

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

$$[1 - (\text{with index time}/\text{without index time})] * 100\%$$

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 clause.

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

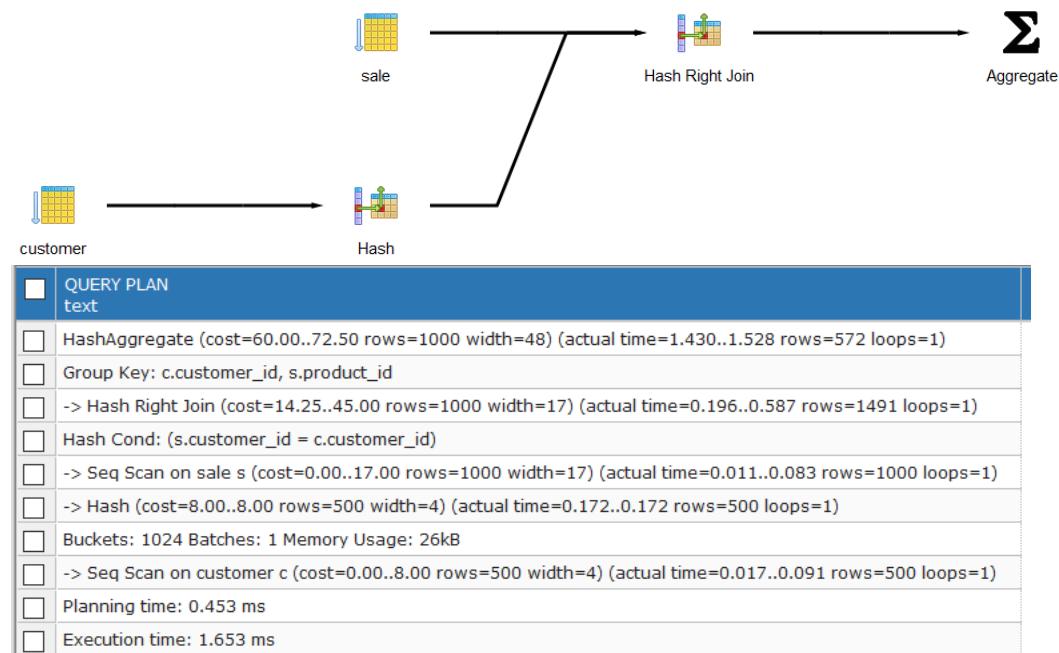
Appendix

Sales

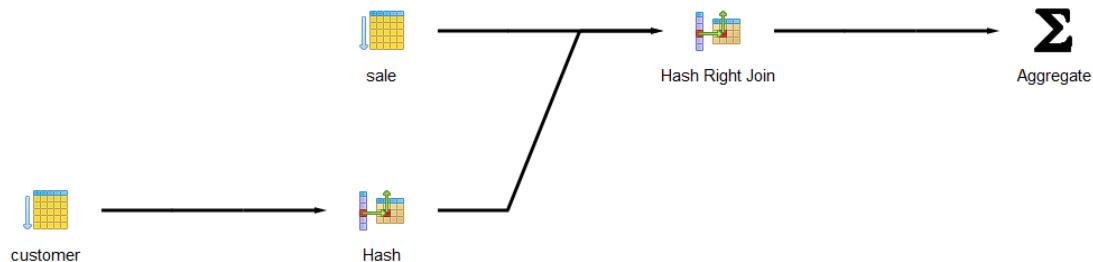
Query 1:

Sales_1

No Index



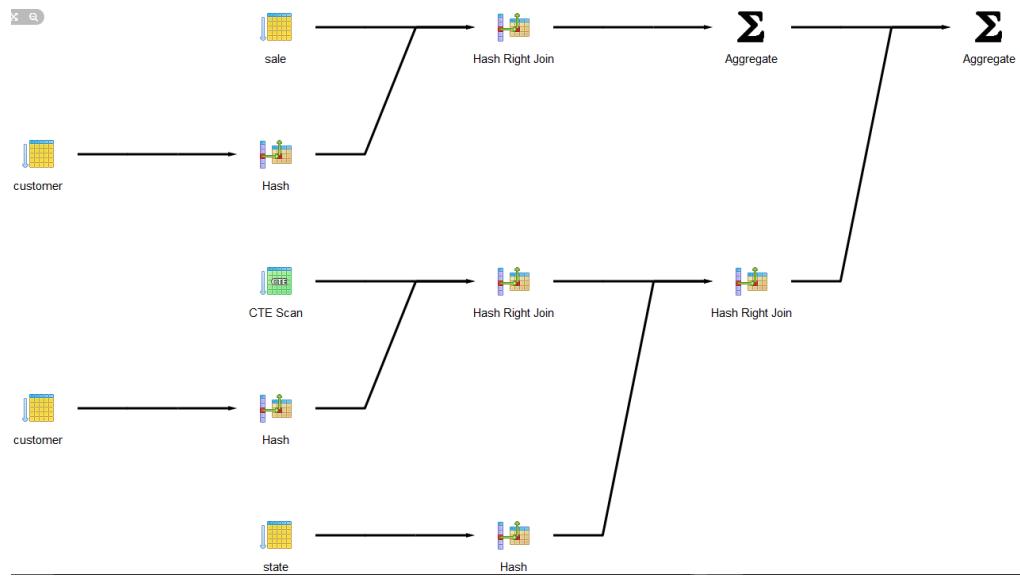
CREATE INDEX sales_cust_id ON sales.sale(customer_id); -- No index needed



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

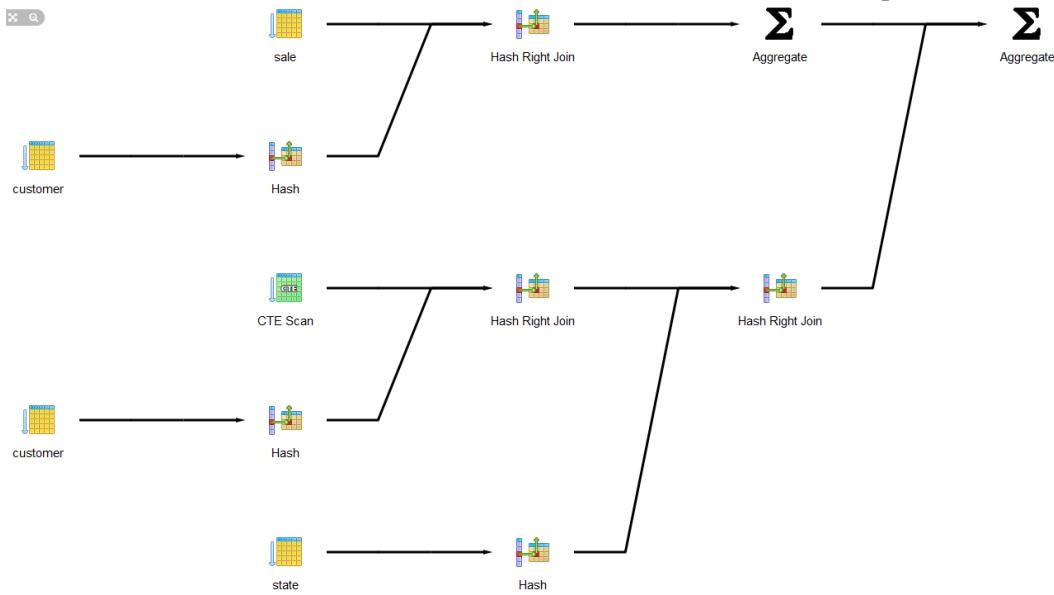


```

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

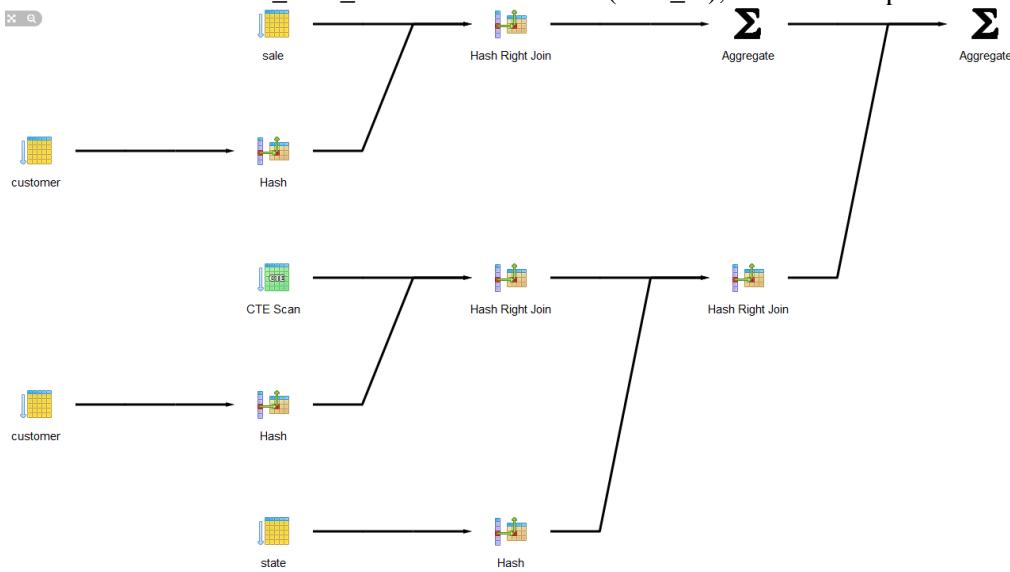
```

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



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

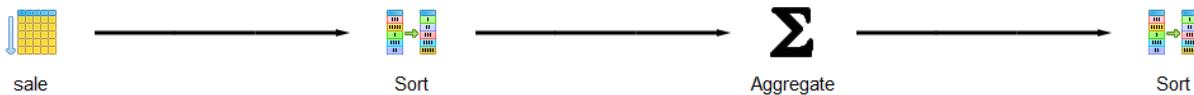
CREATE INDEX cust_state_id ON sales.customer(state_id); -- doesn't help



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

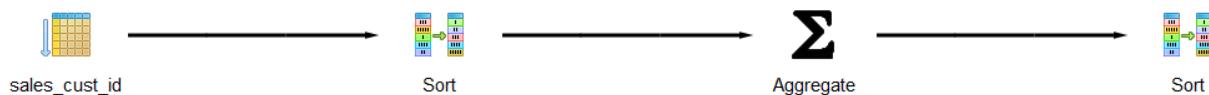
Sales_3

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

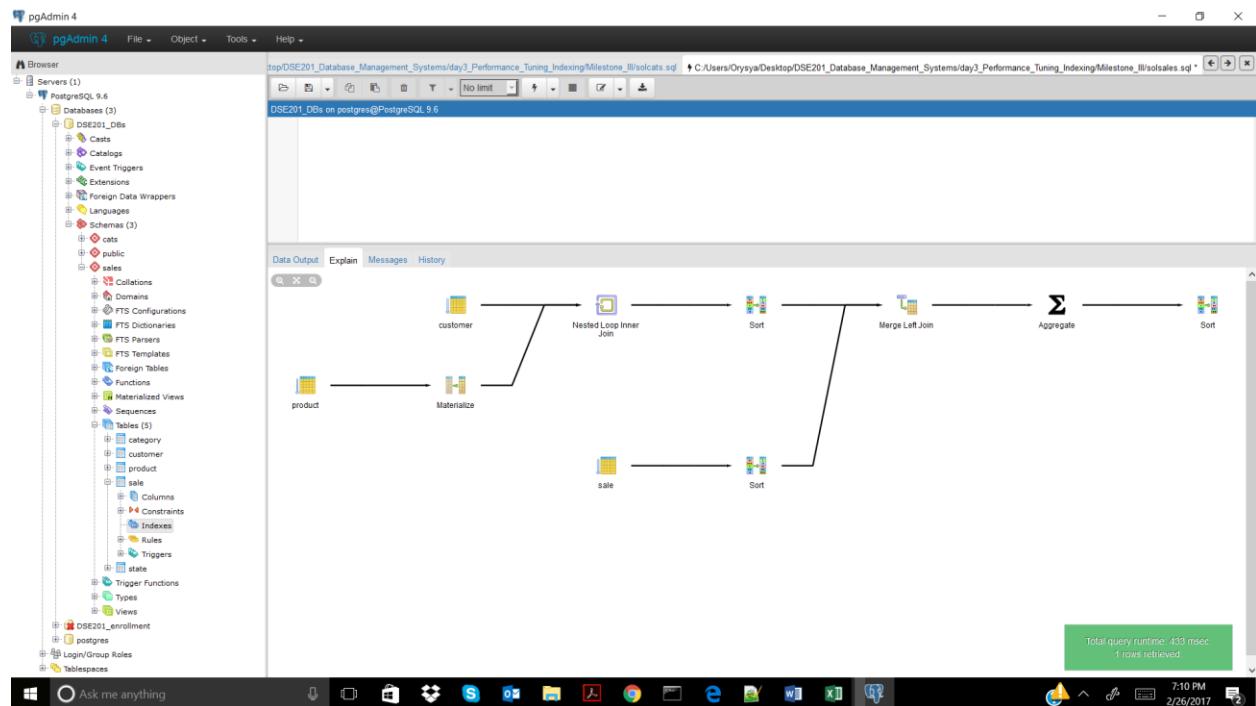
CREATE INDEX sales_cust_id ON sales.sale(customer_id); -- this helped



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



pgAdmin 4

DSE201_DBs on postgres@PostgreSQL 9.6

```

SELECT sum(s.quantity) * p.price
FROM sales s
JOIN customer c ON s.customer_id = c.id
JOIN product p ON s.product_id = p.id
WHERE c.state = 'CA'
GROUP BY p.product_id
ORDER BY p.product_id;

```

Execution Plan

```

QUERY PLAN
TEXT
Sort (cost=21257.40..12282.40 rows=50000 width=60) (actual time=233.865..243.534 rows=50000 loops=1)
  Sort Key: c.customer_id, COALESCE(sum((s.quantity)::numeric * s.price)), 0.0) DESC
  Sort Method: external sort Disk: 224kB
  -> GroupAggregate (cost=4604.49..6371.99 rows=50000 width=60) (actual time=77.663..145.843 rows=50000 loops=1)
    Group Key: c.customer_id, p.product_id
    -> Merge Left Join (cost=4604.49..4996.99 rows=50000 width=29) (actual time=77.642..99.571 rows=50919 loops=1)
      Merge Cond: ((c.customer_id = s.customer_id) AND (p.product_id = s.product_id))
      -> Sort (cost=4537.66..4662.66 rows=50000 width=20) (actual time=76.478..88.124 rows=50000 loops=1)
        Sort Key: c.customer_id, p.product_id
        Sort Method: external sort Disk: 1472kB
        -> Nested Loop (cost=0.00..435.25 rows=50000 width=20) (actual time=0.074..9.240 rows=50000 loops=1)
          -> Seq Scan on customer c (cost=0.00..8.00 rows=500 width=12) (actual time=0.040..0.173 rows=500 loops=1)
          -> Materialize (cost=0.00..2.50 rows=100 width=8) (actual time=0.000..0.005 rows=100 loops=500)
            -> Seq Scan on product p (cost=0.00..2.00 rows=100 width=8) (actual time=0.021..0.036 rows=100 loops=1)
            -> Sort (cost=66.83..69.33 rows=1000 width=17) (actual time=1.157..1.363 rows=1000 loops=1)
              Sort Key: s.customer_id, s.product_id
              Sort Method: quicksort Memory: 103kB
            -> Seq Scan on sale s (cost=0.00..17.00 rows=1000 width=17) (actual time=0.021..0.211 rows=1000 loops=1)
      Planning time: 0.393 ms
      Execution time: 258.390 ms

