

Query1:

Before:

- Planning time:0.426 ms
- Execution time:0.568 ms
- Part of the plan that has:
 - Highest cost: Seq Scan on film
 - Slowest runtime: Seq Scan on film
 - Largest num of rows: Seq Scan on film

QUERY PLAN		text	
1	Sort (cost=74.24..74.69 rows=180 width=21) (actual time=0.467..0.476 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=0.016..0.410 rows=176 loops=1)		
5	Filter: (length > 160)		
6	Rows Removed by Filter: 824		
7	Planning Time: 0.426 ms		
8	Execution Time: 0.568 ms		

After:

QUERY PLAN		text	
1	Sort (cost=30.69..31.14 rows=180 width=21) (actual time=0.175..0.182 rows=176 loops=1)		
2	Sort Key: length DESC		
3	Sort Method: quicksort Memory: 38kB		
4	-> Index Scan using index1 on film (cost=0.14..23.94 rows=180 width=21) (actual time=0.025..0.132 rows=176 loops=1)		
5	Planning Time: 0.107 ms		
6	Execution Time: 0.205 ms		

Index Used: CREATE INDEX INDEX1 ON film USING BTREE (film_id) where film.length >160;

Instead of seq scan film table ,We created here partial index on film_id where film.length is only bigger than 160 ,to make the planner avoid all the rows that don't fulfill this condition

- Planning time:0.107 ms
- Execution time:0.205 ms
- Part of the plan that has:
 - Highest cost : Index Scan using index1
 - Slowest runtime: Index Scan using index1
 - Largest num of rows: Index Scan using index1

Query2:

Before:

•

QUERY PLAN	
	text
1	Sort (cost=26.15..26.19 rows=16 width=40) (actual time=0.400..0.404 rows=16 loops=1)
2	Sort Key: (count(category.name)) DESC
3	Sort Method: quicksort Memory: 25kB
4	-> HashAggregate (cost=25.67..25.83 rows=16 width=40) (actual time=0.380..0.386 rows=16 loops=1)
5	Group Key: category.name
6	Batches: 1 Memory Usage: 24kB
7	-> Hash Right Join (cost=1.36..20.67 rows=1000 width=32) (actual time=0.026..0.237 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=0.004..0.070 rows=1000 loops=1)
10	-> Hash (cost=1.16..1.16 rows=16 width=36) (actual time=0.017..0.018 rows=16 loops=1)
11	Buckets: 1024 Batches: 1 Memory Usage: 9kB
12	-> Seq Scan on category (cost=0.00..1.16 rows=16 width=36) (actual time=0.011..0.012 rows=16 loops=1)
13	Planning Time: 0.146 ms
14	Execution Time: 0.440 ms

- Planning time:0.146 ms
- Execution time:0.440 ms
- Part of the plan that has:
 - Highest cost : seq scan on film category
 - Slowest runtime: seq scan on film category
 - Largest num of rows: seq scan on film category (1000 rows)

After:

QUERY PLAN	
text	
1	Sort (cost=66.64..66.68 rows=16 width=40) (actual time=0.668..0.670 rows=16 loops=1)
2	Sort Key: (count(category.name)) DESC
3	Sort Method: quicksort Memory: 25kB
4	-> HashAggregate (cost=66.16..66.32 rows=16 width=40) (actual time=0.619..0.622 rows=16 loops=1)
5	Group Key: category.name
6	Batches: 1 Memory Usage: 24kB
7	-> Merge Right Join (cost=1.63..61.16 rows=1000 width=32) (actual time=0.183..0.482 rows=1000 loops=1)
8	Merge Cond: (film_category.category_id = category.category_id)
9	-> Index Scan using index2 on film_category (cost=0.15..47.10 rows=1000 width=8) (actual time=0.134..0.330 rows=1000 loops=1)
10	-> Sort (cost=1.48..1.52 rows=16 width=36) (actual time=0.043..0.044 rows=16 loops=1)
11	Sort Key: category.category_id
12	Sort Method: quicksort Memory: 25kB
13	-> Seq Scan on category (cost=0.00..1.16 rows=16 width=36) (actual time=0.020..0.021 rows=16 loops=1)
14	Planning Time: 0.398 ms
15	Execution Time: 0.729 ms

Index Used: CREATE INDEX INDEX2 ON film_category USING BTREE(category_id)

We disabled hashjoin technique to force the planner work with our index, Instead of seq scan on film_category which makes the planner scan the table sequentially now the planner uses index2 Which is BTREE index on attribute category_id that let us get any row in film_category table using attribute category_id in O(LOG(N))

- Planning time:0.398ms
- Execution time:0.729 ms
- Part of the plan that has:
 - Highest cost: Index scan using index2 on film category
 - Slowest runtime: Index scan using index2 on film category
 - Largest num of rows: Index scan using index2 on film category has (1000 rows)

Query3:

QUERY PLAN		text	
1	Sort (cost=152.48..152.80 rows=128 width=21) (actual time=4.107..4.120 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=4.028..4.062 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=0.289..2.232 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=0.007..0.468 rows=5462 loops=1)		
10	-> Hash (cost=4.00..4.00 rows=200 width=17) (actual time=0.276..0.276 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..0.041 rows=200 loops=1)		
13	Planning Time: 0.582 ms		
14	Execution Time: 4.367 ms		

Before:

- Planning time:0.582 ms
- Execution time:4.367 ms
- Part of the plan that has:
 - Highest cost: seq scan on film_actor
 - Slowest runtime: seq scan on film_actor
 - Largest num of rows: seq scan on film_actor

After:

	QUERY PLAN	
	text	
1	Sort (cost=194.22..194.54 rows=128 width=21) (actual time=2.010..2.018 rows=199 loops=1)	
2	Sort Key: (count(actor.first_name)) DESC	
3	Sort Method: quicksort Memory: 40kB	
4	-> HashAggregate (cost=188.46..189.74 rows=128 width=21) (actual time=1.955..1.977 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..147.50 rows=5462 width=13) (actual time=0.072..1.054 rows=5462 loops=1)	
8	Hash Cond: (film_actor.actor_id = actor.actor_id)	
9	-> Index Only Scan using index3 on film_actor (cost=0.28..114.21 rows=5462 width=4) (actual time=0.007..0.320 rows=5462 loops=1)	
10	Heap Fetches: 0	
11	-> Hash (cost=16.14..16.14 rows=200 width=17) (actual time=0.059..0.060 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=0.006..0.031 rows=200 loops=1)	
14	Planning Time: 0.186 ms	
15	Execution Time: 2.058 ms	

Index Used: CREATE INDEX INDEX3 ON film_actor USING BTREE(actor_id)

We disabled seqscan technique to force the planner work with our index, Instead of seq scan on film_actor which makes the planner scan the table sequentially now the planner uses index3 Which is BTREE index on attribute actor_id that let us get any row in film_actor table in $O(\log(N))$

- Planning time:0.186 ms
- Execution time:2.058 ms
- Part of the plan that has:
 - Highest cost: index only scan using index3 on film_actor
 - Slowest runtime: Hashaggregate
 - Largest num of rows: index only scan using index3 on film_actor

Query4:

Before:

QUERY PLAN		text	
1	Sort	(cost=136.66..137.16 rows=200 width=44) (actual time=2.854..2.859 rows=24 loops=1)	
2	Sort Key:	c.name, (ntile(4) OVER (?))	
3	Sort Method:	quicksort	Memory: 26kB
4	-> HashAggregate	(cost=127.01..129.01 rows=200 width=44) (actual time=2.657..2.664 rows=24 loops=1)	
5	Group Key:	c.name, ntile(4) OVER (?)	
6	Batches:	1	Memory Usage: 40kB
7	-> WindowAgg	(cost=113.89..120.45 rows=375 width=70) (actual time=2.520..2.587 rows=361 loops=1)	
8	-> Sort	(cost=113.89..114.83 rows=375 width=34) (actual time=1.993..2.015 rows=361 loops=1)	
9	Sort Key:	f.rental_duration	
10	Sort Method:	quicksort	Memory: 41kB
11	-> Hash Join	(cost=25.36..97.86 rows=375 width=34) (actual time=1.370..1.702 rows=361 loops=1)	
12	Hash Cond:	(f.film_id = fc.film_id)	
13	-> Seq Scan on film f	(cost=0.00..65.00 rows=1000 width=6) (actual time=0.022..0.202 rows=1000 loops=1)	
14	-> Hash	(cost=20.67..20.67 rows=375 width=36) (actual time=1.225..1.227 rows=361 loops=1)	
15	Buckets:	1024	Batches: 1
16	-> Hash Join	(cost=1.35..20.67 rows=375 width=36) (actual time=0.402..0.947 rows=361 loops=1)	
17	Hash Cond:	(fc.category_id = c.category_id)	
18	-> Seq Scan on film_category fc	(cost=0.00..16.00 rows=1000 width=8) (actual time=0.007..0.143 rows=1000 loops=1)	
19	-> Hash	(cost=1.28..1.28 rows=6 width=36) (actual time=0.094..0.095 rows=6 loops=1)	
20	Buckets:	1024	Batches: 1
21	-> Seq Scan on category c	(cost=0.00..1.28 rows=6 width=36) (actual time=0.029..0.033 rows=6 loops=1)	
22	Filter:	(name = ANY ('(Animation,Children,Classics,Comedy,Family,Music)':text[]))	
23	Rows Removed by Filter:	10	
24	Planning Time:	1.064 ms	
25	Execution Time:	3.153 ms	

