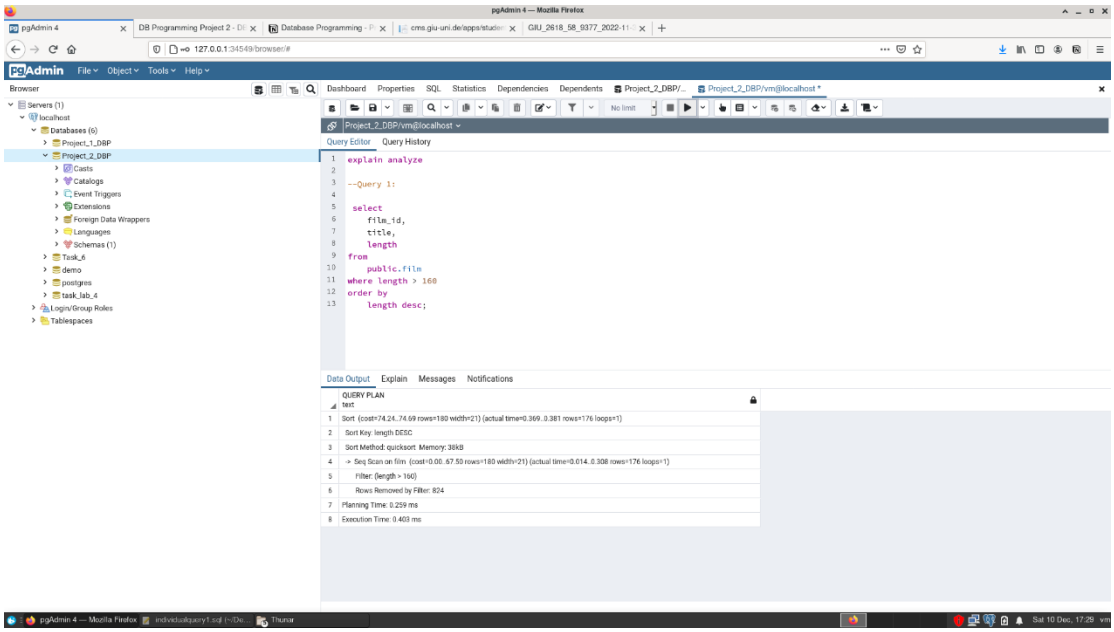
Database Programming

Project 2

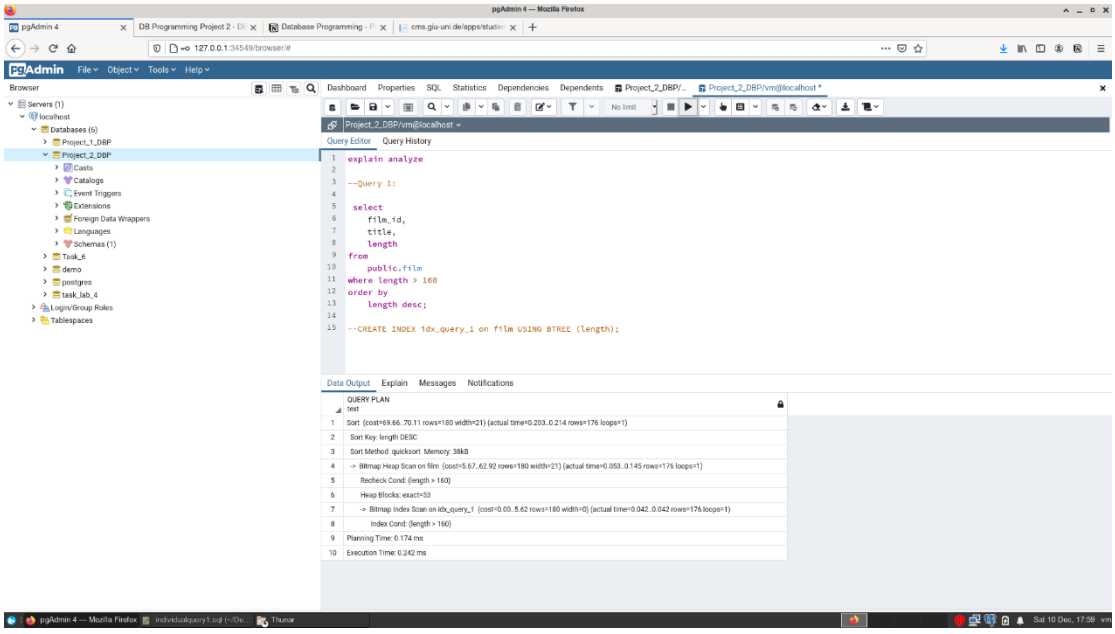
Query 1:

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

Before:



After:



Commands:

The commands we used are:

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

Comparison:

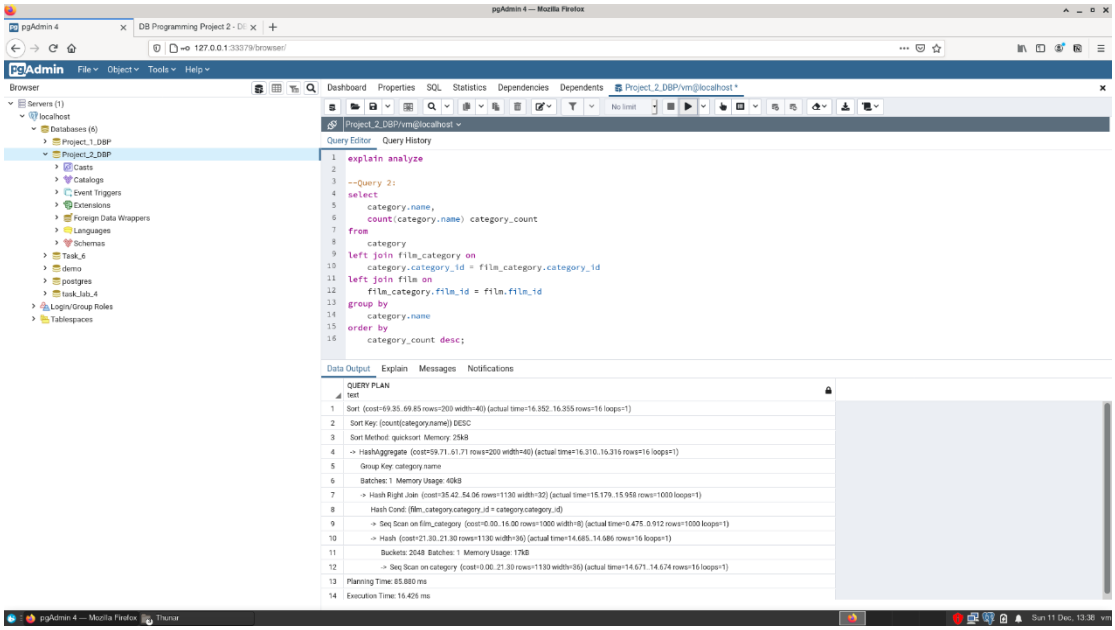
	Before	After
Highest Cost	Sort = 7.19	Bitmap heap scan on film = 57.3
Slowest Runtime	Sort = 0.073 ms	Bitmap heap scan on film = 0.103 ms
Largest Number of Rows	Sort/Seqscan = 180	Sort/Bitmap index and Heap scan = 180
Planning Time	0.259 ms	0.174 ms
Execution Time	0.403 ms	0.242 ms

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

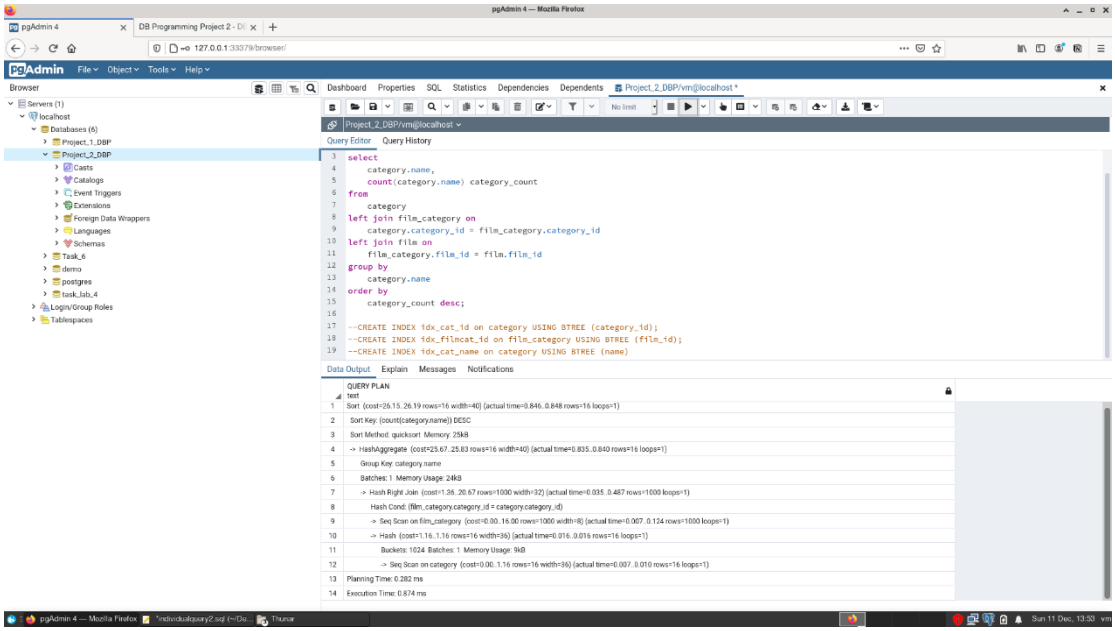
Query 2:

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

Before:



After:



Commands:

The commands we used are:

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

Comparison:

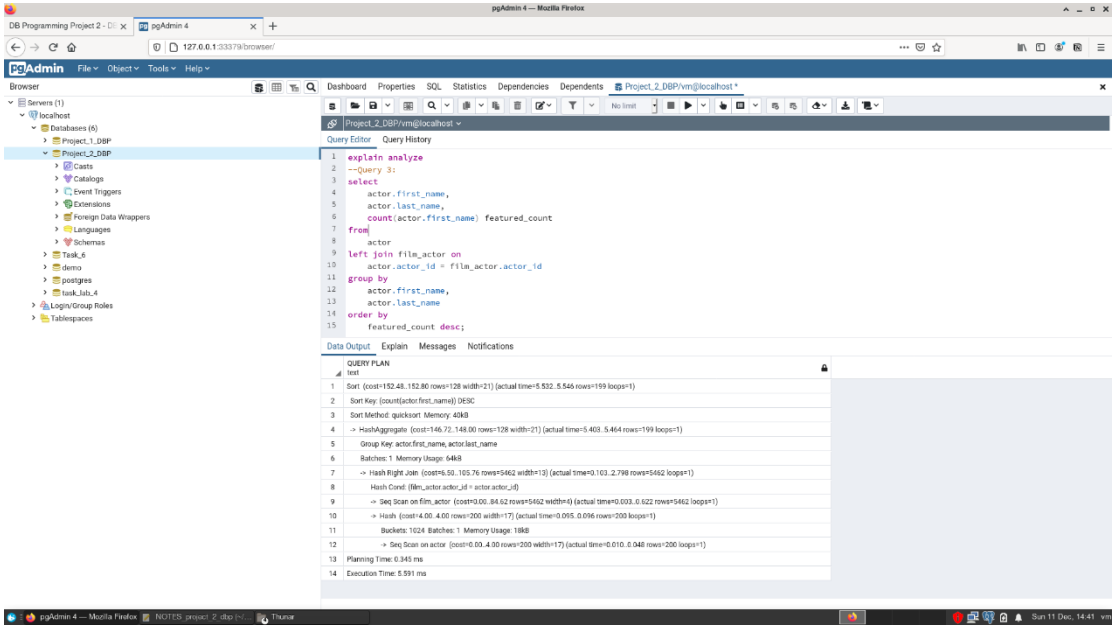
	Before	After
Highest Cost	Hash right join = 7.65	Hash agg = 5.16
Slowest Runtime	Hash right join = 15.046 ms	Hash right join = 0.363 ms
Largest Number of Rows	Hash right join/Hash/seq scan on category = 1130	Hash right join = 1000
Planning Time	85.880 ms	0.282 ms
Execution Time	16.426 ms	0.874 ms

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

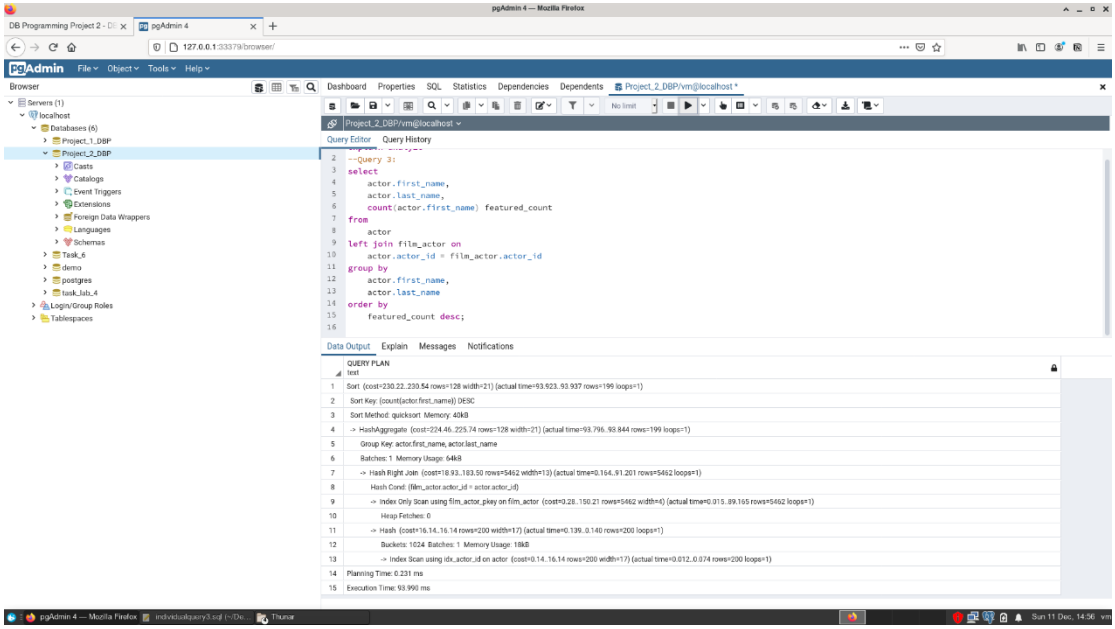
Query 3:

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

Before:



After:



Commands:

The commands we used are:

- CREATE INDEX idx_actor_id on actor USING BTREE (actor_id);
- CREATE INDEX idx_act_frname on actor USING BTREE (first_name);
- CREATE INDEX idx_act_laname on actor USING BTREE (last_name)

Comparison:

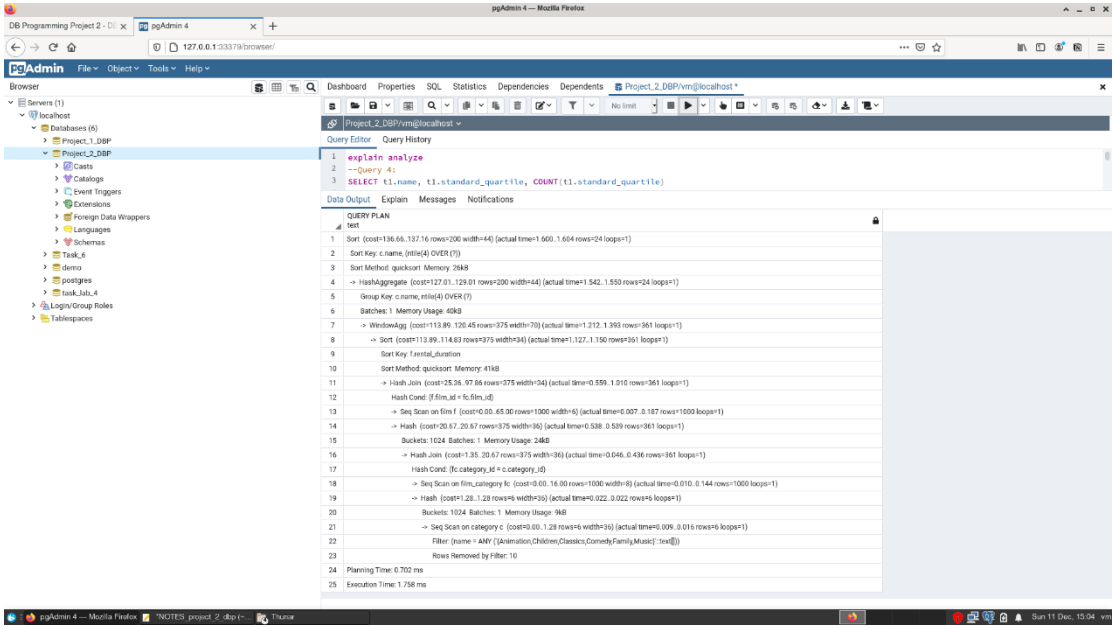
	Before	After
Highest Cost	Hash agg = 42.24	Hash agg = 42.24
Slowest Runtime	Hash agg = 2.666 ms	Hash agg = 2.643 ms
Largest Number of Rows	Hash right join/seq scan on film_actor = 5462	Hash right join/index only scan film_actor = 5462
Planning Time	0.345 ms	0.231 ms
Execution Time	5.591 ms	93.990 ms

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

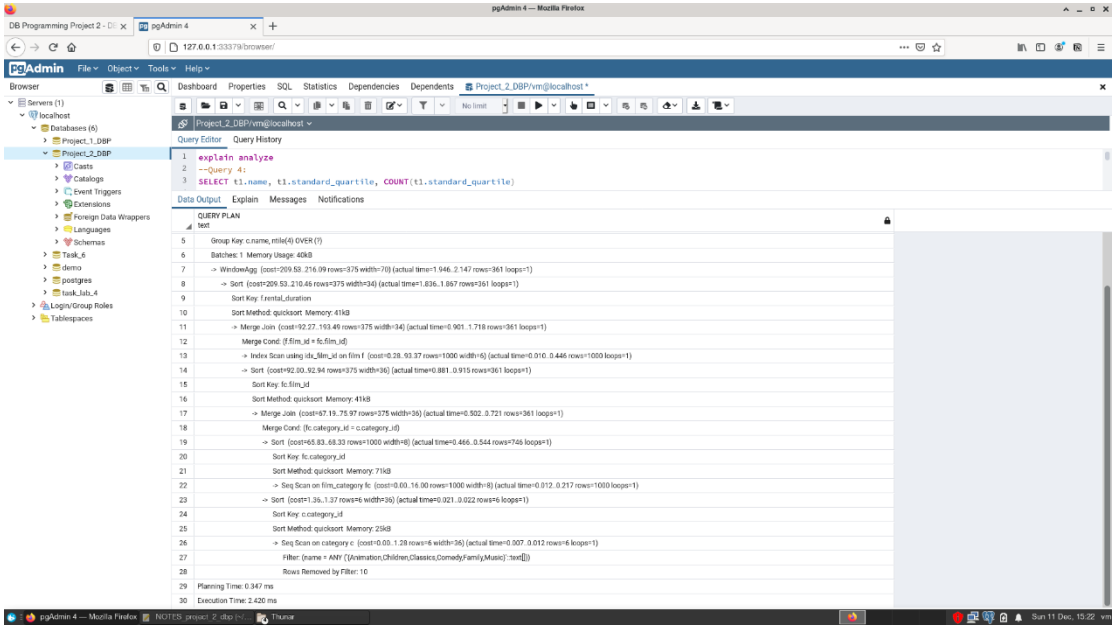
Query 4:

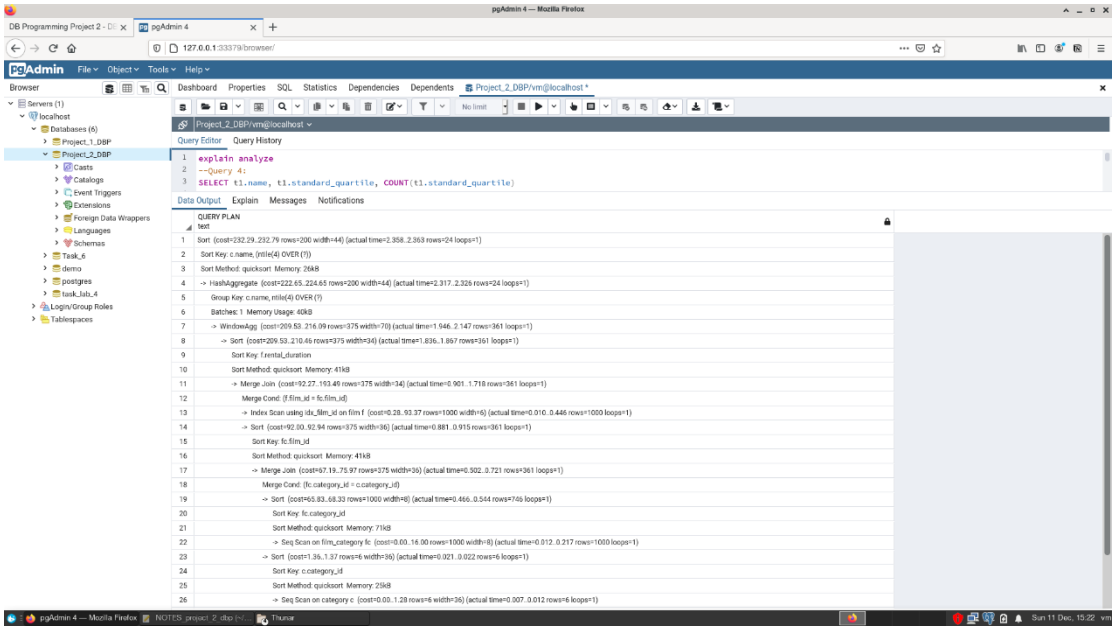
For query 4 we used an index on the (category_id, film_id, name, rental_duration) attributes using BTREE on all four of them. For better optimization of the query and for more flexibility in searching on the attributes that were used in the where and the joins clauses.

Before:



After (2 screenshots):





Commands:

The commands we used are:

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

Comparison:

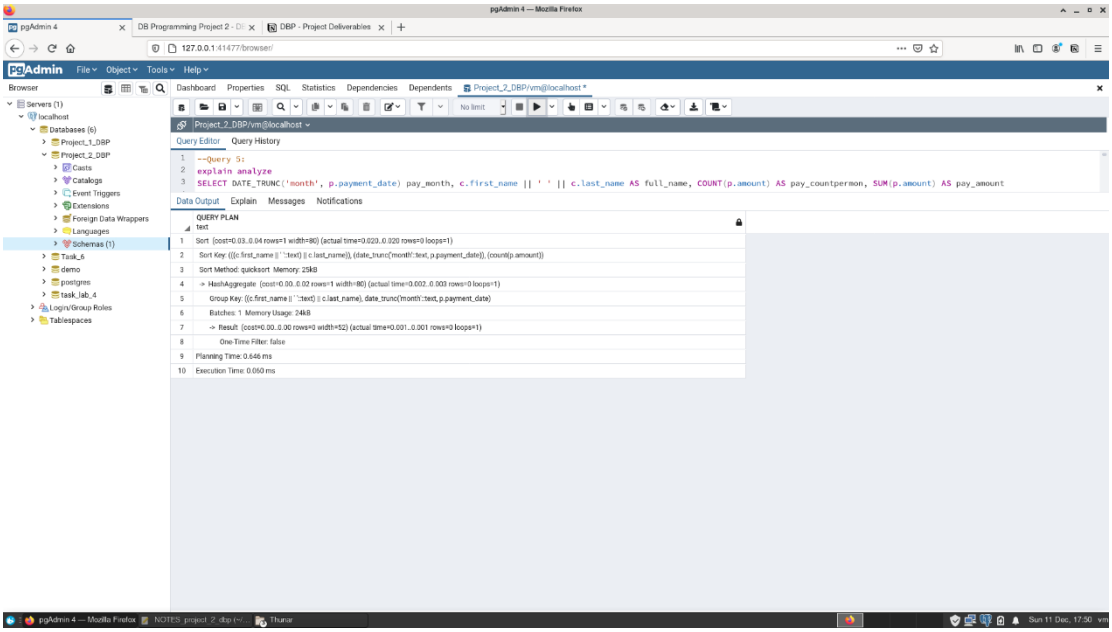
	Before	After
Highest Cost	Sort = 16.14	Sort = 16.97
Slowest Runtime	Hash join = 0.284 ms	Merge join = 0.357 ms
Largest Number of Rows	Seq scan on film f/seq scan on film_category = 1000	Index scan using idx_film_id/ sort/seq scan on film_category = 1000
Planning Time	0.702 ms	0.347 ms
Execution Time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

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

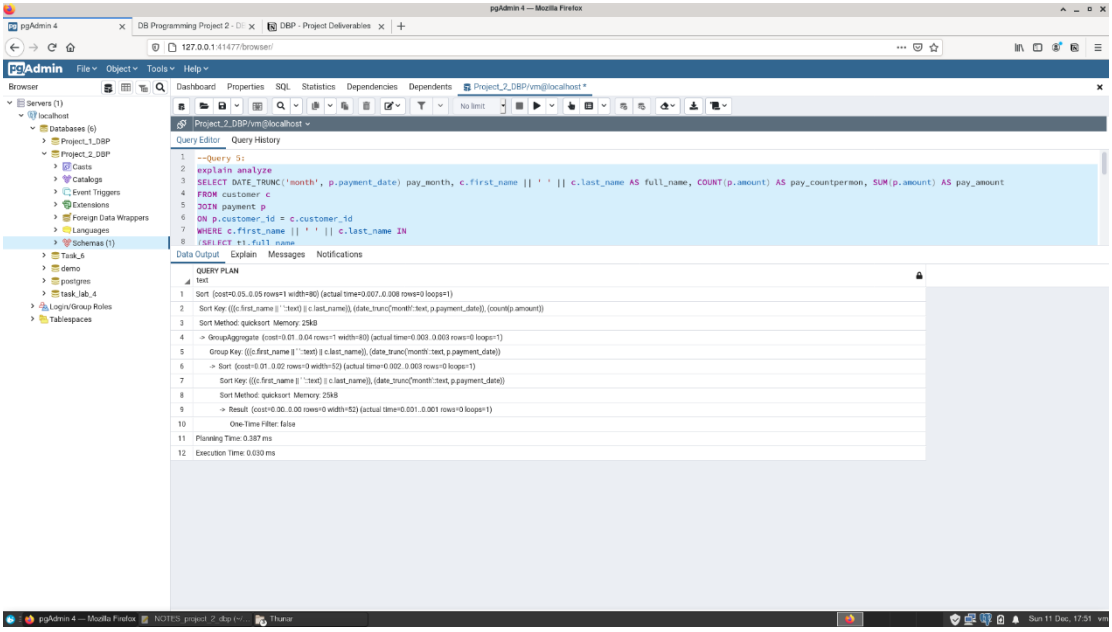
Query 5:

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

Before:



After:



Commands:

The commands we used are:

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

Comparison:

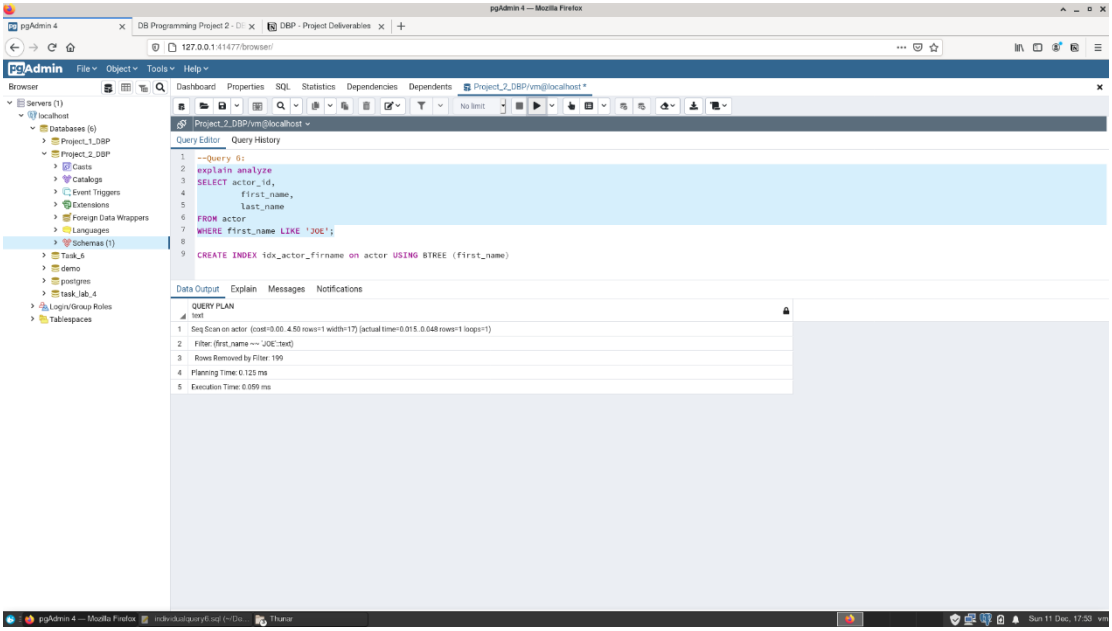
	Before	After
Highest Cost	Sort/hash agg = 0.02	Group agg/sort = 0.02
Slowest Runtime	Sort = 0.17 ms	Sort = 0.005 ms
Largest Number of Rows	Sort/Hash agg = 1	Sort/Group agg = 1
Planning Time	0.646 ms	0.387 ms
Execution Time	0.060 ms	0.030 ms

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

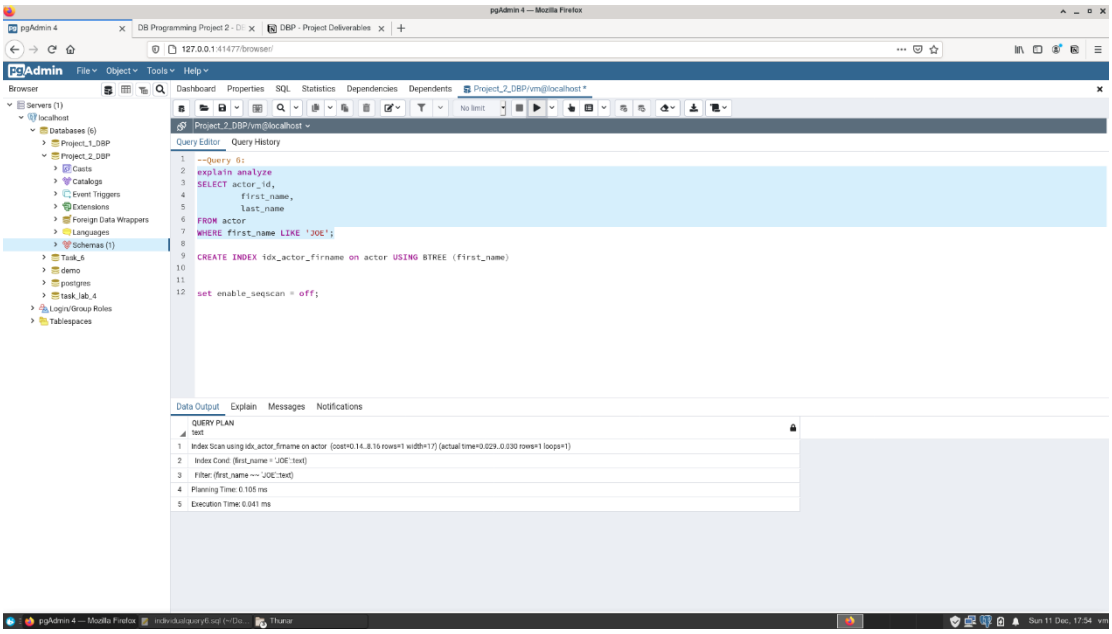
Query 6:

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

Before:



After:



Commands:

The commands we used are:

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

Comparison:

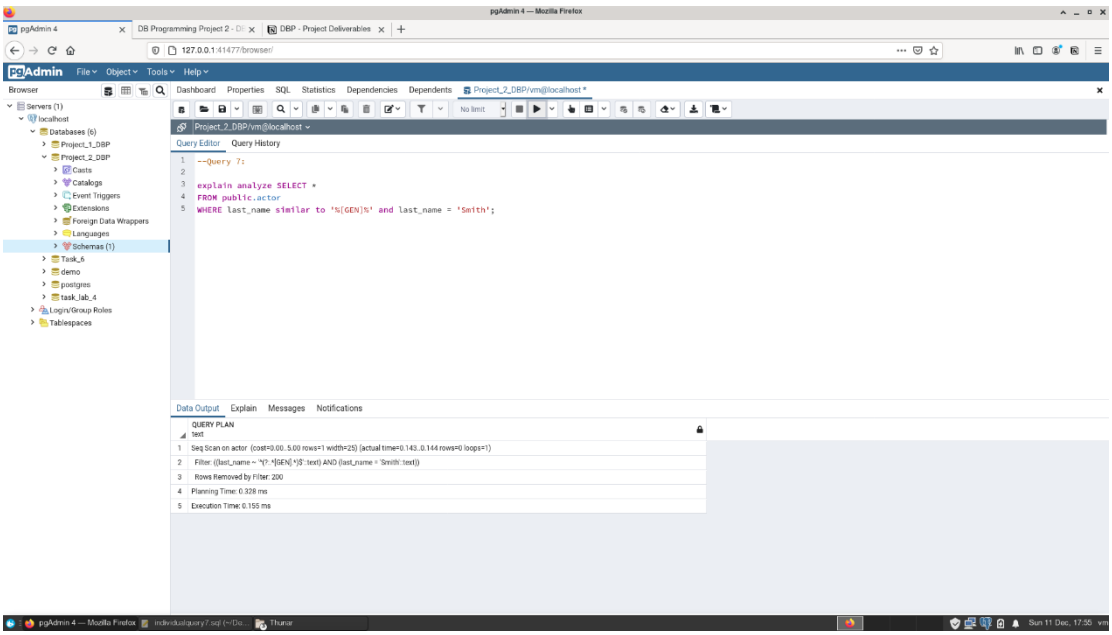
	Before	After
Highest Cost	Seq scan = 4.5	Index scan using idx_actor_firname on actor = 8.16
Slowest Runtime	Seq scan = 0.048 ms	Index scan using idx_actor_firname on actor = 0.03 ms
Largest Number of Rows	Seq scan = 1	Index scan = 1
Planning Time	0.125 ms	0.105 ms
Execution Time	0.059 ms	0.041 ms

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

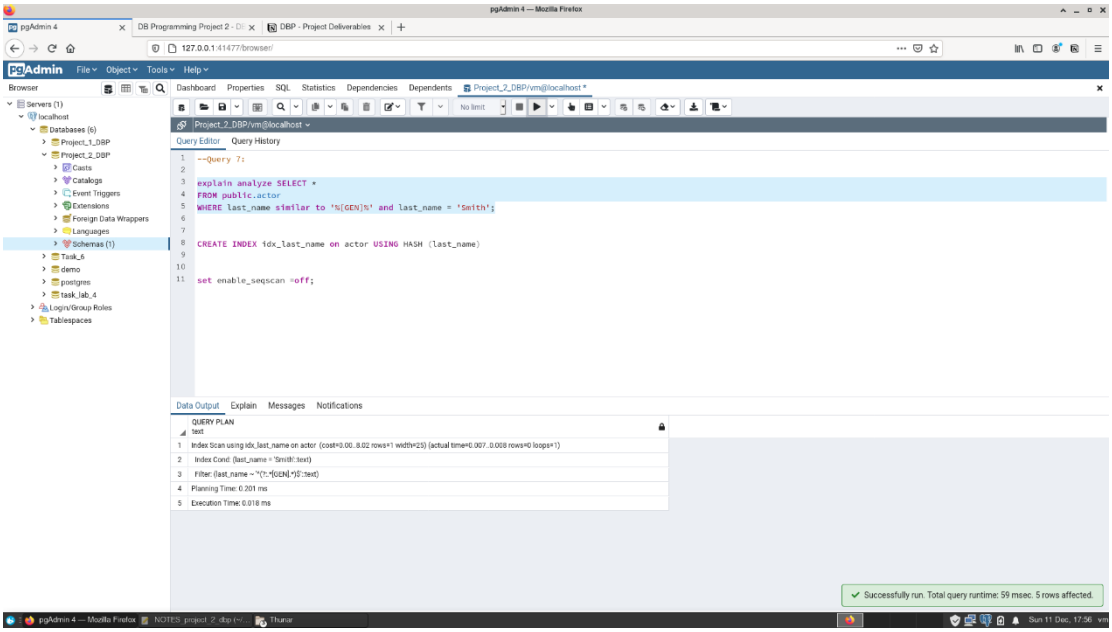
Query 7:

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

Before:



After:



Commands:

The commands we used are:

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

Comparison:

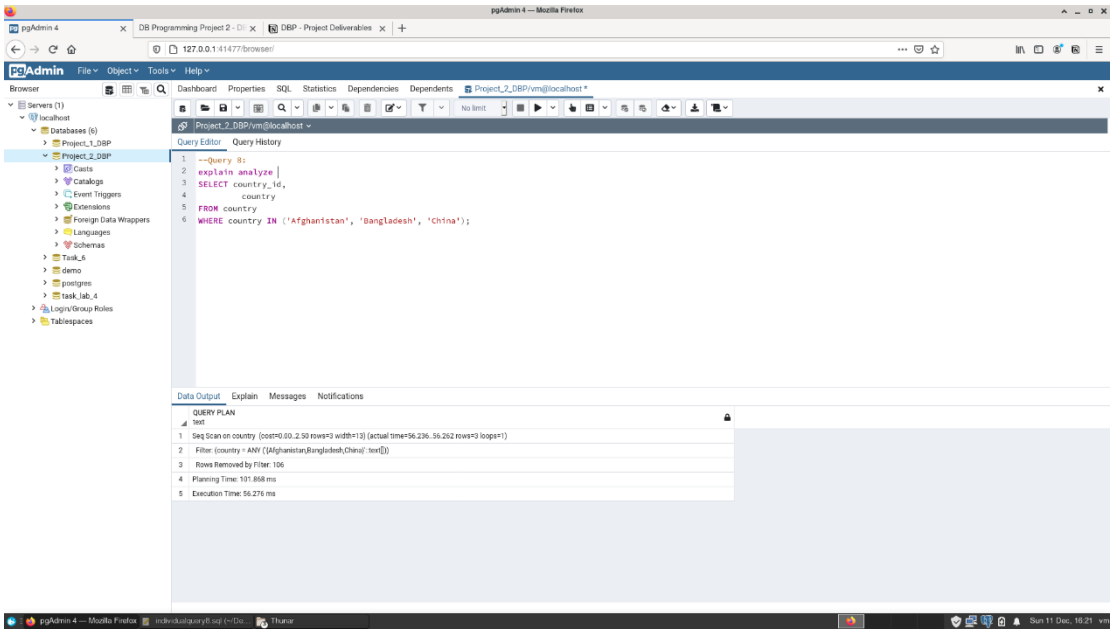
	Before	After
Highest Cost	Seq scan = 5	Index scan using idx_last_name on actor = 8.02
Slowest Runtime	Seq scan = 0.144 ms	Index scan using idx_last_name on actor = 0.008 ms
Largest Number of Rows	Seq scan = 1	Index scan = 1
Planning Time	0.328 ms	0.201 ms
Execution Time	0.155 ms	0.018 ms

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

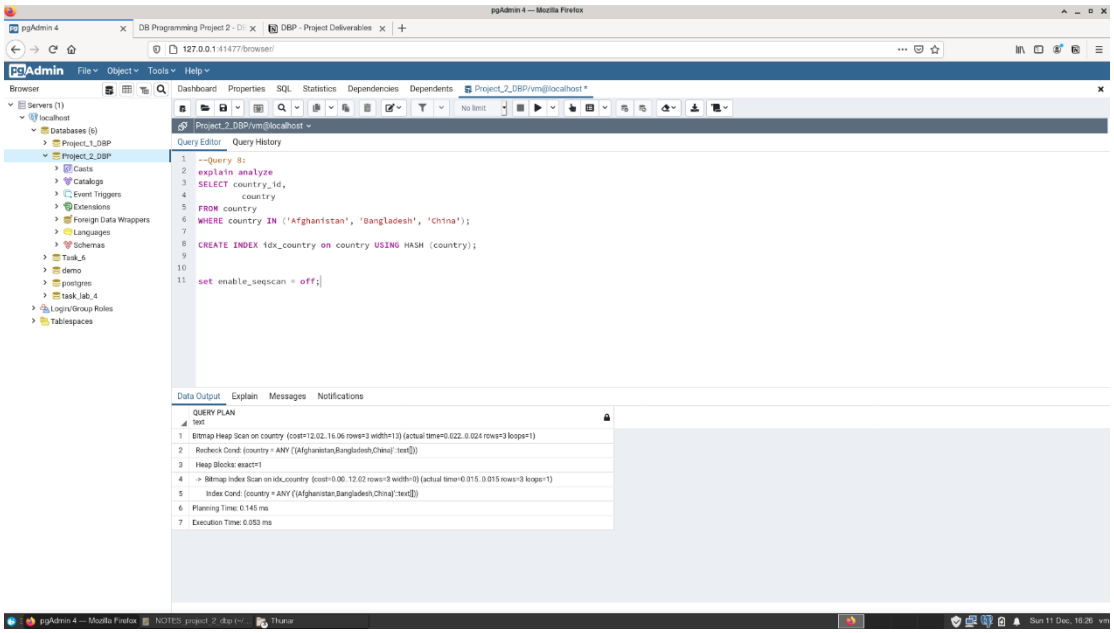
Query 8:

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

Before:



After:



Commands:

The commands we used are:

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

Comparison:

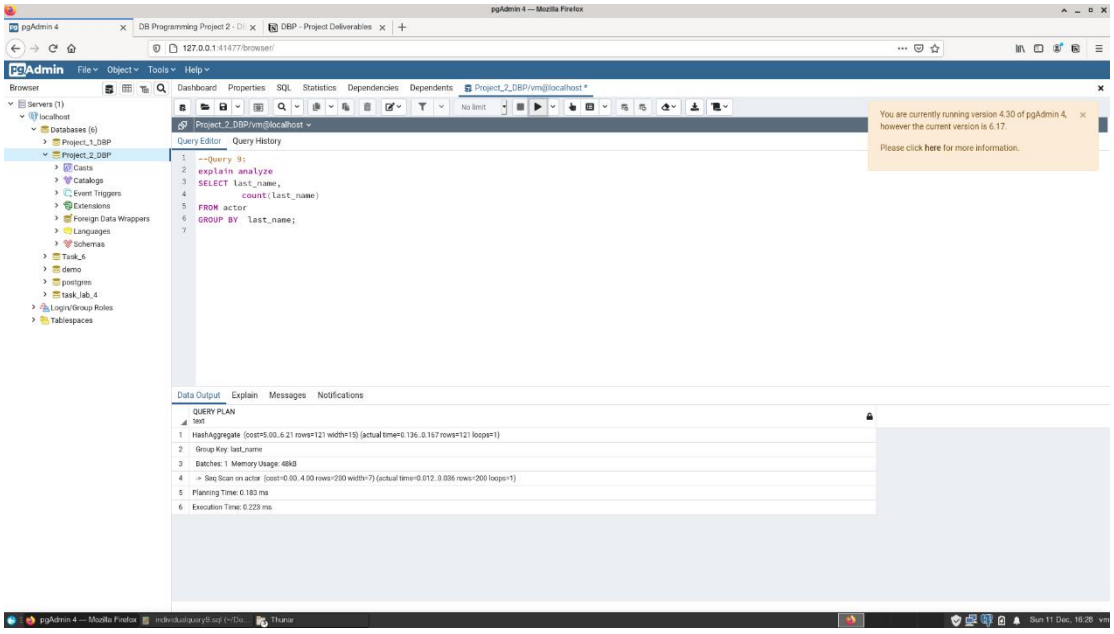
	Before	After
Highest Cost	Seq scan = 2.5	Bitmap heap scan on country = 4.04
Slowest Runtime	Seq scan = 56.262 ms	Bitmap heap scan on country = 0.009 ms
Largest Number of Rows	Seq scan = 3	Bitmap heap scan on country/bitmap index on idx_country = 3
Planning Time	101.868 ms	0.145 ms
Execution Time	56.276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

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

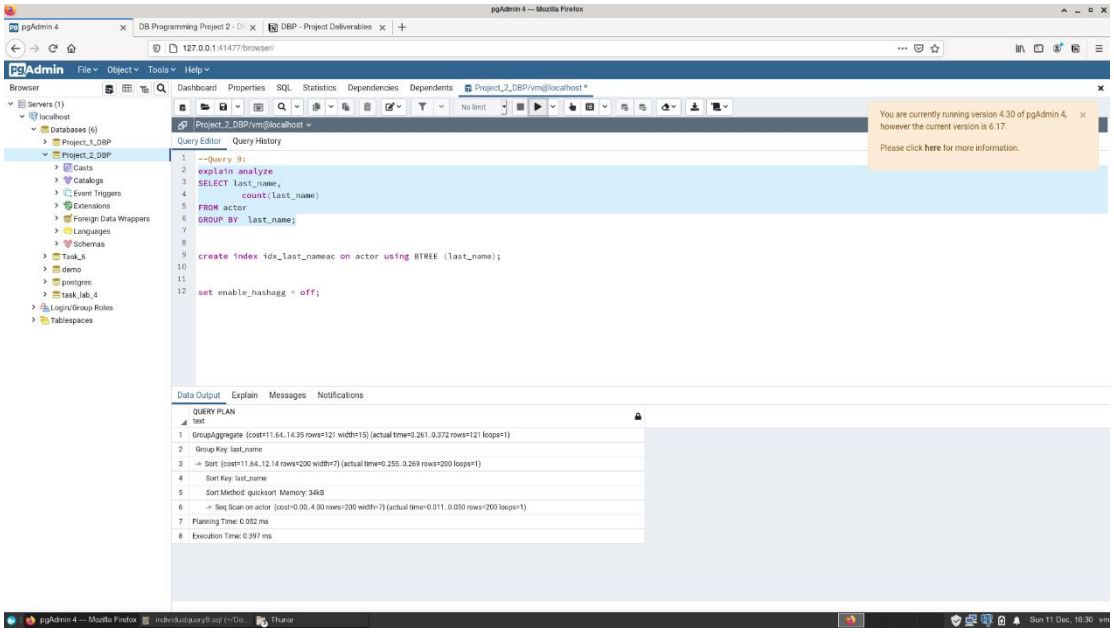
Query 9:

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

Before:



After:



Commands:

The commands we used are:

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

Comparison:

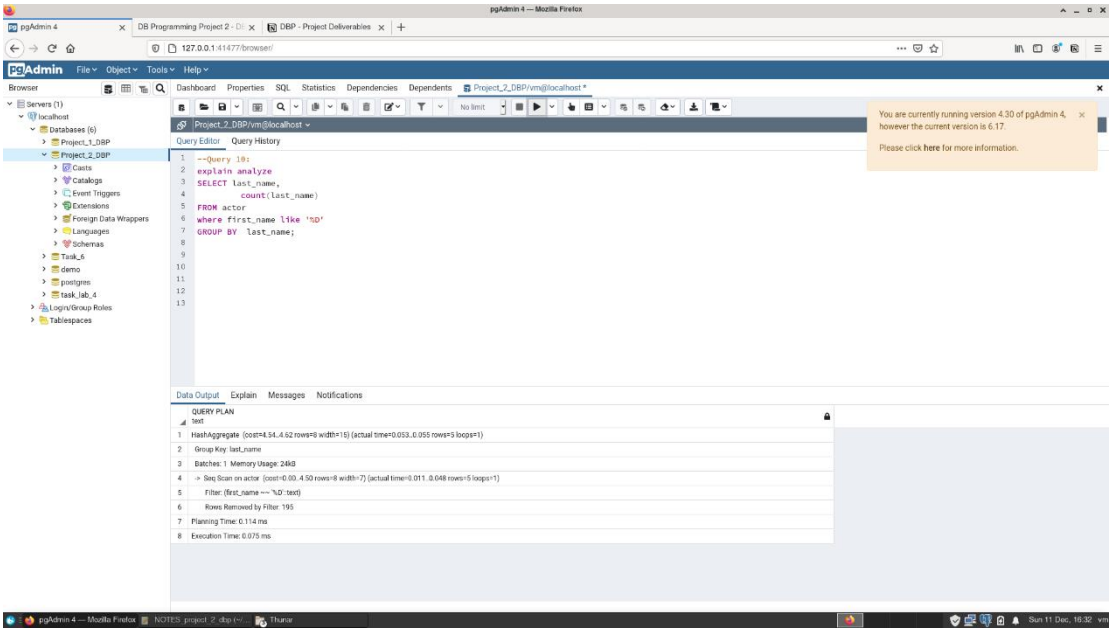
	Before	After
Highest Cost	Hash agg = 2.21	Sort = 8.14
Slowest Runtime	Hash agg = 0.131 ms	Sort = 0.219 ms
Largest Number of Rows	Seq scan on actor = 200	Sort/seq scan on actor = 200
Planning Time	0.183 ms	0.052 ms
Execution Time	0.223 ms	0.397 ms

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

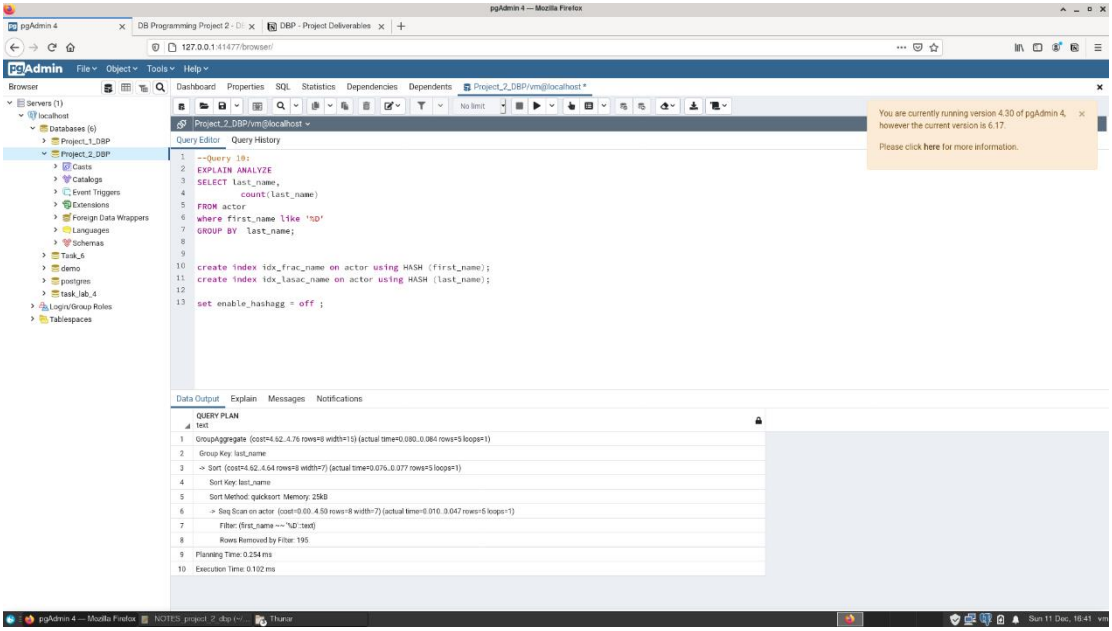
Query 10:

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

Before:



After:



Commands:

The commands we used are:

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

Comparison:

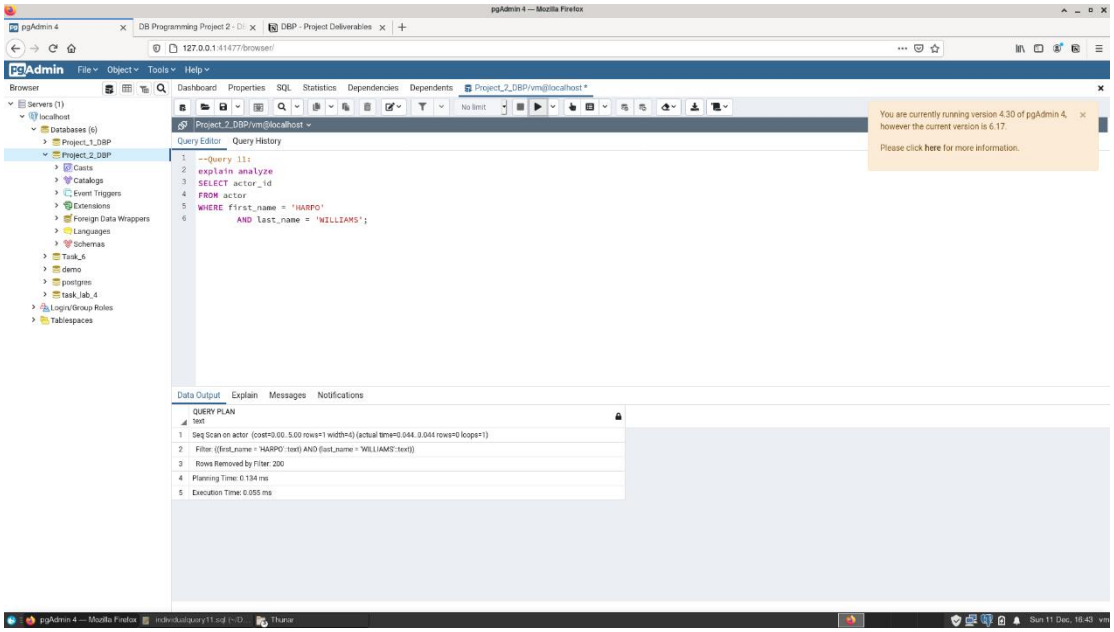
	Before	After
Highest Cost	Hash agg = 4.12	Sort = 4.14
Slowest Runtime	Hash agg = 0.007 ms	Sort = 0.03 ms
Largest Number of Rows	Hash agg/ seq scan on actor = 8	Group agg/sort/seq scan on actor = 8
Planning Time	0.114 ms	0.254 ms
Execution Time	0.075 ms	0.102 ms

