop (cost=10000000012.32..70000000973.43 rows=160 width=10) (actual time=173.798..181.004 rows=16049 loops=1)
    Join Filter: (stf.staff_id = p.staff_id)
    3 Rows Removed by Join Filter: 16049
    4 Merge Join (cost=12.32..90.10 rows=2 width=8) (actual time=173.783..173.812 rows=2 loops=1)
      Merge Cond: (stf.store_id = stf.store_id)
    5 Index Only Scan using store_pkey on store str (cost=0.15..73.65 rows=1700 width=4) (actual time=0.024..0.042 rows=2 loops=1)
    6 Heap Fetcher: 2
    7 Sort (cost=12.17..12.17 rows=2 width=8) (actual time=0.074..0.077 rows=2 loops=1)
      Sort Key: stf.store_id
    8 Sort Method: quicksort Memory: 25kB
    9 Index Scan using staff_pkey on staff stf [cost=0.13..12.16 rows=2 width=8] (actual time=0.058..0.060 rows=2 loops=1)
    10 11 12 Materialize (cost=100000000000.00..70000000441.98 rows=16049 width=10) (actual time=0.006..2.502 rows=16049 loops=2)
    13 Append (cost=100000000000.00..70000000361.74 rows=16049 width=10) (actual time=0.010..1.937 rows=16049 loops=1)
    14 Seq Scan on payment_p2022_01 p_1 (cost=100000000000.00..10000000013.23 rows=723 width=10) (actual time=0.009..0.058 rows=723 loops=1)
    15 Seq Scan on payment_p2022_02 p_2 (cost=100000000000.00..10000000042.01 rows=2401 width=10) (actual time=0.004..0.151 rows=2401 loops=1)
    16 Seq Scan on payment_p2022_03 p_3 (cost=100000000000.00..10000000047.13 rows=2713 width=10) (actual time=0.003..0.173 rows=2713 loops=1)
    17 Seq Scan on payment_p2022_04 p_4 (cost=100000000000.00..1000000044.47 rows=2547 width=10) (actual time=0.005..0.171 rows=2547 loops=1)
    18 Seq Scan on payment_p2022_05 p_5 (cost=100000000000.00..1000000046.77 rows=2677 width=10) (actual time=0.010..0.182 rows=2677 loops=1)
    19 Seq Scan on payment_p2022_06 p_6 (cost=100000000000.00..1000000046.54 rows=2654 width=10) (actual time=0.004..0.174 rows=2654 loops=1)
    20 Seq Scan on payment_p2022_07 p_7 (cost=100000000000.00..1000000041.34 rows=2334 width=10) (actual time=0.009..0.176 rows=2334 loops=1)
    21 Planning Time: 0.172 ms
    22 JIT:
    23 Functions: 28
    24 Options: Inlining true, Optimization true, Expressions true, Deforming true
    25 Timing: Generation 2.290 ms, Inlining 22.847 ms, Optimization 101.720 ms, Emission 48.872 ms, Total 175.730 ms
    26 Execution Time: 185.114 ms

```

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. : l.film_id = fc.film_id)
- **Slowest Runtime:** Hash Join (cond. : l.film_id = fc.film_id)
- **Largest Number of Rows:** Hash Join (cond. : l.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 = 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 File Home Recent Help

Query Editor Query History

Project 2

File Home Recent Help

```

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)

```

tmp... Project 2

File Home Recent Help

```

project2/vm@localhost ~
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
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
Planning Time: 0.504 ms
Execution Time: 20.071 ms

```

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

1717/browser/