- Planning time:1.064 ms
- Execution time:3.153 ms
- Part of the plan that has:
 - Highest cost: seqscan on film f
 - Slowest runtime: seqscan on film f
 - Largest num of rows: Seq scan on film f and Seq scan on film_category fc(1000 rows)

After:

QUERY PLAN	
	text
1	Sort (cost=199.96..200.46 rows=200 width=44) (actual time=0.789..0.792 rows=24 loops=1)
2	Sort Key: c.name, (ntile(4) OVER (?))
3	Sort Method: quicksort Memory: 26kB
4	-> HashAggregate (cost=190.31..192.31 rows=200 width=44) (actual time=0.763..0.768 rows=24 loops=1)
5	Group Key: c.name, ntile(4) OVER (?)
6	Batches: 1 Memory Usage: 40kB
7	-> WindowAgg (cost=177.19..183.75 rows=375 width=70) (actual time=0.637..0.704 rows=361 loops=1)
8	-> Sort (cost=177.19..178.12 rows=375 width=34) (actual time=0.600..0.613 rows=361 loops=1)
9	Sort Key: f.rental_duration
10	Sort Method: quicksort Memory: 41kB
11	-> Hash Join (cost=60.56..161.15 rows=375 width=34) (actual time=0.241..0.556 rows=361 loops=1)
12	Hash Cond: (f.film_id = fc.film_id)
13	-> Index Scan using film_pkey on film f (cost=0.28..93.37 rows=1000 width=6) (actual time=0.006..0.162 rows=1000 loops=1)
14	-> Hash (cost=55.60..55.60 rows=375 width=36) (actual time=0.191..0.192 rows=361 loops=1)
15	Buckets: 1024 Batches: 1 Memory Usage: 24kB
16	-> Nested Loop (cost=0.29..55.60 rows=375 width=36) (actual time=0.012..0.154 rows=361 loops=1)
17	-> Index Scan using category_pkey on category c (cost=0.14..12.49 rows=6 width=36) (actual time=0.005..0.009 rows=6 loops=1)
18	Filter: (name = ANY ('{Animation,Children,Classics,Comedy,Family,Music}':text[]))
19	Rows Removed by Filter: 10
20	-> Index Scan using index4 on film_category fc (cost=0.15..6.56 rows=62 width=8) (actual time=0.001..0.019 rows=60 loops=6)
21	Index Cond: (category_id = c.category_id)
22	Planning Time: 0.265 ms
23	Execution Time: 0.919 ms


Index Used: CREATE INDEX INDEX4 ON film_category USING BTREE(category_id);

We disabled seqscan technique to force the planner work with our index, Instead of seq scan on film_category which makes the planner scan the table sequentially now the planner uses index4 Which is BTREE index on attribute category_id that let us get any row in film_category table in O(LOG(N)) which enhance the query planning and execution time

- Planning time:0.265 ms
- Execution time:0.919 ms
- Part of the plan that has:
 - Highest cost: Index scan using film_pkey on film f
 - Slowest runtime: Index scan using film_pkey on film f
 - Largest num of rows: Index scan using film_pkey on film f

Query5:

Before:

QUERY PLAN		
	text	
1	Sort (cost=0.03..0.04 rows=1 width=80) (actual time=0.016..0.016 rows=0 loops=1)	
2	Sort Key: (((c.first_name '':text) c.last_name)), (date_trunc('month':text, p.payment_date)), (count(p.amount))	
3	Sort Method: quicksort Memory: 25kB	
4	-> HashAggregate (cost=0.00..0.02 rows=1 width=80) (actual time=0.002..0.002 rows=0 loops=1)	
5	Group Key: ((c.first_name '':text) c.last_name), date_trunc('month':text, p.payment_date)	
6	Batches: 1 Memory Usage: 24kB	
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: false	
9	Planning Time: 5.056 ms	
10	Execution Time: 0.114 ms	

- Planning time:5.056 ms
- Execution time:0.114 ms
- Part of the plan that has:
 - Highest cost : HashAggregate
 - Slowest runtime: HashAggregate
 - Largest num of rows: HashAggregate

After:

QUERY PLAN	
text	
1	Sort (cost=0.03..0.04 rows=1 width=80) (actual time=0.011..0.012 rows=0 loops=1)
2	Sort Key: (((c.first_name '':text) c.last_name)), (date_trunc('month':text, p.payment_date)), (count(p.amount))
3	Sort Method: quicksort Memory: 25kB
4	-> HashAggregate (cost=0.00..0.02 rows=1 width=80) (actual time=0.005..0.005 rows=0 loops=1)
5	Group Key: ((c.first_name '':text) c.last_name), date_trunc('month':text, p.payment_date)
6	Batches: 1 Memory Usage: 24kB
7	-> Result (cost=0.00..0.00 rows=0 width=52) (actual time=0.002..0.002 rows=0 loops=1)
8	One-Time Filter: false
9	Planning Time: 0.895 ms
10	Execution Time: 0.060 ms

Index Used:

- CREATE INDEX INDEX5 ON payment USING BTREE(payment_id) where payment_date BETWEEN '2007-01-01' AND '2008-01-01';
- CREATE INDEX INDEX6 ON payment USING BTREE(customer_id);

Notes: We created index5 and index6 to optimize the query plan,as index5 is a partial index that is optimum for payment dates between 1/2007 and 1/2008,however it didn't work as we discovered that payment dates are only at year 2022,and index6 let us get any row using customer_id in table payment in $O(\log(n))$,however it didn't work too

- Planning time:0.895 ms
- Execution time:0.060 ms
- Part of the plan that has:
 - Highest cost : HashAggregate
 - Slowest runtime: HashAggregate
 - Largest num of rows: HashAggregate

Query6:

Before:

QUERY PLAN		text	
1	Seq Scan on actor	(cost=0.00..4.50 rows=1 width=17) (actual time=0.014..0.025 rows=1 loops=1)	
2	Filter: (first_name ~~ 'JOE'::text)		
3	Rows Removed by Filter: 199		
4	Planning Time: 0.059 ms		
5	Execution Time: 0.036 ms		

- Planning time:0.059 ms
- Execution time:0.036 ms
- Part of the plan that has:
 - Highest cost:Seq scan on actor
 - Slowest runtime: Seq scan on actor
 - Largest num of rows: Seq scan on actor

After:

QUERY PLAN		text	
1	Index Scan using index7 on actor	(cost=0.00..8.02 rows=1 width=17) (actual time=0.076..0.078 rows=1 loops=1)	
2	Index Cond: (first_name = 'JOE'::text)		
3	Filter: (first_name ~~ 'JOE'::text)		
4	Planning Time: 0.380 ms		
5	Execution Time: 0.099 ms		

Index Used: CREATE INDEX INDEX7 ON actor USING HASH(first_name) ;

Notes:We disabled seqscan technique to force the planner work with our index, Instead of seq scan on table actor which makes the planner scan the table sequentially now the planner uses index4 Which is Hash index on attribute first_name that let us get any row in actor table that has first_name like “JOE” in $O(1)$ which is only one row

- Planning time:0.380 ms
- Execution time:0.099 ms
- Part of the plan that has:
 - Highest cost:Index scan using index7 on actor
 - Slowest runtime: Index scan using index7 on actor
 - Largest num of rows: Index scan using index7 on actor

Query7:

Before:

QUERY PLAN	
	text
1	Seq Scan on actor (cost=0.00..5.00 rows=1 width=25) (actual time=0.077..0.077 rows=0 loops=1)
2	Filter: ((last_name ~ '^(?:[GEN].*)\$':text) AND (last_name = 'Smith':text))
3	Rows Removed by Filter: 200
4	Planning Time: 0.233 ms
5	Execution Time: 0.084 ms

- Planning time:0.233 ms
- Execution time:0.084 ms
- Part of the plan that has:
 - Highest cost:Seq scan on actor
 - Slowest runtime: Seq scan on actor
 - Largest num of rows: Seq scan on actor

After:

QUERY PLAN	
	text
1	Index Scan using index8 on actor (cost=0.00..8.02 rows=1 width=25) (actual time=0.010..0.011 rows=0 loops=1)
2	Index Cond: (last_name = 'Smith':text)
3	Filter: (last_name ~ '^(?:[GEN].*)\$':text)
4	Planning Time: 0.663 ms
5	Execution Time: 0.030 ms

Index Used: CREATE INDEX INDEX8 ON actor USING HASH(last_name) ;

