

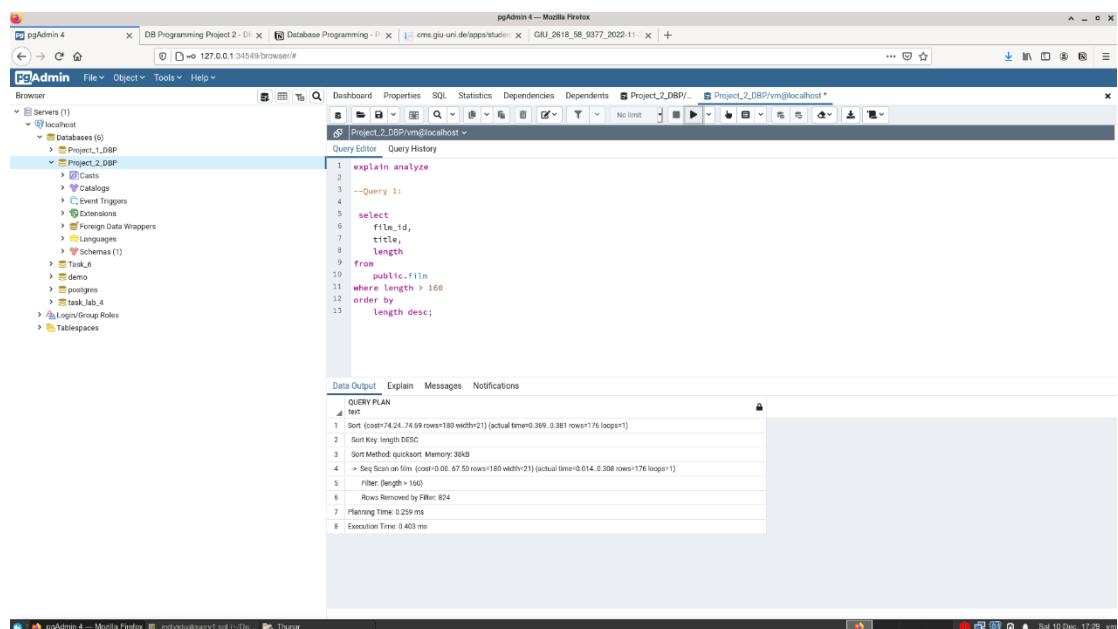
Database Programming

Project 2

Query 1:

For query 1 we used an index on the (length) attribute using BTREE, because it was the only attribute in the query used in the where clause and the order by clause, we would apply an index on it and using a btree would be the best index as its more flexible for ranges of numbers.

Before:

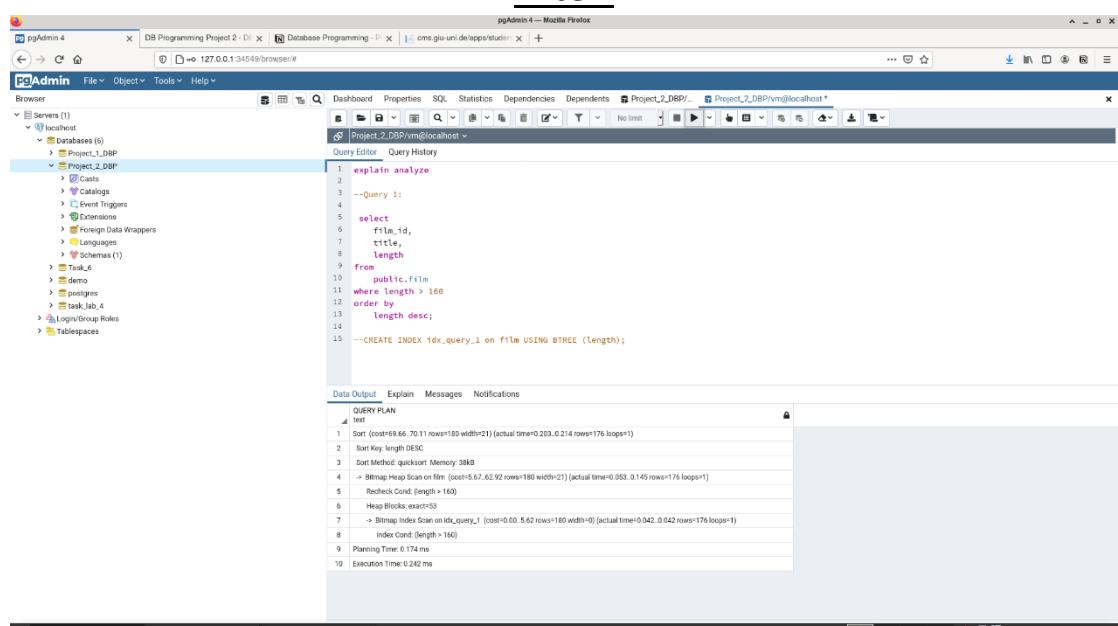


```
explain analyze
--Query 1:
select
film_id,
title,
length
from
public.film
where length > 160
order by
length desc;
```

QUERY PLAN

```
text
1 Sort (cost=74.24..74.69 rows=180 width=21) (actual time=0.369..0.381 rows=176 loops=1)
  2 Sort Key length DE(9)
  3 Sort Method: quicksort Memory: 39KB
  4 -> Seq Scan on film (cost=0.00..47.59 rows=180 width=21) (actual time=0.014..0.308 rows=176 loops=1)
      Filter: (length > 160)
      Rows Removed by Filter: 824
    Planning Time: 0.259 ms
  Execution Time: 0.403 ms
```

After:



```
explain analyze
--Query 1:
select
film_id,
title,
length
from
public.film
where length > 160
order by
length desc;
--CREATE INDEX idx_query_1 on film USING BTREE (length);
```

QUERY PLAN

```
text
1 Sort (cost=94.66..70.11 rows=180 width=21) (actual time=0.203..0.214 rows=176 loops=1)
  2 Sort Key length DE(9)
  3 Sort Method: quicksort Memory: 39KB
  4 -> Bitmap Heap Scan on film (cost=5.67..62.92 rows=180 width=21) (actual time=0.053..0.145 rows=176 loops=1)
      Recheck Cond: (length > 160)
      Heap Blocks: external=53
    7 -> Bitmap Index Scan on idx_query_1 (cost=0.00..5.62 rows=180 width=0) (actual time=0.042..0.042 rows=176 loops=1)
        Index Cond: (length > 160)
      Planning Time: 0.174 ms
    8 Execution Time: 0.242 ms
```

Commands:

The commands we used are:

- CREATE INDEX idx_query_1 on film USING BTREE (length);

Comparison:

	Before	After
Highest Cost	Sort = 7.19	Bitmap heap scan on film = 57.3
Slowest Runtime	Sort = 0.073 ms	Bitmap heap scan on film = 0.103 ms
Largest Number of Rows	Sort/Seqscan = 180	Sort/Bitmap index and Heap scan = 180
Planning Time	0.259 ms	0.174 ms
Execution Time	0.403 ms	0.242 ms

- The comparison of the physical plans shows that the index made some changes on the cost and the time and made it less than the older results of the physical plan also the bitmap index and the bitmap heap was included in the physical plan for better indexing and optimization.

Query 2:

For query 2 we used an index on the (category_id, film_id, name) attributes using BTREE on all three of them for better optimization and because those attributes would be used frequently, and they are used in the join clauses.

Before:

The screenshot shows the pgAdmin 4 interface with the following details:

- Left Panel (Browser):** Shows the project structure with databases (Project_1_DBP, Project_2_DBP), tables (Casts, Categories, Event Triggers, Extensions, Foreign Data Wrappers, Languages, Schemas, Task_A, demo, postages, Task_Job_A, Logins/Group Roles, Tablespaces).
- Top Bar:** Includes File, Object, Tools, Help menus, and a connection bar (Project_2_DBP/vm@localhost).
- Central Area:**
 - Query Editor:** Displays the SQL query:

```
1 explain analyze
2
3 --Query 2:
4 select
5   category.name,
6   count(category.name) category_count
7   from
8   category
9   left join film_category on
10    category.category_id = film_category.category_id
11   left join film on
12    film_category.film_id = film.film_id
13   group by
14     category.name
15   order by
16     category_count desc;
```
 - Data Plan:** A table showing the execution plan with steps like Sort, HashAggregate, HashJoin, and SeqScan.
- Bottom Status Bar:** Shows the planning and execution times (Planning Time: 85.880 ms, Execution Time: 16.426 ms).

After:

The screenshot shows the pgAdmin 4 interface with the following details:

- Servers:** localhost (selected)
- Databases:** Project_1_DBP, Project_2_DBP (selected)
- Tables:** Casts, Catalogs, Categories, Countries, Extensions, Foreign Data Wrappers, Languages, Schemas
- Query Editor:**

```

3 select
4     category.name,
5     count(category.name) category_count
6   from
7     category
8   left join film_category on
9     category.category_id = film_category.category_id
10  left join film on
11    film_category.film_id = film.film_id
12  group by
13    category.name
14  order by
15    category_count desc;
16
17 --CREATE INDEX idx_cat_id on category USING BTREE (category_id);
18 --CREATE INDEX idx_filmcat_id on film_category USING BTREE (film_id);
19 --CREATE INDEX idx_cat_name on category USING BTREE (name)

```
- Query Plan:**

```

QUERY PLAN
1  Sort  (cost=29.15..26.19 rows=16 width=40) [actual time=0.846..0.848 rows=16 loops=1]
  1. Sort Key: (count(category.name)) DESC
  2. Sort Method: quicksort Memory: 254B
  3. HashAggregate (cost=29.07..25.83 rows=16 width=40) [actual time=0.835..0.840 rows=16 loops=1]
    4. > HashAggregate (cost=29.07..25.83 rows=16 width=40) [actual time=0.835..0.840 rows=16 loops=1]
      5. Group Key: category.name
      6. Batches: 1 Memory Usage: 24kB
    7. > Hash Right Join (cost=1..38..26.67 rows=1000 width=32) [actual time=0.035..0.487 rows=1000 loops=1]
      8. Hash Cond: (film_category.category_id = category.category_id)
      9. > Hash Scan on film_category (cost=0..0..16.00 rows=1000 width=8) [actual time=0.007..0.124 rows=1000 loops=1]
        10. > Hash (cost=1..16..1.16 rows=16 width=8) [actual time=0.016..0.016 rows=16 loops=1]
        11. Buckets: 1024 Batches: 1 Memory Usage: 9kB
        12. > Seq Scan on category (cost=0..0..1.16 rows=16 width=36) [actual time=0.007..0.010 rows=16 loops=1]
    13. Planning Time: 0.282 ms
  14. Execution Time: 0.874 ms

```

Commands:

The commands we used are:

- CREATE INDEX idx_cat_id on category USING BTREE (category_id);
- CREATE INDEX idx_filmcat_id on film_category USING BTREE (film_id);
- CREATE INDEX idx_cat_name on category USING BTREE (name)

Comparison:

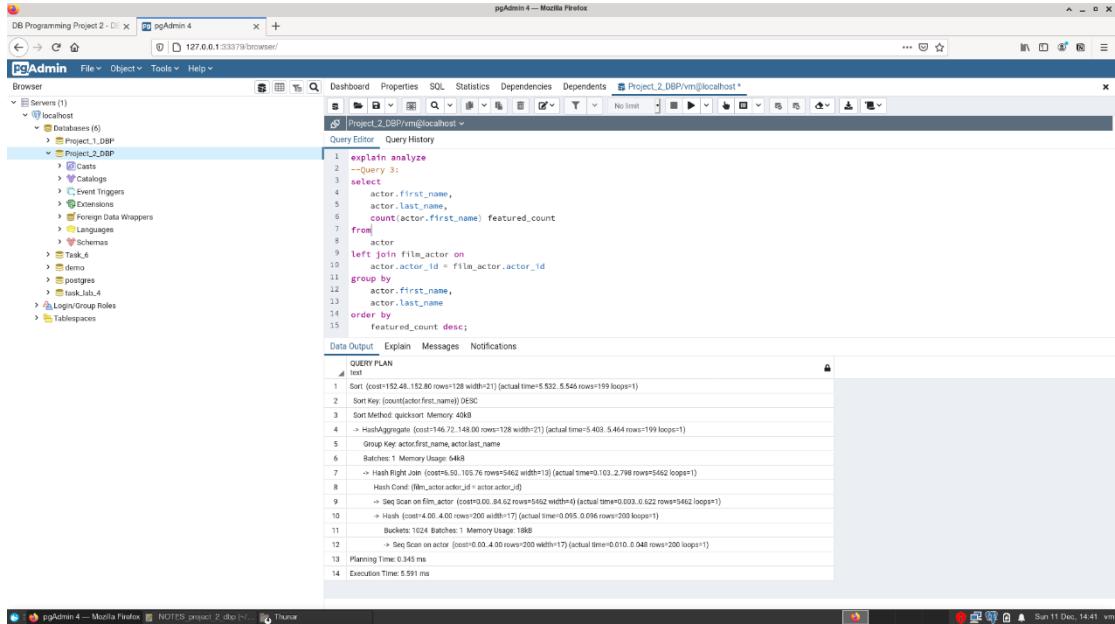
	Before	After
Highest Cost	Hash right join = 7.65	Hash agg = 5.16
Slowest Runtime	Hash right join = 15.046 ms	Hash right join = 0.363 ms
Largest Number of Rows	Hash right join/Hash/seq scan on category = 1130	Hash right join = 1000
Planning Time	85.880 ms	0.282 ms
Execution Time	16.426 ms	0.874 ms

- The comparison between the physical plans showing that the cost and the time are less than how they were before this shows that the index has optimized the query performance and no indexing techniques were added for the optimization process.

Query 3:

For query 3 we used an index on the (actor_id, first_name, last_name) attributes using BTREE on all three of them. Using btree for better optimization on those attributes that were used in the join and the group by clauses.