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

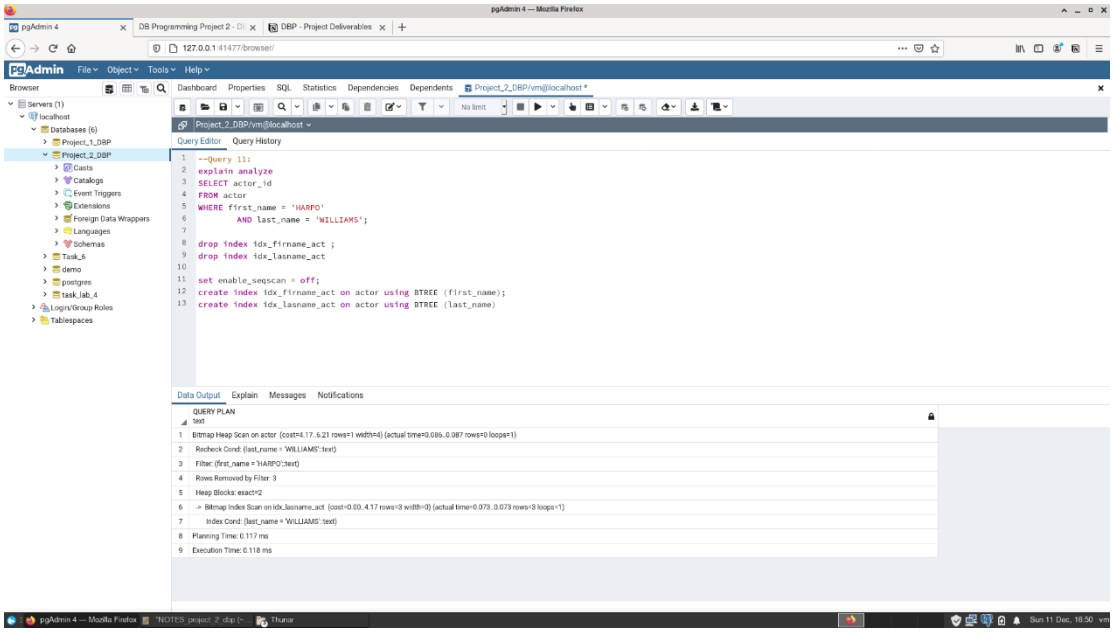
Query 11:

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

Before:



After:



Commands:

The commands we used are:

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

Comparison:

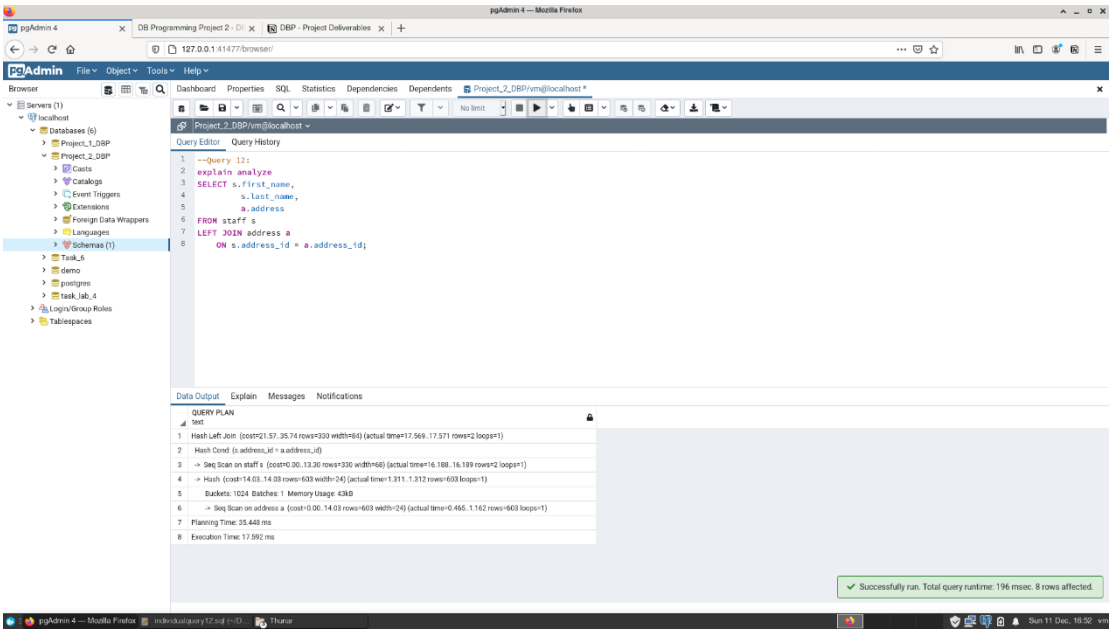
	Before	After
Highest Cost	Seq scan = 5	Bitmap heap scan on actor = 2.04
Slowest Runtime	Seq scan = 0.044 ms	Bitmap heap scan on actor = 0.014 ms
Largest Number of Rows	Seq scan on actor = 1	Bitmap index on idx_lasname_act = 3
Planning Time	0.134 ms	0.117 ms
Execution Time	0.055 ms	0.118 ms

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

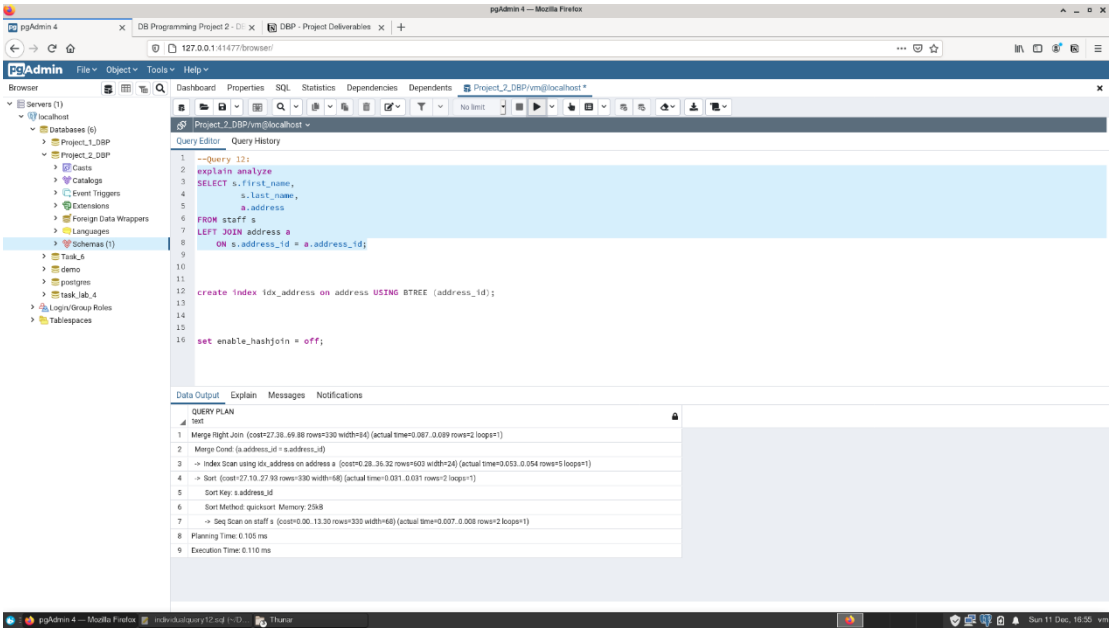
Query 12:

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

Before:



After:



Commands:

The commands we used are:

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

Comparison:

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

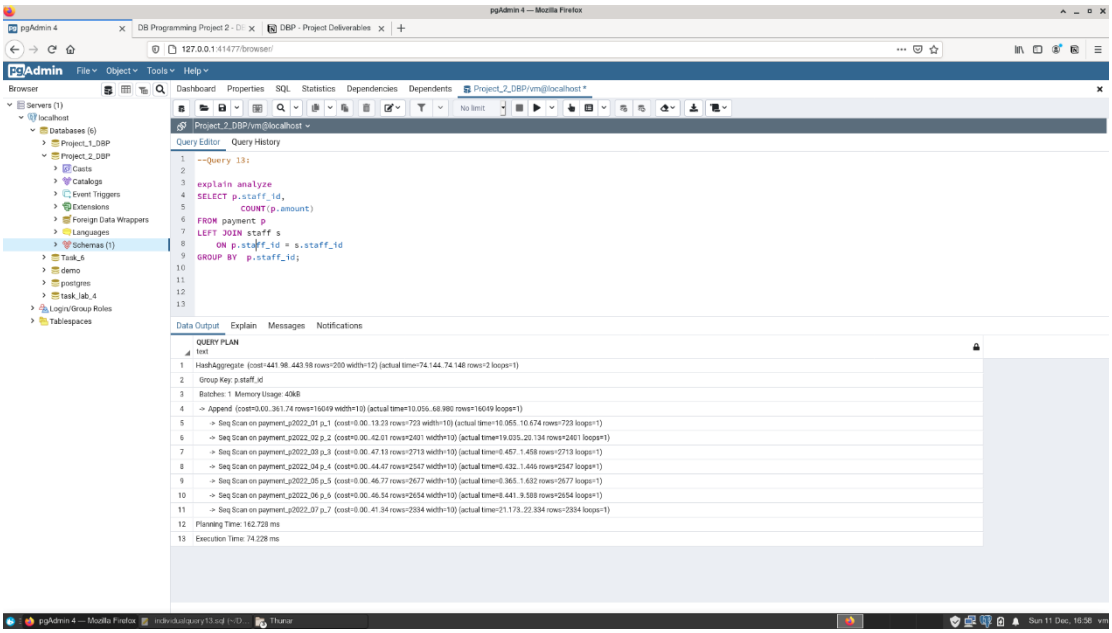
-The comparison shows that the cost is higher, and the time is less as the hash left join exchanged to merge right join for better optimization.

Query 13:

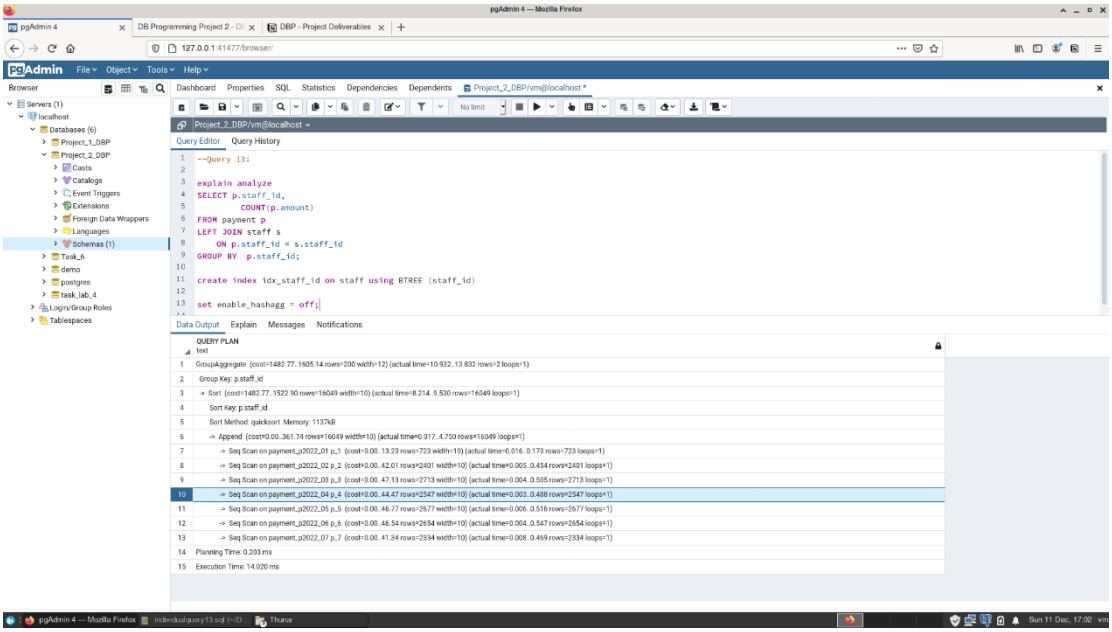
For query 13 we used an index on the (staff_id) attribute using BTREE.

For better optimization and fast searching.

Before:



After:



Commands:

The commands we used are:

- create index idx_staff_id on staff using BTREE (staff_id)
- set enable_hashagg = off;

Comparison:

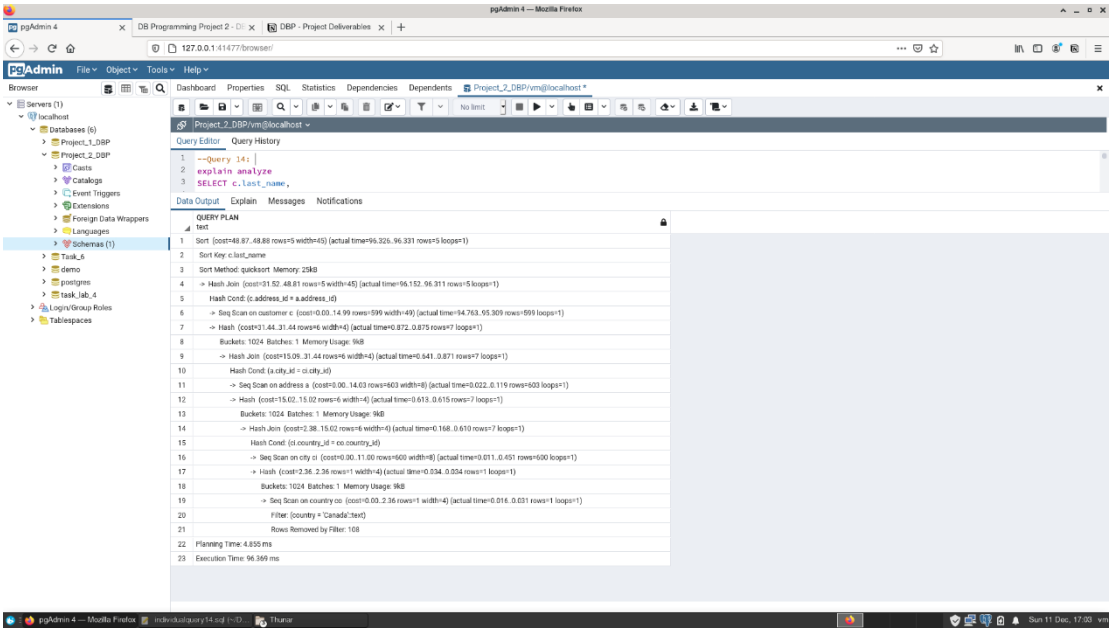
	Before	After
Highest Cost	Hash agg = 82.24	Append = 15767.51
Slowest Runtime	Hash agg = 5.168 ms	Sort = 4.78 ms
Largest Number of Rows	Seq scan on payment_p2022_03 p_3 = 2713	Seq scan on payment_p2022_03 p_3 = 2713
Planning Time	162.728 ms	0.203 ms
Execution Time	74.228 ms	14.020 ms

-The comparison shows that the cost is higher, and the time was less, and the hash aggregate exchanged with group aggregate for better optimization.

