Orysya Stus
2.27.2017

Milestone III: Query Performance

Work flow

Professor's solutions were used for performance testing. The ipython code for creating random data was used from Kyle (Cats) and Tony/Sanjay(Sales). For performance testing, queries were run 3 times with the average time taken as the performance metric, the % change was calculated using
[1 – (with index time/without index time)]*100%
Screenshots & raw data from queries are included at the end.

Sales

Tested the following indexes:
CREATE INDEX sales_cust_id ON sales.sale(customer_id);
CREATE INDEX sales_prod_id ON sales.sale(product_id);
CREATE INDEX cust_state_id ON sales.customer(state_id);
CREATE INDEX product_cat_id ON sales.product(category_id);

Query 1: None
Tried: CREATE INDEX sales_cust_id ON sales.sale(customer_id);
Analysis: 0.4% slower compared to default query, not incorporated by optimizer
Reasoning: Do not use this index. Sequential scanning on customer_id requires less cost to execute.

Query 2: None
Tried: CREATE INDEX sales_cust_id ON sales.sale(customer_id);
Analysis: 0.9% slower compared to default query, not incorporated by optimizer
Reasoning: Do not use this index. Sequential scanning on sale table is preferred since less cost is necessary.

Tried: CREATE INDEX cust_state_id ON sales.customer(state_id);
Analysis: 0.4% slower compared to default query, not incorporated by optimizer
Reasoning: Do not use this index. Sequential scanning on state table is preferred since less cost is necessary.

Query 3: sales_cust_id
Tried: CREATE INDEX sales_cust_id ON sales.sale(customer_id);
Analysis: 19.3% faster compared to default query, incorporated by optimizer
Reasoning: Do use this index. By subsetting the data using an index scan, the time required to satisfy the index condition is faster and the cost for scanning is faster. The 'WHERE' clause in this query allows for optimal indexing.

Query 4: None
Tried: CREATE INDEX sales_cust_id ON sales.sale(customer_id);
Analysis: 0% change compared to default query, not incorporated by optimizer
Reasoning: Do not use this index. Sequential scanning on the sale table is favorable, lower cost for the query. No need to index the foreign key.

Tried: CREATE INDEX sales_prod_id ON sales.sale(product_id);
Analysis: 0% change compared to default query, not incorporated by optimizer
Reasoning: Do not use this index. Sequential scanning on the sale table is favorable, lower cost for the query. No need to index the foreign key.

Query 5: None
Tried: CREATE INDEX sales_cust_id ON sales.sale(customer_id);
Analysis: 0.5% faster compared to default query, not incorporated by optimizer
Reasoning: Do not use this index. Sequential scanning on the sale table is favorable, lower cost for the query. No need to index the foreign key.

Tried: CREATE INDEX sales_prod_id ON sales.sale(product_id);
Analysis: 7.3% slower compared to default query, not incorporated by optimizer
Reasoning: Do not use this index. Sequential scanning on the sale table is favorable, lower cost for the query. No need to index the foreign key.

Query 6: None
Tried: CREATE INDEX sales_cust_id ON sales.sale(customer_id);
Analysis: 3.1% slower compared to default query, not incorporated by optimizer
Reasoning: Do not use this index. Sequential scanning on the sale table is favorable, lower cost for the query. No need to index the foreign key.

Tried: CREATE INDEX sales_prod_id ON sales.sale(product_id);
Analysis: 10.7% slower compared to default query, not incorporated by optimizer
Reasoning: Do not use this index. Sequential scanning on the sale table is favorable, lower cost for the query. No need to index the foreign key.

Cats

Tested the following indexes:
CREATE INDEX login_user_id_index ON cats.login(user_id);
CREATE INDEX friend_user_id_index ON cats.friend(user_id);
CREATE INDEX watch_user_id_index ON cats.watch(user_id);
CREATE INDEX watch_video_id_index ON cats.watch(video_id);
CREATE INDEX likes_user_id_index ON cats.likes(user_id);
CREATE INDEX likes_video_id_index ON cats.likes(video_id);

Query 1: watch_user_id_index, likes_user_id_index, likes_video_id_index
Tried: CREATE INDEX watch_user_id_index ON cats.watch(user_id);
Analysis: 6.3% slower compared to default query, is incorporated by optimizer
Reasoning: Do use this index. Although the performance time did not decrease, the optimizer used this index instead of a sequential scan of the watch table as this allowed for heap blocks to be created and condition checks. With larger data sets provided, performance differences should be more noticeable. The where clause made this index optimal.

Tried: CREATE INDEX watch_video_id_index ON cats.watch(video_id);
Analysis: 2.8% slower compared to default query, not incorporated by optimizer
Reasoning: Do not use this index. Indexing the query on video_id in the watch table does not increase the performance of the query

Tried: CREATE INDEX likes_user_id_index ON cats.likes(user_id);
Analysis: 5.3% slower compared to default query, incorporated by optimizer
Reasoning: Do use this index. Although the performance time did not decrease, the optimizer used this index instead of a sequential scan of the likes table as this allowed for heap blocks to be created and condition checks. With larger data sets provided, performance differences should be more noticeable. The where clause made this index optimal.

Tried: CREATE INDEX likes_video_id_index ON cats.likes(video_id);
Analysis: 7.7% slower compared to default query, incorporated by optimizer
Reasoning: Do use this index. Although the performance time did not decrease, the optimizer used this index instead of a sequential scan of the likes table as this allowed for heap blocks to be created and condition checks. With larger data sets provided, performance differences should be more noticeable. The where clause made this index optimal.

Query 2: friend_user_id_index, watch_user_id_index, likes_user_id_index, likes_video_id_index
Tried: CREATE INDEX friend_user_id_index ON cats.friend(user_id);
Analysis: 5.8% faster compared to default query, incorporated by optimizer
Reasoning: Do use this index. Index scanning allowed the query to run faster, and with larger data sets the performance should only increase. The where clause made this index optimal.

Tried: CREATE INDEX watch_user_id_index ON cats.watch(user_id);
Analysis: 2.5% faster compared to default query, incorporated by optimizer
Reasoning: Do use this index. Index scanning allowed the query to run faster, and with larger data sets the performance should only increase. The where clause made this index optimal.

Tried: CREATE INDEX watch_video_id_index ON cats.watch(video_id);
Analysis: 3.9% faster compared to default query, not incorporated by optimizer
Reasoning: Do not use this index. Although this was contained in a where clause, the cost to index was higher than the cost to perform sequential scanning.

Tried: CREATE INDEX likes_user_id_index ON cats.likes(user_id);
Analysis: 0.9% faster compared to default query, incorporated by optimizer
Reasoning: Do use this index. Index scanning allowed the query to run faster, and with larger data sets the performance should only increase. The where clause made this index optimal.

Tried: CREATE INDEX likes_video_id_index ON cats.likes(video_id);
Analysis: 5.4% faster compared to default query, incorporated by optimizer
Reasoning: Do use this index. Index scanning allowed the query to run faster, and with larger data sets the performance should only increase. The where clause made this index optimal.

Query 3: friend_user_id_index, watch_user_id_index, likes_user_id_index
Tried: CREATE INDEX friend_user_id_index ON cats.friend(user_id);
Analysis: 2.9% faster compared to default query, incorporated by optimizer
Reasoning: Do use this index. Index scanning allowed the query to run faster, and with larger data sets the performance should only increase. The where clause made this index optimal.

Tried: CREATE INDEX watch_user_id_index ON cats.watch(user_id);
Analysis: 4.9% faster compared to default query, incorporated by optimizer
Reasoning: Do use this index. Index scanning allowed the query to run faster, and with larger data sets the performance should only increase. The where clause made this index optimal.

Tried: CREATE INDEX watch_video_id_index ON cats.watch(video_id);
Analysis: 2.9% faster compared to default query, not incorporated by optimizer
Reasoning: Do not use this index. Although this was contained in a where clause, the cost to index was higher than the cost to perform sequential scanning.

Tried: CREATE INDEX likes_user_id_index ON cats.likes(user_id);
Analysis: 4.4% faster compared to default query, incorporated by optimizer

Reasoning: Do use this index. Index scanning allowed the query to run faster, and with larger data sets the performance should only increase. The where clause made this index optimal.

Tried: CREATE INDEX likes_video_id_index ON cats.likes(video_id);
Analysis: 5.4% slower compared to default query, not incorporated by optimizer
Reasoning: Do not use this index. Although this was contained in a where clause, the cost to index was higher than the cost to perform sequential scanning.

Query 4: watch_user_id_index, likes_user_id_index, likes_video_id_index
Tried: CREATE INDEX watch_user_id_index ON cats.watch(user_id);
Analysis: 1.9% slower compared to default query, incorporated by optimizer
Reasoning: Do use this index. Although the performance time did not decrease, the optimizer used this index instead of a sequential scan of the watch table as this allowed for heap blocks to be created and condition checks. With larger data sets provided, performance differences should be more noticeable. The where clause made this index optimal.

Tried: CREATE INDEX watch_video_id_index ON cats.watch(video_id);
Analysis: 7.2% faster compared to default query, not incorporated by optimizer
Reasoning: Do not use this index. The index was not used by the optimizer.

Tried: CREATE INDEX likes_user_id_index ON cats.likes(user_id);
Analysis: 10.6% slower compared to default query, incorporated by optimizer
Reasoning: Do use this index. Although the query performance did not become better, the optimizer included the index and with larger data sets the performance should increase.

Tried: CREATE INDEX likes_video_id_index ON cats.likes(video_id);
Analysis: 5.8% faster compared to default query, incorporated by optimizer
Reasoning: Do use this index. The performance increased and the index was incorporated therefore the where clause was of assistance and allowed for better performance

Query 5: friend_user_id_index, watch_user_id_index, likes_user_id_index, likes_video_id_index
Tried: CREATE INDEX friend_user_id_index ON cats.friend(user_id);
Analysis: 8.5% slower compared to default query, incorporated by optimizer
Reasoning: Do use this index. Although the query performance did not become better, the optimizer included the index and with larger data sets the performance should increase.

Tried: CREATE INDEX watch_user_id_index ON cats.watch(user_id);
Analysis: 3% slower compared to default query, incorporated by optimizer
Reasoning: Do use this index. Although the query performance did not become better, the optimizer included the index and with larger data sets the performance should increase.

Tried: CREATE INDEX watch_video_id_index ON cats.watch(video_id);
Analysis: 3.5% slower compared to default query, not incorporated by optimizer
Reasoning: Do not use this index. The index was not used by the optimizer, despite being associated with a where clases.

Tried: CREATE INDEX likes_user_id_index ON cats.likes(user_id);
Analysis: 1% faster compared to default query, incorporated by optimizer
Reasoning: Do use this index. The performance increased and the index was incorporated therefore the where clause was of assistance and allowed for better performance

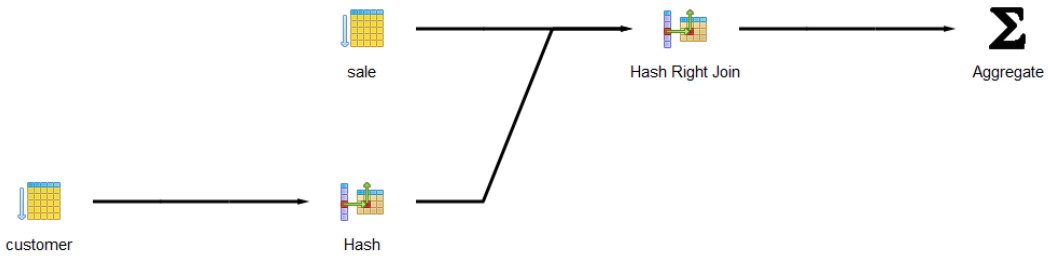
Tried: CREATE INDEX likes_video_id_index ON cats.likes(video_id
Analysis: 10% faster compared to default query, incorporated by optimizer
Reasoning: Do use this index. The performance increased and the index was incorporated therefore the where clause was of assistance and allowed for better performance

Appendix

Sales

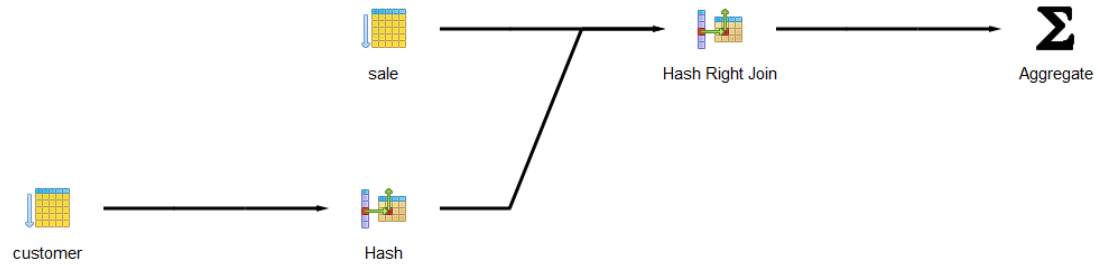
Query 1:

Sales_1
No Index



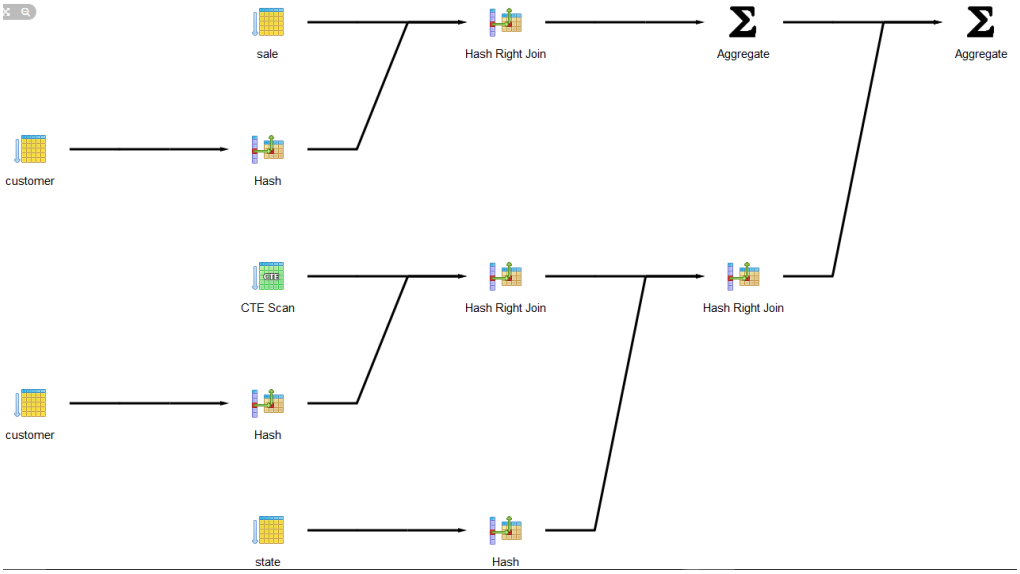
QUERY PLAN text
HashAggregate (cost=60.00..72.50 rows=1000 width=48) (actual time=1.430..1.528 rows=572 loops=1)
Group Key: c.customer_id, s.product_id
-> Hash Right Join (cost=14.25..45.00 rows=1000 width=17) (actual time=0.196..0.587 rows=1491 loops=1)
Hash Cond: (s.customer_id = c.customer_id)
-> Seq Scan on sale s (cost=0.00..17.00 rows=1000 width=17) (actual time=0.011..0.083 rows=1000 loops=1)
-> Hash (cost=8.00..8.00 rows=500 width=4) (actual time=0.172..0.172 rows=500 loops=1)
Buckets: 1024 Batches: 1 Memory Usage: 26kB
-> Seq Scan on customer c (cost=0.00..8.00 rows=500 width=4) (actual time=0.017..0.091 rows=500 loops=1)
Planning time: 0.453 ms
Execution time: 1.653 ms

CREATE INDEX sales_cust_id ON sales.sale(customer_id); -- No index needed



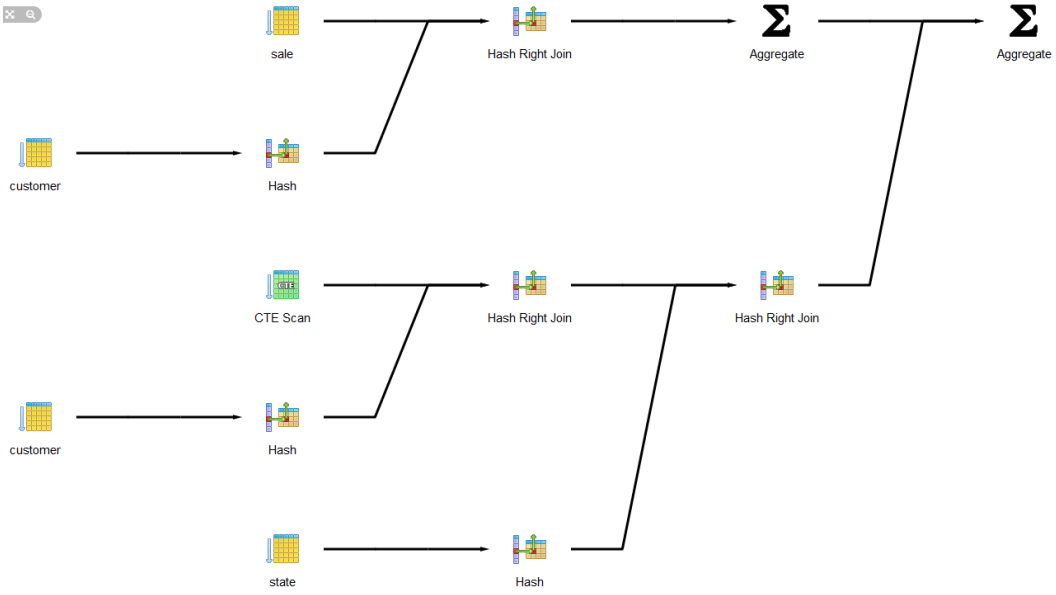
HashAggregate (cost=60.00..72.50 rows=1000 width=48) (actual time=1.348..1.454 rows=572 loops=1)
Group Key: c.customer_id, s.product_id
-> Hash Right Join (cost=14.25..45.00 rows=1000 width=17) (actual time=0.145..0.527 rows=1491 loops=1)
Hash Cond: (s.customer_id = c.customer_id)
-> Seq Scan on sale s (cost=0.00..17.00 rows=1000 width=17) (actual time=0.009..0.079 rows=1000 loops=1)
-> Hash (cost=8.00..8.00 rows=500 width=4) (actual time=0.127..0.127 rows=500 loops=1)
Buckets: 1024 Batches: 1 Memory Usage: 26kB
-> Seq Scan on customer c (cost=0.00..8.00 rows=500 width=4) (actual time=0.016..0.070 rows=500 loops=1)
Planning time: 0.516 ms
Execution time: 1.547 ms

Sales_2
No Index



HashAggregate (cost=127.95..136.05 rows=540 width=186) (actual time=3.685..3.710 rows=50 loops=1)
Group Key: s.state_id
CTE q1
-> HashAggregate (cost=57.50..63.75 rows=500 width=44) (actual time=2.155..2.317 rows=500 loops=1)
Group Key: c_1.customer_id
-> Hash Right Join (cost=14.25..45.00 rows=1000 width=13) (actual time=0.246..0.877 rows=1491 loops=1)
Hash Cond: (s_1.customer_id = c_1.customer_id)
-> Seq Scan on sale s_1 (cost=0.00..17.00 rows=1000 width=13) (actual time=0.012..0.124 rows=1000 loop...
-> Hash (cost=8.00..8.00 rows=500 width=4) (actual time=0.224..0.224 rows=500 loops=1)
Buckets: 1024 Batches: 1 Memory Usage: 26kB
-> Seq Scan on customer c_1 (cost=0.00..8.00 rows=500 width=4) (actual time=0.010..0.105 rows=500 loop...
-> Hash Right Join (cost=36.40..60.15 rows=540 width=162) (actual time=2.478..3.287 rows=501 loops=1)
Hash Cond: (c.state_id = s.state_id)
-> Hash Right Join (cost=14.25..31.13 rows=500 width=44) (actual time=2.421..3.037 rows=500 loops=1)
Hash Cond: (q.customer_id = c.customer_id)
-> CTE Scan on q1 q (cost=0.00..10.00 rows=500 width=44) (actual time=2.158..2.543 rows=500 loops=1)
-> Hash (cost=8.00..8.00 rows=500 width=8) (actual time=0.253..0.253 rows=500 loops=1)
Buckets: 1024 Batches: 1 Memory Usage: 28kB
-> Seq Scan on customer c (cost=0.00..8.00 rows=500 width=8) (actual time=0.011..0.118 rows=500 loops=1)
-> Hash (cost=15.40..15.40 rows=540 width=122) (actual time=0.045..0.045 rows=50 loops=1)
Buckets: 1024 Batches: 1 Memory Usage: 11kB
-> Seq Scan on state s (cost=0.00..15.40 rows=540 width=122) (actual time=0.016..0.023 rows=50 loops=1)
Planning time: 0.432 ms
Execution time: 3.939 ms

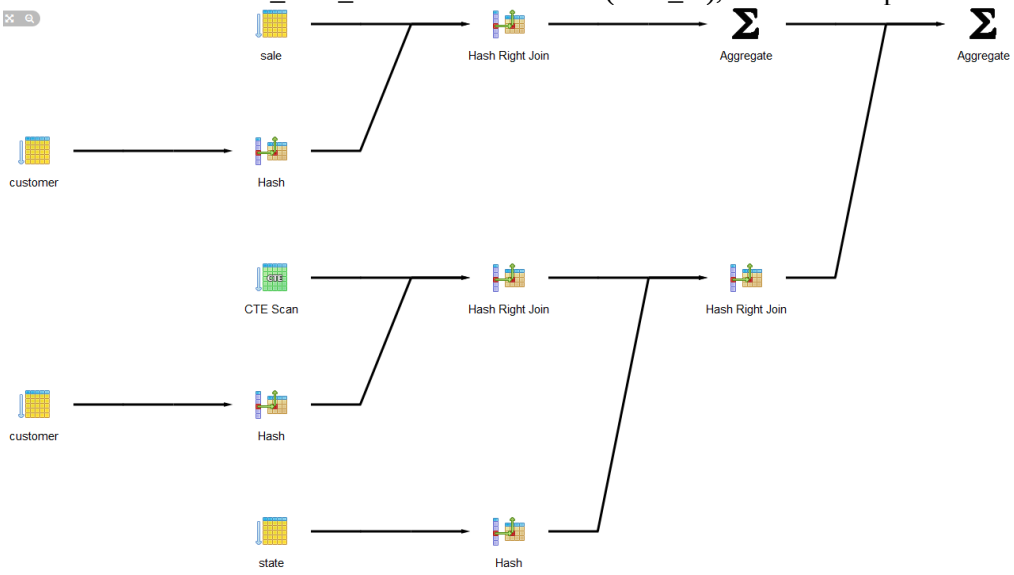
CREATE INDEX sales_cust_id ON sales.sale(customer_id); -- doesn't help