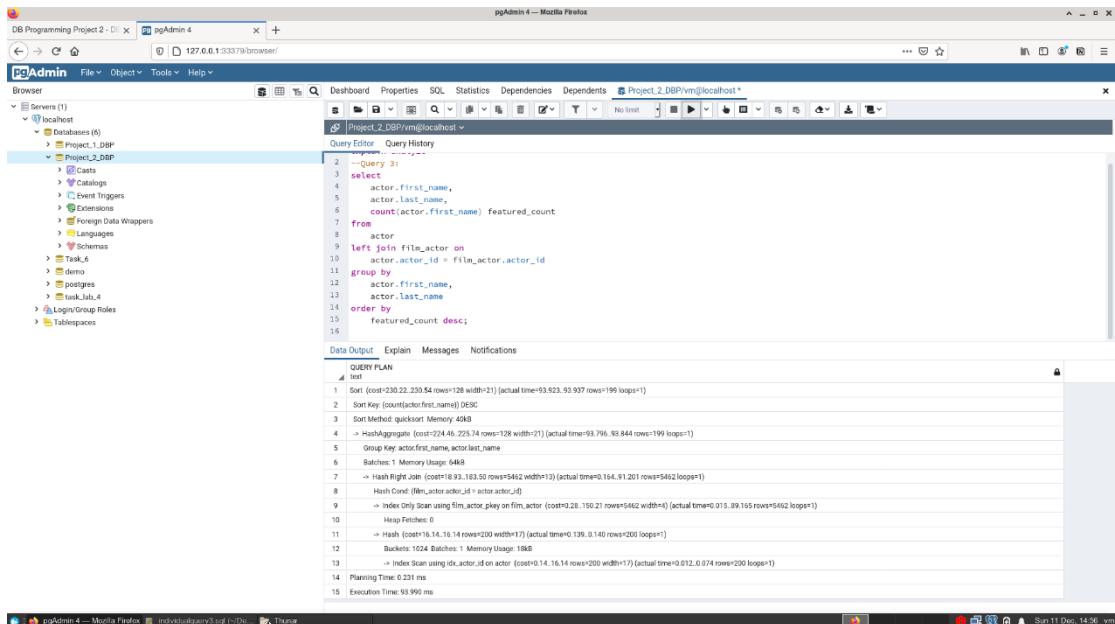
Before:



The screenshot shows the pgAdmin 4 interface with the 'Before' state of the execution plan for Query 3. The 'Data Output' tab is selected, displaying the results of the query. The 'QUERY PLAN' tab shows the following execution plan:

```
1  Sort (cost=152.48, 152.80 rows=128 width=21) (actual timer=5.532, 5.545 rows=199 loops=1)
2  Sort Key (count(actor.first_name)) DESC
3  Sort Method: quicksort Memory: 4048
4  > HashAggregate (cost=146.72, 148.00 rows=128 width=21) (actual timer=5.403, 5.464 rows=199 loops=1)
5  Group Key actor.first_name, actor.last_name
6  Batches: 1 Memory Usage: 64KB
7  >> Hash Right Join (cost=55.105, 76.76 rows=5462 width=13) (actual time=0.103, 2.798 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, 0.622 rows=5462 loops=1)
10 >> Hash (cost=1.00, 4.00 rows=200 width=17) (actual time=0.095, 0.096 rows=200 loops=1)
11 Buckets: 1024 Batches: 1 Memory Usage: 198B
12 >> Seq Scan on actor (cost=0.00, 4.00 rows=200 width=17) (actual time=0.010, 0.048 rows=200 loops=1)
13 Planning Time: 0.445 ms
14 Execution Time: 5.591 ms
```

After:



The screenshot shows the pgAdmin 4 interface with the 'After' state of the execution plan for Query 3. The 'Data Output' tab is selected, displaying the results of the query. The 'QUERY PLAN' tab shows the following execution plan:

```
1  Sort (cost=230.22, 239.54 rows=128 width=21) (actual timer=93.923, 93.937 rows=199 loops=1)
2  Sort Key (count(actor.first_name)) DESC
3  Sort Method: quicksort Memory: 4048
4  > HashAggregate (cost=224.48, 225.74 rows=128 width=21) (actual timer=93.796, 93.844 rows=199 loops=1)
5  Group Key actor.first_name, actor.last_name
6  Batches: 1 Memory Usage: 64KB
7  >> Hash Right Join (cost=93.183, 180.50 rows=5462 width=13) (actual time=0.164, 51.201 rows=5462 loops=1)
8  Hash Cond: (film_actor.actor_id = actor.actor_id)
9  >> Index Only Scan using film_actor_idx on film_actor (cost=0.28, 150.21 rows=5462 width=4) (actual time=0.015, 89.105 rows=5462 loops=1)
10 >> Heap Fetches: 0
11 >> Hash (cost=16.14, 16.14 rows=200 width=17) (actual time=0.139, 0.140 rows=200 loops=1)
12 Buckets: 1024 Batches: 1 Memory Usage: 198B
13 >> Index Scan using idx_actor_id on actor (cost=0.14, 16.14 rows=200 width=17) (actual time=0.012, 0.074 rows=200 loops=1)
14 Planning Time: 0.231 ms
15 Execution Time: 93.990 ms
```

Commands:

The commands we used are:

- CREATE INDEX idx_actor_id on actor USING BTREE (actor_id);
- CREATE INDEX idx_act_fname on actor USING BTREE (first_name);
- CREATE INDEX idx_act_lname on actor USING BTREE (last_name)

Comparison:

	Before	After
Highest Cost	Hash agg = 42.24	Hash agg = 42.24
Slowest Runtime	Hash agg = 2.666 ms	Hash agg = 2.643 ms
Largest Number of Rows	Hash right join/seq scan on film_actor = 5462	Hash right join/index only scan film_actor = 5462
Planning Time	0.345 ms	0.231 ms
Execution Time	5.591 ms	93.990 ms

- The comparison between the physical plans showing that the cost and the time are higher than how they were in the before physical plan this is maybe because of the background processes that were happening also what is observed is that the seqscan technique was exchanged with the index only scan technique.

Query 4:

For query 4 we used an index on the (category_id, film_id, name, rental_duration) attributes using BTREE on all four of them. For better optimization of the query and for more flexibility in searching on the attributes that were used in the where and the joins clauses.

Before:

```

1 explain analyze
2 --Query 4:
3 SELECT t1.name, t1.standard_quartile, COUNT(t1.standard_quartile)
Data Output Explain Messages Notifications
QUERY PLAN
text
1 Sort (cost=136.66, 137.16 rows=120 width=44) (actual time=1.600..1.604 rows=24 loops=1)
2 Sort Key c.name, milid4 OVER(?) 
3 Sort Method: quicksort Memory: 24kB
4 Sort Method: quicksort (cost=127.01..129.01 rows=200 width=44) (actual time=1.542..1.550 rows=24 loops=1)
5 Group Key c.name, milid4 OVER(?) 
6 Batches: 1 Memory Usage: 40kB
7 WindowAgg (cost=113.89..124.45 rows=975 width=70) (actual time=1.512..1.493 rows=951 loops=1)
8   > Sort (cost=113.89..114.83 rows=975 width=34) (actual time=1.127..1.150 rows=951 loops=1)
9     Sort Key f.title, duration
10    Sort Method: quicksort Memory: 41kB
11      > Hash Join (cost=24.56..97.88 rows=975 width=34) (actual time=0.559..1.510 rows=951 loops=1)
12        Hash Cond: (f.film_id = fc.film_id)
13          > Seq Scan on film_category fc (cost=0.00..65.00 rows=1000 width=16) (actual time=0.007..0.187 rows=1000 loops=1)
14            > Hash (cost=20.67..20.67 rows=975 width=36) (actual time=0.538..0.539 rows=951 loops=1)
15              Buckets: 1024 Batches: 1 Memory Usage: 24kB
16                > Hash Join (cost=1.31..20.67 rows=975 width=36) (actual time=0.046..0.436 rows=951 loops=1)
17                  Hash Cond: (fc.category_id = category_id)
18                    > Seq Scan on film_category fc (cost=0.00..16.00 rows=1000 width=8) (actual time=0.010..0.144 rows=1000 loops=1)
19                      > Hash (cost=1.28..1.28 rows=975 width=30) (actual time=0.022..0.022 rows=951 loops=1)
20                        Buckets: 1024 Batches: 1 Memory Usage: 9kB
21                          > Seq Scan on category c (cost=0.00..1.28 rows=6 width=36) (actual time=0.009..0.016 rows=6 loops=1)
22                            Filter: (name = ANY ('Animation,Children,Classics,Comedy,Family,Musical'))::text[]
23                            Rows Removed by Filter: 10
24 Planning Time: 0.702 ms
25 Execution Time: 1.708 ms
  
```

After (2 screenshots):

```

1 explain analyze
2 --Query 4:
3 SELECT t1.name, t1.standard_quartile, COUNT(t1.standard_quartile)
Data Output Explain Messages Notifications
QUERY PLAN
text
1 Group Key c.name, milid4 OVER(?) 
2 Batches: 1 Memory Usage: 40kB
3 WindowAgg (cost=209.53..216.99 rows=975 width=70) (actual time=1.946..2.147 rows=951 loops=1)
4   > Sort (cost=209.53..210.40 rows=975 width=34) (actual time=1.836..1.897 rows=951 loops=1)
5     Sort Key f.title, duration
6     Sort Method: quicksort Memory: 41kB
7       > Merge Join (cost=92.27..193.49 rows=975 width=34) (actual time=0.901..1.718 rows=951 loops=1)
8         Merge Cond: (f.film_id = fc.film_id)
9           > Index Scan using idx_film_id on film f (cost=0.28..93.37 rows=1000 width=16) (actual time=0.010..0.446 rows=1000 loops=1)
10          > Sort (cost=92.00..92.94 rows=975 width=16) (actual time=0.881..0.915 rows=951 loops=1)
11          Sort Key fc.film_id
12          Sort Method: quicksort Memory: 71kB
13           > Merge Join (cost=11.75..75.97 rows=975 width=36) (actual time=0.502..0.721 rows=951 loops=1)
14             Merge Cond: (fc.category_id = c.category_id)
15               > Sort (cost=65.80..68.33 rows=1000 width=16) (actual time=0.466..0.544 rows=746 loops=1)
16               Sort Key fc.category_id
17               Sort Method: quicksort Memory: 71kB
18               > Seq Scan on film_category fc (cost=0.00..16.00 rows=1000 width=8) (actual time=0.012..0.217 rows=1000 loops=1)
19               > Sort (cost=11.39..11.39 rows=975 width=36) (actual time=0.021..0.022 rows=951 loops=1)
20                 Sort Key c.category_id
21                 Sort Method: quicksort Memory: 29kB
22                   > Seq Scan on category c (cost=0.00..1.28 rows=6 width=36) (actual time=0.007..0.012 rows=6 loops=1)
23                     Filter: (name = ANY ('Animation,Children,Classics,Comedy,Family,Musical'))::text[]
24                     Rows Removed by Filter: 10
25 Planning Time: 0.347 ms
26 Execution Time: 1.420 ms
  
```

```

1  explain analyze
2  --Query 4:
3  SELECT t1.name, t1.standard_quartile, COUNT(t1.standard_quartile)
4
5  QUERY PLAN
6  test
7  1  Sort  (cost=232.29..232.79 rows=200 width=44) (actual time=2,358.2383 rows=24 loops=1)
8    2  Sort Key: c.name, (mtl4@4) OVER(?)()
9    3  Sort Method: quicksort Memory: 724B
10   4  -> HashAggregate  (cost=222.65..224.65 rows=200 width=44) (actual time=2,317.2,326 rows=24 loops=1)
11     5  Group Key: c.name, mtl4@4 OVER(?)()
12     6  Batches: 1 Memory Usage: 404B
13     7  -> WindowAgg  (cost=209.53..216.09 rows=375 width=70) (actual time=1,946.2,147 rows=361 loops=1)
14       8  -> Sort  (cost=209.53..210.45 rows=375 width=34) (actual time=1,836.1,897 rows=361 loops=1)
15         9  Sort Key: rental_duration
16         10  Sort Method: quicksort Memory: 41kB
17         11  -> Merge Join  (cost=12.27..193.49 rows=375 width=34) (actual time=0.901..1,718 rows=361 loops=1)
18           12  Merge Cond: (f.film_id = fc.film_id)
19           13  -> Index Scan using idx_film_id on film f  (cost=0.28..93.37 rows=1000 width=6) (actual time=0.010..0.446 rows=1000 loops=1)
20             14  -> Sort  (cost=92.00..92.94 rows=375 width=36) (actual time=1,881.0,915 rows=361 loops=1)
21               15  Sort Key: fc.film_id
22               16  Sort Method: quicksort Memory: 41kB
23               17  -> Merge Join  (cost=19..75.97 rows=375 width=36) (actual time=0.502..0.721 rows=361 loops=1)
24                 18  Merge Cond: (fc.category_id = c.category_id)
25                 19  -> Sort  (cost=45.85..48.33 rows=1000 width=40) (actual time=0.466..0.544 rows=765 loops=1)
26                   20  Sort Key: fc.category_id
27                   21  Sort Method: quicksort Memory: 71kB
28                     22  -> Seq Scan on film_category fc  (cost=0.00..16.00 rows=1000 width=8) (actual time=0.012..0.217 rows=1000 loops=1)
29                     23  -> Sort  (cost=1.36..1.37 rows=4 width=36) (actual time=0.021..0.022 rows=4 loops=1)
30                       24  Sort Key: category_id
31                       25  Sort Method: quicksort Memory: 71kB
32                         26  -> Seq Scan on category c  (cost=0.00..1.29 rows=4 width=8) (actual time=0.007..0.012 rows=4 loops=1)

```

Commands:

The commands we used are:

- CREATE INDEX idx_cat2_id on category USING BTREE (category_id);
- CREATE INDEX idx_film_id on film USING BTREE (film_id);
- CREATE INDEX idx_cat_name2 on category USING BTREE (name);
- CREATE INDEX idx_rental on film USING BTREE (rental_duration)

Comparison:

	Before	After
Highest Cost	Sort = 16.14	Sort = 16.97
Slowest Runtime	Hash join = 0.284 ms	Merge join = 0.357 ms
Largest Number of Rows	Seq scan on film f/seq scan on film_category = 1000	Index scan using idx_film_id/ sort/seq scan on film_category = 1000
Planning Time	0.702 ms	0.347 ms
Execution Time	1.758 ms	2.420 ms

-The comparison between the physical plans shows that the cost and the time are higher after the optimization because of the many indexing techniques were used on the query for optimization plus on that a hash aggregate technique was added to them that made the cost and time higher than the old results.

Query 5:

For query 5 we used an index on the (customer_id, first_name, last_name, amount, payment_date) attributes using BTREE on all except (payment_date) which we used HASH. Using btree on them for faster flexible searching and using the hash on the payment_date attribute to target the date immediately in o(1). Attributes were used in the where and join clauses.

