Database project

Indexes used for optmizing the database schema in general:

I chose these indexes based on there num of records,how many the 17 queries uses them,which tables are vital and play as a bridge between diffirent tables,so thee indexes are optimizing these specif tables in genereal film_category,film_actor,film,rental and inventory tables as these are the tables that have these charactarestics.

```
CREATE INDEX Index1 on film_category USING BTREE (film_id);
```

This index used to optimize 3 different queries :1,9,11 by using BTREE index on table film_category using film_id which let us get any record in film_category table in LOG(N) using film id which let us make the process of joining film_category table and film table much faster

```
CREATE INDEX Index2 on film_actor USING BTREE (actor_id);
```

This index used to optimize 2 different queries :2,3 by using BTREE index on table film_actor using actor_id which let us get any record in film_actor table in LOG(N) using actor id which let us make the process of joining film_actor table and actor table much faster

```
CREATE INDEX Index3 on rental USING BTREE(customer_id );
```

This index used to optimize 3 different queries :6,7,8 by using BTREE index on table rental using customer_id which let us get any record in rental table in LOG(N) using customer id which let us make the process of joining customer table and rental table much faster

```
CREATE INDEX Index4  on rental USING HASH (inventory_id);
```

This index used to optimize 2 different queries :9,11 by using HASH index on table rental using film_id which let us get any record in rental table in O(N) using inventory id which let us make the process of joining rental table and inventory table much faster

```
CREATE INDEX Index5 on inventory USING BTREE (film_id);
```

This index used to optimize 2 different queries :9,13 by using BTREE index on table inventory using film_id which let us get any record in inventory table in LOG(N) using film id which let us make the process of joining film table and inventory table much faster

Query1:

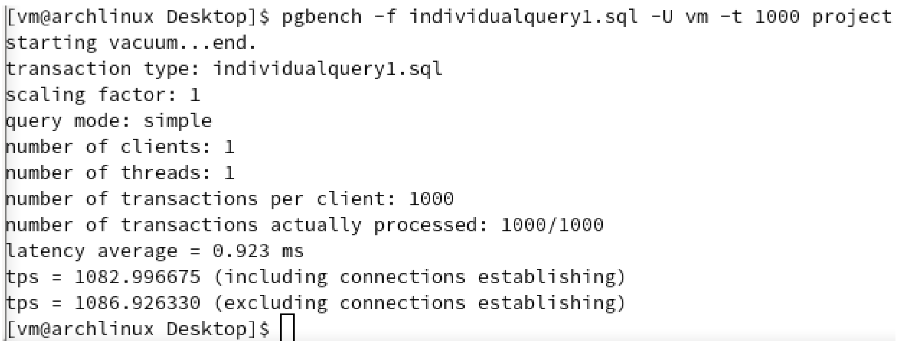
Indexes used for optimizing this query:

This query used Index1 that also used in optimizing the database in general

CREATE INDEX film_film_id_tree on film USING BTREE(film_id) where film.title like 'A%'

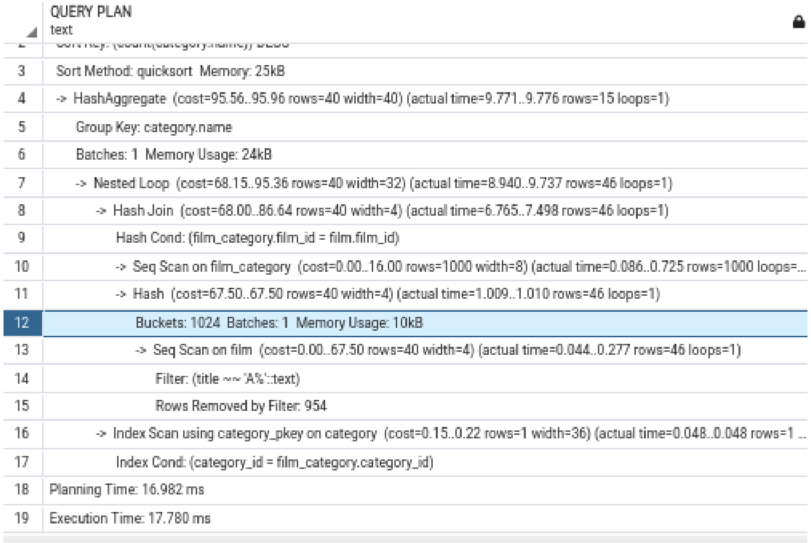
This is a BTREE and partial index that is only done in the values where film.title starts with A,since most values don't start with A then this is very efficient value for search for titles starts with A,moreover BTREE index on film_id is LOG(N) and this will make the process of searching for a specific film_id faster

Latency and tps before index:



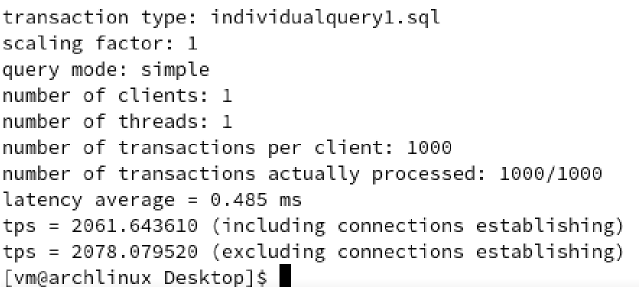
```
[vm@archlinux Desktop]$ pgbench -f individualquery1.sql -U vm -t 1000 project
starting vacuum...end.
transaction type: individualquery1.sql
scaling factor: 1
query mode: simple
number of clients: 1
number of threads: 1
number of transactions per client: 1000
number of transactions actually processed: 1000/1000
latency average = 0.923 ms
tps = 1082.996675 (including connections establishing)
tps = 1086.926330 (excluding connections establishing)
[vm@archlinux Desktop]$
```

Query plantime and execution time before index:



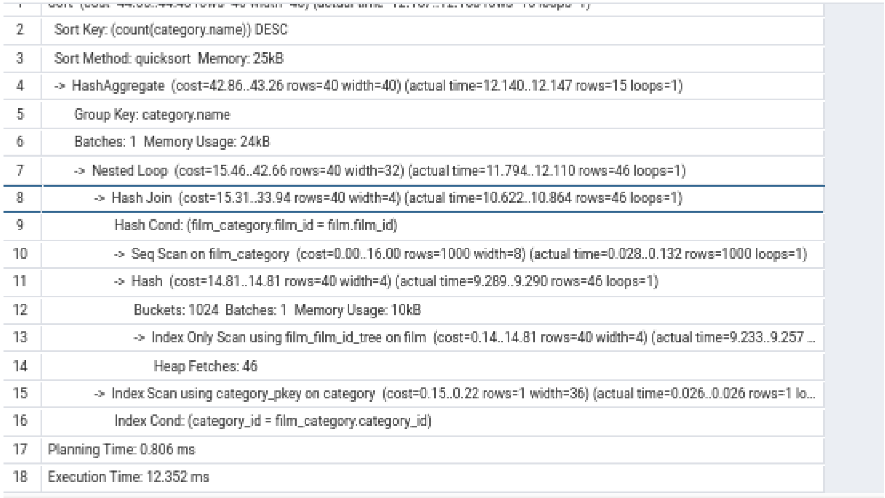
```
    QUERY PLAN
    text
 2  Sort Key: (count(category.name)) DESC
 3  Sort Method: quicksort  Memory: 25kB
 4  -> HashAggregate  (cost=95.56..95.96 rows=40 width=40) (actual time=9.771..9.776 rows=15 loops=1)
 5       Group Key: category.name
 6       Batches: 1  Memory Usage: 24kB
 7       -> Nested Loop  (cost=68.15..95.36 rows=40 width=32) (actual time=8.940..9.737 rows=46 loops=1)
 8            -> Hash Join  (cost=68.00..86.64 rows=40 width=4) (actual time=6.765..7.498 rows=46 loops=1)
 9                 Hash Cond: (film_category.film_id = film.film_id)
10                 -> Seq Scan on film_category  (cost=0.00..16.00 rows=1000 width=8) (actual time=0.086..0.725 rows=1000 loops=...
11                 -> Hash  (cost=67.50..67.50 rows=40 width=4) (actual time=1.009..1.010 rows=46 loops=1)
12                      Buckets: 1024  Batches: 1  Memory Usage: 10kB
13                      -> Seq Scan on film  (cost=0.00..67.50 rows=40 width=4) (actual time=0.044..0.277 rows=46 loops=1)
14                           Filter: (title ~~ 'A%'::text)
15                           Rows Removed by Filter: 954
16            -> Index Scan using category_pkey on category  (cost=0.15..0.22 rows=1 width=36) (actual time=0.048..0.048 rows=1 ...
17                 Index Cond: (category_id = film_category.category_id)
18  Planning Time: 16.982 ms
19  Execution Time: 17.780 ms
```

Latency and tps After index:



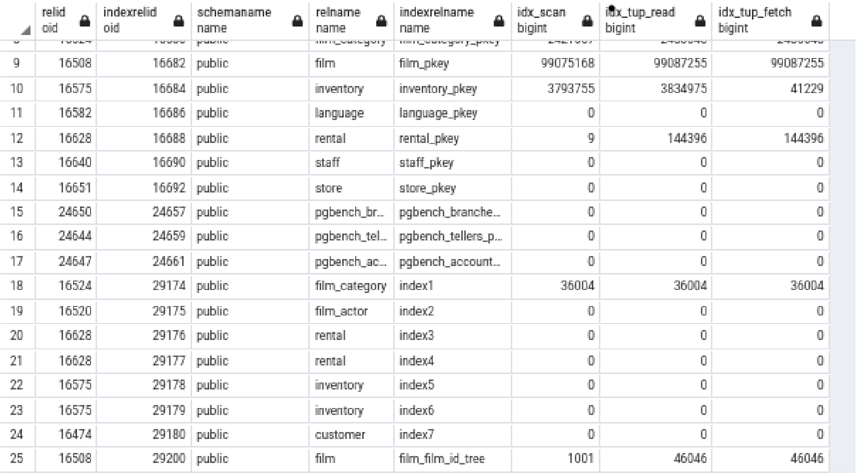
```
transaction type: individualquery1.sql
scaling factor: 1
query mode: simple
number of clients: 1
number of threads: 1
number of transactions per client: 1000
number of transactions actually processed: 1000/1000
latency average = 0.485 ms
tps = 2061.643610 (including connections establishing)
tps = 2078.079520 (excluding connections establishing)
[vm@archlinux Desktop]$
```

Query plantime and execution time After indexes:



```
Sort [illegible]
 Sort Key: (count(category.name)) DESC
 Sort Method: quicksort  Memory: 25kB
 -> HashAggregate  (cost=42.86..43.26 rows=40 width=40) (actual time=12.140..12.147 rows=15 loops=1)
    Group Key: category.name
    Batches: 1  Memory Usage: 24kB
    -> Nested Loop  (cost=15.46..42.66 rows=40 width=32) (actual time=11.794..12.110 rows=46 loops=1)
       -> Hash Join  (cost=15.31..33.94 rows=40 width=4) (actual time=10.622..10.864 rows=46 loops=1)
          Hash Cond: (film_category.film_id = film.film_id)
          -> Seq Scan on film_category  (cost=0.00..16.00 rows=1000 width=8) (actual time=0.028..0.132 rows=1000 loops=1)
          -> Hash  (cost=14.81..14.81 rows=40 width=4) (actual time=9.289..9.290 rows=46 loops=1)
             Buckets: 1024  Batches: 1  Memory Usage: 10kB
             -> Index Only Scan using film_film_id_tree on film  (cost=0.14..14.81 rows=40 width=4) (actual time=9.233..9.257 ...
                Heap Fetches: 46
       -> Index Scan using category_pkey on category  (cost=0.15..0.22 rows=1 width=36) (actual time=0.026..0.026 rows=1 lo...
          Index Cond: (category_id = film_category.category_id)
Planning Time: 0.806 ms
Execution Time: 12.352 ms
```

Index film_film_id_tree usage and the indexes used for the whole database:



	relid oid	indexrelid oid	schemaname name	relname name	indexrelname name	idx_scan bigint	idx_tup_read bigint	idx_tup_fetch bigint
[illegible]	[illegible]	[illegible]	public	film_category	film_category_pkey	[illegible]	[illegible]	[illegible]
9	16508	16682	public	film	film_pkey	99075168	99087255	99087255
10	16575	16684	public	inventory	inventory_pkey	3793755	3834975	41229
11	16582	16686	public	language	language_pkey	0	0	0
12	16628	16688	public	rental	rental_pkey	9	144396	144396
13	16640	16690	public	staff	staff_pkey	0	0	0
14	16651	16692	public	store	store_pkey	0	0	0
15	24650	24657	public	pgbench_br...	pgbench_branche...	0	0	0
16	24644	24659	public	pgbench_tel...	pgbench_tellers_p...	0	0	0
17	24647	24661	public	pgbench_ac...	pgbench_account...	0	0	0
18	16524	29174	public	film_category	index1	36004	36004	36004
19	16520	29175	public	film_actor	index2	0	0	0
20	16628	29176	public	rental	index3	0	0	0
21	16628	29177	public	rental	index4	0	0	0
22	16575	29178	public	inventory	index5	0	0	0
23	16575	29179	public	inventory	index6	0	0	0
24	16474	29180	public	customer	index7	0	0	0
25	16508	29200	public	film	film_film_id_tree	1001	46046	46046

QUERY2:

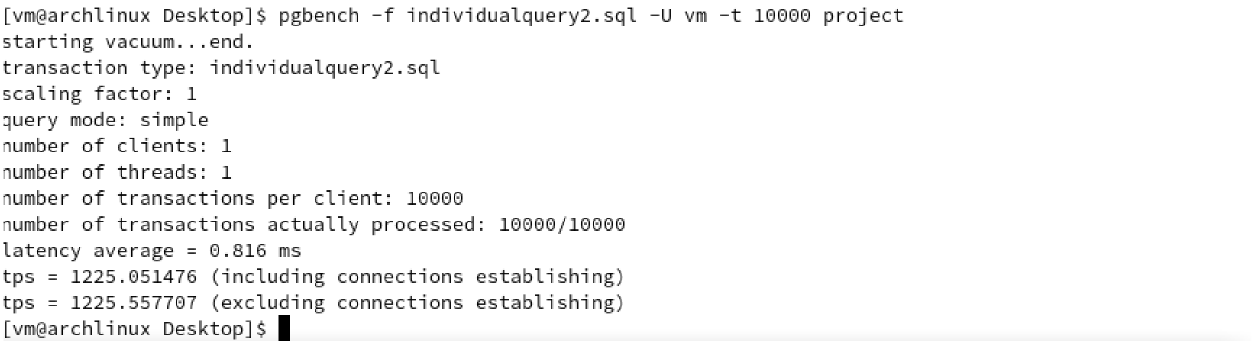
Indexes used for optimizing this query:

This query used Index2 that also used in optimizing the database in general

CREATE INDEX actor_last_name_tree on actor USING BTREE(actor_id) WHERE actor.last_name like 'D%';

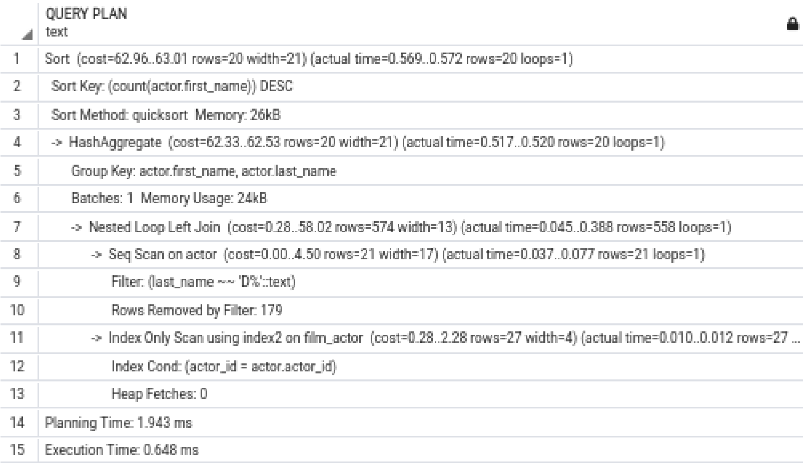
This index didnt work,I made this index so I can index all last_name that starts with D and then count these names that starts with start D but the planner saw that it is cheaper that not to use it

Latency and tps before indexes:



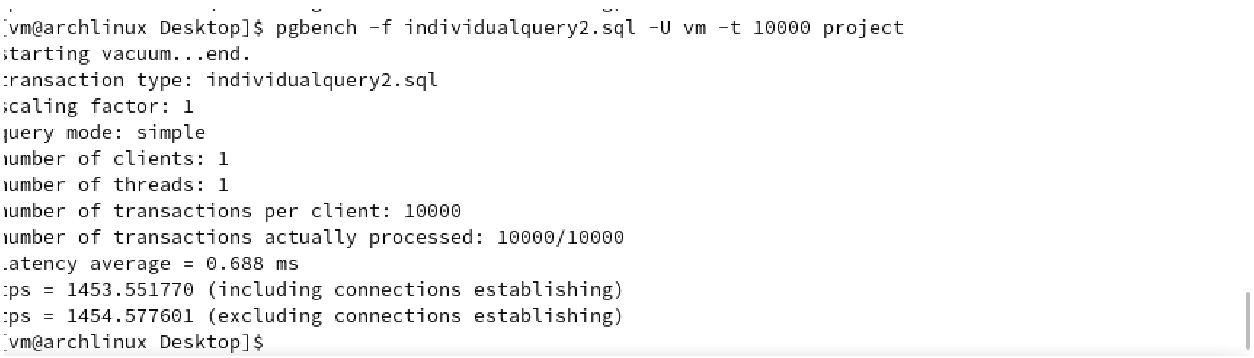
```
[vm@archlinux Desktop]$ pgbench -f individualquery2.sql -U vm -t 10000 project
starting vacuum...end.
transaction type: individualquery2.sql
scaling factor: 1
query mode: simple
number of clients: 1
number of threads: 1
number of transactions per client: 10000
number of transactions actually processed: 10000/10000
latency average = 0.816 ms
tps = 1225.051476 (including connections establishing)
tps = 1225.557707 (excluding connections establishing)
[vm@archlinux Desktop]$
```

Query plantime and execution time before indexes:



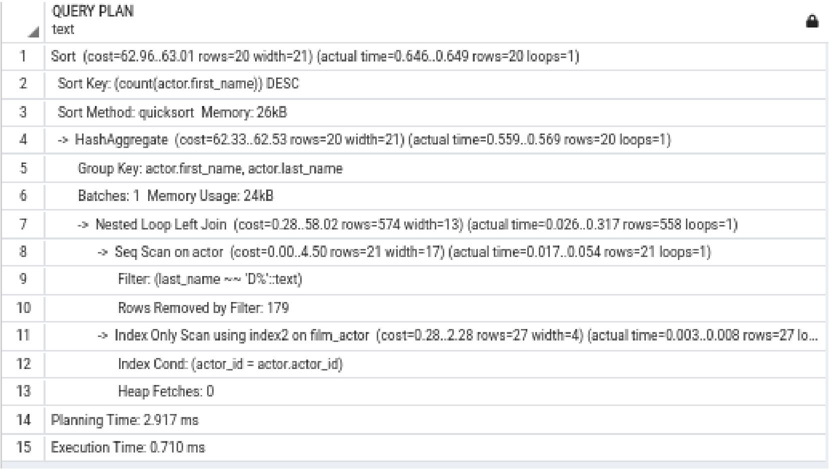
	QUERY PLAN text
1	Sort (cost=62.96..63.01 rows=20 width=21) (actual time=0.569..0.572 rows=20 loops=1)
2	Sort Key: (count(actor.first_name)) DESC
3	Sort Method: quicksort Memory: 26kB
4	-> HashAggregate (cost=62.33..62.53 rows=20 width=21) (actual time=0.517..0.520 rows=20 loops=1)
5	Group Key: actor.first_name, actor.last_name
6	Batches: 1 Memory Usage: 24kB
7	-> Nested Loop Left Join (cost=0.28..58.02 rows=574 width=13) (actual time=0.045..0.388 rows=558 loops=1)
8	-> Seq Scan on actor (cost=0.00..4.50 rows=21 width=17) (actual time=0.037..0.077 rows=21 loops=1)
9	Filter: (last_name ~~ 'D%'::text)
10	Rows Removed by Filter: 179
11	-> Index Only Scan using index2 on film_actor (cost=0.28..2.28 rows=27 width=4) (actual time=0.010..0.012 rows=27 ...
12	Index Cond: (actor_id = actor.actor_id)
13	Heap Fetches: 0
14	Planning Time: 1.943 ms
15	Execution Time: 0.648 ms

Latency and tps After indexes:



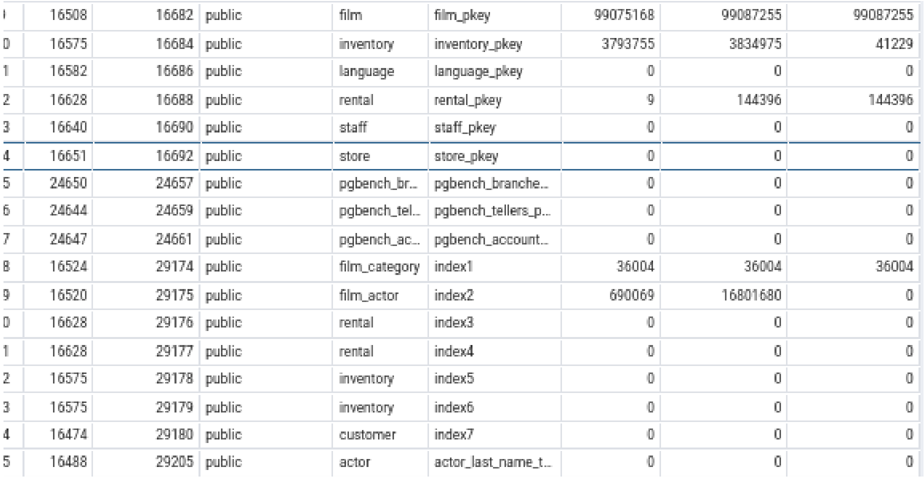
```
[vm@archlinux Desktop]$ pgbench -f individualquery2.sql -U vm -t 10000 project
starting vacuum...end.
:ransaction type: individualquery2.sql
;caling factor: 1
query mode: simple
number of clients: 1
number of threads: 1
number of transactions per client: 10000
number of transactions actually processed: 10000/10000
.atency average = 0.688 ms
:ps = 1453.551770 (including connections establishing)
:ps = 1454.577601 (excluding connections establishing)
[vm@archlinux Desktop]$
```