Orysya Stus
2.27.2017

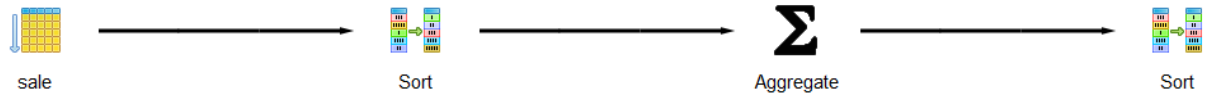
HashAggregate (cost=127.95..136.05 rows=540 width=186) (actual time=3.088..3.101 rows=50 loops=1)
Group Key: s.state_id
CTE q1
-> HashAggregate (cost=57.50..63.75 rows=500 width=44) (actual time=1.983..2.084 rows=500 loops=1)
Group Key: c_1.customer_id
-> Hash Right Join (cost=14.25..45.00 rows=1000 width=13) (actual time=0.310..0.938 rows=1491 loops=1)
Hash Cond: (s_1.customer_id = c_1.customer_id)
-> Seq Scan on sale s_1 (cost=0.00..17.00 rows=1000 width=13) (actual time=0.033..0.230 rows=1000 loops=1)
-> Hash (cost=8.00..8.00 rows=500 width=4) (actual time=0.247..0.247 rows=500 loops=1)
Buckets: 1024 Batches: 1 Memory Usage: 26kB
-> Seq Scan on customer c_1 (cost=0.00..8.00 rows=500 width=4) (actual time=0.023..0.124 rows=500 loops=1)
-> Hash Right Join (cost=36.40..60.15 rows=540 width=162) (actual time=2.372..2.853 rows=501 loops=1)
Hash Cond: (c.state_id = s.state_id)
-> Hash Right Join (cost=14.25..31.13 rows=500 width=44) (actual time=2.301..2.685 rows=500 loops=1)
Hash Cond: (q.customer_id = c.customer_id)
-> CTE Scan on q1 q (cost=0.00..10.00 rows=500 width=44) (actual time=1.987..2.223 rows=500 loops=1)
-> Hash (cost=8.00..8.00 rows=500 width=8) (actual time=0.295..0.295 rows=500 loops=1)
Buckets: 1024 Batches: 1 Memory Usage: 28kB
-> Seq Scan on customer c (cost=0.00..8.00 rows=500 width=8) (actual time=0.029..0.168 rows=500 loops=1)
-> Hash (cost=15.40..15.40 rows=540 width=122) (actual time=0.059..0.059 rows=50 loops=1)
Buckets: 1024 Batches: 1 Memory Usage: 11kB
-> Seq Scan on state s (cost=0.00..15.40 rows=540 width=122) (actual time=0.028..0.039 rows=50 loops=1)
Planning time: 0.647 ms
Execution time: 3.312 ms

CREATE INDEX cust_state_id ON sales.customer(state_id); -- doesn't help



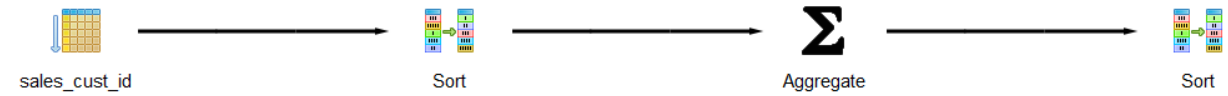
HashAggregate (cost=127.95..136.05 rows=540 width=186) (actual time=2.361..2.378 rows=50 loops=1)
Group Key: s.state_id
CTE q1
-> HashAggregate (cost=57.50..63.75 rows=500 width=44) (actual time=1.410..1.519 rows=500 loops=1)
Group Key: c_1.customer_id
-> Hash Right Join (cost=14.25..45.00 rows=1000 width=13) (actual time=0.243..0.614 rows=1491 loops=1)
Hash Cond: (s_1.customer_id = c_1.customer_id)
-> Seq Scan on sale s_1 (cost=0.00..17.00 rows=1000 width=13) (actual time=0.025..0.094 rows=1000 loops=1)
-> Hash (cost=8.00..8.00 rows=500 width=4) (actual time=0.202..0.202 rows=500 loops=1)
Buckets: 1024 Batches: 1 Memory Usage: 26kB
-> Seq Scan on customer c_1 (cost=0.00..8.00 rows=500 width=4) (actual time=0.014..0.099 rows=500 loops=1)
-> Hash Right Join (cost=36.40..60.15 rows=540 width=162) (actual time=1.638..2.114 rows=501 loops=1)
Hash Cond: (c.state_id = s.state_id)
-> Hash Right Join (cost=14.25..31.13 rows=500 width=44) (actual time=1.587..1.967 rows=500 loops=1)
Hash Cond: (q.customer_id = c.customer_id)
-> CTE Scan on q1 q (cost=0.00..10.00 rows=500 width=44) (actual time=1.413..1.652 rows=500 loops=1)
-> Hash (cost=8.00..8.00 rows=500 width=8) (actual time=0.166..0.166 rows=500 loops=1)
Buckets: 1024 Batches: 1 Memory Usage: 28kB
-> Seq Scan on customer c (cost=0.00..8.00 rows=500 width=8) (actual time=0.010..0.082 rows=500 loops=1)
-> Hash (cost=15.40..15.40 rows=540 width=122) (actual time=0.040..0.040 rows=50 loops=1)
Buckets: 1024 Batches: 1 Memory Usage: 11kB
-> Seq Scan on state s (cost=0.00..15.40 rows=540 width=122) (actual time=0.020..0.026 rows=50 loops=1)
Planning time: 0.875 ms
Execution time: 2.595 ms

Sales_3
No Index



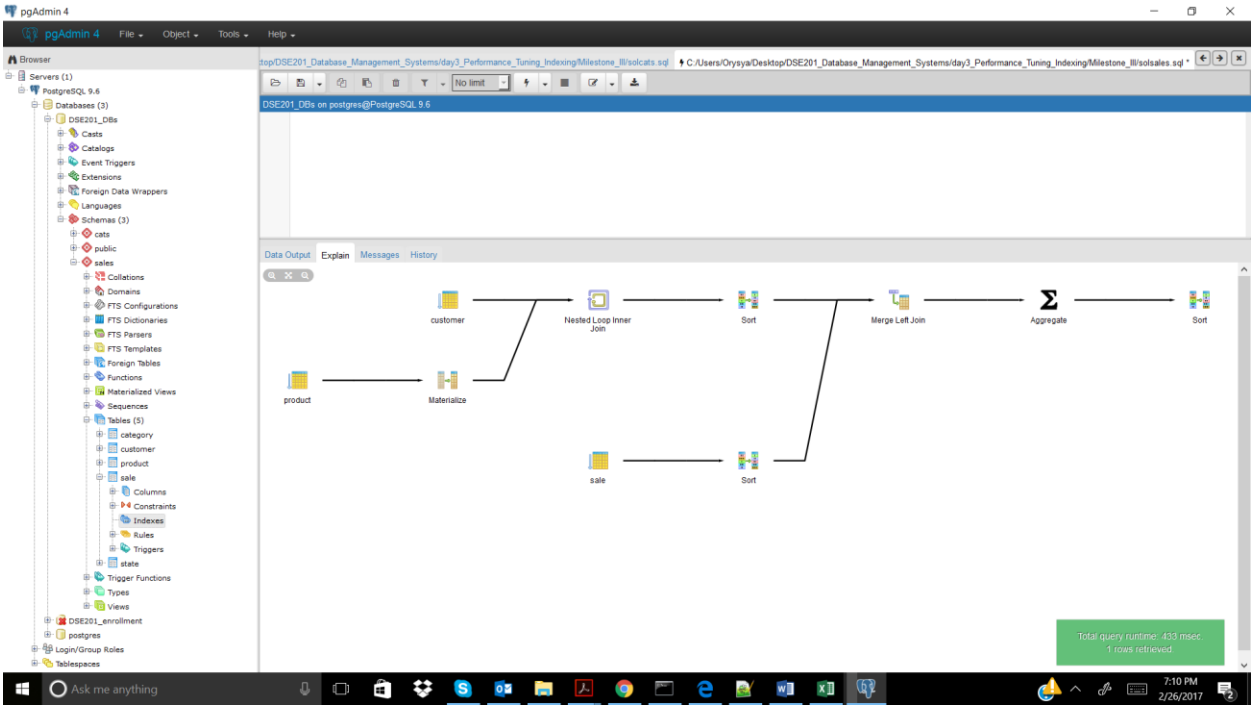
Sort (cost=19.55..19.56 rows=1 width=44) (actual time=0.262..0.262 rows=0 loops=1)
Sort Key: (sum(((quantity)::numeric * price))) DESC
Sort Method: quicksort Memory: 25kB
-> GroupAggregate (cost=19.51..19.54 rows=1 width=44) (actual time=0.229..0.229 rows=0 loops=1)
Group Key: product_id
-> Sort (cost=19.51..19.52 rows=1 width=13) (actual time=0.223..0.223 rows=0 loops=1)
Sort Key: product_id
Sort Method: quicksort Memory: 25kB
-> Seq Scan on sale (cost=0.00..19.50 rows=1 width=13) (actual time=0.205..0.205 rows=0 loops=1)
Filter: (customer_id = 13)
Rows Removed by Filter: 1000
Planning time: 0.495 ms
Execution time: 0.422 ms

CREATE INDEX sales_cust_id ON sales.sale(customer_id); -- this helped

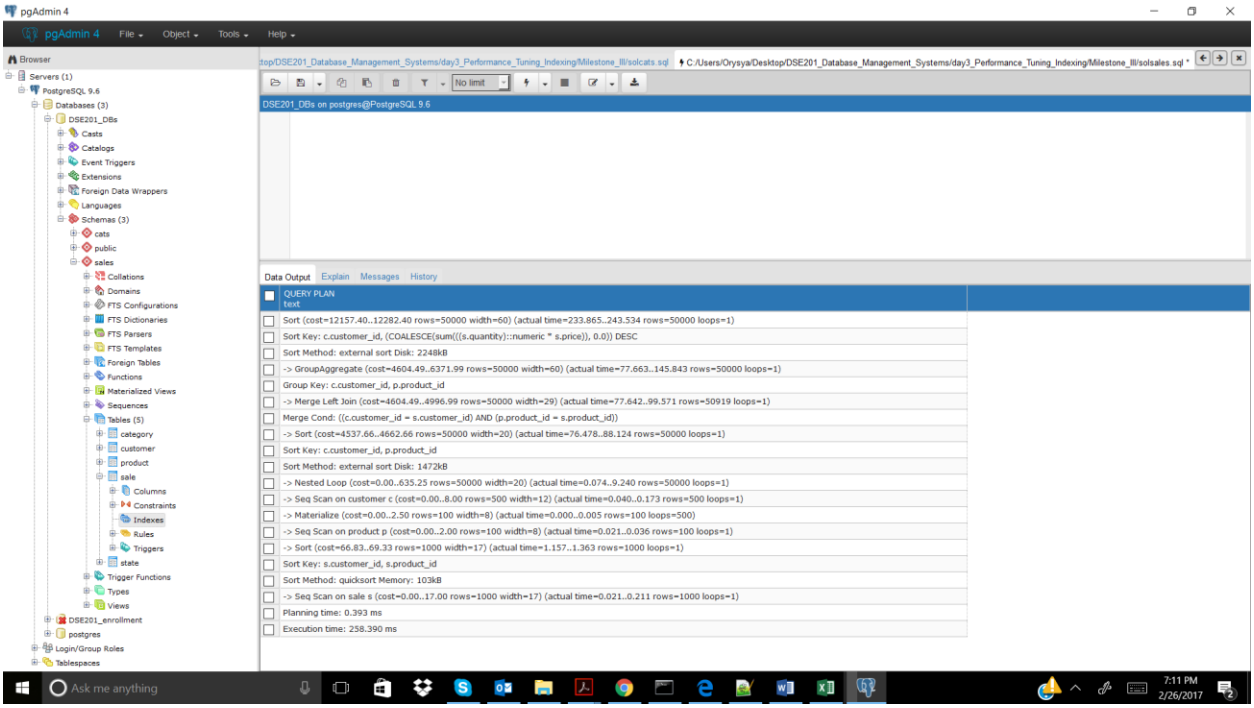


☐	Sort (cost=8.29..8.29 rows=1 width=44) (actual time=0.023..0.023 rows=0 loops=1)
☐	Sort Key: (sum(((quantity)::numeric * price))) DESC
☐	Sort Method: quicksort Memory: 25kB
☐	-> GroupAggregate (cost=8.25..8.28 rows=1 width=44) (actual time=0.017..0.017 rows=0 loops=1)
☐	Group Key: product_id
☐	-> Sort (cost=8.25..8.25 rows=1 width=13) (actual time=0.016..0.016 rows=0 loops=1)
☐	Sort Key: product_id
☐	Sort Method: quicksort Memory: 25kB
☐	-> Index Scan using sales_cust_id on sale (cost=0.28..8.24 rows=1 width=13) (actual time=0.011..0.011 rows=0 loops=1)
☐	Index Cond: (customer_id = 13)
☐	Planning time: 0.292 ms
☐	Execution time: 0.120 ms

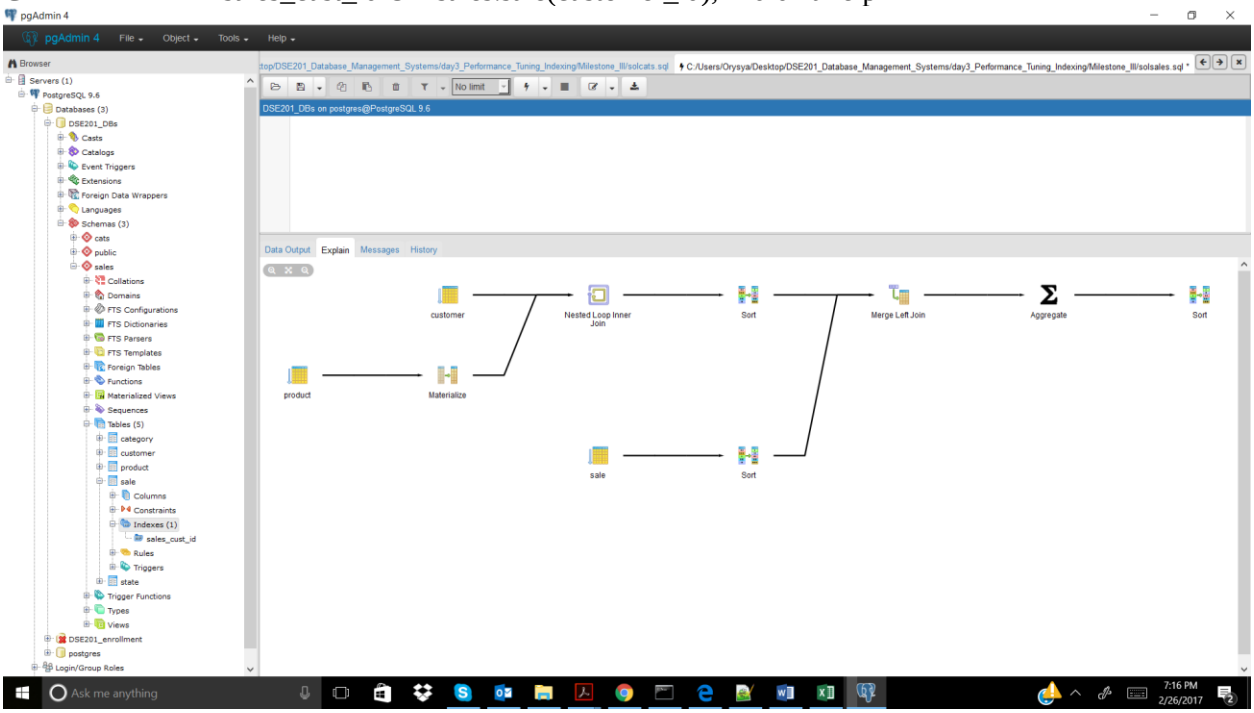
Sales_4
No Index



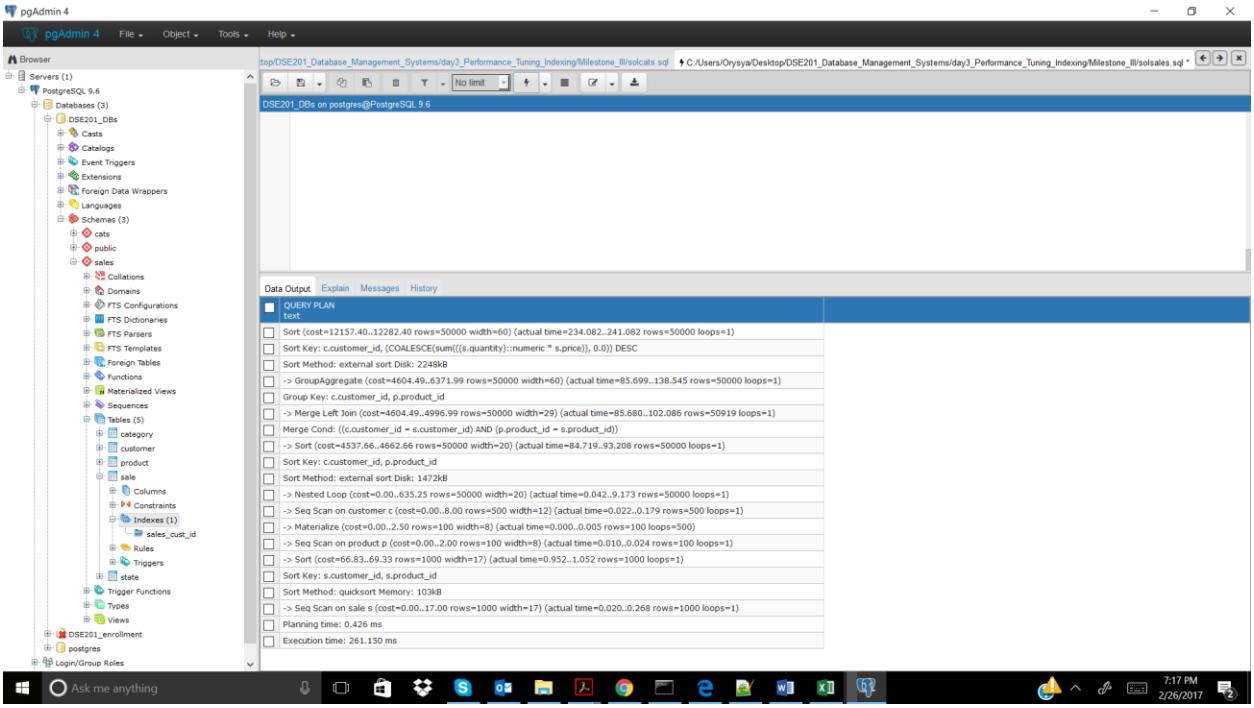
Orysya Stus
2.27.2017



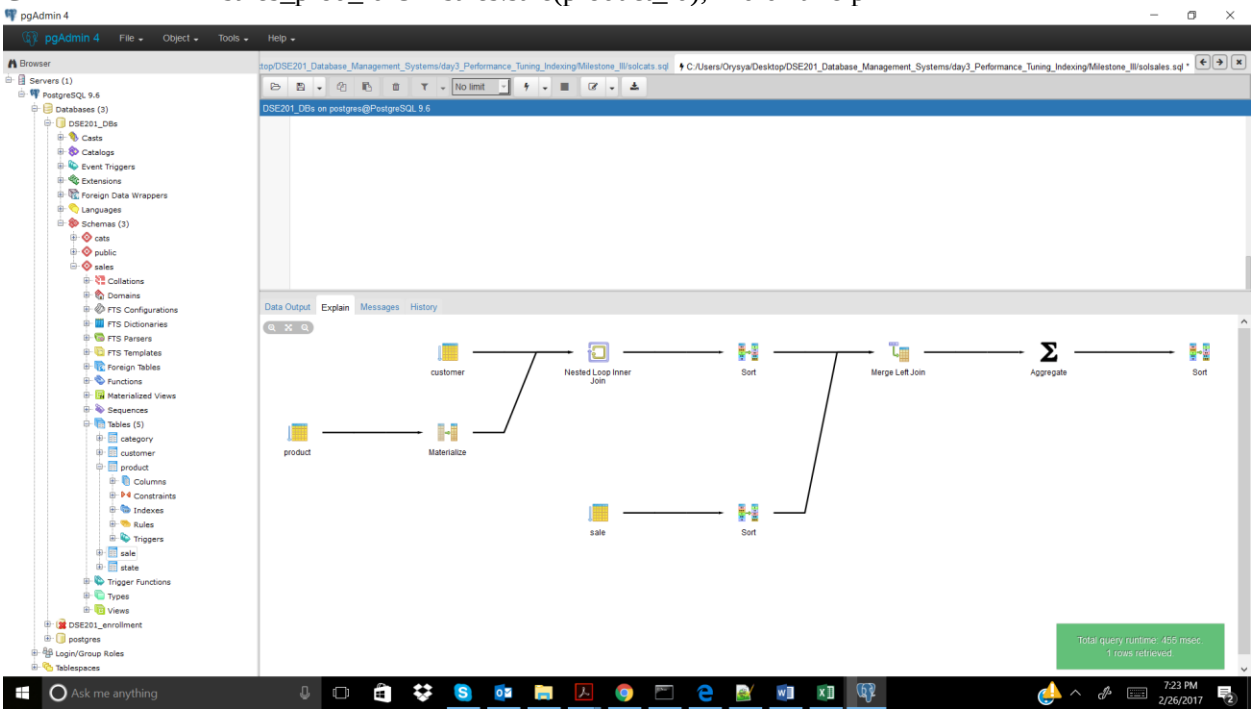
CREATE INDEX sales_cust_id ON sales.sale(customer_id); -- didn't help



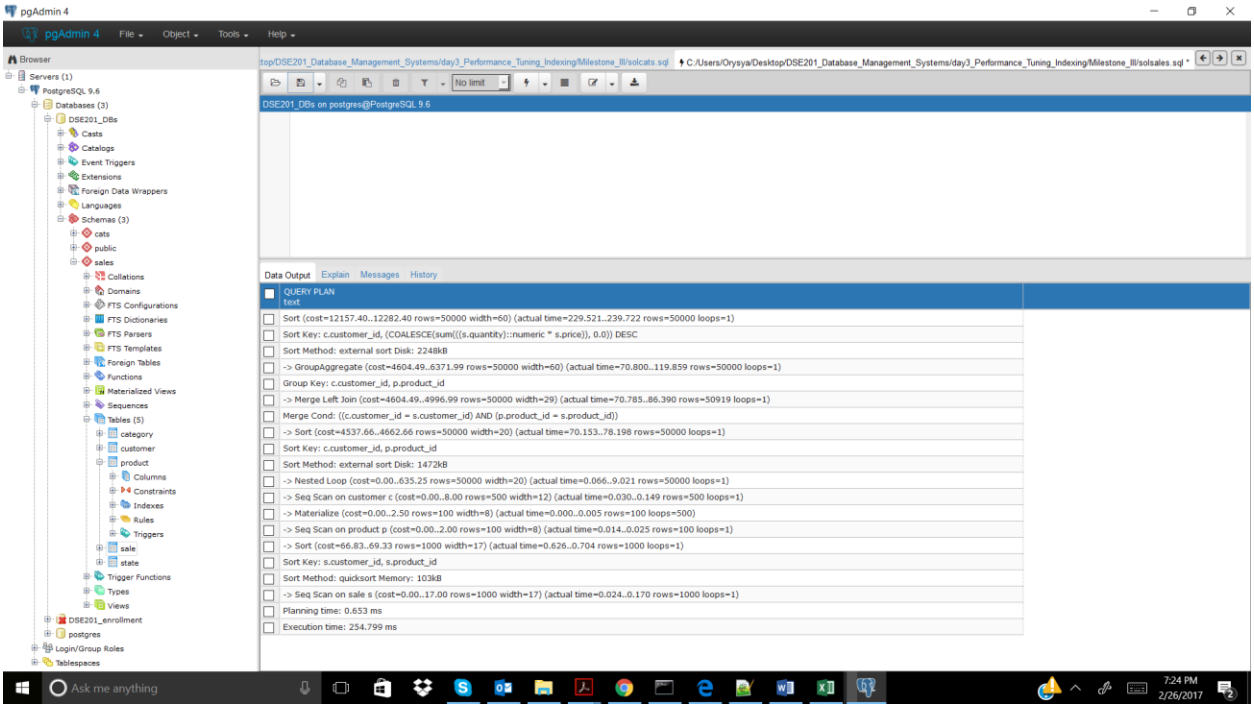
Orysya Stus
2.27.2017



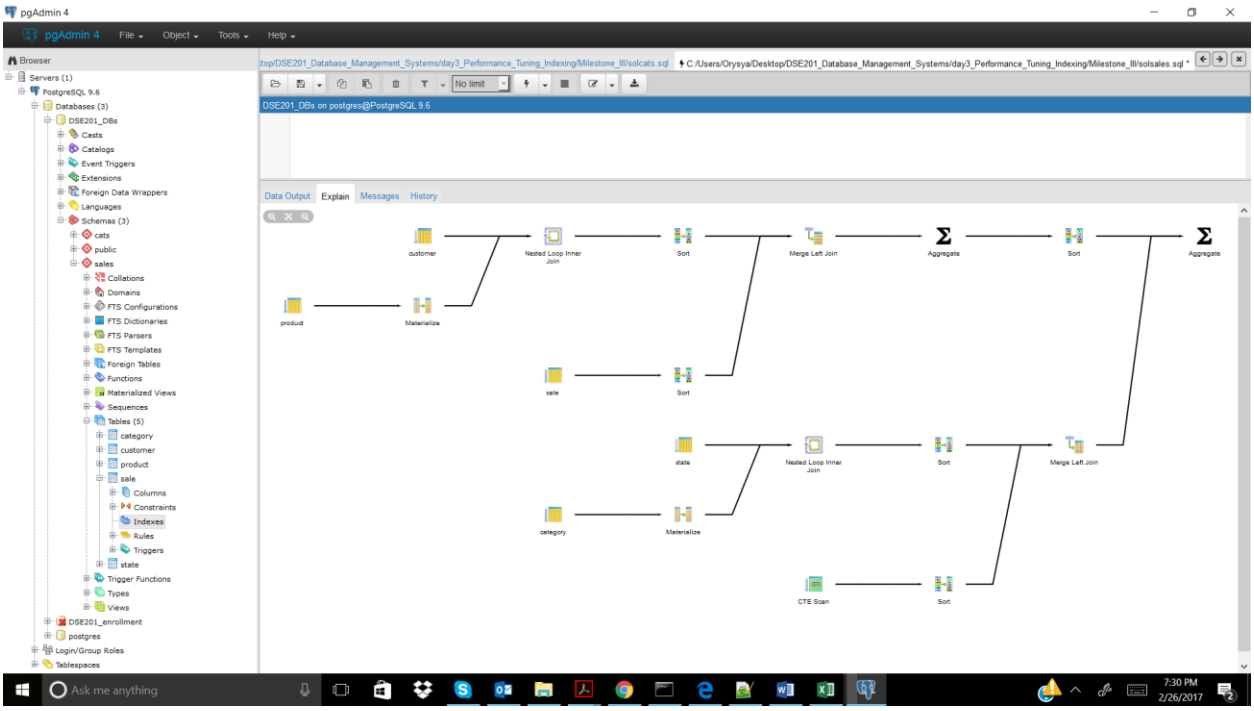
CREATE INDEX sales_prod_id ON sales.sale(product_id); -- didn't help