```

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

pgAdmin 4

DSE201_DBs on postgres@PostgreSQL 9.6

```

SELECT sum(s.quantity) * p.price
FROM sales s
JOIN customer c ON s.customer_id = c.id
JOIN product p ON s.product_id = p.id
WHERE c.state = 'CA'
GROUP BY p.product_id
ORDER BY p.product_id;

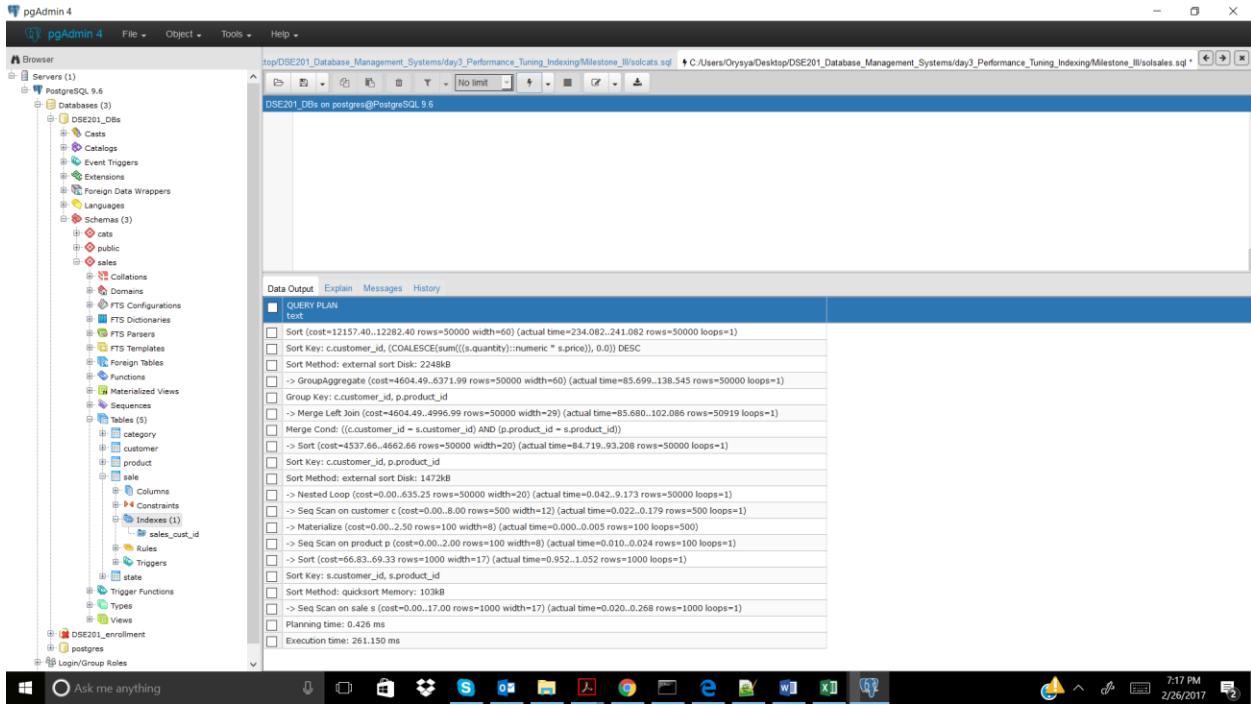
```

Execution Plan

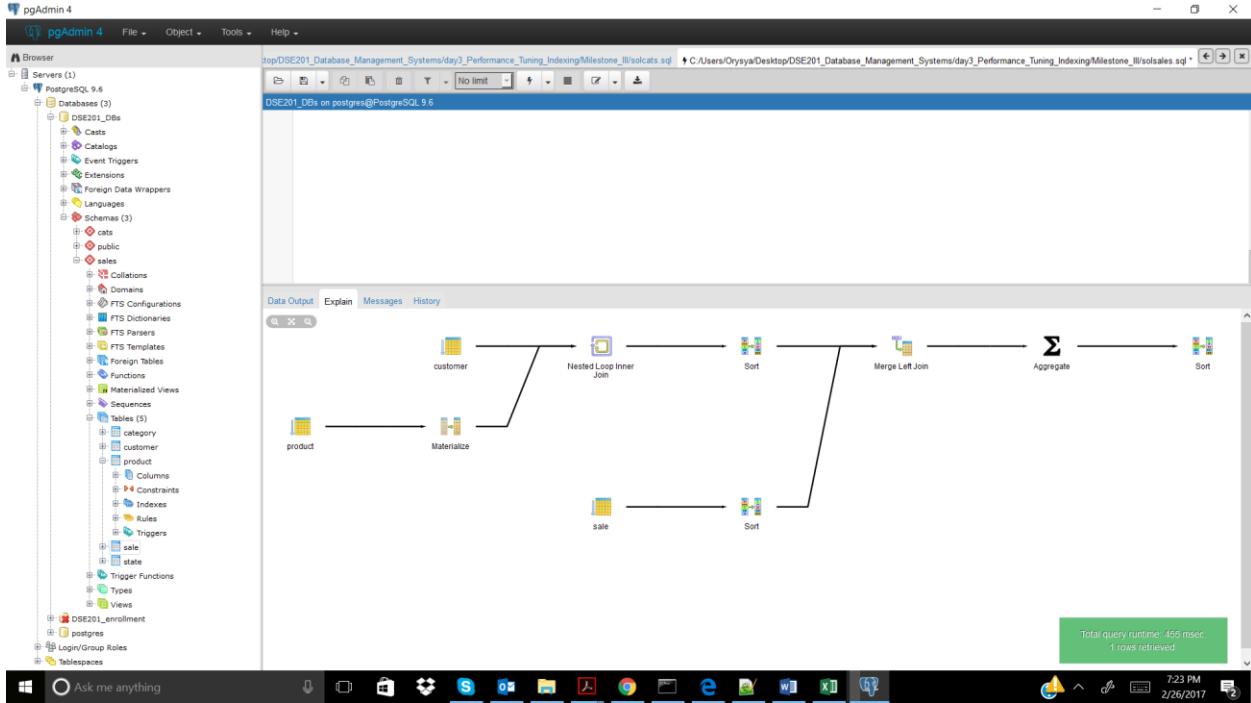
```

graph LR
    customer[customer] --> NestedJoin[Nested Loop Inner Join]
    product[product] --> Materialize[Materialize]
    NestedJoin --> Set[Set]
    Set --> MergeJoin[Merge Left Join]
    MergeJoin --> Aggregate[Aggregate]
    Aggregate --> Sort[Sort]
    Materialize --> Sort

```



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



pgAdmin 4

DSE201_DBs on postgres@PostgreSQL 9.6

```

top/DSE201_Database_Management_Systems/day3_Performance_Tuning_Indexing/Milestone_III/solcats.sql | C:/Users/Orysya/Desktop/DSE201_Database_Management_Systems/day3_Performance_Tuning_Indexing/Milestone_III/solsales.sql
No limit
DSE201_DBs on postgres@PostgreSQL 9.6
Data Output Explain Messages History
QUERY PLAN
Text
Sort (cost=12157.40..12282.40 rows=50000 width=60) (actual time=229.52..239.722 rows=50000 loops=1)
  Sort Key: c.customer_id, (COALESCE(sum((s.quantity)::numeric * s.price)), 0.0) DESC
  Sort Method: external sort Disk: 224kB
  >> GroupAggregate (cost=4604.49..6371.99 rows=50000 width=60) (actual time=70.80..119.859 rows=50000 loops=1)
    Group Key: c.customer_id, p.product_id
    >>> Merge Left Join (cost=4604.49..4990.99 rows=50000 width=29) (actual time=70.785..86.390 rows=50000 loops=1)
      Merge Cond: ((c.customer_id = s.customer_id) AND (p.product_id = s.product_id))
      >>> Sort (cost=4537.66..4662.66 rows=50000 width=20) (actual time=70.153..78.198 rows=50000 loops=1)
        Sort Key: c.customer_id, p.product_id
        Sort Method: external sort Disk: 1472kB
        >>> Nested Loop (cost=0.00..435.25 rows=50000 width=20) (actual time=0.066..9.021 rows=50000 loops=1)
          >>> Seq Scan on customer c (cost=0.00..8.00 rows=500 width=12) (actual time=0.030..0.149 rows=500 loops=1)
          >>> Materialize (cost=0.00..2.50 rows=100 width=8) (actual time=0.000..0.005 rows=100 loops=500)
          >>> Seq Scan on product p (cost=0.00..2.00 rows=100 width=8) (actual time=0.014..0.025 rows=100 loops=1)
          >>> Sort (cost=6.83..69.33 rows=1000 width=17) (actual time=0.626..0.704 rows=1000 loops=1)
            Sort Key: c.customer_id, p.product_id
            Sort Method: quicksort Memory: 103kB
            >>> Seq Scan on sales s (cost=0.00..17.00 rows=1000 width=17) (actual time=0.024..0.170 rows=1000 loops=1)
              Sort Key: s.customer_id, s.product_id
              Sort Method: quicksort Memory: 103kB
              >>> Aggregate (cost=0.00..0.00 rows=1 loops=1)
                >>> Planning time: 0.653 ms
                >>> Execution time: 254.799 ms
  Planning time: 0.653 ms
  Execution time: 254.799 ms

```

7:24 PM 2/26/2017

Sales_5 No Index

pgAdmin 4

DSE201_DBs on postgres@PostgreSQL 9.6

```

top/DSE201_Database_Management_Systems/day3_Performance_Tuning_Indexing/Milestone_III/solcats.sql | C:/Users/Orysya/Desktop/DSE201_Database_Management_Systems/day3_Performance_Tuning_Indexing/Milestone_III/solsales.sql
No limit
DSE201_DBs on postgres@PostgreSQL 9.6
Data Output Explain Messages History
EXPLAIN
Text
customer
  Nested Loop Inner Join
    customer
    Sort
    Merge Left Join
      Aggregate
      Sort
      Aggregate
      state
        Nested Loop Inner Join
          state
          Sort
          Merge Left Join
            CTE Scan
            Sort
product
  Materialize
  sales
    sort
category
  Materialize
  sales
    sort

```

7:30 PM 2/26/2017

pgAdmin 4

DSE201_DBs on postgres@PostgreSQL 9.6

```

SELECT * FROM sales.soldcats

```

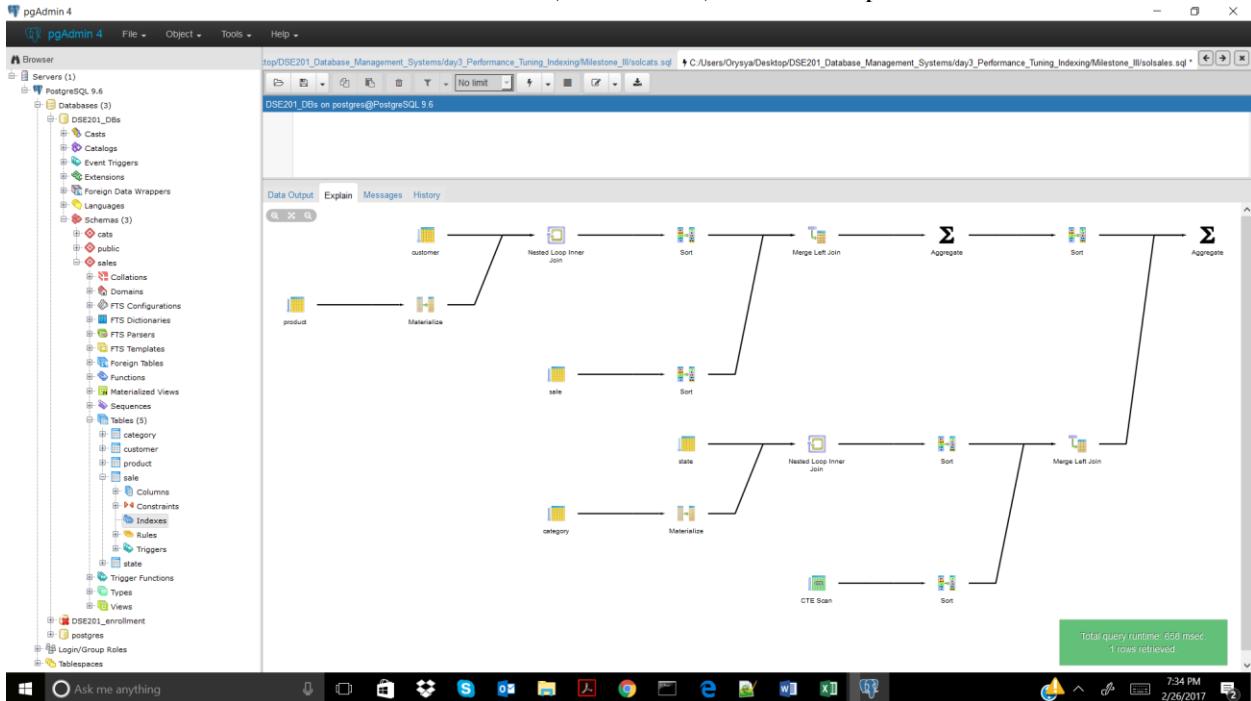
QUERY PLAN

```

text
-> Sort (cost=4537.66..4662.66 rows=50000 width=20) (actual time=74.745..84.311 rows=50000 loops=1)
  Sort Key: c_1.customer_id, p.product_id
  Sort Method: external sort Disk: 1472kB
-> Nested Loop (cost=0.00..635.25 rows=50000 width=20) (actual time=0.077..8.127 rows=50000 loops=1)
  -> Seq Scan on customer c_1 (cost=0.00..8.00 rows=500 width=12) (actual time=0.025..0.115 rows=500 loops=1)
  -> Materialize (cost=0.00..2.50 rows=100 width=8) (actual time=0.000..0.004 rows=100 loops=500)
    -> Seq Scan on product p (cost=0.00..2.00 rows=100 width=8) (actual time=0.043..0.064 rows=100 loops=1)
    -> Sort (cost=60.83..69.93 rows=1000 width=17) (actual time=1.951..2.081 rows=1000 loops=1)
      Sort Key: s_1.customer_id, s_1.product_id
      Sort Method: quicksort Memory: 103kB
-> Seq Scan on sale s_1 (cost=0.00..17.00 rows=1000 width=17) (actual time=0.070..0.683 rows=1000 loops=1)
-> Merge Left Join (cost=32370.32..35027.32 rows=237600 width=48) (actual time=369.882..395.987 rows=50010 loops=1)
  Merge Cond: ((s.state_id = c.state_id) AND (c.category_id = q.category_id))
  -> Sort (cost=27467.91..28061.91 rows=237600 width=48) (actual time=0.220..0.426 rows=500 loops=1)
    Sort Key: s.state_id, category_id
    Sort Method: quicksort Memory: 48kB
  -> Nested Loop (cost=0.00..3000.90 rows=237600 width=8) (actual time=0.065..0.138 rows=500 loops=1)
    -> Seq Scan on state s (cost=0.00..15.40 rows=540 width=4) (actual time=0.034..0.038 rows=50 loops=1)
    -> Materialize (cost=0.00..16.60 rows=440 width=4) (actual time=0.001..0.001 rows=10 loops=50)
      -> Seq Scan on category c (cost=0.00..14.40 rows=440 width=4) (actual time=0.012..0.013 rows=10 loops=1)
      -> Sort (cost=4902.41..5027.41 rows=50000 width=48) (actual time=369.650..383.826 rows=50000 loops=1)
        Sort Key: q.state_id, category_id
        Sort Method: external sort Disk: 1664kB
-> CTE Scan on q q (cost=0.00..1000.00 rows=50000 width=48) (actual time=228.269..257.235 rows=50000 loops=1)
Planning time: 1.111 ms
Execution time: 451.814 ms

```

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



pgAdmin 4

DSE201_DBs on postgres@PostgreSQL 9.6

File -> Object -> Tools -> Help ->

Browser

Databases (3)