Notes:We disabled seqscan technique to force the planner work with our index, Instead of seq scan on table actor which makes the planner scan the table sequentially now the planner uses index8 Which is Hash index on attribute last_name that let us get any row in actor table that has last_name similar to '%[GEN]%' and last_name = 'Smith' in O(1)

- Planning time:0.663 ms
- Execution time:0.030 ms
- Part of the plan that has:
 - Highest cost:Index Scan using index8
 - Slowest runtime: Index Scan using index8
 - Largest num of rows: Index Scan using index8

Query8:

	QUERY PLAN text	
1	Seq Scan on country (cost=0.00..2.50 rows=3 width=13) (actual time=1.259..1.268 rows=3 loops=1)	
2	Filter: (country = ANY ({Afghanistan,Bangladesh,China}::text[]))	
3	Rows Removed by Filter: 106	
4	Planning Time: 1.768 ms	
5	Execution Time: 1.282 ms	

Before:

- Planning time:1.768 ms
- Execution time:1.282 ms
- Part of the plan that has:
 - Highest cost:Seq scan on country
 - Slowest runtime: Seq scan on country
 - Largest num of rows: Seq scan on country(3)

After:

	QUERY PLAN text	
1	Index Scan using index9 on country (cost=0.13..12.18 rows=3 width=13) (actual time=0.024..0.026 rows=3 loops=1)	
2	Planning Time: 0.661 ms	
3	Execution Time: 0.046 ms	

Index Used: CREATE INDEX INDEX9 ON country USING BTREE(country) WHERE country IN ('Afghanistan', 'Bangladesh', 'China');

Notes:We disabled seqscan technique to force the planner work with our index, Instead of seq scan on table actor which makes the planner scan the table sequentially now the planner uses index9 Which is BTREE partial index on attribute country that let us get any row in country table that has country equal to any of these countries('Afghanistan', 'Bangladesh', 'China') in $O(\log(n))$

- Planning time:0.661 ms
- Execution time:0.046 ms
- Part of the plan that has:
 - Highest cost: Index scan using index9 on country
 - Slowest runtime: Index scan using index9 on country
 - Largest num of rows:Index scan using index9 on country

Query9:

Before:

	QUERY PLAN	
	text	
1	HashAggregate (cost=5.00..6.21 rows=121 width=15) (actual time=0.664..0.677 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=0.216..0.231 rows=200 loops=1)	
5	Planning Time: 0.286 ms	
6	Execution Time: 1.306 ms	

- Planning time:0.118 ms
- Execution time:0.111 ms
- Part of the plan that has:
 - Highest cost:Seq scan on actor
 - Slowest runtime:Hash aggregate
 - Largest num of rows: Seq scan on actor

After:

We didn't use any index as there is nothing to optimize here,the query will count all last_names in table actor and group each last_name with its count

Query10:

Before:

	QUERY PLAN
	text
1	HashAggregate (cost=4.54..4.62 rows=8 width=15) (actual time=0.030..0.031 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=0.009..0.023 rows=5 loops=1)
5	Filter: (first_name ~~ '%D':text)
6	Rows Removed by Filter: 195
7	Planning Time: 0.115 ms
8	Execution Time: 0.058 ms

- Planning time:0.115 ms
- Execution time:0.058 ms
- Part of the plan that has:
 - Highest cost: Seq scan on actor
 - Slowest runtime: Seq scan on actor
 - Largest num of rows: Seq scan on actor

After:

	QUERY PLAN
	text
1	HashAggregate (cost=6.34..6.42 rows=8 width=15) (actual time=0.343..0.346 rows=5 loops=1)
2	Group Key: last_name
3	Batches: 1 Memory Usage: 24kB
4	-> Bitmap Heap Scan on actor (cost=4.20..6.30 rows=8 width=7) (actual time=0.302..0.334 rows=5 loops=1)
5	Recheck Cond: (first_name ~~ '%D':text)
6	Rows Removed by Index Recheck: 195
7	Heap Blocks: exact=2
8	-> Bitmap Index Scan on index10 (cost=0.00..4.20 rows=8 width=0) (actual time=0.285..0.285 rows=200 loops=1)
9	Index Cond: (first_name ~~ '%D':text)
10	Planning Time: 0.752 ms
11	Execution Time: 0.405 ms

Index Used: CREATE INDEX INDEX10 ON actor USING gist(first_name gist_trgm_ops) ;

Notes:We disabled seqscan technique to force the planner work with our index, Instead of seq scan on table actor which makes the planner scan the table sequentially now the planner uses index10 Which is gist index on attribute first_name in table actor, we used gist as we searched for the best indices on like operator and found gist and gin are the best.

- Planning time:0.752 ms
- Execution time:0.305 ms
- Part of the plan that has:
 - Highest cost:Bitmap index scan on index10
 - Slowest runtime: Bitmap index scan on index10
 - Largest num of rows: Bitmap index scan on index10

Query11:

Before:

	QUERY PLAN	
	text	
1	Seq Scan on actor (cost=0.00..5.00 rows=1 width=4) (actual time=0.025..0.025 rows=0 loops=1)	
2	Filter: ((first_name = 'HARPO'::text) AND (last_name = 'WILLIAMS'::text))	
3	Rows Removed by Filter: 200	
4	Planning Time: 0.099 ms	
5	Execution Time: 0.037 ms	

- Planning time:0.099 ms
- Execution time:0.047 ms
- Part of the plan that has:
 - Highest cost: Seq scan on actor
 - Slowest runtime: Seq scan on actor
 - Largest num of rows: Seq scan on actor

After

	QUERY PLAN	
	text	
1	Bitmap Heap Scan on actor (cost=4.02..6.07 rows=1 width=4) (actual time=0.027..0.028 rows=0 loops=1)	
2	Recheck Cond: (last_name = 'WILLIAMS'::text)	
3	Filter: (first_name = 'HARPO'::text)	
4	Rows Removed by Filter: 3	
5	Heap Blocks: exact=2	
6	-> Bitmap Index Scan on index11 (cost=0.00..4.02 rows=3 width=0) (actual time=0.015..0.016 rows=3 loops=1)	
7	Index Cond: (last_name = 'WILLIAMS'::text)	
8	Planning Time: 0.081 ms	
9	Execution Time: 0.366 ms	

Index Used: CREATE INDEX INDEX11 ON actor USING HASH(last_name);

Notes:We disabled seqscan technique to force the planner work with our index, Instead of seq scan on table actor which makes the planner scan the table sequentially now the planner uses bitmap index on index11 Which is Hash index on attribute last_name in table actor,We tried partial index WHERE first_name = 'HARPO' AND last_name = 'WILLIAMS' but the planner ignore it and uses hash index on last_name which let us get any row in table actor using last_name attribute in $O(1)$

- Planning time:0.081 ms
- Execution time:0.366 ms
- Part of the plan that has:
 - Highest cost: Bitmap index scan on index11
 - Slowest runtime: Bitmap heap scan on actor
 - Largest num of rows: Bitmap index scan on index11

Query12:

Before:

	QUERY PLAN text	
1	Hash Right Join (cost=1.04..17.36 rows=2 width=84) (actual time=1.573..3.428 rows=2 loops=1)	
2	Hash Cond: (a.address_id = s.address_id)	
3	-> Seq Scan on address a (cost=0.00..14.03 rows=603 width=24) (actual time=0.719..2.530 rows=603 loops=1)	
4	-> Hash (cost=1.02..1.02 rows=2 width=68) (actual time=0.844..0.846 rows=2 loops=1)	
5	Buckets: 1024 Batches: 1 Memory Usage: 9kB	
6	-> Seq Scan on staff s (cost=0.00..1.02 rows=2 width=68) (actual time=0.839..0.840 rows=2 loops=1)	
7	Planning Time: 3.929 ms	
8	Execution Time: 3.451 ms	

- Planning time:3.929 ms
- Execution time:3.451 ms
- Part of the plan that has:
 - Highest cost: Seq scan on address a
 - Slowest runtime: Seq scan on address a
 - Largest num of rows: Seq scan on address a

After:

	QUERY PLAN text	
1	Nested Loop Left Join (cost=0.40..28.77 rows=2 width=84) (actual time=0.168..0.174 rows=2 loops=1)	
2	-> Index Scan using index12 on staff s (cost=0.13..12.16 rows=2 width=68) (actual time=0.013..0.014 rows=2 loops=1)	
3	-> Index Scan using address_pkey on address a (cost=0.28..8.29 rows=1 width=24) (actual time=0.076..0.077 rows=1 loops=2)	
4	Index Cond: (address_id = s.address_id)	
5	Planning Time: 0.088 ms	
6	Execution Time: 0.192 ms	

Index Used: CREATE INDEX INDEX12 ON staff USING BTREE(address_id);

Notes: We disabled seqscan technique to force the planner work with our index, Instead of seq scan on table staff which makes the planner scan the table sequentially now the planner uses index12 Which is BTREE index on attribute address_id in table staff, which let us get any row in table staff using address_id attribute in $O(\log(n))$