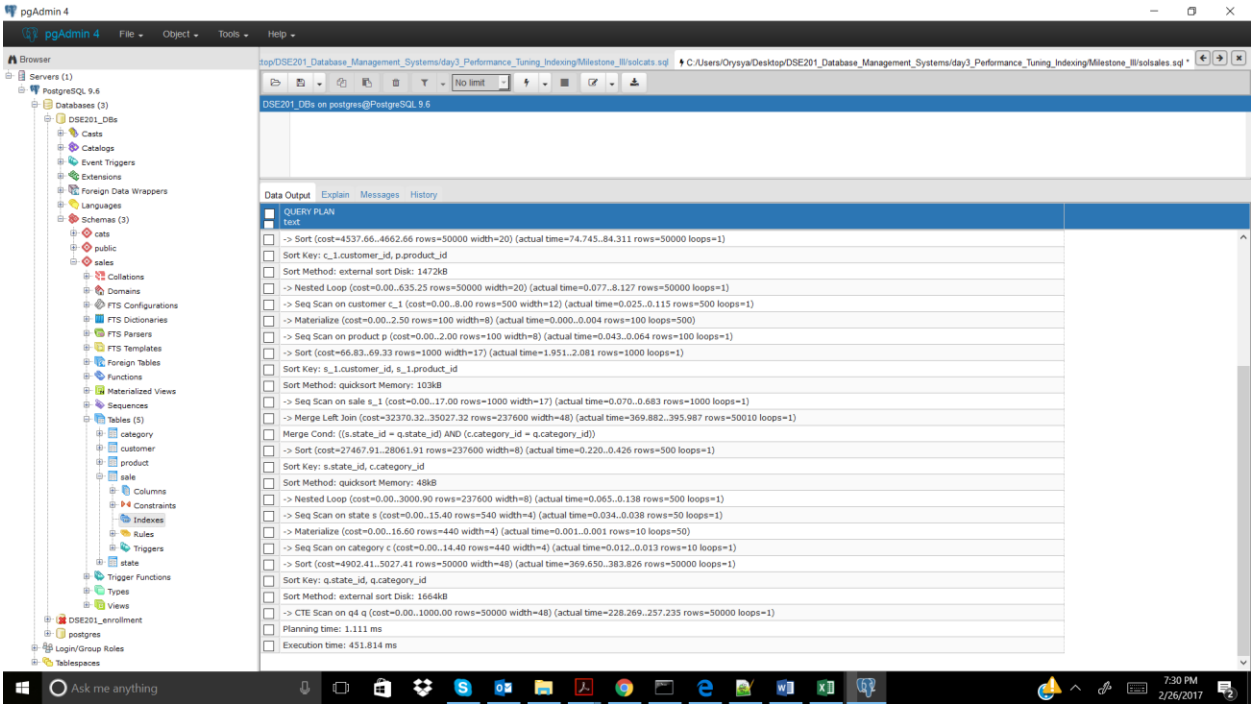
Orysya Stus
2.27.2017



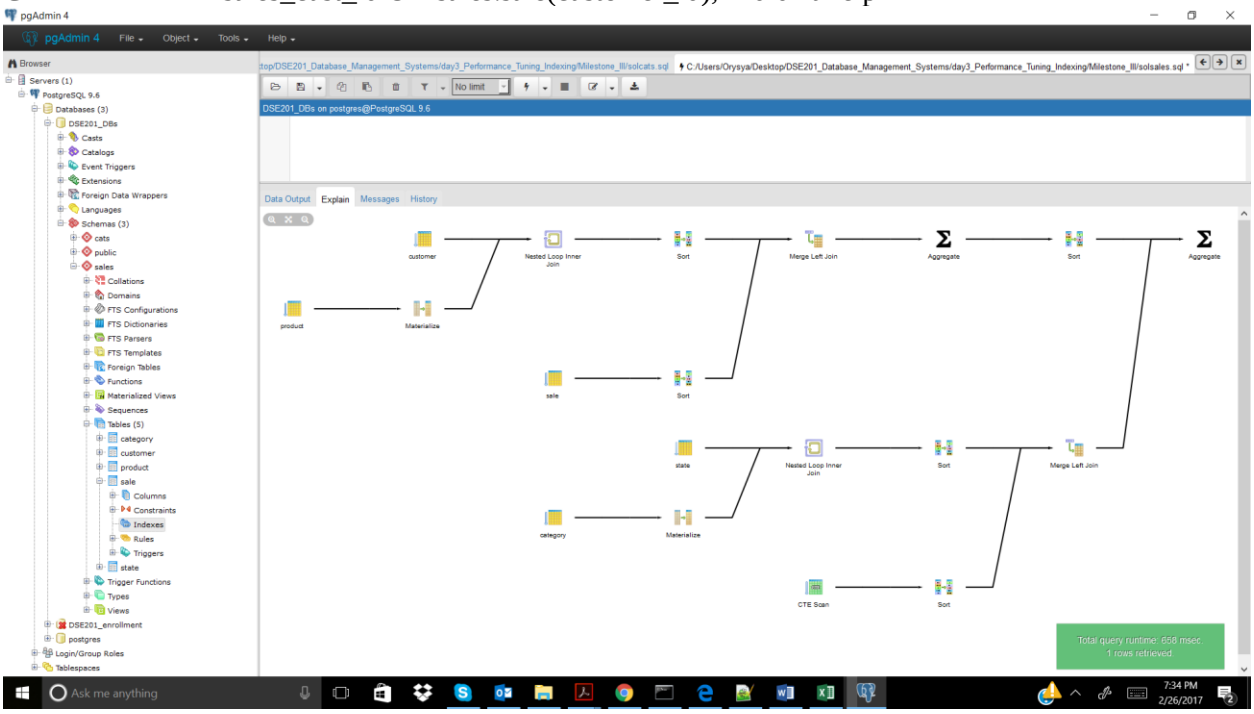
Sales_5
No Index



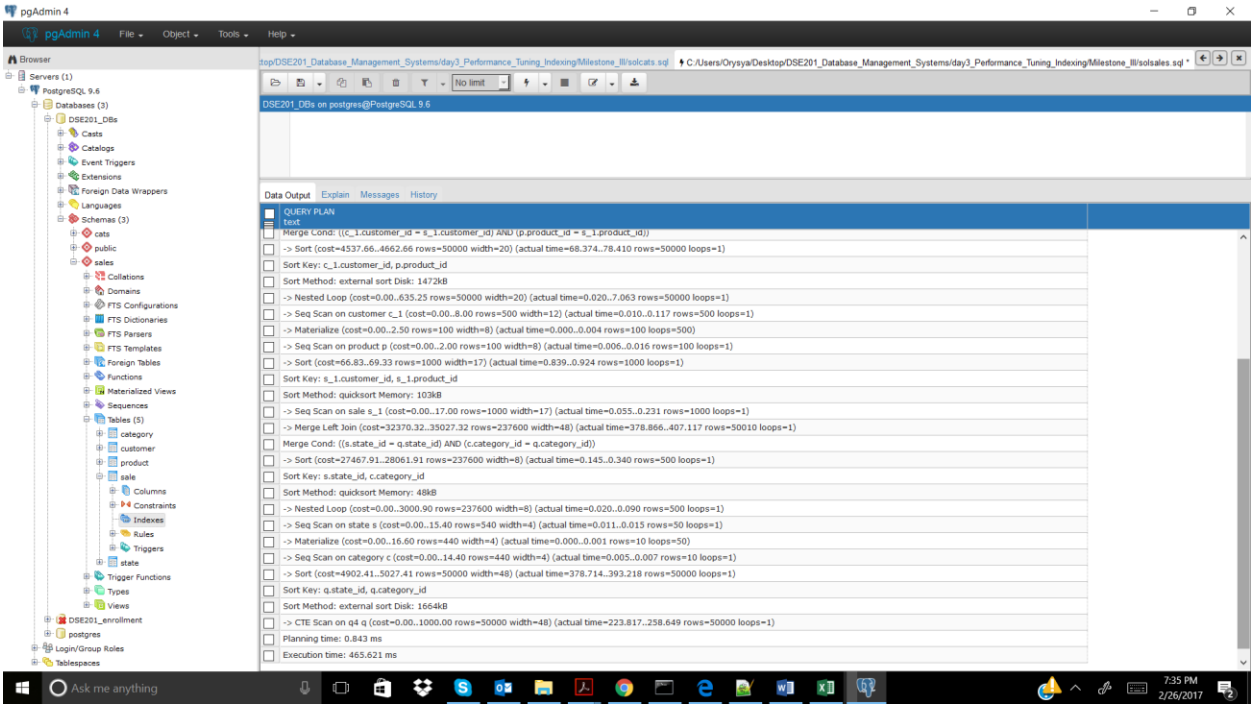
Orysya Stus
2.27.2017



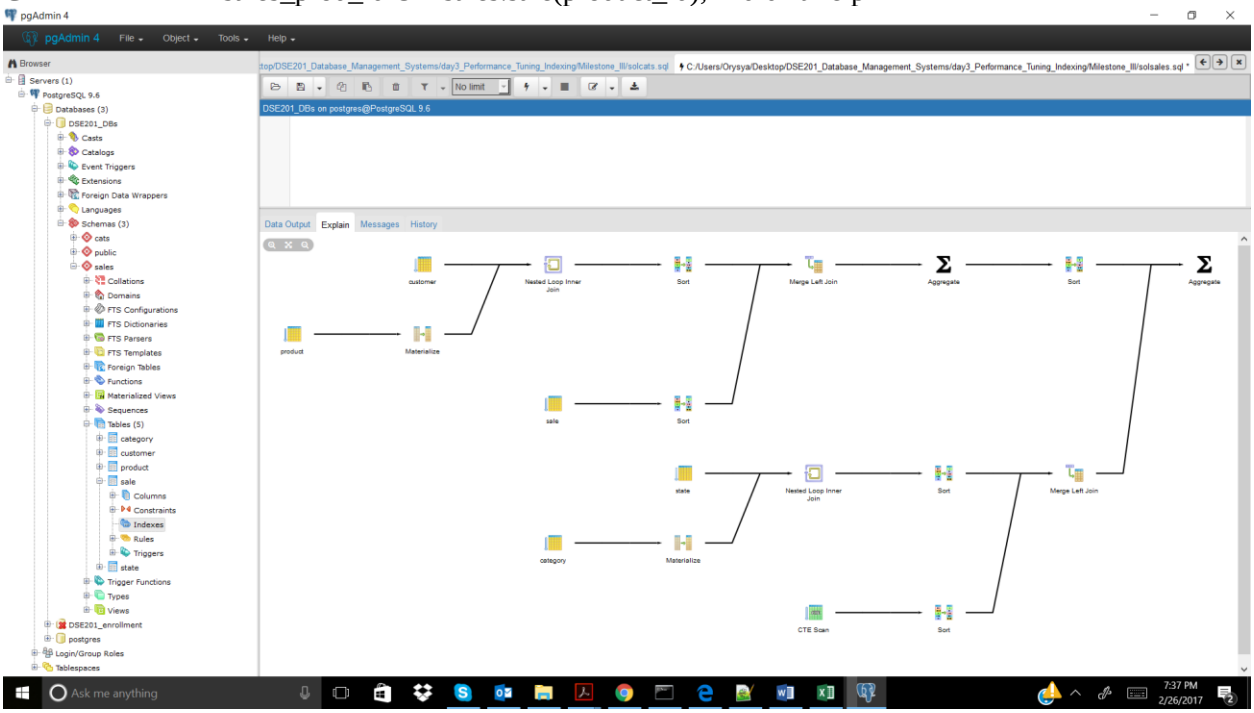
CREATE INDEX sales_cust_id ON sales.sale(customer_id); -- didn't help



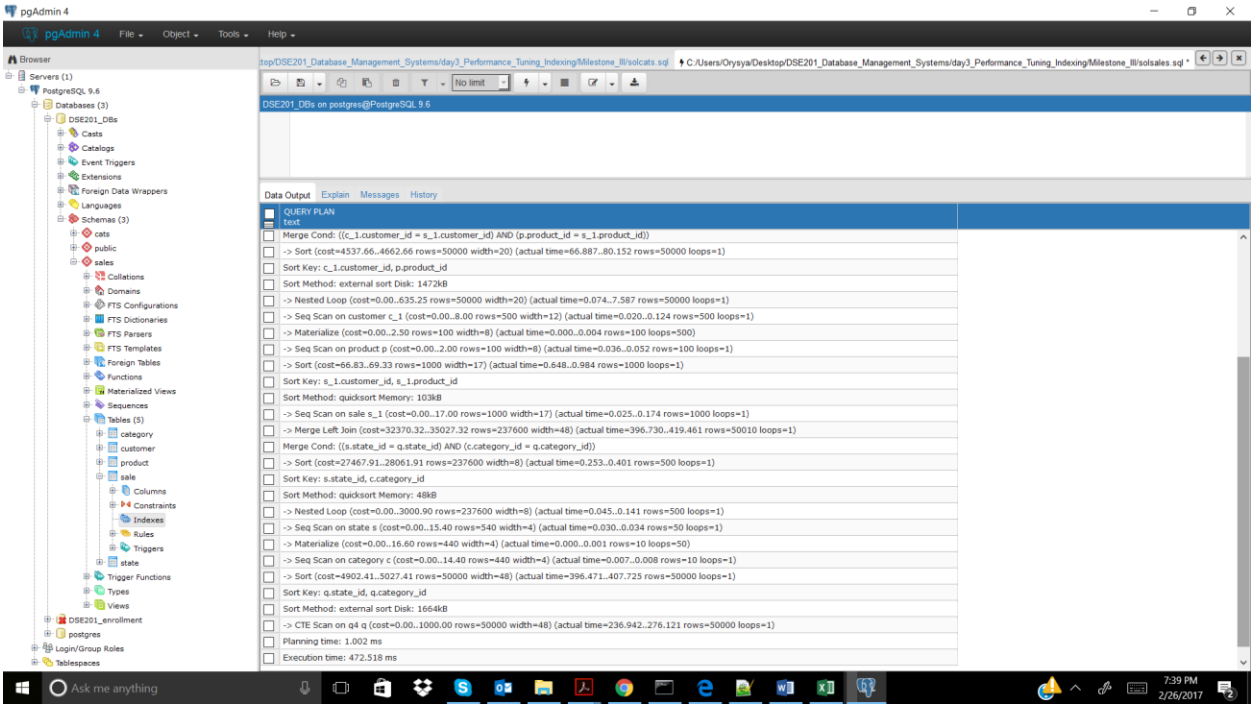
Orysya Stus
2.27.2017



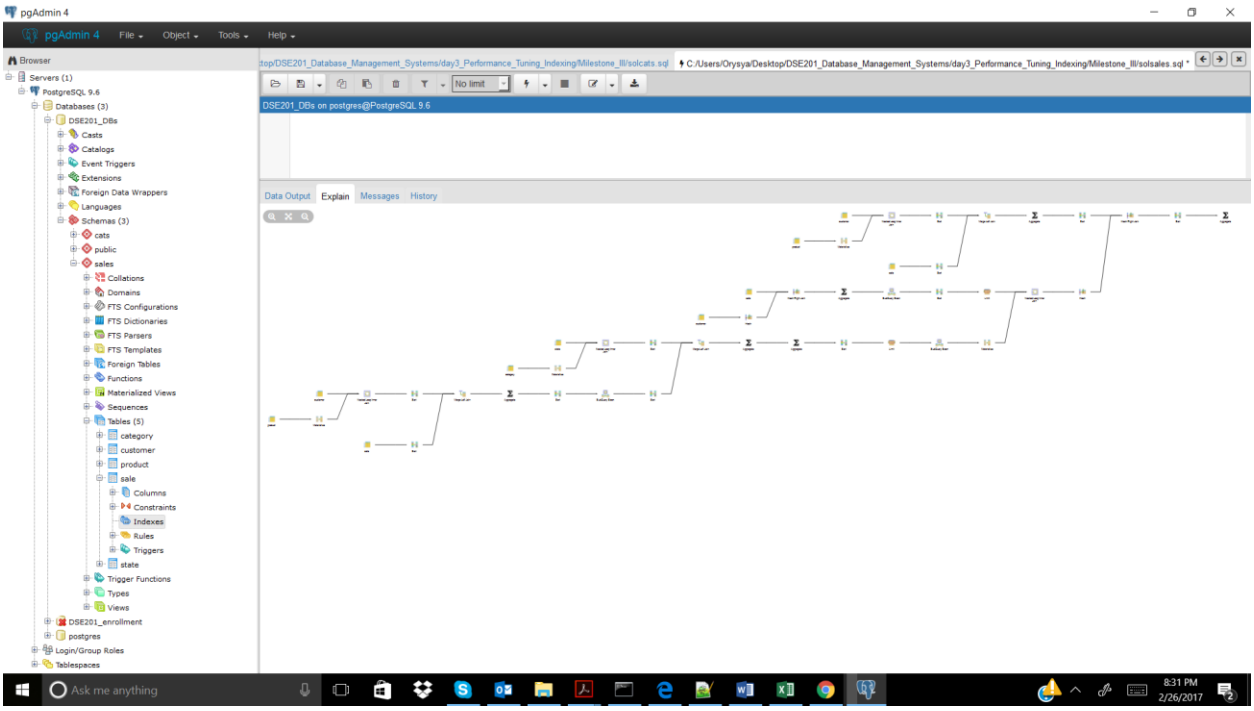
CREATE INDEX sales_prod_id ON sales.sale(product_id); -- didn't help