Query plantime and execution time After indexes:



	QUERY PLAN text
1	Sort (cost=62.96..63.01 rows=20 width=21) (actual time=0.646..0.649 rows=20 loops=1)
2	Sort Key: (count(actor.first_name)) DESC
3	Sort Method: quicksort Memory: 26kB
4	-> HashAggregate (cost=62.33..62.53 rows=20 width=21) (actual time=0.559..0.569 rows=20 loops=1)
5	Group Key: actor.first_name, actor.last_name
6	Batches: 1 Memory Usage: 24kB
7	-> Nested Loop Left Join (cost=0.28..58.02 rows=574 width=13) (actual time=0.026..0.317 rows=558 loops=1)
8	-> Seq Scan on actor (cost=0.00..4.50 rows=21 width=17) (actual time=0.017..0.054 rows=21 loops=1)
9	Filter: (last_name ~~ 'D%'::text)
10	Rows Removed by Filter: 179
11	-> Index Only Scan using index2 on film_actor (cost=0.28..2.28 rows=27 width=4) (actual time=0.003..0.008 rows=27 lo...
12	Index Cond: (actor_id = actor.actor_id)
13	Heap Fetches: 0
14	Planning Time: 2.917 ms
15	Execution Time: 0.710 ms

Index actor_last_name_tree Usage and the indexes used for the whole database:



9	16508	16682	public	film	film_pkey	99075168	99087255	99087255
0	16575	16684	public	inventory	inventory_pkey	3793755	3834975	41229
1	16582	16686	public	language	language_pkey	0	0	0
2	16628	16688	public	rental	rental_pkey	9	144396	144396
3	16640	16690	public	staff	staff_pkey	0	0	0
4	16651	16692	public	store	store_pkey	0	0	0
5	24650	24657	public	pgbench_br...	pgbench_branche...	0	0	0
6	24644	24659	public	pgbench_tel...	pgbench_tellers_p...	0	0	0
7	24647	24661	public	pgbench_ac...	pgbench_account...	0	0	0
8	16524	29174	public	film_category	index1	36004	36004	36004
9	16520	29175	public	film_actor	index2	690069	16801680	0
0	16628	29176	public	rental	index3	0	0	0
1	16628	29177	public	rental	index4	0	0	0
2	16575	29178	public	inventory	index5	0	0	0
3	16575	29179	public	inventory	index6	0	0	0
4	16474	29180	public	customer	index7	0	0	0
5	16488	29205	public	actor	actor_last_name_t...	0	0	0

QUERY3:

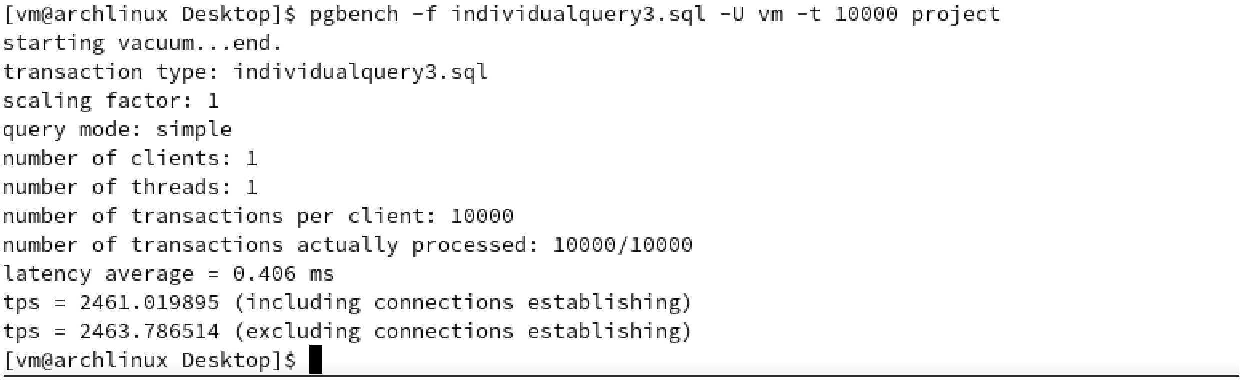
Indexes used for optimizing this query:

This query used Index2 that also used in optimizing the database in general

CREATE INDEX actor_last_name_tree on actor USING BTREE(actor_id) where actor.last_name like '%a%';

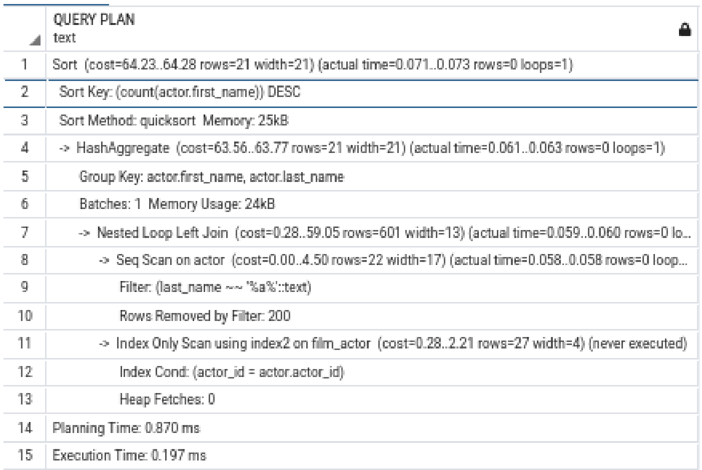
This index didn’t work as all names in last_names in actor are capital not small,so there is no use of using the index as there is noting to index

Latency and tps before indexes:



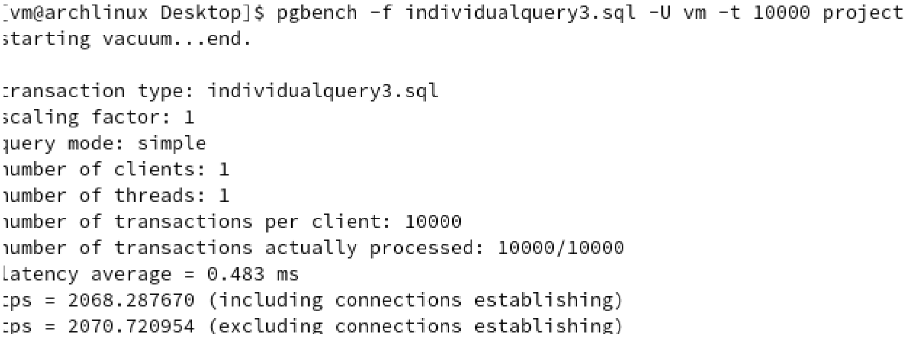
```
[vm@archlinux Desktop]$ pgbench -f individualquery3.sql -U vm -t 10000 project
starting vacuum...end.
transaction type: individualquery3.sql
scaling factor: 1
query mode: simple
number of clients: 1
number of threads: 1
number of transactions per client: 10000
number of transactions actually processed: 10000/10000
latency average = 0.406 ms
tps = 2461.019895 (including connections establishing)
tps = 2463.786514 (excluding connections establishing)
[vm@archlinux Desktop]$ 
```

Query plantime and execution time before indexes:



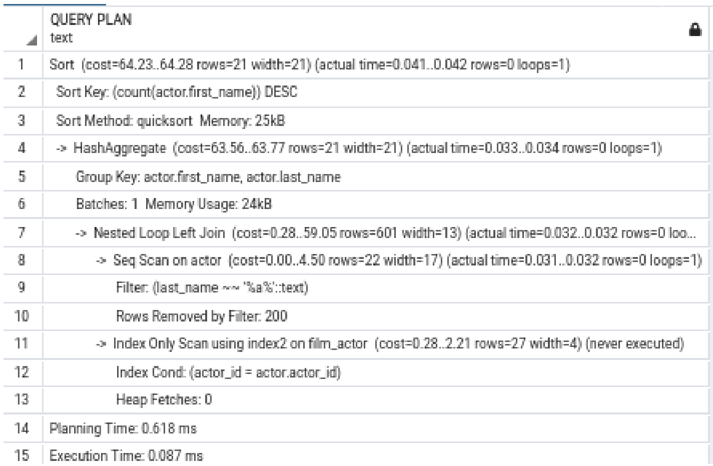
	QUERY PLAN text
1	Sort (cost=64.23..64.28 rows=21 width=21) (actual time=0.071..0.073 rows=0 loops=1)
2	Sort Key: (count(actor.first_name)) DESC
3	Sort Method: quicksort Memory: 25kB
4	-> HashAggregate (cost=63.56..63.77 rows=21 width=21) (actual time=0.061..0.063 rows=0 loops=1)
5	Group Key: actor.first_name, actor.last_name
6	Batches: 1 Memory Usage: 24kB
7	-> Nested Loop Left Join (cost=0.28..59.05 rows=601 width=13) (actual time=0.059..0.060 rows=0 lo...
8	-> Seq Scan on actor (cost=0.00..4.50 rows=22 width=17) (actual time=0.058..0.058 rows=0 loop...
9	Filter: (last_name ~~ '%a%'::text)
10	Rows Removed by Filter: 200
11	-> Index Only Scan using index2 on film_actor (cost=0.28..2.21 rows=27 width=4) (never executed)
12	Index Cond: (actor_id = actor.actor_id)
13	Heap Fetches: 0
14	Planning Time: 0.870 ms
15	Execution Time: 0.197 ms

Latency and tps AFTER indexes:



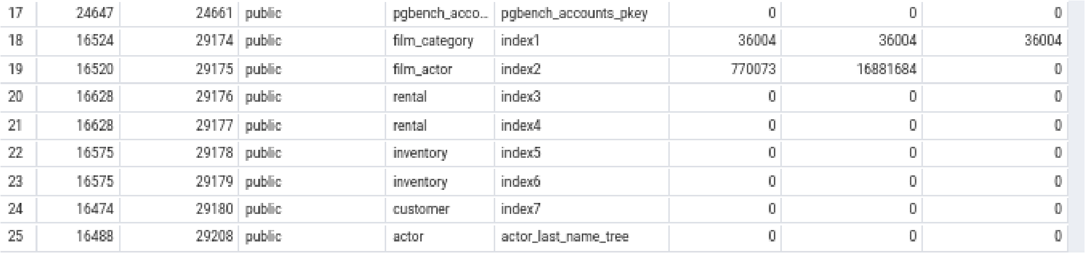
```
[vm@archlinux Desktop]$ pgbench -f individualquery3.sql -U vm -t 10000 project
starting vacuum...end.

transaction type: individualquery3.sql
scaling factor: 1
query mode: simple
number of clients: 1
number of threads: 1
number of transactions per client: 10000
number of transactions actually processed: 10000/10000
latency average = 0.483 ms
tps = 2068.287670 (including connections establishing)
tps = 2070.720954 (excluding connections establishing)
```

Query plantime and execution time after indexes:



	QUERY PLAN text
1	Sort (cost=64.23..64.28 rows=21 width=21) (actual time=0.041..0.042 rows=0 loops=1)
2	Sort Key: (count(actor.first_name)) DESC
3	Sort Method: quicksort Memory: 25kB
4	-> HashAggregate (cost=63.56..63.77 rows=21 width=21) (actual time=0.033..0.034 rows=0 loops=1)
5	Group Key: actor.first_name, actor.last_name
6	Batches: 1 Memory Usage: 24kB
7	-> Nested Loop Left Join (cost=0.28..59.05 rows=601 width=13) (actual time=0.032..0.032 rows=0 loo...
8	-> Seq Scan on actor (cost=0.00..4.50 rows=22 width=17) (actual time=0.031..0.032 rows=0 loops=1)
9	Filter: (last_name ~~ '%a%'::text)
10	Rows Removed by Filter: 200
11	-> Index Only Scan using index2 on film_actor (cost=0.28..2.21 rows=27 width=4) (never executed)
12	Index Cond: (actor_id = actor.actor_id)
13	Heap Fetches: 0
14	Planning Time: 0.618 ms
15	Execution Time: 0.087 ms

Index actor_last_name_tree usage and the indexes used for the whole database:



17	24647	24661	public	pgbench_acco...	pgbench_accounts_pkey	0	0	0
18	16524	29174	public	film_category	index1	36004	36004	36004
19	16520	29175	public	film_actor	index2	770073	16881684	0
20	16628	29176	public	rental	index3	0	0	0
21	16628	29177	public	rental	index4	0	0	0
22	16575	29178	public	inventory	index5	0	0	0
23	16575	29179	public	inventory	index6	0	0	0
24	16474	29180	public	customer	index7	0	0	0
25	16488	29208	public	actor	actor_last_name_tree	0	0	0

Query 4:

Indexes used for optimizing this query:

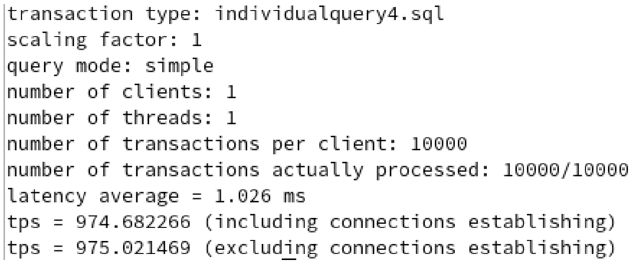
CREATE INDEX customer_first_name on customer USING BTREE(first_name ASC);

CREATE INDEX customer_active on customer USING BTREE(active);

Customer_first_name Index didnt work because sorting is a blocking operation and Index customer_active didn’t work because about 99% of the rows have active =1

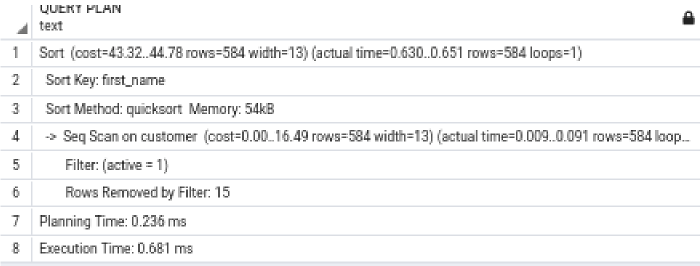
so it's cheaper to no do an index

Latency and tps before indexes:



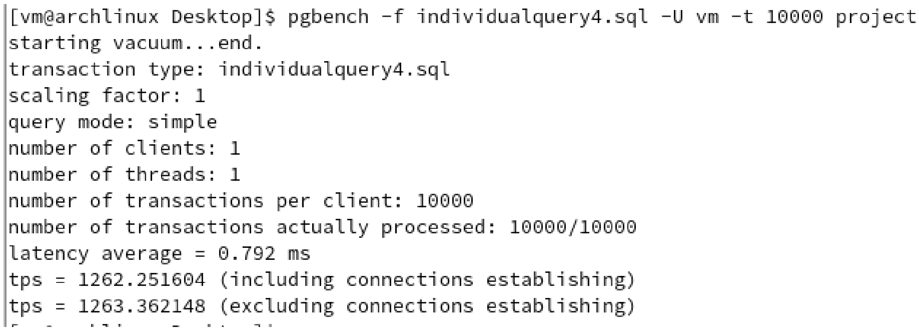
```
transaction type: individualquery4.sql
scaling factor: 1
query mode: simple
number of clients: 1
number of threads: 1
number of transactions per client: 10000
number of transactions actually processed: 10000/10000
latency average = 1.026 ms
tps = 974.682266 (including connections establishing)
tps = 975.021469 (excluding connections establishing)
```

Query plantime and execution time before indexes:



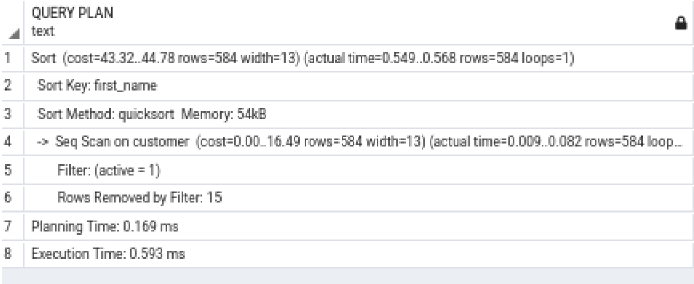
	QUERY PLAN text
1	Sort (cost=43.32..44.78 rows=584 width=13) (actual time=0.630..0.651 rows=584 loops=1)
2	Sort Key: first_name
3	Sort Method: quicksort Memory: 54kB
4	-> Seq Scan on customer (cost=0.00..16.49 rows=584 width=13) (actual time=0.009..0.091 rows=584 loop...
5	Filter: (active = 1)
6	Rows Removed by Filter: 15
7	Planning Time: 0.236 ms
8	Execution Time: 0.681 ms

Latency and tps AFTER indexes:



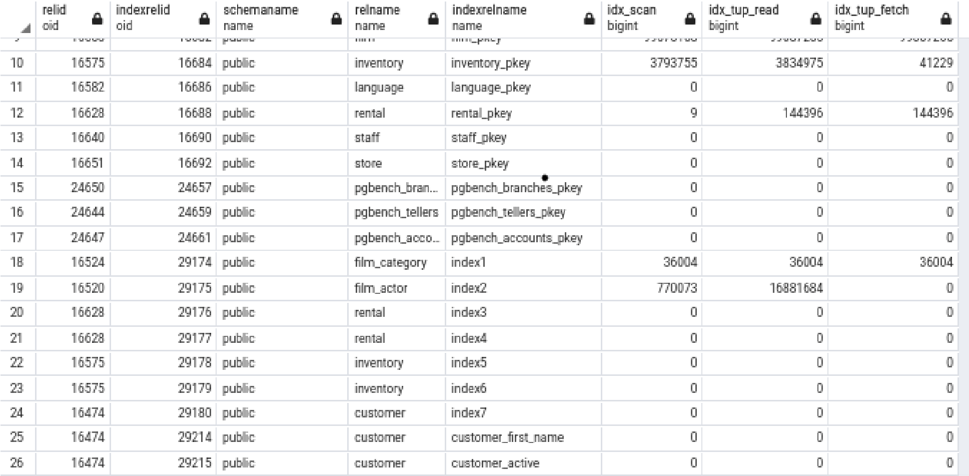
```
[vm@archlinux Desktop]$ pgbench -f individualquery4.sql -U vm -t 10000 project
starting vacuum...end.
transaction type: individualquery4.sql
scaling factor: 1
query mode: simple
number of clients: 1
number of threads: 1
number of transactions per client: 10000
number of transactions actually processed: 10000/10000
latency average = 0.792 ms
tps = 1262.251604 (including connections establishing)
tps = 1263.362148 (excluding connections establishing)
```

Query plantime and execution time after indexes:



	QUERY PLAN text
1	Sort (cost=43.32..44.78 rows=584 width=13) (actual time=0.549..0.568 rows=584 loops=1)
2	Sort Key: first_name
3	Sort Method: quicksort Memory: 54kB
4	-> Seq Scan on customer (cost=0.00..16.49 rows=584 width=13) (actual time=0.009..0.082 rows=584 loop...
5	Filter: (active = 1)
6	Rows Removed by Filter: 15
7	Planning Time: 0.169 ms
8	Execution Time: 0.593 ms

Indexes customer_first_name and customer_active usage and the indexes used for the whole database:



	relid oid	indexrelid oid	schemaname name	relname name	indexrelname name	idx_scan bigint	idx_tup_read bigint	idx_tup_fetch bigint
10	16575	16684	public	inventory	inventory_pkey	3793755	3834975	41229
11	16582	16686	public	language	language_pkey	0	0	0
12	16628	16688	public	rental	rental_pkey	9	144396	144396
13	16640	16690	public	staff	staff_pkey	0	0	0
14	16651	16692	public	store	store_pkey	0	0	0
15	24650	24657	public	pgbench_bran...	pgbench_branches_pkey	0	0	0
16	24644	24659	public	pgbench_tellers	pgbench_tellers_pkey	0	0	0
17	24647	24661	public	pgbench_acco...	pgbench_accounts_pkey	0	0	0
18	16524	29174	public	film_category	index1	36004	36004	36004
19	16520	29175	public	film_actor	index2	770073	16881684	0
20	16628	29176	public	rental	index3	0	0	0
21	16628	29177	public	rental	index4	0	0	0
22	16575	29178	public	inventory	index5	0	0	0
23	16575	29179	public	inventory	index6	0	0	0
24	16474	29180	public	customer	index7	0	0	0
25	16474	29214	public	customer	customer_first_name	0	0	0
26	16474	29215	public	customer	customer_active	0	0	0

Query 5:

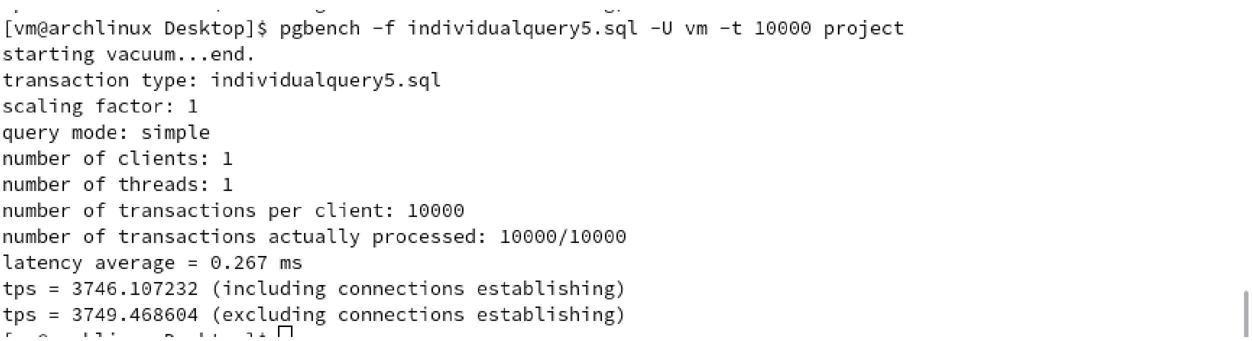
Indexes used for optimizing this query:

CREATE INDEX customer_first_name on customer USING BTREE(first_name ASC) WHERE last_name like 'A%';