- Planning time:0.088 ms
- Execution time:0.192 ms
- Part of the plan that has:
 - Highest cost: Index scan using index12 on staff s
 - Slowest runtime: Index scan using address_pkey on address a
 - Largest num of rows: Index scan using address_pkey on address a

Query13:

before:

QUERY PLAN		text	
1	HashAggregate	(cost=441.98..443.98 rows=200 width=12) (actual time=4.300..4.303 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=0.022..2.577 rows=16049 loops=1)	
5	-> Seq Scan on payment_p2022_01 p_1	(cost=0.00..13.23 rows=723 width=10) (actual time=0.021..0.120 rows=723 loops=1)	
6	-> Seq Scan on payment_p2022_02 p_2	(cost=0.00..42.01 rows=2401 width=10) (actual time=0.020..0.227 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..0.244 rows=2713 loops=1)	
8	-> Seq Scan on payment_p2022_04 p_4	(cost=0.00..44.47 rows=2547 width=10) (actual time=0.010..0.293 rows=2547 loops=1)	
9	-> Seq Scan on payment_p2022_05 p_5	(cost=0.00..46.77 rows=2677 width=10) (actual time=0.005..0.238 rows=2677 loops=1)	
10	-> Seq Scan on payment_p2022_06 p_6	(cost=0.00..46.54 rows=2654 width=10) (actual time=0.003..0.220 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..0.177 rows=2334 loops=1)	
12	Planning Time:	0.283 ms	
13	Execution Time:	4.368 ms	

- Planning time:0.282 ms
- Execution time:4.368 ms
- Part of the plan that has:
 - Highest cost: HashAggregate
 - Slowest runtime: HashAggregate
 - Largest num of rows: Append

After:

QUERY PLAN	
text	
1	HashAggregate (cost=918.76..920.76 rows=200 width=12) (actual time=4.618..4.642 rows=2 loops=1)
2	Group Key: p.staff_id
3	Batches: 1 Memory Usage: 40kB
4	-> Append (cost=0.15..838.51 rows=16049 width=10) (actual time=0.014..3.026 rows=16049 loops=1)
5	-> Index Scan using payment_p2022_01_staff_id_idx on payment_p2022_01 p_1 (cost=0.15..39.27 rows=723 width=10) (actual time=0.013..0.105 rows=723 loops=1)
6	-> Index Scan using payment_p2022_02_staff_id_idx on payment_p2022_02 p_2 (cost=0.28..111.95 rows=2401 width=10) (actual time=0.005..0.325 rows=2401 loops=1)
7	-> Index Scan using payment_p2022_03_staff_id_idx on payment_p2022_03 p_3 (cost=0.28..125.62 rows=2713 width=10) (actual time=0.006..0.399 rows=2713 loops=1)
8	-> Index Scan using payment_p2022_04_staff_id_idx on payment_p2022_04 p_4 (cost=0.28..118.92 rows=2547 width=10) (actual time=0.006..0.322 rows=2547 loops=1)
9	-> Index Scan using payment_p2022_05_staff_id_idx on payment_p2022_05 p_5 (cost=0.28..125.32 rows=2677 width=10) (actual time=0.005..0.334 rows=2677 loops=1)
10	-> Index Scan using payment_p2022_06_staff_id_idx on payment_p2022_06 p_6 (cost=0.28..125.27 rows=2654 width=10) (actual time=0.006..0.375 rows=2654 loops=1)
11	-> Index Scan using payment_p2022_07_staff_id_idx on payment_p2022_07 p_7 (cost=0.28..111.91 rows=2334 width=10) (actual time=0.008..0.380 rows=2334 loops=1)
12	Planning Time: 0.129 ms
13	Execution Time: 4.683 ms

Index Used: CREATE INDEX INDEX13 ON payment USING BTREE(staff_id);

Notes: We disabled seqscan technique to force the planner work with our index, Instead of seq scan on table payment which makes the planner scan the table sequentially now the planner uses index13 Which is BTREE index on attribute staff_id in table payment, which let us get any row in table payment using staff_id attribute in $O(\log(n))$

- Planning time: 0.129 ms
- Execution time: 3.683 ms
- Part of the plan that has:
 - Highest cost: Index scan using payment_p2022_03_staff_id_idx on payment_p2022_03_p_3
 - Slowest runtime: HashAggregate
 - Largest num of rows: Append

Query14:

Before:

QUERY PLAN		text	
1	Sort	(cost=48.87..48.88 rows=5 width=45) (actual time=4.247..4.257 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=3.652..4.134 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=0.755..2.045 rows=599 loops=1)	
7	-> Hash	(cost=31.44..31.44 rows=6 width=4) (actual time=1.982..1.989 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=1.199..1.985 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=0.028..0.755 rows=603 loops=1)	
12	-> Hash	(cost=15.02..15.02 rows=6 width=4) (actual time=1.063..1.068 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=0.297..1.062 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..0.902 rows=600 loops=1)	
17	-> Hash	(cost=2.36..2.36 rows=1 width=4) (actual time=0.037..0.039 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..0.016 rows=1 loops=1)	
20	Filter:	(country = 'Canada':text)	
21	Rows Removed by Filter:	108	
22	Planning Time:	6.371 ms	
23	Execution Time:	4.459 ms	

- Planning time:6.371 ms
- Execution time:4.459 ms
- Part of the plan that has:
 - Highest cost: Seq scan on customer c
 - Slowest runtime: Hash join in row 4
 - Largest num of rows: Seq scan on address a

After:

	QUERY PLAN	
	text	
1	Sort (cost=13.20..13.21 rows=5 width=45) (actual time=0.119..0.120 rows=5 loops=1)	
2	Sort Key: c.last_name	
3	Sort Method: quicksort Memory: 25kB	
4	-> Nested Loop (cost=4.05..13.14 rows=5 width=45) (actual time=0.059..0.084 rows=5 loops=1)	
5	-> Nested Loop (cost=4.05..12.46 rows=6 width=4) (actual time=0.052..0.070 rows=7 loops=1)	
6	-> Nested Loop (cost=4.05..11.81 rows=6 width=4) (actual time=0.026..0.035 rows=7 loops=1)	
7	-> Seq Scan on country co (cost=0.00..2.36 rows=1 width=4) (actual time=0.009..0.014 rows=1 loops=1)	
8	Filter: (country = 'Canada':text)	
9	Rows Removed by Filter: 108	
10	-> Bitmap Heap Scan on city ci (cost=4.05..9.39 rows=6 width=8) (actual time=0.015..0.018 rows=7 loops=1)	
11	Recheck Cond: (country_id = co.country_id)	
12	Heap Blocks: exact=3	
13	-> Bitmap Index Scan on index16 (cost=0.00..4.04 rows=6 width=0) (actual time=0.010..0.010 rows=7 loops=1)	
14	Index Cond: (country_id = co.country_id)	
15	-> Index Scan using index15 on address a (cost=0.00..0.10 rows=1 width=8) (actual time=0.004..0.005 rows=1 loops=7)	
16	Index Cond: (city_id = ci.city_id)	
17	-> Index Scan using index14 on customer c (cost=0.00..0.10 rows=1 width=49) (actual time=0.002..0.002 rows=1 loops=7)	
18	Index Cond: (address_id = a.address_id)	
19	Planning Time: 0.613 ms	
20	Execution Time: 0.152 ms	

Index Used:

```
CREATE INDEX INDEX14 ON customer USING HASH(address_id);
CREATE INDEX INDEX15 ON address USING HASH(city_id);
CREATE INDEX INDEX16 ON city USING HASH(country_id);
```

Notes : Instead of sequential scan on tables customer, address, and city, we created hash indexes on these tables foreign keys which let us get any row in these tables on O(1) on each of the above attributes(address_id, city_id, country_id)

- Planning time: 0.613 ms
- Execution time: 0.152 ms
- Part of the plan that has:
 - Highest cost: Bitmap heap scan on city ci
 - Slowest runtime: Bitmap heap scan on city ci
 - Largest num of rows: Bitmap heap scan on city ci

Query15:

Before:

QUERY PLAN	
text	
1	Nested Loop (cost=19.36..88.89 rows=69 width=15) (actual time=0.314..3.432 rows=69 loops=1)
2	-> Seq Scan on category c (cost=0.00..1.20 rows=1 width=4) (actual time=0.034..0.036 rows=1 loops=1)
3	Filter: (category_id = 8)
4	Rows Removed by Filter: 15
5	-> Hash Join (cost=19.36..87.00 rows=69 width=19) (actual time=0.278..3.382 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=0.023..2.988 rows=1000 loops=1)
8	-> Hash (cost=18.50..18.50 rows=69 width=8) (actual time=0.214..0.215 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=0.025..0.201 rows=69 loops=1)
11	Filter: (category_id = 8)
12	Rows Removed by Filter: 931
13	Planning Time: 8.395 ms
14	Execution Time: 3.470 ms