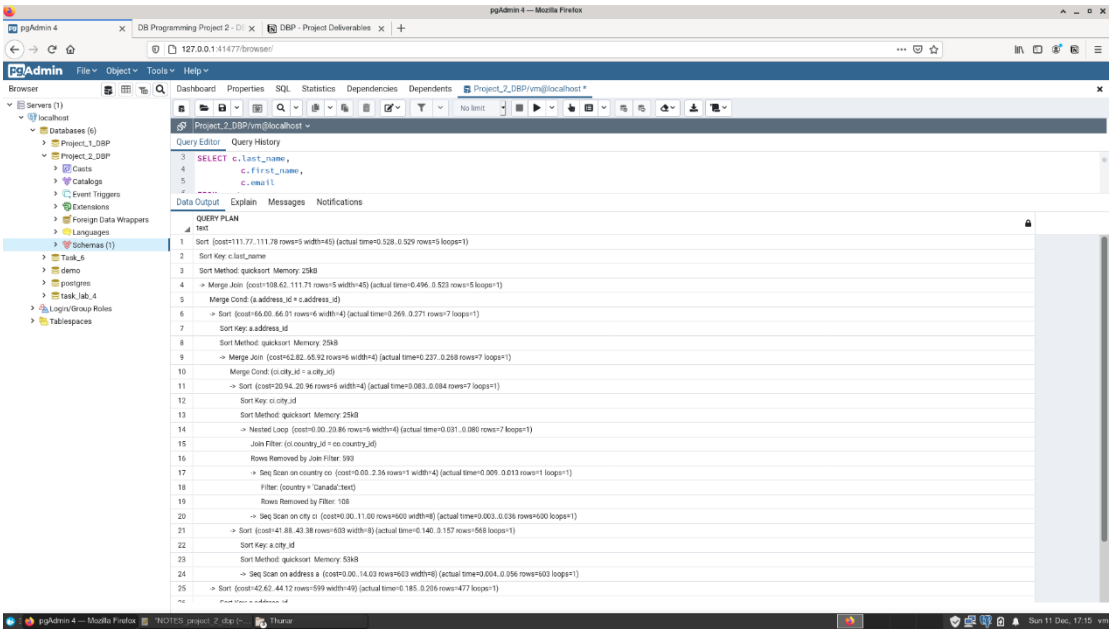
Query 14:

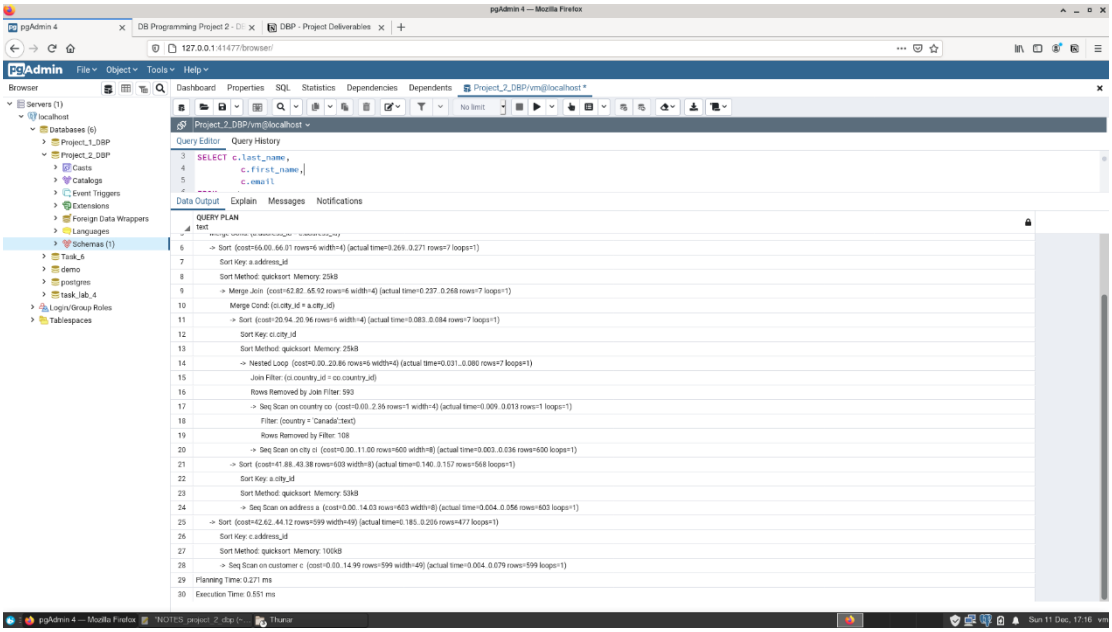
For query 14 we used an index on the (address_id, city_id, country, last_name) attributes using BTREE for the (address_id, last_name) attributes and HASH for the (city_id, country) attributes. Using btree and hash to give better performance for faster searching. Sometimes we used hash instead of btree to give better results than using the btree.

Before:



After (2 Screenshots):





Commands:

The commands we used are:

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

Comparison:

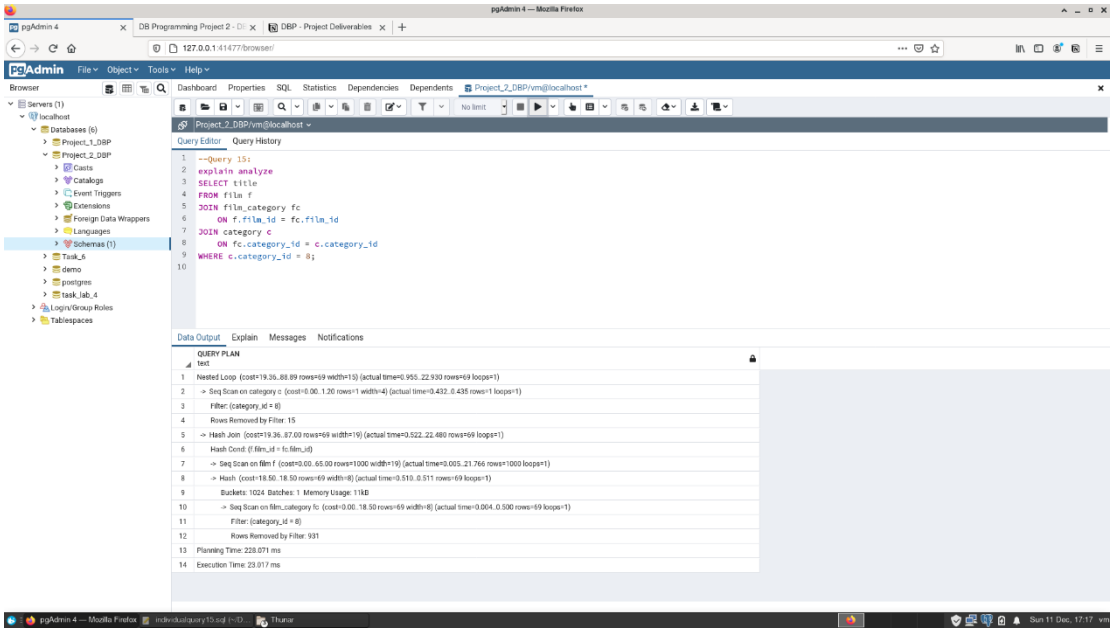
	Before	After
Highest Cost	Hash join = 2.39	Merge join = 45.7
Slowest Runtime	Hash join = 0.137 ms	Merge join = 0.252 ms
Largest Number of Rows	Seq scan on address a = 603	Seq scan on address a = 603
Planning Time	4.855 ms	0.271 ms
Execution Time	96.369 ms	0.551 ms

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

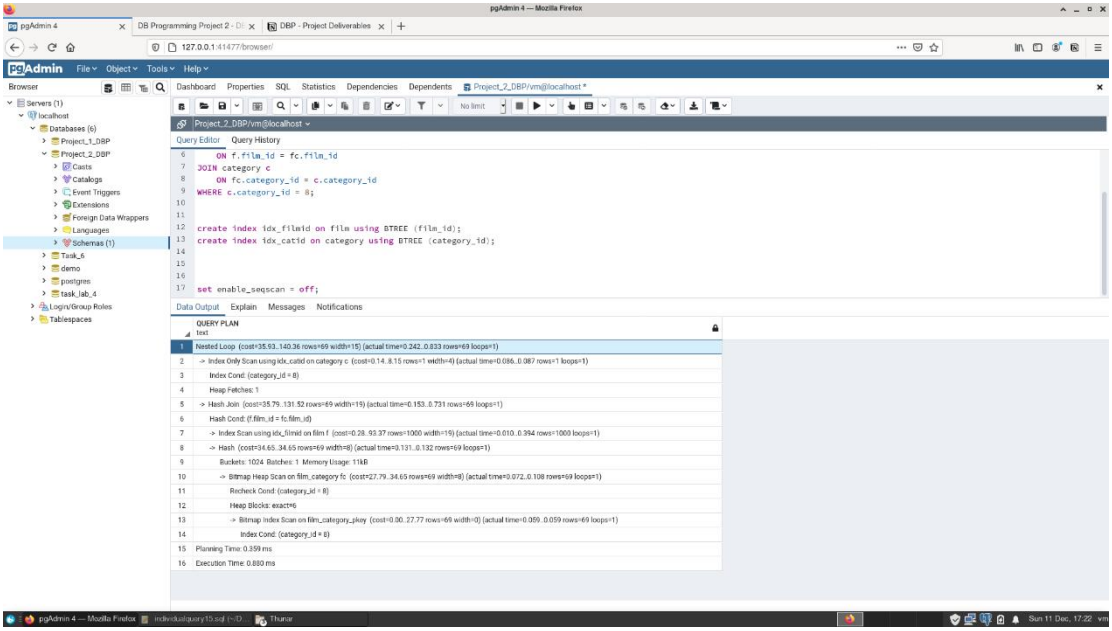
Query 15:

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

Before:



After:



Commands:

The commands we used are:

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

Comparison:

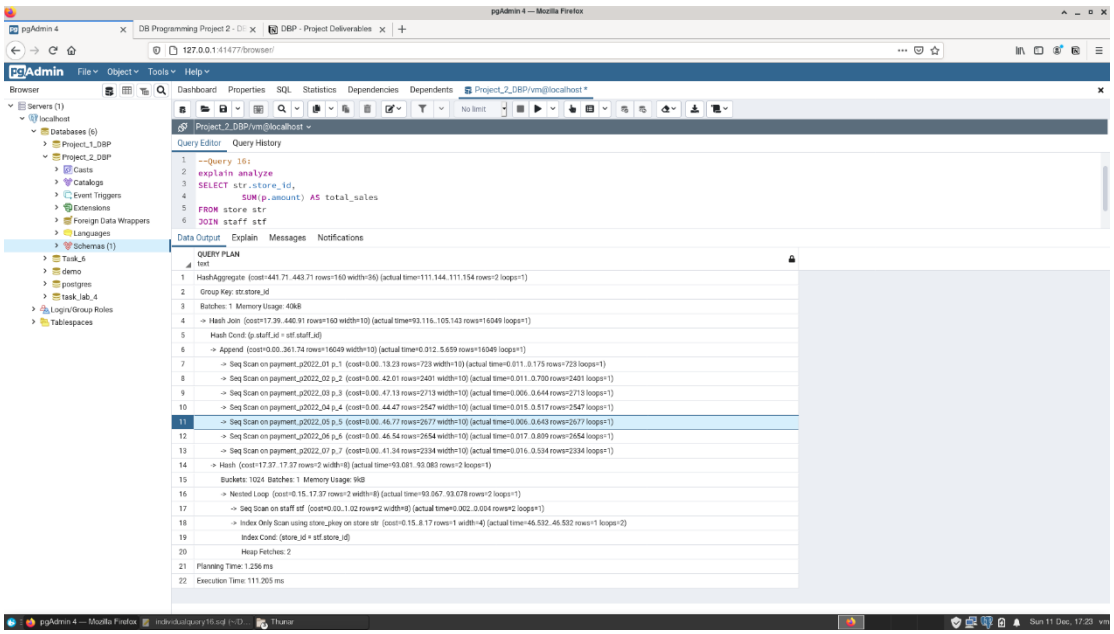
	Before	After
Highest Cost	Hash join = 3.5	Hash join = 3.5
Slowest Runtime	Hash join = 0.203 ms	Index scan using idx_filmid on film f = 0.205 ms
Largest Number of Rows	Seq scan on film f = 1000	Index scan using idx_filmid on film f = 1000
Planning Time	228.071 ms	0.359 ms
Execution Time	23.017 ms	0.880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