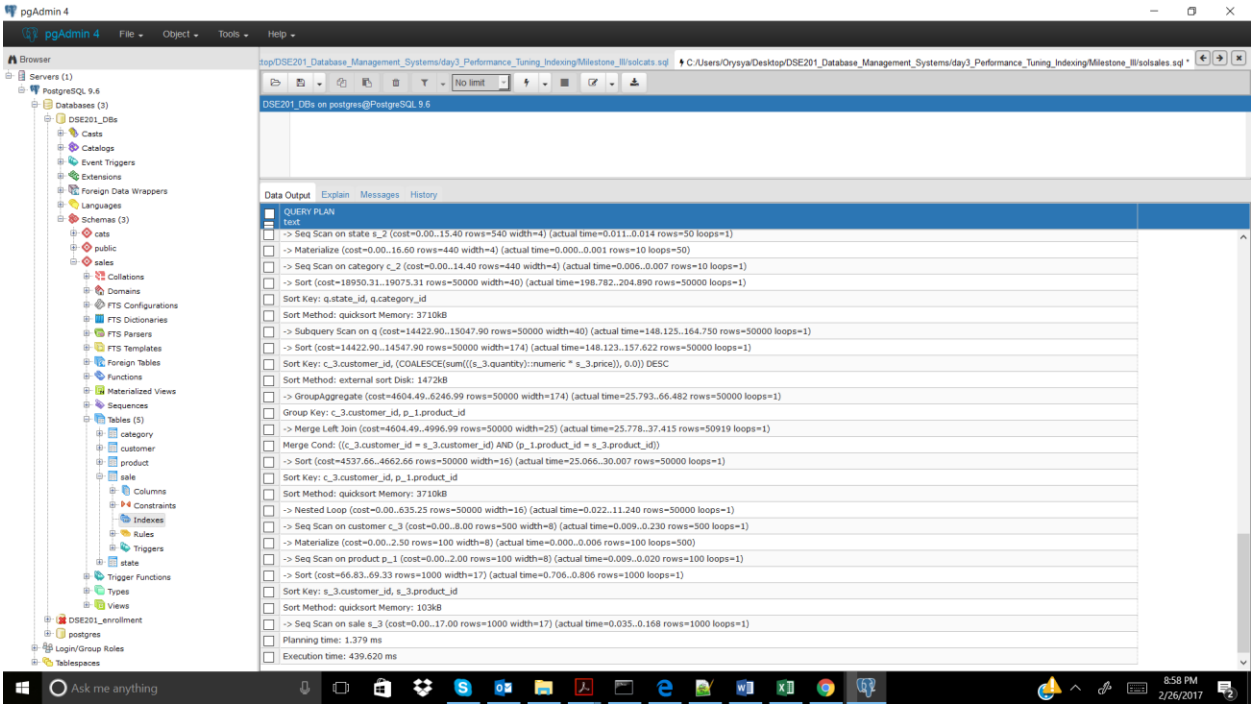
Orysya Stus
2.27.2017



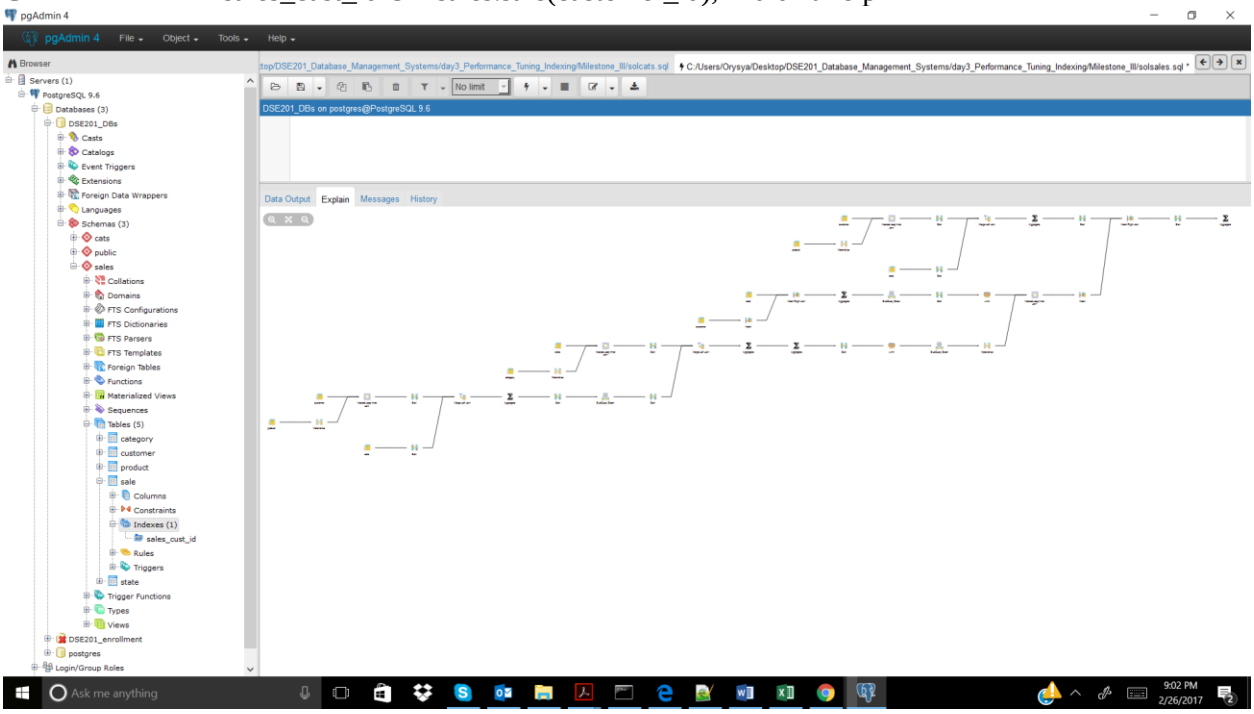
Sales_6
No Index



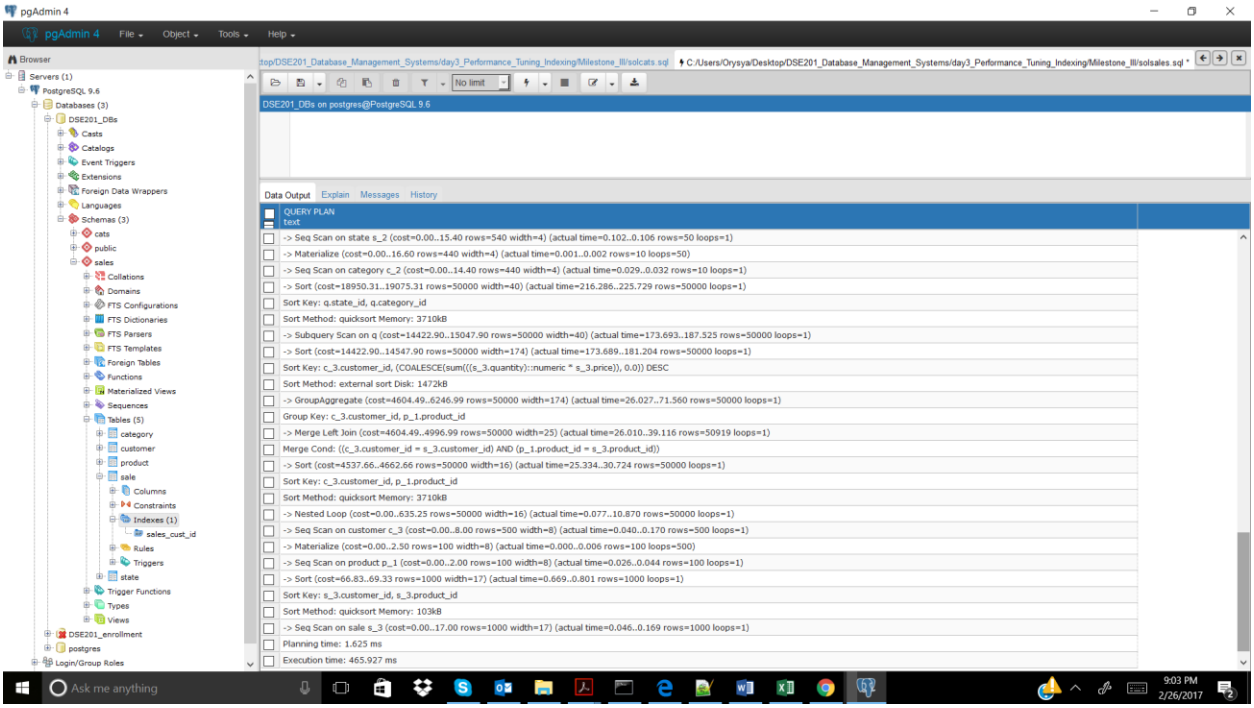
Orysya Stus
2.27.2017



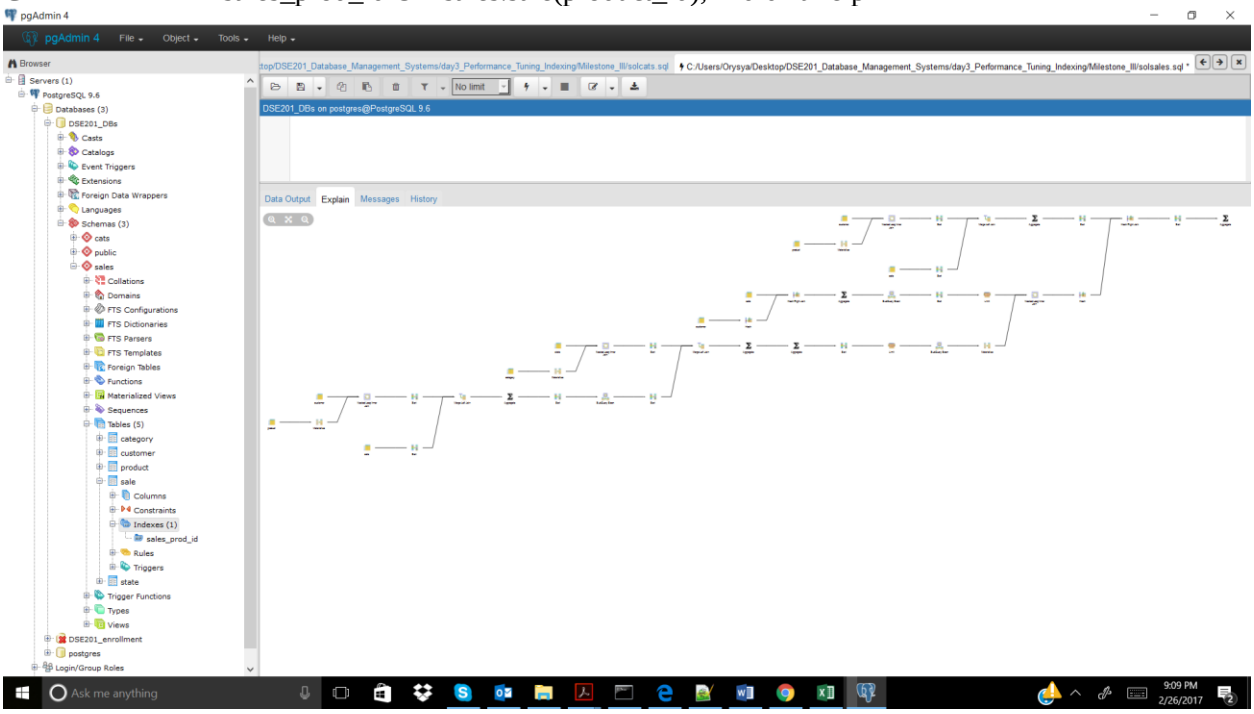
CREATE INDEX sales_cust_id ON sales.sale(customer_id); -- didn't help



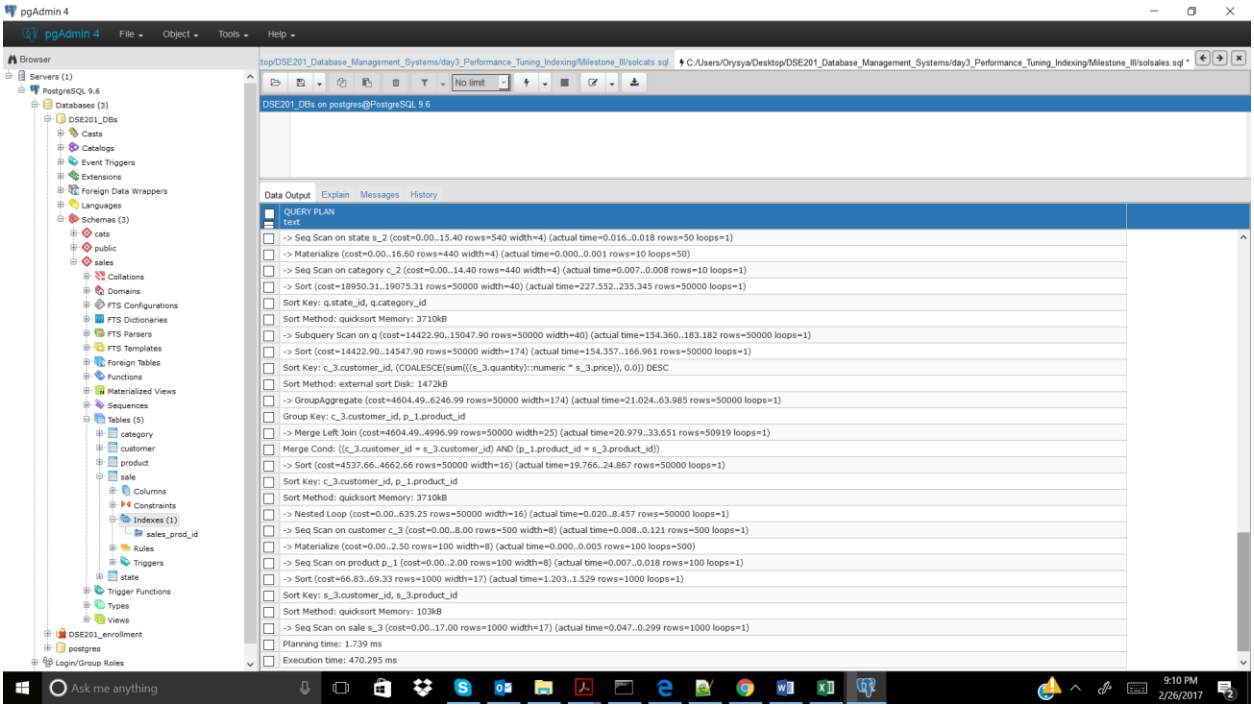
Orysya Stus
2.27.2017



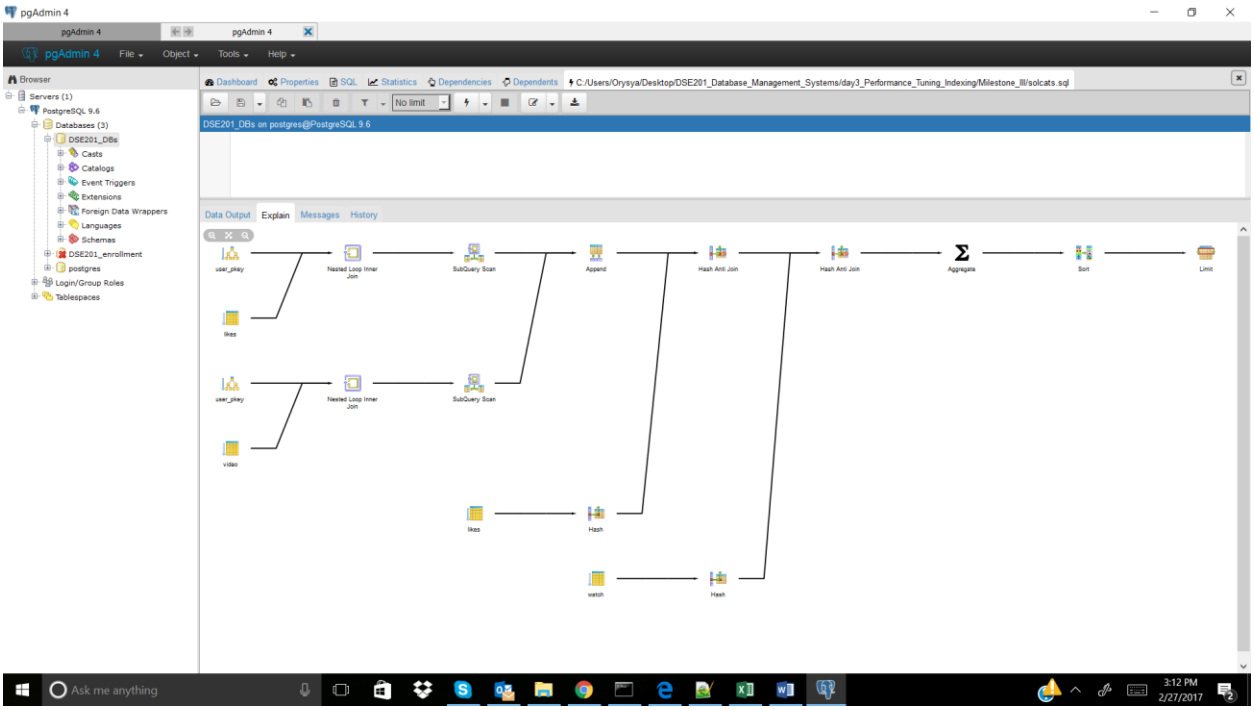
CREATE INDEX sales_prod_id ON sales.sale(product_id); -- didn't help



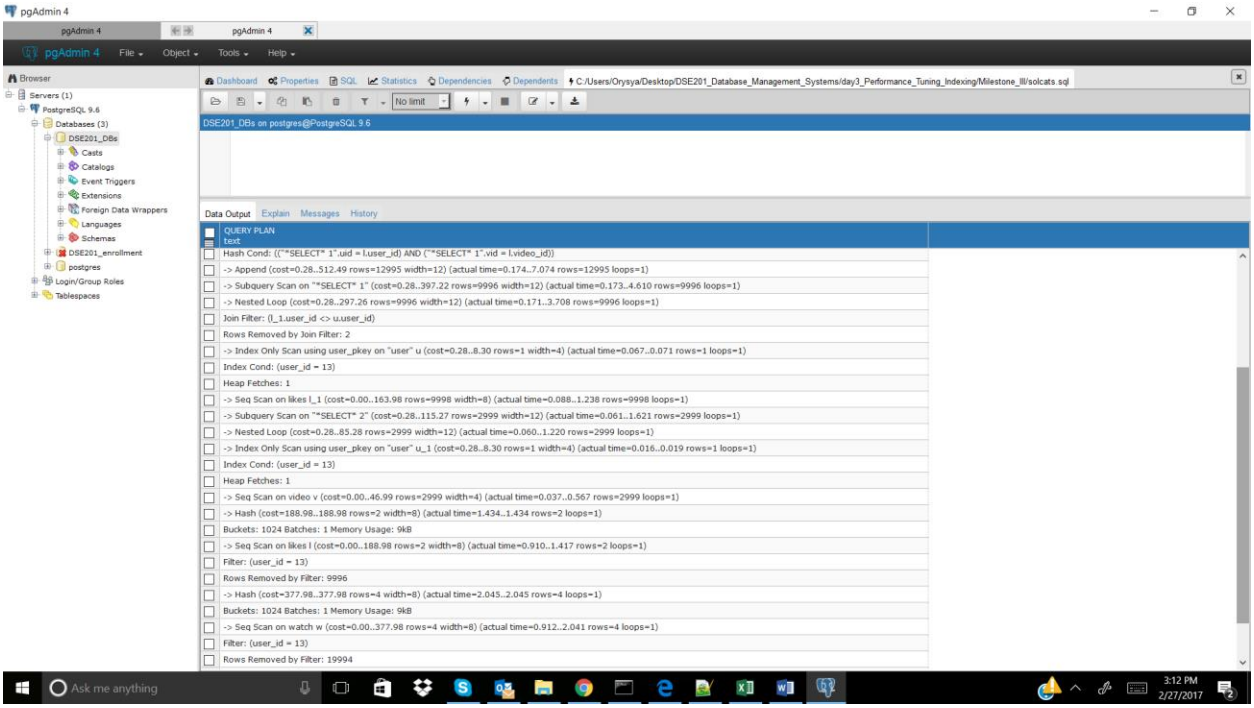
Orysya Stus
2.27.2017



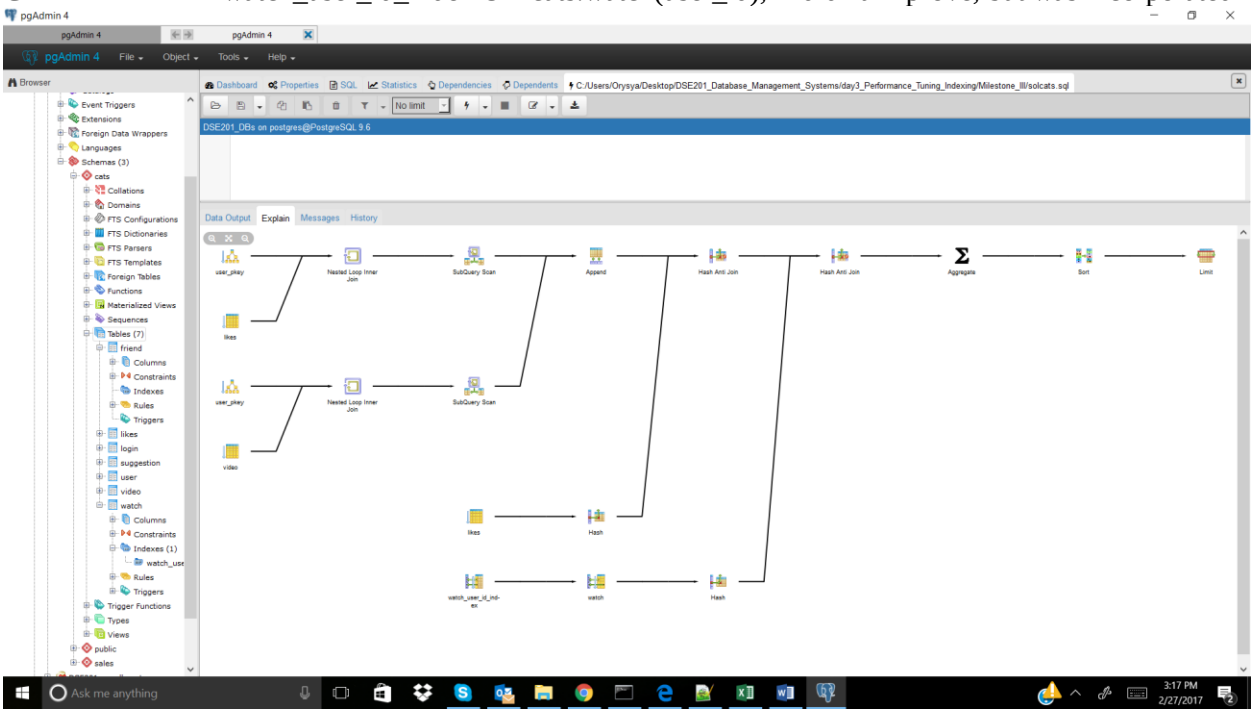
Cats_1
No Index



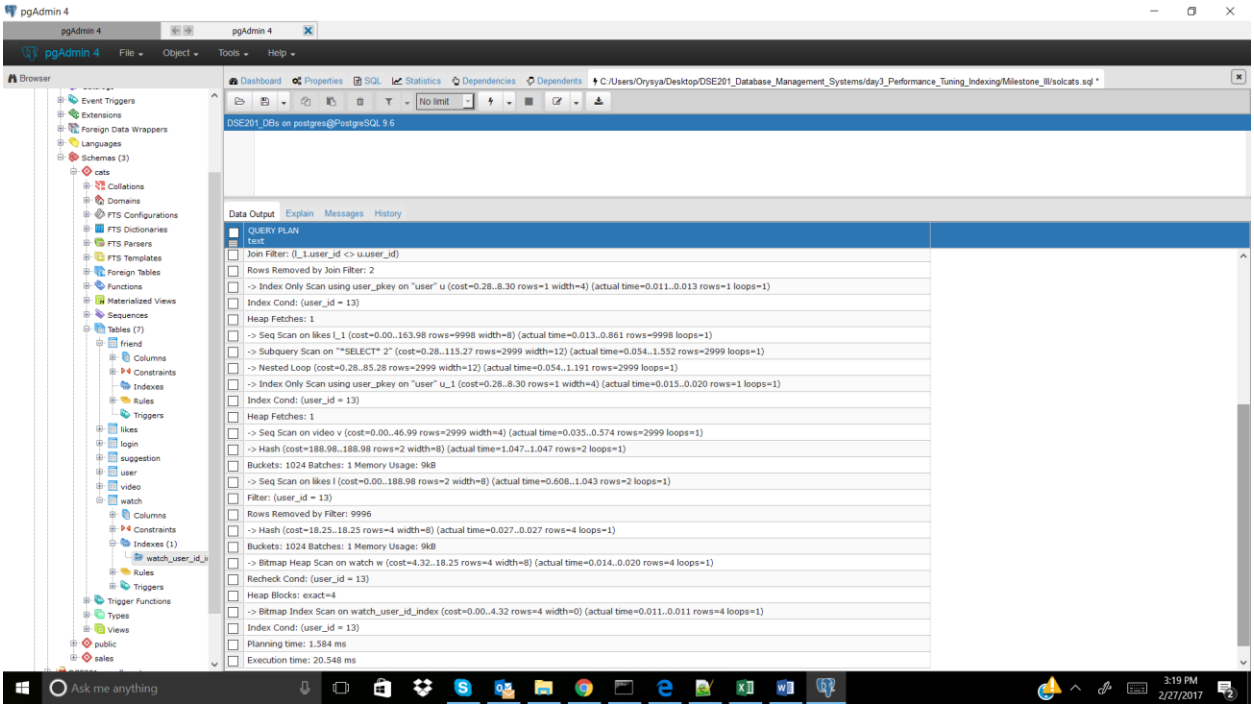
Orysya Stus
2.27.2017



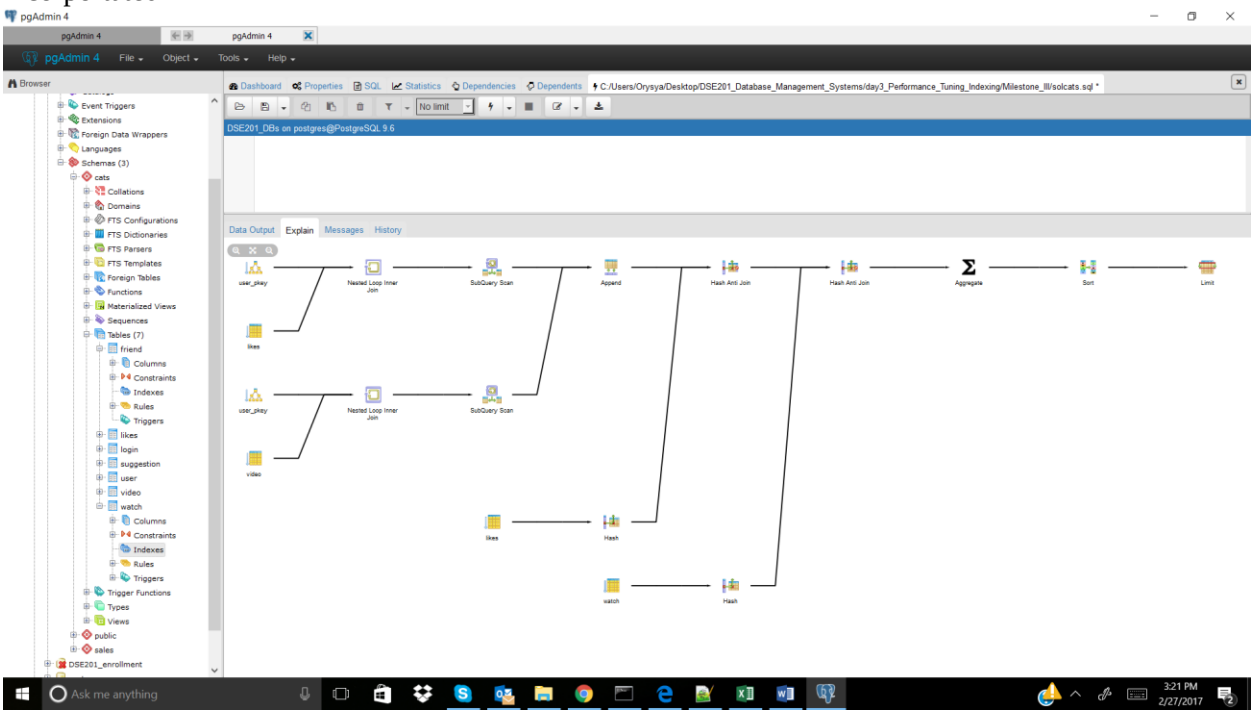
CREATE INDEX watch_user_id_index ON cats.watch(user_id); -- didn't improve, but was incorporated