- DS201_DBs
 - Cats
 - Catalogs
 - Event Triggers
 - Extensions
 - Foreign Data Wrappers
 - Languages
 - Schemas (3)
 - cats
 - public
 - sales
 - Tables (5)
 - category
 - customer
 - product
 - sale
 - state
 - Trigger Functions
 - Types
 - Views
- DSE201_enrollment
- postgres
- Tablespaces

Data Output Explain Messages History

QUERY PLAN

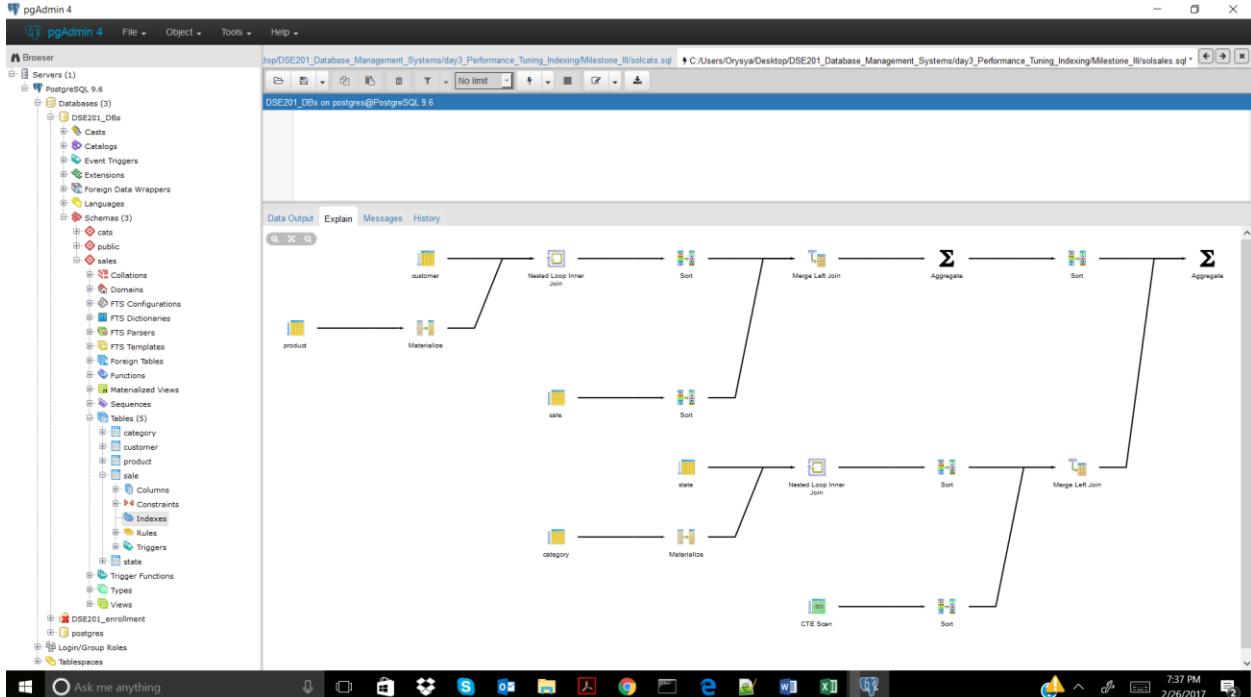
```

text
Merge Cond: ((c_1.customer_id = s_1.customer_id) AND (p.product_id = s_1.product_id))
-> Sort Key: c_1.customer_id, p.product_id
Sort Method: external sort Disk: 1472kB
-> Nested Loop (cost=0.00..4535.25 rows=50000 width=20) (actual time=0.020..7.063 rows=50000 loops=1)
    >- Seq Scan on customer c_1 (cost=0.00..8.00 rows=500 width=12) (actual time=0.010..0.117 rows=500 loops=1)
    >- Materialize (cost=0.00..2.50 rows=100 width=8) (actual time=0.000..0.004 rows=100 loops=500)
    >- Seq Scan on product p (cost=0.00..2.00 rows=100 width=8) (actual time=0.006..0.016 rows=100 loops=1)
    >- Sort (cost=6.83..59.33 rows=1000 width=17) (actual time=0.539..0.924 rows=1000 loops=1)
        Sort Key: s_1.customer_id, s_1.product_id
        Sort Method: quicksort Memory: 103kB
    >- Seq Scan on sale s_1 (cost=0.00..17.00 rows=1000 width=17) (actual time=0.055..0.231 rows=1000 loops=1)
-> Merge Left Join (cost=32370.32..35027.32 rows=237600 width=48) (actual time=378.866..407.117 rows=50010 loops=1)
    Merge Cond: ((s.state_id = q.state_id) AND (c.category_id = q.category_id))
    >- Sort (cost=2740.91..28061.91 rows=237600 width=48) (actual time=0.145..0.340 rows=500 loops=1)
        Sort Key: s.state_id, c.category_id
        Sort Method: quicksort Memory: 48kB
    >- Nested Loop (cost=0.00..3000.90 rows=237600 width=8) (actual time=0.020..0.090 rows=500 loops=1)
        >- Seq Scan on state s (cost=0.00..15.40 rows=540 width=4) (actual time=0.011..0.015 rows=50 loops=1)
        >- Materialize (cost=0.00..16.60 rows=440 width=4) (actual time=0.000..0.001 rows=10 loops=50)
        >- Seq Scan on category c (cost=0.00..14.40 rows=440 width=4) (actual time=0.005..0.007 rows=10 loops=1)
        >- Sort (cost=4902.41..5027.41 rows=50000 width=48) (actual time=378.714..393.218 rows=50000 loops=1)
            Sort Key: q.state_id, q.category_id
            Sort Method: external sort Disk: 1664kB
        >- CTE Scan on q q (cost=0.00..1000.00 rows=50000 width=48) (actual time=223.817..258.649 rows=50000 loops=1)
            Planning time: 0.843 ms
            Execution time: 465.621 ms

```

7:35 PM 2/26/2017

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



pgAdmin 4

DSE201_DBs on postgres@PostgreSQL 9.6

Data Output Explain Messages History

```

QUERY PLAN
text
Merge Cond: ((c_.1.customer_id = s_.1.customer_id) AND (p.product_id = s_.1.product_id))
--> Sort (cost=4537.66..4662.66 rows=50000 width=20) (actual time=66.887..80.152 rows=50000 loops=1)
  Sort Key: c_.1.customer_id, p.product_id
  Sort Method: external sort Disk: 1472kB
--> Nested Loop (cost=0.00..635.25 rows=50000 width=20) (actual time=0.074..7.587 rows=50000 loops=1)
    --> Seq Scan on customer c_1 (cost=0.00..8.00 rows=500 width=12) (actual time=0.020..0.124 rows=500 loops=1)
    --> Materialize (cost=0.00..2.50 rows=100 width=8) (actual time=0.000..0.004 rows=100 loops=500)
    --> Seq Scan on product p (cost=0.00..2.00 rows=100 width=8) (actual time=0.036..0.052 rows=100 loops=1)
    --> Sort (cost=65.83..69.33 rows=1000 width=17) (actual time=0.648..0.984 rows=1000 loops=1)
      Sort Key: s_.1.customer_id, s_.1.product_id
      Sort Method: quicksort Memory: 103kB
--> Seq Scan on sales s_1 (cost=0.00..17.00 rows=1000 width=17) (actual time=0.025..0.174 rows=1000 loops=1)
--> Merge Left Join (cost=32370.32..35027.32 rows=237600 width=48) (actual time=396.730..419.461 rows=50010 loops=1)
  Merge Cond: (s_.state_id = q.state_id) AND (c.category_id = q.category_id)
  --> Sort (cost=27467.91..28061.91 rows=237600 width=48) (actual time=0.253..0.401 rows=500 loops=1)
    Sort Key: s.state_id, c.category_id
    Sort Method: quicksort Memory: 48kB
  --> Nested Loop (cost=0.00..3000.90 rows=237600 width=48) (actual time=0.045..0.141 rows=500 loops=1)
    --> Seq Scan on state s (cost=0.00..15.40 rows=540 width=4) (actual time=0.030..0.034 rows=50 loops=1)
    --> Materialize (cost=0.00..16.60 rows=440 width=4) (actual time=0.000..0.001 rows=10 loops=50)
    --> Seq Scan on category c (cost=0.00..14.40 rows=440 width=4) (actual time=0.007..0.008 rows=50 loops=1)
    --> Sort (cost=4902.41..5027.41 rows=50000 width=48) (actual time=396.471..407.725 rows=50000 loops=1)
      Sort Key: q.state_id, q.category_id
      Sort Method: external sort Disk: 1664kB
    --> CTE Scan on q@q (cost=0.00..1000.00 rows=50000 width=48) (actual time=236.942..276.121 rows=50000 loops=1)
  Planning time: 1.002 ms
  Execution time: 472.518 ms

```

7:39 PM 2/26/2017

Sales_6 No Index

pgAdmin 4

DSE201_DBs on postgres@PostgreSQL 9.6

Data Output Explain Messages History

The diagram illustrates a highly complex query execution plan, likely due to the absence of indexes. It features numerous nested loops and joins between four main tables: customer, product, sales, and state. The plan is represented by a network of nodes connected by lines, where each node represents a specific operation like a scan or a sort. The connections between nodes represent the flow of data from one step to the next, forming a dense web of relationships that reflect the intricate logic of the query.

8:31 PM 2/26/2017

pgAdmin 4

DSE201_DBs on postgres@PostgreSQL 9.6

```

DSE201_DBs on postgres@PostgreSQL 9.6
No limit

Data Output Explain Messages History
QUERY PLAN
text
--> Seq Scan on state s_2 (cost=0.00..15.40 rows=540 width=4) (actual time=0.011..0.014 rows=50 loops=1)
--> Materialize (cost=0.00..16.60 rows=440 width=4) (actual time=0.000..0.001 rows=10 loops=50)
--> Seq Scan on category c_2 (cost=0.00..14.40 rows=440 width=4) (actual time=0.006..0.007 rows=10 loops=1)
--> Sort (cost=18950.31..19075.31 rows=50000 width=40) (actual time=198.782..204.890 rows=50000 loops=1)
--> Sort Key: c.state_id, c.category_id
--> Sort Method: quicksort Memory: 3710kB
--> Sort Scan on q (cost=14422.90..15047.90 rows=50000 width=40) (actual time=148.125..164.750 rows=50000 loops=1)
--> Sort (cost=14422.90..14547.90 rows=50000 width=174) (actual time=148.123..157.622 rows=50000 loops=1)
--> Sort Key: c_3.customer_id, (COALESCE(sum((c_3.quantity)::numeric * s_3.price)), 0)) DESC
--> Sort Method: external sort Disk: 1472kB
--> GroupAggregate (cost=4604.49..6246.99 rows=50000 width=174) (actual time=25.793..66.482 rows=50000 loops=1)
--> Group Key: c_3.customer_id, p_1.product_id
--> Merge Left Join (cost=4604.49..4996.99 rows=50000 width=25) (actual time=25.778..37.415 rows=50919 loops=1)
--> Merge Cond: ((c_3.customer_id = s_3.customer_id) AND (p_1.product_id = s_3.product_id))
--> Sort (cost=4537.66..4662.66 rows=50000 width=16) (actual time=25.066..30.007 rows=50000 loops=1)
--> Sort Key: c_3.customer_id, p_1.product_id
--> Sort Method: quicksort Memory: 3710kB
--> Nested Loop (cost=0.00..635.25 rows=50000 width=16) (actual time=0.022..11.240 rows=50000 loops=1)
--> > Seq Scan on customer c_3 (cost=0.00..8.00 rows=500 width=8) (actual time=0.009..0.230 rows=500 loops=1)
--> > Materialize (cost=0.00..2.50 rows=100 width=8) (actual time=0.000..0.006 rows=100 loops=500)
--> > Seq Scan on product p_1 (cost=0.00..2.00 rows=100 width=8) (actual time=0.009..0.020 rows=100 loops=1)
--> > Sort (cost=60.83..69.33 rows=1000 width=17) (actual time=0.706..0.806 rows=1000 loops=1)
--> > Sort Key: s_3.customer_id, s_3.product_id
--> > Sort Method: quicksort Memory: 1038B
--> > Seq Scan on sale s_3 (cost=0.00..17.00 rows=1000 width=17) (actual time=0.035..0.168 rows=1000 loops=1)
--> Planning time: 1.379 ms
--> Execution time: 439.620 ms

```

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

pgAdmin 4

DSE201_DBs on postgres@PostgreSQL 9.6

pgAdmin 4

DSE201_DBs on postgres@PostgreSQL 9.6

File -> Object -> Tools -> Help ->

Browser

Servers (1)

PostgreSQL 9.6

DSE201_DBs

- Cats
- Catalogs
- Event Triggers
- Extensions
- Foreign Data Wrappers
- Languages
- Schemas (3)
 - cats
 - public
 - sales
- Collations
- Domains
- FTS Configurations
- FTS Dictionaries
- FTS Parsers
- FTS Templates
- Foreign Tables
- Functions
- Materialized Views
- Sequences

Tables (5)

- category
- customer
- product
- sale
- state

- Columns
- Constraints
- Indexes (1)
 - sales_cust_id
- Rules
- Triggers

Trigger Functions

Types

Views

DSE201_enrollment

postgres

Login/Group Roles

Data Output Explain Messages History

QUERY PLAN

```

text
--> Seq Scan on state s_2 (cost=0.00..15.40 rows=540 width=4) (actual time=0.102..0.106 rows=50 loops=1)
--> Materialize (cost=0.00..16.60 rows=440 width=4) (actual time=0.001..0.002 rows=10 loops=50)
--> Sort (cost=18950.31..19075.31 rows=50000 width=40) (actual time=216.286..225.729 rows=50000 loops=1)
  Sort Key: q.state_id, q.category_id
  Sort Method: quicksort Memory: 3710kB
--> Subquery Scan on c (cost=14422.90..15047.90 rows=50000 width=40) (actual time=173.693..187.525 rows=50000 loops=1)
  Sort (cost=14422.90..14547.90 rows=50000 width=174) (actual time=173.689..181.204 rows=50000 loops=1)
  Sort Key: c_3.customer_id, (COALESCE(sum((c_3.quantity)::numeric * s_3.price)), 0.0) DESC
  Sort Method: external sort Disk: 1472kB
--> GroupAggregate (cost=4604.49..6246.99 rows=50000 width=174) (actual time=26.027..71.560 rows=50000 loops=1)
  Group Key: c_3.customer_id, p_1.product_id
--> Merge Left Join (cost=4604.49..4996.99 rows=50000 width=25) (actual time=26.010..39.116 rows=50919 loops=1)
  Merge Cond: ((c_3.customer_id = s_3.customer_id) AND (p_1.product_id = s_3.product_id))
  Sort (cost=4537.66..4662.66 rows=50000 width=16) (actual time=25.334..30.724 rows=50000 loops=1)
  Sort Key: c_3.customer_id, p_1.product_id
  Sort Method: quicksort Memory: 3710kB
--> Nested Loop (cost=0.00..635.25 rows=50000 width=16) (actual time=0.077..10.870 rows=50000 loops=1)
  --> Seq Scan on customer c_3 (cost=0.00..0.00 rows=500 width=8) (actual time=0.040..0.170 rows=500 loops=1)
  --> Materialize (cost=0.00..2.50 rows=100 width=8) (actual time=0.000..0.004 rows=100 loops=500)
  --> Seq Scan on product p_1 (cost=0.00..2.00 rows=100 width=8) (actual time=0.026..0.044 rows=100 loops=1)
  --> Sort (cost=66.83..69.93 rows=1000 width=17) (actual time=0.669..0.801 rows=1000 loops=1)
  Sort Key: s_3.customer_id, s_3.product_id
  Sort Method: quicksort Memory: 1038kB
--> Seq Scan on sale s_3 (cost=0.00..17.00 rows=1000 width=17) (actual time=0.046..0.169 rows=1000 loops=1)
  Planning time: 1.625 ms
  Execution time: 465.927 ms
  
```

9:03 PM 2/26/2017

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

pgAdmin 4

DSE201_DBs on postgres@PostgreSQL 9.6

File -> Object -> Tools -> Help ->

Browser

Servers (1)

PostgreSQL 9.6

DSE201_DBs

- Cats
- Catalogs
- Event Triggers
- Extensions
- Foreign Data Wrappers
- Languages
- Schemas (3)
 - cats
 - public
 - sales
- Collations
- Domains
- FTS Configurations
- FTS Dictionaries
- FTS Parsers
- FTS Templates
- Foreign Tables
- Functions
- Materialized Views
- Sequences

Tables (5)

- category
- customer
- product
- sale
- state

- Columns
- Constraints
- Indexes (1)
 - sales_prod_id
- Rules
- Triggers

Trigger Functions

Types

Views

DSE201_enrollment

postgres

Login/Group Roles

Data Output Explain Messages History

QUERY PLAN

The diagram illustrates a complex query execution plan. It features several nested loops and joins. At the top level, there is a 'Seq Scan on state s_2' followed by a 'Materialize' step. Below this, a 'Subquery Scan on c' performs a 'GroupAggregate' operation on columns customer_id and product_id. This is followed by a 'Merge Left Join' on customer and product tables. Further down, there are multiple 'Seq Scan' and 'Materialize' steps for tables category, customer, and product. Finally, a 'Sort' step is shown at the bottom. The entire process involves numerous temporary tables and disk usage, indicating a complex and resource-intensive query execution.