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

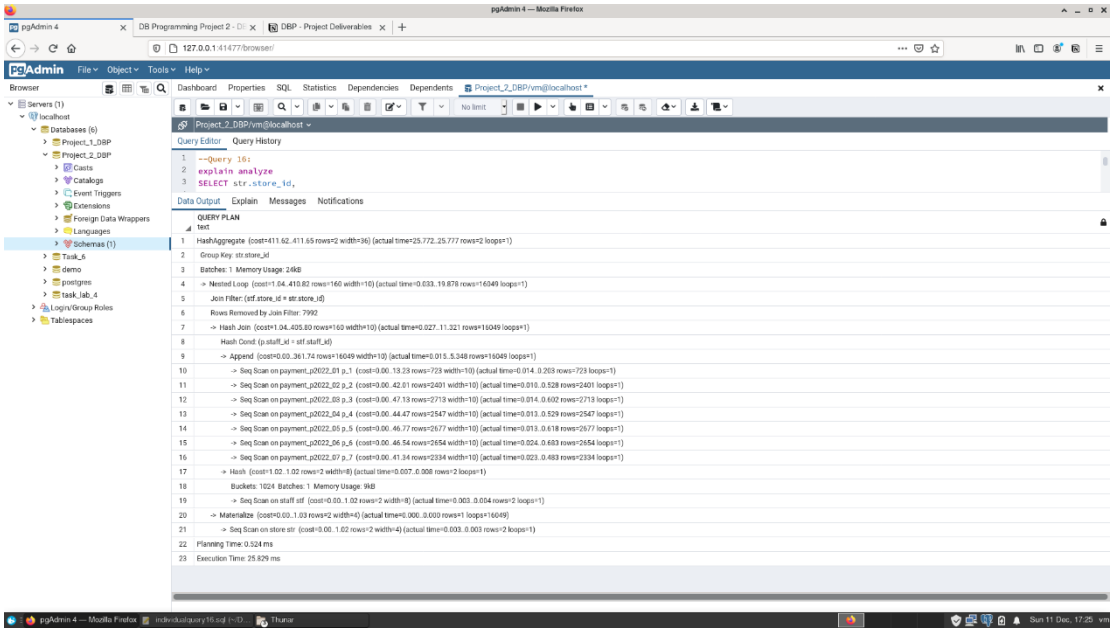
Query 16:

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

Before:



After:



Commands:

The commands we used are:

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

Comparison:

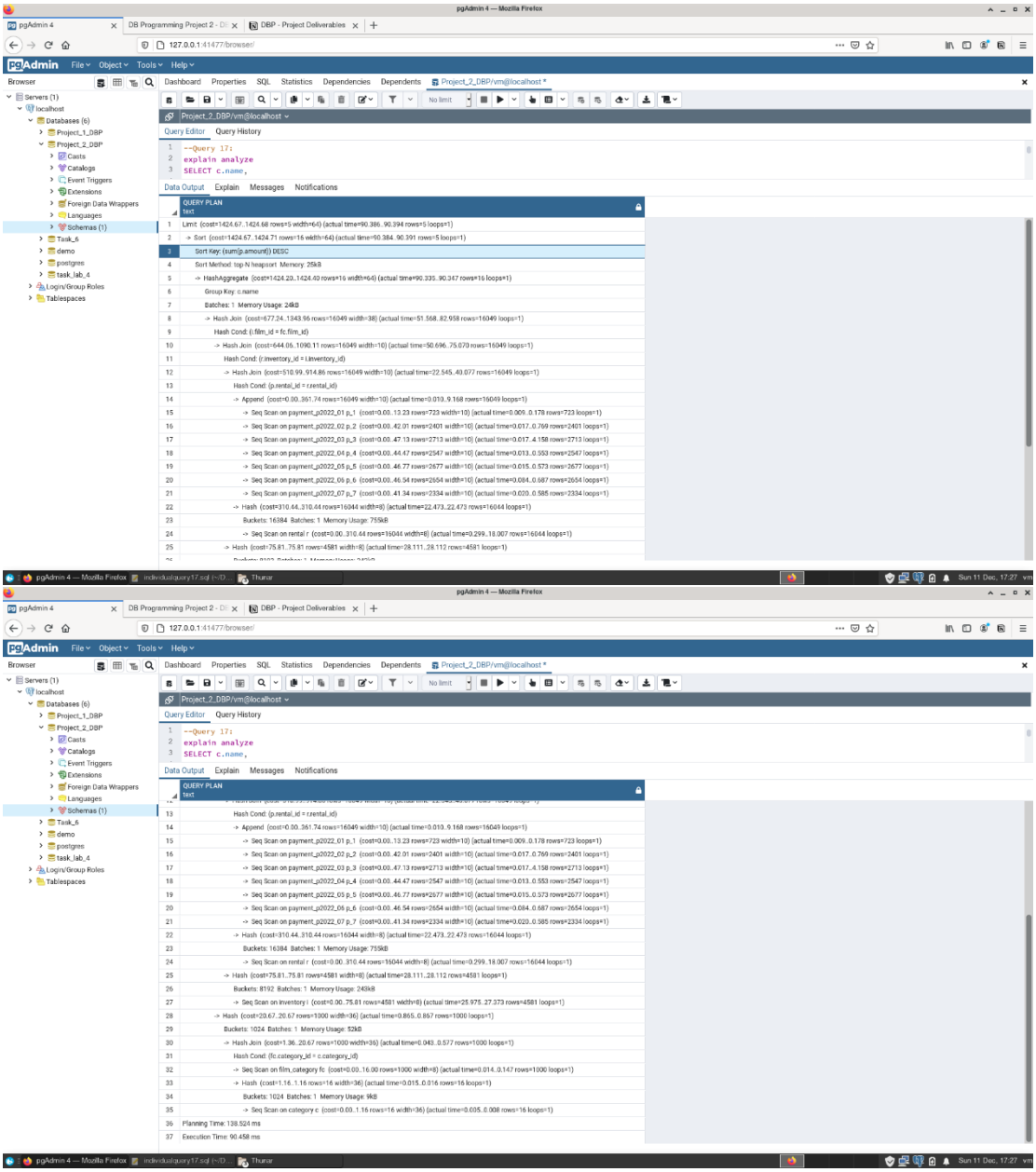
	Before	After
Highest Cost	Append = 80.25	Append = 80.25
Slowest Runtime	Nested loop = 46.542 ms	Nested loop = 8.557 ms
Largest Number of Rows	Seq scan on payment_p2022_03 p_3 = 2713	Seq scan on payment_p2022_03 p_3 = 2713
Planning Time	1.256 ms	0.524 ms
Execution Time	111.205 ms	25.829 ms

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

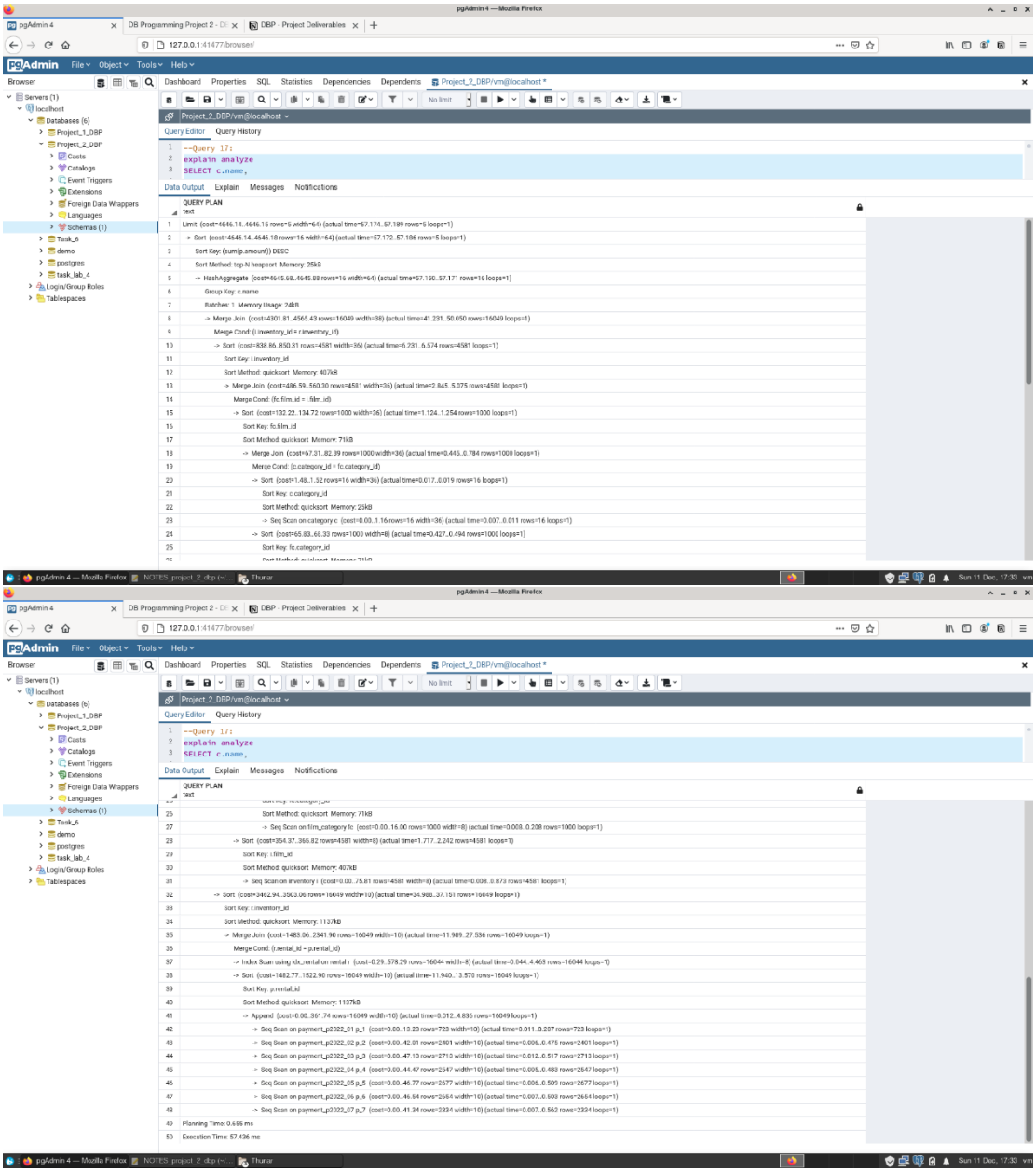
Query 17:

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

Before (2 Screenshots):



After (2 screenshots):



Commands:

The commands we used are:

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

Comparison:

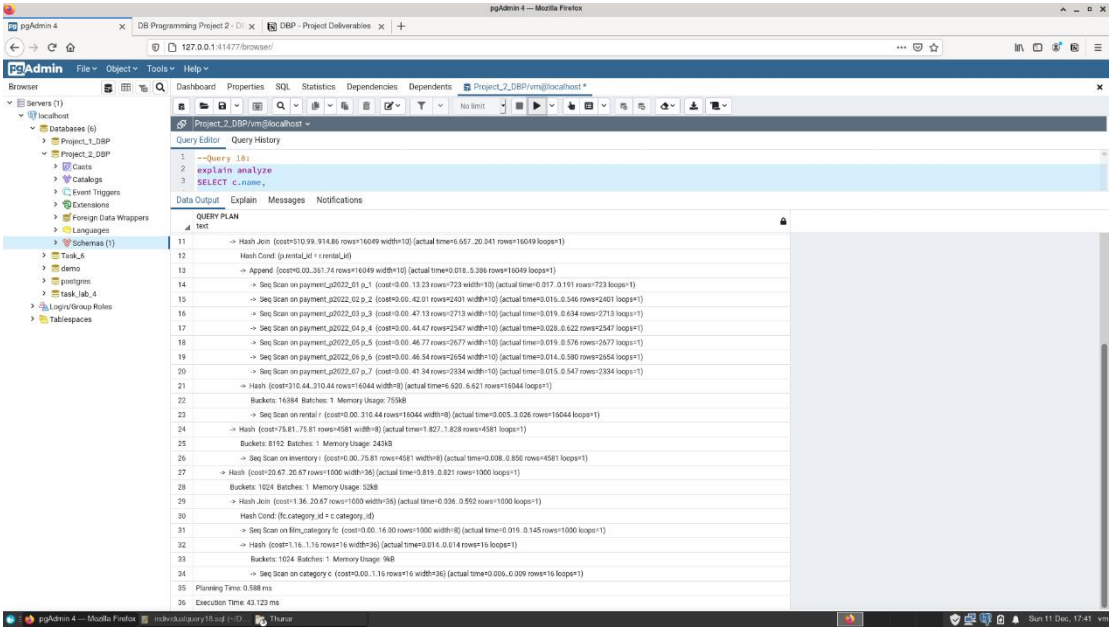
	Before	After
Highest Cost	Hash join = 579.12	Hash agg = 3715.12
Slowest Runtime	Hash join = 34.993 ms	Merge join = 43.476 ms
Largest Number of Rows	Hash join = 16049	Merge join = 16049
Planning Time	138.524 ms	0.655 ms
Execution Time	90.458 ms	57.436 ms

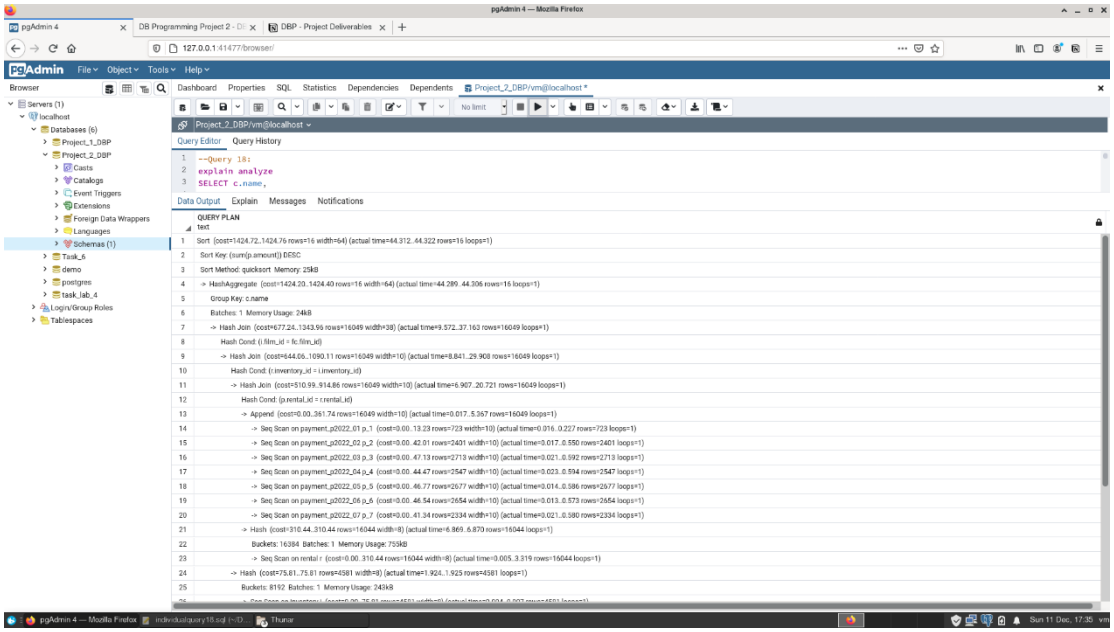
-The comparison shows that the cost is higher, and the time is less, and the hash join exchanged into merge join for better optimization.