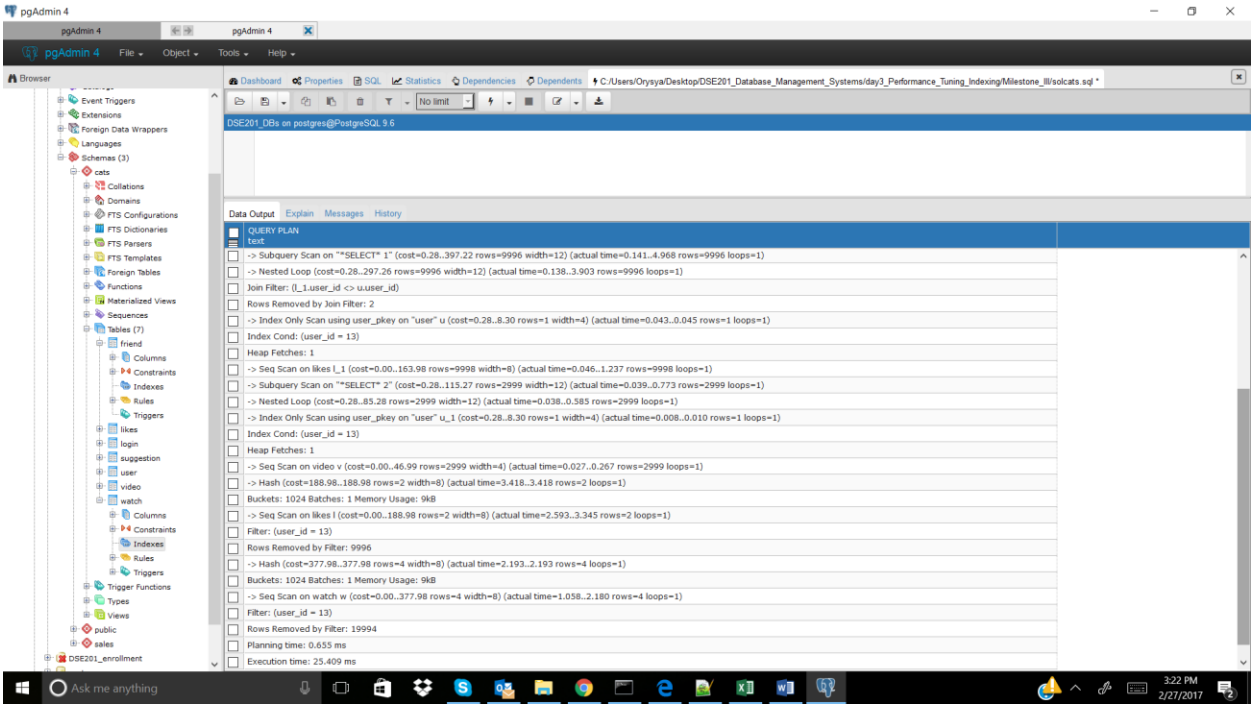
Orysya Stus
2.27.2017



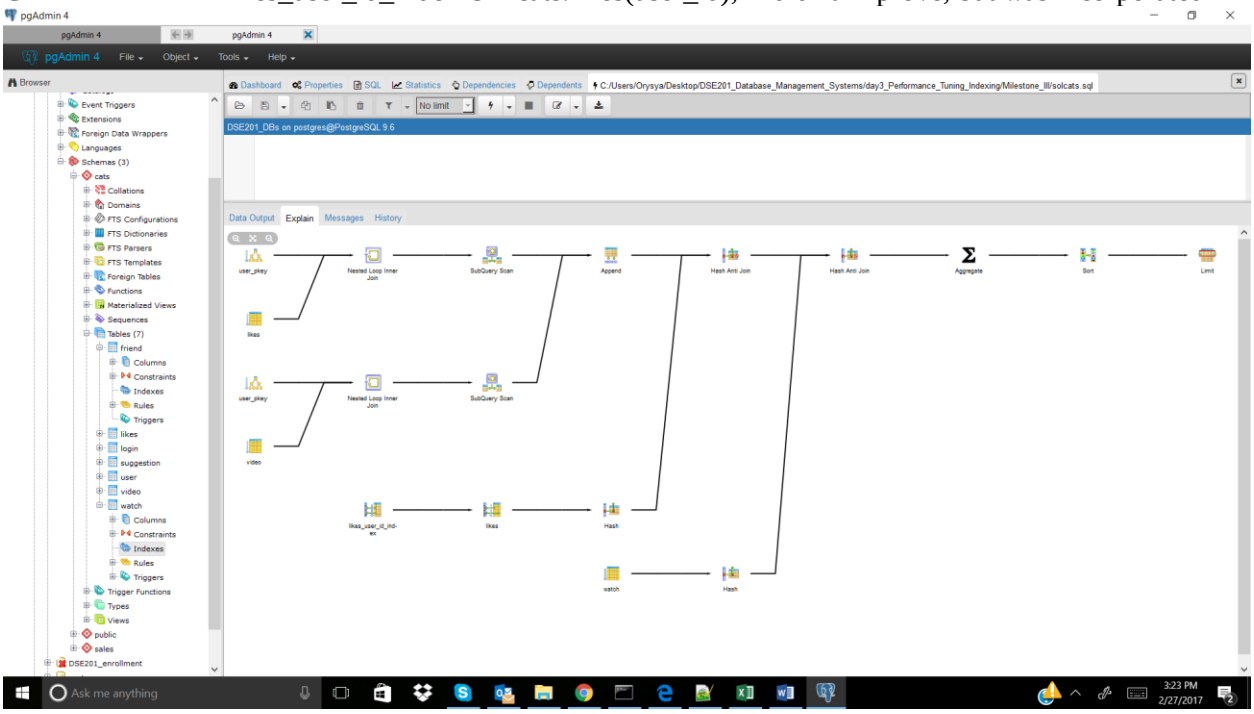
CREATE INDEX watch_video_id_index ON cats.watch(video_id); -- didn't improve, was not incorporated



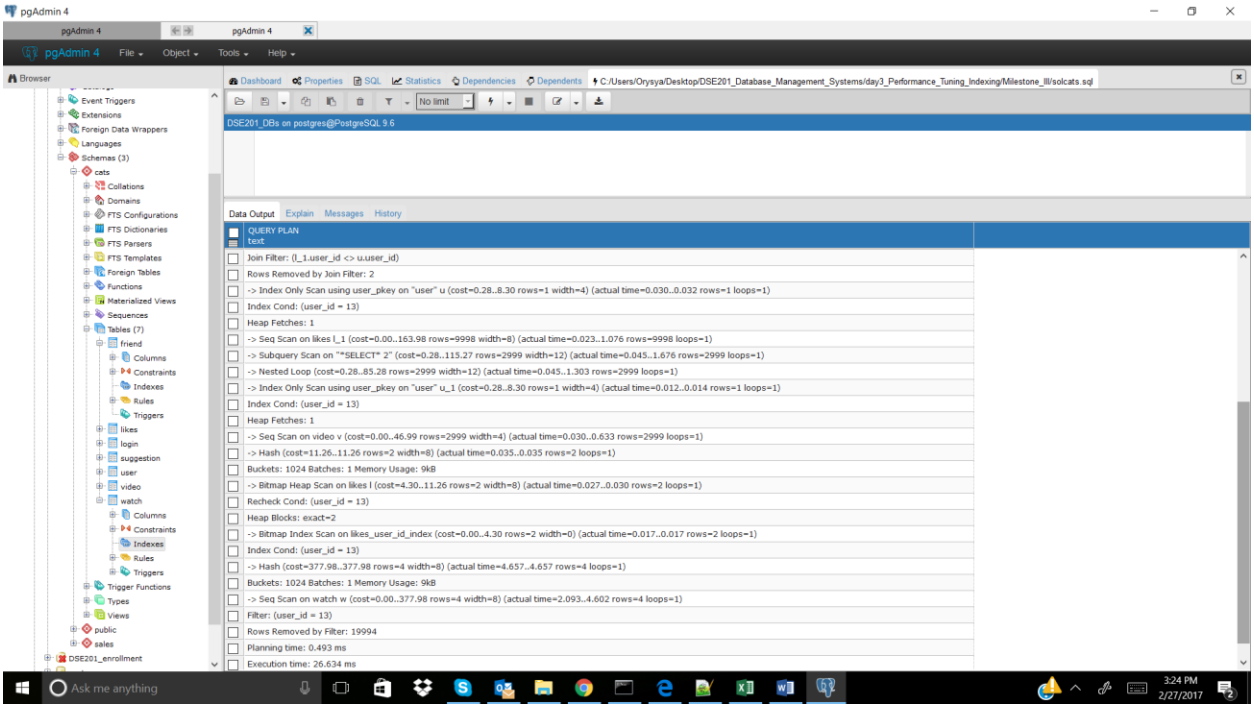
Orysya Stus
2.27.2017



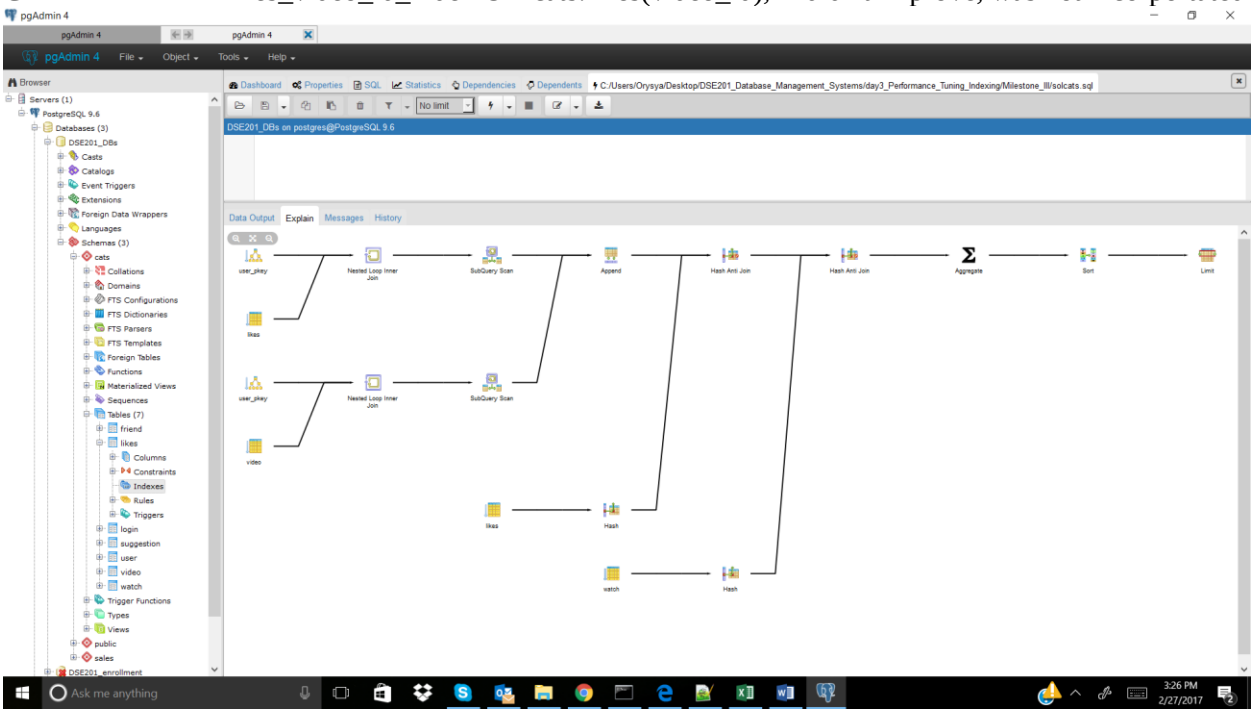
CREATE INDEX likes_user_id_index ON cats.likes(user_id); -- didn't improve, but was incorporated



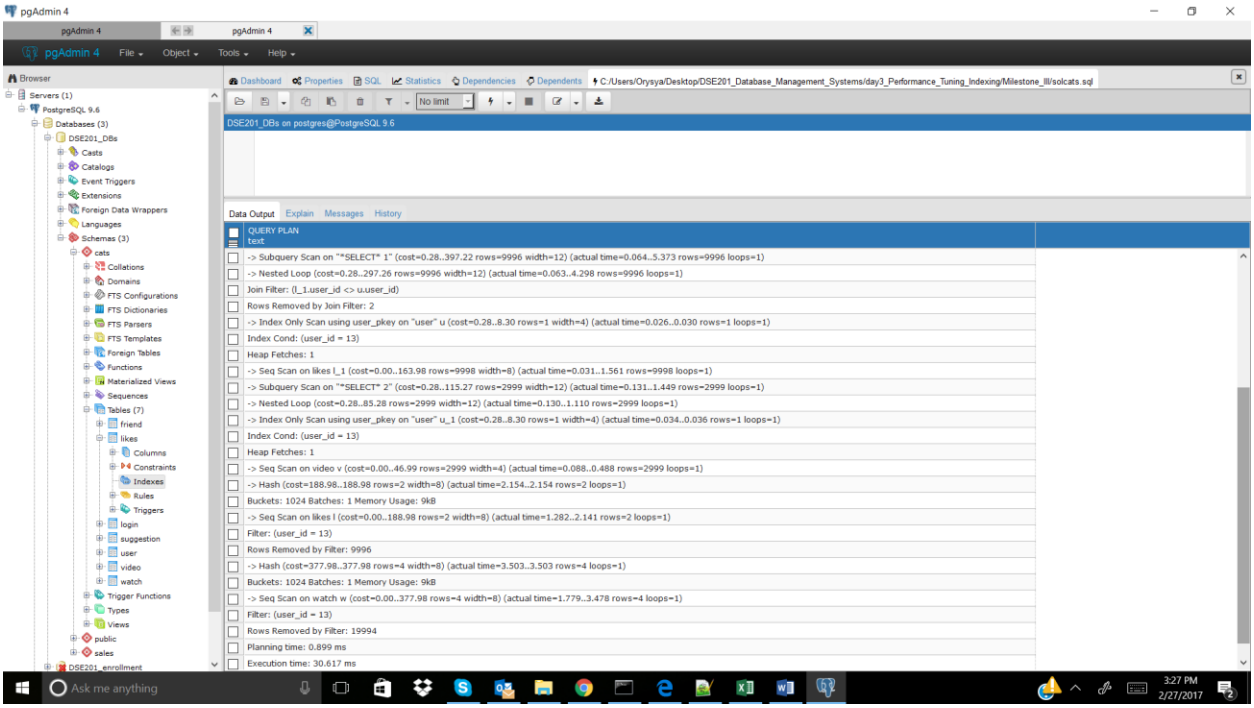
Orysya Stus
2.27.2017



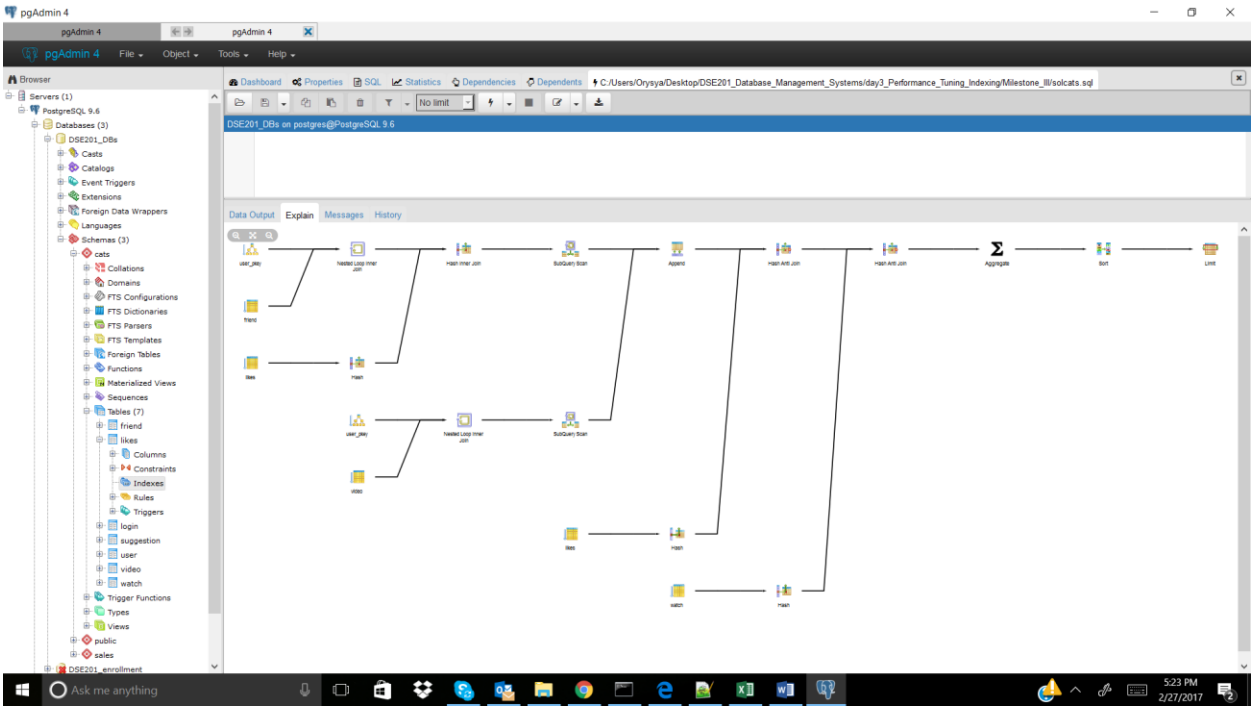
CREATE INDEX likes_video_id_index ON cats.likes(video_id); -- didn't improve, was not incorportated