- Planning time:8.395 ms
- Execution time:3.470 ms
- Part of the plan that has:
 - Highest cost: seq scan on film f
 - Slowest runtime: seq scan on film f
 - Largest num of rows: seq scan on film f

After:

QUERY PLAN	
	text
1	Nested Loop (cost=12.26..81.79 rows=69 width=15) (actual time=0.058..0.700 rows=69 loops=1)
2	-> Seq Scan on category c (cost=0.00..1.20 rows=1 width=4) (actual time=0.007..0.009 rows=1 loops=1)
3	Filter: (category_id = 8)
4	Rows Removed by Filter: 15
5	-> Hash Join (cost=12.26..79.90 rows=69 width=19) (actual time=0.049..0.682 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=0.003..0.541 rows=1000 loops=1)
8	-> Hash (cost=11.40..11.40 rows=69 width=8) (actual time=0.038..0.039 rows=69 loops=1)
9	Buckets: 1024 Batches: 1 Memory Usage: 11kB
10	-> Bitmap Heap Scan on film_category fc (cost=4.53..11.40 rows=69 width=8) (actual time=0.015..0.031 rows=69 loops=1)
11	Recheck Cond: (category_id = 8)
12	Heap Blocks: exact=6
13	-> Bitmap Index Scan on index17 (cost=0.00..4.52 rows=69 width=0) (actual time=0.010..0.010 rows=69 loops=1)
14	Index Cond: (category_id = 8)
15	Planning Time: 1.849 ms
16	Execution Time: 0.752 ms

Index Used: CREATE INDEX INDEX17 ON film_category USING HASH(category_id);

Notes: Instead of sequential scan on film_category table filtering rows that have category_id = 8, now the planner uses index17 Which is Hash index on attribute category_id in table film_category, which let us get any row in table film_category that its category_id = 8 in o(1)

- Planning time: 1.849 ms
- Execution time: 0.752 ms
- Part of the plan that has:
 - Highest cost: seq scan on film f
 - Slowest runtime: seq scan on film f
 - Largest num of rows: seq scan on film f

Query16:

Before:

	QUERY PLAN	
	text	
1	HashAggregate (cost=441.71..443.71 rows=160 width=36) (actual time=7.936..7.941 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=0.309..5.440 rows=16049 loops=1)	
5	Hash Cond: (p.staff_id = stf.staff_id)	
6	-> Append (cost=0.00..361.74 rows=16049 width=10) (actual time=0.012..2.876 rows=16049 loops=1)	
7	-> Seq Scan on payment_p2022_01 p_1 (cost=0.00..13.23 rows=723 width=10) (actual time=0.011..0.084 rows=723 loops=1)	
8	-> Seq Scan on payment_p2022_02 p_2 (cost=0.00..42.01 rows=2401 width=10) (actual time=0.013..0.303 rows=2401 loops=1)	
9	-> Seq Scan on payment_p2022_03 p_3 (cost=0.00..47.13 rows=2713 width=10) (actual time=0.063..0.330 rows=2713 loops=1)	
10	-> Seq Scan on payment_p2022_04 p_4 (cost=0.00..44.47 rows=2547 width=10) (actual time=0.009..0.239 rows=2547 loops=1)	
11	-> Seq Scan on payment_p2022_05 p_5 (cost=0.00..46.77 rows=2677 width=10) (actual time=0.010..0.308 rows=2677 loops=1)	
12	-> Seq Scan on payment_p2022_06 p_6 (cost=0.00..46.54 rows=2654 width=10) (actual time=0.007..0.299 rows=2654 loops=1)	
13	-> Seq Scan on payment_p2022_07 p_7 (cost=0.00..41.34 rows=2334 width=10) (actual time=0.005..0.201 rows=2334 loops=1)	
14	-> Hash (cost=17.37..17.37 rows=2 width=8) (actual time=0.129..0.131 rows=2 loops=1)	
15	Buckets: 1024 Batches: 1 Memory Usage: 9kB	
16	-> Nested Loop (cost=0.15..17.37 rows=2 width=8) (actual time=0.122..0.127 rows=2 loops=1)	
17	-> Seq Scan on staff stf (cost=0.00..1.02 rows=2 width=8) (actual time=0.009..0.010 rows=2 loops=1)	
18	-> Index Only Scan using store_pkey on store str (cost=0.15..8.17 rows=1 width=4) (actual time=0.055..0.055 rows=1 loops=2)	
19	Index Cond: (store_id = stf.store_id)	
20	Heap Fetches: 2	
21	Planning Time: 1.326 ms	
22	Execution Time: 7.993 ms	

- Planning time:1.326 ms
- Execution time:7.993 ms
- Part of the plan that has:
 - Highest cost : Append
 - Slowest runtime: HashAggregate
 - Largest num of rows: Append

After:

	QUERY PLAN text	
1	GroupAggregate (cost=7.24..924.76 rows=160 width=36) (actual time=3.348..6.254 rows=2 loops=1)	
2	Group Key: str.store_id	
3	-> Nested Loop (cost=7.24..921.96 rows=160 width=10) (actual time=0.044..4.583 rows=16049 loops=1)	
4	-> Nested Loop (cost=0.28..28.51 rows=2 width=8) (actual time=0.023..0.035 rows=2 loops=1)	
5	-> Index Scan using index18 on staff stf (cost=0.13..12.16 rows=2 width=8) (actual time=0.016..0.020 rows=2 loops=1)	
6	-> Index Only Scan using store_pkey on store str (cost=0.15..8.17 rows=1 width=4) (actual time=0.004..0.004 rows=1 loops=2)	
7	Index Cond: (store_id = stf.store_id)	
8	Heap Fetches: 2	
9	-> Append (cost=6.96..366.48 rows=8024 width=10) (actual time=0.018..1.731 rows=8024 loops=2)	
10	-> Bitmap Heap Scan on payment_p2022_01 p_1 (cost=6.96..17.12 rows=362 width=10) (actual time=0.015..0.053 rows=362 loops=2)	
11	Recheck Cond: (staff_id = stf.staff_id)	
12	Heap Blocks: exact=12	
13	-> Bitmap Index Scan on payment_p2022_01_staff_id_idx (cost=0.00..6.87 rows=362 width=0) (actual time=0.009..0.010 rows=362 loops=2)	
14	Index Cond: (staff_id = stf.staff_id)	
15	-> Bitmap Heap Scan on payment_p2022_02 p_2 (cost=15.58..47.49 rows=1200 width=10) (actual time=0.057..0.183 rows=1200 loops=2)	
16	Recheck Cond: (staff_id = stf.staff_id)	
17	Heap Blocks: exact=36	
18	-> Bitmap Index Scan on payment_p2022_02_staff_id_idx (cost=0.00..15.28 rows=1200 width=0) (actual time=0.051..0.052 rows=1200 loops=2)	
19	Index Cond: (staff_id = stf.staff_id)	
20	-> Bitmap Heap Scan on payment_p2022_03 p_3 (cost=18.79..54.53 rows=1356 width=10) (actual time=0.044..0.191 rows=1356 loops=2)	
21	Recheck Cond: (staff_id = stf.staff_id)	
22	Heap Blocks: exact=40	
23	-> Bitmap Index Scan on payment_p2022_03_staff_id_idx (cost=0.00..18.45 rows=1356 width=0) (actual time=0.032..0.032 rows=1356 loops=2)	
24	Index Cond: (staff_id = stf.staff_id)	
25	-> Bitmap Heap Scan on payment_p2022_04 p_4 (cost=18.15..52.31 rows=1274 width=10) (actual time=0.030..0.172 rows=1274 loops=2)	
26	Recheck Cond: (staff_id = stf.staff_id)	
27	Heap Blocks: exact=38	
28	-> Bitmap Index Scan on payment_p2022_04_staff_id_idx (cost=0.00..17.84 rows=1274 width=0) (actual time=0.025..0.025 rows=1274 loops=2)	
29	Index Cond: (staff_id = stf.staff_id)	
30	-> Bitmap Heap Scan on payment_p2022_05 p_5 (cost=18.65..54.16 rows=1338 width=10) (actual time=0.053..0.243 rows=1338 loops=2)	
31	Recheck Cond: (staff_id = stf.staff_id)	
32	Heap Blocks: exact=40	
33	-> Bitmap Index Scan on payment_p2022_05_staff_id_idx (cost=0.00..18.32 rows=1338 width=0) (actual time=0.042..0.043 rows=1338 loops=2)	
34	Index Cond: (staff_id = stf.staff_id)	
35	-> Bitmap Heap Scan on payment_p2022_06 p_6 (cost=18.56..53.94 rows=1327 width=10) (actual time=0.047..0.255 rows=1327 loops=2)	
36	Recheck Cond: (staff_id = stf.staff_id)	
37	Heap Blocks: exact=40	
38	-> Bitmap Index Scan on payment_p2022_06_staff_id_idx (cost=0.00..18.23 rows=1327 width=0) (actual time=0.040..0.040 rows=1327 loops=2)	
39	Index Cond: (staff_id = stf.staff_id)	
40	-> Bitmap Heap Scan on payment_p2022_07 p_7 (cost=15.32..46.82 rows=1167 width=10) (actual time=0.040..0.185 rows=1167 loops=2)	
41	Recheck Cond: (staff_id = stf.staff_id)	
42	Heap Blocks: exact=36	
43	-> Bitmap Index Scan on payment_p2022_07_staff_id_idx (cost=0.00..15.03 rows=1167 width=0) (actual time=0.034..0.034 rows=1167 loops=2)	
44	Index Cond: (staff_id = stf.staff_id)	
45	Planning Time: 0.768 ms	
46	Execution Time: 6.321 ms	