```

Dashboard Properties SQL Statistics Dependencies Dependents project2/vm@localhost *
project2/vm@localhost ▾
Query Editor Query History
5
6
7 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
1 Limit (cost=4640.19..4640.20 rows=5 width=64) (actual time=28.528..28.536 rows=5 loops=1)
2   > Sort (cost=4640.19..4640.23 rows=16 width=64) (actual time=28.527..28.534 rows=5 loops=1)
3     Sort Key: (sum(p.amount)) DESC
4     Sort Method: top-N heapsort Memory: 25kB
5       > HashAggregate (cost=4639.73..4639.93 rows=16 width=64) (actual time=28.513..28.525 rows=16 loops=1)
6         Group Key: c.name
7         Batches: 1 Memory Usage: 24kB
8           > Merge Join (cost=4295.86..4559.48 rows=16049 width=38) (actual time=20.825..25.899 rows=16049 loops=1)
9             Merge Cond: (i.inventory_id = ri.inventory_id)
10            > Sort (cost=832.91..844.36 rows=4581 width=36) (actual time=1.820..2.032 rows=4581 loops=1)
11              Sort Key: i.inventory_id
12              Sort Method: quicksort Memory: 407kB
13            > Merge Join (cost=480.64..554.35 rows=4581 width=36) (actual time=0.916..1.479 rows=4581 loops=1)
14              Merge Cond: (fc.film_id = f.film_id)
15              > Sort (cost=126.27..128.77 rows=1000 width=36) (actual time=0.292..0.334 rows=1000 loops=1)
16                Sort Key: fc.film_id
17                Sort Method: quicksort Memory: 71kB
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34

```

Help ▾

Dashboard Properties SQL Statistics Dependencies Dependents project2/vm@localhost *

project2/vm@localhost ▾

Query Editor Query History

```

5
6
7 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
17 Sort Method: quicksort Memory: 71kB
18   > Nested Loop (cost=0.00..76.44 rows=1000 width=36) (actual time=0.014..0.169 rows=1000 loops=1)
19     > Seq Scan on category c (cost=0.00..1.16 rows=16 width=36) (actual time=0.005..0.007 rows=16 loops=1)
20     > 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)
21       Index Cond: (category_id = c.category_id)
22     > Sort (cost=354.37..365.82 rows=4581 width=8) (actual time=0.621..0.744 rows=4581 loops=1)
23       Sort Key: f.film_id
24       Sort Method: quicksort Memory: 407kB
25     > Seq Scan on inventory i (cost=0.00..75.81 rows=4581 width=8) (actual time=0.003..0.290 rows=4581 loops=1)
26     > Sort (cost=3462.94..3503.06 rows=16049 width=10) (actual time=18.993..21.487 rows=16049 loops=1)
27       Sort Key: i.inventory_id
28       Sort Method: quicksort Memory: 1137kB
29     > Merge Join (cost=1483.06..2341.90 rows=16049 width=10) (actual time=8.194..15.909 rows=16049 loops=1)
30       Merge Cond: (r.rental_id = p.rental_id)
31       > Index Scan using rental_pk on rental r (cost=0.29..578.29 rows=16044 width=8) (actual time=2.174..4.364 rows=16044 loops=1)
32       > Sort (cost=1482.77..1522.90 rows=16049 width=10) (actual time=6.004..8.719 rows=16049 loops=1)
33       Sort Key: p.rental_id
34       Sort Method: quicksort Memory: 1137kB

```

Project 2 queryCommands