CREATE INDEX customer_active on customer USING BTREE(active);

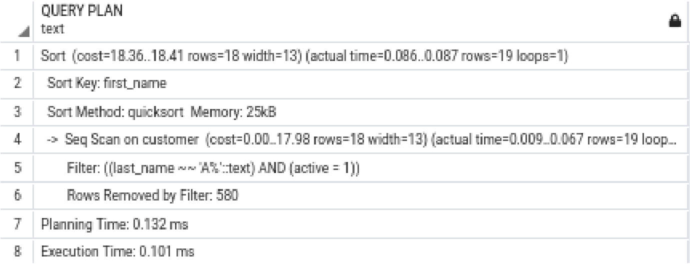
customer_active didn’t work because about 99% of the rows have active =1 so it's cheaper to no do an index

Latency and tps before indexes:



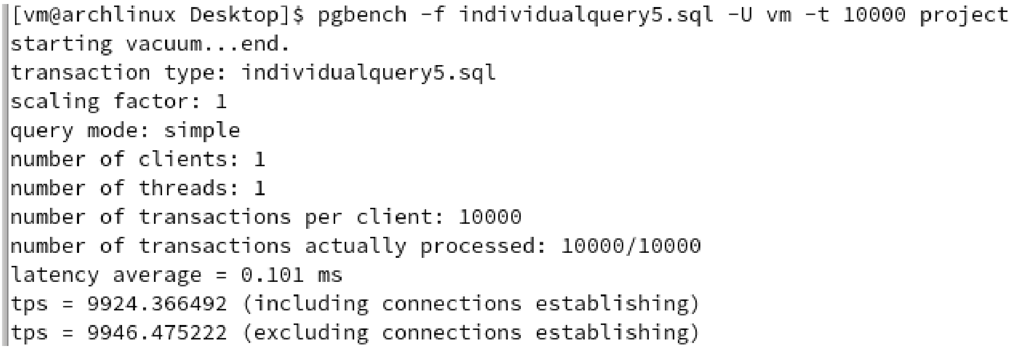
```
[vm@archlinux Desktop]$ pgbench -f individualquery5.sql -U vm -t 10000 project
starting vacuum...end.
transaction type: individualquery5.sql
scaling factor: 1
query mode: simple
number of clients: 1
number of threads: 1
number of transactions per client: 10000
number of transactions actually processed: 10000/10000
latency average = 0.267 ms
tps = 3746.107232 (including connections establishing)
tps = 3749.468604 (excluding connections establishing)
```

Query plantime and execution time before indexes:



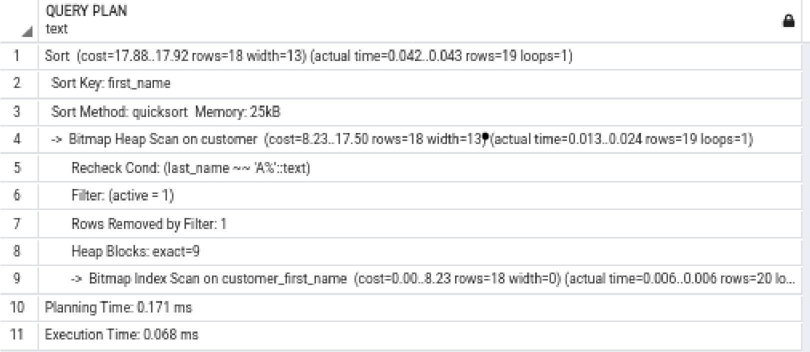
	QUERY PLAN text
1	Sort (cost=18.36..18.41 rows=18 width=13) (actual time=0.086..0.087 rows=19 loops=1)
2	Sort Key: first_name
3	Sort Method: quicksort Memory: 25kB
4	-> Seq Scan on customer (cost=0.00..17.98 rows=18 width=13) (actual time=0.009..0.067 rows=19 loop...
5	Filter: ((last_name ~~ 'A%'::text) AND (active = 1))
6	Rows Removed by Filter: 580
7	Planning Time: 0.132 ms
8	Execution Time: 0.101 ms

Latency and tps after indexes:



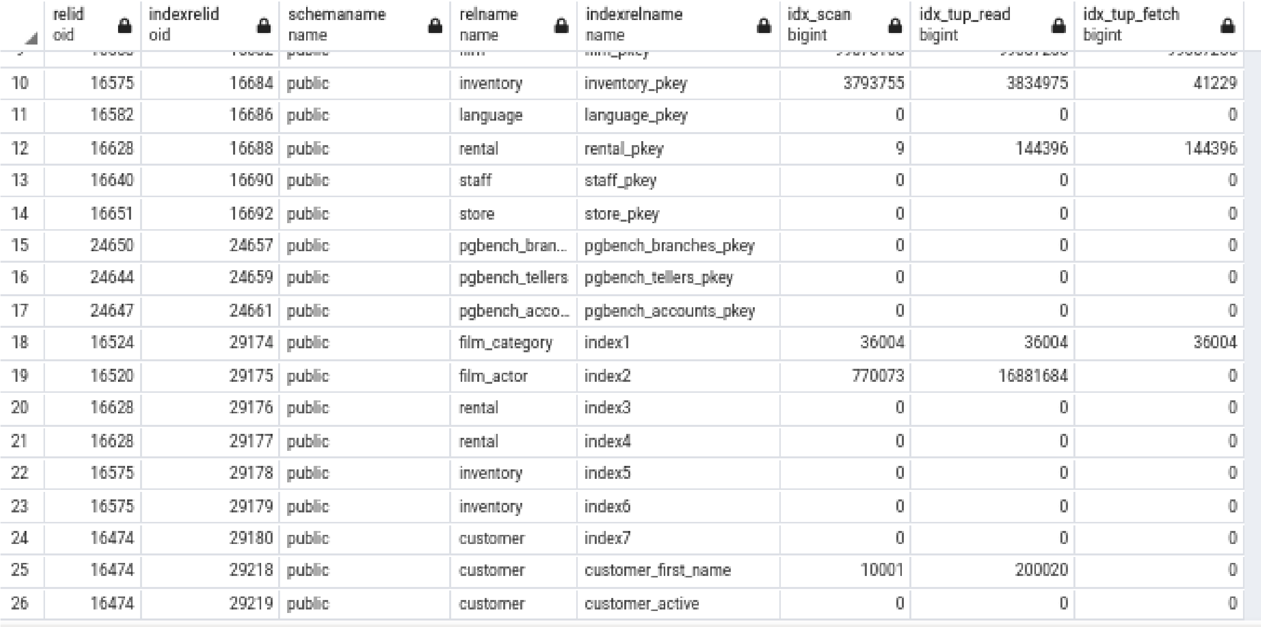
```
[vm@archlinux Desktop]$ pgbench -f individualquery5.sql -U vm -t 10000 project
starting vacuum...end.
transaction type: individualquery5.sql
scaling factor: 1
query mode: simple
number of clients: 1
number of threads: 1
number of transactions per client: 10000
number of transactions actually processed: 10000/10000
latency average = 0.101 ms
tps = 9924.366492 (including connections establishing)
tps = 9946.475222 (excluding connections establishing)
```

Query plantime and execution time after indexes:



```
QUERY PLAN
text
Sort  (cost=17.88..17.92 rows=18 width=13) (actual time=0.042..0.043 rows=19 loops=1)
  Sort Key: first_name
  Sort Method: quicksort  Memory: 25kB
  ->  Bitmap Heap Scan on customer  (cost=8.23..17.50 rows=18 width=13) (actual time=0.013..0.024 rows=19 loops=1)
        Recheck Cond: (last_name ~~ 'A%'::text)
        Filter: (active = 1)
        Rows Removed by Filter: 1
        Heap Blocks: exact=9
        ->  Bitmap Index Scan on customer_first_name  (cost=0.00..8.23 rows=18 width=0) (actual time=0.006..0.006 rows=20 lo...
Planning Time: 0.171 ms
Execution Time: 0.068 ms
```

Index film_film_id_tree usage and the indexes used for the whole database:



	relid oid	indexrelid oid	schemaname name	relname name	indexrelname name	idx_scan bigint	idx_tup_read bigint	idx_tup_fetch bigint
10	16575	16684	public	inventory	inventory_pkey	3793755	3834975	41229
11	16582	16686	public	language	language_pkey	0	0	0
12	16628	16688	public	rental	rental_pkey	9	144396	144396
13	16640	16690	public	staff	staff_pkey	0	0	0
14	16651	16692	public	store	store_pkey	0	0	0
15	24650	24657	public	pgbench_bran...	pgbench_branches_pkey	0	0	0
16	24644	24659	public	pgbench_tellers	pgbench_tellers_pkey	0	0	0
17	24647	24661	public	pgbench_acco...	pgbench_accounts_pkey	0	0	0
18	16524	29174	public	film_category	index1	36004	36004	36004
19	16520	29175	public	film_actor	index2	770073	16881684	0
20	16628	29176	public	rental	index3	0	0	0
21	16628	29177	public	rental	index4	0	0	0
22	16575	29178	public	inventory	index5	0	0	0
23	16575	29179	public	inventory	index6	0	0	0
24	16474	29180	public	customer	index7	0	0	0
25	16474	29218	public	customer	customer_first_name	10001	200020	0
26	16474	29219	public	customer	customer_active	0	0	0

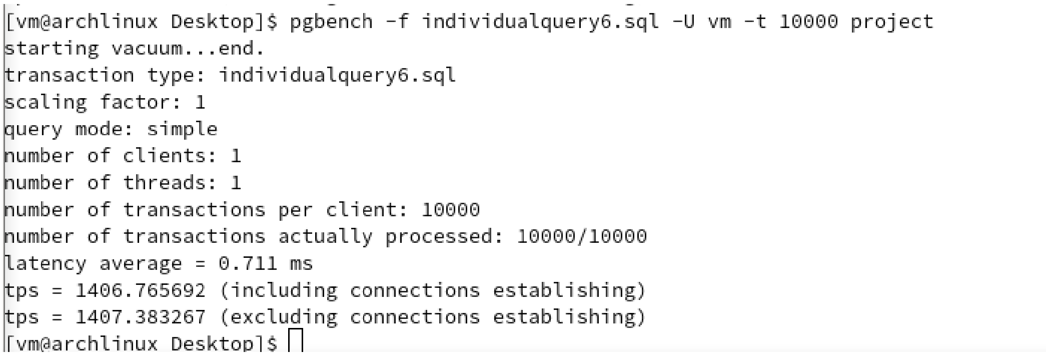
QUERY6:

This query used Index3 that also used in optmizing the database in general

CREATE INDEX customer_last_name on customer USING BTREE(customer_id) where customer.last_name like 'A%';

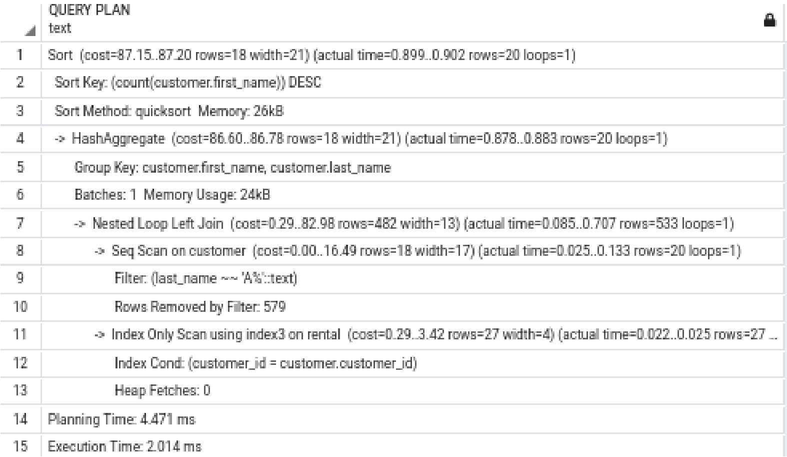
Since there is only 20 names that starts with A,it is very useful to do partial index,moreover BTREE tree let us find any record in LOG(N)

Latency and tps before indexes:



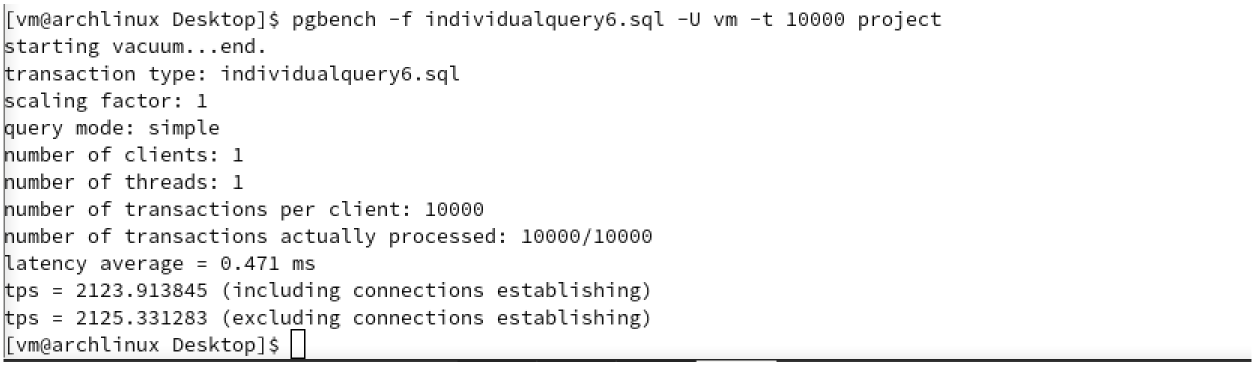
```
[vm@archlinux Desktop]$ pgbench -f individualquery6.sql -U vm -t 10000 project
starting vacuum...end.
transaction type: individualquery6.sql
scaling factor: 1
query mode: simple
number of clients: 1
number of threads: 1
number of transactions per client: 10000
number of transactions actually processed: 10000/10000
latency average = 0.711 ms
tps = 1406.765692 (including connections establishing)
tps = 1407.383267 (excluding connections establishing)
[vm@archlinux Desktop]$
```

Query plantime and execution time before indexes:



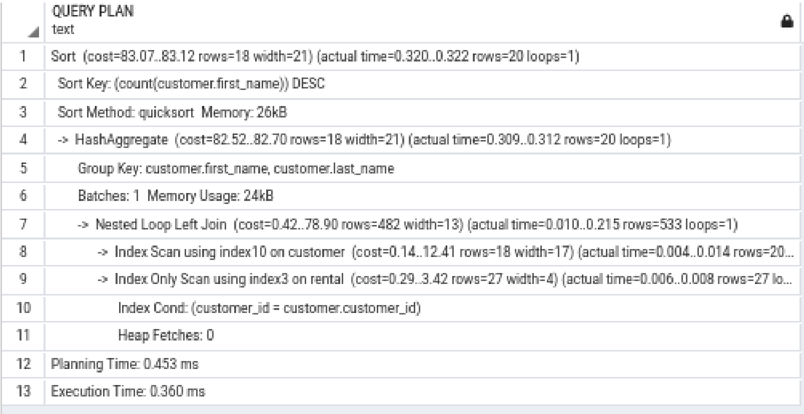
	QUERY PLAN text
1	Sort (cost=87.15..87.20 rows=18 width=21) (actual time=0.899..0.902 rows=20 loops=1)
2	Sort Key: (count(customer.first_name)) DESC
3	Sort Method: quicksort Memory: 26kB
4	-> HashAggregate (cost=86.60..86.78 rows=18 width=21) (actual time=0.878..0.883 rows=20 loops=1)
5	Group Key: customer.first_name, customer.last_name
6	Batches: 1 Memory Usage: 24kB
7	-> Nested Loop Left Join (cost=0.29..82.98 rows=482 width=13) (actual time=0.085..0.707 rows=533 loops=1)
8	-> Seq Scan on customer (cost=0.00..16.49 rows=18 width=17) (actual time=0.025..0.133 rows=20 loops=1)
9	Filter: (last_name ~~ 'A%'::text)
10	Rows Removed by Filter: 579
11	-> Index Only Scan using index3 on rental (cost=0.29..3.42 rows=27 width=4) (actual time=0.022..0.025 rows=27 ...
12	Index Cond: (customer_id = customer.customer_id)
13	Heap Fetches: 0
14	Planning Time: 4.471 ms
15	Execution Time: 2.014 ms

Latency and tps after indexes:



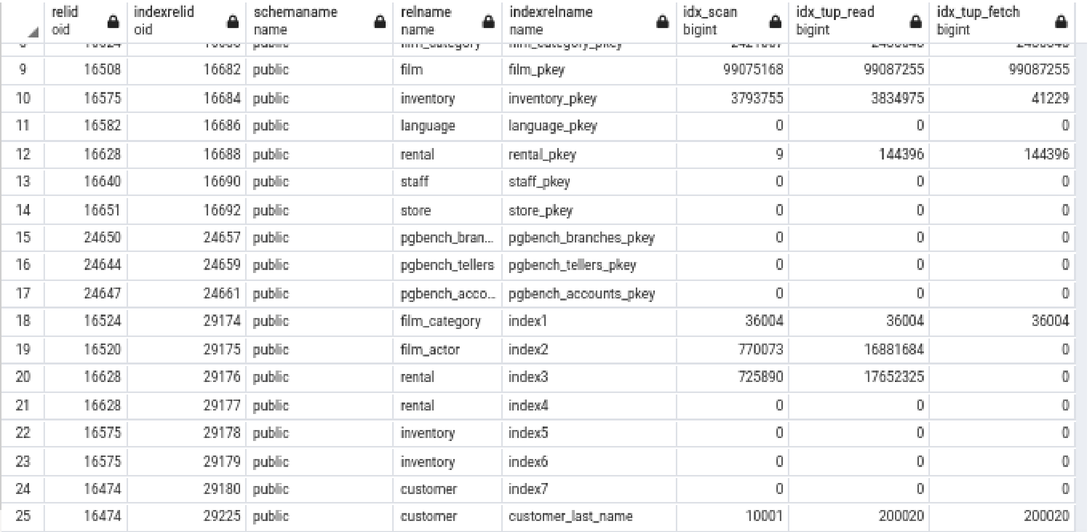
```
[vm@archlinux Desktop]$ pgbench -f individualquery6.sql -U vm -t 10000 project
starting vacuum...end.
transaction type: individualquery6.sql
scaling factor: 1
query mode: simple
number of clients: 1
number of threads: 1
number of transactions per client: 10000
number of transactions actually processed: 10000/10000
latency average = 0.471 ms
tps = 2123.913845 (including connections establishing)
tps = 2125.331283 (excluding connections establishing)
[vm@archlinux Desktop]$
```

Query plantime and execution time after indexes:



```
   QUERY PLAN
   text
1  Sort  (cost=83.07..83.12 rows=18 width=21) (actual time=0.320..0.322 rows=20 loops=1)
2   Sort Key: (count(customer.first_name)) DESC
3   Sort Method: quicksort  Memory: 26kB
4   ->  HashAggregate  (cost=82.52..82.70 rows=18 width=21) (actual time=0.309..0.312 rows=20 loops=1)
5         Group Key: customer.first_name, customer.last_name
6         Batches: 1  Memory Usage: 24kB
7         ->  Nested Loop Left Join  (cost=0.42..78.90 rows=482 width=13) (actual time=0.010..0.215 rows=533 loops=1)
8               ->  Index Scan using index10 on customer  (cost=0.14..12.41 rows=18 width=17) (actual time=0.004..0.014 rows=20...
9               ->  Index Only Scan using index3 on rental  (cost=0.29..3.42 rows=27 width=4) (actual time=0.006..0.008 rows=27 lo...
10                    Index Cond: (customer_id = customer.customer_id)
11                    Heap Fetches: 0
12 Planning Time: 0.453 ms
13 Execution Time: 0.360 ms
```

Index customer_last_name usage and the indexes used for the whole database:



	relid oid	indexrelid oid	schemaname name	relname name	indexrelname name	idx_scan bigint	idx_tup_read bigint	idx_tup_fetch bigint
9	16508	16682	public	film	film_pkey	99075168	99087255	99087255
10	16575	16684	public	inventory	inventory_pkey	3793755	3834975	41229
11	16582	16686	public	language	language_pkey	0	0	0
12	16628	16688	public	rental	rental_pkey	9	144396	144396
13	16640	16690	public	staff	staff_pkey	0	0	0
14	16651	16692	public	store	store_pkey	0	0	0
15	24650	24657	public	pgbench_bran...	pgbench_branches_pkey	0	0	0
16	24644	24659	public	pgbench_tellers	pgbench_tellers_pkey	0	0	0
17	24647	24661	public	pgbench_acco...	pgbench_accounts_pkey	0	0	0
18	16524	29174	public	film_category	index1	36004	36004	36004
19	16520	29175	public	film_actor	index2	770073	16881684	0
20	16628	29176	public	rental	index3	725890	17652325	0
21	16628	29177	public	rental	index4	0	0	0
22	16575	29178	public	inventory	index5	0	0	0
23	16575	29179	public	inventory	index6	0	0	0
24	16474	29180	public	customer	index7	0	0	0
25	16474	29225	public	customer	customer_last_name	10001	200020	200020

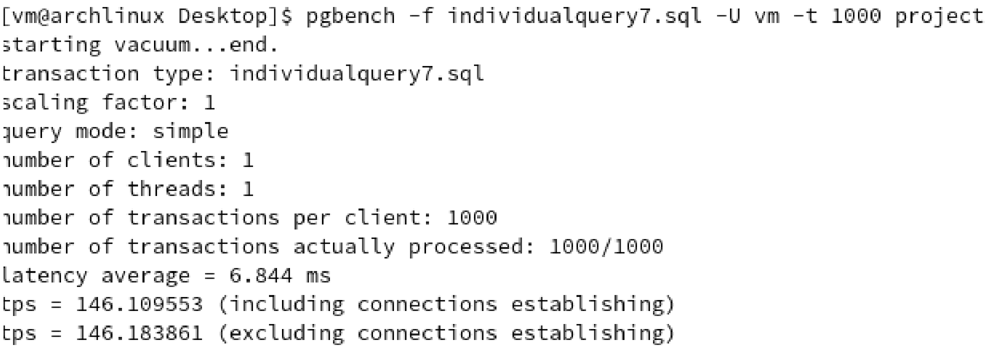
QUERY7:

This query used Index3 that also used in optmizing the database in general

CREATE INDEX customer_last_name on customer USING BTREE(customer_id) where customer.last_name not like 'A%;