Index Used:

```
CREATE INDEX INDEX18 ON staff USING BTREE(store_id);  
CREATE INDEX INDEX19 ON payment USING BTREE(staff_id);
```

Notes: We disabled seqscan technique to force the planner work with our index, Instead of seq scan on tables staff and payment which makes the planner scan the table sequentially now the planner uses indexes 18 and 19 Which are BTREE indexes on attributes staff_id in table payment, and store_id in table staff which let us get any row in these tables using these attributes in $O(\log(n))$

- Planning time: 0.768 ms
- Execution time: 6.321 ms
- Part of the plan that has:
 - Highest cost : Nested loop
 - Slowest runtime: GroupAggregate
 - Largest num of rows : Nested loop

Query17:

Before:

	QUERY PLAN	
	text	
1	Limit (cost=1424.67..1424.68 rows=5 width=64) (actual time=155.099..155.106 rows=5 loops=1)	
2	-> Sort (cost=1424.67..1424.71 rows=16 width=64) (actual time=155.097..155.104 rows=5 loops=1)	
3	Sort Key: (sum(p.amount)) DESC	
4	Sort Method: top-N heapsort Memory: 25kB	
5	-> HashAggregate (cost=1424.20..1424.40 rows=16 width=64) (actual time=155.046..155.056 rows=16 loops=1)	
6	Group Key: c.name	
7	Batches: 1 Memory Usage: 24kB	
8	-> Hash Join (cost=677.24..1343.96 rows=16049 width=38) (actual time=141.724..151.952 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=141.298..149.100 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=115.340..120.630 rows=16049 loops=1)	
13	Hash Cond: (p.rental_id = r.rental_id)	
14	-> Append (cost=0.00..361.74 rows=16049 width=10) (actual time=0.022..2.334 rows=16049 loops=1)	
15	-> Seq Scan on payment_p2022_01 p_1 (cost=0.00..13.23 rows=723 width=10) (actual time=0.021..0.085 rows=723 loops=1)	
16	-> Seq Scan on payment_p2022_02 p_2 (cost=0.00..42.01 rows=2401 width=10) (actual time=0.017..0.218 rows=2401 loops=1)	
17	-> Seq Scan on payment_p2022_03 p_3 (cost=0.00..47.13 rows=2713 width=10) (actual time=0.005..0.235 rows=2713 loops=1)	
18	-> Seq Scan on payment_p2022_04 p_4 (cost=0.00..44.47 rows=2547 width=10) (actual time=0.014..0.233 rows=2547 loops=1)	
19	-> Seq Scan on payment_p2022_05 p_5 (cost=0.00..46.77 rows=2677 width=10) (actual time=0.008..0.299 rows=2677 loops=1)	
20	-> Seq Scan on payment_p2022_06 p_6 (cost=0.00..46.54 rows=2654 width=10) (actual time=0.005..0.216 rows=2654 loops=1)	
21	-> Seq Scan on payment_p2022_07 p_7 (cost=0.00..41.34 rows=2334 width=10) (actual time=0.003..0.214 rows=2334 loops=1)	
22	-> Hash (cost=310.44..310.44 rows=16044 width=8) (actual time=115.245..115.245 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=1.721..110.617 rows=16044 loops=1)	
25	-> Hash (cost=75.81..75.81 rows=4581 width=8) (actual time=25.916..25.916 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=0.072..24.977 rows=4581 loops=1)	
28	-> Hash (cost=20.67..20.67 rows=1000 width=36) (actual time=0.417..0.419 rows=1000 loops=1)	
29	Buckets: 1024 Batches: 1 Memory Usage: 52kB	
30	-> Hash Join (cost=1.36..20.67 rows=1000 width=36) (actual time=0.024..0.270 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=0.006..0.062 rows=1000 loops=1)	
33	-> Hash (cost=1.16..1.16 rows=16 width=36) (actual time=0.010..0.010 rows=16 loops=1)	
34	Buckets: 1024 Batches: 1 Memory Usage: 9kB	
35	-> Seq Scan on category c (cost=0.00..1.16 rows=16 width=36) (actual time=0.004..0.006 rows=16 loops=1)	
36	Planning Time: 4.311 ms	
37	Execution Time: 155.214 ms	

- Planning time:4.311 ms
- Execution time:155.214 ms
- Part of the plan that has:
 - Highest cost:Hash join in row 8
 - Slowest runtime: Hash join in row 8
 - Largest num of rows: Hash join and append

After:

	QUERY PLAN	
	text	
1	Limit (cost=1424.67..1424.68 rows=5 width=64) (actual time=21.602..21.611 rows=5 loops=1)	
2	-> Sort (cost=1424.67..1424.71 rows=16 width=64) (actual time=21.600..21.608 rows=5 loops=1)	
3	Sort Key: (sum(p.amount)) DESC	
4	Sort Method: top-N heapsort Memory: 25kB	
5	-> HashAggregate (cost=1424.20..1424.40 rows=16 width=64) (actual time=21.581..21.592 rows=16 loops=1)	
6	Group Key: c.name	
7	Batches: 1 Memory Usage: 24kB	
8	-> Hash Join (cost=677.24..1343.96 rows=16049 width=38) (actual time=4.721..18.372 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=4.381..15.564 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=3.463..12.051 rows=16049 loops=1)	
13	Hash Cond: (p.rental_id = r.rental_id)	
14	-> Append (cost=0.00..361.74 rows=16049 width=10) (actual time=0.012..4.688 rows=16049 loops=1)	
15	-> Seq Scan on payment_p2022_01 p_1 (cost=0.00..13.23 rows=723 width=10) (actual time=0.011..0.075 rows=723 loops=1)	
16	-> Seq Scan on payment_p2022_02 p_2 (cost=0.00..42.01 rows=2401 width=10) (actual time=0.014..0.242 rows=2401 loops=1)	
17	-> Seq Scan on payment_p2022_03 p_3 (cost=0.00..47.13 rows=2713 width=10) (actual time=0.008..0.249 rows=2713 loops=1)	
18	-> Seq Scan on payment_p2022_04 p_4 (cost=0.00..44.47 rows=2547 width=10) (actual time=0.010..0.245 rows=2547 loops=1)	
19	-> Seq Scan on payment_p2022_05 p_5 (cost=0.00..46.77 rows=2677 width=10) (actual time=0.011..0.260 rows=2677 loops=1)	
20	-> Seq Scan on payment_p2022_06 p_6 (cost=0.00..46.54 rows=2654 width=10) (actual time=0.040..0.266 rows=2654 loops=1)	
21	-> Seq Scan on payment_p2022_07 p_7 (cost=0.00..41.34 rows=2334 width=10) (actual time=0.010..0.216 rows=2334 loops=1)	
22	-> Hash (cost=310.44..310.44 rows=16044 width=8) (actual time=3.425..3.426 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..1.866 rows=16044 loops=1)	
25	-> Hash (cost=75.81..75.81 rows=4581 width=8) (actual time=0.870..0.870 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=0.008..0.434 rows=4581 loops=1)	
28	-> Hash (cost=20.67..20.67 rows=1000 width=36) (actual time=0.332..0.333 rows=1000 loops=1)	
29	Buckets: 1024 Batches: 1 Memory Usage: 52kB	
30	-> Hash Join (cost=1.36..20.67 rows=1000 width=36) (actual time=0.048..0.230 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=0.012..0.065 rows=1000 loops=1)	
33	-> Hash (cost=1.16..1.16 rows=16 width=36) (actual time=0.013..0.014 rows=16 loops=1)	
34	Buckets: 1024 Batches: 1 Memory Usage: 9kB	
35	-> Seq Scan on category c (cost=0.00..1.16 rows=16 width=36) (actual time=0.004..0.006 rows=16 loops=1)	
36	Planning Time: 0.717 ms	
37	Execution Time: 21.690 ms	

Index Used:

```
CREATE INDEX INDEX20 ON film_category USING BTREE(film_id);
CREATE INDEX INDEX21 ON inventory USING BTREE(film_id);
CREATE INDEX INDEX22 ON rental USING BTREE(inventory_id);
```

Notes: These indexes didn't show up in the planner however the planner uses them as shown below that these indexes are used, indexes 20, 21 and 22 are BTREE indexes on attributes film_id in tables film_category and inventory, and inventory_id in table rental which let us get any row in these tables using these attributes in $O(\log(n))$, moreover plan time and execution time become significantly better