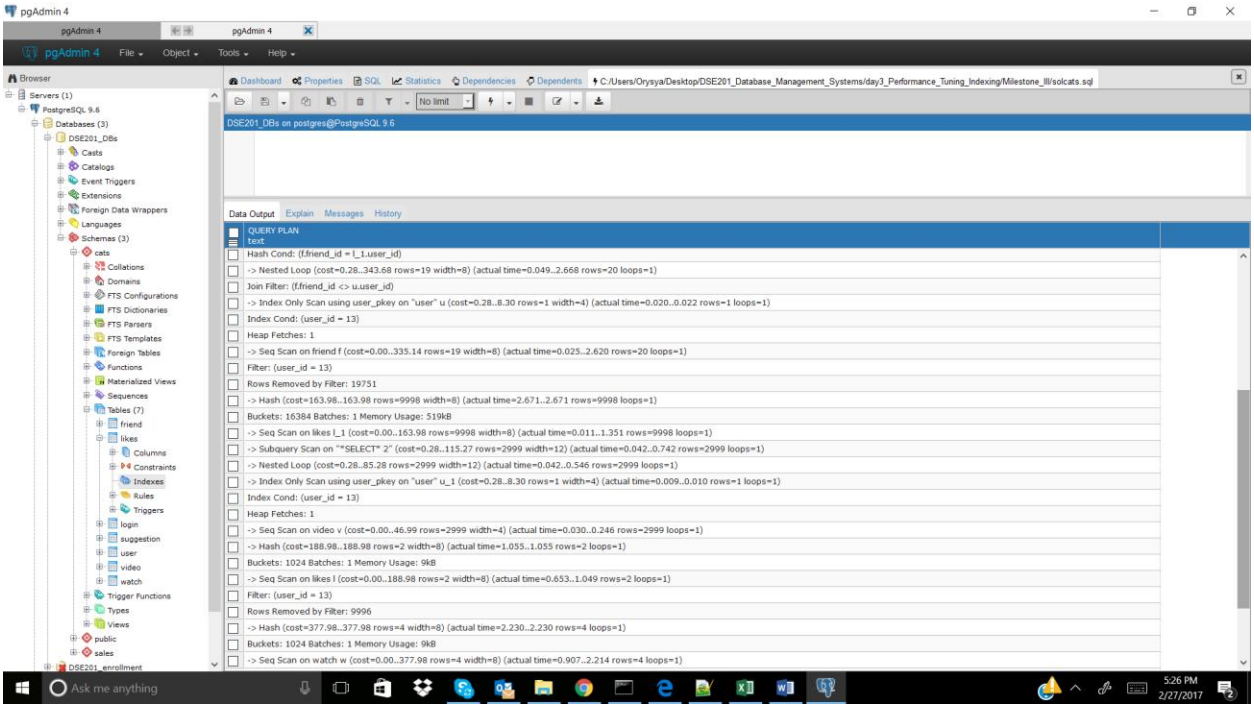
Orysya Stus
2.27.2017



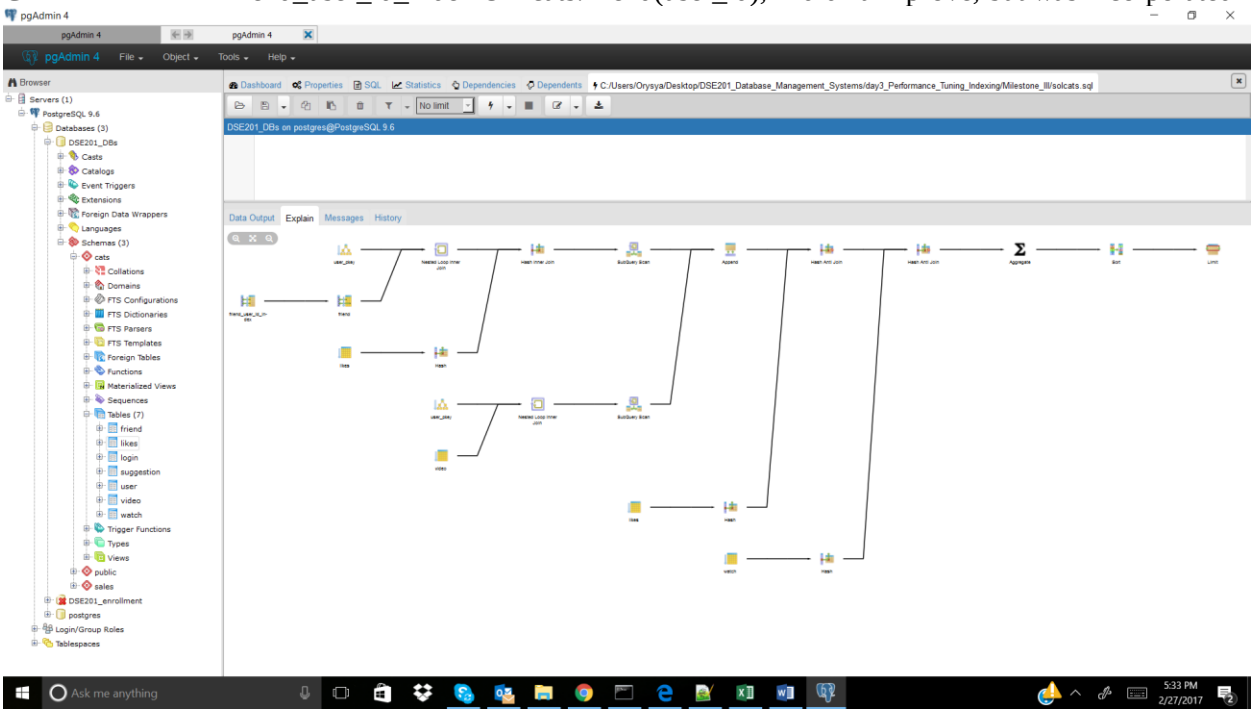
Cats_2
No Index



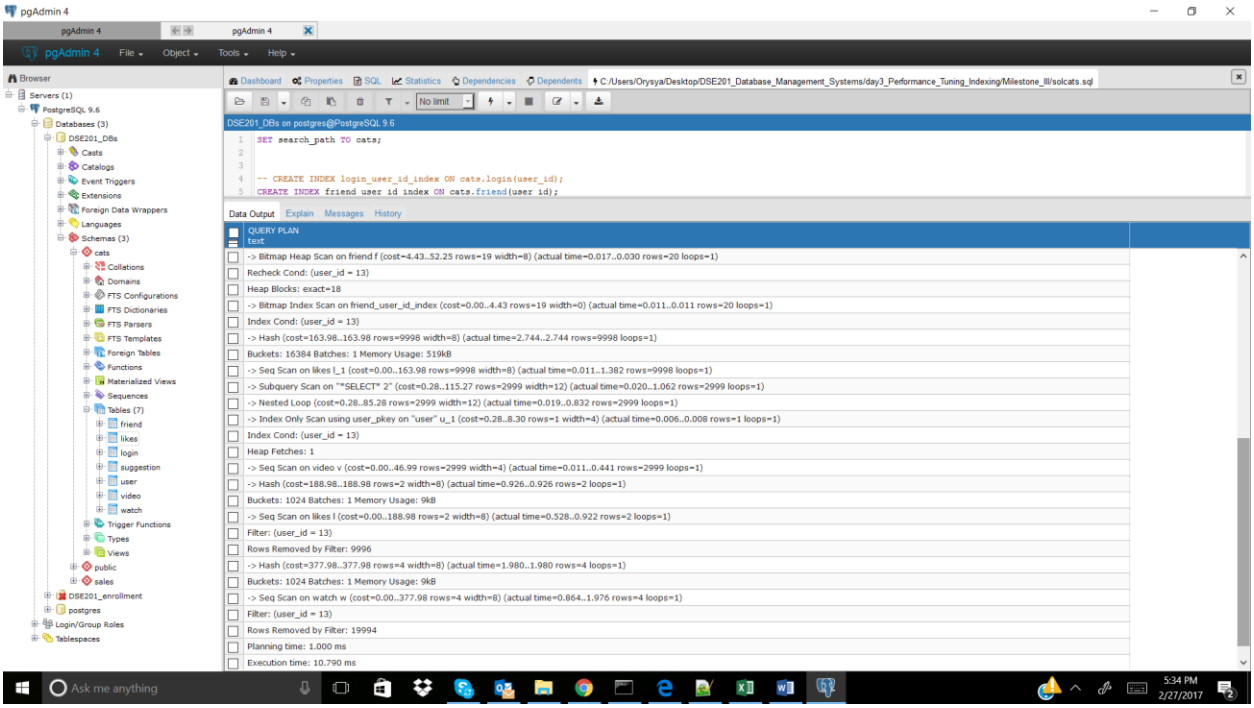
Orysya Stus
2.27.2017



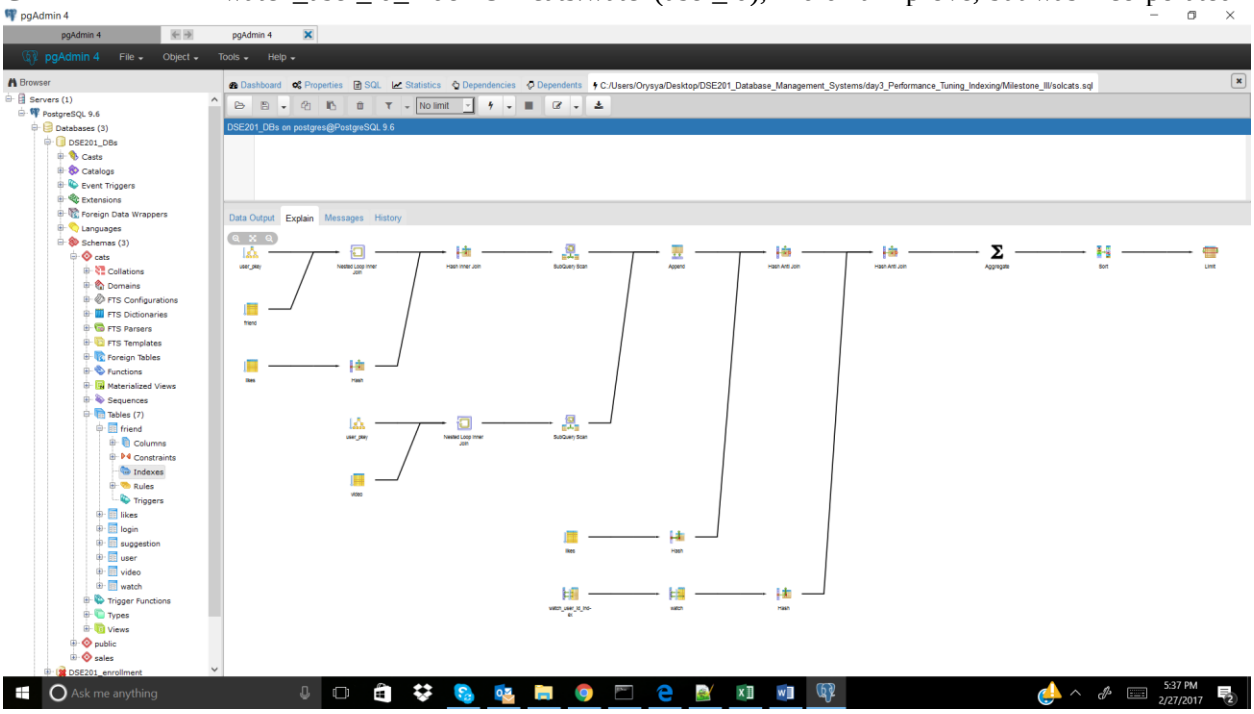
CREATE INDEX friend_user_id_index ON cats.friend(user_id); -- didn't improve, but was incorporated



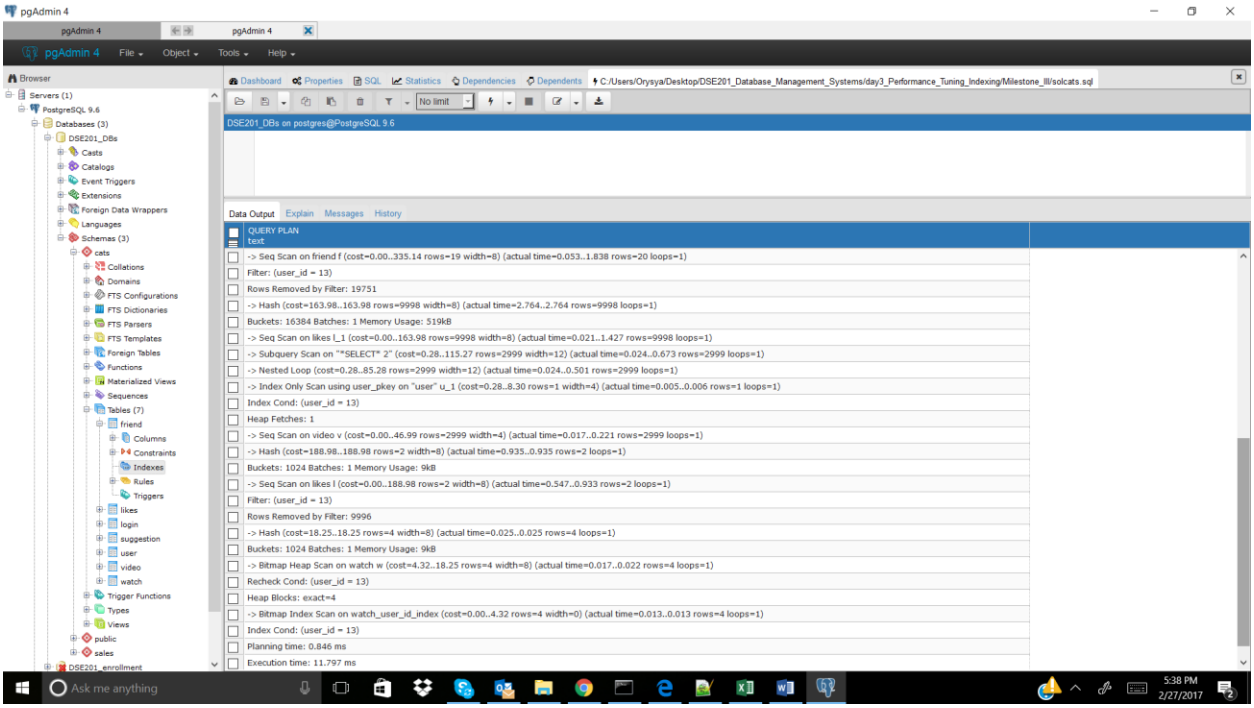
Orysya Stus
2.27.2017



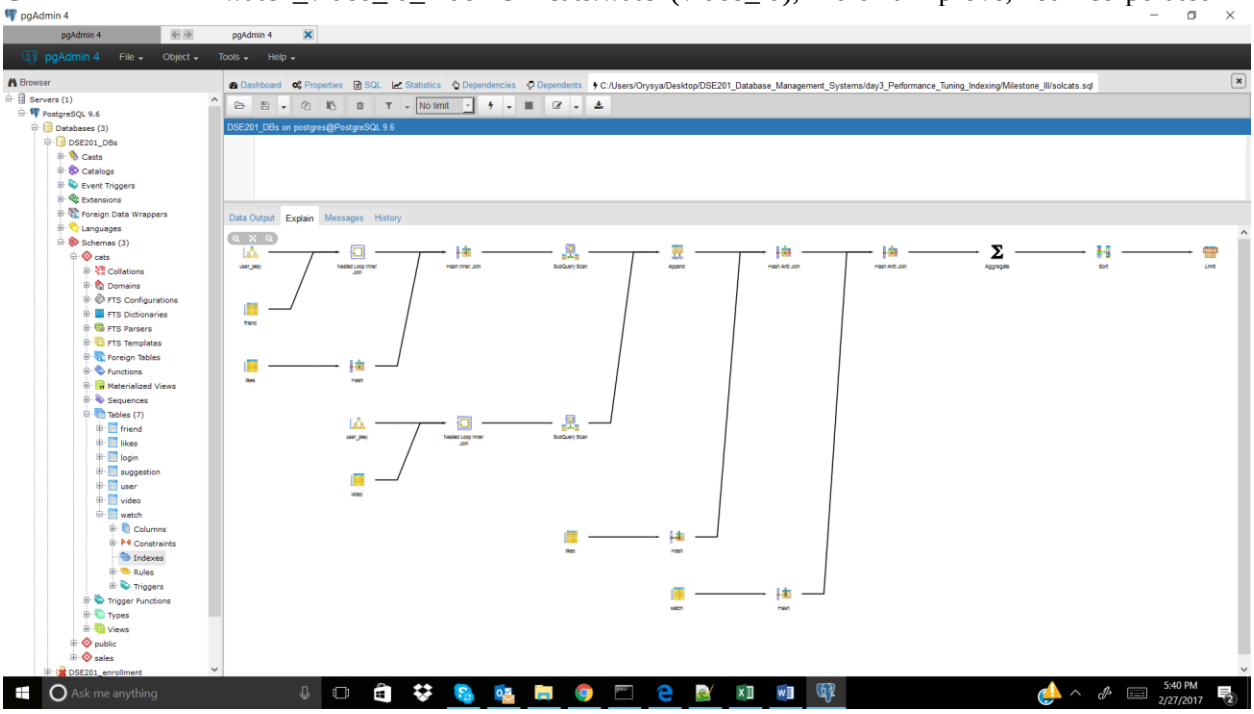
CREATE INDEX watch_user_id_index ON cats.watch(user_id); -- didn't improve, but was incorporated



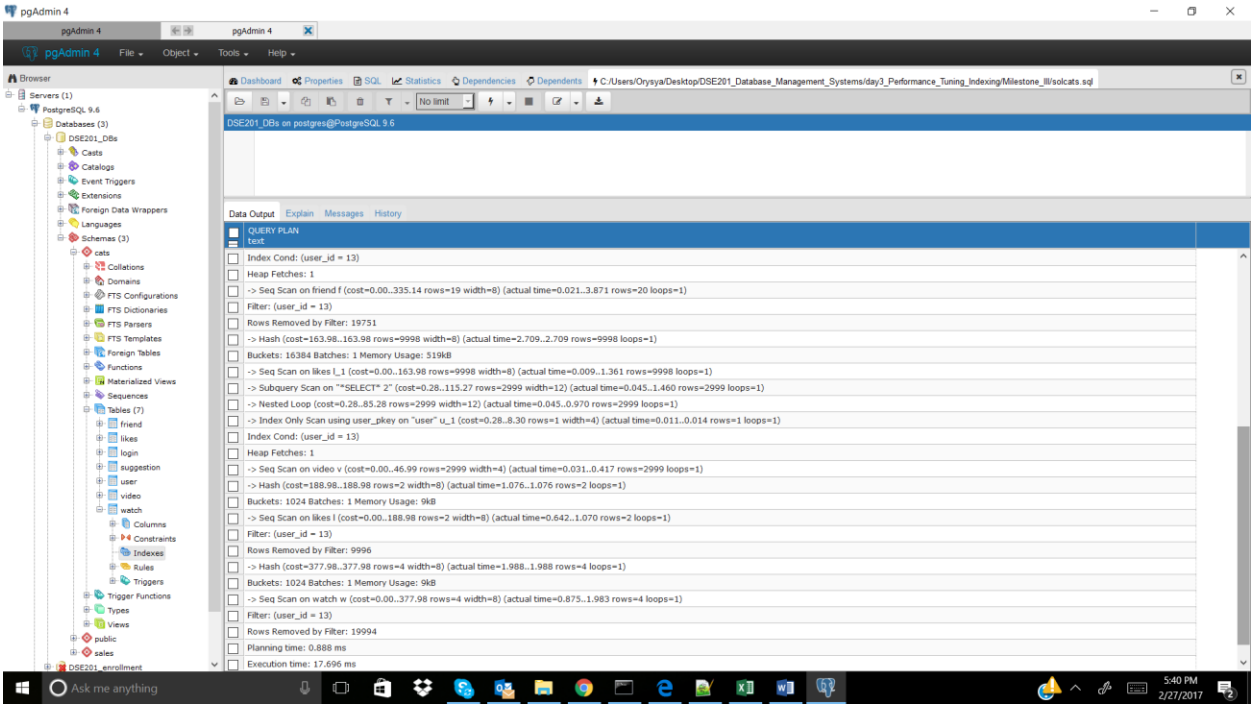
Orysya Stus
2.27.2017



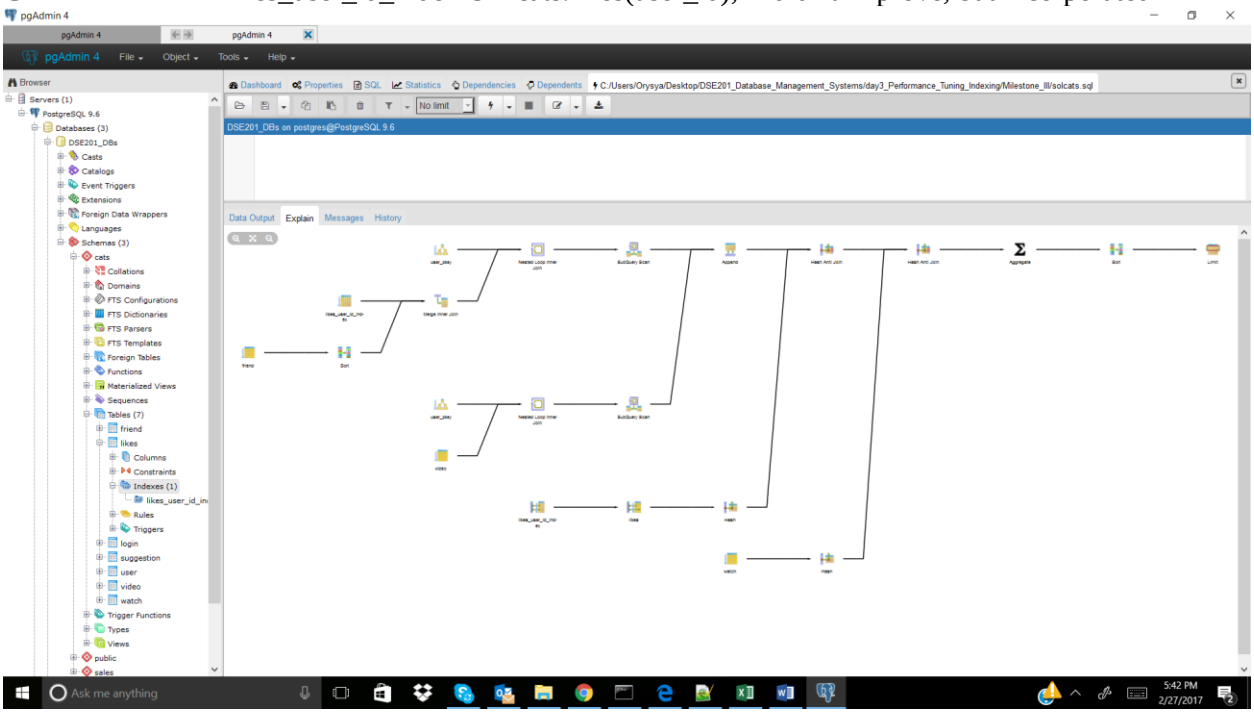
CREATE INDEX watch_video_id_index ON cats.watch(video_id); -- didn't improve, not incorporated



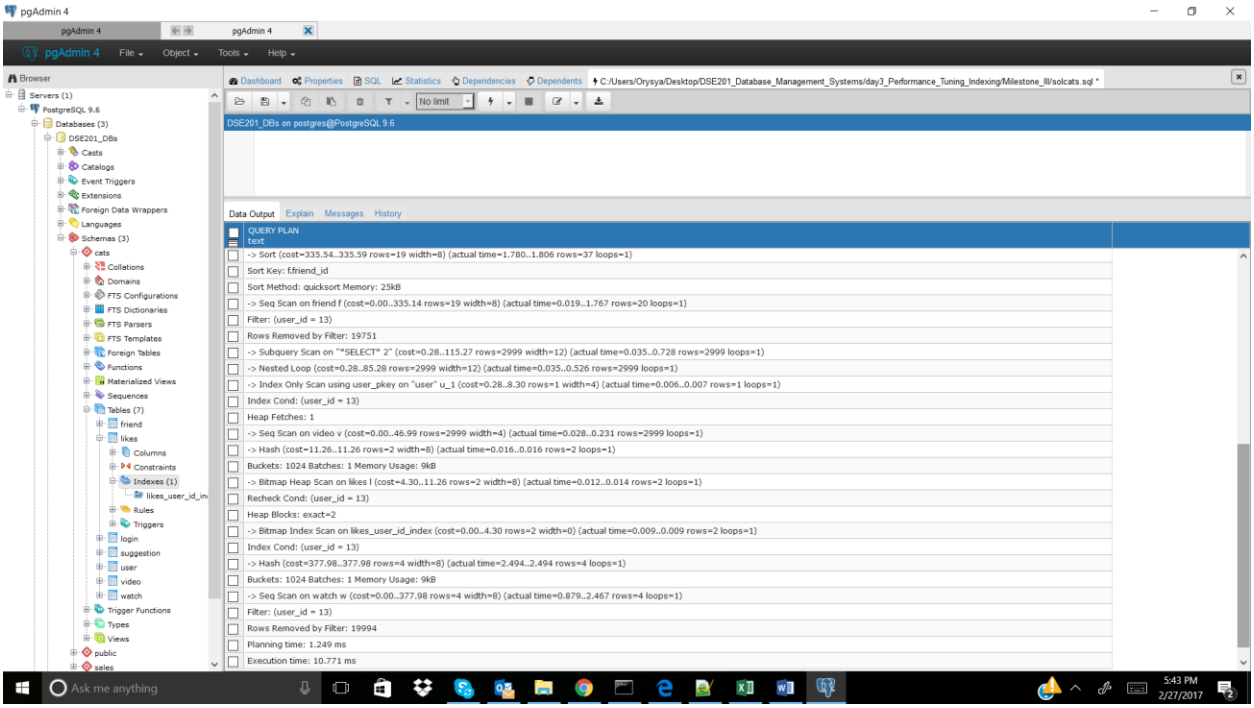
Orysya Stus
2.27.2017



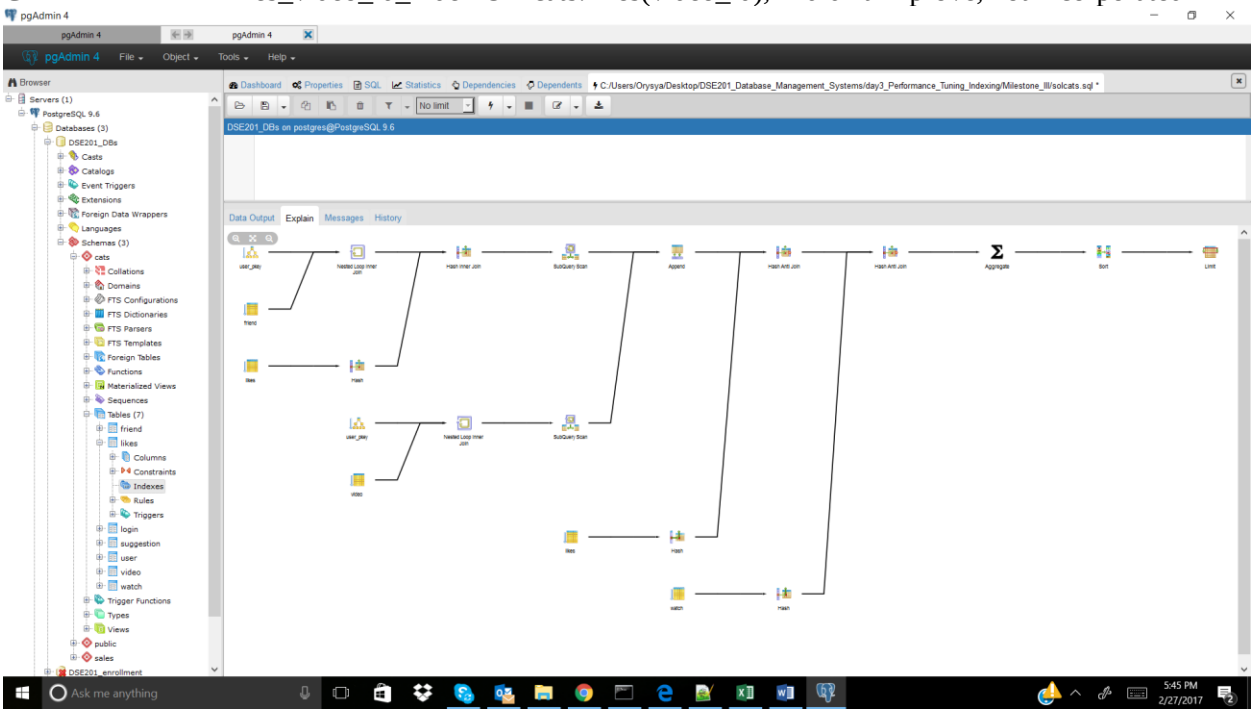
CREATE INDEX likes_user_id_index ON cats.likes(user_id); -- didn't improve, but incorporated