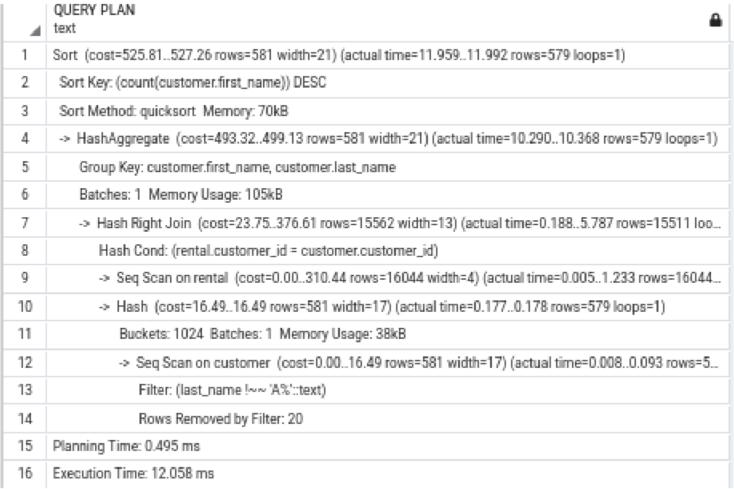
Although The planner used Index3 in scanning and reading but it didnt use it in fetching,This query with NOT LIKE will not use customer_last_name index on NOT LIKE operator statement because the optimizer decides that it is cheaper to do a sequential scan,as most rows their last_name dont start with A

Latency and tps before indexes:



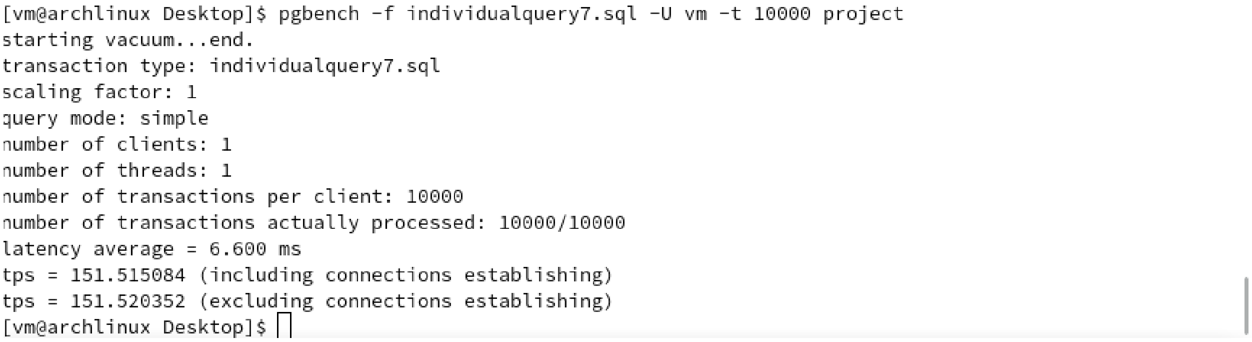
```
[vm@archlinux Desktop]$ pgbench -f individualquery7.sql -U vm -t 1000 project
starting vacuum...end.
transaction type: individualquery7.sql
scaling factor: 1
query mode: simple
number of clients: 1
number of threads: 1
number of transactions per client: 1000
number of transactions actually processed: 1000/1000
latency average = 6.844 ms
tps = 146.109553 (including connections establishing)
tps = 146.183861 (excluding connections establishing)
```

Query plantime and execution time before indexes:



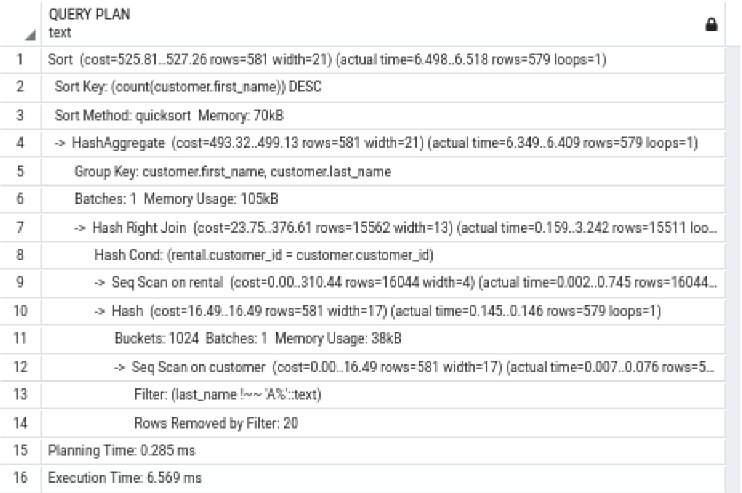
	QUERY PLAN text
1	Sort (cost=525.81..527.26 rows=581 width=21) (actual time=11.959..11.992 rows=579 loops=1)
2	Sort Key: (count(customer.first_name)) DESC
3	Sort Method: quicksort Memory: 70kB
4	-> HashAggregate (cost=493.32..499.13 rows=581 width=21) (actual time=10.290..10.368 rows=579 loops=1)
5	Group Key: customer.first_name, customer.last_name
6	Batches: 1 Memory Usage: 105kB
7	-> Hash Right Join (cost=23.75..376.61 rows=15562 width=13) (actual time=0.188..5.787 rows=15511 loo...
8	Hash Cond: (rental.customer_id = customer.customer_id)
9	-> Seq Scan on rental (cost=0.00..310.44 rows=16044 width=4) (actual time=0.005..1.233 rows=16044...
10	-> Hash (cost=16.49..16.49 rows=581 width=17) (actual time=0.177..0.178 rows=579 loops=1)
11	Buckets: 1024 Batches: 1 Memory Usage: 38kB
12	-> Seq Scan on customer (cost=0.00..16.49 rows=581 width=17) (actual time=0.008..0.093 rows=5...
13	Filter: (last_name !~~ 'A%'::text)
14	Rows Removed by Filter: 20
15	Planning Time: 0.495 ms
16	Execution Time: 12.058 ms

Latency and tps after indexes:



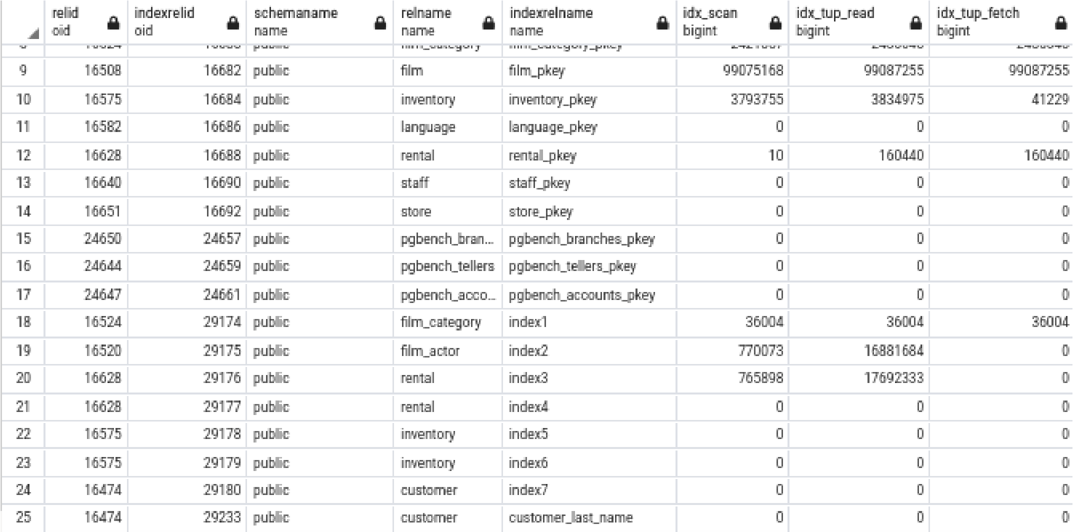
```
[vm@archlinux Desktop]$ pgbench -f individualquery7.sql -U vm -t 10000 project
starting vacuum...end.
transaction type: individualquery7.sql
scaling factor: 1
query mode: simple
number of clients: 1
number of threads: 1
number of transactions per client: 10000
number of transactions actually processed: 10000/10000
latency average = 6.600 ms
tps = 151.515084 (including connections establishing)
tps = 151.520352 (excluding connections establishing)
[vm@archlinux Desktop]$ 
```

Query plantime and execution time after indexes:



```
QUERY PLAN
text
Sort  (cost=525.81..527.26 rows=581 width=21) (actual time=6.498..6.518 rows=579 loops=1)
  Sort Key: (count(customer.first_name)) DESC
  Sort Method: quicksort  Memory: 70kB
  ->  HashAggregate  (cost=493.32..499.13 rows=581 width=21) (actual time=6.349..6.409 rows=579 loops=1)
        Group Key: customer.first_name, customer.last_name
        Batches: 1  Memory Usage: 105kB
        ->  Hash Right Join  (cost=23.75..376.61 rows=15562 width=13) (actual time=0.159..3.242 rows=15511 loo...
              Hash Cond: (rental.customer_id = customer.customer_id)
              ->  Seq Scan on rental  (cost=0.00..310.44 rows=16044 width=4) (actual time=0.002..0.745 rows=16044...
              ->  Hash  (cost=16.49..16.49 rows=581 width=17) (actual time=0.145..0.146 rows=579 loops=1)
                    Buckets: 1024  Batches: 1  Memory Usage: 38kB
                    ->  Seq Scan on customer  (cost=0.00..16.49 rows=581 width=17) (actual time=0.007..0.076 rows=5...
                          Filter: (last_name !~~ 'A%'::text)
                          Rows Removed by Filter: 20
Planning Time: 0.285 ms
Execution Time: 6.569 ms
```

Index customer_last_name usage and the indexes used for the whole database:



	relid oid	indexrelid oid	schemaname name	relname name	indexrelname name	idx_scan bigint	idx_tup_read bigint	idx_tup_fetch bigint
9	16508	16682	public	film	film_pkey	99075168	99087255	99087255
10	16575	16684	public	inventory	inventory_pkey	3793755	3834975	41229
11	16582	16686	public	language	language_pkey	0	0	0
12	16628	16688	public	rental	rental_pkey	10	160440	160440
13	16640	16690	public	staff	staff_pkey	0	0	0
14	16651	16692	public	store	store_pkey	0	0	0
15	24650	24657	public	pgbench_bran...	pgbench_branches_pkey	0	0	0
16	24644	24659	public	pgbench_tellers	pgbench_tellers_pkey	0	0	0
17	24647	24661	public	pgbench_acco...	pgbench_accounts_pkey	0	0	0
18	16524	29174	public	film_category	index1	36004	36004	36004
19	16520	29175	public	film_actor	index2	770073	16881684	0
20	16628	29176	public	rental	index3	765898	17692333	0
21	16628	29177	public	rental	index4	0	0	0
22	16575	29178	public	inventory	index5	0	0	0
23	16575	29179	public	inventory	index6	0	0	0
24	16474	29180	public	customer	index7	0	0	0
25	16474	29233	public	customer	customer_last_name	0	0	0

QUERY8:

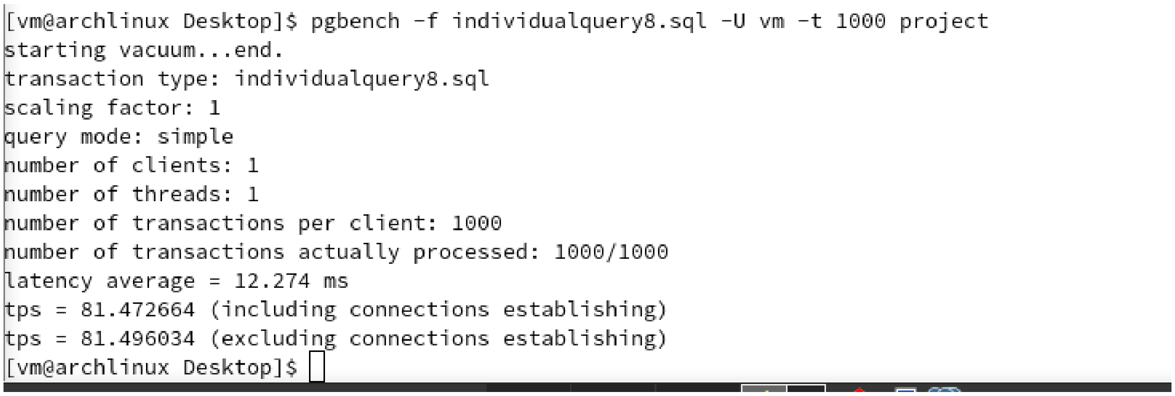
This query used Index3 that also used in optmizing the database in general

CREATE INDEX payment_rental_id_hash on payment USING HASH (rental_id);

CREATE INDEX payment_amount_hash on payment USING HASH (amount);

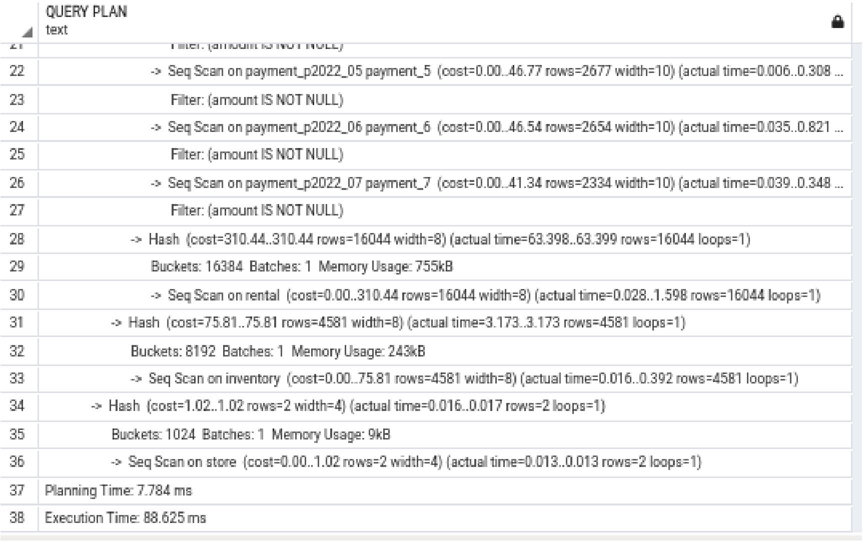
Although Index3 used for scanning and reading but it didnt use in fetching , payment_amount_hash indexes didnt work,if 99% of the rows the amount IS NOT NULL, the index isn't buying you anything over just letting a full table scan happen; in fact, it would be less efficient since it would require extra disk reads. If however, only 1% of rows have name IS NOT NULL, then this represents huge savings as PostgreSQL can ignore most of the table for your query. If your table is very large, even eliminating 50% of the rows might be worth it,moreover payment_rental_id_hash didn’t work because it will scan all the payment table ro get the sum of the amount

Latency and tps before indexes:



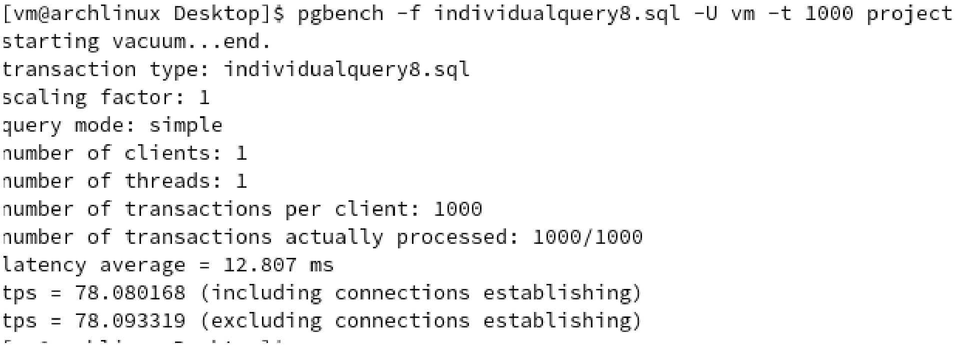
```
[vm@archlinux Desktop]$ pgbench -f individualquery8.sql -U vm -t 1000 project
starting vacuum...end.
transaction type: individualquery8.sql
scaling factor: 1
query mode: simple
number of clients: 1
number of threads: 1
number of transactions per client: 1000
number of transactions actually processed: 1000/1000
latency average = 12.274 ms
tps = 81.472664 (including connections establishing)
tps = 81.496034 (excluding connections establishing)
[vm@archlinux Desktop]$
```

Query plantime and execution time before indexes:



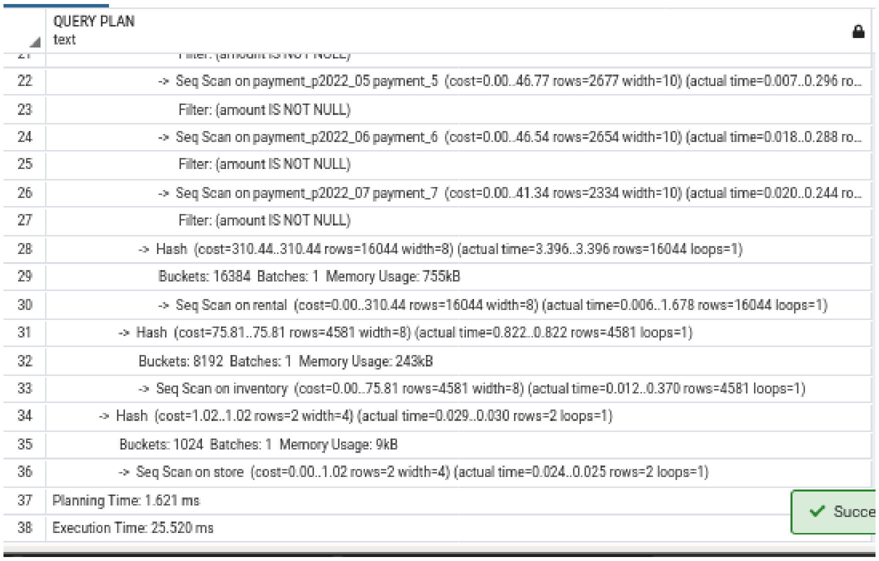
	QUERY PLAN text
21	Filter: (amount IS NOT NULL)
22	-> Seq Scan on payment_p2022_05 payment_5 (cost=0.00..46.77 rows=2677 width=10) (actual time=0.006..0.308 ...
23	Filter: (amount IS NOT NULL)
24	-> Seq Scan on payment_p2022_06 payment_6 (cost=0.00..46.54 rows=2654 width=10) (actual time=0.035..0.821 ...
25	Filter: (amount IS NOT NULL)
26	-> Seq Scan on payment_p2022_07 payment_7 (cost=0.00..41.34 rows=2334 width=10) (actual time=0.039..0.348 ...
27	Filter: (amount IS NOT NULL)
28	-> Hash (cost=310.44..310.44 rows=16044 width=8) (actual time=63.398..63.399 rows=16044 loops=1)
29	Buckets: 16384 Batches: 1 Memory Usage: 755kB
30	-> Seq Scan on rental (cost=0.00..310.44 rows=16044 width=8) (actual time=0.028..1.598 rows=16044 loops=1)
31	-> Hash (cost=75.81..75.81 rows=4581 width=8) (actual time=3.173..3.173 rows=4581 loops=1)
32	Buckets: 8192 Batches: 1 Memory Usage: 243kB
33	-> Seq Scan on inventory (cost=0.00..75.81 rows=4581 width=8) (actual time=0.016..0.392 rows=4581 loops=1)
34	-> Hash (cost=1.02..1.02 rows=2 width=4) (actual time=0.016..0.017 rows=2 loops=1)
35	Buckets: 1024 Batches: 1 Memory Usage: 9kB
36	-> Seq Scan on store (cost=0.00..1.02 rows=2 width=4) (actual time=0.013..0.013 rows=2 loops=1)
37	Planning Time: 7.784 ms
38	Execution Time: 88.625 ms

Latency and tps after indexes:



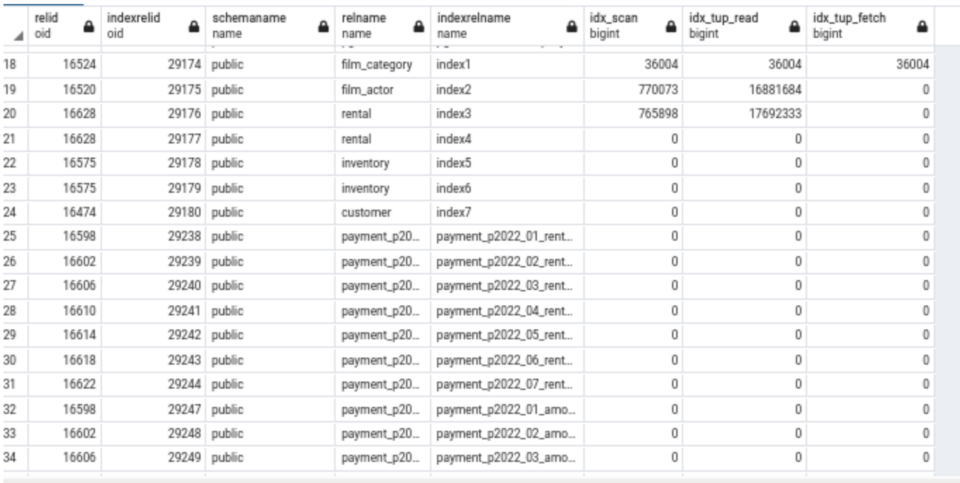
```
[vm@archlinux Desktop]$ pgbench -f individualquery8.sql -U vm -t 1000 project
starting vacuum...end.
transaction type: individualquery8.sql
scaling factor: 1
query mode: simple
number of clients: 1
number of threads: 1
number of transactions per client: 1000
number of transactions actually processed: 1000/1000
latency average = 12.807 ms
tps = 78.080168 (including connections establishing)
tps = 78.093319 (excluding connections establishing)
```