```

5
6
7 CREATE INDEX idx_film_cat_catid ON film_category USING HASH(category_id) ;
8 --query 17 , hashjoin is off
Data Output Explain Messages Notifications
QUERY PLAN
text
28 Sort Method: quicksort, Memory: 1137kB
29 -> Merge Join (cost=1483.05. 2341.99 rows=16049 width=10) (actual time=8.194..15.909 rows=16049 loops=1)
   Merge Cond: (r.rental_id = p.rental_id)
30   -> 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)
31   -> Sort (cost=1482.77..1522.90 rows=16049 width=10) (actual time=6.004..8.719 rows=16049 loops=1)
32     Sort Key: p.rental_id
33     Sort Method: quicksort, Memory: 1137kB
34     -> Append (cost=0.00..361.74 rows=16049 width=10) (actual time=0.008..2.116 rows=16049 loops=1)
35       -> Seq Scan on payment_p2022_01 p_1 (cost=0.00..13.23 rows=723 width=10) (actual time=0.008..0.062 rows=723 loops=1)
36       -> Seq Scan on payment_p2022_02 p_2 (cost=0.00..42.01 rows=2401 width=10) (actual time=0.012..0.237 rows=2401 loops=1)
37       -> Seq Scan on payment_p2022_03 p_3 (cost=0.00..47.13 rows=2713 width=10) (actual time=0.023..0.222 rows=2713 loops=1)
38       -> Seq Scan on payment_p2022_04 p_4 (cost=0.00..44.47 rows=2547 width=10) (actual time=0.007..0.188 rows=2547 loops=1)
39       -> Seq Scan on payment_p2022_05 p_5 (cost=0.00..46.77 rows=2677 width=10) (actual time=0.010..0.243 rows=2677 loops=1)
40       -> Seq Scan on payment_p2022_06 p_6 (cost=0.00..46.54 rows=2654 width=10) (actual time=0.006..0.240 rows=2654 loops=1)
41       -> Seq Scan on payment_p2022_07 p_7 (cost=0.00..41.34 rows=2334 width=10) (actual time=0.036..0.287 rows=2334 loops=1)
42 Planning Time: 0.496 ms
43 Execution Time: 28.594 ms

```

Query (17) Justification:

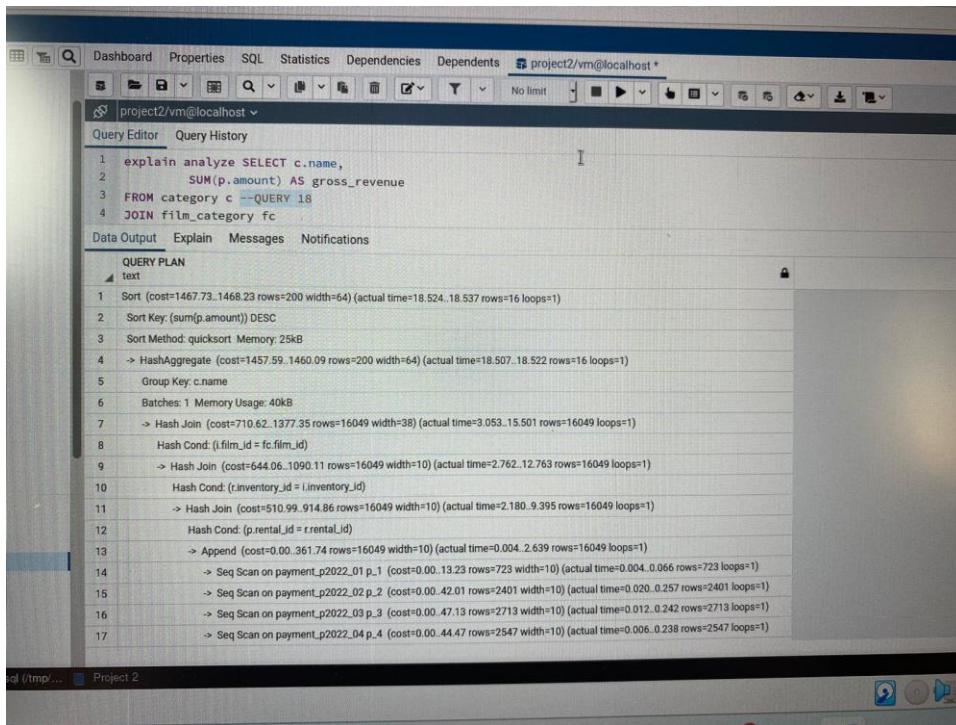
- Index On Attribute: Table-Name = category, Attribute-Name = category_id, we used hash index as we're creating an index on the foreign key so, we're filtering based on the equality of the IDs.
- We turned off the hash Join to make the planner use our index.
- The planner uses our index which results 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. : l.film_id = fc.film_id)
- **Slowest Runtime:** hash join (cond. : l.film_id = fc.film_id)
- **Largest Number of Rows:** hash join (cond. : l.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



The screenshot shows a PostgreSQL Explain Analyze output in a query editor. The query is:

```

1  explain analyze SELECT c.name,
2        SUM(p.amount) AS gross_revenue
3  FROM category c --QUERY 18
4  JOIN film_category fc

```

The Explain output shows a detailed query plan with 17 steps:

- Sort (cost=1467.73..1468.23 rows=200 width=64) (actual time=18.524..18.537 rows=16 loops=1)
- Sort Key: (sum(p.amount)) DESC
- Sort Method: quicksort Memory: 25kB
- > HashAggregate (cost=1457.59..1460.09 rows=200 width=64) (actual time=18.507..18.522 rows=16 loops=1)
- Group Key: c.name
- Batches: 1 Memory Usage: 40kB
- > Hash Join (cost=710.62..1377.35 rows=16049 width=38) (actual time=3.053..15.501 rows=16049 loops=1)
- Hash Cond: (l.film_id = fc.film_id)
- > Hash Join (cost=644.06..1090.11 rows=16049 width=10) (actual time=2.762..12.763 rows=16049 loops=1)
- Hash Cond: (r.inventory_id = i.inventory_id)
- > Hash Join (cost=510.99..914.86 rows=16049 width=10) (actual time=2.180..9.395 rows=16049 loops=1)
- Hash Cond: (p.rental_id = rental_id)
- > Append (cost=0.00..361.74 rows=16049 width=10) (actual time=0.004..2.639 rows=16049 loops=1)
- > Seq Scan on payment_p2022_01 p_1 (cost=0.00..13.23 rows=723 width=10) (actual time=0.004..0.066 rows=723 loops=1)
- > Seq Scan on payment_p2022_02 p_2 (cost=0.00..42.01 rows=2401 width=10) (actual time=0.020..0.257 rows=2401 loops=1)
- > Seq Scan on payment_p2022_03 p_3 (cost=0.00..47.13 rows=2713 width=10) (actual time=0.012..0.242 rows=2713 loops=1)
- > Seq Scan on payment_p2022_04 p_4 (cost=0.00..44.47 rows=2547 width=10) (actual time=0.006..0.238 rows=2547 loops=1)

```

1  explain analyze SELECT c.name,
2      SUM(p.amount) AS gross_revenue
3  FROM category c --QUERY 18
4  JOIN film_category fc
Data Output Explain Messages Notifications
QUERY PLAN
text
17      -> Seq Scan on payment_p2022_04_p_4 (cost=0.00..44.47 rows=2547 width=10) (actual time=0.006..0.238 rows=2547 loops=1)
18      -> Seq Scan on payment_p2022_05_p_5 (cost=0.00..46.77 rows=2677 width=10) (actual time=0.007..0.367 rows=2677 loops=1)
19      -> Seq Scan on payment_p2022_06_p_6 (cost=0.00..46.54 rows=2654 width=10) (actual time=0.148..0.410 rows=2654 loops=1)
20      -> Seq Scan on payment_p2022_07_p_7 (cost=0.00..41.34 rows=2334 width=10) (actual time=0.039..0.311 rows=2334 loops=1)
21      -> Hash (cost=310.44..310.44 rows=16044 width=8) (actual time=2.162..2.162 rows=16044 loops=1)
Buckets: 16384 Batches: 1 Memory Usage: 75kB
23      -> Seq Scan on rental r (cost=0.00..310.44 rows=16044 width=8) (actual time=0.002..0.999 rows=16044 loops=1)
24      -> Hash (cost=75.81..75.81 rows=4581 width=8) (actual time=0.574..0.574 rows=4581 loops=1)
Buckets: 8192 Batches: 1 Memory Usage: 243kB
26      -> Seq Scan on inventory i (cost=0.00..75.81 rows=4581 width=8) (actual time=0.003..0.269 rows=4581 loops=1)
27      -> Hash (cost=54.06..54.06 rows=1000 width=36) (actual time=0.284..0.287 rows=1000 loops=1)
Buckets: 1024 Batches: 1 Memory Usage: 52kB
29      -> Hash Join (cost=35.42..54.06 rows=1000 width=36) (actual time=0.019..0.201 rows=1000 loops=1)
30      Hash Cond: (fc.category_id = c.category_id)
31      -> Seq Scan on film_category fc (cost=0.00..16.00 rows=1000 width=8) (actual time=0.007..0.062 rows=1000 loops=1)
32      -> Hash (cost=21.30..21.30 rows=1130 width=36) (actual time=0.008..0.010 rows=16 loops=1)
Buckets: 2048 Batches: 1 Memory Usage: 17kB
33      -> Seq Scan on category c (cost=0.00..21.30 rows=1130 width=36) (actual time=0.004..0.005 rows=16 loops=1)
34      -> Planning Time: 0.368 ms
35      Execution Time: 18.609 ms

```

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