Planning time: 1.625 ms

Execution time: 465.927 ms

9:09 PM 2/26/2017

pgAdmin 4

DSE201_DBs on postgres@PostgreSQL 9.6

Data Output Explain Messages History

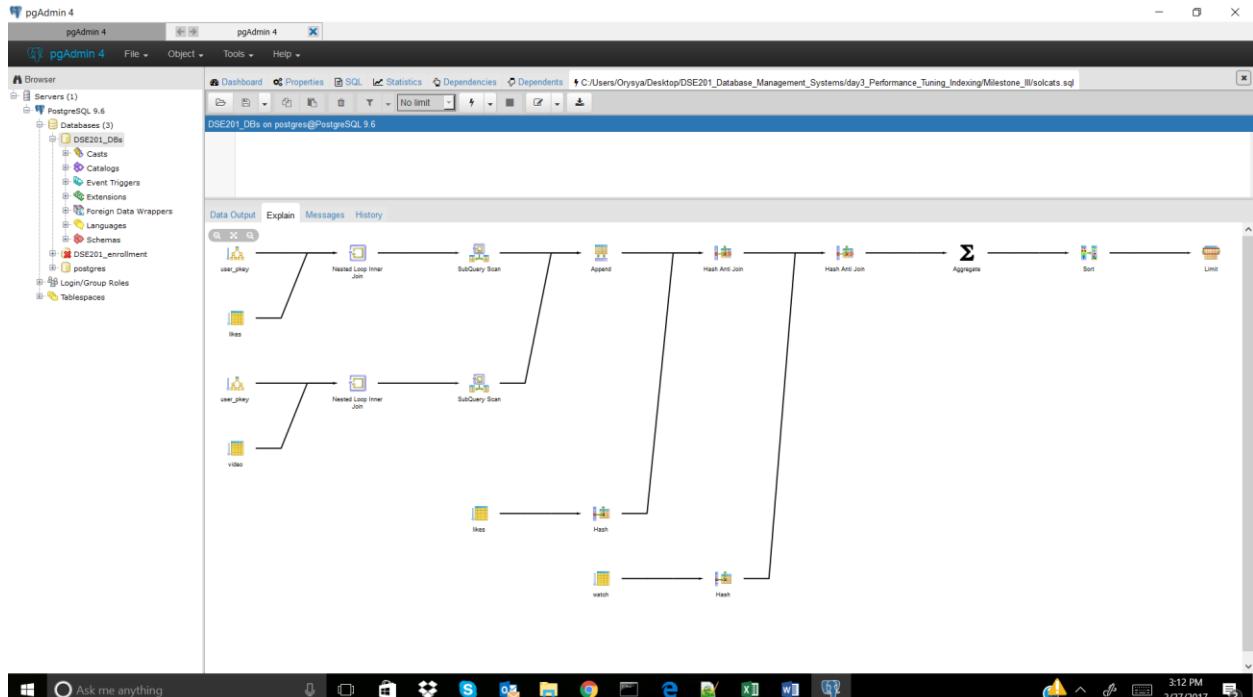
QUERY PLAN

```

text
--> Seq Scan on state s_2 (cost=0.00..15.40 rows=540 width=4) (actual time=0.016..0.018 rows=50 loops=1)
--> Materialize (cost=0.00..16.60 rows=440 width=4) (actual time=0.000..0.001 rows=10 loops=50)
--> Seq Scan on category c_2 (cost=0.00..14.40 rows=440 width=4) (actual time=0.007..0.008 rows=10 loops=50)
--> Sort (cost=18950.31..19075.31 rows=50000 width=40) (actual time=227.552..235.345 rows=50000 loops=1)
  Sort Key: q.state_id, q.category_id
  Sort Method: quicksort Memory: 3710kB
--> Subquery Scan on t (cost=14422.90..15047.90 rows=50000 width=40) (actual time=154.360..183.182 rows=50000 loops=1)
  Sort (cost=14422.90..14547.90 rows=50000 width=174) (actual time=154.357..166.961 rows=50000 loops=1)
  Sort Key: c_3.customer_id, (COALESCE(sum((c_3.quantity)::numeric * s_3.price)), 0.0) DESC
  Sort Method: external sort disk: 1472kB
  GroupAggregate (cost=4604.49..6246.99 rows=50000 width=174) (actual time=21.024..63.985 rows=50000 loops=1)
  Group Key: c_3.customer_id, p_1.product_id
  Merge Left Join (cost=4604.49..4996.99 rows=50000 width=25) (actual time=20.979..33.651 rows=50919 loops=1)
    Merge Cond: ((c_3.customer_id = s_3.customer_id) AND (p_1.product_id = s_3.product_id))
    Sort (cost=4537.66..4662.66 rows=50000 width=16) (actual time=19.766..24.867 rows=50000 loops=1)
    Sort Key: c_3.customer_id, p_1.product_id
    Sort Method: quicksort Memory: 3710kB
    Nested Loop (cost=0.00..635.25 rows=50000 width=16) (actual time=0.020..8.457 rows=50000 loops=1)
      --> Seq Scan on customer c_3 (cost=0.00..8.00 rows=500 width=8) (actual time=0.008..0.121 rows=500 loops=1)
      --> Materialize (cost=0.00..2.50 rows=100 width=8) (actual time=0.000..0.005 rows=100 loops=500)
      --> Seq Scan on product p_1 (cost=0.00..2.00 rows=100 width=8) (actual time=0.007..0.018 rows=100 loops=1)
      --> Sort (cost=66.83..69.33 rows=1000 width=17) (actual time=1.203..1.529 rows=1000 loops=1)
      Sort Key: s_3.customer_id, s_3.product_id
      Sort Method: quicksort Memory: 103kB
      --> Seq Scan on sale s_3 (cost=0.00..17.00 rows=1000 width=17) (actual time=0.047..0.299 rows=1000 loops=1)
      Planning time: 1.739 ms
      Execution time: 470.295 ms
  
```

Ask me anything 9:01 PM 2/26/2017

Cats_1 No Index



pgAdmin 4

pgAdmin 4 File Object Tools Help

Browser

DSE201_Dbs

Databases (3)

- DSE201_Db
- Cats
- Catalogs
- Event Triggers
- Extensions
- Foreign Data Wrappers
- Languages
- Schemas

DSE201_enrollment

postgres

Login/Group Roles

Tablespaces

Dashboard Properties SQL Statistics Dependencies Dependents C:/Users/Orysya/Desktop/DSE201_Database_Management_Systems/day3_Performance_Tuning_Indexing/Milestone_III/solcats.sql

No limit

DSE201_DBs on postgres@PostgreSQL 9.6

Data Output Explain Messages History

QUERY PLAN

text

```

Hash Cond: ((<*>SELECT 1.uid = l.user_id) AND (<*>SELECT 1.vid = l.video_id))
-> Append (cost=0.28..512.49 rows=12995 width=12) (actual time=0.174..7.074 rows=12995 loops=1)
  <-- Subquery Scan on "SELECT 1" (cost=0.28..397.22 rows=9996 width=12) (actual time=0.173..4.610 rows=9996 loops=1)
    <-- Nested Loop (cost=0.28..297.26 rows=9996 width=12) (actual time=0.171..3.708 rows=9996 loops=1)
      <-- Join Filter: (l.user_id <> user_id)
      <-- Rows Removed by Join Filter: 2
      <-- Index Scan using user_pkey on "user" u (cost=0.28..8.30 rows=1 width=4) (actual time=0.067..0.071 rows=1 loops=1)
      <-- Index Cond: (user_id = 13)
      <-- Heap Fetches: 1
      <-- Seq Scan on likes_l_1 (cost=0.00..163.98 rows=9998 width=8) (actual time=0.088..1.238 rows=9998 loops=1)
        <-- Subquery Scan on "SELECT 2" (cost=0.28..115.27 rows=2999 width=12) (actual time=0.061..1.621 rows=2999 loops=1)
          <-- Nested Loop (cost=0.28..85.28 rows=2999 width=12) (actual time=0.060..1.220 rows=2999 loops=1)
            <-- Index Only Scan using user_pkey on "user" u_1 (cost=0.28..8.30 rows=1 width=4) (actual time=0.016..0.019 rows=1 loops=1)
            <-- Index Cond: (user_id = 13)
            <-- Heap Fetches: 1
            <-- Seq Scan on video v (cost=0.00..46.99 rows=2999 width=4) (actual time=0.037..0.567 rows=2999 loops=1)
            <-- Hash (cost=188.99..188.98 rows=2 width=8) (actual time=1.424..1.434 rows=2 loops=1)
            <-- Buckets: 1024 Batches: 1 Memory Usage: 9KB
            <-- Seq Scan on likes_l (cost=0.00..188.98 rows=2 width=8) (actual time=0.910..1.417 rows=2 loops=1)
            <-- Filter: (user_id = 13)
            <-- Rows Removed by Filter: 9996
            <-- Hash (cost=377.98..377.98 rows=4 width=8) (actual time=2.045..2.045 rows=4 loops=1)
            <-- Buckets: 1024 Batches: 1 Memory Usage: 9KB
            <-- Seq Scan on watch w (cost=0.00..377.98 rows=4 width=8) (actual time=0.912..2.041 rows=4 loops=1)
            <-- Filter: (user_id = 13)
            <-- Rows Removed by Filter: 19994
          
```

3:12 PM 2/27/2017

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

pgAdmin 4

pgAdmin 4 File Object Tools Help

Browser

Schemas (3)

- cats
- event_triggers
- extensions
- foreign_data_wrappers
- languages
- schemas

Tables (7)

- friend
- likes
- login
- suggestion
- user
- video
- watch

Sequences

Dashboard Properties SQL Statistics Dependencies Dependents C:/Users/Orysya/Desktop/DSE201_Database_Management_Systems/day3_Performance_Tuning_Indexing/Milestone_III/solcats.sql

No limit

DSE201_DBs on postgres@PostgreSQL 9.6

Data Output Explain Messages History

QUERY PLAN

```

graph TD
    subgraph Likes [Likes]
        likes1[likes] --> NLJoin1[Nested Loop Inner Join]
        NLJoin1 --> SubQScan1[SubQuery Scan]
        SubQScan1 --> Append1[Append]
        Append1 --> HashAntiJoin1[Hash Anti Join]
        HashAntiJoin1 --> Agg1[Aggregate]
        Agg1 --> Sort1[Sort]
        Sort1 --> Limit1[Limit]
    end

    subgraph User [User]
        user_like1[user_like] --> NLJoin2[Nested Loop Inner Join]
        NLJoin2 --> SubQScan2[SubQuery Scan]
        SubQScan2 --> Append2[Append]
        Append2 --> HashAntiJoin2[Hash Anti Join]
        HashAntiJoin2 --> Agg2[Aggregate]
        Agg2 --> Sort2[Sort]
        Sort2 --> Limit2[Limit]
    end

    subgraph Video [Video]
        video1[video] --> Hash1[Hash]
        Hash1 --> HashAntiJoin3[Hash Anti Join]
        HashAntiJoin3 --> Agg3[Aggregate]
        Agg3 --> Sort3[Sort]
        Sort3 --> Limit3[Limit]
    end

    subgraph Watch [Watch]
        watch1[watch] --> Hash4[Hash]
        Hash4 --> HashAntiJoin4[Hash Anti Join]
        HashAntiJoin4 --> Agg4[Aggregate]
        Agg4 --> Sort4[Sort]
        Sort4 --> Limit4[Limit]
    end

    subgraph WatchUserIndex [watch_user_id_idx]
        watchUserIndex1[watch_user_id_idx] --> Hash5[Hash]
        Hash5 --> HashAntiJoin5[Hash Anti Join]
        HashAntiJoin5 --> Agg5[Aggregate]
        Agg5 --> Sort5[Sort]
        Sort5 --> Limit5[Limit]
    end

    Hash1 --> HashAntiJoin1
    HashAntiJoin1 --> HashAntiJoin2
    HashAntiJoin2 --> HashAntiJoin3
    HashAntiJoin3 --> HashAntiJoin4
    HashAntiJoin4 --> HashAntiJoin5
    HashAntiJoin5 --> HashAntiJoin1

```

3:17 PM 2/27/2017

pgAdmin 4

pgAdmin 4 File Object Tools Help

Browser DSE201 DBs on postgres@PostgreSQL 9.6

Data Output Explain Messages History

QUERY PLAN text

```

text
Join Filter: (l_1.user_id <> u.user_id)
Rows Removed by Join Filter: 2
-> Index Only Scan using user_pkey on "user" u (cost=0.28..8.30 rows=1 width=4) (actual time=0.011..0.013 rows=1 loops=1)
Index Cond: (user_id = 13)
Heap Fetches: 1
-> Seq Scan on likes l_1 (cost=0..0..163.98 rows=9998 width=8) (actual time=0.013..0.861 rows=9998 loops=1)
-> Subquery Scan on "SELECT" 2" (cost=0.28..115.27 rows=2999 width=12) (actual time=0.054..1.552 rows=2999 loops=1)
-> Nested Loop (cost=0.28..85.28 rows=2999 width=12) (actual time=0.054..1.191 rows=2999 loops=1)
-> Index Only Scan using user_pkey on "user" u_1 (cost=0.28..8.30 rows=1 width=4) (actual time=0.015..0.020 rows=1 loops=1)
Index Cond: (user_id = 13)
Heap Fetches: 1
-> Seq Scan on video v (cost=0.00..46.99 rows=2999 width=4) (actual time=0.035..0.574 rows=2999 loops=1)
-> Hash (cost=188.96..188.98 rows=2 width=8) (actual time=1.047..1.047 rows=2 loops=1)
Buckets: 1024 Batches: 1 Memory Usage: 9KB
-> Bitmap Heap Scan on likes l (cost=0.00..188.98 rows=2 width=8) (actual time=0.608..1.043 rows=2 loops=1)
Filter: (user_id = 13)
Rows Removed by Filter: 9996
-> Hash (cost=18.25..18.25 rows=4 width=8) (actual time=0.027..0.027 rows=4 loops=1)
Buckets: 1024 Batches: 1 Memory Usage: 9KB
-> Bitmap Heap Scan on watch_user_id_index w (cost=4.32..18.25 rows=4 width=8) (actual time=0.014..0.020 rows=4 loops=1)
Recheck Cond: (user_id = 13)
Heap Blocks: exact=4
-> Bitmap Index Scan on watch_user_id_index (cost=0.00..4.32 rows=4 width=8) (actual time=0.011..0.011 rows=4 loops=1)
Index Cond: (user_id = 13)
Planning time: 1.584 ms
Execution time: 20.548 ms