Query plantime and execution time after indexes:



```
QUERY PLAN
text
21                          Filter: (amount IS NOT NULL)
22                      ->  Seq Scan on payment_p2022_05 payment_5  (cost=0.00..46.77 rows=2677 width=10) (actual time=0.007..0.296 ro...
23                          Filter: (amount IS NOT NULL)
24                      ->  Seq Scan on payment_p2022_06 payment_6  (cost=0.00..46.54 rows=2654 width=10) (actual time=0.018..0.288 ro...
25                          Filter: (amount IS NOT NULL)
26                      ->  Seq Scan on payment_p2022_07 payment_7  (cost=0.00..41.34 rows=2334 width=10) (actual time=0.020..0.244 ro...
27                          Filter: (amount IS NOT NULL)
28                  ->  Hash  (cost=310.44..310.44 rows=16044 width=8) (actual time=3.396..3.396 rows=16044 loops=1)
29                      Buckets: 16384  Batches: 1  Memory Usage: 755kB
30                      ->  Seq Scan on rental  (cost=0.00..310.44 rows=16044 width=8) (actual time=0.006..1.678 rows=16044 loops=1)
31              ->  Hash  (cost=75.81..75.81 rows=4581 width=8) (actual time=0.822..0.822 rows=4581 loops=1)
32                  Buckets: 8192  Batches: 1  Memory Usage: 243kB
33                  ->  Seq Scan on inventory  (cost=0.00..75.81 rows=4581 width=8) (actual time=0.012..0.370 rows=4581 loops=1)
34          ->  Hash  (cost=1.02..1.02 rows=2 width=4) (actual time=0.029..0.030 rows=2 loops=1)
35              Buckets: 1024  Batches: 1  Memory Usage: 9kB
36              ->  Seq Scan on store  (cost=0.00..1.02 rows=2 width=4) (actual time=0.024..0.025 rows=2 loops=1)
37  Planning Time: 1.621 ms
38  Execution Time: 25.520 ms
```

Index payment_rental_id_hash and payment_amount_hash usage and the indexes used for the whole database:



	relid oid	indexrelid oid	schemaname name	relname name	indexrelname name	idx_scan bigint	idx_tup_read bigint	idx_tup_fetch bigint
18	16524	29174	public	film_category	index1	36004	36004	36004
19	16520	29175	public	film_actor	index2	770073	16881684	0
20	16628	29176	public	rental	index3	765898	17692333	0
21	16628	29177	public	rental	index4	0	0	0
22	16575	29178	public	inventory	index5	0	0	0
23	16575	29179	public	inventory	index6	0	0	0
24	16474	29180	public	customer	index7	0	0	0
25	16598	29238	public	payment_p20...	payment_p2022_01_rent...	0	0	0
26	16602	29239	public	payment_p20...	payment_p2022_02_rent...	0	0	0
27	16606	29240	public	payment_p20...	payment_p2022_03_rent...	0	0	0
28	16610	29241	public	payment_p20...	payment_p2022_04_rent...	0	0	0
29	16614	29242	public	payment_p20...	payment_p2022_05_rent...	0	0	0
30	16618	29243	public	payment_p20...	payment_p2022_06_rent...	0	0	0
31	16622	29244	public	payment_p20...	payment_p2022_07_rent...	0	0	0
32	16598	29247	public	payment_p20...	payment_p2022_01_amo...	0	0	0
33	16602	29248	public	payment_p20...	payment_p2022_02_amo...	0	0	0
34	16606	29249	public	payment_p20...	payment_p2022_03_amo...	0	0	0

QUERY9:

This query used Index1,4,5 that also used in optmizing the database in general

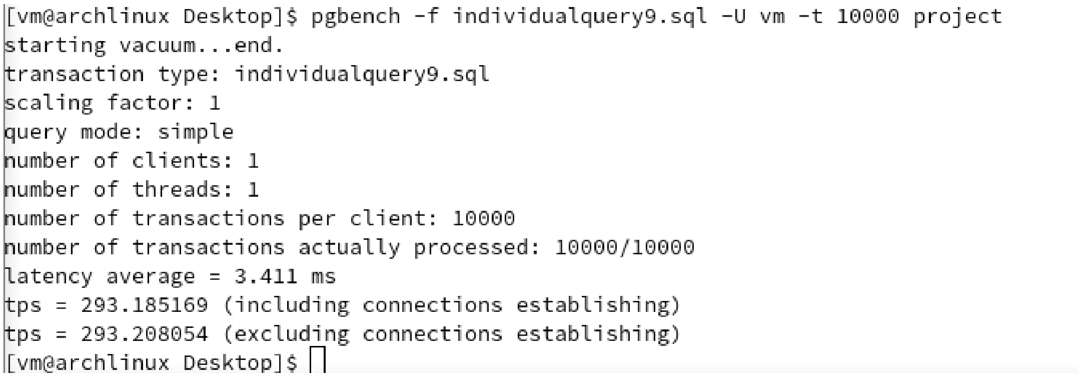
CREATE INDEX rental_inventory_id_tree on rental USING BTREE (inventory_id);

CREATE INDEX payment_rental_id_hash on payment USING HASH (rental_id);

CREATE INDEX film_film_id_tree on film USING BTREE (film_id) where film.title like '%A';

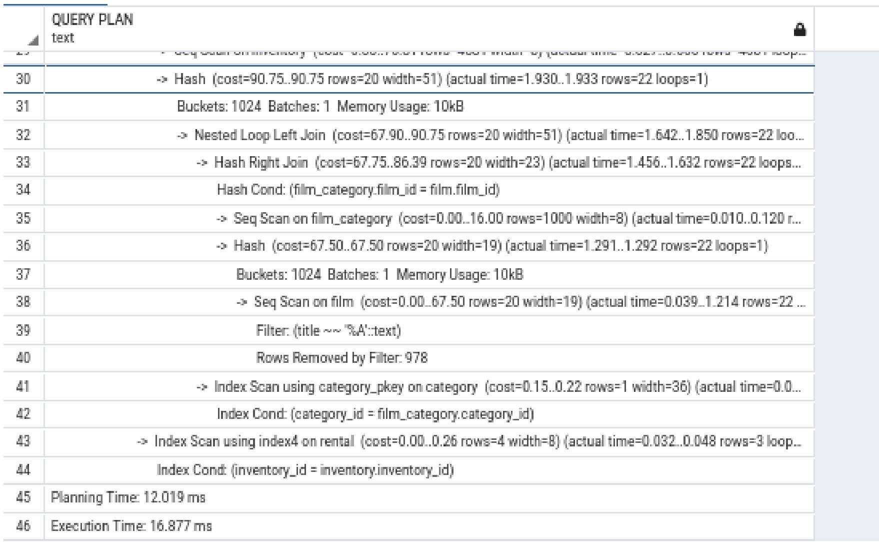
BTREE let us get any record in tabel rental using inventory_id LOG(N),AND hash lesy us get any record in payment using rental_id in O(1),moreover it will be benefical using partial index on film_id because we are only intrested in records end with A

Latency and tps before indexes:



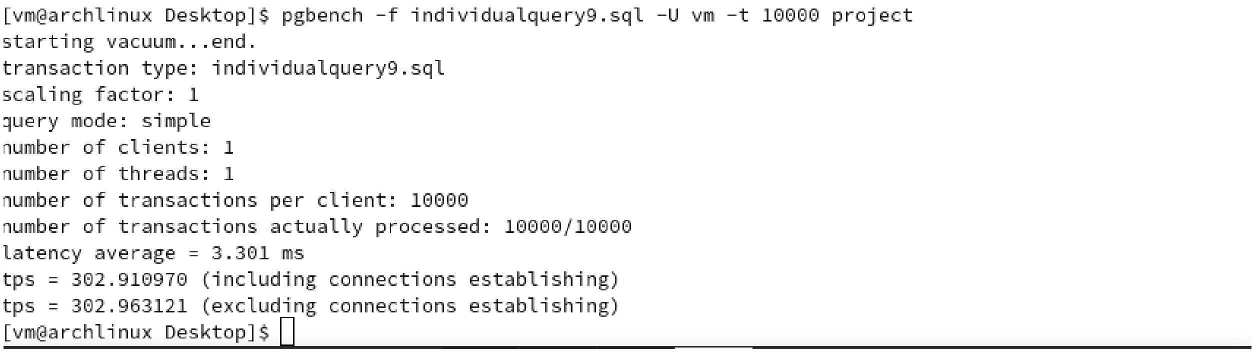
```
[vm@archlinux Desktop]$ pgbench -f individualquery9.sql -U vm -t 10000 project
starting vacuum...end.
transaction type: individualquery9.sql
scaling factor: 1
query mode: simple
number of clients: 1
number of threads: 1
number of transactions per client: 10000
number of transactions actually processed: 10000/10000
latency average = 3.411 ms
tps = 293.185169 (including connections establishing)
tps = 293.208054 (excluding connections establishing)
[vm@archlinux Desktop]$ 
```

Query plantime and execution time before indexes:



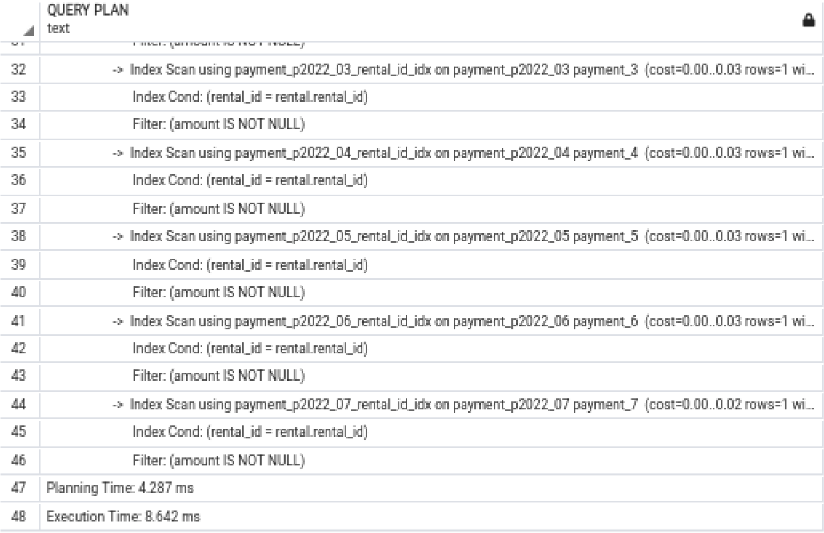
```
    QUERY PLAN
    text
30              -> Hash  (cost=90.75..90.75 rows=20 width=51) (actual time=1.930..1.933 rows=22 loops=1)
31                   Buckets: 1024  Batches: 1  Memory Usage: 10kB
32                   -> Nested Loop Left Join  (cost=67.90..90.75 rows=20 width=51) (actual time=1.642..1.850 rows=22 loo...
33                         -> Hash Right Join  (cost=67.75..86.39 rows=20 width=23) (actual time=1.456..1.632 rows=22 loops...
34                               Hash Cond: (film_category.film_id = film.film_id)
35                               -> Seq Scan on film_category  (cost=0.00..16.00 rows=1000 width=8) (actual time=0.010..0.120 r...
36                               -> Hash  (cost=67.50..67.50 rows=20 width=19) (actual time=1.291..1.292 rows=22 loops=1)
37                                     Buckets: 1024  Batches: 1  Memory Usage: 10kB
38                                     -> Seq Scan on film  (cost=0.00..67.50 rows=20 width=19) (actual time=0.039..1.214 rows=22 ...
39                                           Filter: (title ~~ '%A'::text)
40                                           Rows Removed by Filter: 978
41                         -> Index Scan using category_pkey on category  (cost=0.15..0.22 rows=1 width=36) (actual time=0.0...
42                               Index Cond: (category_id = film_category.category_id)
43             -> Index Scan using index4 on rental  (cost=0.00..0.26 rows=4 width=8) (actual time=0.032..0.048 rows=3 loop...
44                   Index Cond: (inventory_id = inventory.inventory_id)
45  Planning Time: 12.019 ms
46  Execution Time: 16.877 ms
```

Latency and tps after indexes:



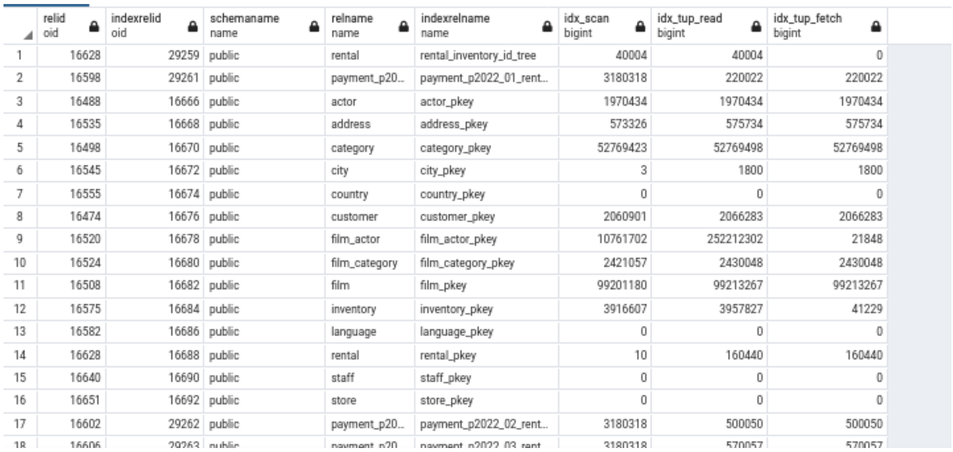
```
[vm@archlinux Desktop]$ pgbench -f individualquery9.sql -U vm -t 10000 project
starting vacuum...end.
transaction type: individualquery9.sql
scaling factor: 1
query mode: simple
number of clients: 1
number of threads: 1
number of transactions per client: 10000
number of transactions actually processed: 10000/10000
latency average = 3.301 ms
tps = 302.910970 (including connections establishing)
tps = 302.963121 (excluding connections establishing)
[vm@archlinux Desktop]$ 
```

Query plantime and execution time after indexes:

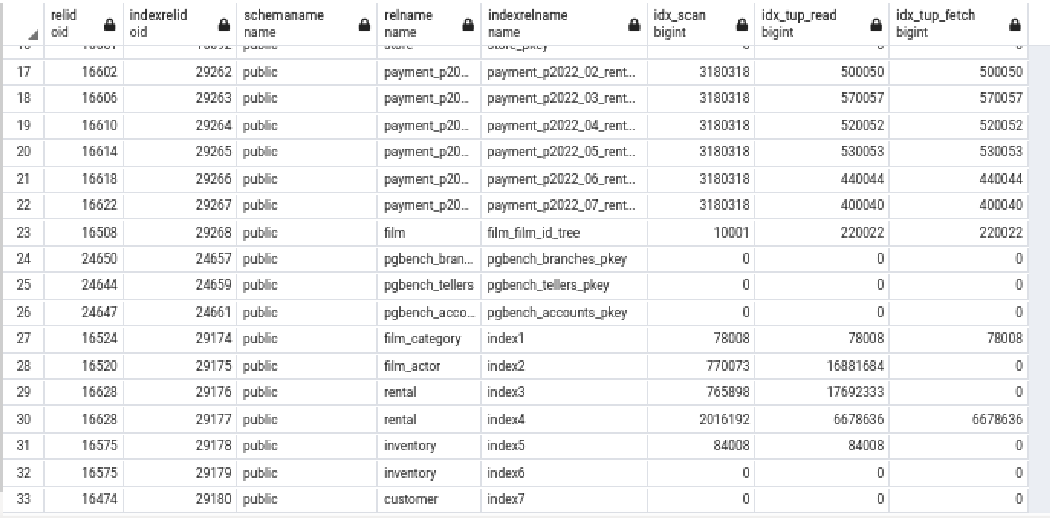


	QUERY PLAN text
31	Filter: (amount IS NOT NULL)
32	-> Index Scan using payment_p2022_03_rental_id_idx on payment_p2022_03 payment_3 (cost=0.00..0.03 rows=1 wi...
33	Index Cond: (rental_id = rental.rental_id)
34	Filter: (amount IS NOT NULL)
35	-> Index Scan using payment_p2022_04_rental_id_idx on payment_p2022_04 payment_4 (cost=0.00..0.03 rows=1 wi...
36	Index Cond: (rental_id = rental.rental_id)
37	Filter: (amount IS NOT NULL)
38	-> Index Scan using payment_p2022_05_rental_id_idx on payment_p2022_05 payment_5 (cost=0.00..0.03 rows=1 wi...
39	Index Cond: (rental_id = rental.rental_id)
40	Filter: (amount IS NOT NULL)
41	-> Index Scan using payment_p2022_06_rental_id_idx on payment_p2022_06 payment_6 (cost=0.00..0.03 rows=1 wi...
42	Index Cond: (rental_id = rental.rental_id)
43	Filter: (amount IS NOT NULL)
44	-> Index Scan using payment_p2022_07_rental_id_idx on payment_p2022_07 payment_7 (cost=0.00..0.02 rows=1 wi...
45	Index Cond: (rental_id = rental.rental_id)
46	Filter: (amount IS NOT NULL)
47	Planning Time: 4.287 ms
48	Execution Time: 8.642 ms

Indexes rental_inventory_id_tree and payment_rental_id_hash and film_film_id_tree usage and the indexes used for the whole database:



	relid oid	indexrelid oid	schemaname name	relname name	indexrelname name	idx_scan bigint	idx_tup_read bigint	idx_tup_fetch bigint
1	16628	29259	public	rental	rental_inventory_id_tree	40004	40004	0
2	16598	29261	public	payment_p20...	payment_p2022_01_rent...	3180318	220022	220022
3	16488	16666	public	actor	actor_pkey	1970434	1970434	1970434
4	16535	16668	public	address	address_pkey	573326	575734	575734
5	16498	16670	public	category	category_pkey	52769423	52769498	52769498
6	16545	16672	public	city	city_pkey	3	1800	1800
7	16555	16674	public	country	country_pkey	0	0	0
8	16474	16676	public	customer	customer_pkey	2060901	2066283	2066283
9	16520	16678	public	film_actor	film_actor_pkey	10761702	252212302	21848
10	16524	16680	public	film_category	film_category_pkey	2421057	2430048	2430048
11	16508	16682	public	film	film_pkey	99201180	99213267	99213267
12	16575	16684	public	inventory	inventory_pkey	3916607	3957827	41229
13	16582	16686	public	language	language_pkey	0	0	0
14	16628	16688	public	rental	rental_pkey	10	160440	160440
15	16640	16690	public	staff	staff_pkey	0	0	0
16	16651	16692	public	store	store_pkey	0	0	0
17	16602	29262	public	payment_p20...	payment_p2022_02_rent...	3180318	500050	500050
18	16606	29263	public	payment_p20...	payment_p2022_03_rent...	3180318	570057	570057

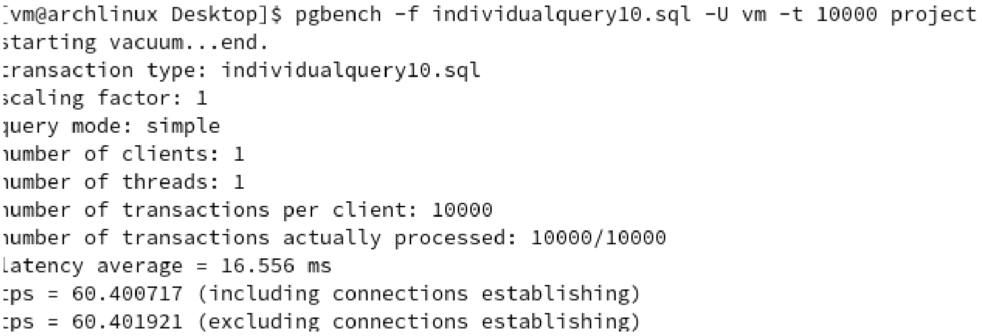


	relid oid	indexrelid oid	schemaname name	relname name	indexrelname name	idx_scan bigint	idx_tup_read bigint	idx_tup_fetch bigint
17	16602	29262	public	payment_p20...	payment_p2022_02_rent...	3180318	500050	500050
18	16606	29263	public	payment_p20...	payment_p2022_03_rent...	3180318	570057	570057
19	16610	29264	public	payment_p20...	payment_p2022_04_rent...	3180318	520052	520052
20	16614	29265	public	payment_p20...	payment_p2022_05_rent...	3180318	530053	530053
21	16618	29266	public	payment_p20...	payment_p2022_06_rent...	3180318	440044	440044
22	16622	29267	public	payment_p20...	payment_p2022_07_rent...	3180318	400040	400040
23	16508	29268	public	film	film_film_id_tree	10001	220022	220022
24	24650	24657	public	pgbench_bran...	pgbench_branches_pkey	0	0	0
25	24644	24659	public	pgbench_tellers	pgbench_tellers_pkey	0	0	0
26	24647	24661	public	pgbench_acco...	pgbench_accounts_pkey	0	0	0
27	16524	29174	public	film_category	index1	78008	78008	78008
28	16520	29175	public	film_actor	index2	770073	16881684	0
29	16628	29176	public	rental	index3	765898	17692333	0
30	16628	29177	public	rental	index4	2016192	6678636	6678636
31	16575	29178	public	inventory	index5	84008	84008	0
32	16575	29179	public	inventory	index6	0	0	0
33	16474	29180	public	customer	index7	0	0	0

QUERY 10:

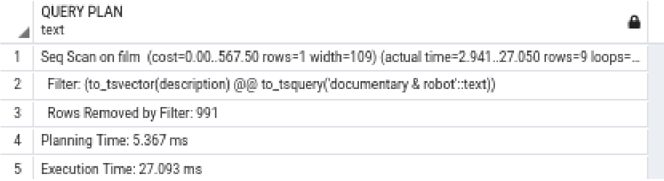
CREATE INDEX film_film_description ON film USING HASH(description);

Latency and tps before indexes:



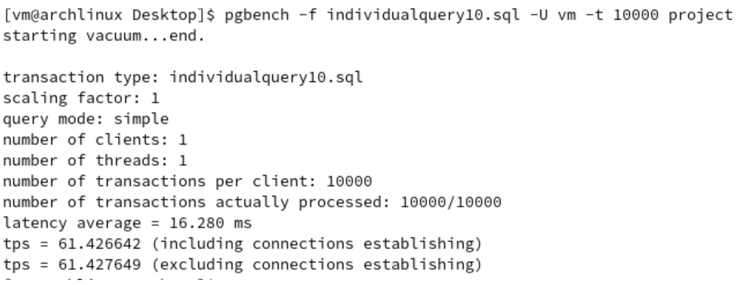
```
[vm@archlinux Desktop]$ pgbench -f individualquery10.sql -U vm -t 10000 project
starting vacuum...end.
transaction type: individualquery10.sql
scaling factor: 1
query mode: simple
number of clients: 1
number of threads: 1
number of transactions per client: 10000
number of transactions actually processed: 10000/10000
latency average = 16.556 ms
tps = 60.400717 (including connections establishing)
tps = 60.401921 (excluding connections establishing)
```