Before:

The screenshot shows the pgAdmin 4 interface with the 'Query Editor' tab selected. The query being run is:

```
--query 5;
explain analyze
SELECT DATE_TRUNC('month', p.payment_date) pay_month, c.first_name || ' ' || c.last_name AS full_name, COUNT(p.amount) AS pay_countpermon, SUM(p.amount) AS pay_amount
```

The 'QUERY PLAN' pane displays the execution plan:

```
1 Sort (cost=0.03..0.04 rows=1 width=80) (actual time=0.023..0.020 rows=0 loops=1)
  2 Sort Key ((c.first_name || ' ' || c.last_name)),(date_trunc(month:text,p.payment_date)),(count(p.amount))
  3 Sort Method: quicksort Memory: 2KB
  4 HashAggregate (cost=0.00..0.02 rows=1 width=80) (actual time=0.002..0.003 rows=0 loops=1)
    5 Group Key ((c.first_name || ' ' || c.last_name),date_trunc(month:text,p.payment_date))
    6 Batches: 1 Memory Usage: 248B
    7 > Result (cost=0.00..0.00 rows=0 width=52) (actual time=0.001..0.001 rows=0 loops=1)
      8 One-Time Filter
      9 Planning Time: 0.646 ms
     10 Execution Time: 0.010 ms
```

After:

The screenshot shows the pgAdmin 4 interface with the 'Query Editor' tab selected. The query remains the same as before:

```
--query 5;
explain analyze
SELECT DATE_TRUNC('month', p.payment_date) pay_month, c.first_name || ' ' || c.last_name AS full_name, COUNT(p.amount) AS pay_countpermon, SUM(p.amount) AS pay_amount
```

The 'QUERY PLAN' pane shows a significant improvement in the execution plan:

```
1 --query 5;
2 explain analyze
3 SELECT DATE_TRUNC('month', p.payment_date) pay_month, c.first_name || ' ' || c.last_name AS full_name, COUNT(p.amount) AS pay_countpermon, SUM(p.amount) AS pay_amount
4 FROM customer c
5 JOIN payment p
6 ON p.customer_id = c.customer_id
7 WHERE c.first_name || ' ' || c.last_name IN
8 ('John', 'Tom', 'Mike')
9 ORDER BY c.first_name
10 Planning Time: 0.387 ms
11 Execution Time: 0.030 ms
```

Commands:

The commands we used are:

- CREATE INDEX idx_cus_id on customer USING BTREE (customer_id);
- CREATE INDEX idx_fir_name on customer USING BTREE (first_name);
- CREATE INDEX idx_las_name on customer USING BTREE (last_name);
- CREATE INDEX idx_amount on payment USING BTREE (amount);
- CREATE INDEX idx_payment_date on payment USING HASH (payment_date)
- set enable_hashagg = off;

Comparison:

	Before	After
Highest Cost	Sort/hash agg = 0.02	Group agg/sort = 0.02
Slowest Runtime	Sort = 0.17 ms	Sort = 0.005 ms
Largest Number of Rows	Sort/Hash agg = 1	Sort/Group agg = 1
Planning Time	0.646 ms	0.387 ms
Execution Time	0.060 ms	0.030 ms

-The comparison shows that the cost got higher, and the time was less than the old results and instead of using the hash aggregate it used the group aggregate technique.

Query 6:

For query 6 we used an index on the (first_name) attribute using BTREE. Because it's the only attribute was used in a clause and using btree for faster better optimization.

Before:

```

--query 6
explain analyze
SELECT actor_id,
       first_name,
       last_name
FROM actor
WHERE first_name LIKE 'JOE';

CREATE INDEX idx_actor_fname ON actor USING BTREE (first_name);

```

QUERY PLAN

```

1 Seq Scan on actor (cost=0.00..4.50 rows=1 width=17) (actual time=0.015..0.048 rows=1 loops=1)
  Filter: (first_name ~~ 'JOE'::text)
  Rows Removed by Filter: 199
Planning Time: 0.125 ms
Execution Time: 0.099 ms

```

After:

```

--query 6
explain analyze
SELECT actor_id,
       first_name,
       last_name
FROM actor
WHERE first_name LIKE 'JOE';

CREATE INDEX idx_actor_fname ON actor USING BTREE (first_name);

```

QUERY PLAN

```

1 Index Scan using idx_actor_fname on actor (cost=0.14..8.10 rows=1 width=17) (actual time=0.029..0.030 rows=1 loops=1)
  Index Cond: (first_name = 'JOE'::text)
  Filter: (first_name ~~ 'JOE'::text)
  Planning Time: 0.105 ms
  Execution Time: 0.041 ms

```

Commands:

The commands we used are:

- CREATE INDEX idx_actor_fname on actor USING BTREE (first_name)
- set enable_seqscan = off;

Comparison:

	Before	After
Highest Cost	Seq scan = 4.5	Index scan using idx_actor_fname on actor = 8.16
Slowest Runtime	Seq scan = 0.048 ms	Index scan using idx_actor_fname on actor = 0.03 ms
Largest Number of Rows	Seq scan = 1	Index scan = 1
Planning Time	0.125 ms	0.105 ms
Execution Time	0.059 ms	0.041 ms

The comparison shows that the cost got higher, and the time was less and observation that the seqscan was changed into index scan technique.

Query 7:

For query 7 we used an index on the (last_name) attribute using HASH. For faster performance and to target the result in o(1).

Before:

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

QUERY PLAN
1 Seq Scan on actor (cost=0.00, 5.00 rows=1 width=25) (actual time=0.143.0.144 rows=0 loops=1) 2 Filter: ((last_name ~ '%GEN%') AND (last_name = 'Smith')) 3 Rows Removed by Filter: 200 4 Planning Time: 0.328 ms 5 Execution Time: 0.155 ms

After:

```
--Query 7:
1   explain analyze SELECT *
2   FROM public.actor
3   WHERE last_name similar to '%GEN%' and last_name = 'Smith';
4
5   CREATE INDEX idx_last_name on actor USING HASH (last_name)
6
7   set enable_seqscan =off;
```

Data Output Explain Messages Notifications

QUERY PLAN

1 Index Scan using idx_last_name on actor (cost=0.00..8.02 rows=1 width=25) (actual time=0.007..0.008 rows=0 loops=1)

2 Index Cond: (last_name >= 'Smith')

3 Filter: (last_name ~ '%GEN%')::text

4 Planning Time: 0.201 ms

5 Execution Time: 0.018 ms

Successfully run. Total query runtime: 59 msec. 5 rows affected.

Commands:

The commands we used are:

- CREATE INDEX idx_last_name on actor USING HASH (last_name)
- set enable_seqscan = off;

Comparison:

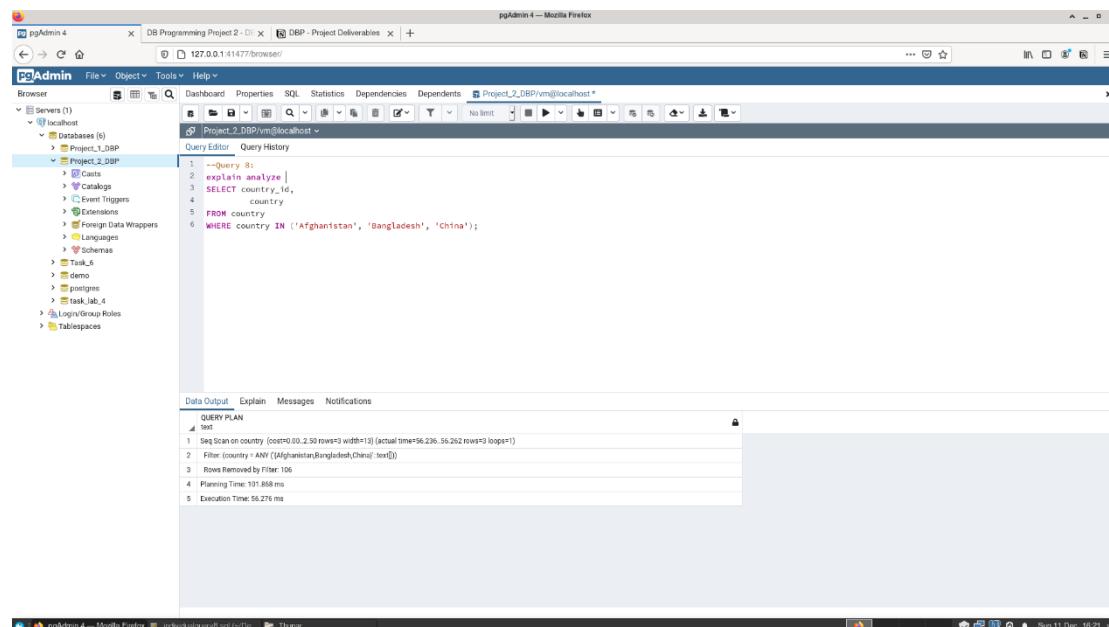
	Before	After
Highest Cost	Seq scan = 5	Index scan using idx_last_name on actor = 8.02
Slowest Runtime	Seq scan = 0.144 ms	Index scan using idx_last_name on actor = 0.008 ms
Largest Number of Rows	Seq scan = 1	Index scan = 1
Planning Time	0.328 ms	0.201 ms
Execution Time	0.155 ms	0.018 ms

-The comparison shows that the cost was higher, and the time was less than before results and instead of using seqscan index scan was used for the optimization.

Query 8:

For query 8 we used an index on the (country) attribute using HASH. For faster performance and to target the result in o(1).

Before:



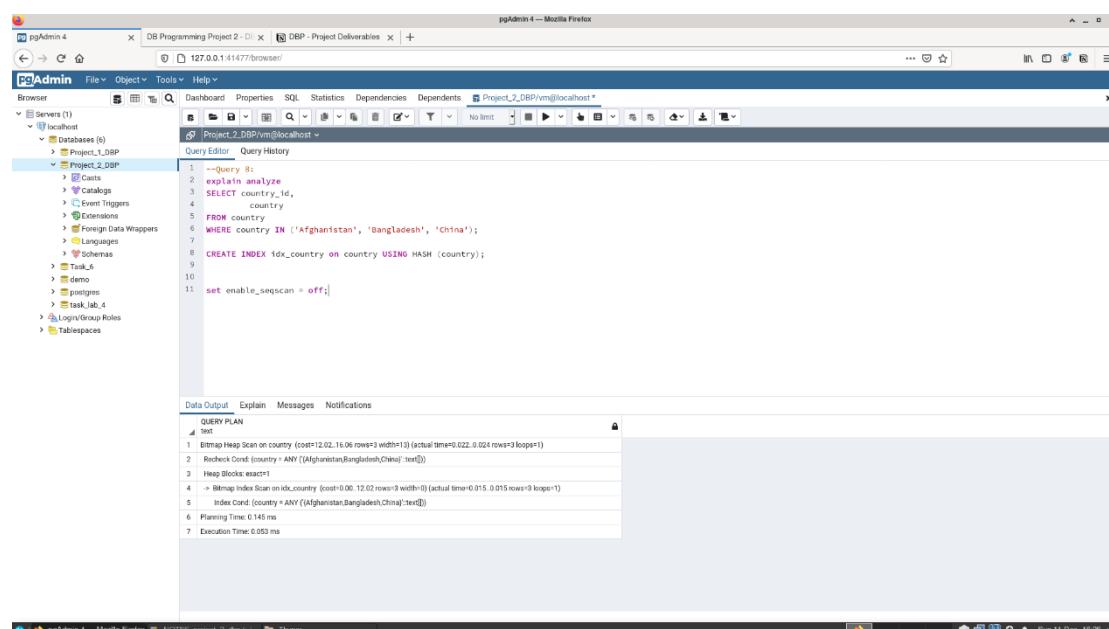
```
--Query 8;
explain analyze
SELECT country_id,
       country
  FROM country
 WHERE country IN ('Afghanistan', 'Bangladesh', 'China');
```

DATA OUTPUT

QUERY PLAN

```
1 Seq Scan on country (cost=0.00..2.50 rows=3 width=13) (actual time=56.236..56.262 rows=3 loops=1)
  2 Filter (country = ANY ('Afghanistan,Bangladesh,China'))
  3 Rows Removed by Filter: 106
  Planning Time: 101.868 ms
  Execution Time: 56.276 ms
```

After:



```
--Query 8;
explain analyze
SELECT country_id,
       country
  FROM country
 WHERE country IN ('Afghanistan', 'Bangladesh', 'China');

CREATE INDEX idx_country ON country USING HASH (country);

set enable_seqscan = off;
```

DATA OUTPUT

QUERY PLAN

```
1 Bitmap Scan on country (cost=12.02..16.06 rows=3 width=13) (actual time=0.022..0.024 rows=3 loops=1)
  2 Redeck Cond (country = ANY ('Afghanistan,Bangladesh,China'))
  3 Heap Blocks: exact1
  4 > Bitmap Index Scan on idx_country (cost=0.00..12.02 rows=3 width=0) (actual time=0.015..0.015 rows=3 loops=1)
  5   Index Cond (country = ANY ('Afghanistan,Bangladesh,China'))
  Planning Time: 0.145 ms
  Execution Time: 0.053 ms
```

Commands:

The commands we used are:

- CREATE INDEX idx_country on country USING HASH (country);
- set enable_seqscan = off;

Comparison:

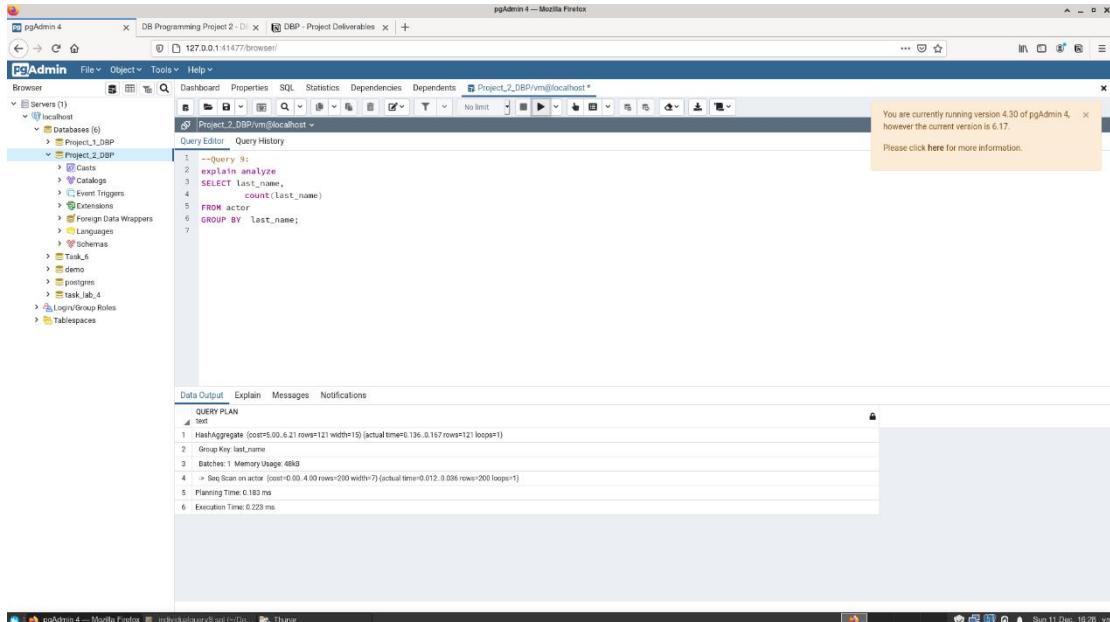
	Before	After
Highest Cost	Seq scan = 2.5	Bitmap heap scan on country = 4.04
Slowest Runtime	Seq scan = 56.262 ms	Bitmap heap scan on country = 0.009 ms
Largest Number of Rows	Seq scan = 3	Bitmap heap scan on country(bitmap index on idx_country) = 3
Planning Time	101.868 ms	0.145 ms
Execution Time	56.276 ms	0.053 ms

-The comparison shows that the cost was higher, and the time was less than before results and instead of using seqscan, bitmap heap was used for the optimization for better results.