```

3:19 PM 2/27/2017

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

pgAdmin 4

pgAdmin 4 File Object Tools Help

Browser DSE201 DBs on postgres@PostgreSQL 9.6

Data Output Explain Messages History

QUERY PLAN text

```

text
Nested Loop Inter Join
SubQuery Scan
Append
Hash And Join
Hash And Join
Aggregates
Sort
Limit

```

3:21 PM 2/27/2017

pgAdmin 4

pgAdmin 4 pgAdmin 4

Browser DSE201 DBs on postgres@PostgreSQL 9.6

Data Output Explain Messages History

QUERY PLAN text

```

--> Subquery Scan on "SELECT" 1 (cost=0.28..397.22 rows=9996 width=12) (actual time=0.141..4.908 rows=9996 loops=1)
--> Nested Loop (cost=0.28..297.26 rows=9996 width=12) (actual time=0.138..3.903 rows=9996 loops=1)
  Join Filter: (l_1.user_id <> u.user_id)
  Rows Removed by Join Filter: 2
--> Index Only Scan using user_pkey on "user" u (cost=0.28..8.30 rows=1 width=4) (actual time=0.043..0.045 rows=1 loops=1)
  Index Cond: (user_id = 13)
--> Index Cond: (user_id = 13)
  Heap Fetches: 1
--> Seq Scan on likes_l_1 (cost=0.00..163.98 rows=9998 width=8) (actual time=0.046..1.237 rows=9998 loops=1)
--> Subquery Scan on "SELECT" 2 (cost=0.28..115.27 rows=2999 width=12) (actual time=0.039..0.773 rows=2999 loops=1)
--> Nested Loop (cost=0.28..85.28 rows=2999 width=12) (actual time=0.038..0.585 rows=2999 loops=1)
--> Index Only Scan using user_pkey on "user" u_1 (cost=0.28..8.30 rows=1 width=4) (actual time=0.008..0.010 rows=1 loops=1)
  Index Cond: (user_id = 13)
  Index Cond: (user_id = 13)
  Heap Fetches: 1
--> Seq Scan on video v (cost=0.00..46.99 rows=2999 width=4) (actual time=0.027..0.267 rows=2999 loops=1)
--> Hash (cost=188.98..188.98 rows=2 width=8) (actual time=3.418..3.418 rows=2 loops=1)
  Buckets: 1024 Batches: 1 Memory Usage: 9kB
--> Seq Scan on likes_l_1 (cost=0.00..188.98 rows=2 width=8) (actual time=2.593..3.345 rows=2 loops=1)
  Filter: (user_id = 13)
  Rows Removed by Filter: 9996
--> Hash (cost=377.98..377.98 rows=4 width=8) (actual time=2.193..2.193 rows=4 loops=1)
  Buckets: 1024 Batches: 1 Memory Usage: 9kB
--> Seq Scan on watch w (cost=0.00..377.98 rows=4 width=8) (actual time=1.058..2.180 rows=4 loops=1)
  Filter: (user_id = 13)
  Rows Removed by Filter: 19994
  Planning time: 0.655 ms
  Execution time: 25.409 ms

```

3:22 PM 2/27/2017

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

pgAdmin 4 pgAdmin 4

Browser DSE201 DBs on postgres@PostgreSQL 9.6

Data Output Explain Messages History

```

graph LR
    likes[likes] --> user[user]
    likes --> video[video]
    user --> likes
    user --> video
    video --> likes
    likes --> subq1[SubQuery Scan]
    user --> subq2[SubQuery Scan]
    video --> subq3[SubQuery Scan]
    subq1 --> nested1[Nested Loop Join]
    subq2 --> nested1
    nested1 --> append[Append]
    subq3 --> nested2[Nested Loop Join]
    nested2 --> hash1[Hash]
    likes --> hash2[Hash]
    hash1 --> hash3[Hash]
    hash2 --> hash3
    hash3 --> hash4[Hash]
    hash4 --> hash5[Hash]
    hash5 --> agg[Aggregate]
    agg --> sort[Sort]
    sort --> limit[Limit]

```

3:23 PM 2/27/2017

The screenshot shows the pgAdmin 4 interface with two windows. The left window displays the database schema browser for the 'DSE201_enrollment' database, showing various schemas, tables, and indexes. The right window shows the 'Data Output' tab of the query results, which includes the 'QUERY PLAN' section. The query plan details the execution steps, including joins, index scans, and heap scans, along with their costs and execution times.

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

This screenshot shows the pgAdmin 4 interface with the same schema browser and query results as the previous one. However, the focus is on the 'Explain' tab of the results window, which provides a detailed graphical representation of the query execution plan. The diagram illustrates the flow from the 'likes' table through nested loop joins and subquery scans to various hash and append operations, finally leading to an aggregate and sort step.

Cats_2

No Index

pgAdmin 4

DSE201 DBs on postgres@PostgreSQL 9.6

DATA OUTPUT

```

QUERY PLAN
text
-> Subquery Scan on "SELECT *" (cost=0.28..397.22 rows=9996 width=12) (actual time=0.064..5.373 rows=9996 loops=1)
  -> Nested Loop (cost=0.28..297.26 rows=9996 width=12) (actual time=0.063..4.298 rows=9996 loops=1)
    Join Filter: (l.user_id <> user_id)
    Rows Removed by Join Filter: 2
    -> Index Only Scan using user_pkey on "user" u (cost=0.28..8.30 rows=1 width=4) (actual time=0.026..0.030 rows=1 loops=1)
      Index Cond: (user_id = 13)
    -> Heap Fetches: 1
    -> Seq Scan on likes l_1 (cost=0.00..163.98 rows=9998 width=8) (actual time=0.031..1.561 rows=9998 loops=1)
      -> Subquery Scan on "SELECT *" (cost=0.28..115.27 rows=2999 width=12) (actual time=0.131..1.449 rows=2999 loops=1)
        -> Nested Loop (cost=0.28..115.28 rows=2999 width=12) (actual time=0.130..1.110 rows=2999 loops=1)
          -> Index Only Scan using user_pkey on "user" u_1 (cost=0.28..8.30 rows=1 width=4) (actual time=0.034..0.036 rows=1 loops=1)
            Index Cond: (user_id = 13)
          -> Heap Fetches: 1
          -> Seq Scan on video v (cost=0.00..46.99 rows=2999 width=4) (actual time=0.088..0.488 rows=2999 loops=1)
            -> Hash (cost=188.98..188.98 rows=2 width=8) (actual time=2.154..2.154 rows=2 loops=1)
            Buckets: 1024 Batches: 1 Memory Usage: 9KB
            -> Seq Scan on likes l (cost=0.00..188.98 rows=2 width=8) (actual time=1.282..2.141 rows=2 loops=1)
              Filter: (user_id = 13)
              Rows Removed by Filter: 9996
              -> Hash (cost=377.98..377.98 rows=4 width=8) (actual time=3.503..3.503 rows=4 loops=1)
              Buckets: 1024 Batches: 1 Memory Usage: 9KB
              -> Seq Scan on watch w (cost=0.00..377.98 rows=4 width=8) (actual time=1.779..3.478 rows=4 loops=1)
                Filter: (user_id = 13)
                Rows Removed by Filter: 19994
                Planning time: 0.899 ms
                Execution time: 30.617 ms
  
```

3:27 PM 2/27/2017

pgAdmin 4

DSE201 DBs on postgres@PostgreSQL 9.6

DATA OUTPUT

```

QUERY PLAN
text
+-> Subquery Scan on "SELECT *" (cost=0.28..397.22 rows=9996 width=12) (actual time=0.064..5.373 rows=9996 loops=1)
   +-> Nested Loop (cost=0.28..297.26 rows=9996 width=12) (actual time=0.063..4.298 rows=9996 loops=1)
     Join Filter: (l.user_id <> user_id)
     Rows Removed by Join Filter: 2
     +-> Index Only Scan using user_pkey on "user" u (cost=0.28..8.30 rows=1 width=4) (actual time=0.026..0.030 rows=1 loops=1)
       Index Cond: (user_id = 13)
     +-> Seq Scan on likes l_1 (cost=0.00..163.98 rows=9998 width=8) (actual time=0.031..1.561 rows=9998 loops=1)
       +-> Subquery Scan on "SELECT *" (cost=0.28..115.27 rows=2999 width=12) (actual time=0.131..1.449 rows=2999 loops=1)
         +-> Nested Loop (cost=0.28..115.28 rows=2999 width=12) (actual time=0.130..1.110 rows=2999 loops=1)
           +-> Index Only Scan using user_pkey on "user" u_1 (cost=0.28..8.30 rows=1 width=4) (actual time=0.034..0.036 rows=1 loops=1)
             Index Cond: (user_id = 13)
           +-> Hash (cost=188.98..188.98 rows=2 width=8) (actual time=2.154..2.154 rows=2 loops=1)
           Buckets: 1024 Batches: 1 Memory Usage: 9KB
           +-> Seq Scan on likes l (cost=0.00..188.98 rows=2 width=8) (actual time=1.282..2.141 rows=2 loops=1)
             Filter: (user_id = 13)
             Rows Removed by Filter: 9996
             +-> Hash (cost=377.98..377.98 rows=4 width=8) (actual time=3.503..3.503 rows=4 loops=1)
             Buckets: 1024 Batches: 1 Memory Usage: 9KB
             +-> Seq Scan on watch w (cost=0.00..377.98 rows=4 width=8) (actual time=1.779..3.478 rows=4 loops=1)
               Filter: (user_id = 13)
               Rows Removed by Filter: 19994
               Planning time: 0.899 ms
               Execution time: 30.617 ms
  
```

5:23 PM 2/27/2017

pgAdmin 4

pgAdmin 4 File Object Tools Help

Browser

Servers (1)

PostgreSQL 9.6

DSE201_Obs

Casts Catalogs Event Triggers Extensions Foreign Data Wrappers Languages Schemas (3)

cats friend likes login suggestion user video watch

Tables (7)

friend likes login suggestion user video watch

Trigger Functions Types Views public sales

Sequences

Triggers

Trigger Functions Types Views public sales

DSE201_enrollment

Dashboard Properties SQL Statistics Dependencies Dependents C:/Users/Orysya/Desktop/DSE201_Database_Management_Systems/day3_Performance_Tuning_Indexing/Milestone_III/solcats.sql

No limit

Data Output Explain Messages History

QUERY PLAN

text

Hash Cond: (t.friend_id = l_1.user_id)

> Nested Loop (cost=0.28..343.68 rows=19 width=8) (actual time=0.049..2.668 rows=20 loops=1)

 Join Filter: (t.friend_id <> u.user_id)

 -> Index Only Scan using user_pkey on "user" u (cost=0.28..8.30 rows=1 width=4) (actual time=0.020..0.022 rows=1 loops=1)

 Index Cond: (user_id = 13)

 Heap Fetches: 1

 -> Seq Scan on friend t (cost=0.00..335.14 rows=19 width=8) (actual time=0.025..2.620 rows=20 loops=1)

 Filter: (user_id = 13)

 Rows Removed by Filter: 19751

 -> Hash (cost=163.98..163.98 rows=9998 width=8) (actual time=2.671..2.671 rows=9998 loops=1)

 Buckets: 16384 Batches: 1 Memory Usage: 519KB

 -> Seq Scan on likes l_1 (cost=0.00..163.98 rows=9998 width=8) (actual time=0.011..1.351 rows=9998 loops=1)

 -> Subquery Scan on "SELECT 2" (cost=0.28..115.27 rows=2999 width=12) (actual time=0.042..0.742 rows=2999 loops=1)

 -> Nested Loop (cost=0.28..85.28 rows=2999 width=12) (actual time=0.042..0.546 rows=2999 loops=1)

 -> Index Only Scan using user_pkey on "user" u_1 (cost=0.28..8.30 rows=1 width=4) (actual time=0.009..0.010 rows=1 loops=1)

 Index Cond: (user_id = 13)

 Heap Fetches: 1

 -> Seq Scan on video v (cost=0.00..46.99 rows=2999 width=4) (actual time=0.030..0.246 rows=2999 loops=1)

 -> Hash (cost=188.98..188.98 rows=2 width=8) (actual time=1.055..1.055 rows=2 loops=1)

 Buckets: 1024 Batches: 1 Memory Usage: 9KB

 -> Seq Scan on likes l (cost=0.00..188.98 rows=2 width=8) (actual time=0.653..1.049 rows=2 loops=1)

 Filter: (user_id = 13)

 Rows Removed by Filter: 9996

 -> Hash (cost=377.98..377.98 rows=4 width=8) (actual time=2.230..2.230 rows=4 loops=1)

 Buckets: 1024 Batches: 1 Memory Usage: 9KB

 -> Seq Scan on watch w (cost=0.00..377.98 rows=4 width=8) (actual time=0.907..2.214 rows=4 loops=1)

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

5:26 PM 2/27/2017

pgAdmin 4

pgAdmin 4 File Object Tools Help

Browser

Servers (1)

PostgreSQL 9.6

DSE201_Obs

Casts Catalogs Event Triggers Extensions Foreign Data Wrappers Languages Schemas (3)

cats friend likes login suggestion user video watch

Tables (7)

friend likes login suggestion user video watch

Trigger Functions Types Views public sales

Sequences

Triggers

Trigger Functions Types Views public sales

DSE201_enrollment

postgres

Login/Group Roles

Tablespaces

Dashboard Properties SQL Statistics Dependencies Dependents C:/Users/Orysya/Desktop/DSE201_Database_Management_Systems/day3_Performance_Tuning_Indexing/Milestone_III/solcats.sql

No limit

Data Output Explain Messages History

QUERY PLAN

text

The diagram illustrates the query plan for the CREATE INDEX statement. It shows the flow of data from multiple tables (user, friend, likes, video, watch) through nested loops, hash joins, and a final aggregate operation to calculate the sum of values.