Query plantime and execution time before indexes:



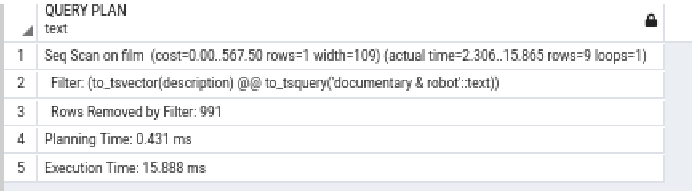
	QUERY PLAN text
1	Seq Scan on film (cost=0.00..567.50 rows=1 width=109) (actual time=2.941..27.050 rows=9 loops=...
2	Filter: (to_tsvector(description) @@ to_tsquery('documentary & robot'::text))
3	Rows Removed by Filter: 991
4	Planning Time: 5.367 ms
5	Execution Time: 27.093 ms

Latency and tps after indexes:



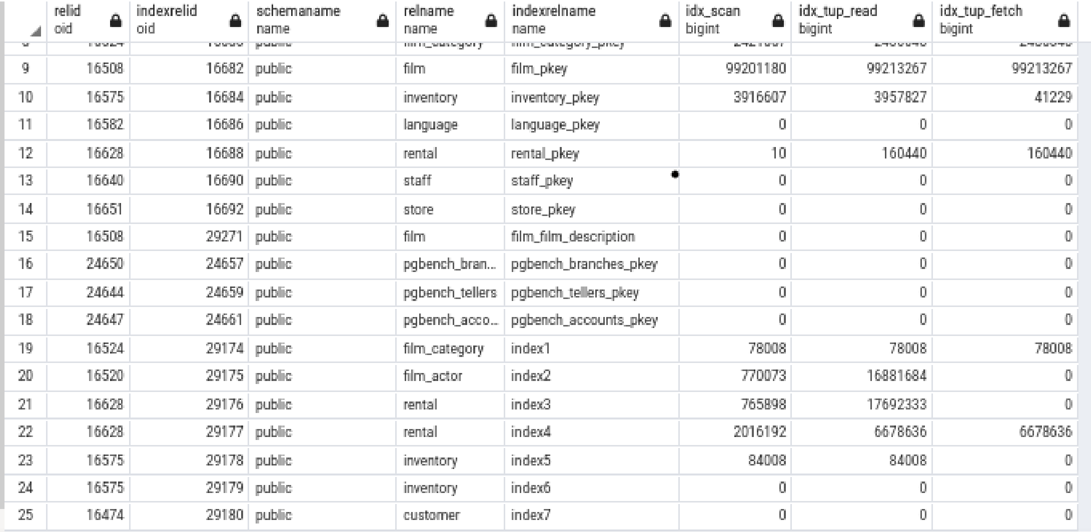
```
[vm@archlinux Desktop]$ pgbench -f individualquery10.sql -U vm -t 10000 project
starting vacuum...end.

transaction type: individualquery10.sql
scaling factor: 1
query mode: simple
number of clients: 1
number of threads: 1
number of transactions per client: 10000
number of transactions actually processed: 10000/10000
latency average = 16.280 ms
tps = 61.426642 (including connections establishing)
tps = 61.427649 (excluding connections establishing)
```

Query plantime and execution time after indexes:



	QUERY PLAN text
1	Seq Scan on film (cost=0.00..567.50 rows=1 width=109) (actual time=2.306..15.865 rows=9 loops=1)
2	Filter: (to_tsvector(description) @@ to_tsquery('documentary & robot'::text))
3	Rows Removed by Filter: 991
4	Planning Time: 0.431 ms
5	Execution Time: 15.888 ms

Index film_film_description usage and the indexes used for the whole database:



	relid oid	indexrelid oid	schemaname name	relname name	indexrelname name	idx_scan bigint	idx_tup_read bigint	idx_tup_fetch bigint
9	16508	16682	public	film	film_pkey	99201180	99213267	99213267
10	16575	16684	public	inventory	inventory_pkey	3916607	3957827	41229
11	16582	16686	public	language	language_pkey	0	0	0
12	16628	16688	public	rental	rental_pkey	10	160440	160440
13	16640	16690	public	staff	staff_pkey	0	0	0
14	16651	16692	public	store	store_pkey	0	0	0
15	16508	29271	public	film	film_film_description	0	0	0
16	24650	24657	public	pgbench_bran...	pgbench_branches_pkey	0	0	0
17	24644	24659	public	pgbench_tellers	pgbench_tellers_pkey	0	0	0
18	24647	24661	public	pgbench_acco...	pgbench_accounts_pkey	0	0	0
19	16524	29174	public	film_category	index1	78008	78008	78008
20	16520	29175	public	film_actor	index2	770073	16881684	0
21	16628	29176	public	rental	index3	765898	17692333	0
22	16628	29177	public	rental	index4	2016192	6678636	6678636
23	16575	29178	public	inventory	index5	84008	84008	0
24	16575	29179	public	inventory	index6	0	0	0
25	16474	29180	public	customer	index7	0	0	0

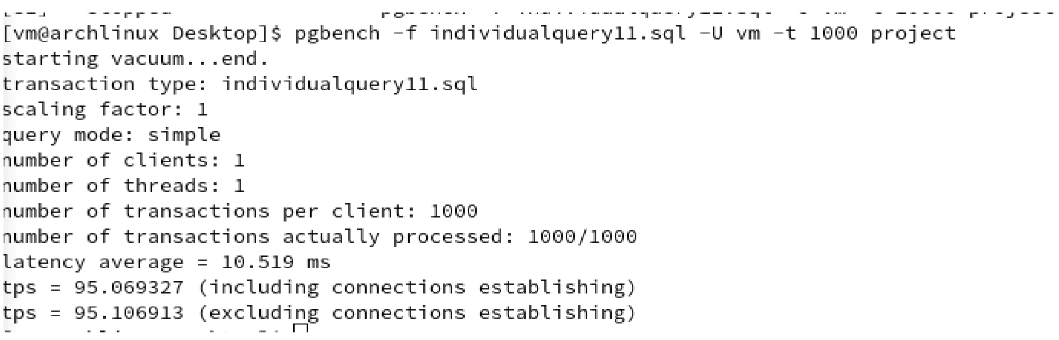
QUERY11:

This query used Index1,4 that also used in optmizing the database in general

CREATE INDEX category_category_name ON category using HASH(name);

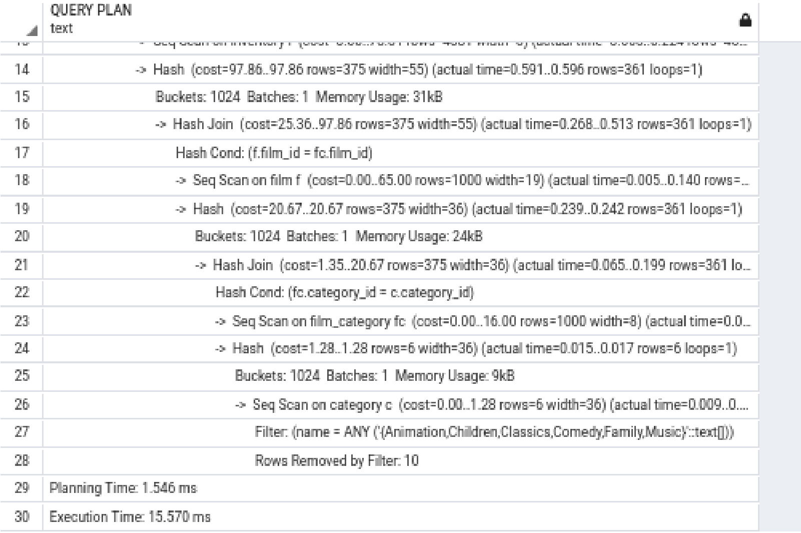
category_category_name index didnt work because category table is too small so it is not effiecient

Latency and tps before indexes:



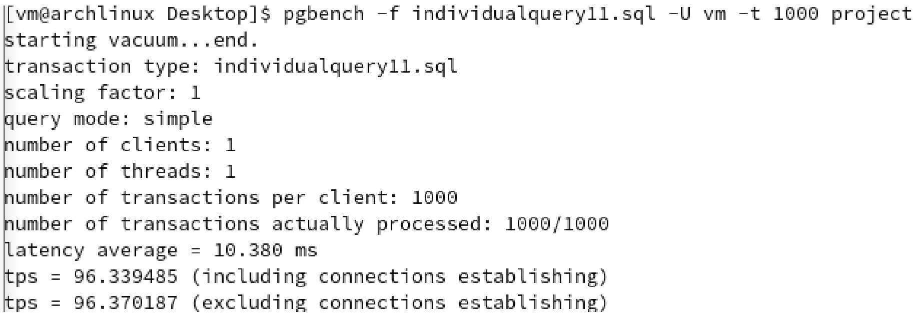
```
[vm@archlinux Desktop]$ pgbench -f individualquery11.sql -U vm -t 1000 project
starting vacuum...end.
transaction type: individualquery11.sql
scaling factor: 1
query mode: simple
number of clients: 1
number of threads: 1
number of transactions per client: 1000
number of transactions actually processed: 1000/1000
latency average = 10.519 ms
tps = 95.069327 (including connections establishing)
tps = 95.106913 (excluding connections establishing)
```

Query plantime and execution time before indexes:



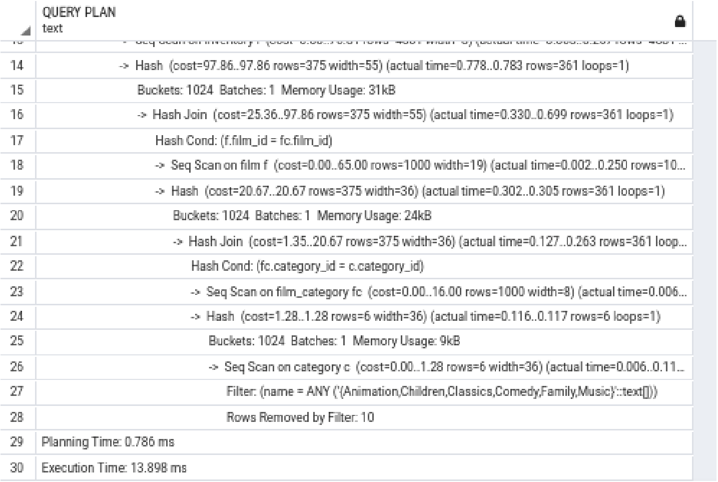
	QUERY PLAN text
14	-> Hash (cost=97.86..97.86 rows=375 width=55) (actual time=0.591..0.596 rows=361 loops=1)
15	Buckets: 1024 Batches: 1 Memory Usage: 31kB
16	-> Hash Join (cost=25.36..97.86 rows=375 width=55) (actual time=0.268..0.513 rows=361 loops=1)
17	Hash Cond: (f.film_id = fc.film_id)
18	-> Seq Scan on film f (cost=0.00..65.00 rows=1000 width=19) (actual time=0.005..0.140 rows=...
19	-> Hash (cost=20.67..20.67 rows=375 width=36) (actual time=0.239..0.242 rows=361 loops=1)
20	Buckets: 1024 Batches: 1 Memory Usage: 24kB
21	-> Hash Join (cost=1.35..20.67 rows=375 width=36) (actual time=0.065..0.199 rows=361 lo...
22	Hash Cond: (fc.category_id = c.category_id)
23	-> Seq Scan on film_category fc (cost=0.00..16.00 rows=1000 width=8) (actual time=0.0...
24	-> Hash (cost=1.28..1.28 rows=6 width=36) (actual time=0.015..0.017 rows=6 loops=1)
25	Buckets: 1024 Batches: 1 Memory Usage: 9kB
26	-> Seq Scan on category c (cost=0.00..1.28 rows=6 width=36) (actual time=0.009..0....
27	Filter: (name = ANY ('{Animation,Children,Classics,Comedy,Family,Music}'::text[]))
28	Rows Removed by Filter: 10
29	Planning Time: 1.546 ms
30	Execution Time: 15.570 ms

Latency and tps after indexes:



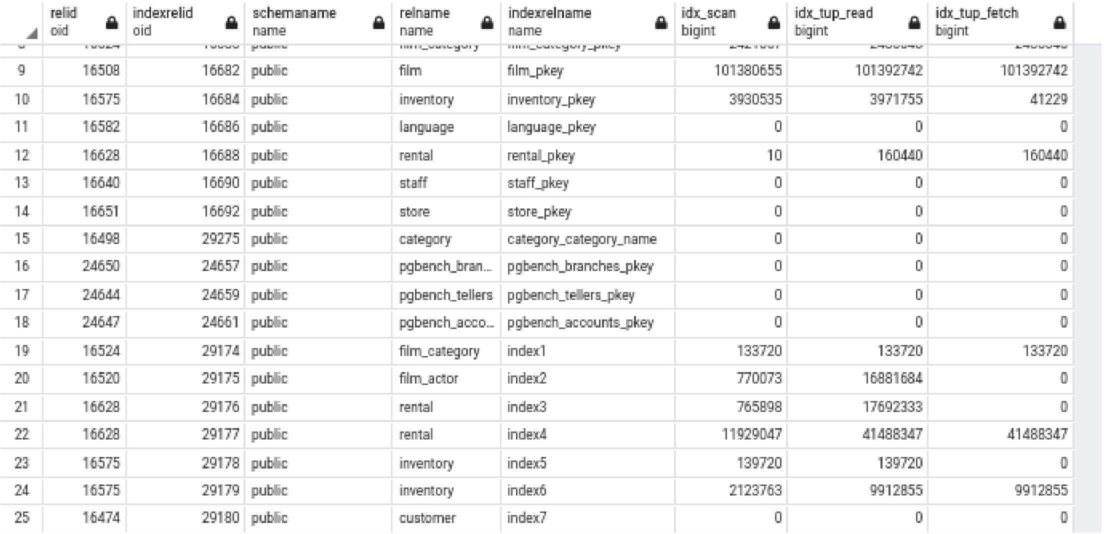
```
[vm@archlinux Desktop]$ pgbench -f individualquery11.sql -U vm -t 1000 project
starting vacuum...end.
transaction type: individualquery11.sql
scaling factor: 1
query mode: simple
number of clients: 1
number of threads: 1
number of transactions per client: 1000
number of transactions actually processed: 1000/1000
latency average = 10.380 ms
tps = 96.339485 (including connections establishing)
tps = 96.370187 (excluding connections establishing)
```

Query plantime and execution time after indexes:



	QUERY PLAN text
14	-> Hash (cost=97.86..97.86 rows=375 width=55) (actual time=0.778..0.783 rows=361 loops=1)
15	Buckets: 1024 Batches: 1 Memory Usage: 31kB
16	-> Hash Join (cost=25.36..97.86 rows=375 width=55) (actual time=0.330..0.699 rows=361 loops=1)
17	Hash Cond: (f.film_id = fc.film_id)
18	-> Seq Scan on film f (cost=0.00..65.00 rows=1000 width=19) (actual time=0.002..0.250 rows=10...
19	-> Hash (cost=20.67..20.67 rows=375 width=36) (actual time=0.302..0.305 rows=361 loops=1)
20	Buckets: 1024 Batches: 1 Memory Usage: 24kB
21	-> Hash Join (cost=1.35..20.67 rows=375 width=36) (actual time=0.127..0.263 rows=361 loop...
22	Hash Cond: (fc.category_id = c.category_id)
23	-> Seq Scan on film_category fc (cost=0.00..16.00 rows=1000 width=8) (actual time=0.006...
24	-> Hash (cost=1.28..1.28 rows=6 width=36) (actual time=0.116..0.117 rows=6 loops=1)
25	Buckets: 1024 Batches: 1 Memory Usage: 9kB
26	-> Seq Scan on category c (cost=0.00..1.28 rows=6 width=36) (actual time=0.006..0.11...
27	Filter: (name = ANY ('{Animation,Children,Classics,Comedy,Family,Music}'::text[]))
28	Rows Removed by Filter: 10
29	Planning Time: 0.786 ms
30	Execution Time: 13.898 ms

Index category_category_name usage and the indexes used for the whole database:



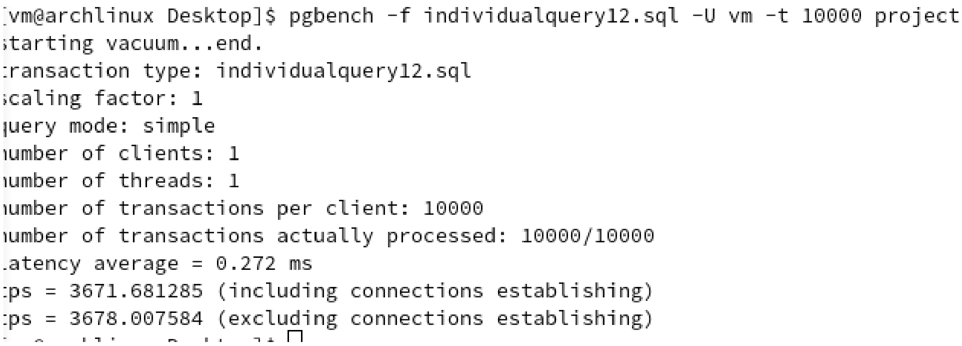
	relid oid	indexrelid oid	schemaname name	relname name	indexrelname name	idx_scan bigint	idx_tup_read bigint	idx_tup_fetch bigint
9	16508	16682	public	film	film_pkey	101380655	101392742	101392742
10	16575	16684	public	inventory	inventory_pkey	3930535	3971755	41229
11	16582	16686	public	language	language_pkey	0	0	0
12	16628	16688	public	rental	rental_pkey	10	160440	160440
13	16640	16690	public	staff	staff_pkey	0	0	0
14	16651	16692	public	store	store_pkey	0	0	0
15	16498	29275	public	category	category_category_name	0	0	0
16	24650	24657	public	pgbench_bran...	pgbench_branches_pkey	0	0	0
17	24644	24659	public	pgbench_tellers	pgbench_tellers_pkey	0	0	0
18	24647	24661	public	pgbench_acco...	pgbench_accounts_pkey	0	0	0
19	16524	29174	public	film_category	index1	133720	133720	133720
20	16520	29175	public	film_actor	index2	770073	16881684	0
21	16628	29176	public	rental	index3	765898	17692333	0
22	16628	29177	public	rental	index4	11929047	41488347	41488347
23	16575	29178	public	inventory	index5	139720	139720	0
24	16575	29179	public	inventory	index6	2123763	9912855	9912855
25	16474	29180	public	customer	index7	0	0	0

QUERY12:

CREATE INDEX actor_last_name_tree on actor USING BTREE (last_name) ;

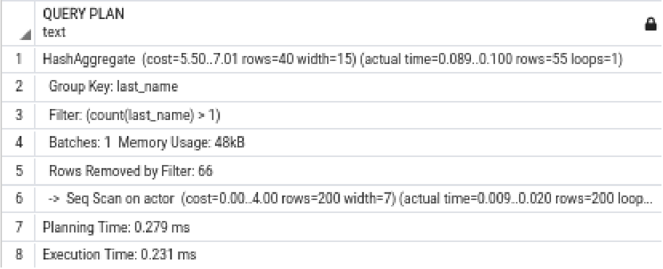
I tried to put an index on last_name.actor so the planner can count last_name of specific value together as the values will be next to each other in the leaf,however it didnt work

Latency and tps before indexes:



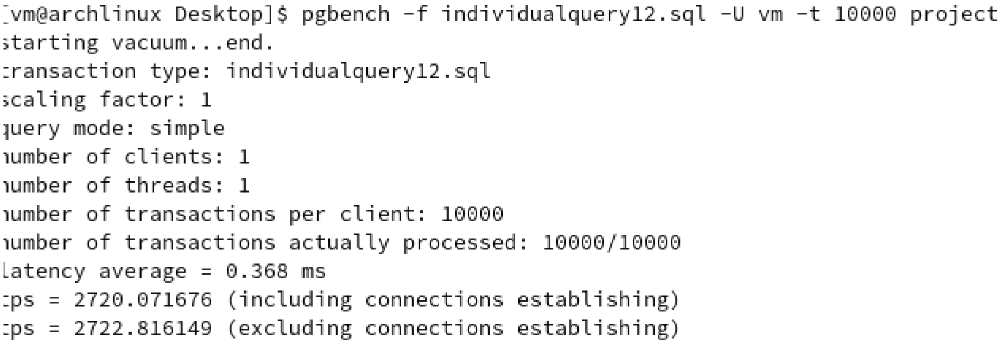
```
[vm@archlinux Desktop]$ pgbench -f individualquery12.sql -U vm -t 10000 project
starting vacuum...end.
transaction type: individualquery12.sql
scaling factor: 1
query mode: simple
number of clients: 1
number of threads: 1
number of transactions per client: 10000
number of transactions actually processed: 10000/10000
latency average = 0.272 ms
tps = 3671.681285 (including connections establishing)
tps = 3678.007584 (excluding connections establishing)
```

Query plantime and execution time before indexes:



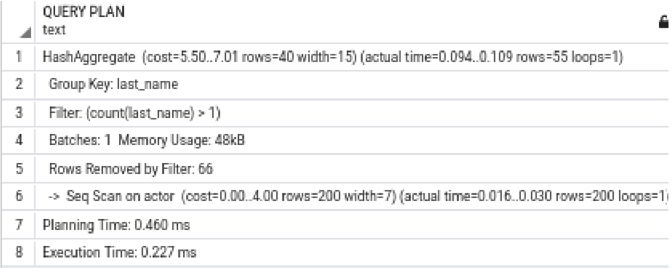
	QUERY PLAN text
1	HashAggregate (cost=5.50..7.01 rows=40 width=15) (actual time=0.089..0.100 rows=55 loops=1)
2	Group Key: last_name
3	Filter: (count(last_name) > 1)
4	Batches: 1 Memory Usage: 48kB
5	Rows Removed by Filter: 66
6	-> Seq Scan on actor (cost=0.00..4.00 rows=200 width=7) (actual time=0.009..0.020 rows=200 loop...
7	Planning Time: 0.279 ms
8	Execution Time: 0.231 ms