Query 9:

For query 9 we used an index on the (last_name) attribute using BTREE. Because it's the only attribute in a clause can be indexed for better optimization.

Before:



```
--query 9:
explain analyze
SELECT last_name,
       count(last_name)
  FROM actor
 GROUP BY last_name;
```

QUERY PLAN

```
1 HashAggregate (cost=6.00..6.21 rows=121 width=15) (actual time=0.136..0.167 rows=121 loops=1)
  2 Group Key last_name
  3 Batches: 1 Memory Usage: 484B
  4 > Seq Scan on actor (cost=0.00..4.00 rows=200 width=7) (actual time=0.012..0.036 rows=200 loops=1)
  5 Planning Time: 0.103 ms
  6 Execution Time: 0.223 ms
```

After:

```

--query 9:
explain analyze
SELECT last_name,
       count(last_name)
FROM actor
GROUP BY last_name;
create index idx_last_nameac on actor using BTREE (last_name);
set enable_hashagg = off;

```

The screenshot shows the pgAdmin 4 interface with a query editor containing the provided SQL code. The code includes an explain analyze statement, a SELECT query to count actors by last name, a create index command for the last_name column, and a set command to disable hash aggregation. Below the code, the 'QUERY PLAN' section shows the execution plan with steps like GroupAggregate, Sort Key last_name, Sort Key last_name, and Seq Scan on actor. The execution time is listed as 0.397 ms.

Commands:

The commands we used are:

- create index idx_last_nameac on actor using BTREE (last_name);
- set enable_hashagg = off;

Comparison:

	Before	After
Highest Cost	Hash agg = 2.21	Sort = 8.14
Slowest Runtime	Hash agg = 0.131 ms	Sort = 0.219 ms
Largest Number of Rows	Seq scan on actor = 200	Sort/seq scan on actor = 200
Planning Time	0.183 ms	0.052 ms
Execution Time	0.223 ms	0.397 ms

-The comparison shows that the cost and the time got higher and the hash aggregate instead of using it the group aggregate was used.

Query 10:

For query 10 we used an index on the (first_name, last_name) attributes using HASH on both of them. For better optimization and fast searching.

Before:

```

--Query 1:
EXPLAIN ANALYZE
SELECT last_name,
       count(last_name)
FROM actor
WHERE first_name like '%AD'
GROUP BY last_name;

```

EXPLAIN PLAN:

```

QUERY PLAN
Text
1 HashAggregate (cost=4.54..4.62 rows=8 width=15) (actual time=0.053..0.055 rows=5 loops=1)
  2 Group Key last_name
  3 Batches: 1 Memory Usage: 24B
  4 > Seq Scan on actor (cost=0.0..4.50 rows=8 width=7) (actual time=0.011..0.048 rows=5 loops=1)
  5   Filter (first_name ~ '%AD')
  6   Rows Removed by Filter: 195
  7 Planning Time: 0.14ms
  8 Execution Time: 0.075 ms

```

After:

```

--Query 1:
EXPLAIN ANALYZE
SELECT last_name,
       count(last_name)
FROM actor
WHERE first_name like '%AD'
GROUP BY last_name;

10 create index idx_frac_name on actor using HASH (first_name);
11 create index idx_lasac_name on actor using HASH (last_name);
12
13 set enable_hashagg = off;

```

EXPLAIN PLAN:

```

QUERY PLAN
Text
1 HashAggregate (cost=4.52..4.76 rows=8 width=15) (actual time=0.080..0.084 rows=5 loops=1)
  2 Group Key last_name
  3 > Sort (cost=4.52..4.64 rows=8 width=7) (actual time=0.075..0.077 rows=5 loops=1)
  4   Sort Key last_name
  5   Sort Method: quicksort Memory: 25kB
  6   > Seq Scan on actor (cost=0.0..4.50 rows=8 width=7) (actual time=0.010..0.047 rows=5 loops=1)
  7     Filter (first_name ~ '%AD')
  8     Rows Removed by Filter: 195
  9   Planning Time: 0.254 ms
10  Execution Time: 0.102 ms

```

Commands:

The commands we used are:

- `create index idx_frac_name on actor using HASH (first_name);`
- `create index idx_lasac_name on actor using HASH (last_name);`
- `set enable_hashagg = off;`

Comparison:

	Before	After
Highest Cost	Hash agg = 4.12	Sort = 4.14
Slowest Runtime	Hash agg = 0.007 ms	Sort = 0.03 ms
Largest Number of Rows	Hash agg/ seq scan on actor = 8	Group agg/sort/seq scan on actor = 8
Planning Time	0.114 ms	0.254 ms
Execution Time	0.075 ms	0.102 ms

-The comparison shows that the cost and the time got higher and the hash aggregate instead of using it the group aggregate was used.

Query 11:

For query 11 we used an index on the (first_name, last_name) attributes using HASH on both of them. For better optimization and fast searching.

Before:

```

pgAdmin 4 — Mozilla Firefox
pgAdmin 4 - Mozilla Firefox | DB Programming Project 2 - D... | DBP - Project Deliverables | + | 127.0.0.1:41477/browser/
PgAdmin File Object Tools Help
Browser Servers (1) Databases (6) Project_1_DBP Project_2_DBP
localhost Project_1_DBP Project_2_DBP
  3 Casts 3 Explain analyze
  3 Catalogs 3 SELECT actor_id
  2 Event Triggers 4 FROM actor
  3 Extensions 5 WHERE first_name = 'HARPO';
  2 Foreign Data Wrappers 6 AND last_name = 'WILLIAMS';
  2 Functions
  1 Indexes
  1 Tables
  1 Tasks
  1 demo
  1 postgres
  2 task_id, 4
> Logins/Group Roles
> Tablespaces

```

You are currently running version 4.30 of pgAdmin 4, however the current version is 6.17.
Please click here for more information.

Query Editor Query History

```

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

```

Data Output Explain Messages Notifications

QUERY PLAN

```

1 Seq Scan on actor (cost=0.00..5.00 rows=1 width=4) (actual time=0.044..0.044 rows=0 loops=1)
2 Filter ((first_name = 'HARPO'::text) AND (last_name = 'WILLIAMS'::text))
3 Rows Removed by Filter: 200
4 Planning Time: 0.134 ms
5 Execution Time: 0.055 ms

```

After:

The screenshot shows the pgAdmin 4 interface. In the left sidebar, under 'Servers (1)', there is one entry for 'localhost'. Under 'localhost', there are several databases listed: 'Project_1_DBP', 'Project_2_DBP', 'task_A', 'demo', 'postgres', 'task_AB', and 'Login/Group Roles'. The 'Project_2_DBP' database is selected. In the main area, the 'Query Editor' tab is active, displaying the following SQL code:

```

1 --query 11
2 explain analyze
3 SELECT actor_id
4   FROM actor
5  WHERE first_name = 'MARPO'
6    AND last_name = 'WILLIAMS';
7
8 drop index idx_firname_act ;
9 drop index idx_lasname_act ;
10
11 set enable_seqscan = off;
12 create index idx_firname_act on actor using BTREE (first_name);
13 create index idx_lasname_act on actor using BTREE (last_name);

```

Below the code, the 'Data Output' tab is selected, showing the results of the query. The 'Explain' tab is also visible, showing the execution plan for the query.

Commands:

The commands we used are:

- create index idx_firname_act on actor using HASH (first_name);
- create index idx_lasname_act on actor using HASH (last_name)
- set enable_seqscan = off;

Comparison:

	Before	After
Highest Cost	Seq scan = 5	Bitmap heap scan on actor = 2.04
Slowest Runtime	Seq scan = 0.044 ms	Bitmap heap scan on actor = 0.014 ms
Largest Number of Rows	Seq scan on actor = 1	Bitmap index on idx_lasname_act = 3
Planning Time	0.134 ms	0.117 ms
Execution Time	0.055 ms	0.118 ms

-The comparison shows that the cost got higher, and the time is less and instead of using the seqscan the bitmap heap was used for better optimization and performance. Sometimes the cost gets higher because of the added new techniques or the background processes that's happening frequently in the machine.

Query 12:

For query 12 we used an index on the (address_id) attribute using BTREE. For better optimization and flexible searching in ranges.

Before:

```
--Query 12;
explain analyze
SELECT s.first_name,
       s.last_name,
       a.address
  FROM staff s
 LEFT JOIN address a
    ON s.address_id = a.address_id;
```

QUERY PLAN

```
1 Hash Left Join (cost=21.57 35.74 rows=530 width=80) (actual time=17.569..17.571 rows=8 loops=1)
  2 Hash Cond (s.address_id = a.address_id)
  3 > Seq Scan on staff s (cost=0.00..13.30 rows=530 width=66) (actual time=16.188..16.189 rows=2 loops=1)
  4 > Hash (cost=14.03..14.03 rows=530 width=24) (actual time=1.311..1.312 rows=530 loops=1)
  5   Buckets: 1024 Batches: 1 Memory Usage: 4KB
  6     -> Seq Scan on address a (cost=0.00..14.03 rows=530 width=24) (actual time=0.465..1.162 rows=530 loops=1)
  7 Planning Time: 35.448 ms
  8 Execution Time: 17.592 ms
```

Successfully run. Total query runtime: 196 msec. 8 rows affected.

After:

```
--Query 12;
explain analyze
SELECT s.first_name,
       s.last_name,
       a.address
  FROM staff s
 LEFT JOIN address a
    ON s.address_id = a.address_id;
create index idx_address on address USING BTREE (address_id);
set enable_hashjoin = off;
```

QUERY PLAN

```
1 Merge Right Join (cost=27.38..49.88 rows=1330 width=80) (actual time=0.087..0.089 rows=2 loops=1)
  2 Merge Cond (a.address_id = s.address_id)
  3   > Index Scan using idx_address on address a (cost=0.28..26.32 rows=1330 width=24) (actual time=0.053..0.054 rows=5 loops=1)
  4   > Sort (cost=27.10..27.93 rows=1330 width=68) (actual time=0.031..0.031 rows=2 loops=1)
  5     Sort Key: a.address_id
  6     Sort Method: quicksort Memory: 25KB
  7     -> Seq Scan on staff s (cost=0.00..13.30 rows=1330 width=66) (actual time=0.007..0.008 rows=2 loops=1)
  8 Planning Time: 0.105 ms
  9 Execution Time: 0.110 ms
```

Commands:

The commands we used are:

- create index idx_address on address USING BTREE (address_id);
- set enable_hashjoin = off;

Comparison:

	Before	After
Highest Cost	Hash left join = 8.41	Sort = 14.63
Slowest Runtime	Hash = 0.15 ms	Merge right join = 0.035 ms
Largest Number of Rows	Seq scan on address a = 603	Merge right join/sort/seq scan on staff s = 330
Planning Time	35.448 ms	0.105 ms
Execution Time	17.592 ms	0.110 ms

-The comparison shows that the cost is higher, and the time is less as the hash left join exchanged to merge right join for better optimization.

Query 13:

For query 13 we used an index on the (staff_id) attribute using BTREE.

For better optimization and fast searching.

Before:

The screenshot shows the pgAdmin 4 interface with the 'Explain' tab selected. The query being analyzed is:

```
--query 13a
explain analyze
SELECT p.staff_id,
       COUNT(p.amount)
FROM payment p
LEFT JOIN staff s
ON p.staff_id = s.staff_id
GROUP BY p.staff_id;
```

The 'EXPLAIN PLAN' section displays the following details:

- 1 HashAggregate (cost=441.98..443.98 rows=200 width=12) (actual time=74.144..74.148 rows=2 loops=1)
- 2 Group Key: p.staff_id
- 3 Batches: 1 Memory Usage: 4096
- 4 Append (cost=0.00..361.74 rows=16049 width=10) (actual time=10.056..68.880 rows=16049 loops=1)
- 5 Seq Scan on payment_p2022_01_p_1 (cost=0.00..12.33 rows=723 width=10) (actual time=10.051..10.674 rows=723 loops=1)
- 6 Seq Scan on payment_p2022_02_p_2 (cost=0.00..42.01 rows=2401 width=10) (actual time=19.052..21.134 rows=2401 loops=1)
- 7 Seq Scan on payment_p2022_03_p_3 (cost=0.00..47.13 rows=2713 width=10) (actual time=45.457..1.458 rows=2713 loops=1)
- 8 Seq Scan on payment_p2022_04_p_4 (cost=0.00..41.47 rows=2547 width=10) (actual time=0.432..1.456 rows=2547 loops=1)
- 9 Seq Scan on payment_p2022_05_p_5 (cost=0.00..46.77 rows=2677 width=10) (actual time=0.365..1.632 rows=2677 loops=1)
- 10 Seq Scan on payment_p2022_06_p_6 (cost=0.00..44.54 rows=2654 width=10) (actual time=0.441..9.508 rows=2654 loops=1)
- 11 Seq Scan on payment_p2022_07_p_7 (cost=0.00..41.54 rows=2534 width=10) (actual time=21.178..22.334 rows=2534 loops=1)
- 12 Planning Time: 162.723 ms
- 13 Execution Time: 74.228 ms