```

5
6
7 CREATE INDEX idx_HAHS_film_cat_catid on film_category USING HASH(category_id) ;
--query 18 , hashjoin is off
8
9
10
Data Output Explain Messages Notifications
QUERY PLAN
text
1 Sort (cost=4640.25..4640.29 rows=16 width=64) (actual time=20.242..20.250 rows=16 loops=1)
2 Sort Key: (sum(p.amount)) DESC
3 Sort Method: quicksort Memory: 25kB
4 -> HashAggregate (cost=4639.73..4639.93 rows=16 width=64) (actual time=20.230..20.239 rows=16 loops=1)
5 Group Key: c.name
6 Batches: 1 Memory Usage: 24kB
7 -> Merge Join (cost=4295.86..4559.48 rows=16049 width=38) (actual time=15.077..18.121 rows=16049 loops=1)
8   Merge Cond: (l.inventory_id = r.inventory_id)
9     -> Sort (cost=832.91..844.36 rows=4581 width=36) (actual time=1.920..2.055 rows=4581 loops=1)
10   Sort Key: l.inventory_id
11   Sort Method: quicksort Memory: 407kB
12   -> Merge Join (cost=480.64..554.35 rows=4581 width=36) (actual time=0.917..1.542 rows=4581 loops=1)
13     Merge Cond: (fc.film_id = lf.film_id)
14     -> Sort (cost=126.27..128.77 rows=1000 width=36) (actual time=0.303..0.363 rows=1000 loops=1)
15     Sort Key: fc.film_id
16     Sort Method: quicksort Memory: 71kB

```

Project 2 queryCommands

project2/vm@localhost

Query Editor Query History

```

5
6
7 CREATE INDEX idx_HAHS_film_cat_catid on film_category USING HASH(category_id) ;
--query 18 , hashjoin is off
9
10
Data Output Explain Messages Notifications
QUERY PLAN
text
15 SORT KEY: IC.FILM_ID
16 Sort Method: quicksort Memory: 71kB
17 -> Nested Loop (cost=0.00..76.44 rows=1000 width=36) (actual time=0.020..0.178 rows=1000 loops=1)
18   -> Seq Scan on category c (cost=0.00..1.16 rows=16 width=36) (actual time=0.009..0.010 rows=16 loops=1)
19     -> 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=1)
20       Index Cond: (category_id = c.category_id)
21     -> Sort (cost=354.37..365.82 rows=4581 width=8) (actual time=0.611..0.741 rows=4581 loops=1)
22       Sort Key: f.film_id
23       Sort Method: quicksort Memory: 407kB
24     -> Seq Scan on inventory i (cost=0.00..75.81 rows=4581 width=8) (actual time=0.003..0.290 rows=4581 loops=1)
25   -> Sort (cost=3462.94..3503.05 rows=16049 width=10) (actual time=13.152..14.262 rows=16049 loops=1)
26     Sort Key: r.inventory_id
27     Sort Method: quicksort Memory: 1137kB
28   -> Merge Join (cost=1483.06..2341.90 rows=16049 width=10) (actual time=4.616..10.333 rows=16049 loops=1)
29     Merge Cond: (r.rental_id = p.rental_id)
30     -> Index Scan using rental_pk 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

The screenshot shows the MySQL Workbench Query Editor. The top tab bar has 'Query Editor' selected. Below the tabs, there is a code editor with the following SQL command:

```

5
6
7 CREATE INDEX idx_HAHS_film_cat_catid ON film_category USING HASH(category_id) ;
8 --query 18 , hashjoin is off
9
10

```

Below the code editor is a 'QUERY PLAN' section with the 'text' tab selected. It displays the execution plan for the query, which includes a Merge Join operation and various sequential scans on different tables.

At the bottom of the window, there is a toolbar with icons for Project, New, Open, Save, Import, Export, and other database management functions. The system tray at the bottom of the screen shows the Windows taskbar with icons for search, file explorer, control panel, and other applications like Microsoft Teams and Google Chrome.