Latency and tps after indexes:



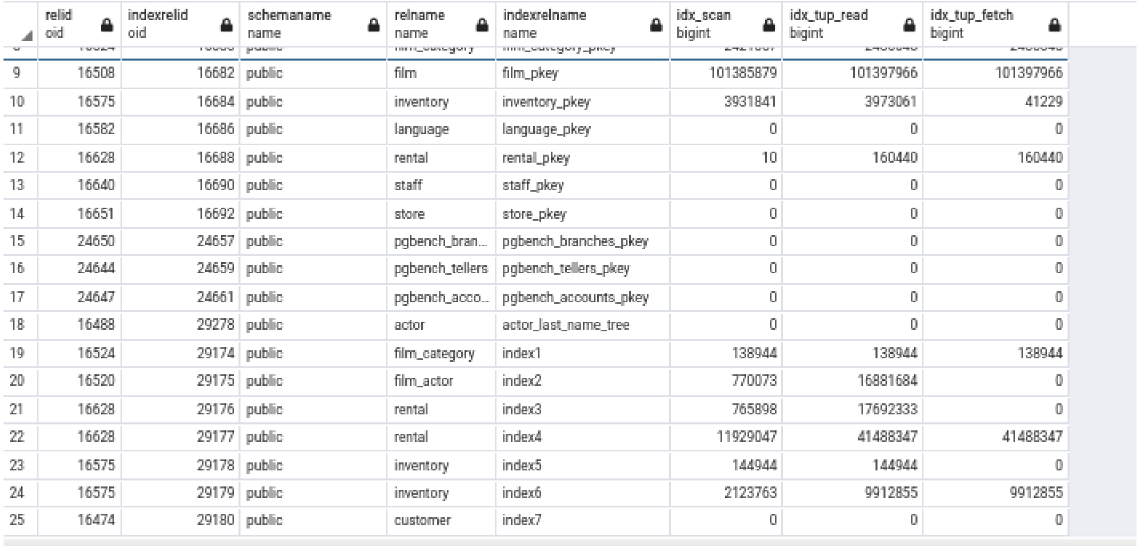
```
[vm@archlinux Desktop]$ pgbench -f individualquery12.sql -U vm -t 10000 project
starting vacuum...end.
transaction type: individualquery12.sql
scaling factor: 1
query mode: simple
number of clients: 1
number of threads: 1
number of transactions per client: 10000
number of transactions actually processed: 10000/10000
latency average = 0.368 ms
tps = 2720.071676 (including connections establishing)
tps = 2722.816149 (excluding connections establishing)
```

Query plantime and execution time after indexes:



	QUERY PLAN text
1	HashAggregate (cost=5.50..7.01 rows=40 width=15) (actual time=0.094..0.109 rows=55 loops=1)
2	Group Key: last_name
3	Filter: (count(last_name) > 1)
4	Batches: 1 Memory Usage: 48kB
5	Rows Removed by Filter: 66
6	-> Seq Scan on actor (cost=0.00..4.00 rows=200 width=7) (actual time=0.016..0.030 rows=200 loops=1)
7	Planning Time: 0.460 ms
8	Execution Time: 0.227 ms

Index actor_last_name_tree usage and the indexes used for the whole database:



	relid oid	indexrelid oid	schemaname name	relname name	indexrelname name	idx_scan bigint	idx_tup_read bigint	idx_tup_fetch bigint
9	16508	16682	public	film	film_pkey	101385879	101397966	101397966
10	16575	16684	public	inventory	inventory_pkey	3931841	3973061	41229
11	16582	16686	public	language	language_pkey	0	0	0
12	16628	16688	public	rental	rental_pkey	10	160440	160440
13	16640	16690	public	staff	staff_pkey	0	0	0
14	16651	16692	public	store	store_pkey	0	0	0
15	24650	24657	public	pgbench_bran...	pgbench_branches_pkey	0	0	0
16	24644	24659	public	pgbench_tellers	pgbench_tellers_pkey	0	0	0
17	24647	24661	public	pgbench_acco...	pgbench_accounts_pkey	0	0	0
18	16488	29278	public	actor	actor_last_name_tree	0	0	0
19	16524	29174	public	film_category	index1	138944	138944	138944
20	16520	29175	public	film_actor	index2	770073	16881684	0
21	16628	29176	public	rental	index3	765898	17692333	0
22	16628	29177	public	rental	index4	11929047	41488347	41488347
23	16575	29178	public	inventory	index5	144944	144944	0
24	16575	29179	public	inventory	index6	2123763	9912855	9912855
25	16474	29180	public	customer	index7	0	0	0

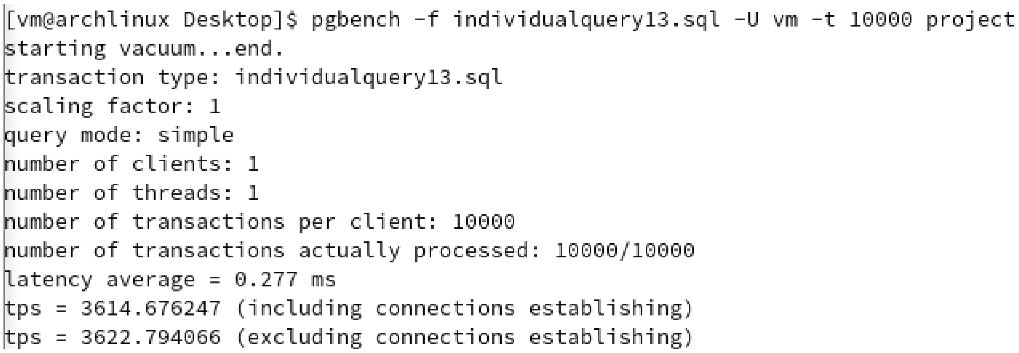
Query13:

This query used Index5 that also used in optmizing the database in general

CREATE INDEX film_title_hash on film USING HASH (title);

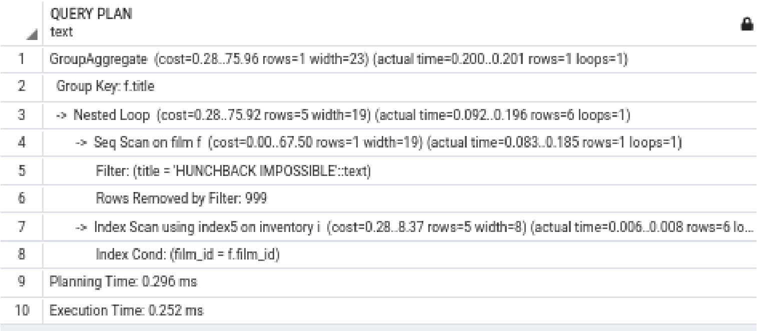
Film_title_hash index let us access records that has title ‘HUNCHBACK IMPOSSIBILE’ in 0(1)

Latency and tps before indexes:



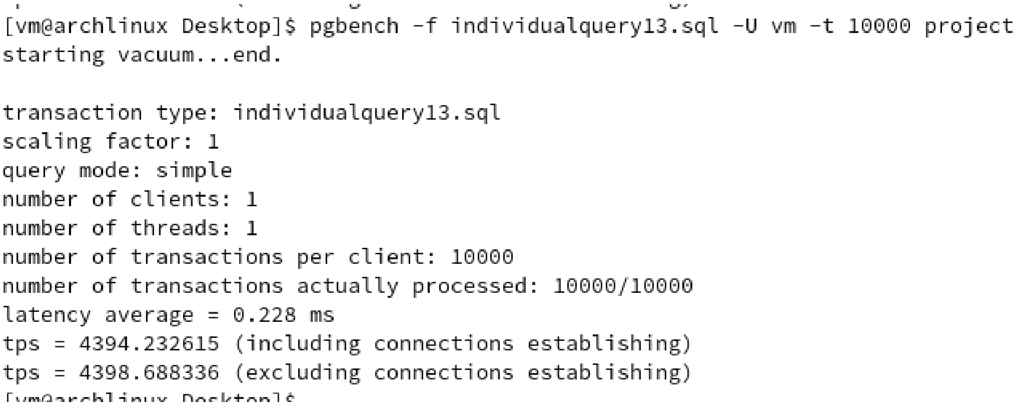
```
[vm@archlinux Desktop]$ pgbench -f individualquery13.sql -U vm -t 10000 project
starting vacuum...end.
transaction type: individualquery13.sql
scaling factor: 1
query mode: simple
number of clients: 1
number of threads: 1
number of transactions per client: 10000
number of transactions actually processed: 10000/10000
latency average = 0.277 ms
tps = 3614.676247 (including connections establishing)
tps = 3622.794066 (excluding connections establishing)
```

Query plantime and execution time before indexes:



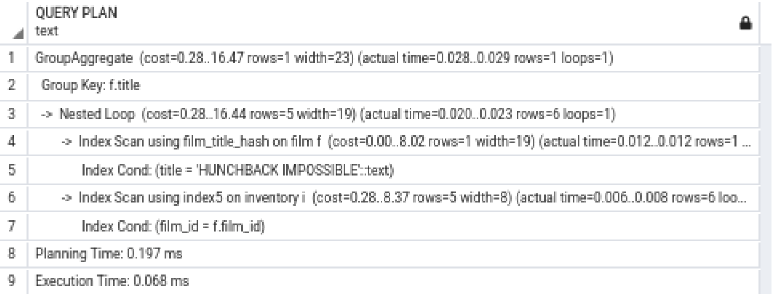
	QUERY PLAN text
1	GroupAggregate (cost=0.28..75.96 rows=1 width=23) (actual time=0.200..0.201 rows=1 loops=1)
2	Group Key: f.title
3	-> Nested Loop (cost=0.28..75.92 rows=5 width=19) (actual time=0.092..0.196 rows=6 loops=1)
4	-> Seq Scan on film f (cost=0.00..67.50 rows=1 width=19) (actual time=0.083..0.185 rows=1 loops=1)
5	Filter: (title = 'HUNCHBACK IMPOSSIBLE'::text)
6	Rows Removed by Filter: 999
7	-> Index Scan using index5 on inventory i (cost=0.28..8.37 rows=5 width=8) (actual time=0.006..0.008 rows=6 lo...
8	Index Cond: (film_id = f.film_id)
9	Planning Time: 0.296 ms
10	Execution Time: 0.252 ms

Latency and tps after indexes:



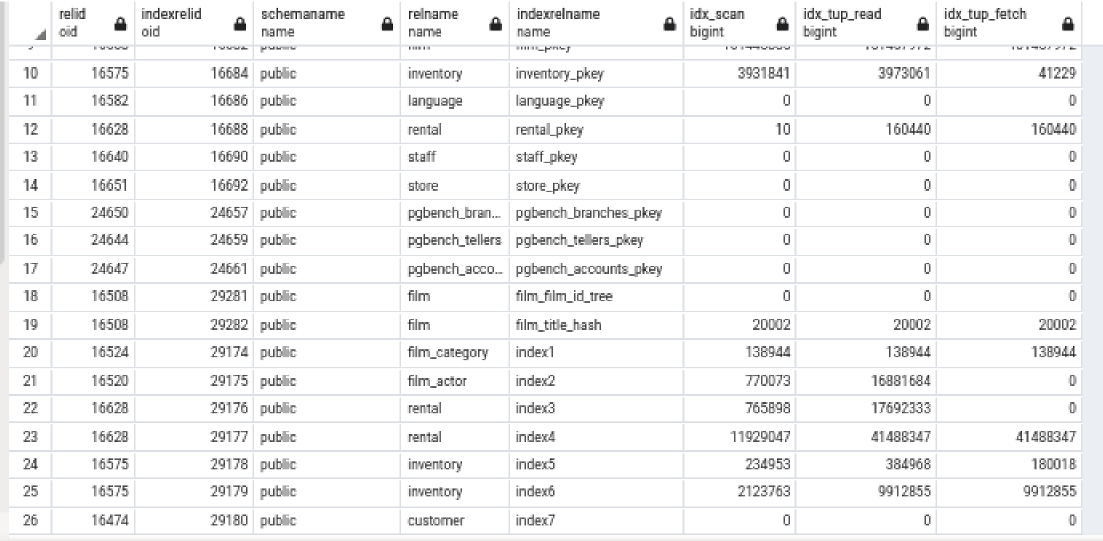
```
[vm@archlinux Desktop]$ pgbench -f individualquery13.sql -U vm -t 10000 project
starting vacuum...end.

transaction type: individualquery13.sql
scaling factor: 1
query mode: simple
number of clients: 1
number of threads: 1
number of transactions per client: 10000
number of transactions actually processed: 10000/10000
latency average = 0.228 ms
tps = 4394.232615 (including connections establishing)
tps = 4398.688336 (excluding connections establishing)
[vm@archlinux Desktop]$
```

Query plantime and execution time after indexes:



	QUERY PLAN text
1	GroupAggregate (cost=0.28..16.47 rows=1 width=23) (actual time=0.028..0.029 rows=1 loops=1)
2	Group Key: f.title
3	-> Nested Loop (cost=0.28..16.44 rows=5 width=19) (actual time=0.020..0.023 rows=6 loops=1)
4	-> Index Scan using film_title_hash on film f (cost=0.00..8.02 rows=1 width=19) (actual time=0.012..0.012 rows=1 ...
5	Index Cond: (title = 'HUNCHBACK IMPOSSIBLE'::text)
6	-> Index Scan using index5 on inventory i (cost=0.28..8.37 rows=5 width=8) (actual time=0.006..0.008 rows=6 loo...
7	Index Cond: (film_id = f.film_id)
8	Planning Time: 0.197 ms
9	Execution Time: 0.068 ms

Index film_title_hash usage and the indexes used for the whole database:

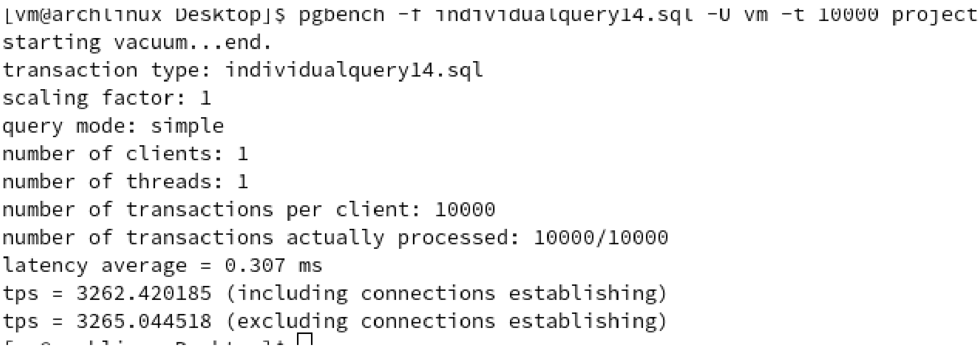


	relid oid	indexrelid oid	schemaname name	relname name	indexrelname name	idx_scan bigint	idx_tup_read bigint	idx_tup_fetch bigint
10	16575	16684	public	inventory	inventory_pkey	3931841	3973061	41229
11	16582	16686	public	language	language_pkey	0	0	0
12	16628	16688	public	rental	rental_pkey	10	160440	160440
13	16640	16690	public	staff	staff_pkey	0	0	0
14	16651	16692	public	store	store_pkey	0	0	0
15	24650	24657	public	pgbench_bran...	pgbench_branches_pkey	0	0	0
16	24644	24659	public	pgbench_tellers	pgbench_tellers_pkey	0	0	0
17	24647	24661	public	pgbench_acco...	pgbench_accounts_pkey	0	0	0
18	16508	29281	public	film	film_film_id_tree	0	0	0
19	16508	29282	public	film	film_title_hash	20002	20002	20002
20	16524	29174	public	film_category	index1	138944	138944	138944
21	16520	29175	public	film_actor	index2	770073	16881684	0
22	16628	29176	public	rental	index3	765898	17692333	0
23	16628	29177	public	rental	index4	11929047	41488347	41488347
24	16575	29178	public	inventory	index5	234953	384968	180018
25	16575	29179	public	inventory	index6	2123763	9912855	9912855
26	16474	29180	public	customer	index7	0	0	0

Query14:

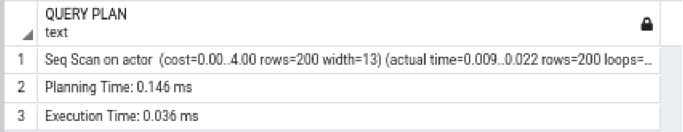
I couldnt make an index to optimize this query as the planner dont want to avoid any record,so it will simply fetch all the data sequently.

Latency and tps before indexes:



```
[vm@archlinux Desktop]$ pgbench -f individualquery14.sql -U vm -t 10000 project
starting vacuum...end.
transaction type: individualquery14.sql
scaling factor: 1
query mode: simple
number of clients: 1
number of threads: 1
number of transactions per client: 10000
number of transactions actually processed: 10000/10000
latency average = 0.307 ms
tps = 3262.420185 (including connections establishing)
tps = 3265.044518 (excluding connections establishing)
```

Query plantime and execution time before indexes:



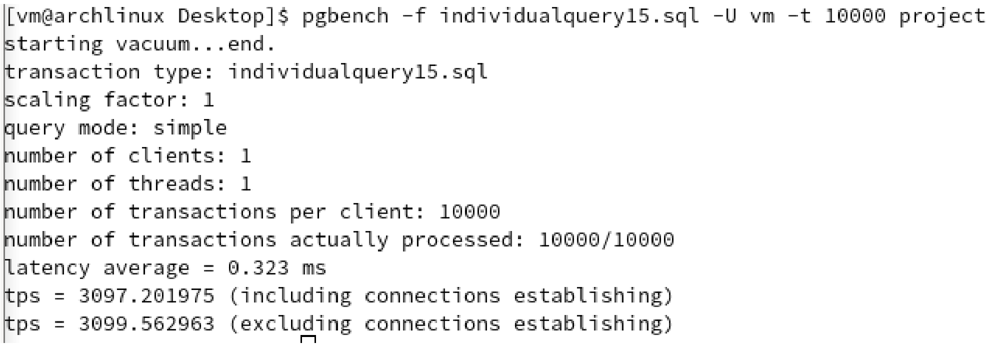
	QUERY PLAN text
1	Seq Scan on actor (cost=0.00..4.00 rows=200 width=13) (actual time=0.009..0.022 rows=200 loops=...
2	Planning Time: 0.146 ms
3	Execution Time: 0.036 ms

Query15:

CREATE INDEX customer_active_tree on customer USING BTREE (active);

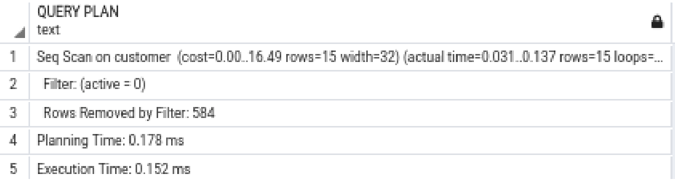
Since there is only 15 records that has active value equal 0 so it is very powerful using BTREE index on active column

Latency and tps before indexes:



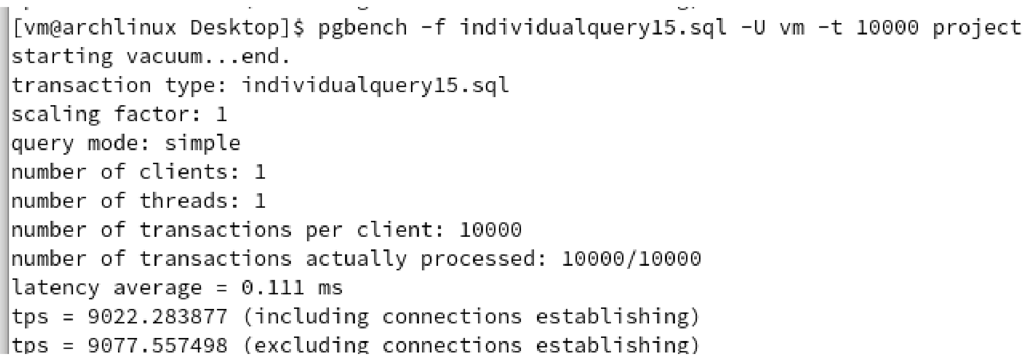
```
[vm@archlinux Desktop]$ pgbench -f individualquery15.sql -U vm -t 10000 project
starting vacuum...end.
transaction type: individualquery15.sql
scaling factor: 1
query mode: simple
number of clients: 1
number of threads: 1
number of transactions per client: 10000
number of transactions actually processed: 10000/10000
latency average = 0.323 ms
tps = 3097.201975 (including connections establishing)
tps = 3099.562963 (excluding connections establishing)
```

Query plantime and execution time before indexes:



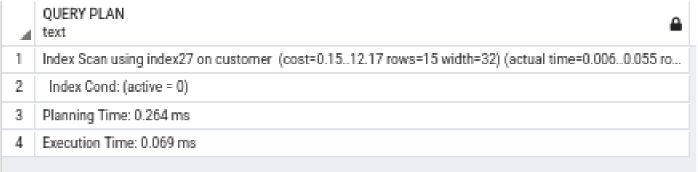
	QUERY PLAN text
1	Seq Scan on customer (cost=0.00..16.49 rows=15 width=32) (actual time=0.031..0.137 rows=15 loops=...
2	Filter: (active = 0)
3	Rows Removed by Filter: 584
4	Planning Time: 0.178 ms
5	Execution Time: 0.152 ms

Latency and tps after indexes:



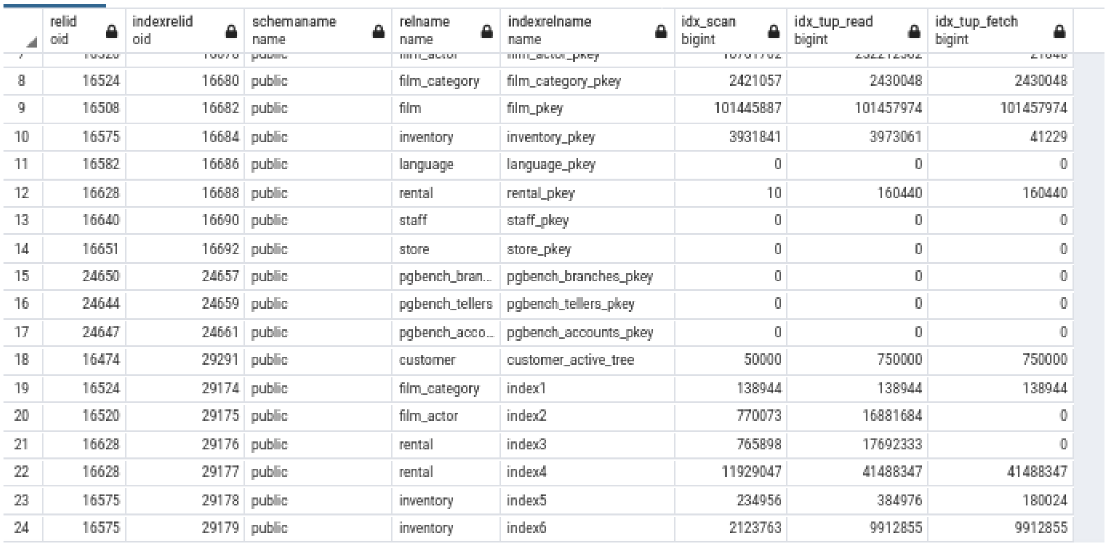
```
[vm@archlinux Desktop]$ pgbench -f individualquery15.sql -U vm -t 10000 project
starting vacuum...end.
transaction type: individualquery15.sql
scaling factor: 1
query mode: simple
number of clients: 1
number of threads: 1
number of transactions per client: 10000
number of transactions actually processed: 10000/10000
latency average = 0.111 ms
tps = 9022.283877 (including connections establishing)
tps = 9077.557498 (excluding connections establishing)
```

Query plantime and execution time after indexes:



	QUERY PLAN text
1	Index Scan using index27 on customer (cost=0.15..12.17 rows=15 width=32) (actual time=0.006..0.055 ro...
2	Index Cond: (active = 0)
3	Planning Time: 0.264 ms
4	Execution Time: 0.069 ms

Index customer_active_tree usage and the indexes used for the whole database:



	relid oid	indexrelid oid	schemaname name	relname name	indexrelname name	idx_scan bigint	idx_tup_read bigint	idx_tup_fetch bigint
7	[illegible]	[illegible]	public	film_actor	film_actor_pkey	[illegible]	[illegible]	[illegible]
8	16524	16680	public	film_category	film_category_pkey	2421057	2430048	2430048
9	16508	16682	public	film	film_pkey	101445887	101457974	101457974
10	16575	16684	public	inventory	inventory_pkey	3931841	3973061	41229
11	16582	16686	public	language	language_pkey	0	0	0
12	16628	16688	public	rental	rental_pkey	10	160440	160440
13	16640	16690	public	staff	staff_pkey	0	0	0
14	16651	16692	public	store	store_pkey	0	0	0
15	24650	24657	public	pgbench_bran...	pgbench_branches_pkey	0	0	0
16	24644	24659	public	pgbench_tellers	pgbench_tellers_pkey	0	0	0
17	24647	24661	public	pgbench_acco...	pgbench_accounts_pkey	0	0	0
18	16474	29291	public	customer	customer_active_tree	50000	750000	750000
19	16524	29174	public	film_category	index1	138944	138944	138944
20	16520	29175	public	film_actor	index2	770073	16881684	0
21	16628	29176	public	rental	index3	765898	17692333	0
22	16628	29177	public	rental	index4	11929047	41488347	41488347
23	16575	29178	public	inventory	index5	234956	384976	180024
24	16575	29179	public	inventory	index6	2123763	9912855	9912855

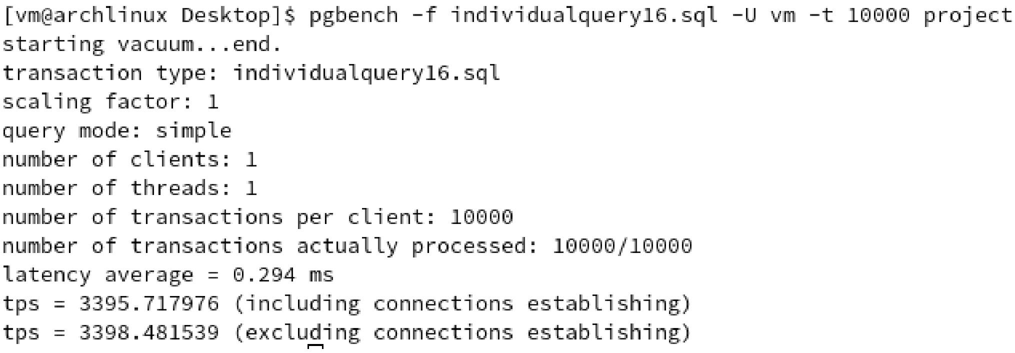
Query16:

CREATE INDEX customer_address_id_tree on customer USING BTREE (address_id);

CREATE INDEX address_district_hash on address USING HASH (district);

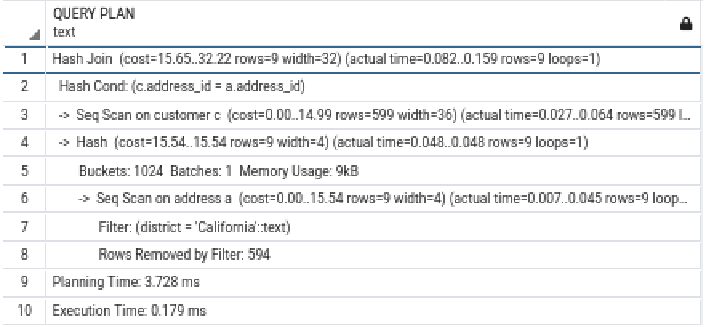
Address_distinct_hash index will let us access records that has district value equal to ‘California’ in O(1) which they are only 9 records,moreover BTREE will let us find any record with it is specifi id in LOG(N) which help us in the join process customer table with address table

Latency and tps before indexes:



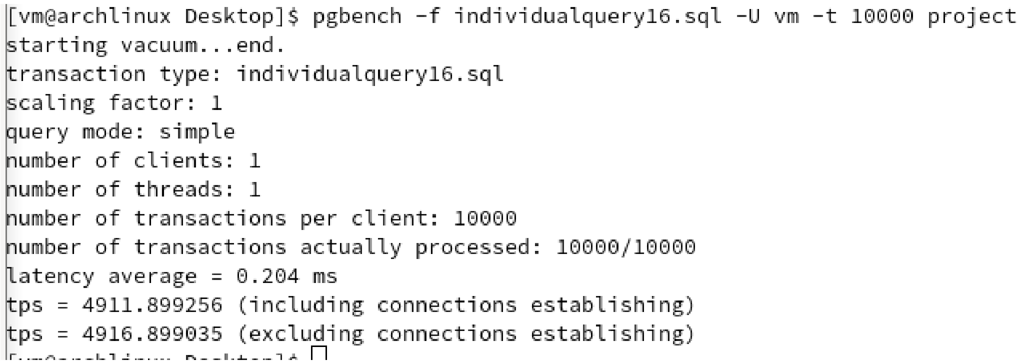
```
[vm@archlinux Desktop]$ pgbench -f individualquery16.sql -U vm -t 10000 project
starting vacuum...end.
transaction type: individualquery16.sql
scaling factor: 1
query mode: simple
number of clients: 1
number of threads: 1
number of transactions per client: 10000
number of transactions actually processed: 10000/10000
latency average = 0.294 ms
tps = 3395.717976 (including connections establishing)
tps = 3398.481539 (excluding connections establishing)
```

Query plantime and execution time before indexes:



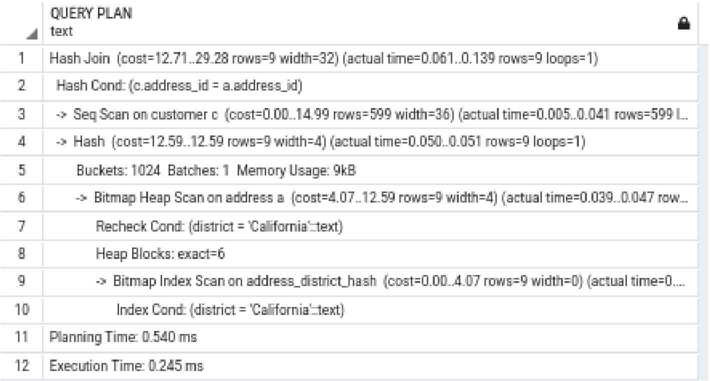
	QUERY PLAN text
1	Hash Join (cost=15.65..32.22 rows=9 width=32) (actual time=0.082..0.159 rows=9 loops=1)
2	Hash Cond: (c.address_id = a.address_id)
3	-> Seq Scan on customer c (cost=0.00..14.99 rows=599 width=36) (actual time=0.027..0.064 rows=599 l...
4	-> Hash (cost=15.54..15.54 rows=9 width=4) (actual time=0.048..0.048 rows=9 loops=1)
5	Buckets: 1024 Batches: 1 Memory Usage: 9kB
6	-> Seq Scan on address a (cost=0.00..15.54 rows=9 width=4) (actual time=0.007..0.045 rows=9 loop...
7	Filter: (district = 'California'::text)
8	Rows Removed by Filter: 594
9	Planning Time: 3.728 ms
10	Execution Time: 0.179 ms

Latency and tps after indexes:



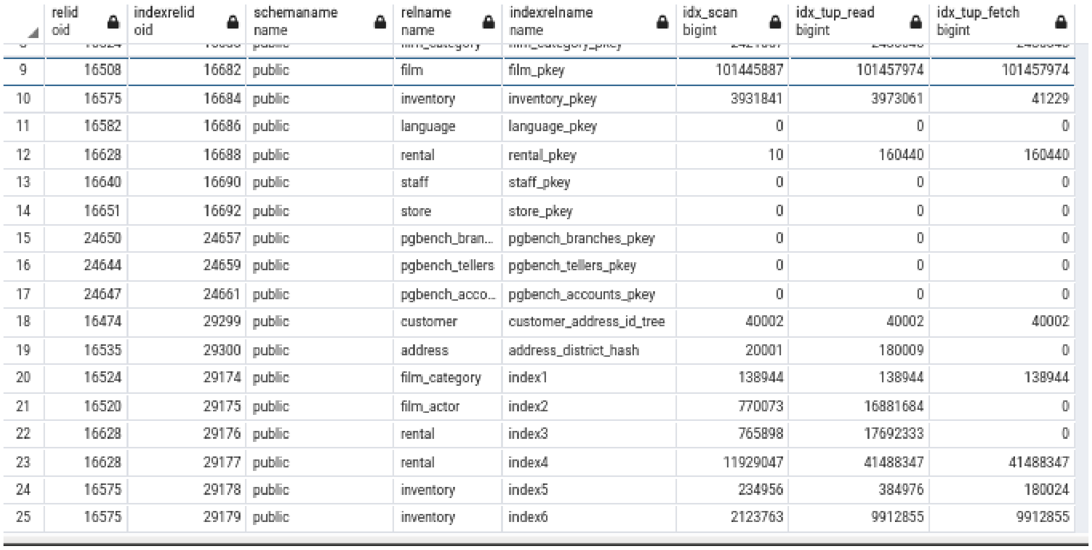
```
[vm@archlinux Desktop]$ pgbench -f individualquery16.sql -U vm -t 10000 project
starting vacuum...end.
transaction type: individualquery16.sql
scaling factor: 1
query mode: simple
number of clients: 1
number of threads: 1
number of transactions per client: 10000
number of transactions actually processed: 10000/10000
latency average = 0.204 ms
tps = 4911.899256 (including connections establishing)
tps = 4916.899035 (excluding connections establishing)
[vm@archlinux Desktop]$
```

Query plantime and execution time after indexes:



	QUERY PLAN text
1	Hash Join (cost=12.71..29.28 rows=9 width=32) (actual time=0.061..0.139 rows=9 loops=1)
2	Hash Cond: (c.address_id = a.address_id)
3	-> Seq Scan on customer c (cost=0.00..14.99 rows=599 width=36) (actual time=0.005..0.041 rows=599 l...
4	-> Hash (cost=12.59..12.59 rows=9 width=4) (actual time=0.050..0.051 rows=9 loops=1)
5	Buckets: 1024 Batches: 1 Memory Usage: 9kB
6	-> Bitmap Heap Scan on address a (cost=4.07..12.59 rows=9 width=4) (actual time=0.039..0.047 row...
7	Recheck Cond: (district = 'California'::text)
8	Heap Blocks: exact=6
9	-> Bitmap Index Scan on address_district_hash (cost=0.00..4.07 rows=9 width=0) (actual time=0....
10	Index Cond: (district = 'California'::text)
11	Planning Time: 0.540 ms
12	Execution Time: 0.245 ms

Index customer_address_id_tree and address_district_hash usage and the indexes used for the whole database:



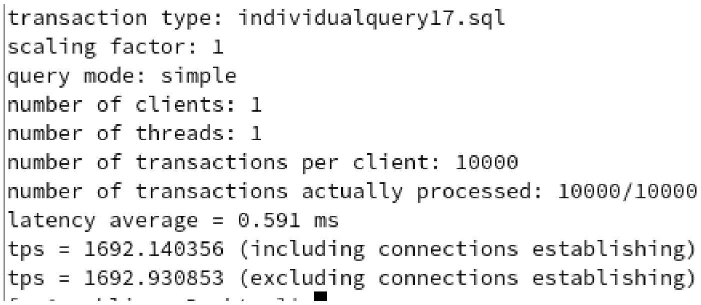
	relid oid	indexrelid oid	schemaname name	relname name	indexrelname name	idx_scan bigint	idx_tup_read bigint	idx_tup_fetch bigint
9	16508	16682	public	film	film_pkey	101445887	101457974	101457974
10	16575	16684	public	inventory	inventory_pkey	3931841	3973061	41229
11	16582	16686	public	language	language_pkey	0	0	0
12	16628	16688	public	rental	rental_pkey	10	160440	160440
13	16640	16690	public	staff	staff_pkey	0	0	0
14	16651	16692	public	store	store_pkey	0	0	0
15	24650	24657	public	pgbench_bran...	pgbench_branches_pkey	0	0	0
16	24644	24659	public	pgbench_tellers	pgbench_tellers_pkey	0	0	0
17	24647	24661	public	pgbench_acco...	pgbench_accounts_pkey	0	0	0
18	16474	29299	public	customer	customer_address_id_tree	40002	40002	40002
19	16535	29300	public	address	address_district_hash	20001	180009	0
20	16524	29174	public	film_category	index1	138944	138944	138944
21	16520	29175	public	film_actor	index2	770073	16881684	0
22	16628	29176	public	rental	index3	765898	17692333	0
23	16628	29177	public	rental	index4	11929047	41488347	41488347
24	16575	29178	public	inventory	index5	234956	384976	180024
25	16575	29179	public	inventory	index6	2123763	9912855	9912855

Query17:

CREATE INDEX film_title_tree on film USING BTREE (title) WHERE title LIKE 'K%'OR title LIKE 'Q%';

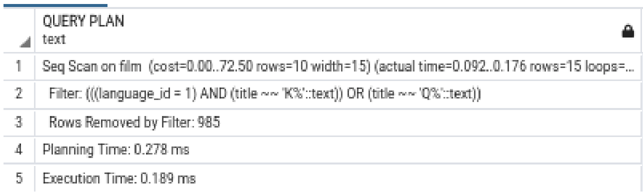
Since there is only 15 record that has titles begin with K OR Q so it is very powerful using partial index

Latency and tps before indexes:



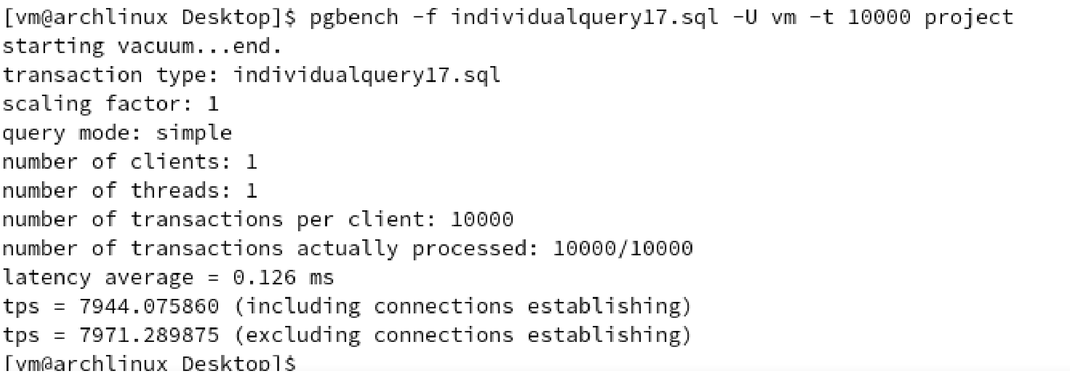
```
transaction type: individualquery17.sql
scaling factor: 1
query mode: simple
number of clients: 1
number of threads: 1
number of transactions per client: 10000
number of transactions actually processed: 10000/10000
latency average = 0.591 ms
tps = 1692.140356 (including connections establishing)
tps = 1692.930853 (excluding connections establishing)
```

Query plantime and execution time before indexes:



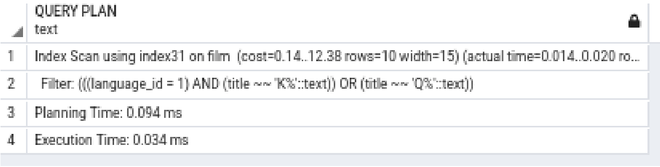
	QUERY PLAN text
1	Seq Scan on film (cost=0.00..72.50 rows=10 width=15) (actual time=0.092..0.176 rows=15 loops=...
2	Filter: (((language_id = 1) AND (title ~~ 'K%'::text)) OR (title ~~ 'Q%'::text))
3	Rows Removed by Filter: 985
4	Planning Time: 0.278 ms
5	Execution Time: 0.189 ms

Latency and tps after indexes:



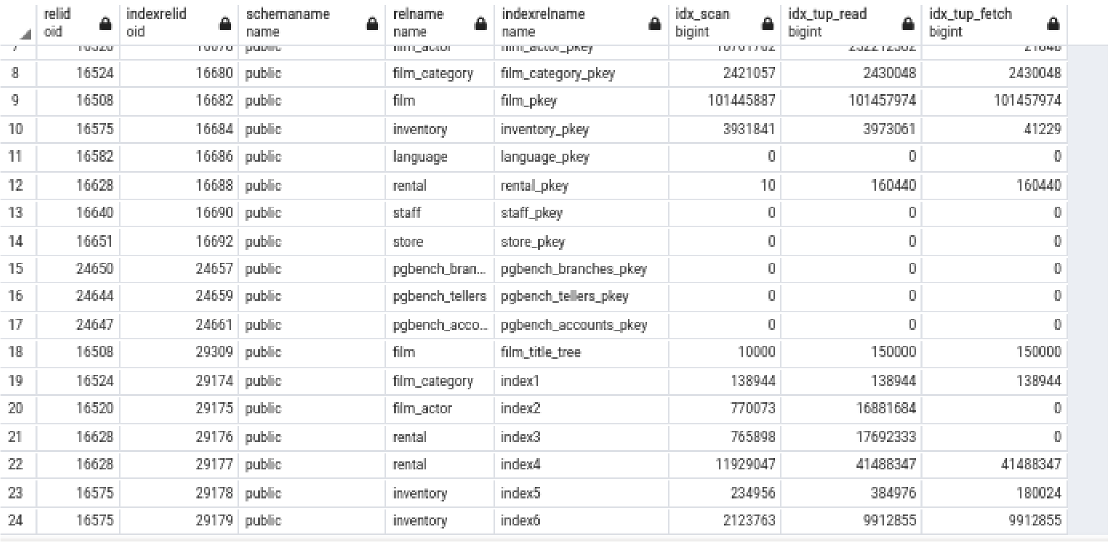
```
[vm@archlinux Desktop]$ pgbench -f individualquery17.sql -U vm -t 10000 project
starting vacuum...end.
transaction type: individualquery17.sql
scaling factor: 1
query mode: simple
number of clients: 1
number of threads: 1
number of transactions per client: 10000
number of transactions actually processed: 10000/10000
latency average = 0.126 ms
tps = 7944.075860 (including connections establishing)
tps = 7971.289875 (excluding connections establishing)
[vm@archlinux Desktop]$
```

Query plantime and execution time after indexes:



	QUERY PLAN text
1	Index Scan using index31 on film (cost=0.14..12.38 rows=10 width=15) (actual time=0.014..0.020 ro...
2	Filter: (((language_id = 1) AND (title ~~ 'K%'::text)) OR (title ~~ 'Q%'::text))
3	Planning Time: 0.094 ms
4	Execution Time: 0.034 ms

Index film_title_tree on film usage and the indexes used for the whole database:



	relid oid	indexrelid oid	schemaname name	relname name	indexrelname name	idx_scan bigint	idx_tup_read bigint	idx_tup_fetch bigint
7	[illegible]	[illegible]	public	film_actor	film_actor_pkey	[illegible]	[illegible]	[illegible]
8	16524	16680	public	film_category	film_category_pkey	2421057	2430048	2430048
9	16508	16682	public	film	film_pkey	101445887	101457974	101457974
10	16575	16684	public	inventory	inventory_pkey	3931841	3973061	41229
11	16582	16686	public	language	language_pkey	0	0	0
12	16628	16688	public	rental	rental_pkey	10	160440	160440
13	16640	16690	public	staff	staff_pkey	0	0	0
14	16651	16692	public	store	store_pkey	0	0	0
15	24650	24657	public	pgbench_bran...	pgbench_branches_pkey	0	0	0
16	24644	24659	public	pgbench_tellers	pgbench_tellers_pkey	0	0	0
17	24647	24661	public	pgbench_acco...	pgbench_accounts_pkey	0	0	0
18	16508	29309	public	film	film_title_tree	10000	150000	150000
19	16524	29174	public	film_category	index1	138944	138944	138944
20	16520	29175	public	film_actor	index2	770073	16881684	0
21	16628	29176	public	rental	index3	765898	17692333	0
22	16628	29177	public	rental	index4	11929047	41488347	41488347
23	16575	29178	public	inventory	index5	234956	384976	180024
24	16575	29179	public	inventory	index6	2123763	9912855	9912855