After:

```

pgAdmin 4 — Mozilla Firefox
pgAdmin 4 — Mozilla Firefox
DB Programming Project 2 - D... DBP - Project Deliverables
127.0.0.1:41477/browser/
PgAdmin File Object Tools Help
Servers (1)
localhost
  Databases (6)
    > Project_1_DBP
    > Project_2_DBP
    > Casts
    > Catalogs
    > Checksums
    > Extensions
    > Foreign Data Wrappers
    > Languages
    > Schemas (1)
      > Task_5
      > demo
      > postgres
      > task_lab_4
    > Logins/Group Roles
    > Tablespaces
  Catalogs
  Checksums
  Extensions
  Foreign Data Wrappers
  Languages
  Schemas (1)
  Task_5
  demo
  postgres
  task_lab_4
  Logins/Group Roles
  Tablespaces
  Project_2_DBP/vm@localhost
  Query Editor Query History
  --Query 13:
  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
  38
  39
  40
  41
  42
  43
  44
  45
  46
  47
  48
  49
  50
  51
  52
  53
  54
  55
  56
  57
  58
  59
  60
  61
  62
  63
  64
  65
  66
  67
  68
  69
  70
  71
  72
  73
  74
  75
  76
  77
  78
  79
  80
  81
  82
  83
  84
  85
  86
  87
  88
  89
  90
  91
  92
  93
  94
  95
  96
  97
  98
  99
  100
  101
  102
  103
  104
  105
  106
  107
  108
  109
  110
  111
  112
  113
  114
  115
  116
  117
  118
  119
  120
  121
  122
  123
  124
  125
  126
  127
  128
  129
  130
  131
  132
  133
  134
  135
  136
  137
  138
  139
  140
  141
  142
  143
  144
  145
  146
  147
  148
  149
  150
  151
  152
  153
  154
  155
  156
  157
  158
  159
  160
  161
  162
  163
  164
  165
  166
  167
  168
  169
  170
  171
  172
  173
  174
  175
  176
  177
  178
  179
  180
  181
  182
  183
  184
  185
  186
  187
  188
  189
  190
  191
  192
  193
  194
  195
  196
  197
  198
  199
  200
  201
  202
  203
  204
  205
  206
  207
  208
  209
  210
  211
  212
  213
  214
  215
  216
  217
  218
  219
  220
  221
  222
  223
  224
  225
  226
  227
  228
  229
  230
  231
  232
  233
  234
  235
  236
  237
  238
  239
  240
  241
  242
  243
  244
  245
  246
  247
  248
  249
  250
  251
  252
  253
  254
  255
  256
  257
  258
  259
  260
  261
  262
  263
  264
  265
  266
  267
  268
  269
  270
  271
  272
  273
  274
  275
  276
  277
  278
  279
  280
  281
  282
  283
  284
  285
  286
  287
  288
  289
  290
  291
  292
  293
  294
  295
  296
  297
  298
  299
  300
  301
  302
  303
  304
  305
  306
  307
  308
  309
  310
  311
  312
  313
  314
  315
  316
  317
  318
  319
  320
  321
  322
  323
  324
  325
  326
  327
  328
  329
  330
  331
  332
  333
  334
  335
  336
  337
  338
  339
  340
  341
  342
  343
  344
  345
  346
  347
  348
  349
  350
  351
  352
  353
  354
  355
  356
  357
  358
  359
  360
  361
  362
  363
  364
  365
  366
  367
  368
  369
  370
  371
  372
  373
  374
  375
  376
  377
  378
  379
  380
  381
  382
  383
  384
  385
  386
  387
  388
  389
  390
  391
  392
  393
  394
  395
  396
  397
  398
  399
  400
  401
  402
  403
  404
  405
  406
  407
  408
  409
  410
  411
  412
  413
  414
  415
  416
  417
  418
  419
  420
  421
  422
  423
  424
  425
  426
  427
  428
  429
  430
  431
  432
  433
  434
  435
  436
  437
  438
  439
  440
  441
  442
  443
  444
  445
  446
  447
  448
  449
  450
  451
  452
  453
  454
  455
  456
  457
  458
  459
  460
  461
  462
  463
  464
  465
  466
  467
  468
  469
  470
  471
  472
  473
  474
  475
  476
  477
  478
  479
  480
  481
  482
  483
  484
  485
  486
  487
  488
  489
  490
  491
  492
  493
  494
  495
  496
  497
  498
  499
  500
  501
  502
  503
  504
  505
  506
  507
  508
  509
  510
  511
  512
  513
  514
  515
  516
  517
  518
  519
  520
  521
  522
  523
  524
  525
  526
  527
  528
  529
  530
  531
  532
  533
  534
  535
  536
  537
  538
  539
  540
  541
  542
  543
  544
  545
  546
  547
  548
  549
  550
  551
  552
  553
  554
  555
  556
  557
  558
  559
  559
  560
  561
  562
  563
  564
  565
  566
  567
  568
  569
  569
  570
  571
  572
  573
  574
  575
  576
  577
  578
  579
  579
  580
  581
  582
  583
  584
  585
  586
  587
  588
  589
  589
  590
  591
  592
  593
  594
  595
  596
  597
  598
  599
  599
  600
  601
  602
  603
  604
  605
  606
  607
  608
  609
  609
  610
  611
  612
  613
  614
  615
  616
  617
  618
  619
  619
  620
  621
  622
  623
  624
  625
  626
  627
  628
  629
  629
  630
  631
  632
  633
  634
  635
  636
  637
  638
  639
  639
  640
  641
  642
  643
  644
  645
  646
  647
  648
  649
  649
  650
  651
  652
  653
  654
  655
  656
  657
  658
  659
  659
  660
  661
  662
  663
  664
  665
  666
  667
  668
  669
  669
  670
  671
  672
  673
  674
  675
  676
  677
  678
  679
  679
  680
  681
  682
  683
  684
  685
  686
  687
  688
  689
  689
  690
  691
  692
  693
  694
  695
  696
  697
  698
  699
  699
  700
  701
  702
  703
  704
  705
  706
  707
  708
  709
  709
  710
  711
  712
  713
  714
  715
  716
  717
  718
  719
  719
  720
  721
  722
  723
  724
  725
  726
  727
  728
  729
  729
  730
  731
  732
  733
  734
  735
  736
  737
  738
  739
  739
  740
  741
  742
  743
  744
  745
  746
  747
  748
  749
  749
  750
  751
  752
  753
  754
  755
  756
  757
  758
  759
  759
  760
  761
  762
  763
  764
  765
  766
  767
  768
  769
  769
  770
  771
  772
  773
  774
  775
  776
  777
  778
  779
  779
  780
  781
  782
  783
  784
  785
  786
  787
  788
  789
  789
  790
  791
  792
  793
  794
  795
  796
  797
  798
  799
  799
  800
  801
  802
  803
  804
  805
  806
  807
  808
  809
  809
  810
  811
  812
  813
  814
  815
  816
  817
  818
  819
  819
  820
  821
  822
  823
  824
  825
  826
  827
  828
  829
  829
  830
  831
  832
  833
  834
  835
  836
  837
  838
  839
  839
  840
  841
  842
  843
  844
  845
  846
  847
  848
  849
  849
  850
  851
  852
  853
  854
  855
  856
  857
  858
  859
  859
  860
  861
  862
  863
  864
  865
  866
  867
  868
  869
  869
  870
  871
  872
  873
  874
  875
  876
  877
  878
  879
  879
  880
  881
  882
  883
  884
  885
  886
  887
  888
  889
  889
  890
  891
  892
  893
  894
  895
  896
  897
  898
  899
  899
  900
  901
  902
  903
  904
  905
  906
  907
  908
  909
  909
  910
  911
  912
  913
  914
  915
  916
  917
  918
  919
  919
  920
  921
  922
  923
  924
  925
  926
  927
  928
  929
  929
  930
  931
  932
  933
  934
  935
  936
  937
  938
  939
  939
  940
  941
  942
  943
  944
  945
  946
  947
  948
  949
  949
  950
  951
  952
  953
  954
  955
  956
  957
  958
  959
  959
  960
  961
  962
  963
  964
  965
  966
  967
  968
  969
  969
  970
  971
  972
  973
  974
  975
  976
  977
  978
  979
  979
  980
  981
  982
  983
  984
  985
  986
  987
  988
  989
  989
  990
  991
  992
  993
  994
  995
  996
  997
  998
  999
  999
  1000
  1001
  1002
  1003
  1004
  1005
  1006
  1007
  1008
  1009
  1009
  1010
  1011
  1012
  1013
  1014
  1015
  1016
  1017
  1018
  1019
  1019
  1020
  1021
  1022
  1023
  1024
  1025
  1026
  1027
  1028
  1029
  1029
  1030
  1031
  1032
  1033
  1034
  1035
  1036
  1037
  1038
  1039
  1039
  1040
  1041
  1042
  1043
  1044
  1045
  1046
  1047
  1048
  1049
  1049
  1050
  1051
  1052
  1053
  1054
  1055
  1056
  1057
  1058
  1059
  1059
  1060
  1061
  1062
  1063
  1064
  1065
  1066
  1067
  1068
  1069
  1069
  1070
  1071
  1072
  1073
  1074
  1075
  1076
  1077
  1078
  1079
  1079
  1080
  1081
  1082
  1083
  1084
  1085
  1086
  1087
  1088
  1089
  1089
  1090
  1091
  1092
  1093
  1094
  1095
  1096
  1097
  1098
  1099
  1099
  1100
  1101
  1102
  1103
  1104
  1105
  1106
  1107
  1108
  1109
  1109
  1110
  1111
  1112
  1113
  1114
  1115
  1116
  1117
  1118
  1119
  1119
  1120
  1121
  1122
  1123
  1124
  1125
  1126
  1127
  1128
  1129
  1129
  1130
  1131
  1132
  1133
  1134
  1135
  1136
  1137
  1138
  1139
  1139
  1140
  1141
  1142
  1143
  1144
  1145
  1146
  1147
  1148
  1149
  1149
  1150
  1151
  1152
  1153
  1154
  1155
  1156
  1157
  1158
  1159
  1159
  1160
  1161
  1162
  1163
  1164
  1165
  1166
  1167
  1168
  1169
  1169
  1170
  1171
  1172
  1173
  1174
  1175
  1176
  1177
  1178
  1179
  1179
  1180
  1181
  1182
  1183
  1184
  1185
  1186
  1187
  1188
  1188
  1189
  1190
  1191
  1192
  1193
  1194
  1195
  1196
  1197
  1197
  1198
  1199
  1199
  1200
  1201
  1202
  1203
  1204
  1205
  1206
  1207
  1208
  1209
  1209
  1210
  1211
  1212
  1213
  1214
  1215
  1216
  1217
  1218
  1219
  1219
  1220
  1221
  1222
  1223
  1224
  1225
  1226
  1227
  1228
  1229
  1229
  1230
  1231
  1232
  1233
  1234
  1235
  1236
  1237
  1238
  1238
  1239
  1240
  1241
  1242
  1243
  1244
  1245
  1246
  1247
  1248
  1248
  1249
  1250
  1251
  1252
  1253
  1254
  1255
  1256
  1257
  1258
  1258
  1259
  1260
  1261
  1262
  1263
  1264
  1265
  1266
  1267
  1268
  1268
  1269
  1270
  1271
  1272
  1273
  1274
  1275
  1276
  1277
  1278
  1278
  1279
  1280
  1281
  1282
  1283
  1284
  1285
  1286
  1287
  1288
  1288
  1289
  1290
  1291
  1292
  1293
  1294
  1295
  1296
  1297
  1297
  1298
  1299
  1299
  1300
  1301
  1302
  1303
  1304
  1305
  1306
  1307
  1308
  1308
  1309
  1310
  1311
  1312
  1313
  1314
  1315
  1316
  1317
  1317
  1318
  1319
  1319
  1320
  1321
  1322
  1323
  1324
  1325
  1326
  1327
  1328
  1328
  1329
  1330
  1331
  1332
  1333
  1334
  1335
  1336
  1337
  1338
  1338
  1339
  1340
  1341
  1342
  1343
  1344
  1345
  1346
  1347
  1348
  1348
  1349
  1350
  1351
  1352
  1353
  1354
  1355
  1356
  1357
  1358
  1358
  1359
  1360
  1361
  1362
  1363
  1364
  1365
  1366
  1367
  1368
  1368
  1369
  1370
  1371
  1372
  1373
  1374
  1375
  1376
  1377
  1378
  1378
  1379
  1380
  1381
  1382
  1383
  1384
  1385
  1386
  1387
  1388
  1388
  1389
  1390
  1391
  1392
  1393
  1394
  1395
  1396
  1397
  1397
  1398
  1399
  1399
  1400
  1401
  1402
  1403
  1404
  1405
  1406
  1407
  1408
  1408
  1409
  1410
  1411
  1412
  1413
  1414
  1415
  1416
  1417
  1417
  1418
  1419
  1419
  1420
  1421
  1422
  1423
  1424
  1425
  1426
  1427
  1428
  1428
  1429
  1430
  1431
  1432
  1433
  1434
  1435
  1436
  1437
  1438
  1438
  1439
  1440
  1441
  1442
  1443
  1444
  1445
  1446
  1447
  1448
  1449
  1449
  1450
  1451
  1452
  1453
  1454
  1455
  1456
  1457
  1458
  1458
  1459
  1460
  1461
  1462
  1463
  1464
  1465
  1466
  1467
  1468
  1468
  1469
  1470
  1471
  1472
  1473
  1474
  1475
  1476
  1477
  1478
  1478
  1479
  1480
  1481
  1482
  1483
  1484
  1485
  1486
  1487
  1488
  1488
  1489
  1490
  1491
  1492
  1493
  1494
  1495
  1496
  1497
  1497
  1498
  1499
  1499
  1500
  1501
  1502
  1503
  1504
  1505
  1506
  1507
  1508
  1508
  1509
  1510
  1511
  1512
  1513
  1514
  1515
  1516
  1517
  1518
  1518
  1519
  1520
  1521
  1522
  1523
  1524
  1525
  1526
  1527
  1528
  1528
  1529
  1530
  1531
  1532
  1533
  1534
  1535
  1536
  1537
  1538
  1538
  1539
  1540
  1541
  1542
  1543
  1544
  1545
  1546
  1547
  1548
  1548
  1549
  1550
  1551
  1552
  1553
  1554
  1555
  1556
  1557
  1558
  1558
  1559
  1560
  1561
  1562
  1563
  1564
  1565
  1566
  1567
  1568
  1568
  1569
  1570
  1571
  1572
  1573
  1574
  1575
  1576
  1577
  1578
  1579
  1579
  1580
  1581
  1582
  1583
  1584
  1585
  1586
  1587
  1588
  1588
  1589
  1590
  1591
  1592
  1593
  1594
  1595
  1596
  159
```

Before:

```

1 --Query 14:
2 explain analyze
3 SELECT c.last_name,
4      c.first_name,
5      c.email
6 FROM customer c
7 ORDER BY c.last_name;
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23

```

After (2 Screenshots):

```

1 --Query 14:
2 explain analyze
3 SELECT c.last_name,
4      c.first_name,
5      c.email
6 FROM customer c
7 ORDER BY c.last_name;
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23