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

```

project2/vm@localhost ~
Dashboard Properties SQL Statistics Dependencies Dependents project2/vm@localhost *
Query Editor Query History
1 explain analyze SELECT f.film_id,
2      initcap(f.title) AS title, --QUERY 19
3      f.release_year;
Data Output Explain Messages Notifications
QUERY PLAN
text
1 Group (cost=0.75..119777.79 rows=1000 width=157) (actual time=87.044..301.597 rows=1000 loops=1)
2   Group Key: f.film_id, (btmin(l.name):text)
3     -> Incremental Sort (cost=0.75..358.74 rows=1000 width=76) (actual time=85.758..88.548 rows=1000 loops=1)
4       Sort Key: f.film_id, (btmin(l.name):text)
5       Presorted Key: f.film_id
6       Full-sort Groups: 32 Sort Method: quicksort Average Memory: 29kB Peak Memory: 29kB
7         -> Nested Loop (cost=0.43..313.74 rows=1000 width=76) (actual time=85.696..87.994 rows=1000 loops=1)
8           -> Index Scan using film_pkey on film f (cost=0.28..93.37 rows=1000 width=48) (actual time=0.036..0.388 rows=1000 loops=1)
9           -> Index Scan using language_pkey on language l (cost=0.15..0.22 rows=1 width=88) (actual time=0.001..0.001 rows=1 loops=1000)
10          Index Cond: (language_id = f.language_id)
11 SubPlan 2
12   -> Result (cost=16.48..16.49 rows=1 width=32) (actual time=0.004..0.005 rows=1 loops=1000)
13     InitPlan 1 (returns $2)
14       -> Nested Loop (cost=0.43..16.48 rows=1 width=32) (actual time=0.003..0.003 rows=1 loops=1000)
15         -> Index Only Scan using film_category_pkey on film_category fc (cost=0.28..8.29 rows=1 width=4) (actual time=0.001..0.002 rows=1 loops=1000)
16         Index Cond: (film_id = fc.film_id)
17         Heap Fetches: 1000
18         -> Index Scan using category_pkey on category c (cost=0.15..8.17 rows=1 width=36) (actual time=0.001..0.001 rows=1 loops=1000)
19 Data Output
20 Explain
21 Messages
22 Notifications
QUERY PLAN
text
19   Index Cond: (category_id = fc.category_id)
20 SubPlan 4
21   -> Result (cost=102.90..102.91 rows=1 width=32) (actual time=0.206..0.206 rows=1 loops=1000)
22     InitPlan 3 (returns $4)
23       -> Hash Join (cost=98.34..102.90 rows=5 width=32) (actual time=0.183..0.202 rows=5 loops=1000)
24         Hash Cond: (a.actor_id = fa.actor_id)
25           -> Seq Scan on actor a (cost=0.00..4.00 rows=200 width=17) (actual time=0.001..0.011 rows=200 loops=997)
26           -> Hash (cost=98.28..98.28 rows=5 width=4) (actual time=0.175..0.175 rows=5 loops=1000)
27             Buckets: 1024 Batches: 1 Memory Usage: 9kB
28               -> Seq Scan on film_actor fa (cost=0.00..98.28 rows=5 width=4) (actual time=0.032..0.173 rows=5 loops=1000)
29                 Filter: (film_id = f.film_id)
30                 Rows Removed by Filter: 5457
31 Planning Time: 0.410 ms
32 JIT:
33   Functions: 34
34   Options: Inlining false, Optimization false, Expressions true, Deforming true
35   Timing: Generation 3.332 ms, Inlining 0.000 ms, Optimization 11.252 ms, Emission 71.958 ms, Total 86.543 ms
36 Execution Time: 352.745 ms

```

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

```

5 set enable_indexscan = off;
6
7 CREATE INDEX idx_language_lanid on language USING HASH(language_id) ;
--query 19 , indexscan is off
9
10
Data Output   Explain   Messages   Notifications
```

QUERY PLAN

text

```

1 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
  4 -> 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)
    6 -> Seq Scan on film f (cost=0.00.. 65.00 rows=1000 width=48) (actual time=0.008.. 0.102 rows=1000 loops=1)
    7 -> 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
      9 -> Seq Scan on language l (cost=0.00.. 1.06 rows=6 width=88) (actual time=20.830.. 20.833 rows=6 loops=1)
10 SubPlan 2
  11 -> Result (cost=9.52.. 9.54 rows=1 width=32) (actual time=0.014.. 0.014 rows=1 loops=1000)
  12 InitPlan 1 (returns $1)
  13 -> 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)
    15 -> Seq Scan on category c (cost=0.00.. 1.16 rows=16 width=36) (actual time=0.000.. 0.001 rows=16 loops=1000)
    16 -> Hash (cost=8.30.. 8.30 rows=1 width=4) (actual time=0.005.. 0.005 rows=1 loops=1000)
```

mes.sql (tmp... Project 2 queryCommands

Query Editor Query History

```

5 set enable_indexscan = off;
6
7 CREATE INDEX idx_language_lanid on language USING HASH(language_id) ;
--query 19 , indexscan is off
9
10
Data Output   Explain   Messages   Notifications
```