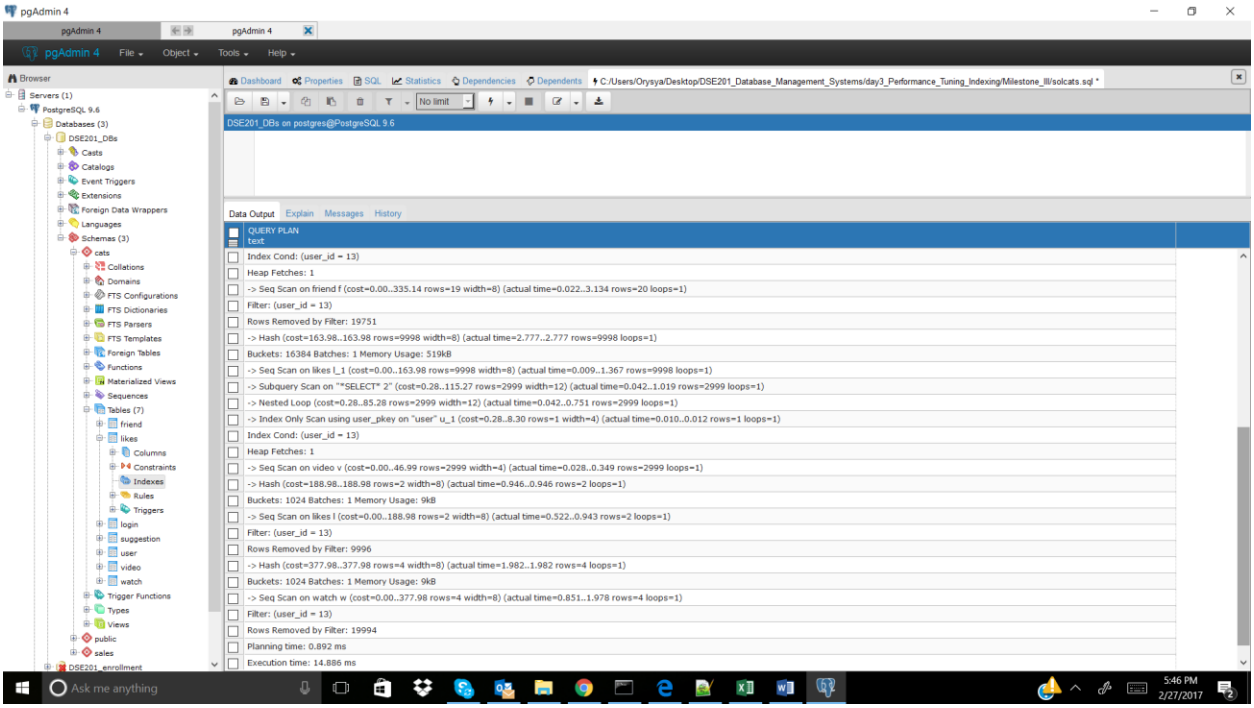
Orysya Stus
2.27.2017



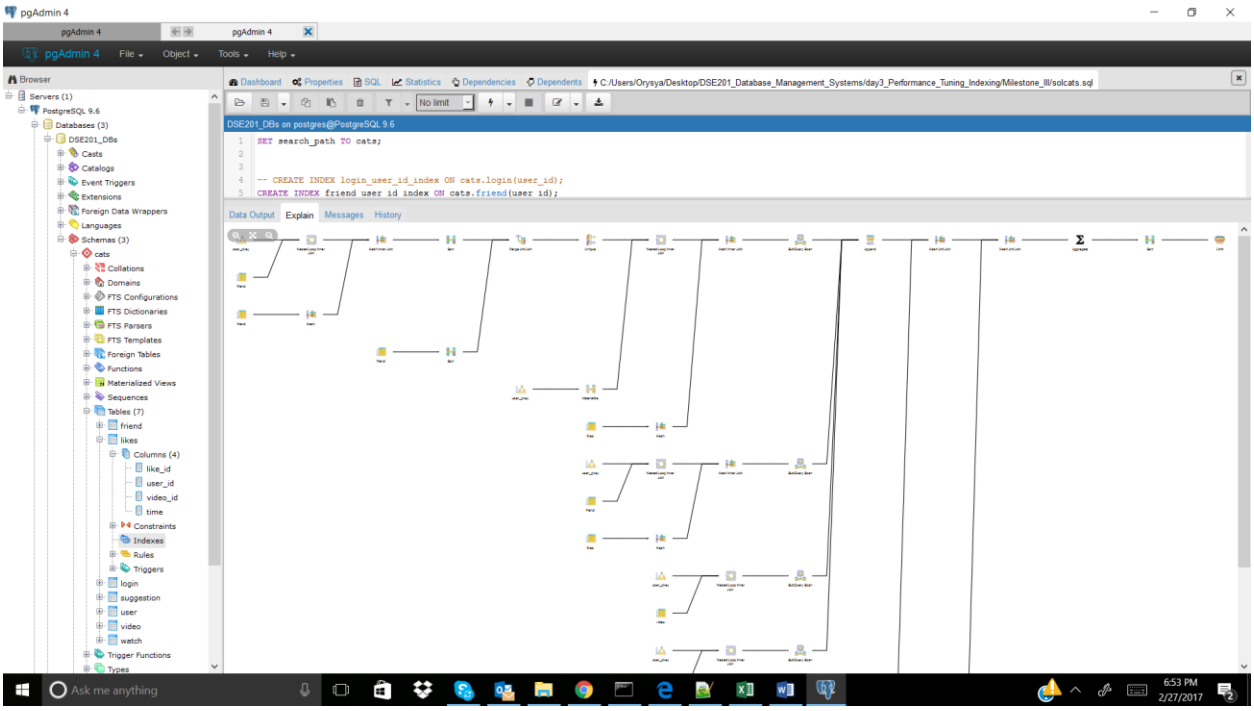
CREATE INDEX likes_video_id_index ON cats.likes(video_id); -- didn't improve, not incorporated



Orysya Stus
2.27.2017

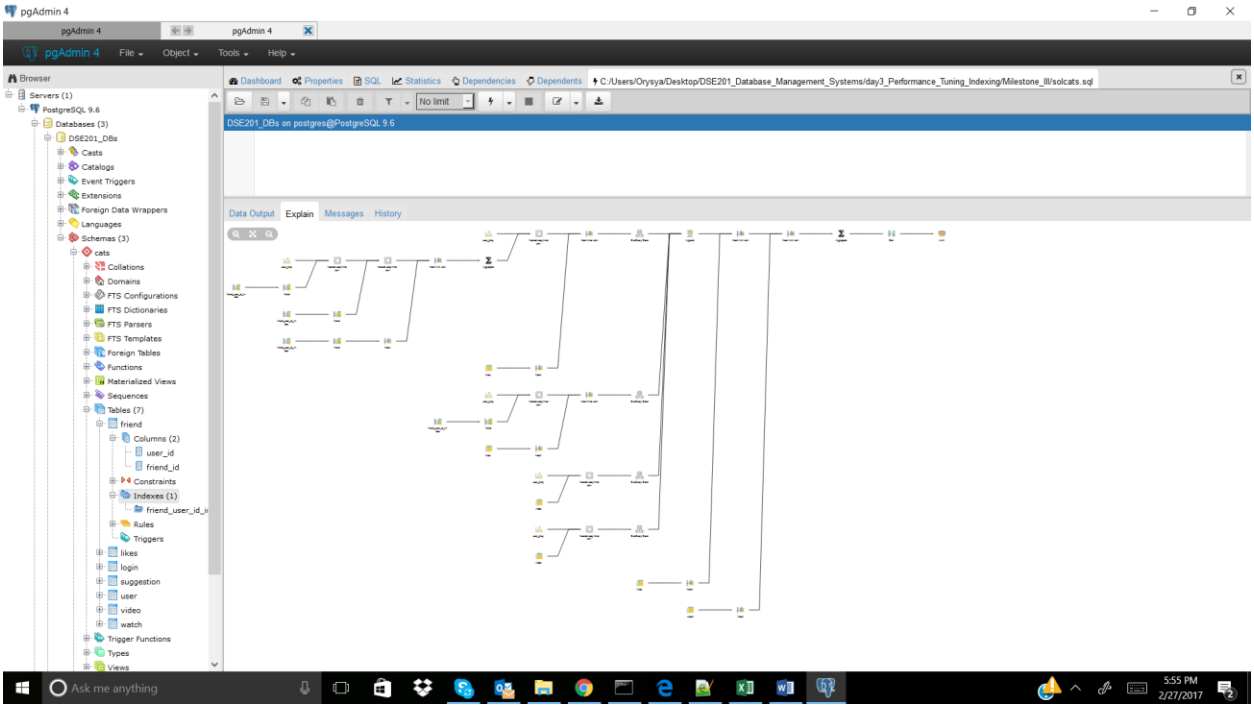


Cats_3
No index

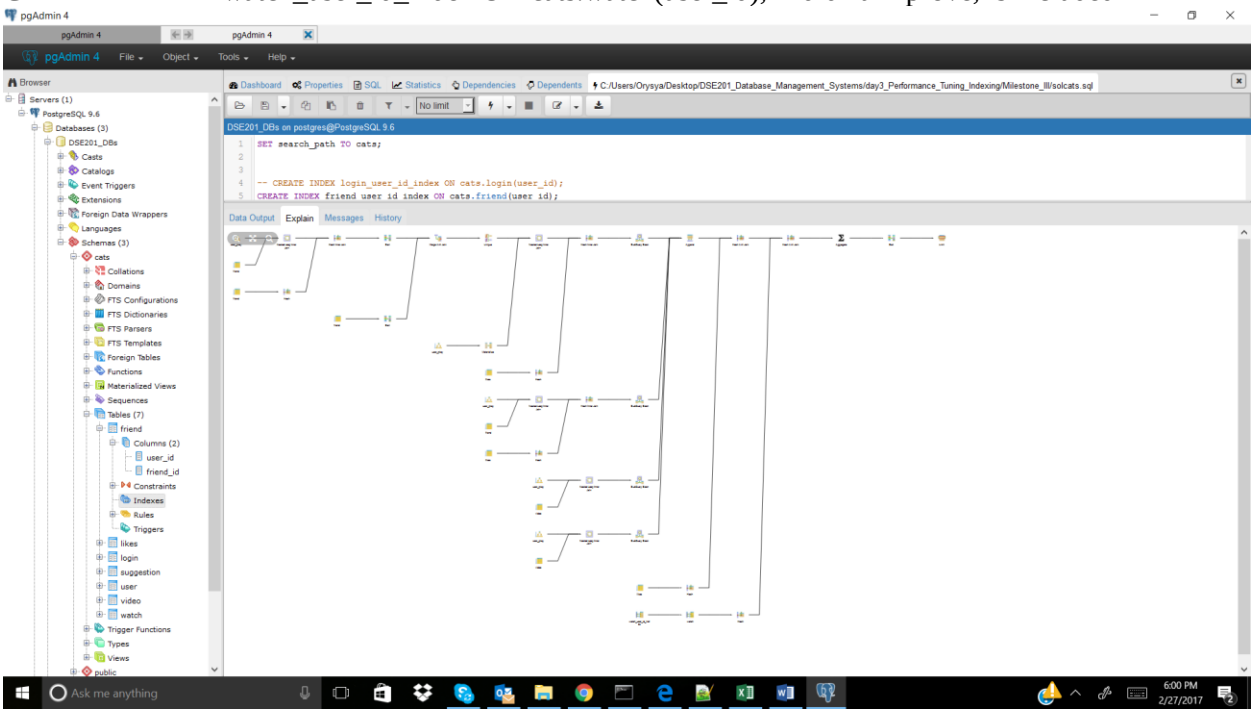


CREATE INDEX friend_user_id_index ON cats.friend(user_id); -- didn't improve, is included

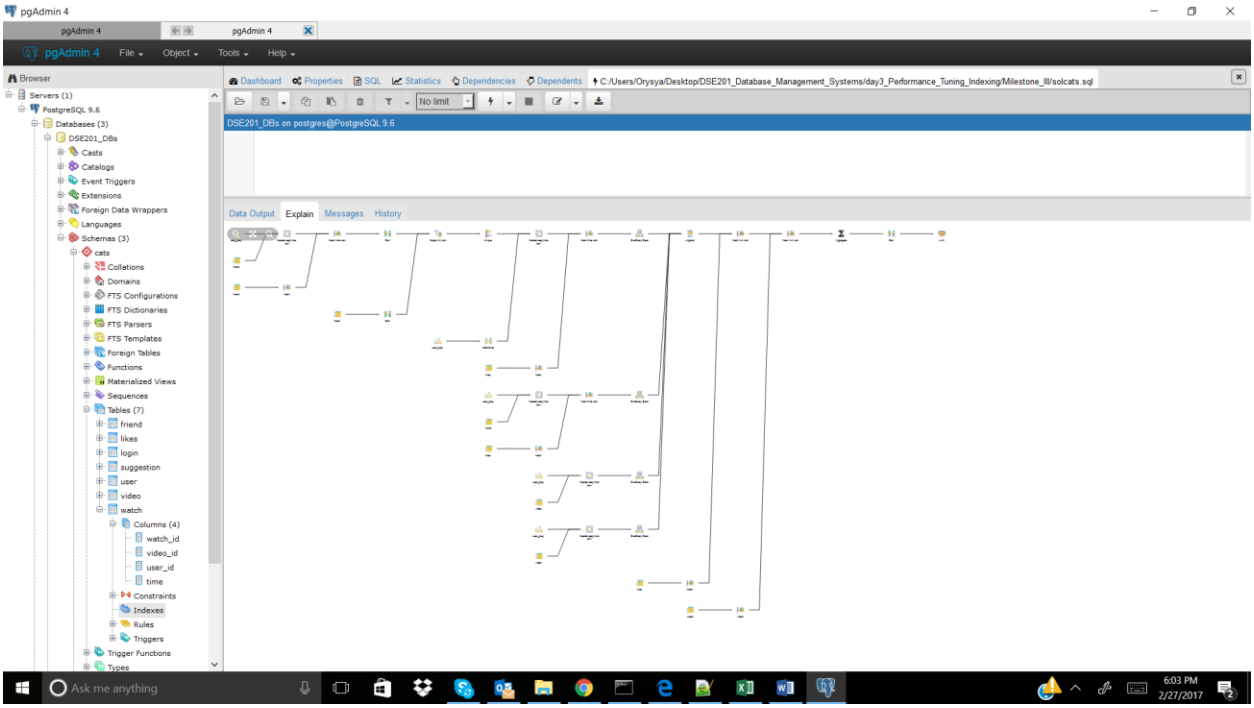
Orysya Stus
2.27.2017



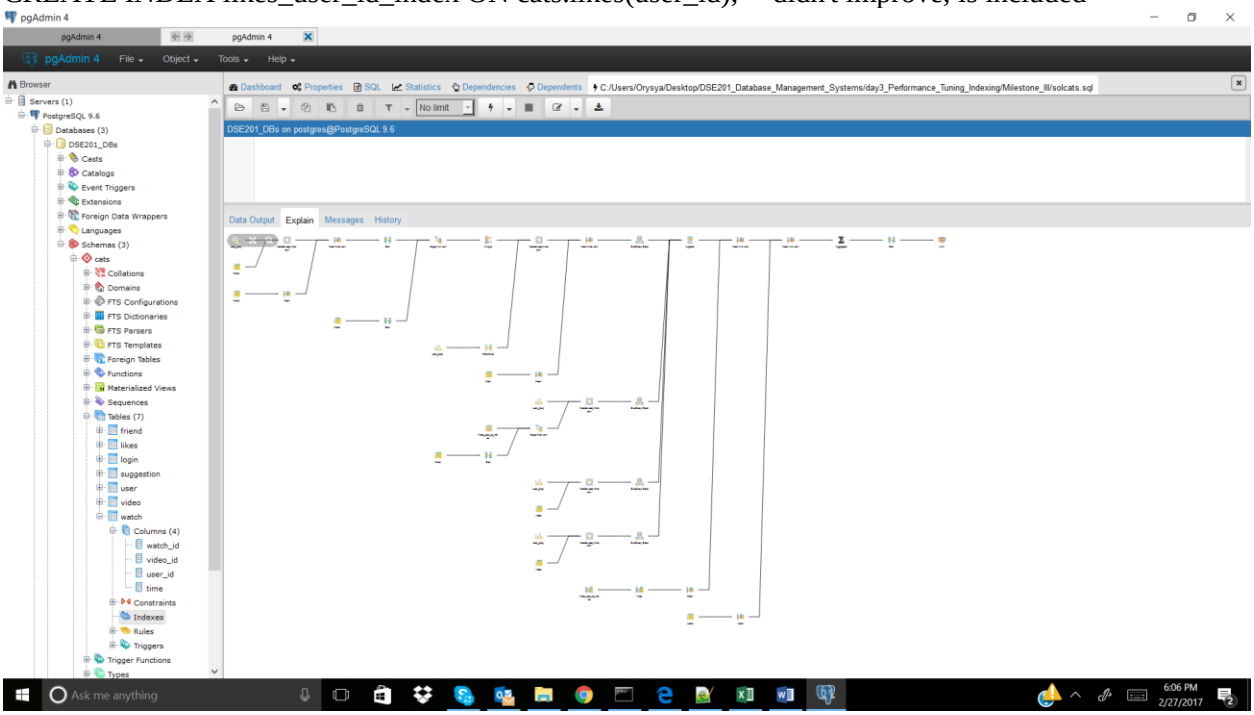
CREATE INDEX watch_user_id_index ON cats.watch(user_id); -- didn't improve, is included



CREATE INDEX watch_video_id_index ON cats.watch(video_id); -- didn't improve, not included

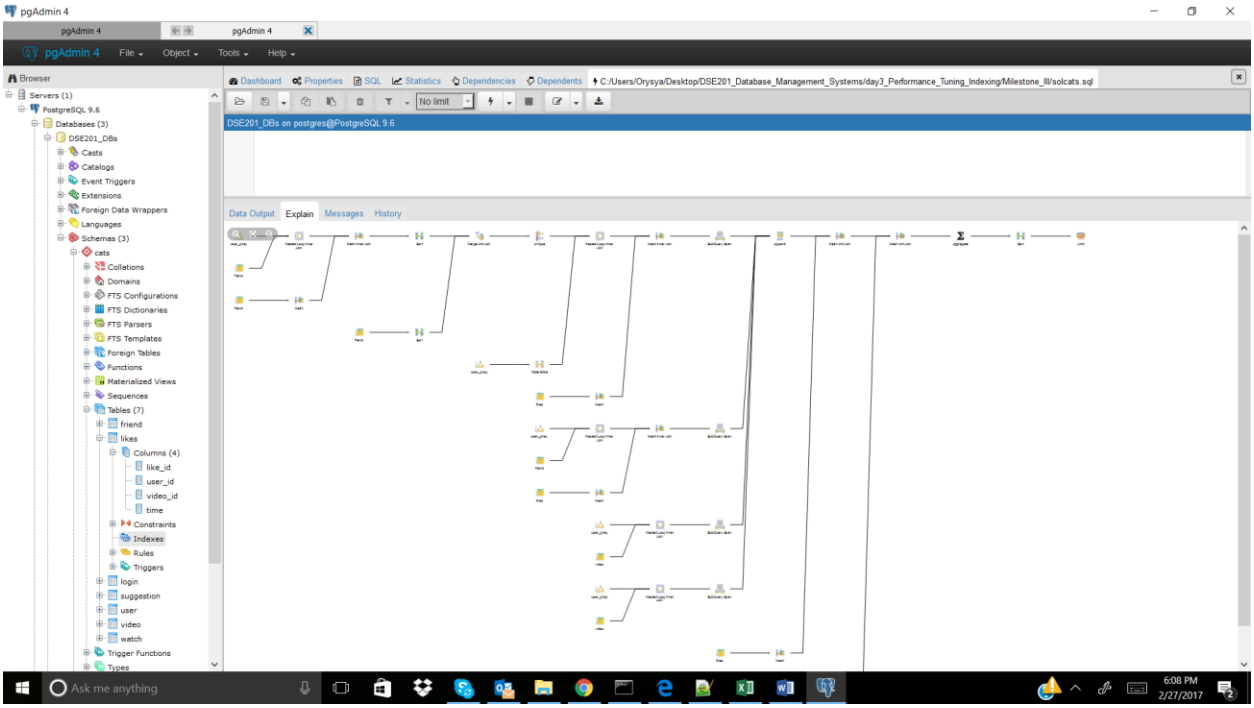


CREATE INDEX likes_user_id_index ON cats.likes(user_id); -- didn't improve, is included

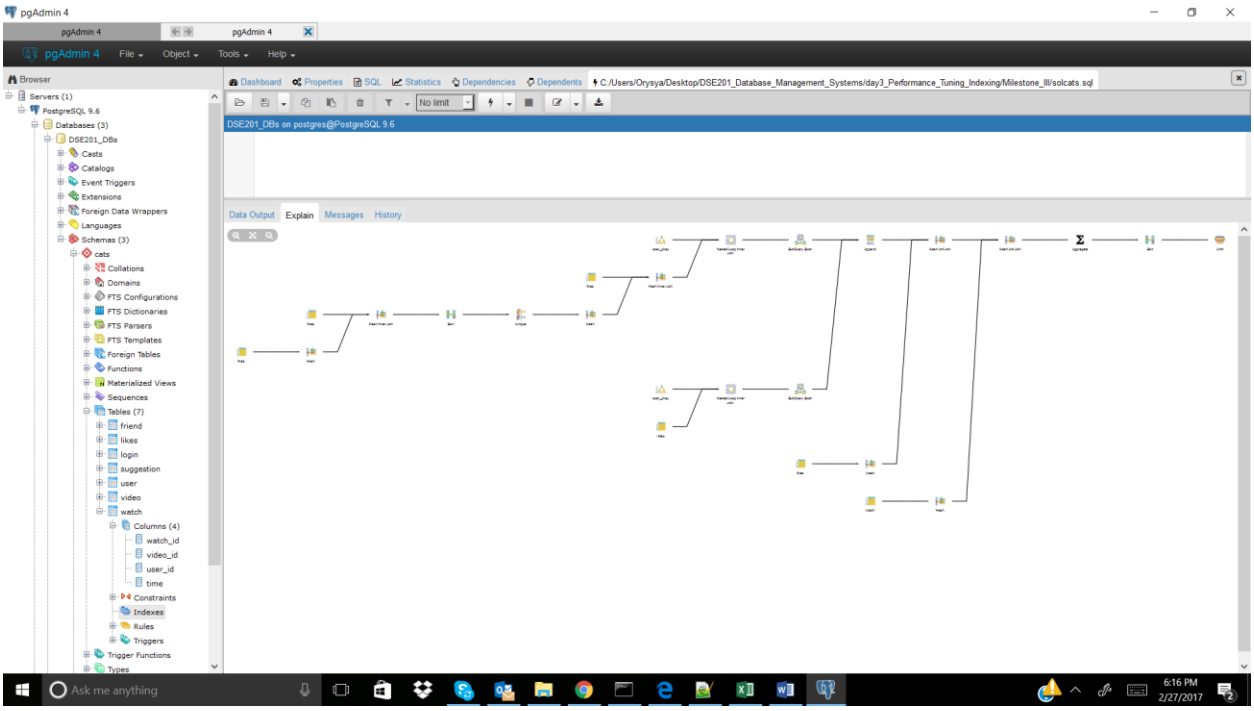


CREATE INDEX likes_video_id_index ON cats.likes(video_id); -- didn't improve, not included