5:33 PM 2/27/2017

pgAdmin 4

pgAdmin 4 File Object Tools Help

Browser

Servers (1)

PostgreSQL 9.6

Databases (3)

DSE201_0bs

Casts Catalogs Event Triggers Extensions Foreign Data Wrappers Languages Schemas (3)

cats

Collations Domains FTS Configurations FTS Dictionaries FTS Parsers FTS Templates Foreign Tables Functions Materialized Views Sequences Tables (7)

friend likes login suggestion user video watch Trigger Functions Types Views

public sales

DSE201_enrollment

Postgres Login/Group Roles Tablespace

Dashboard Properties SQL Statistics Dependencies Dependents C:/Users/Orysya/Desktop/DSE201_Database_Management_Systems/day3_Performance_Tuning_Indexing/Milestone_III/solcats.sql

No limit

DSE201 DBs on postgres@PostgreSQL 9.6

```

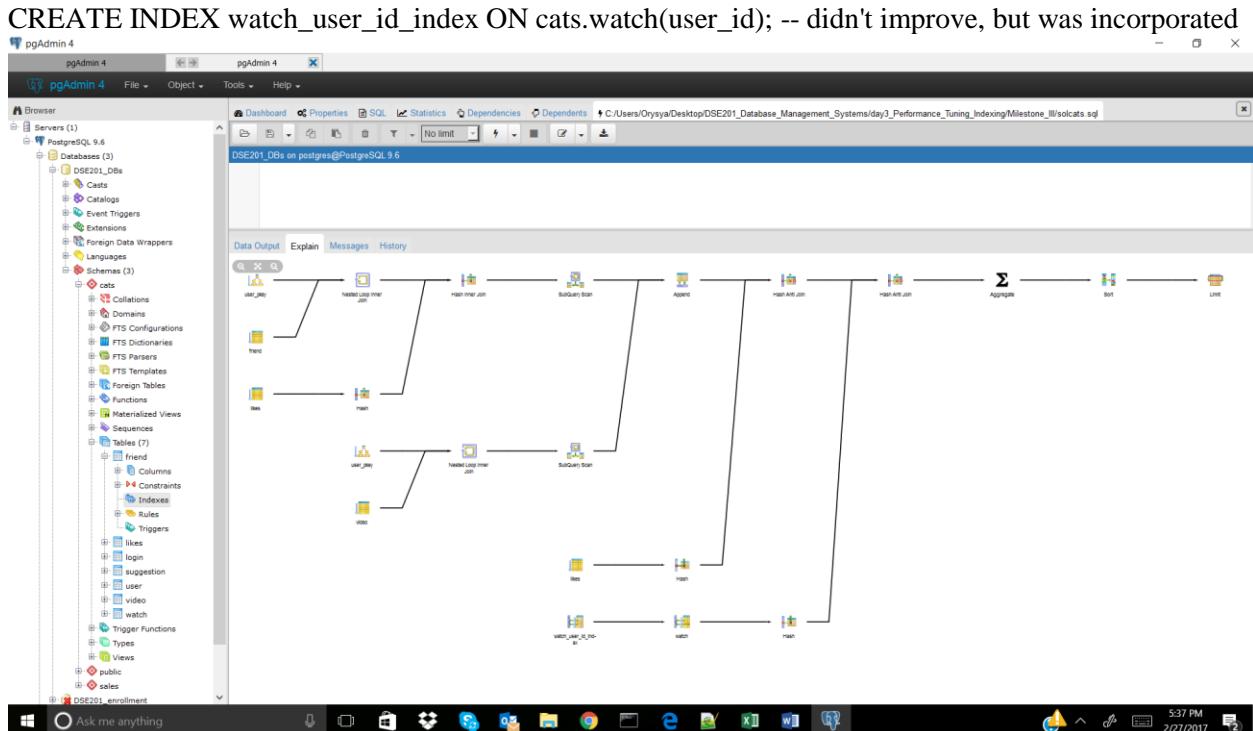
1 SET search_path TO cate;
2
3
4 -- CREATE INDEX login_user_id_index ON cats.login(user_id);
5 CREATE INDEX friend_user_id_index ON cats.friend(user_id);

```

Data Output Explain Messages History

QUERY PLAN text

-> Bitmap Heap Scan on friend f (cost=4.43..52.25 rows=19 width=8) (actual time=0.017..0.030 rows=20 loops=1)
 -> Recheck Cond: (user_id = 13)
 -> Head Blocks: exact=18
 -> Bitmap Index Scan on friend_user_id_index (cost=0.00..4.43 rows=19 width=0) (actual time=0.011..0.011 rows=20 loops=1)
 -> Index Cond: (user_id = 13)
 -> Hash (cost=163.98..163.98 rows=9998 width=8) (actual time=2.744..2.744 rows=9998 loops=1)
 -> Buckets: 1 Memory Usage: 519K
 -> Seq Scan on likes l_1 (cost=0.00..163.98 rows=9998 width=8) (actual time=0.011..1.382 rows=9998 loops=1)
 -> Subquery Scan on "SELECT 2" (cost=0.28..115.27 rows=2999 width=12) (actual time=0.020..1.062 rows=2999 loops=1)
 -> Nested Loop (cost=0.28..85.28 rows=2999 width=12) (actual time=0.019..0.832 rows=2999 loops=1)
 -> Index Only Scan using user_pkey on "user" u_1 (cost=0.28..8.30 rows=1 width=4) (actual time=0.006..0.008 rows=1 loops=1)
 -> Index Cond: (user_id = 13)
 -> Heap Fetches: 1
 -> Seq Scan on video v (cost=0.00..46.99 rows=2999 width=4) (actual time=0.011..0.441 rows=2999 loops=1)
 -> Hash (cost=188.98..188.98 rows=2 width=8) (actual time=0.926..0.926 rows=2 loops=1)
 -> Buckets: 1024 Batches: 1 Memory Usage: 9K
 -> Seq Scan on likes l (cost=0.00..188.98 rows=2 width=8) (actual time=0.528..0.922 rows=2 loops=1)
 -> Filter: (user_id = 13)
 -> Rows Removed by Filter: 9996
 -> Hash (cost=377.98..377.98 rows=4 width=8) (actual time=1.980..1.980 rows=4 loops=1)
 -> Buckets: 1024 Batches: 1 Memory Usage: 9K
 -> Seq Scan on user w (cost=0.00..377.98 rows=4 width=8) (actual time=0.864..1.976 rows=4 loops=1)
 -> Filter: (user_id = 13)
 -> Rows Removed by Filter: 19994
 -> Planning time: 1.000 ms
 -> Execution time: 10.790 ms



pgAdmin 4

pgAdmin 4 File Object Tools Help

Browser

Servers (1)

PostgreSQL 9.6

Databases (3)

DSE201_0bs

Casts Catalogs Event Triggers Extensions Foreign Data Wrappers Languages Schemas (3)

Tables (7)

friend likes login suggestion user video watch

Triggers Functions Materialized Views Sequences

Trigger Functions Types Views

public sales

DSE201_enrollment

Dashboard Properties SQL Statistics Dependencies Dependents C:/Users/Orysya/Desktop/DSE201_Database_Management_Systems/day3_Performance_Tuning_Indexing/Milestone_III/solcats.sql

No limit

Data Output Explain Messages History

QUERY PLAN text

```

-> Seq Scan on friend f (cost=0.00..335.14 rows=19 width=8) (actual time=0.053..1.838 rows=20 loops=1)
  Filter: (user_id = 13)
  Rows Removed by Filter: 19751
-> Hash (cost=163.98..163.98 rows=9998 width=8) (actual time=2.764..2.764 rows=9998 loops=1)
  Buckets: 16384 Batches: 1 Memory Usage: 519KB
-> Subquery Scan on "SELECT" 2" (cost=0.28..115.27 rows=2999 width=12) (actual time=0.024..0.673 rows=2999 loops=1)
-> Nested Loop (cost=0.28..85.28 rows=2999 width=12) (actual time=0.024..0.501 rows=2999 loops=1)
-> Index Only Scan using user_pkey on "user" u_1 (cost=0.28..8.30 rows=1 width=4) (actual time=0.005..0.006 rows=1 loops=1)
  Index Cond: (user_id = 13)
-> Seq Scan on friend f (cost=0.00..335.14 rows=19 width=8) (actual time=0.053..1.838 rows=20 loops=1)
  Filter: (user_id = 13)
  Rows Removed by Filter: 19751
-> Hash (cost=188.98..188.98 rows=2 width=8) (actual time=0.935..0.935 rows=2 loops=1)
  Buckets: 1024 Batches: 1 Memory Usage: 9KB
-> Subquery Scan on "SELECT" 2" (cost=0.28..115.27 rows=2999 width=12) (actual time=0.024..0.673 rows=2999 loops=1)
-> Nested Loop (cost=0.28..85.28 rows=2999 width=12) (actual time=0.024..0.501 rows=2999 loops=1)
-> Index Only Scan using user_pkey on "user" u_1 (cost=0.28..8.30 rows=1 width=4) (actual time=0.005..0.006 rows=1 loops=1)
  Index Cond: (user_id = 13)
-> Seq Scan on likes l (cost=0.00..188.98 rows=2 width=8) (actual time=0.547..0.533 rows=2 loops=1)
  Filter: (user_id = 13)
  Rows Removed by Filter: 9996
-> Hash (cost=18.25..18.25 rows=4 width=8) (actual time=0.025..0.025 rows=4 loops=1)
  Buckets: 1024 Batches: 1 Memory Usage: 9KB
-> Bitmap Heap Scan on watch w (cost=4.32..18.25 rows=4 width=8) (actual time=0.017..0.022 rows=4 loops=1)
  Recheck Cond: (user_id = 13)
-> Bitmap Index Scan on watch_user_id_index (cost=0.00..4.32 rows=4 width=0) (actual time=0.013..0.013 rows=4 loops=1)
-> Index Cond: (user_id = 13)
  Planning time: 0.846 ms
  Execution time: 11.797 ms
  
```

5:38 PM 2/27/2017

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

pgAdmin 4

pgAdmin 4 File Object Tools Help

Browser

Servers (1)

PostgreSQL 9.6

Databases (3)

DSE201_0bs

Casts Catalogs Event Triggers Extensions Foreign Data Wrappers Languages Schemas (3)

Tables (7)

friend likes login suggestion user video watch

Triggers Functions Materialized Views Sequences

Trigger Functions Types Views

public sales

DSE201_enrollment

Dashboard Properties SQL Statistics Dependencies Dependents C:/Users/Orysya/Desktop/DSE201_Database_Management_Systems/day3_Performance_Tuning_Indexing/Milestone_III/solcats.sql

No limit

Data Output Explain Messages History

```

graph LR
    user_ids[user_ids] --> Hash1[Hash]
    Hash1 --> NestedLoop1[Nested Loop join]
    NestedLoop1 --> Hash2[Hash]
    Hash2 --> NestedLoop2[Nested Loop join]
    NestedLoop2 --> Hash3[Hash]
    Hash3 --> SubQueryScan[SubQuery Scan]
    SubQueryScan --> Append[Append]
    Append --> Hash4[Hash]
    Hash4 --> NestedLoop3[Nested Loop join]
    NestedLoop3 --> Hash5[Hash]
    Hash5 --> Sort[Sort]
    Sort --> Limit[Limit]
    
```

5:40 PM 2/27/2017

pgAdmin 4

pgAdmin 4 File Object Tools Help

Browser

Servers (1)

PostgreSQL 9.6

Databases (3)

DSE201_0bs

Casts Catalogs Event Triggers Extensions Foreign Data Wrappers Languages Schemas (3)

cats

domains

fts configurations fts dictionaries fts parsers fts templates foreign tables functions materialized views sequences

Tables (7)

friend likes login suggestion user video watch

Columns Constraints Indexes Rules Triggers

Trigger Functions Types Views

public sales

DSE201_enrollment

Dashboard Properties SQL Statistics Dependencies Dependents C:/Users/Orysya/Desktop/DSE201_Database_Management_Systems/day3_Performance_Tuning_Indexing/Milestone_III/solcats.sql

No limit

Data Output Explain Messages History

QUERY PLAN

text

Index Cond: (user_id = 13)

Heap Fetches: 1

> Seq Scan on friend f (cost=0.00..335.14 rows=19 width=8) (actual time=0.021..3.871 rows=20 loops=1)

Filter: (user_id = 13)

Rows Removed by Filter: 19751

> Hash (cost=163.98..163.98 rows=9998 width=8) (actual time=2.709..2.709 rows=9998 loops=1)

Buckets: 16384 Batches: 1 Memory Usage: 519kB

> Seq Scan on likes_L1 (cost=0.00..163.98 rows=9998 width=8) (actual time=0.009..1.361 rows=9998 loops=1)

> Subquery Scan on "SELECT" (cost=0.28..115.27 rows=2999 width=12) (actual time=0.043..1.460 rows=2999 loops=1)

> Nested Loop (cost=0.28..115.28 rows=2999 width=12) (actual time=0.043..1.460 rows=2999 loops=1)

Filter: (user_id = 13)

> Index Only Scan using user_pkey on "user" u_1 (cost=0.28..8.30 rows=4 width=4) (actual time=0.011..0.014 rows=1 loops=1)

Index Cond: (user_id = 13)

Heap Fetches: 1

> Seq Scan on video v (cost=0.00..46.99 rows=2999 width=4) (actual time=0.031..0.417 rows=2999 loops=1)

> Hash (cost=188.98..188.98 rows=2 width=8) (actual time=1.076..1.076 rows=2 loops=1)

Buckets: 1024 Batches: 1 Memory Usage: 9kB

> Seq Scan on likes l (cost=0.00..188.98 rows=2 width=8) (actual time=0.642..1.070 rows=2 loops=1)

Filter: (user_id = 13)

Rows Removed by Filter: 9996

> Hash (cost=377.98..377.98 rows=4 width=8) (actual time=1.988..1.988 rows=4 loops=1)

Buckets: 1024 Batches: 1 Memory Usage: 9kB

> Seq Scan on watch w (cost=0.00..377.98 rows=4 width=8) (actual time=0.875..1.983 rows=4 loops=1)

Filter: (user_id = 13)

Rows Removed by Filter: 19994

Planning time: 0.888 ms

Execution time: 17.696 ms

5:40 PM 2/27/2017

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

pgAdmin 4

pgAdmin 4 File Object Tools Help

Browser

Servers (1)

PostgreSQL 9.6

Databases (3)

DSE201_0bs

Casts Catalogs Event Triggers Extensions Foreign Data Wrappers Languages Schemas (3)

cats

domains

fts configurations fts dictionaries fts parsers fts templates foreign tables functions materialized views sequences

Tables (7)

friend likes

Columns Constraints Indexes (1)

likes_user_id_idx

rules triggers

login suggestion user video watch

Trigger Functions Types Views

public sales

DSE201_enrollment

Dashboard Properties SQL Statistics Dependencies Dependents C:/Users/Orysya/Desktop/DSE201_Database_Management_Systems/day3_Performance_Tuning_Indexing/Milestone_III/solcats.sql

No limit

Data Output Explain Messages History

```

graph TD
    f[Seq Scan on friends f] --> H1[Hash Join]
    l[Seq Scan on likes l] --> H1
    H1 --> H2[Hash Join]
    v[Seq Scan on video v] --> H2
    l --> H3[Hash Join]
    w[Seq Scan on watch w] --> H3
    H3 --> H4[Hash Join]
    u1["Index Only Scan using user_pkey on user u_1"] --> H4
    H4 --> Agg[Aggregate]
    Agg --> Sort[Sort]
    
```

5:42 PM 2/27/2017

pgAdmin 4

pgAdmin 4 File Object Tools Help

Browser

Servers (1)

PostgreSQL 9.6

Databases (3)

DSE201_0bs

Casts Catalogs Event Triggers Extensions Foreign Data Wrappers Languages Schemas (3)

cats Collations Domains FTS Configurations FTS Dictionaries FTS Templates Foreign Tables Functions Materialized Views Sequences Tables (7)

friend likes Columns Constraints Indexes (1) likes_user_id_index Rules Triggers login suggestion user video watch Trigger Functions Types Views public sales

Dashboard Properties SQL Statistics Dependencies Dependents C:/Users/Orysya/Desktop/DSE201_Database_Management_Systems/day3_Performance_Tuning_Indexing/Milestone_III/solcats.sql