```

```

pgAdmin 4 — Mozilla Firefox
PgAdmin File Object Tools Help
Browser Dashboard Properties SQL Statistics Dependencies Dependents Project_2_DBP\vm@localhost*
Project_2_DBP\vm@localhost
Project_2_DBP
Tables Schemas
SELECT c.last_name,
       c.first_name,
       c.email
FROM customer c
ORDER BY c.last_name;
QUERY PLAN
Nested Loop (cost=0.00..0.269 rows=6 width=4) (actual time=0.269..0.271 rows=7 loops=1)
  > Sort (cost=0.00..0.001 rows=6 width=4) (actual time=0.269..0.271 rows=7 loops=1)
    Sort Key: c.address_id
    Sort Method: quicksort Memory: 256B
    > Merge Join (cost=0.00..0.257..0.268 rows=7 loops=1)
      > Merge Cond: (c.city_id = a.city_id)
      > Sort (cost=0.00..0.001 rows=6 width=4) (actual time=0.083..0.084 rows=7 loops=1)
        Sort Key: c.city_id
        Sort Method: quicksort Memory: 256B
        > Nested Loop (cost=0.00..0.031..0.080 rows=7 loops=1)
          > Join Filter: (c.country_id = cc.country_id)
          Rows Removed by Join Filter: 593
          > Seq Scan on country cc (cost=0.00..0.256 rows=1 width=4) (actual time=0.009..0.013 rows=1 loops=1)
            Filter: (country = 'Canada'::text)
            Rows Removed by Filter: 108
          > Rows Removed by Filter: 108
          > Sort (cost=0.00..0.11..0.100 rows=600 width=8) (actual time=0.003..0.036 rows=600 loops=1)
            > Sort Scan on city c1 (cost=0.00..0.11..0.100 rows=600 width=8) (actual time=0.003..0.036 rows=600 loops=1)
            > Sort (cost=0.00..0.001 rows=603 width=8) (actual time=0.140..0.157 rows=568 loops=1)
            > Sort Key: c1.city_id
            Sort Method: quicksort Memory: 512B
            > Seq Scan on address a (cost=0.00..14.63 rows=603 width=8) (actual time=0.004..0.056 rows=603 loops=1)
            > Sort (cost=0.00..0.001..0.001 rows=599 width=8) (actual time=0.185..0.206 rows=477 loops=1)
            > Sort Key: a.address_id
            Sort Method: quicksort Memory: 100KB
            > Seq Scan on customer c (cost=0.00..14.99 rows=599 width=49) (actual time=0.004..0.079 rows=599 loops=1)
            Planning Time: 0.271 ms
            Execution Time: 0.551 ms
            Execution Time: 0.551 ms
  
```

Commands:

The commands we used are:

- create index idx_address on address using BTREE (address_id);
- create index idx_city_id on city using HASH (city_id);
- create index idx_country on country using HASH (country);
- create index idx_lastname on customer using BTREE (last_name);

Comparison:

	Before	After
Highest Cost	Hash join = 2.39	Merge join = 45.7
Slowest Runtime	Hash join = 0.137 ms	Merge join = 0.252 ms
Largest Number of Rows	Seq scan on address a = 603	Seq scan on address a = 603
Planning Time	4.855 ms	0.271 ms
Execution Time	96.369 ms	0.551 ms

-The comparison shows that the cost is higher, and the time is less and what is observed that the hash join exchanged with merge join for better performance.

Query 15:

For query 15 we used an index on the (film_id, category_id) attributes using BTREE. For better faster optimization.

Before:

The screenshot shows the pgAdmin 4 interface with the following details:

- Header:** pgAdmin 4 - DB Programming Project 2 - DB - DBP - Project Deliveries
- Toolbar:** Includes File, Object, Tools, Help, and various icons for database management.
- Servers:** Localhost is selected, showing databases (6), tables (6), and schemas (1).
- Databases:** Project_2_DB is selected, showing Catalogs, Event Triggers, Extensions, Foreign Data Wrappers, Languages, and Schemas (1).
- Schema:** Task_6 is selected.
- Query Editor:** The query is:

```
1 --Query 15;
2 explain analyze
3 SELECT *
4 FROM film
5 JOIN film_category fc
6 ON f.film_id = fc.film_id
7 JOIN category c
8 ON fc.category_id = c.category_id
9 WHERE c.category_id = 8;
```
- Data Output:** Shows the query plan (QUERY PLAN) which includes:
 - Nested Loop (cost=19.35, 88 rows=69 width=15) (actual time=0.950..22.930 rows=69 loops=1)
 - > Seq Scan on category c (cost=0.00..1.20 rows=1 width=4) (actual time=0.430..0.431 rows=1 loops=1)
 - Filter: (category_id = 8)
 - Rows Removed by Filter: 15
 - > Hash Join (cost=19.36..87.00 rows=69 width=16) (actual time=0.532..22.460 rows=69 loops=1)
 - Hash Cond: (f.film_id = fc.film_id)
 - > Seq Scan on film f (cost=0.00..65.00 rows=1000 width=19) (actual time=0.005..21.766 rows=1000 loops=1)
 - Buckets: 1024 Batches: 1 Memory Usage: 11kB
 - > Seq Scan on film_category fc (cost=0.00..18.50 rows=69 width=8) (actual time=0.004..0.500 rows=69 loops=1)
 - Filter: (category_id = 8)
 - Rows Removed by Filter: 931
 - Planning Time: 23.071 ms
 - Execution Time: 23.917 ms

After:

The screenshot shows the pgAdmin 4 interface with the following details:

- Header:** pgAdmin 4 — Mozilla Firefox, DB Programming Project 2 - D, DBP - Project Deliverables
- Toolbar:** File, Object, Tools, Help
- Servers:** localhost (1)
- Databases:** Project_1_DBP, Project_2_DBP
- Current Database:** Project_2_DBP
- Tables:** f, category
- Query Editor:** SELECT f.film_id, fc.category_id FROM f JOIN category c ON f.category_id = c.category_id WHERE c.category_id = 8;
- Output:** Data Output, Explain, Messages, Notifications
- Explain Plan:** Nested Loop (cost=35.33..142.36 rows=69 width=15) (actual time=0.242..0.333 rows=69 loops=1)
 - Index Only Scan using idx_catid on category c (cost=14.8..11 rows=1 width=4) (actual time=0.036..0.087 rows=1 loops=1)
 - Index Cond: (category_id = 8)
 - Hash Join (cost=35.79..131.52 rows=69 width=15) (actual time=0.153..0.731 rows=69 loops=1)
 - Hash Cond: (f.film_id = fc.film_id)
 - Index Scan using idx_film_id on film f (cost=0.28..53.37 rows=1000 width=15) (actual time=0.010..0.394 rows=1000 loops=1)
 - Hash Cond: (f.film_id = fc.film_id)
 - Index Scan using idx_category_id on category fc (cost=0.28..53.37 rows=1000 width=15) (actual time=0.131..0.709 rows=1000 loops=1)
 - Buckets: 1024 Batches: 1 Memory Usage: 11kB
 - Bitmap Heap Scan on film_category fc (cost=27.79..34.65 rows=69 width=8) (actual time=0.072..0.108 rows=69 loops=1)
 - Recheck Cond: (category_id = 8)
 - Heap Blocks: exact4
 - Index Scan on film_category fc (cost=0.00..27.77 rows=69 width=0) (actual time=0.059..0.059 rows=69 loops=1)
 - Index Cond: (category_id = 8)
- Timing:** Planning Time: 0.359 ms, Execution Time: 0.800 ms

Commands:

The commands we used are:

- create index idx_filmid on film using BTREE (film_id);
 - create index idx_catid on category using BTREE (category_id);
 - set enable_seqscan = off;

Comparison:

	Before	After
Highest Cost	Hash join = 3.5	Hash join = 3.5
Slowest Runtime	Hash join = 0.203 ms	Index scan using idx_filmid on film f = 0.205 ms
Largest Number of Rows	Seq scan on film f = 1000	Index scan using idx_filmid on film f = 1000
Planning Time	228.071 ms	0.359 ms
Execution Time	23.017 ms	0.880 ms

-The comparison shows that the cost is higher, and the time is less because of the added new techniques that it gives the cost more higher values, the seqscan exchanged with the index only scan.

Query 16:

For query 16 we used an index on the (store_id, staff_id) attributes using BTREE. For better performance and fast searching.

Before:

The screenshot shows the pgAdmin 4 interface with the following details:

- Project_2_DBP** is selected in the left sidebar under "Databases".
- The "Tablespace" node is expanded, showing "Task_6" and "task_id".
- The "Query Editor" tab is active, displaying the following SQL query:

```
1 --Query 16:
2 explain analyze
3 SELECT sum(p.amount) AS total_sales
4 FROM store str
5 JOIN staff stf
6 JOIN staff_stf
7 GROUP BY str.store_id
8 ORDER BY total_sales DESC
9 LIMIT 10;
```

Query Plan:

```
1 HashAggregate (cost=441.71..443.71 rows=160 width=36) (actual time=111.144..111.154 rows=2 loops=1)
  2  Group Key: str.store_id
  3  Batches: 1 Memory Usage: 40kB
  4    > Hash Join (cost=17.39..440.91 rows=160 width=10) (actual time=93.116..105.143 rows=16049 loops=1)
  5      > Hash Cond: (p.staff_id = st.staff_id)
  6      > Append (cost=361.74 rows=1600 width=10) (actual time=0.12..545.9 rows=16049 loops=1)
  7        > Seq Scan on payment_2022_01_p_1 (cost=0..0, 12.32 rows=723 width=10) (actual time=0..01..0.175 rows=723 loops=1)
  8        > Seq Scan on payment_2022_02_p_2 (cost=0..0, 42.07 rows=2401 width=10) (actual time=0..01..0.790 rows=2401 loops=1)
  9        > Seq Scan on payment_2022_03_p_3 (cost=0..0, 47.13 rows=2713 width=10) (actual time=0..06..0.644 rows=2713 loops=1)
  10       > Seq Scan on payment_2022_04_p_4 (cost=0..0, 44.47 rows=2547 width=10) (actual time=0..01..0.517 rows=2547 loops=1)
  11       > Seq Scan on payment_2022_05_p_5 (cost=0..0, 46.75 rows=2677 width=10) (actual time=0..06..0.643 rows=2677 loops=1)
  12       > Seq Scan on payment_2022_06_p_6 (cost=0..0, 46.54 rows=2654 width=10) (actual time=0..01..0.899 rows=2654 loops=1)
  13       > Seq Scan on payment_2022_07_p_7 (cost=0..0, 43.81 rows=234 width=10) (actual time=0..01..0.534 rows=234 loops=1)
  14     > Hash (cost=17.39..17.37 rows=2 width=8) (actual time=93.061..93.083 rows=2 loops=1)
  15   Buckets: 1024 Batches: 1 Memory Usage: 9kB
  16   > Nested Loop (cost=0..15..15.37 rows=2 width=8) (actual time=93.067..93.078 rows=2 loops=1)
  17   > Seq Scan on staff (cost=0..0, 102 rows=2 width=8) (actual time=0..02..0.024 rows=2 loops=1)
  18   > Index Only Scan using store_pkey on store (cost=0..15..17 rows=1 width=4) (actual time=46.532..46.532 rows=1 loops=2)
  19   Index Cond: (store_id = stf.store_id)
  20   Heap Fetches: 2
  21 Planning Time: 1.256 ms
  22 Execution Time: 111.203 ms
```

After:

Commands:

The commands we used are:

- create index idx_store on store using BTREE (store_id);
 - create index idx_staffid on staff using BTREE (staff_id)

Comparison:

	Before	After
Highest Cost	Append = 80.25	Append = 80.25
Slowest Runtime	Nested loop = 46.542 ms	Nested loop = 8.557 ms
Largest Number of Rows	Seq scan on payment_p2022_03 p_3 = 2713	Seq scan on payment_p2022_03 p_3 = 2713
Planning Time	1.256 ms	0.524 ms
Execution Time	111.205 ms	25.829 ms

-The comparison shows that the cost and the time are less than before results, and a nested loop was added and instead of using the index only scan, the seqscan was used.

Query 17:

For query 17 we used an index on the (category_id, film_id, inventory_id, rental_id) attributes using BTREE on all four of them. For better performance and fast searching.

Before (2 Screenshots):

The screenshot shows two instances of pgAdmin 4 running in separate browser tabs. Both tabs are connected to the same database project, 'Project_2_DBP@vm[localhost]'. The top tab displays the Explain Analyze output for a complex query involving multiple joins and aggregate functions. The bottom tab shows a similar Explain Analyze output for another query. The interface includes a navigation bar with 'File', 'Object', 'Tools', 'Help', and various database management options. The main area shows the schema tree, query editor, and execution results.