Orysya Stus
2.27.2017

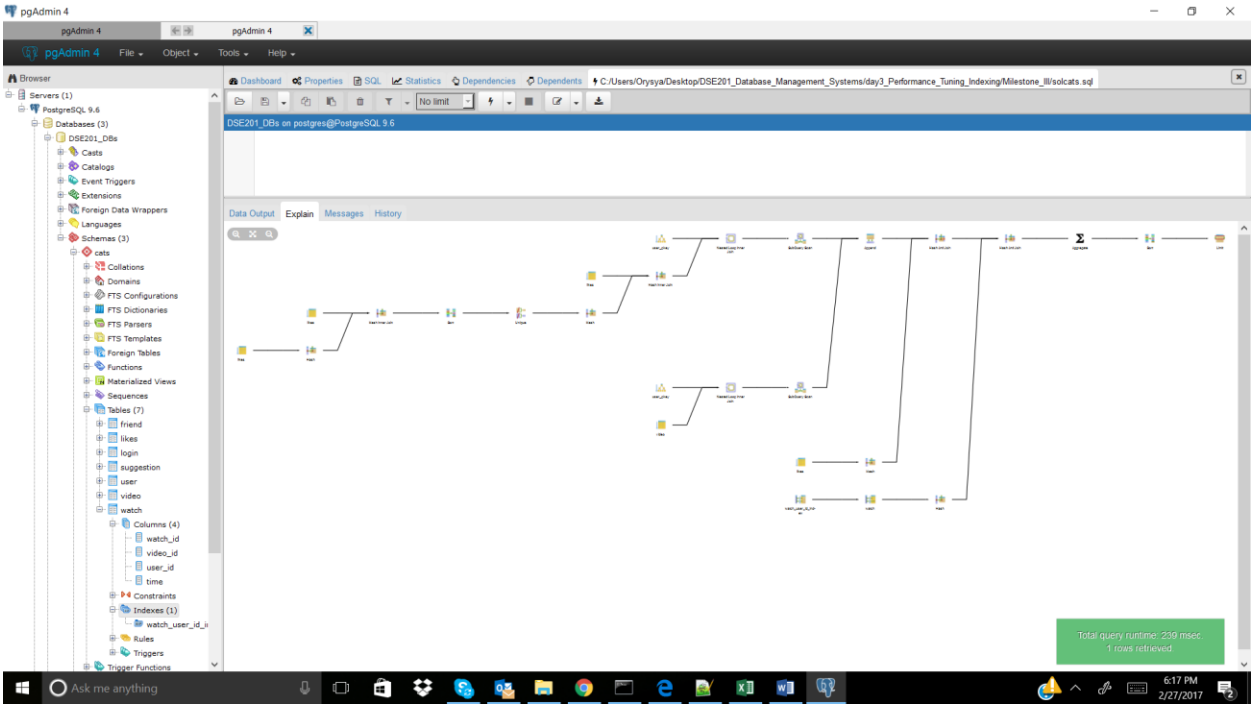


Cats_4
No Index

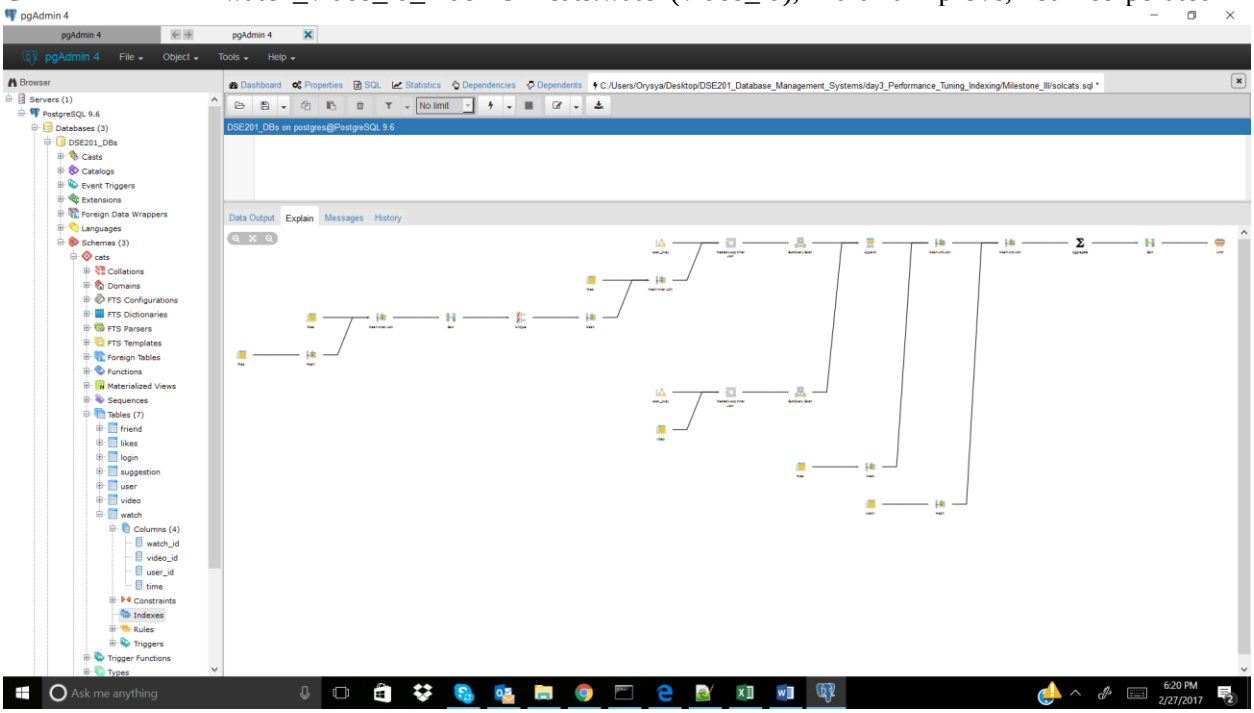


CREATE INDEX watch_user_id_index ON cats.watch(user_id); -- didn't improve, is incorporated

Orysya Stus
2.27.2017

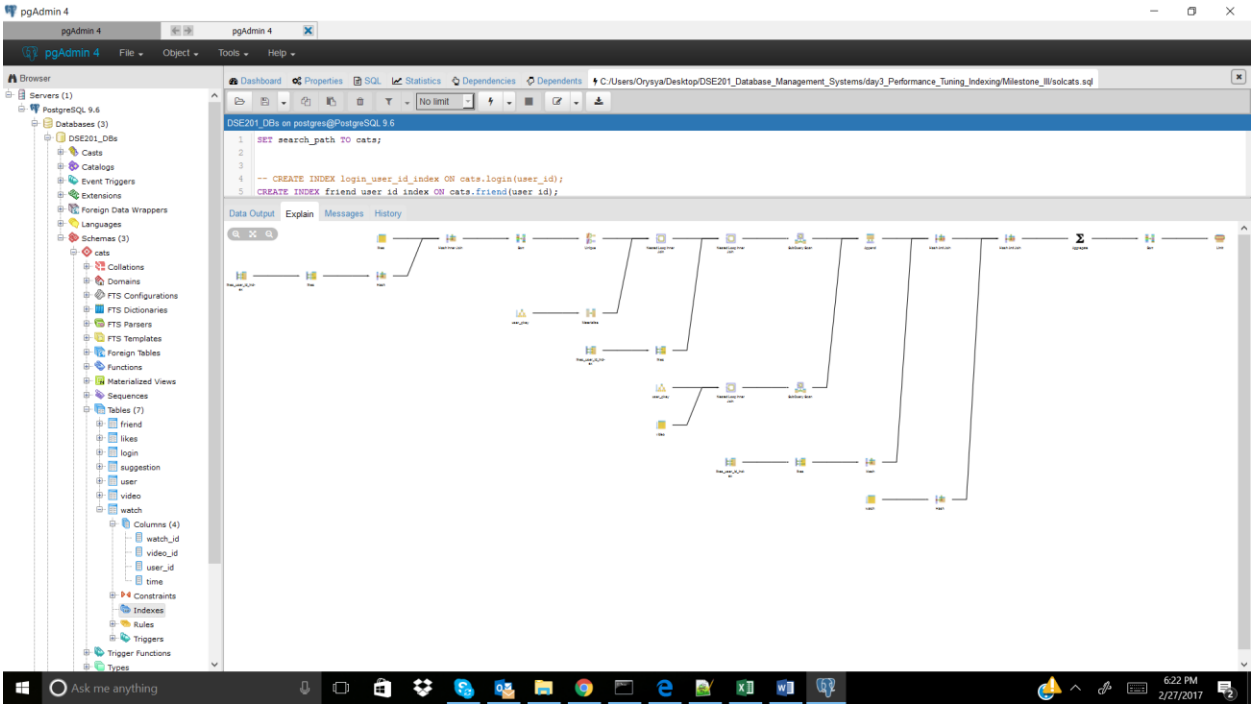


CREATE INDEX watch_video_id_index ON cats.watch(video_id); -- didn't improve, not incorporated

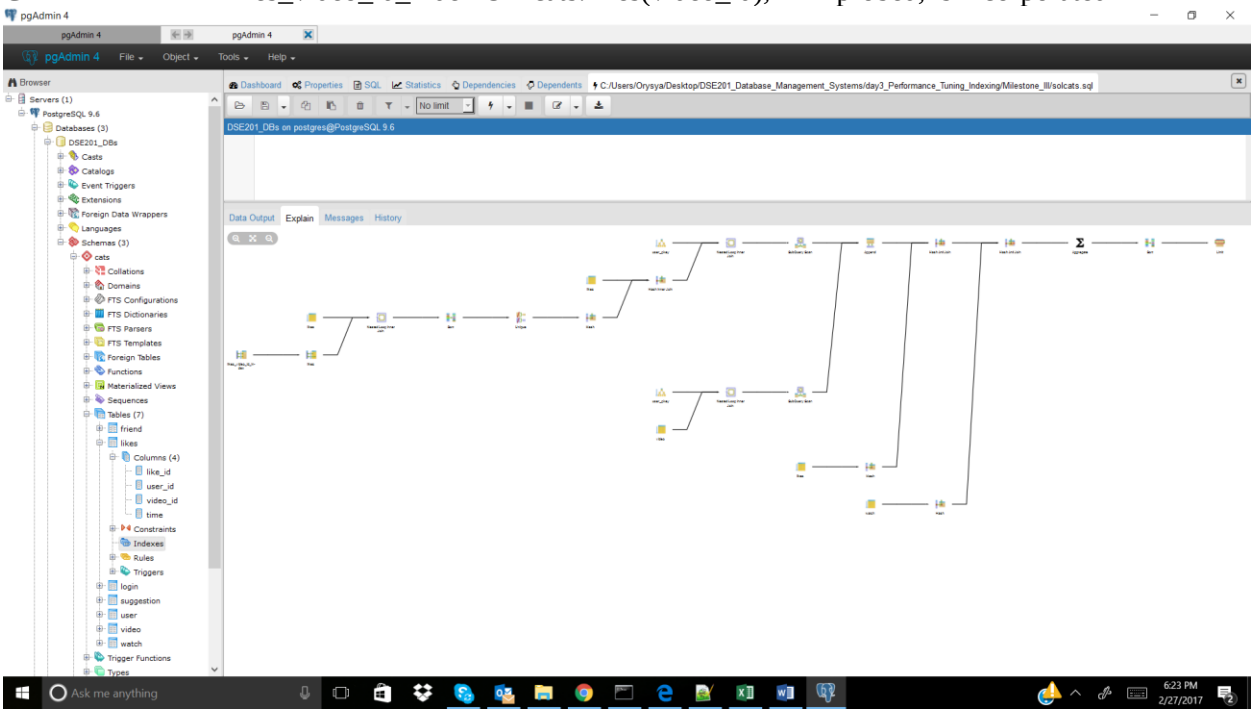


CREATE INDEX likes_user_id_index ON cats.likes(user_id); -- improved, is incorporated

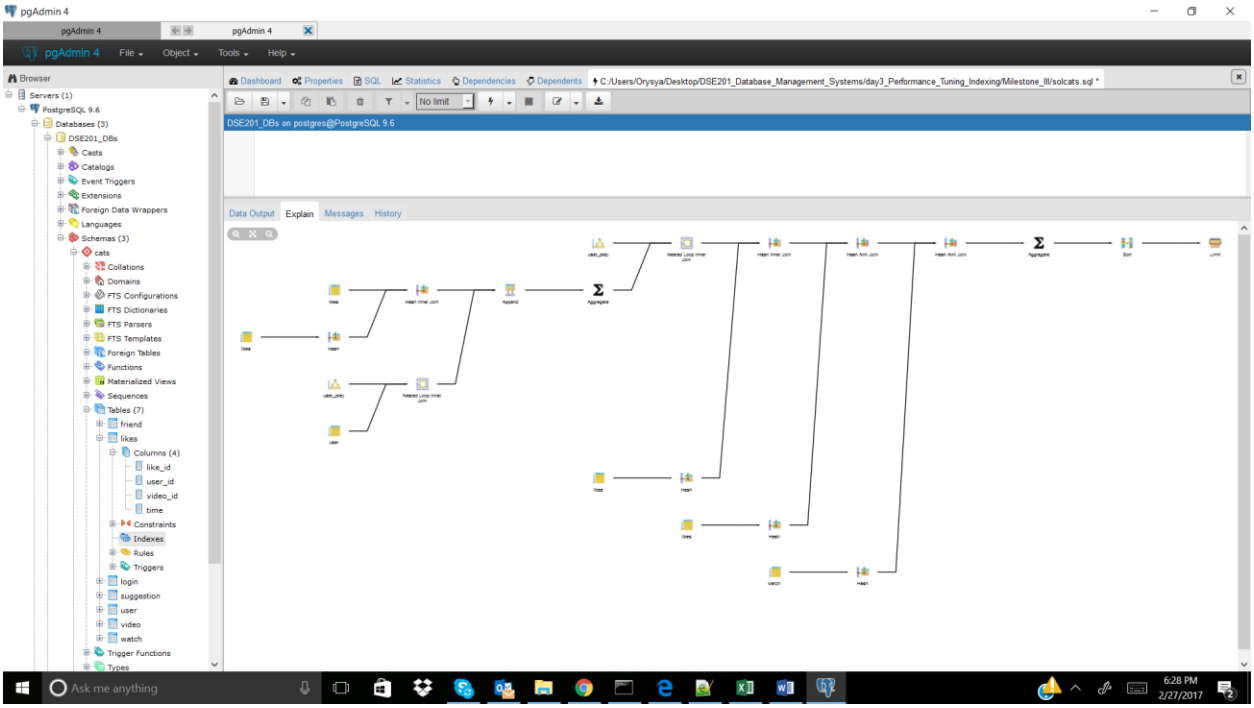
Orysya Stus
2.27.2017



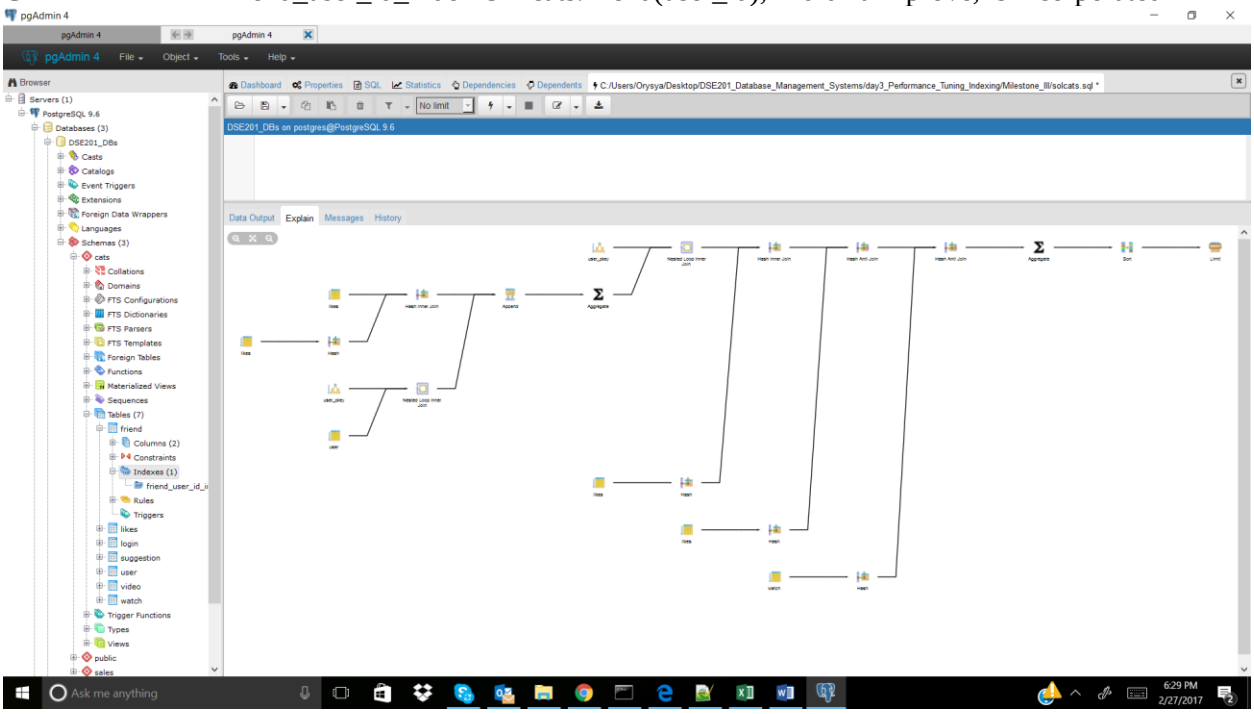
CREATE INDEX likes_video_id_index ON cats.likes(video_id); -- improbed, is incorporated



Cats_5
No index

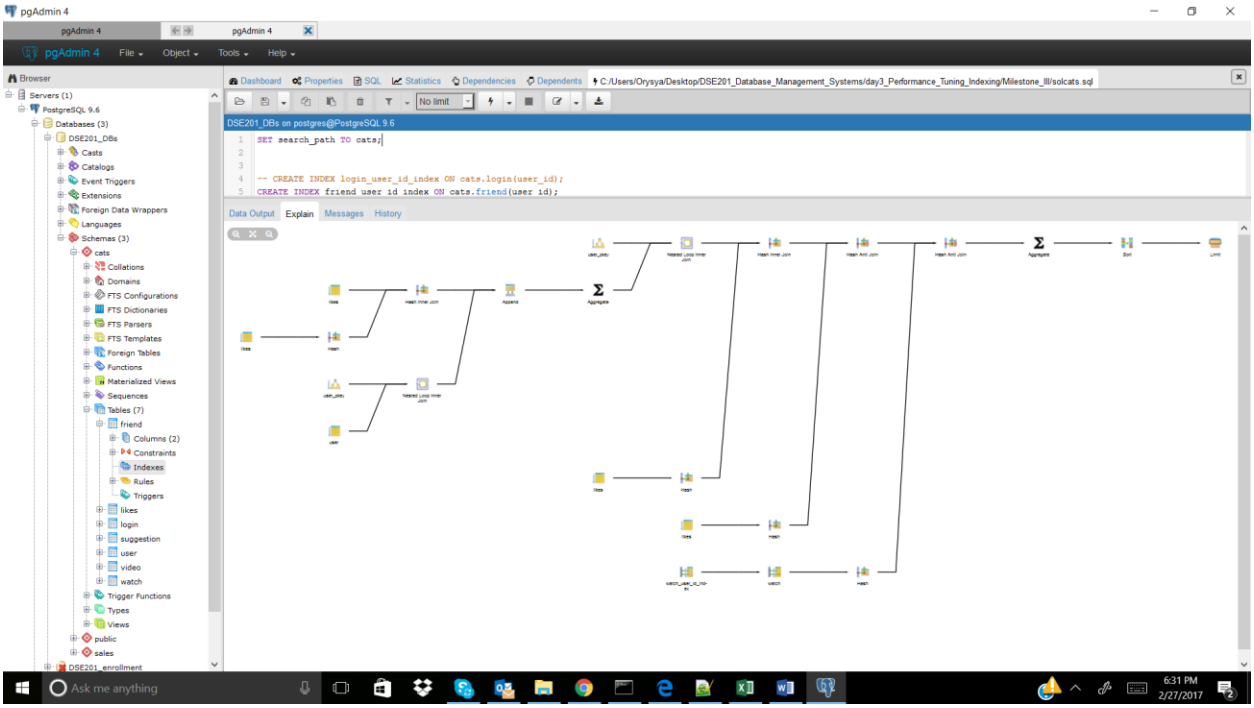


CREATE INDEX friend_user_id_index ON cats.friend(user_id); -- didn't improve, is incorporated

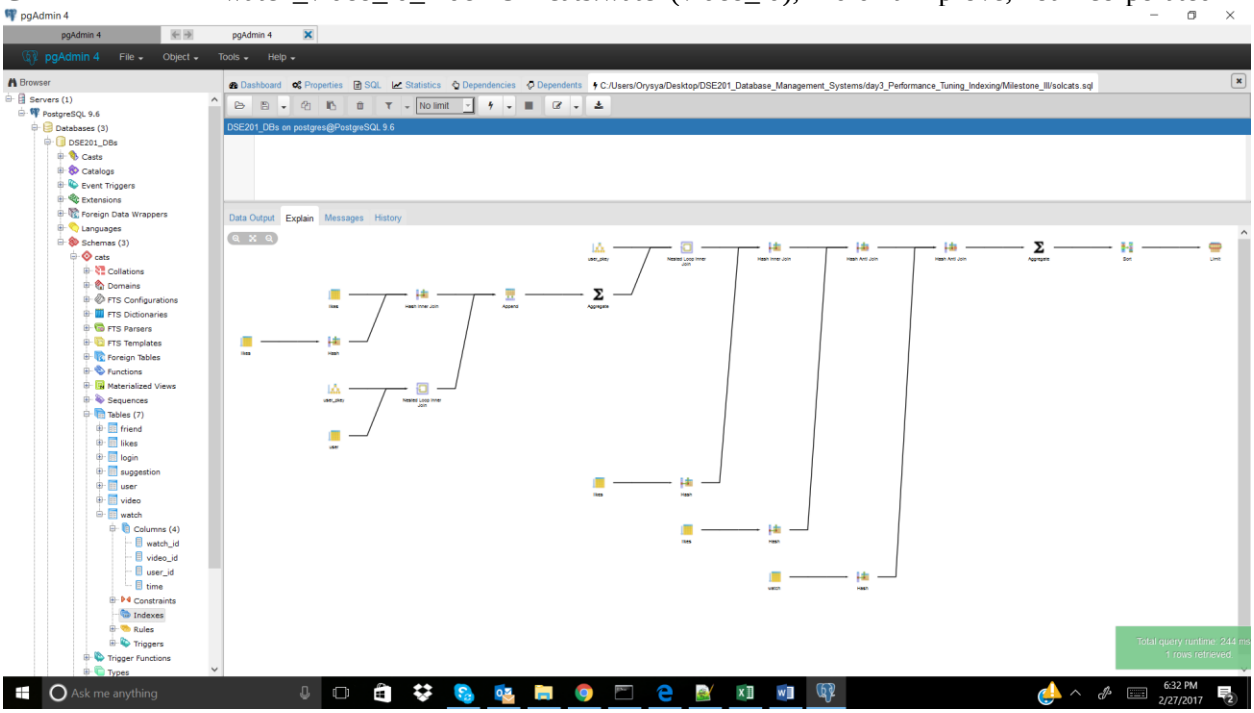


CREATE INDEX watch_user_id_index ON cats.watch(user_id); -- didn't improve, is incorporated

Orysya Stus
2.27.2017

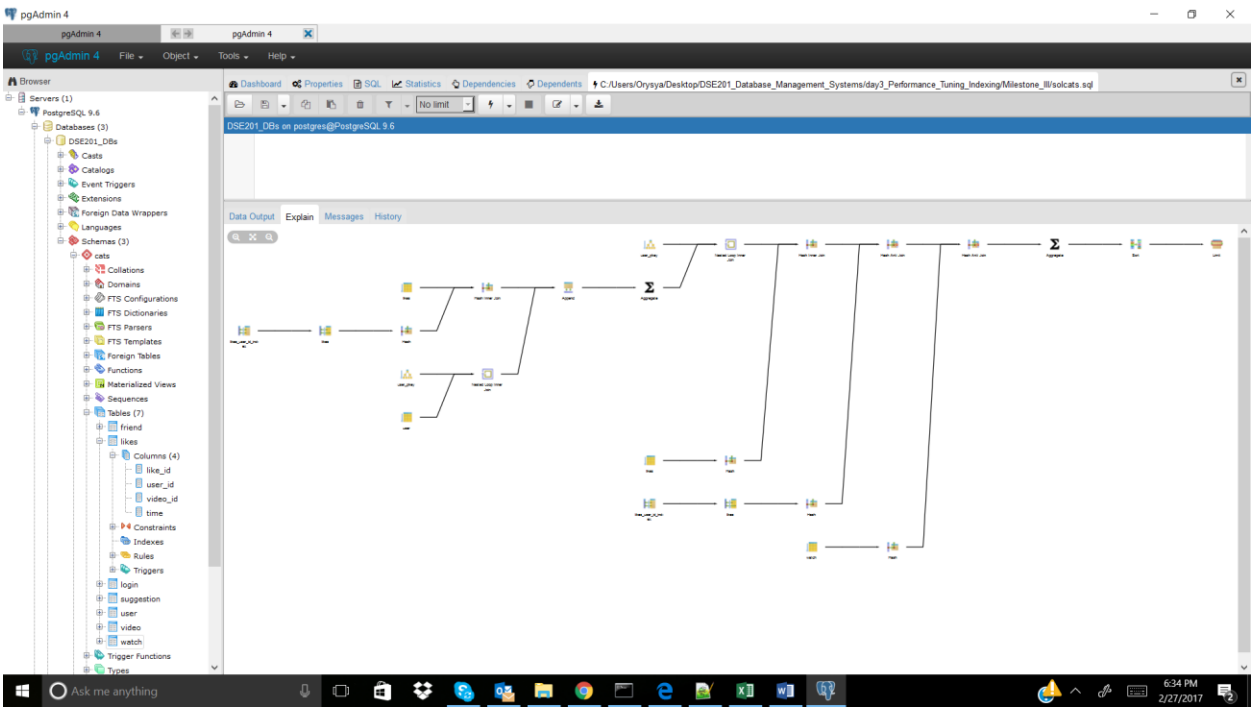


CREATE INDEX watch_video_id_index ON cats.watch(video_id); -- didn't improve, not incorporated



CREATE INDEX likes_user_id_index ON cats.likes(user_id); -- didn't improve, is incorporated

Orysya Stus
2.27.2017



CREATE INDEX likes_video_id_index ON cats.likes(video_id); -- didn't improve, is incorporated