No limit Explain Messages History

QUERY PLAN text

```

-> Sort (cost=335.54..335.59 rows=19 width=8) (actual time=1.780..1.806 rows=37 loops=1)
  Sort Key: ffriend_id
  Sort Method: quicksort Memory: 25kB
-> Seq Scan on friend f (cost=0.00..335.14 rows=19 width=8) (actual time=0.019..1.767 rows=20 loops=1)
  Filter: (user_id = 13)
  Rows Removed by Filter: 19/51
-> Subquery Scan on "SELECT 2" (cost=0.28..115.27 rows=2999 width=12) (actual time=0.035..0.728 rows=2999 loops=1)
-> Nested Loop (cost=0.28..85.28 rows=2999 width=12) (actual time=0.035..0.526 rows=2999 loops=1)
-> Index Only Scan using user_pkkey on "user" u_1 (cost=0.28..8.30 rows=1 width=4) (actual time=0.006..0.007 rows=1 loops=1)
  Index Cond: (user_id = 13)
  Rows Removed by Filter: 19/51
-> Bitmap Heap Scan on likes (cost=4.30..11.26 rows=2 width=8) (actual time=0.012..0.014 rows=2 loops=1)
  Recheck Cond: (user_id = 13)
  Heap Blocks: exact=2
-> Bitmap Index Scan on likes_user_id_index (cost=0.00..4.30 rows=2 width=0) (actual time=0.009..0.009 rows=2 loops=1)
  Index Cond: (user_id = 13)
-> Hash (cost=377.98..377.98 rows=4 width=8) (actual time=2.494..2.494 rows=4 loops=1)
  Buckets: 1024 Batches: 1 Memory Usage: 9kB
-> Bitmap Heap Scan on watch w (cost=0.00..377.98 rows=4 width=8) (actual time=0.879..2.467 rows=4 loops=1)
  Filter: (user_id = 13)
  Rows Removed by Filter: 19994
  Planning time: 1.249 ms
  Execution time: 10.771 ms

```

5:43 PM 2/27/2017

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

pgAdmin 4

pgAdmin 4 File Object Tools Help

Browser

Servers (1)

PostgreSQL 9.6

Databases (3)

DSE201_0bs

Casts Catalogs Event Triggers Extensions Foreign Data Wrappers Languages Schemas (3)

cats Collations Domains FTS Configurations FTS Dictionaries FTS Templates Foreign Tables Functions Materialized Views Sequences Tables (7)

friend likes Columns Constraints Indexes (1) likes_user_id_index Rules Triggers login suggestion user video watch Trigger Functions Types Views public sales

Dashboard Properties SQL Statistics Dependencies Dependents C:/Users/Orysya/Desktop/DSE201_Database_Management_Systems/day3_Performance_Tuning_Indexing/Milestone_III/solcats.sql

No limit Explain Messages History

The diagram illustrates the query plan with nodes representing different database objects and operations:

- Tables:** user, friend, likes, user_day, video.
- Operations:**
 - Hash Join (between user and friend)
 - Hash Join (between friend and likes)
 - Sort (on likes)
 - Aggregate (summarizing likes)
 - Hash Join (between user_day and likes)
 - Hash Join (between video and likes)
 - Sort (on likes)
 - Aggregate (summarizing likes)
 - Sort (on likes)
 - Final Output (Line)

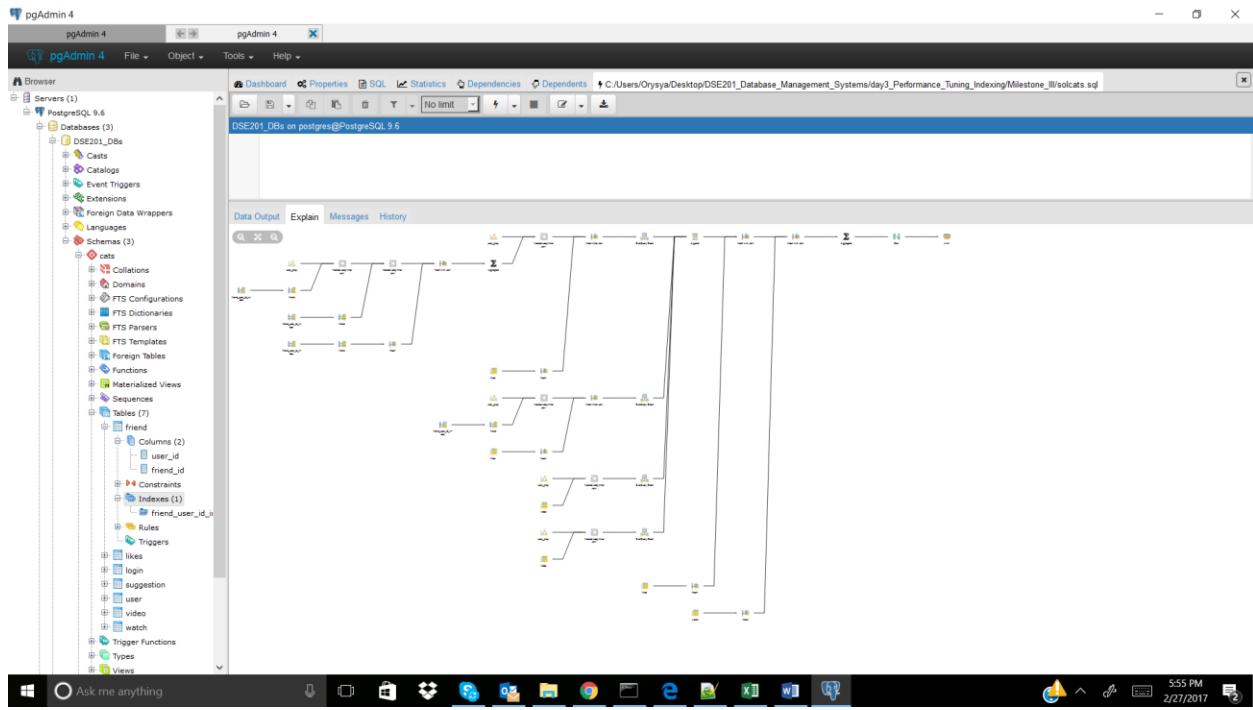
Planning time: 1.249 ms
Execution time: 10.771 ms

The screenshot shows the pgAdmin 4 interface with two panes. The left pane displays the database schema for 'DSE201'. The right pane shows the 'Data Output' tab of the Explain plan for a query. The explain plan details the execution steps, including Index Cond, Heap Fetches, and various Seq Scan and Hash operations, all of which involve multiple rows and loops, suggesting that no specific index is being utilized.

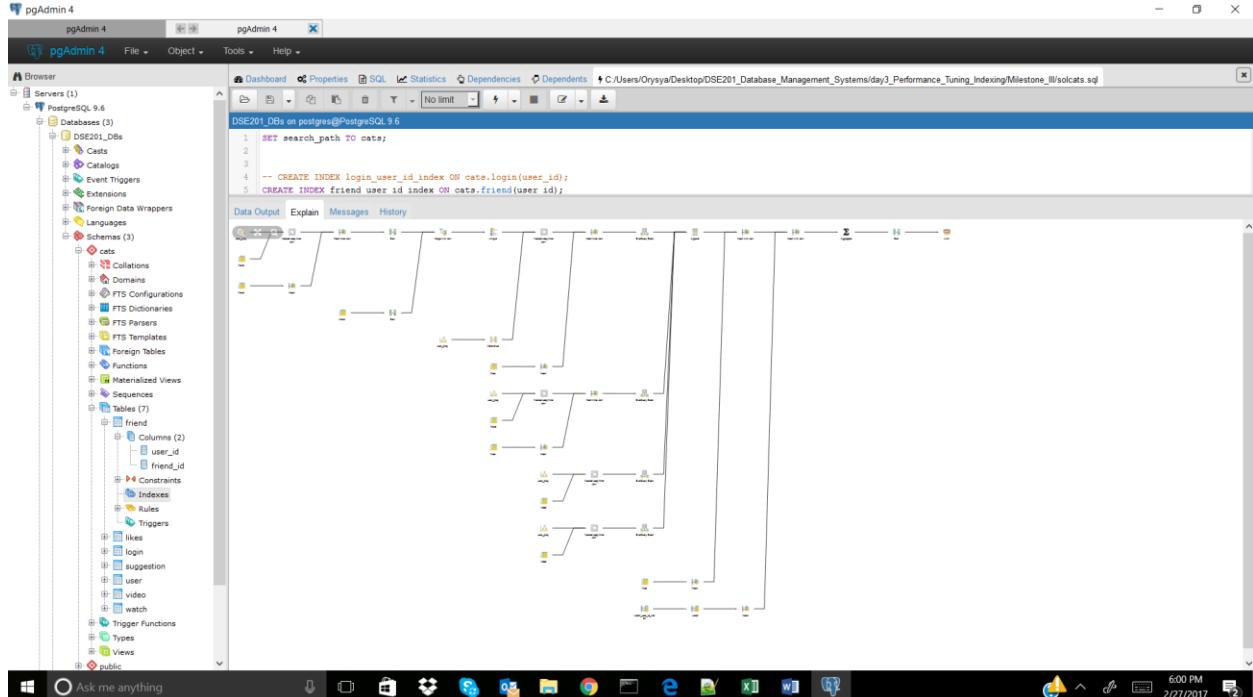
Cats_3
No index

The screenshot shows the pgAdmin 4 interface with two panes. The left pane displays the database schema for 'DSE201'. The right pane shows the 'Data Output' tab of the Explain plan for a query. The query includes commands to set the search path and create two indexes: 'login_user_id_index' on 'cats.login(user_id)' and 'friend_user_id_index' on 'cats.friend(user_id)'. The Explain plan shows a highly detailed and complex network of nodes, representing the optimizer's consideration of the new indexes on the 'user_id' column.

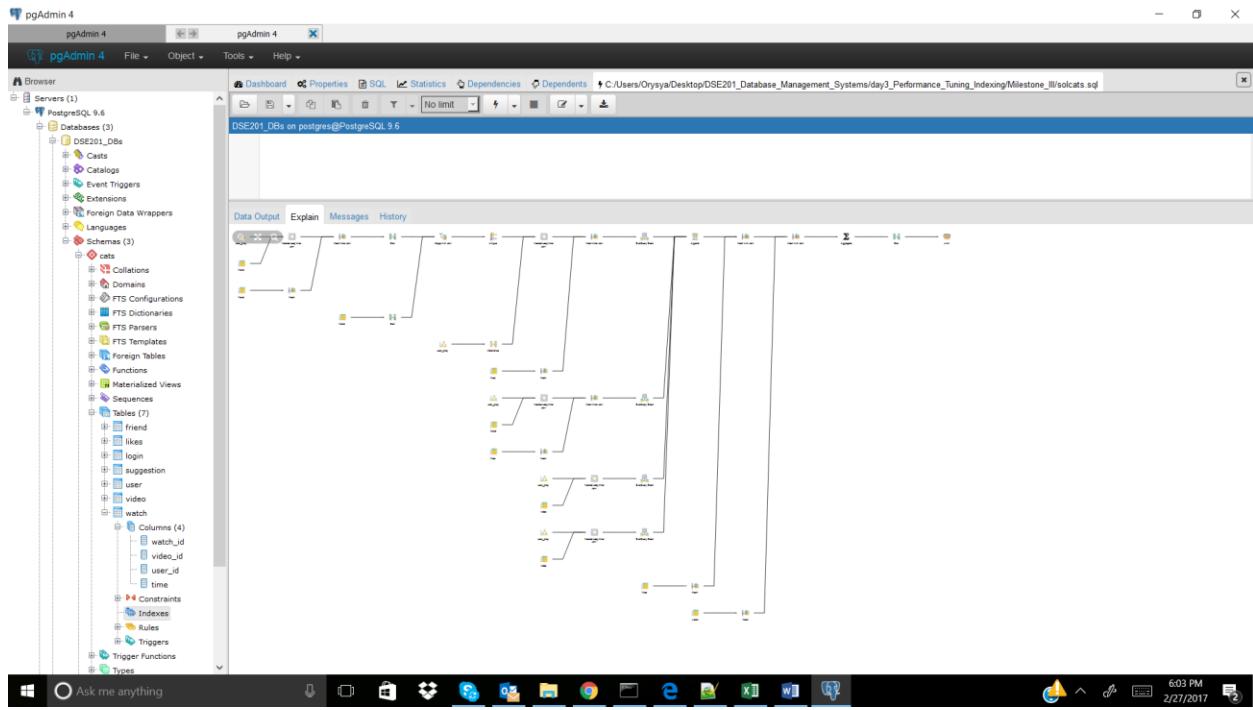
`CREATE INDEX friend_user_id_index ON cats.friend(user_id); -- didn't improve, is included`



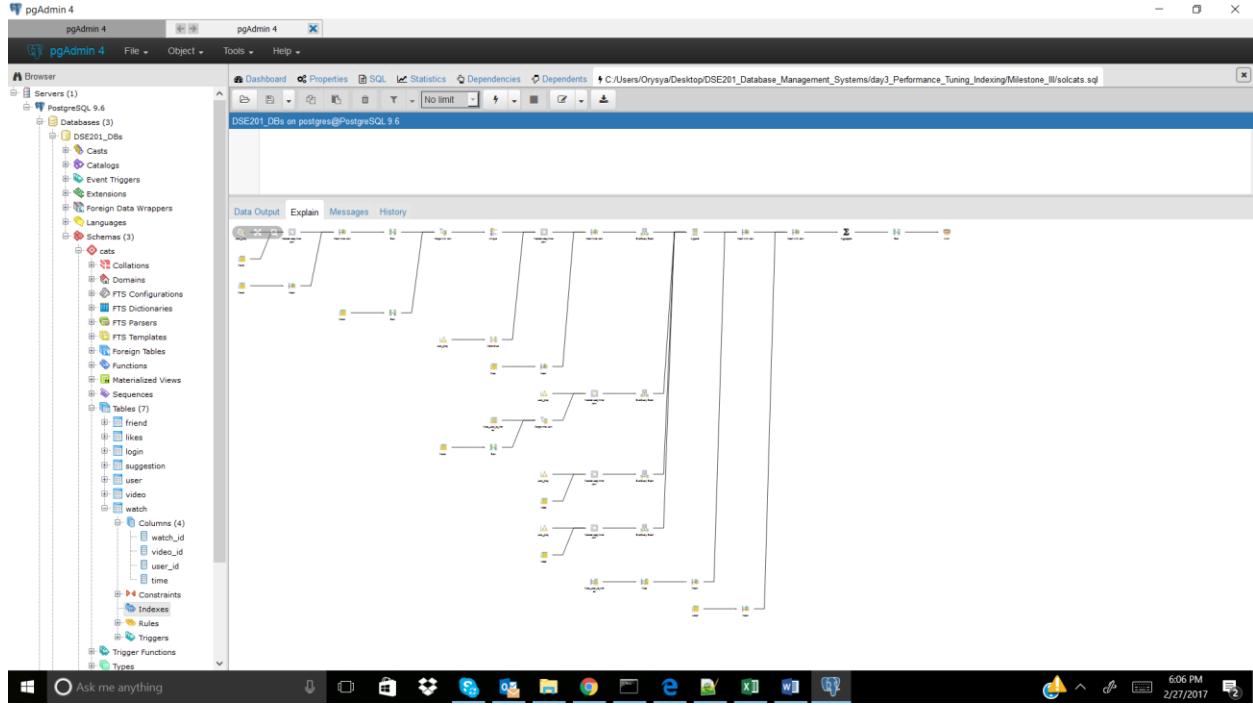
CREATE INDEX watch_user_id_index ON cats.watch(user_id); -- didn't improve, is included