After (2 screenshots):

```

pgAdmin 4 — Mozilla Firefox
DB Programming Project 2 - D | DBP - Project Deliverables | +
127.0.0.1:41477/browser/
PgAdmin File Object Tools Help
Servers (1)
localhost
  Databases (6)
    Project_1_DBP
    Project_2_DBP
    Task_A
    Catalogs
    Extensions
    Foreign Data Wrappers
    Languages
    Schemas (1)
      Task_A
    demo
    postgres
    task_lab_4
  Logins/Group Roles
  Tablespaces
Query Editor Query History
1 --query 17
2 explain analyze
3 SELECT c.name,
Data Output Explain Messages Notifications
QUERY PLAN
  Limit (cost=6466.14..4646.15 rows=5 width=64) (actual time=57.174..57.189 rows=5 loops=1)
    > Sort (cost=4646.14..4646.18 rows=16 width=64) (actual time=57.172..57.186 rows=5 loops=1)
      Sort Key (sum(p.amount)) DESC
      Sort Method: top N heapsort Memory: 29.8
      HashAggregate (cost=6465.08..6465.08 rows=16 width=64) (actual time=57.156..57.171 rows=16 loops=1)
        Group Key c.name
        Batches: 1 Memory Usage: 24.8
        > Merge Join (cost=301.81..4565.43 rows=16049 width=38) (actual time=41.231..56.050 rows=16049 loops=1)
          Merge Cond: (l.inventory_id = r.inventory_id)
          Sort (cost=838.66..850.31 rows=4581 width=36) (actual time=6.231..5.574 rows=4581 loops=1)
            Sort Key l.inventory_id
            Sort Method: quicksort Memory: 40.7K
            HashAggregate (cost=4645.08..4645.08 rows=16 width=64) (actual time=2.845..5.075 rows=4581 loops=1)
              Group Key c.name
              Batches: 1 Memory Usage: 24.8
              > Merge Join (cost=458.59..59.160.30 rows=4581 width=36) (actual time=2.845..5.075 rows=4581 loops=1)
                Merge Cond: (f.film_id = l.film_id)
                Sort (cost=132.22..134.72 rows=1000 width=36) (actual time=1.124..1.254 rows=1000 loops=1)
                  Sort Key f.film_id
                  Sort Method: quicksort Memory: 7.1K
                  HashAggregate (cost=4645.08..4645.08 rows=1000 width=36) (actual time=0.445..0.784 rows=1000 loops=1)
                    Group Key f.category_id
                    Batches: 1 Memory Usage: 23.8K
                    > Merge Join (cost=67.31..82.39 rows=1000 width=36) (actual time=0.017..0.019 rows=1000 loops=1)
                      Merge Cond: (l.category_id = f.category_id)
                      Sort Key f.category_id
                      Sort Method: quicksort Memory: 23.8K
                      HashAggregate (cost=4645.08..4645.08 rows=1000 width=36) (actual time=0.037..0.011 rows=1000 loops=1)
                        Group Key f.category_id
                        Batches: 1 Memory Usage: 23.8K
                        > Seq Scan on category (cost=0.03..1.16 rows=16 width=36) (actual time=0.037..0.011 rows=16 loops=1)
                        > Sort (cost=19.83..19.83 rows=1000 width=36) (actual time=0.427..0.494 rows=1000 loops=1)
                          Sort Key f.category_id
                          Batches: 1 Memory Usage: 23.8K
                          HashAggregate (cost=4645.08..4645.08 rows=1000 width=36) (actual time=0.017..0.019 rows=1000 loops=1)
                            Group Key f.category_id
                            Batches: 1 Memory Usage: 23.8K
                            > Sort (cost=442.94..452.06 rows=16049 width=10) (actual time=24.968..37.151 rows=16049 loops=1)
                              Sort Key l.inventory_id
                              Sort Method: quicksort Memory: 113.7K
                              HashAggregate (cost=4645.08..4645.08 rows=16049 width=10) (actual time=0.008..0.871 rows=4581 loops=1)
                                Group Key l.inventory_id
                                Batches: 1 Memory Usage: 113.7K
                                > Append (cost=0.03..317.74 rows=1004 width=10) (actual time=0.012..4.304 rows=16049 loops=1)
                                  > Seq Scan on payment_p0202_1.p_1 (cost=0.03..12.23 rows=723 width=10) (actual time=0.011..0.207 rows=723 loops=1)
                                  > Seq Scan on payment_p0202_1.p_2 (cost=0.03..42.21 rows=2401 width=10) (actual time=0.036..0.475 rows=2401 loops=1)
                                  > Seq Scan on payment_p0202_1.p_3 (cost=0.03..47.13 rows=2713 width=10) (actual time=0.012..0.517 rows=2713 loops=1)
                                  > Seq Scan on payment_p0202_1.p_4 (cost=0.03..44.47 rows=2547 width=10) (actual time=0.033..0.483 rows=2547 loops=1)
                                  > Seq Scan on payment_p0202_1.p_5 (cost=0.03..46.77 rows=2577 width=10) (actual time=0.036..0.509 rows=2577 loops=1)
                                  > Seq Scan on payment_p0202_1.p_6 (cost=0.03..45.54 rows=2544 width=10) (actual time=0.037..0.503 rows=2544 loops=1)
                                  > Seq Scan on payment_p0202_1.p_7 (cost=0.03..41.34 rows=2334 width=10) (actual time=0.037..0.562 rows=2334 loops=1)
                                  Planning Time: 0.655 ms
                                  Execution Time: 57.436 ms

```

Commands:

The commands we used are:

- create index idx_cat on category using BTREE (category_id);
- create index idx_film on film using BTREE (film_id);
- create index idx_inv on inventory using BTREE (inventory_id);
- create index idx_rental on rental using BTREE (rental_id);
- set enable_hashjoin = off;

Comparison:

	Before	After
Highest Cost	Hash join = 579.12	Hash agg = 3715.12
Slowest Runtime	Hash join = 34.993 ms	Merge join = 43.476 ms
Largest Number of Rows	Hash join = 16049	Merge join = 16049
Planning Time	138.524 ms	0.655 ms
Execution Time	90.458 ms	57.436 ms

-The comparison shows that the cost is higher, and the time is less, and the hash join exchanged into merge join for better optimization.

Query 18:

For query 18 we used an index on the (category_id, film_id, inventory_id, rental_id) attributes using BTREE on all four of them. For better performance and fast searching.

Before (2 Screenshots):

The screenshot shows the pgAdmin 4 interface with multiple tabs open. The main window displays a detailed query plan for a SELECT statement. The plan includes sections for the Query Editor, Data Output, Explain, Messages, and Notifications. The Explain section is expanded, showing the following steps:

- 1 -->query 1.1
- 2 explain analyze
- 3 SELECT c.name

The EXPLAIN PLAN output is as follows:

```
QUERY PLAN
+- Net
   |  +-- Hash Join (cost=91.99..914.00 rows=16049 width=10) (actual time=6.67..20.041 rows=16049 loops=1)
   |    Hash Cond: (parent_id = result_id)
   |    +-- Append (cost=0.36..74.74 rows=1609 width=10) (actual time=0.01..5.286 rows=16049 loops=1)
   |         +> Seq Scan on payment_p0022_01_p1 (cost=0.00..13.23 rows=723 width=10) (actual time=0.01..1.101 rows=723 loops=1)
   |         +> Seq Scan on payment_p0022_01_p2 (cost=0.00..42.01 rows=2401 width=10) (actual time=0.01..6.346 rows=2401 loops=1)
   |         +> Seq Scan on payment_p0022_03_p3 (cost=0.00..47.13 rows=2713 width=10) (actual time=0.01..6.834 rows=2713 loops=1)
   |         +> Seq Scan on payment_p0022_04_p4 (cost=0.00..44.47 rows=2347 width=10) (actual time=0.02..6.622 rows=2347 loops=1)
   |         +> Seq Scan on payment_p0022_05_p5 (cost=0.00..46.77 rows=267 width=10) (actual time=0.01..0.576 rows=267 loops=1)
   |         +> Seq Scan on payment_p0022_06_p6 (cost=0.00..46.54 rows=2054 width=10) (actual time=0.01..0.580 rows=2054 loops=1)
   |         +> Seq Scan on payment_p0022_07_p7 (cost=0.00..41.34 rows=234 width=10) (actual time=0.01..0.547 rows=234 loops=1)
   |    +> Hash (cost=110.4..310.44 rows=16044 width=9) (actual time=620.652..710.154 loops=1)
      Buckets: 16384 Batches: 1 Memory Usage: 32KB
   |    +-- Seq Scan on rental (cost=0.00..310.44 rows=16044 width=9) (actual time=0.005..3.026 rows=16044 loops=1)
   |        +> Hash (cost=78.71..75.81 rows=4581 width=9) (actual time=1.827..1.828 rows=4581 loops=1)
      Buckets: 8116 Batches: 1 Memory Usage: 243KB
   |        +-- Seq Scan on inventory (cost=0.00..78.71 rows=4581 width=9) (actual time=0.008..0.056 rows=4581 loops=1)
   |            +> Hash (cost=20.47..20.57 rows=1000 width=9) (actual time=0.819..0.821 rows=1000 loops=1)
      Buckets: 1024 Batches: 1 Memory Usage: 9KB
   |            +-- Hash Join (cost=13..34.20.67 rows=1000 width=9) (actual time=0.036..0.592 rows=1000 loops=1)
   |                Hash Cond: (fc.category_id = category_id)
   |                +> Seq Scan on film_category fc (cost=0.00..16.90 rows=1000 width=8) (actual time=0.019..0.145 rows=1000 loops=1)
   |                +> Hash (cost=1..16..1.16 rows=16 width=36) (actual time=0.014..0.014 rows=16 loops=1)
      Buckets: 1024 Batches: 1 Memory Usage: 9KB
   |                    +-- Seq Scan on category c (cost=0.00..1.16 rows=16 width=36) (actual time=0.006..0.009 rows=16 loops=1)
35 Planning Time: 0.588 ms
36 Execution Time: 0.123 ms
```

The screenshot shows the pgAdmin 4 interface with the following details:

- Server:** Project_2_DB@vm|localhost
- Query Editor:** Query History
- SQL:**

```
--Query 1:
1 explain analyze
2
3 SELECT c.name,
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
```
- Data Output:** Explain, Messages, Notifications
- Query Plan:**

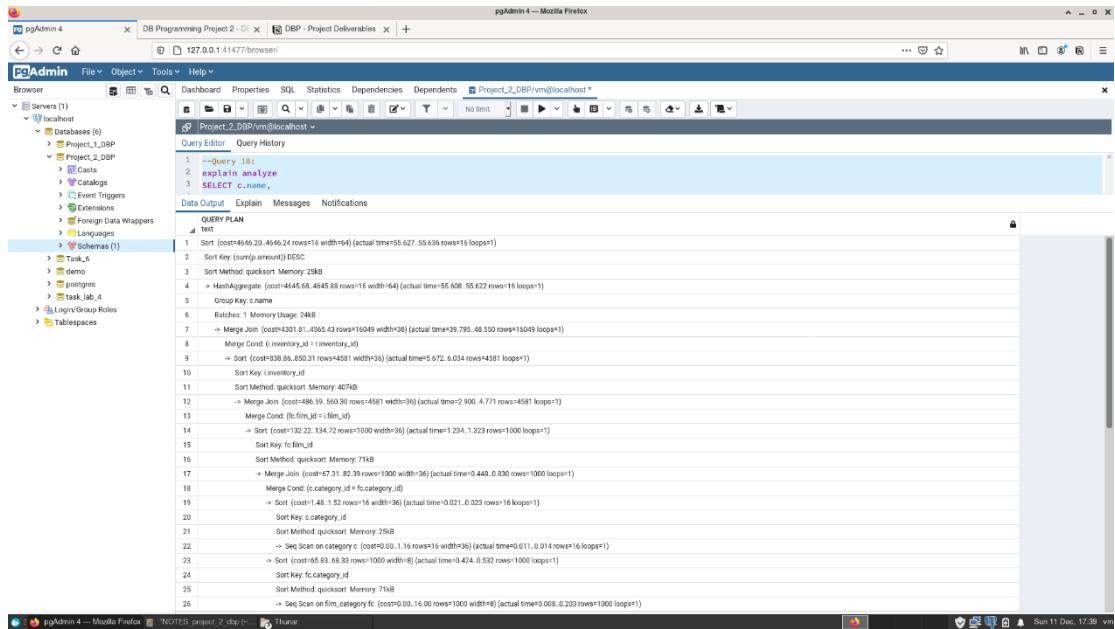
```
QUERY PLAN
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
```
- Execution Plan:**
 - Sort (cost=1424.72..1424.76 rows=1 width=64) (actual time=44.312..44.322 rows=1 loops=1)
 - Sort Key: sum(p.amount) E55C
 - Sort Method: quicksort Memory: 256B
 - > NestedLoopJoin (cost=1424.50..1424.49 rows=1 width=64) (actual time=44.289..44.306 rows=1 loops=1)
 - Group Key: c.name
 - Batches: 1 Memory Usage: 248B
 - > Hash Join (cost=677.24..1342.99 rows=16049 width=93) (actual time=97.32..37.163 rows=16049 loops=1)
 - Hash Cond: (t1.name = t2.name)
 - > Hash Join (cost=644.00..1190.11 rows=16049 width=10) (actual time=8.841..29.908 rows=16049 loops=1)
 - Hash Cond: (t1.id = t2.id)
 - > Hash Join (cost=510.99..91.9188 rows=16049 width=10) (actual time=6.967..20.721 rows=16049 loops=1)
 - Hash Cond: (p.rental_id = rental.id)
 - > Hash Join (cost=0.361..72 rows=16049 width=10) (actual time=0.017..5.567 rows=16049 loops=1)
 - > Seq Scan on payment p0222_0, p_1 (cost=0.00..13.23 rows=723 width=10) (actual time=0.016..0.227 rows=723 loops=1)
 - > Seq Scan on payment p0222_0, p_2 (cost=0.00..41.01 rows=5401 width=10) (actual time=0.017..0.550 rows=2401 loops=1)
 - > Seq Scan on payment p0222_0, p_3 (cost=0.00..47.13 rows=719 width=10) (actual time=0.021..0.992 rows=719 loops=1)
 - > Seq Scan on payment p0222_0, p_4 (cost=0.00..44.77 rows=547 width=10) (actual time=0.023..0.934 rows=547 loops=1)
 - > Seq Scan on payment p0222_0, p_5 (cost=0.00..46.77 rows=527 width=10) (actual time=0.014..0.596 rows=267 loops=1)
 - > Seq Scan on payment p0222_0, p_6 (cost=0.00..46.54 rows=254 width=10) (actual time=0.013..0.573 rows=254 loops=1)
 - > Seq Scan on payment p0222_0, p_7 (cost=0.00..41.39 rows=2334 width=10) (actual time=0.021..0.580 rows=2334 loops=1)
 - > Hash (cost=310.44..310.44 rows=16044 width=10) (actual time=6.869..6.870 rows=16044 loops=1)
 - Buckets: 16384 Batches: 1 Memory Usage: 7598B
 - > Seq Scan on rental r (cost=0.310..44.44 rows=16044 width=8) (actual time=0.005..3.919 rows=16044 loops=1)
 - > Hash (cost=75.81..75.81 rows=4581 width=9) (actual time=92.1..92.1 rows=4581 loops=1)
 - Buckets: 1632 Batches: 1 Memory Usage: 2438B
 - > Seq Scan on payment p (cost=0..76 rows=4581 width=10) (actual time=0.001..0.001 rows=4581 loops=1)

After (2 Screenshots):

The screenshot shows the pgAdmin 4 interface with the following details:

- Left Panel:** Shows the database structure with databases (Project_1_DBP, Project_2_DBP), tables (Cast, Roles, Logins, Task, Logins), and other objects like Schemas, Tablespace, and Logins.
- Top Bar:** pgAdmin 4 - DB Programming Project 2 - DBP - Project Deliverables
- Toolbar:** Includes icons for back, forward, search, and various database operations.
- Query Editor:** Contains the following SQL code:

```
--Query 18:
1  explain analyze
2  SELECT c.name,
3    FROM pg_catalog.pg_constraint c
4    WHERE c.conname = 'fk_task_film'
5    ORDER BY c.conname;
```
- Result Set:** Displays the query plan (QUERY PLAN) for the EXPLAIN ANALYZE command. The plan shows the execution flow from sorting categories to joining rental and payment tables, and finally scanning the film table.
- Bottom Status:** Planning Time: 0.569 ms, Execution Time: 55.990 ms.