- Planning time:0.717 ms
- Execution time:21.690 ms
- Part of the plan that has:
 - Highest cost: seq scan
 - Slowest runtime: hash join
 - Largest num of rows: hash join

indexrelname	idx_scan	idx_tup_read	idx_tup_fetch
name	bigint	bigint	bigint
pgbench_branches_pkey	0	0	0
pgbench_tellers_pkey	0	0	0
pgbench_accounts_pkey	0	0	0
actor_pkey	63	1456	1456
address_pkey	19	621	621
category_pkey	27	432	432
city_pkey	13	612	612
country_pkey	6	6	6
customer_pkey	0	0	0
film_actor_pkey	14	14	0
film_category_pkey	1021	6016	2016
film_pkey	1853	9845	9845
inventory_pkey	40	40	0
language_pkey	1000	1000	1000
rental_pkey	10	160440	160440
staff_pkey	0	0	0
store_pkey	28	28	28
index20	37	9028	9028
index21	38	45838	45810
index22	28	28	0

Query18:

Before:

	QUERY PLAN text
1	Sort (cost=1424.72..1424.76 rows=16 width=64) (actual time=17.392..17.400 rows=16 loops=1)
2	Sort Key: (sum(p.amount)) DESC
3	Sort Method: quicksort Memory: 25kB
4	-> HashAggregate (cost=1424.20..1424.40 rows=16 width=64) (actual time=17.375..17.386 rows=16 loops=1)
5	Group Key: c.name
6	Batches: 1 Memory Usage: 24kB
7	-> Hash Join (cost=677.24..1343.96 rows=16049 width=38) (actual time=3.904..14.151 rows=16049 loops=1)
8	Hash Cond: (i.film_id = fc.film_id)
9	-> Hash Join (cost=644.06..1090.11 rows=16049 width=10) (actual time=3.565..11.445 rows=16049 loops=1)
10	Hash Cond: (r.inventory_id = i.inventory_id)
11	-> Hash Join (cost=510.99..914.86 rows=16049 width=10) (actual time=2.755..8.229 rows=16049 loops=1)
12	Hash Cond: (p.rental_id = r.rental_id)
13	-> Append (cost=0.00..361.74 rows=16049 width=10) (actual time=0.006..2.206 rows=16049 loops=1)
14	-> Seq Scan on payment_p2022_01 p_1 (cost=0.00..13.23 rows=723 width=10) (actual time=0.006..0.068 rows=723 loops=1)
15	-> Seq Scan on payment_p2022_02 p_2 (cost=0.00..42.01 rows=2401 width=10) (actual time=0.006..0.206 rows=2401 loops=1)
16	-> Seq Scan on payment_p2022_03 p_3 (cost=0.00..47.13 rows=2713 width=10) (actual time=0.012..0.233 rows=2713 loops=1)
17	-> Seq Scan on payment_p2022_04 p_4 (cost=0.00..44.47 rows=2547 width=10) (actual time=0.008..0.221 rows=2547 loops=1)
18	-> Seq Scan on payment_p2022_05 p_5 (cost=0.00..46.77 rows=2677 width=10) (actual time=0.011..0.230 rows=2677 loops=1)
19	-> Seq Scan on payment_p2022_06 p_6 (cost=0.00..46.54 rows=2654 width=10) (actual time=0.008..0.231 rows=2654 loops=1)
20	-> Seq Scan on payment_p2022_07 p_7 (cost=0.00..41.34 rows=2334 width=10) (actual time=0.017..0.210 rows=2334 loops=1)
21	-> Hash (cost=310.44..310.44 rows=16044 width=8) (actual time=2.733..2.734 rows=16044 loops=1)
22	Buckets: 16384 Batches: 1 Memory Usage: 755kB
23	-> Seq Scan on rental r (cost=0.00..310.44 rows=16044 width=8) (actual time=0.003..1.345 rows=16044 loops=1)
24	-> Hash (cost=75.81..75.81 rows=4581 width=8) (actual time=0.801..0.801 rows=4581 loops=1)
25	Buckets: 8192 Batches: 1 Memory Usage: 243kB
26	-> Seq Scan on inventory i (cost=0.00..75.81 rows=4581 width=8) (actual time=0.002..0.377 rows=4581 loops=1)
27	-> Hash (cost=20.67..20.67 rows=1000 width=36) (actual time=0.334..0.336 rows=1000 loops=1)
28	Buckets: 1024 Batches: 1 Memory Usage: 52kB
29	-> Hash Join (cost=1.36..20.67 rows=1000 width=36) (actual time=0.054..0.237 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=0.013..0.063 rows=1000 loops=1)
32	-> Hash (cost=1.16..1.16 rows=16 width=36) (actual time=0.012..0.012 rows=16 loops=1)
33	Buckets: 1024 Batches: 1 Memory Usage: 9kB
34	-> Seq Scan on category c (cost=0.00..1.16 rows=16 width=36) (actual time=0.006..0.007 rows=16 loops=1)
35	Planning Time: 0.851 ms
36	Execution Time: 17.454 ms

- Planning time:0.851 ms
- Execution time:17.454 ms
- Part of the plan that has:
 - Highest cost: Seq scan
 - Slowest runtime: Hash join
 - Largest num of rows: Hash join

After:





	QUERY PLAN	
	text	
1	Sort (cost=1424.72..1424.76 rows=16 width=64) (actual time=20.015..20.024 rows=16 loops=1)	
2	Sort Key: (sum(p.amount)) DESC	
3	Sort Method: quicksort Memory: 25kB	
4	-> HashAggregate (cost=1424.20..1424.40 rows=16 width=64) (actual time=19.997..20.008 rows=16 loops=1)	
5	Group Key: c.name	
6	Batches: 1 Memory Usage: 24kB	
7	-> Hash Join (cost=677.24..1343.96 rows=16049 width=38) (actual time=3.749..16.783 rows=16049 loops=1)	
8	Hash Cond: (i.film_id = fc.film_id)	
9	-> Hash Join (cost=644.06..1090.11 rows=16049 width=10) (actual time=3.437..13.867 rows=16049 loops=1)	
10	Hash Cond: (r.inventory_id = i.inventory_id)	
11	-> Hash Join (cost=510.99..914.86 rows=16049 width=10) (actual time=2.681..9.922 rows=16049 loops=1)	
12	Hash Cond: (p.rental_id = r.rental_id)	
13	-> Append (cost=0.00..361.74 rows=16049 width=10) (actual time=0.008..2.596 rows=16049 loops=1)	
14	-> Seq Scan on payment_p2022_01 p_1 (cost=0.00..13.23 rows=723 width=10) (actual time=0.007..0.075 rows=723 loops=1)	
15	-> Seq Scan on payment_p2022_02 p_2 (cost=0.00..42.01 rows=2401 width=10) (actual time=0.018..0.217 rows=2401 loops=1)	
16	-> Seq Scan on payment_p2022_03 p_3 (cost=0.00..47.13 rows=2713 width=10) (actual time=0.009..0.282 rows=2713 loops=1)	
17	-> Seq Scan on payment_p2022_04 p_4 (cost=0.00..44.47 rows=2547 width=10) (actual time=0.020..0.313 rows=2547 loops=1)	
18	-> Seq Scan on payment_p2022_05 p_5 (cost=0.00..46.77 rows=2677 width=10) (actual time=0.019..0.369 rows=2677 loops=1)	
19	-> Seq Scan on payment_p2022_06 p_6 (cost=0.00..46.54 rows=2654 width=10) (actual time=0.013..0.256 rows=2654 loops=1)	
20	-> Seq Scan on payment_p2022_07 p_7 (cost=0.00..41.34 rows=2334 width=10) (actual time=0.014..0.220 rows=2334 loops=1)	
21	-> Hash (cost=310.44..310.44 rows=16044 width=8) (actual time=2.651..2.651 rows=16044 loops=1)	
22	Buckets: 16384 Batches: 1 Memory Usage: 755kB	
23	-> Seq Scan on rental r (cost=0.00..310.44 rows=16044 width=8) (actual time=0.003..1.179 rows=16044 loops=1)	
24	-> Hash (cost=75.81..75.81 rows=4581 width=8) (actual time=0.723..0.723 rows=4581 loops=1)	
25	Buckets: 8192 Batches: 1 Memory Usage: 243kB	
26	-> Seq Scan on inventory i (cost=0.00..75.81 rows=4581 width=8) (actual time=0.006..0.310 rows=4581 loops=1)	
27	-> Hash (cost=20.67..20.67 rows=1000 width=36) (actual time=0.307..0.309 rows=1000 loops=1)	
28	Buckets: 1024 Batches: 1 Memory Usage: 52kB	
29	-> Hash Join (cost=1.36..20.67 rows=1000 width=36) (actual time=0.026..0.206 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=0.011..0.062 rows=1000 loops=1)	
32	-> Hash (cost=1.16..1.16 rows=16 width=36) (actual time=0.011..0.011 rows=16 loops=1)	
33	Buckets: 1024 Batches: 1 Memory Usage: 9kB	
34	-> Seq Scan on category c (cost=0.00..1.16 rows=16 width=36) (actual time=0.005..0.006 rows=16 loops=1)	
35	Planning Time: 0.517 ms	
36	Execution Time: 20.126 ms	