Query 18:

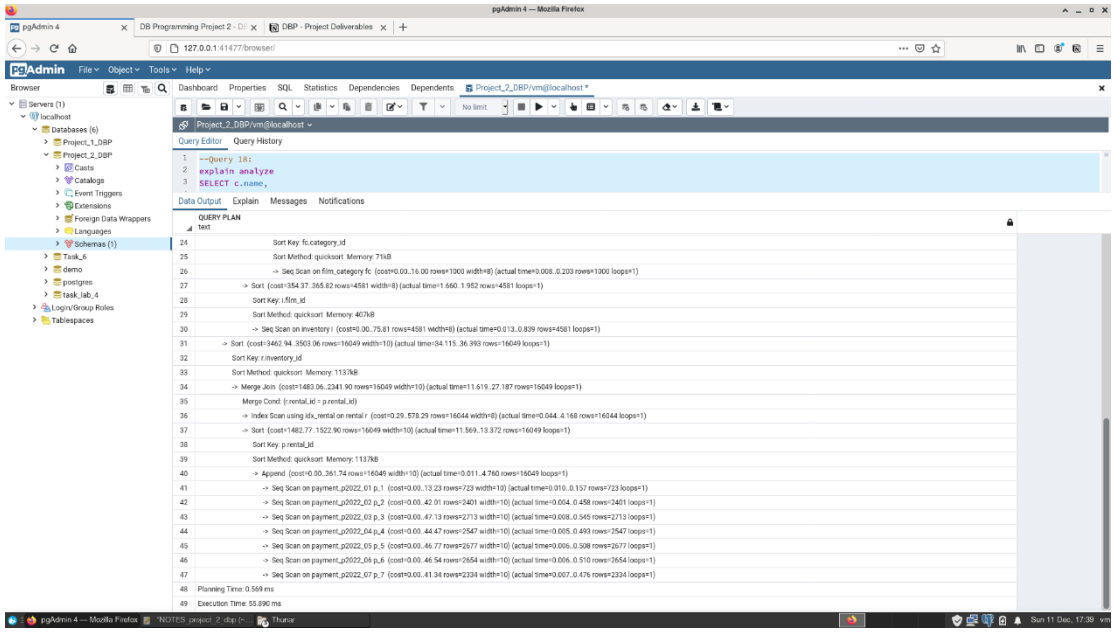
For query 18 we used an index on the (category_id, film_id, inventory_id, rental_id) attributes using BTREE on all four of them. For better performance and fast searching.

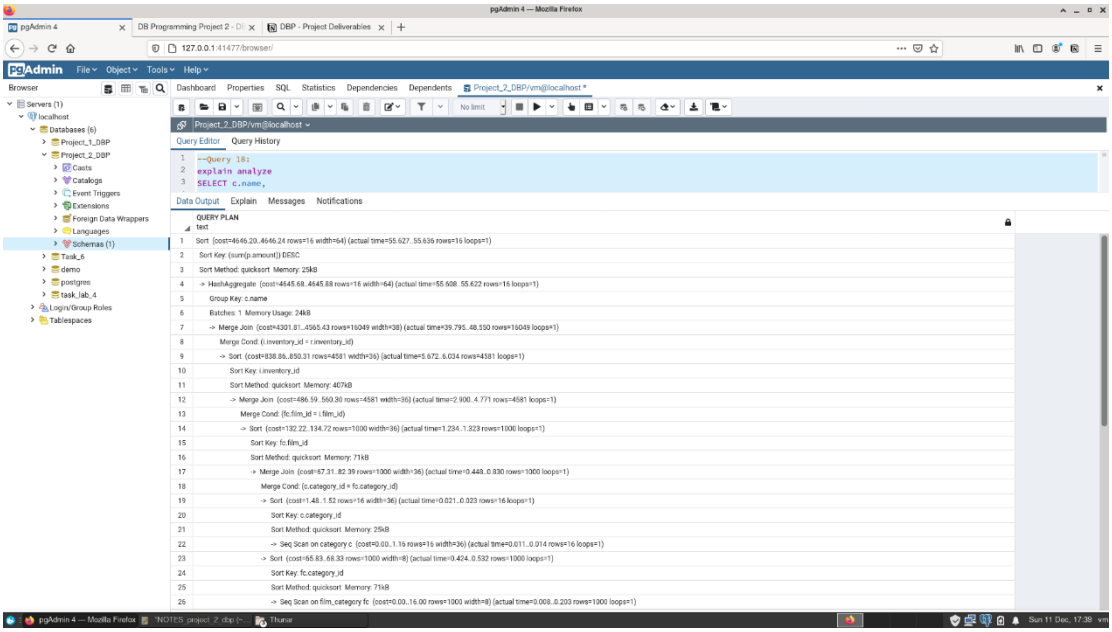
Before (2 Screenshots):





After (2 Screenshots):





Commands:

The commands we used are:

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

Comparison:

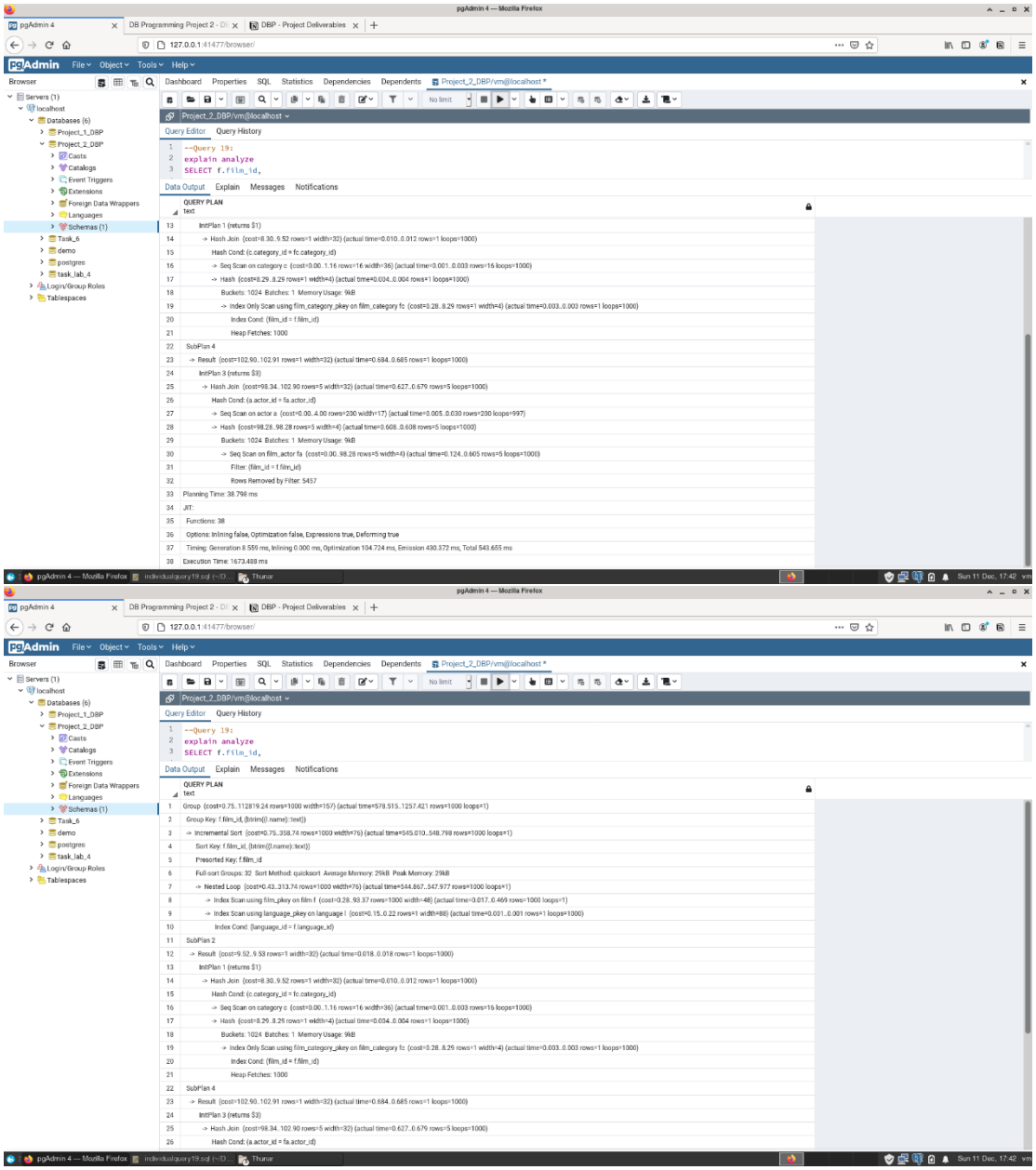
	Before	After
Highest Cost	Hash join = 553.12	Merge join = 3715.12
Slowest Runtime	Hash agg = 7.143 ms	Merge join = 42.516 ms
Largest Number of Rows	Hash join = 16049	Merge join = 16049
Planning Time	0.588 ms	0.569 ms
Execution Time	43.123 ms	55.890 ms

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

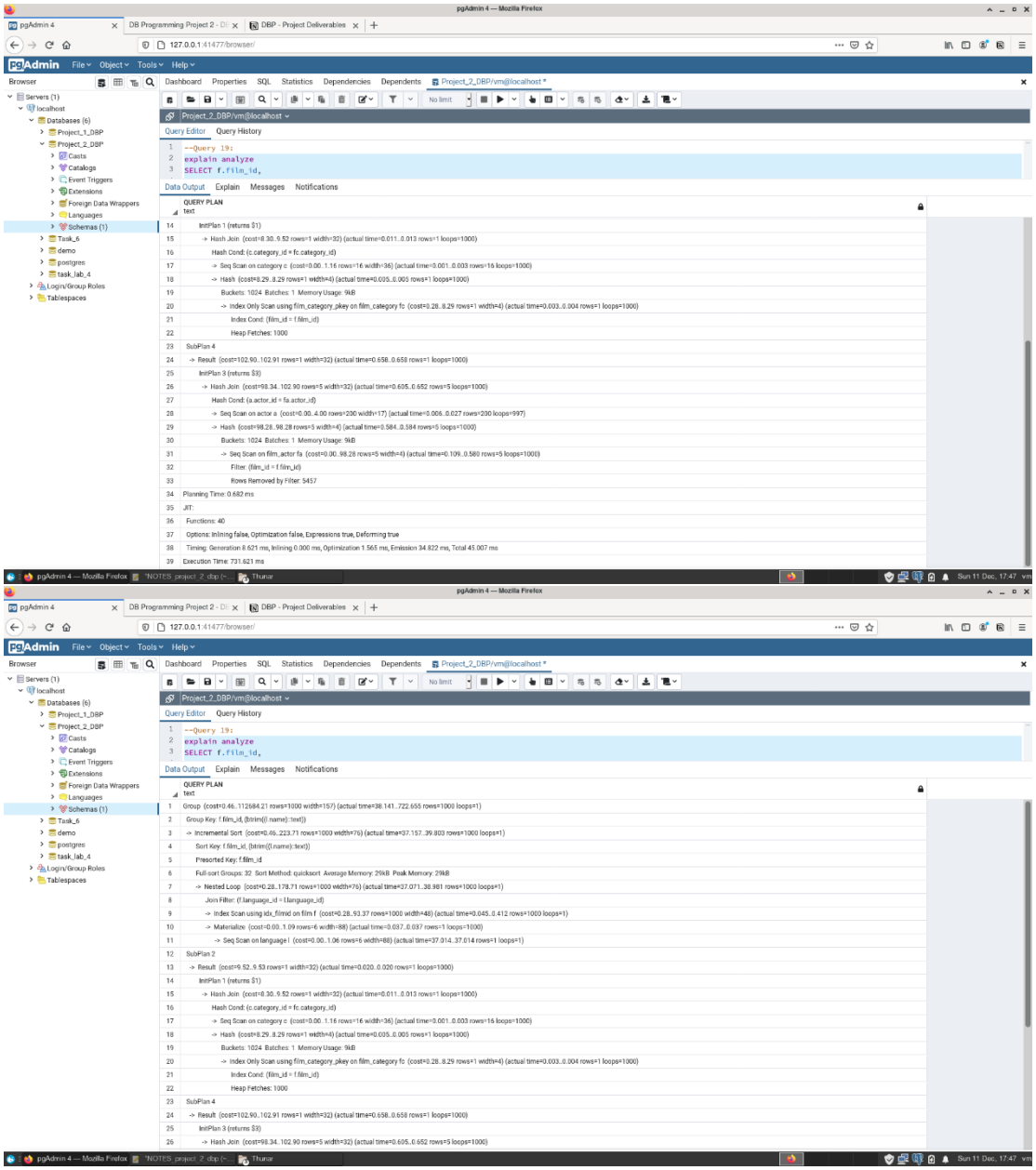
Query 19:

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

Before (2 Screenshots):



After (2 Screenshots):



Commands:

The commands we used are:

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

Comparison:

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

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