QUERY PLAN

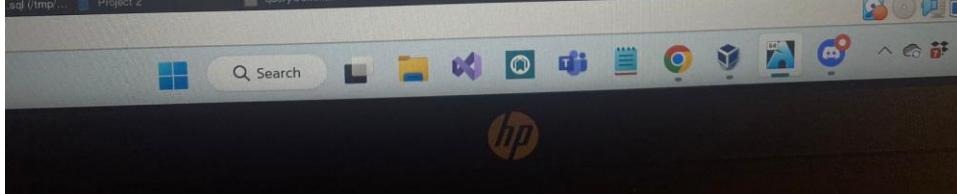
text

```

15 -> Seq Scan on category c (cost=0.00.. 1.16 rows=16 width=36) (actual time=0.000.. 0.001 rows=16 loops=1000)
16 -> Hash (cost=8.30.. 8.30 rows=1 width=4) (actual time=0.005.. 0.005 rows=1 loops=1000)
17 Buckets: 1024 Batches: 1 Memory Usage: 9kB
18 -> 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)
20 Heap Blocks: exact=1000
21 -> 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)
22 Index Cond: (film_id = f.film_id)
23 SubPlan 4
24 -> Result (cost=102.90.. 102.91 rows=1 width=32) (actual time=0.213.. 0.213 rows=1 loops=1000)
25 InitPlan 3 (returns $3)
26 -> Hash Join (cost=98.34.. 102.90 rows=5 width=32) (actual time=0.191.. 0.209 rows=5 loops=1000)
27 Hash Cond: (a.actor_id = fa.actor_id)
28 -> Seq Scan on actor a (cost=0.00.. 4.00 rows=200 width=17) (actual time=0.002.. 0.010 rows=200 loops=997)
29 -> Hash (cost=98.28.. 98.28 rows=5 width=4) (actual time=0.183.. 0.183 rows=5 loops=1000)
30 Buckets: 1024 Batches: 1 Memory Usage: 9kB
31 -> Scan on film_actor fa (cost=0.00.. 0.00 rows=5 width=4) (actual time=0.001.. 0.001 rows=5 loops=1000)
```

project2-queries.sql (tmp... Project 2 queryCommands

```
5 set enable_indexscan = off;
6
7 CREATE INDEX idx_language_lanid on language USING HASH(language_id) ;
8 --query 19 , indexscan is off
9
10
Data Output Explain Messages Notifications
QUERY PLAN
text
24 -> Result (cost=102.90..102.91 rows=1 width=32) (actual time=0.213..0.213 rows=1 loops=1000)
25   InitPlan 3 (returns $3)
26     -> Hash Join (cost=98.34..102.90 rows=5 width=32) (actual time=0.191..0.209 rows=5 loops=1000)
27       Hash Cond: (a.actor_id = fa.actor_id)
28         -> Seq Scan on actor a (cost=0.00..4.00 rows=200 width=17) (actual time=0.002..0.010 rows=200 loops=997)
29           -> Hash (cost=98.28..98.28 rows=5 width=4) (actual time=0.183..0.183 rows=5 loops=1000)
30             Buckets: 1024 Batches: 1 Memory Usage: 9kB
31               -> Seq Scan on film_actor fa (cost=0.00..98.28 rows=5 width=4) (actual time=0.034..0.181 rows=5 loops=1000)
32                 Filter: (film_id = f.film_id)
33                 Rows Removed by Filter: 5457
34   Planning Time: 0.264 ms
35   JIT:
36     Functions: 45
37   Options: Inlining false, Optimization false, Expressions true, Deforming true
38   Timing: Generation 2.186 ms, Inlining 0.000 ms, Optimization 0.642 ms, Emission 19.941 ms, Total 22.770 ms
39   Execution Time: 253.669 ms
```



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.