Index Used:

```
CREATE INDEX INDEX23 ON film_category USING BTREE(film_id);
CREATE INDEX INDEX24 ON inventory USING BTREE(film_id);
CREATE INDEX INDEX25 ON rental USING BTREE(inventory_id);
```

Notes: These indexes didn't show up in the planner however the planner uses them as shown below that these indexes are used, indexes 23, 24 and 25 are BTREE indexes on attributes film_id in tables film_category and inventory, and inventory_id in table rental which let us get any row in these tables using these attributes in $O(\log(n))$

- Planning time: 0.517 ms
- Execution time: 20.126 ms
- Part of the plan that has:
 - Highest cost: Seq scan
 - Slowest runtime: Hash join
 - Largest num of rows: Hash join

indexrelname		idx_scan		idx_tup_read		idx_tup_fetch	
name		bigint		bigint		bigint	
pgbench_branches_pkey		0		0		0	
pgbench_tellers_pkey		0		0		0	
pgbench_accounts_pkey		0		0		0	
actor_pkey		63		1456		1456	
address_pkey		19		621		621	
category_pkey		31		496		496	
city_pkey		13		612		612	
country_pkey		6		6		6	
customer_pkey		0		0		0	
film_actor_pkey		14		14		0	
film_category_pkey		1029		6024		2024	
film_pkey		1853		9845		9845	
inventory_pkey		84		84		0	
language_pkey		1000		1000		1000	
rental_pkey		15		240660		240660	
staff_pkey		0		0		0	
store_pkey		28		28		28	
index23		13		1012		1012	
index24		13		4593		4581	
index25		12		12		0	

Query19:

QUERY PLAN	
	text
1	Group (cost=0.75..112819.24 rows=1000 width=157) (actual time=274.253..557.446 rows=1000 loops=1)
2	Group Key: f.film_id, (btrim((l.name)::text))
3	-> Incremental Sort (cost=0.75..358.74 rows=1000 width=76) (actual time=262.465..265.740 rows=1000 loops=1)
4	Sort Key: f.film_id, (btrim((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=262.396..265.007 rows=1000 loops=1)
8	-> Index Scan using film_pkey on film f (cost=0.28..93.37 rows=1000 width=48) (actual time=0.020..1.000 rows=1000 loops=1)
9	-> Index Scan using language_pkey on language l (cost=0.15..0.22 rows=1 width=88) (actual time=0.002..0.002 rows=1 loops=1000)
10	Index Cond: (language_id = f.language_id)
11	SubPlan 2
12	-> Result (cost=9.52..9.53 rows=1 width=32) (actual time=0.017..0.018 rows=1 loops=1000)
13	InitPlan 1 (returns \$1)
14	-> Hash Join (cost=8.30..9.52 rows=1 width=32) (actual time=0.012..0.013 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..0.002 rows=16 loops=1000)
17	-> Hash (cost=8.29..8.29 rows=1 width=4) (actual time=0.008..0.008 rows=1 loops=1000)
18	Buckets: 1024 Batches: 1 Memory Usage: 9kB
19	-> Index Only Scan using film_category_pkey on film_category fc (cost=0.28..8.29 rows=1 width=4) (actual time=0.006..0.007 row...)
20	Index Cond: (film_id = f.film_id)
21	Heap Fetches: 1000
22	SubPlan 4
23	-> Result (cost=102.90..102.91 rows=1 width=32) (actual time=0.269..0.269 rows=1 loops=1000)
24	InitPlan 3 (returns \$3)
25	-> Hash Join (cost=98.34..102.90 rows=5 width=32) (actual time=0.239..0.264 rows=5 loops=1000)
26	Hash Cond: (a.actor_id = fa.actor_id)
27	-> Seq Scan on actor a (cost=0.00..4.00 rows=200 width=17) (actual time=0.002..0.015 rows=200 loops=997)
28	-> Hash (cost=98.28..98.28 rows=5 width=4) (actual time=0.227..0.227 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=4) (actual time=0.042..0.225 rows=5 loops=1000)
31	Filter: (film_id = f.film_id)
32	Rows Removed by Filter: 5457
33	Planning Time: 6.467 ms
34	JIT:
35	Functions: 38
36	Options: Inlining false, Optimization false, Expressions true, Deforming true
37	Timing: Generation 3.282 ms, Inlining 0.000 ms, Optimization 22.961 ms, Emission 227.539 ms, Total 253.783 ms
38	Execution Time: 674.107 ms

- Planning time:6.467 ms
- Execution time:674.107 ms
- Part of the plan that has:
 - Highest cost: Group
 - Slowest runtime: Result
 - Largest num of rows: Seq scan

After:

	QUERY PLAN	
	text	
1	Group (cost=0.75..112819.24 rows=1000 width=157) (actual time=20.417..293.015 rows=1000 loops=1)	
2	Group Key: f.film_id, (btrim((l.name)::text))	
3	-> Incremental Sort (cost=0.75..358.74 rows=1000 width=76) (actual time=20.143..22.366 rows=1000 loops=1)	
4	Sort Key: f.film_id, (btrim((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=20.086..21.777 rows=1000 loops=1)	
8	-> Index Scan using film_pkey on film f (cost=0.28..93.37 rows=1000 width=48) (actual time=0.017..0.362 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=9.52..9.53 rows=1 width=32) (actual time=0.012..0.012 rows=1 loops=1000)	
13	InitPlan 1 (returns \$1)	
14	-> Hash Join (cost=8.30..9.52 rows=1 width=32) (actual time=0.007..0.008 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..0.001 rows=16 loops=1000)	
17	-> Hash (cost=8.29..8.29 rows=1 width=4) (actual time=0.003..0.003 rows=1 loops=1000)	
18	Buckets: 1024 Batches: 1 Memory Usage: 9kB	
19	-> Index Only Scan using film_category_pkey on film_category fc (cost=0.28..8.29 rows=1 width=4) (actual time=0.002..0.002 rows=1 loops=1000)	
20	Index Cond: (film_id = f.film_id)	
21	Heap Fetches: 1000	
22	SubPlan 4	
23	-> Result (cost=102.90..102.91 rows=1 width=32) (actual time=0.255..0.255 rows=1 loops=1000)	
24	InitPlan 3 (returns \$3)	
25	-> Hash Join (cost=98.34..102.90 rows=5 width=32) (actual time=0.228..0.252 rows=5 loops=1000)	
26	Hash Cond: (a.actor_id = fa.actor_id)	
27	-> Seq Scan on actor a (cost=0.00..4.00 rows=200 width=17) (actual time=0.002..0.014 rows=200 loops=997)	
28	-> Hash (cost=98.28..98.28 rows=5 width=4) (actual time=0.216..0.216 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=4) (actual time=0.039..0.214 rows=5 loops=1000)	
31	Filter: (film_id = f.film_id)	
32	Rows Removed by Filter: 5457	
33	Planning Time: 0.332 ms	
34	JIT:	
35	Functions: 38	
36	Options: Inlining false, Optimization false, Expressions true, Deforming true	
37	Timing: Generation 3.965 ms, Inlining 0.000 ms, Optimization 1.937 ms, Emission 17.749 ms, Total 23.651 ms	
38	Execution Time: 297.387 ms	

Index Used:

CREATE INDEX INDEX26 ON film_actor USING BTREE(actor_id);

Notes: This index didn't show up in the planner however the planner uses it as shown below, index26 is BTREE index on attributes actor_id in table film_ which let us get any row in tables film actor in $O(\log(n))$

- Planning time: 0.332 ms
- Execution time: 297.387 ms
- Part of the plan that has:
 - Highest cost: Group
 - Slowest runtime: Result
 - Largest num of rows: Seq scan

relname name	indexrelname name	idx_scan bigint	idx_tup_read bigint	idx_tup_fetch bigint
pgbench_bran...	pgbench_branches_pkey	0	0	0
pgbench_tellers	pgbench_tellers_pkey	0	0	0
pgbench_accou...	pgbench_accounts_pkey	0	0	0
actor	actor_pkey	4073	656718	656718
address	address_pkey	19	621	621
category	category_pkey	4031	4496	4496
city	city_pkey	13	612	612
country	country_pkey	6	6	6
customer	customer_pkey	0	0	0
film_actor	film_actor_pkey	4014	21862	0
film_category	film_category_pkey	12029	17024	13024
film	film_pkey	1864	20845	20845
inventory	inventory_pkey	88	88	0
language	language_pkey	12000	12000	12000
rental	rental_pkey	15	240660	240660
staff	staff_pkey	0	0	0
store	store_pkey	28	28	28
film_actor	index26	22	22	0