Commands:

The commands we used are:

- create index idx_cat on category using BTREE (category_id);
- create index idx_film on film_category using BTREE (film_id);
- create index idx_inv on inventory using BTREE (inventory_id);
- create index idx_rental on rental using BTREE (rental_id)
- set enable_hashjoin = off;

Comparison:

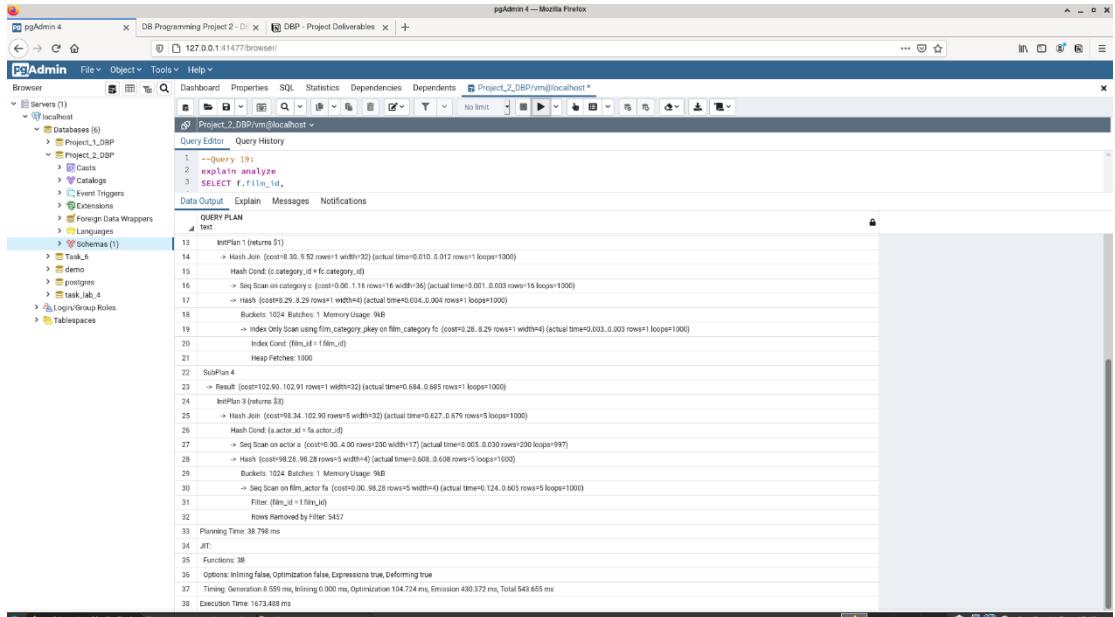
	Before	After
Highest Cost	Hash join = 553.12	Merge join = 3715.12
Slowest Runtime	Hash agg = 7.143 ms	Merge join = 42.516 ms
Largest Number of Rows	Hash join = 16049	Merge join = 16049
Planning Time	0.588 ms	0.569 ms
Execution Time	43.123 ms	55.890 ms

-The comparison shows that the cost and the time got higher, and the hash join became merge join.

Query 19:

For query 19 we used an index on the (category_id, film_id, actor_id, language_id) attributes using BTREE on all four of them. For better performance.

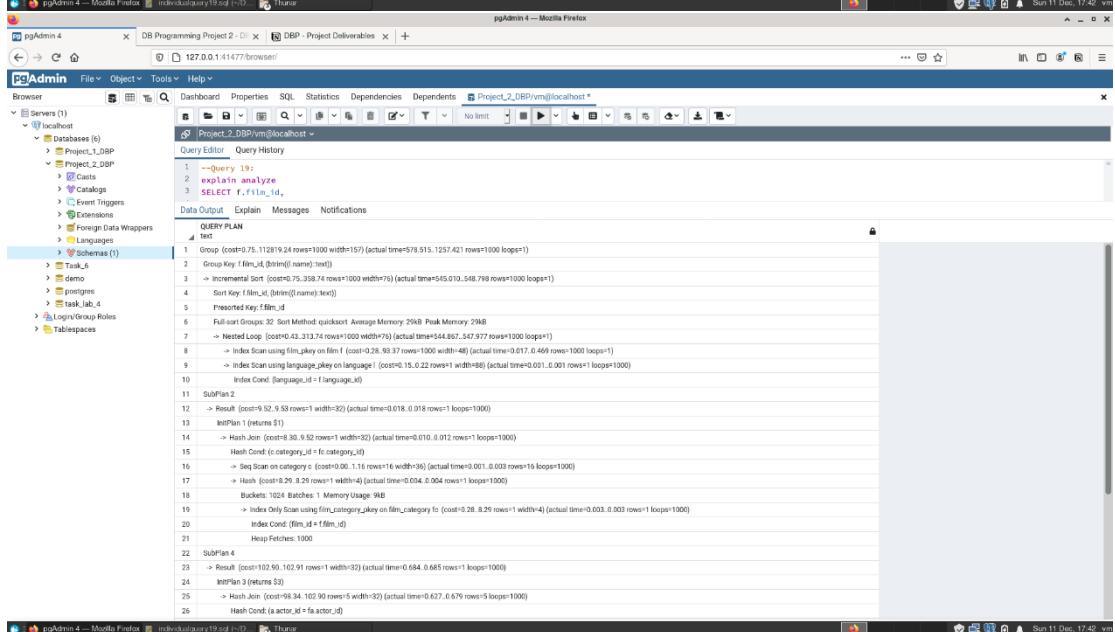
Before (2 Screenshots):



```

pgAdmin 4 — Mozilla Firefox
DB Programming Project 2 - D | DBP - Project Deliverables | + 127.0.0.1:41477/browser/
Project_2_DBP@localhost>
Query Editor | Query History
1 --query 19;
2 explain analyze
3 SELECT f.film_id,
Data Output | Explain | Messages | Notifications
QUERY PLAN
text
13 InitPlan 1 (returns $1)
14   > Hash Join (cost=0.30..9.52 rows=1 width=22) (actual time=0.010..2.012 rows=1 loops=1000)
15     Hash Cond: (c.category_id = fc.category_id)
16       > Seq Scan on category c (cost=0.00..1.16 rows=16 width=36) (actual time=0.001..0.003 rows=15 loops=1000)
17         > Hash (cost=29.29 rows=1 width=4) (actual time=0.034..0.004 rows=1 loops=1000)
18       Buckets: 1024 Batches: 1 Memory Usage: 9KB
19         > Index Only Scan using fm_category_pk on film_category fc (cost=0.28..8.29 rows=1 width=4) (actual time=0.003..0.003 rows=1 loops=1000)
20           Index Cond: (fm_id = fm_id)
21             Heap Fetches: 1000
22 SubPlan 4
23   > Result (cost=102.90..162.91 rows=1 width=32) (actual time=0.684..0.685 rows=1 loops=1000)
24   InitPlan 2 (returns $3)
25     > Hash Join (cost=0.34..102.90 rows=5 width=22) (actual time=0.637..0.679 rows=5 loops=1000)
26       Hash Cond: (actor_id = fa.actor_id)
27         > Seq Scan on actor a (cost=0.00..4.00 rows=200 width=17) (actual time=0.005..0.030 rows=200 loops=1000)
28         > Hash (cost=19.28..98.28 rows=5 width=4) (actual time=0.608..0.608 rows=5 loops=1000)
29       Buckets: 1024 Batches: 1 Memory Usage: 9KB
30         > Seq Scan on film_actor fa (cost=0.00..98.28 rows=5 width=8) (actual time=0.124..0.605 rows=5 loops=1000)
31           Filter: (film_id = f.film_id)
32             Rows Removed by Filter: 5457
33 Planning Time: 38.796 ms
34 JIT:
35 Functions: 38
36 Options: Inlining false, Optimization false, Expressions true, Deforming true
37 Timing: Generation: 8.559 ms, Inlining: 0.000 ms, Optimization: 104.724 ms, Emission: 490.572 ms, Total: 543.655 ms
Execution Time: 1673.408 ms

```

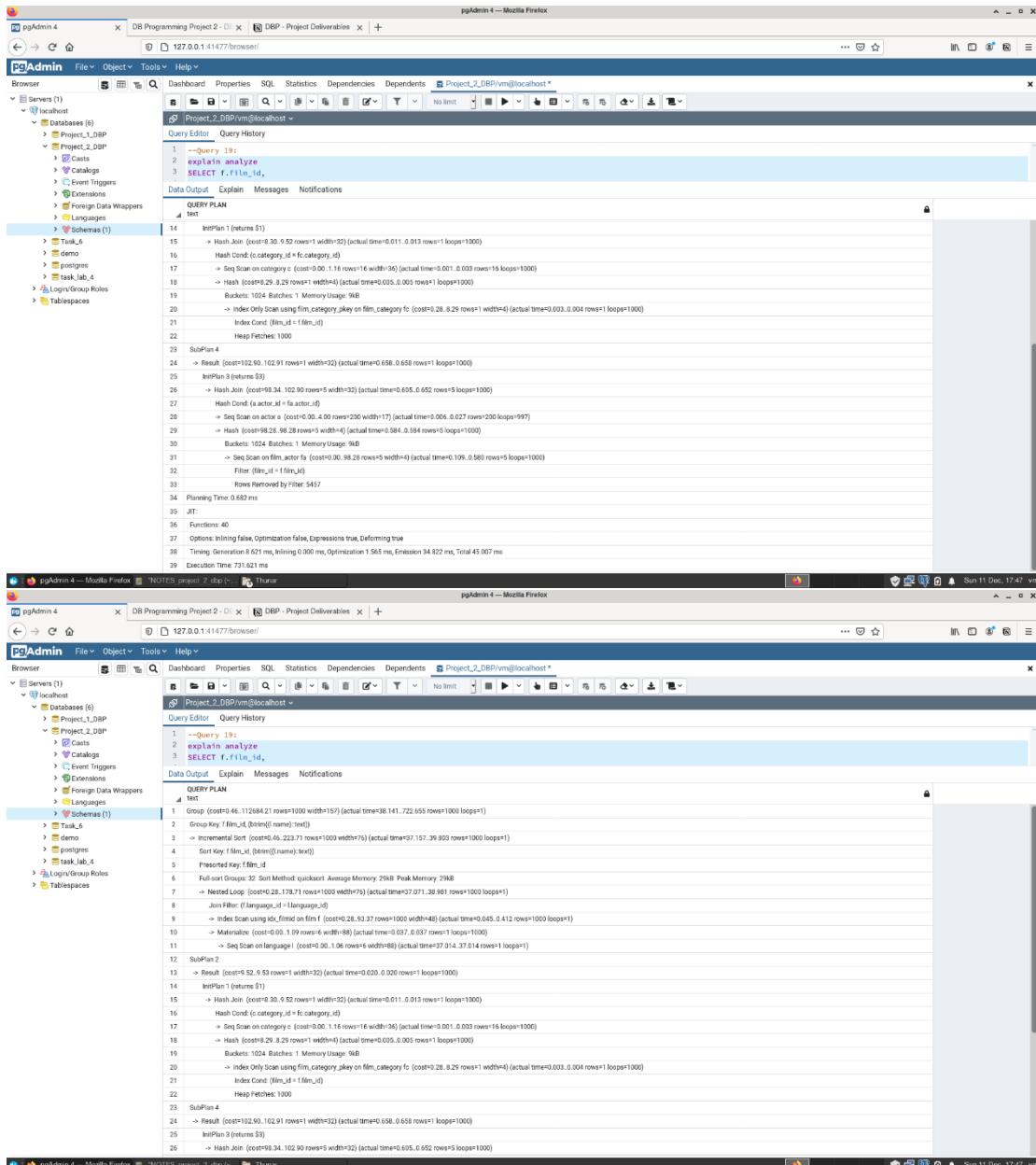


```

pgAdmin 4 — Mozilla Firefox
individuallyquery19.sql ~D | Thunar
127.0.0.1:41477/browser/
Project_2_DBP@localhost>
Query Editor | Query History
1 --query 19;
2 explain analyze
3 SELECT f.film_id,
Data Output | Explain | Messages | Notifications
QUERY PLAN
text
1 Group (cost=0.75..112319.24 rows=1000 width=159) (actual time=578.515..1257.421 rows=1000 loops=1)
2 Group Key (film_id, (bitn((name)::text)))
3 > Incremental Sort (cost=0.75..358.74 rows=1000 width=76) (actual time=545.010..548.798 rows=1000 loops=1)
4   Sort Key (film_id, (bitn((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=544.867..547.977 rows=1000 loops=1)
8     > Index Scan using film_pk on film f (cost=0.28..93.37 rows=1000 width=40) (actual time=0.017..0.469 rows=1000 loops=1)
9     > Index Scan using language_pk on language l (cost=0.15..0.22 rows=1 width=48) (actual time=0.001..0.001 rows=1 loops=1000)
10    Index Cond: (language_id = l.language_id)
11 SubPlan 2
12   > Result (cost=9.52..9.53 rows=1 width=32) (actual time=0.018..0.018 rows=1 loops=1000)
13   InitPlan 1 (returns $1)
14     > Hash Join (cost=0.30..9.52 rows=1 width=22) (actual time=0.010..0.012 rows=1 loops=1000)
15       Hash Cond: (c.category_id = fc.category_id)
16         > Seq Scan on category c (cost=0.00..1.16 rows=16 width=36) (actual time=0.001..0.003 rows=15 loops=1000)
17         > Hash (cost=29.29 rows=1 width=4) (actual time=0.034..0.004 rows=1 loops=1000)
18       Buckets: 1024 Batches: 1 Memory Usage: 9KB
19         > Index Only Scan using fm_category_pk on film_category fc (cost=0.28..8.29 rows=1 width=4) (actual time=0.003..0.003 rows=1 loops=1000)
20           Index Cond: (fm_id = fm_id)
21             Heap Fetches: 1000
22 SubPlan 4
23   > Result (cost=102.90..162.91 rows=1 width=32) (actual time=0.684..0.685 rows=1 loops=1000)
24   InitPlan 3 (returns $3)
25   > Hash Join (cost=0.34..102.90 rows=5 width=22) (actual time=0.637..0.679 rows=5 loops=1000)
26     Hash Cond: (actor_id = fa.actor_id)

```

After (2 Screenshots):



Commands:

The commands we used are:

- create index idx_catid on category using BTREE (category_id);
 - create index idx_filmid on film using BTREE (film_id);
 - create index idx_actorid on actor using BTREE (actor_id);
 - create index idx_lang on language using BTREE (language_id);

Comparison:

	Before	After
Highest Cost	Group = 112460.5	Group = 112460.5
Slowest Runtime	Group = 708.623 ms	Group = 682.852 ms
Largest Number of Rows	Group/incremental sort/nested loop/index scan using film_pkey on film f = 1000	Group/incremental sort/nested loop/index scan using film_pkey on film f = 1000
Planning Time	38.798 ms	0.682 ms
Execution Time	1673.488 ms	731.621 ms

-The comparison shows that the cost and time got less than before and adding materialize technique in join filter under the nested loop.