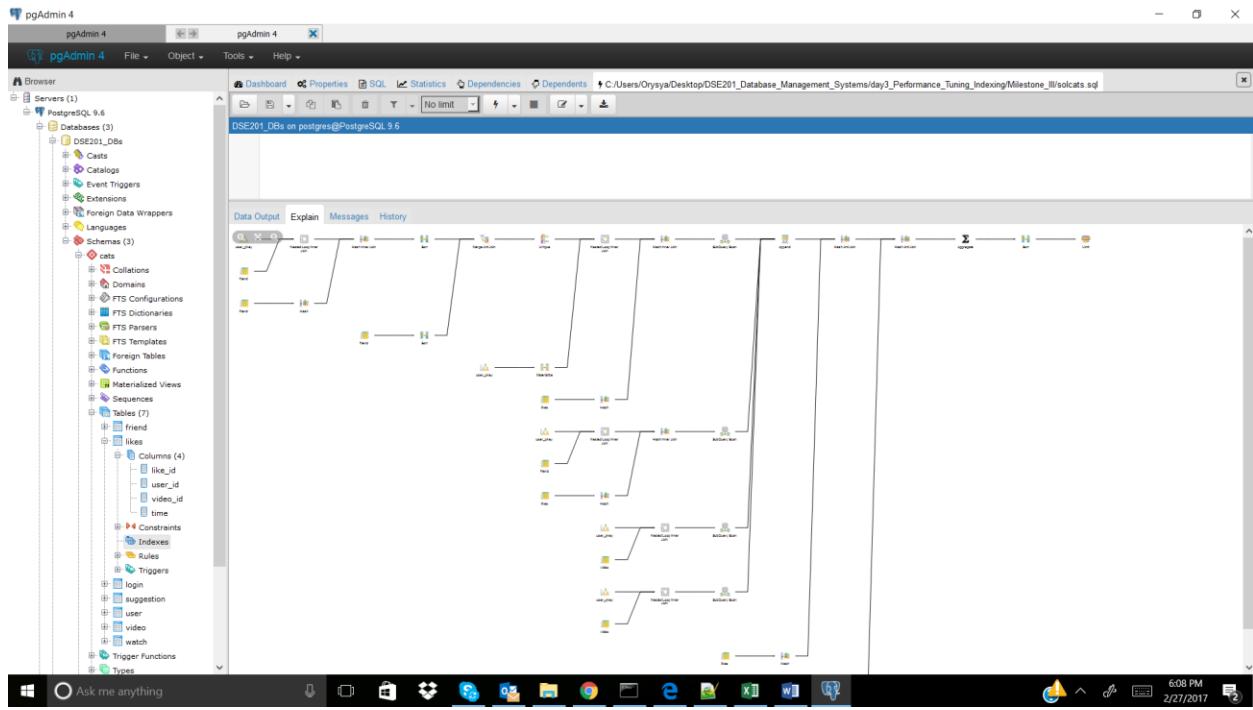
CREATE INDEX watch_video_id_index ON cats.watch(video_id); -- didn't improve, not included



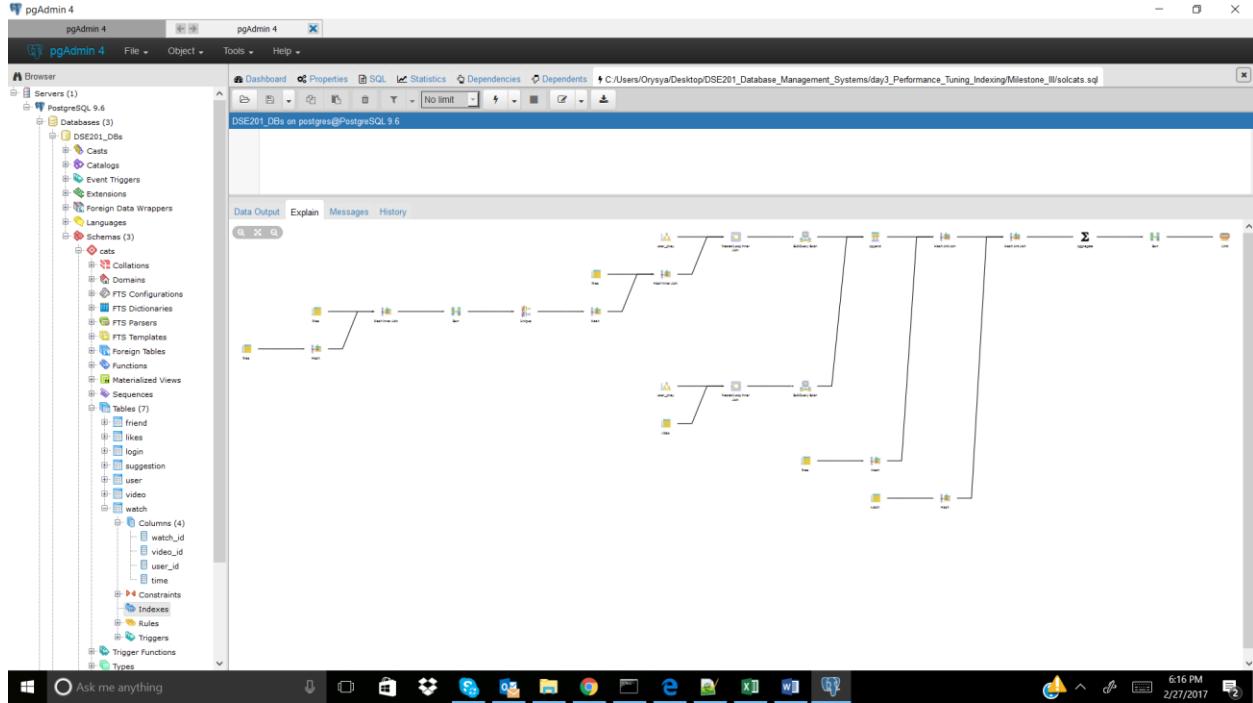
`CREATE INDEX likes_user_id_index ON cats.likes(user_id); -- didn't improve, is included`



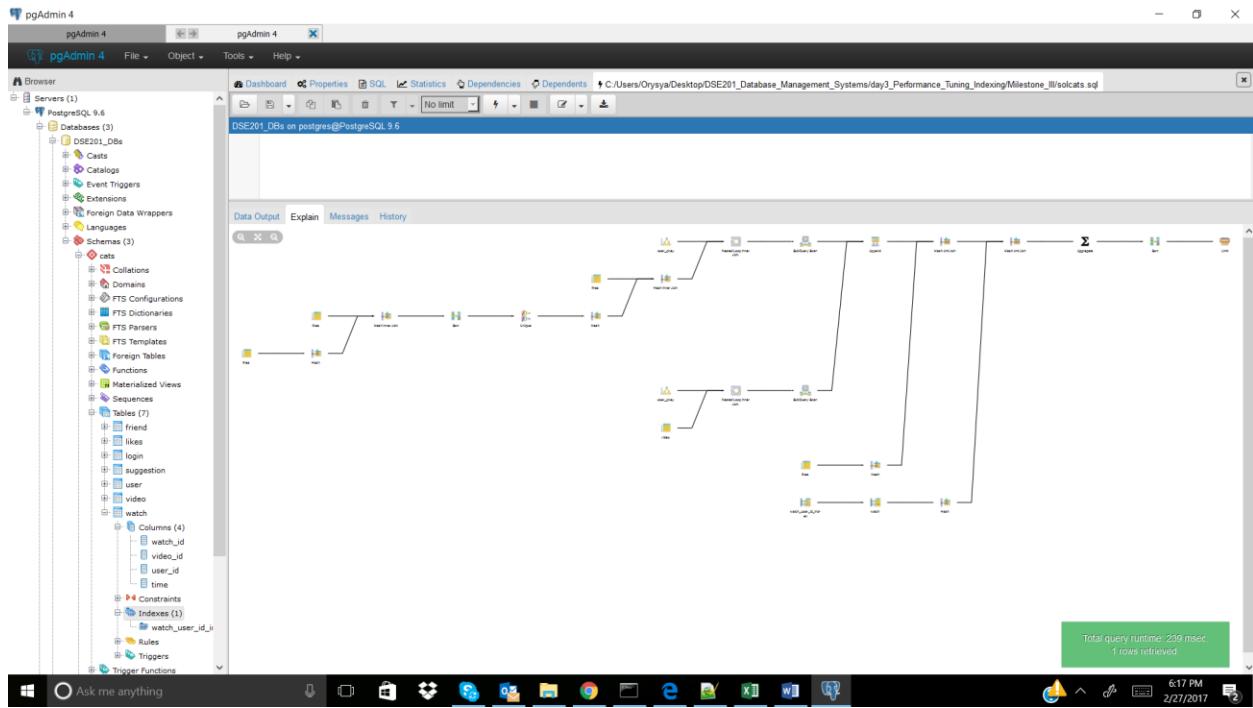
`CREATE INDEX likes_video_id_index ON cats.likes(video_id); -- didn't improve, not included`



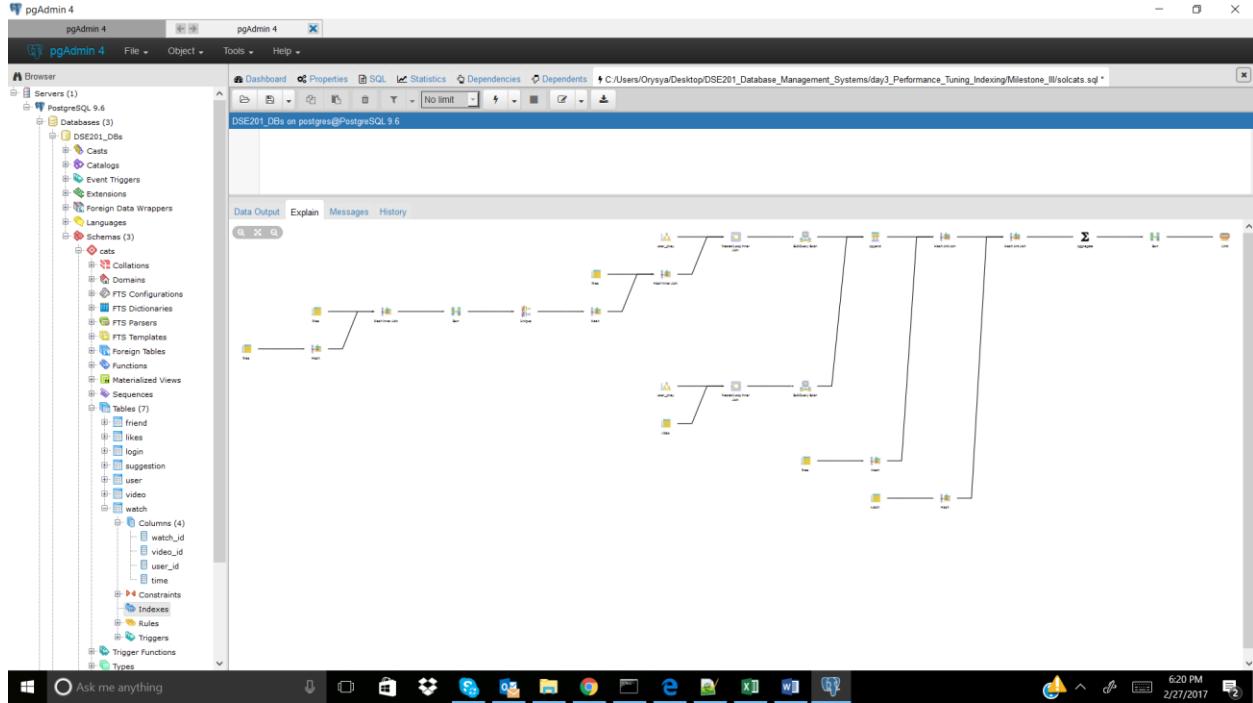
Cats_4
No Index



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



`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`

pgAdmin 4

pgAdmin 4 File Object Tools Help

Browser

Servers (1)

PostgreSQL 9.6

Databases (3)

DSE201_0bs

Cats Catalogs Event Triggers Extensions Foreign Data Wrappers Languages Schemas (3)

cats

Collations Domains FTS Configurations FTS Dictionaries FTS Parsers FTS Templates Foreign Tables Functions Materialized Views Sequences Tables (7)

friend likes login suggestion user video watch

Columns (4)

watch_id video_id user_id time

Constraints Indexes Rules Triggers Trigger Functions Types

Dashboard Properties SQL Statistics Dependencies Dependents C:/Users/Orysya/Desktop/DSE201_Database_Management_Systems/day3_Performance_Tuning_Indexing/Milestone_III/solcats.sql

No limit

DSE201 DBs on postgres@PostgreSQL 9.6

```

1 SET search_path TO cats;
2
3
4 -- CREATE INDEX login_user_id_index ON cats.login(user_id);
5 CREATE INDEX friend user id index ON cats.friend(user id);

```

Data Output Explain Messages History

Ask me anything

6:22 PM 2/27/2017

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

pgAdmin 4

pgAdmin 4 File Object Tools Help

Browser

Servers (1)

PostgreSQL 9.6

Databases (3)

DSE201_0bs

Cats Catalogs Event Triggers Extensions Foreign Data Wrappers Languages Schemas (3)

cats

Collations Domains FTS Configurations FTS Dictionaries FTS Parsers FTS Templates Foreign Tables Functions Materialized Views Sequences Tables (7)

friend likes

Columns (4)

like_id user_id video_id time

Constraints Indexes Rules Triggers Trigger Functions Types

Dashboard Properties SQL Statistics Dependencies Dependents C:/Users/Orysya/Desktop/DSE201_Database_Management_Systems/day3_Performance_Tuning_Indexing/Milestone_III/solcats.sql

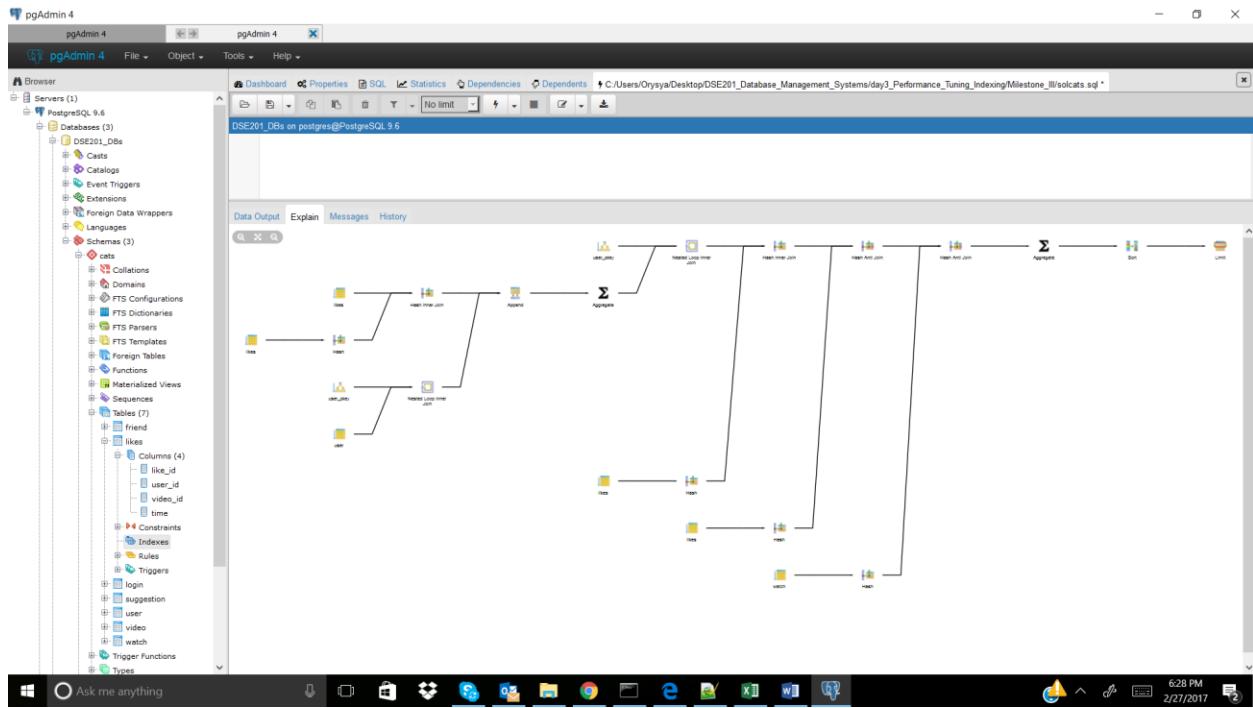
No limit

DSE201 DBs on postgres@PostgreSQL 9.6

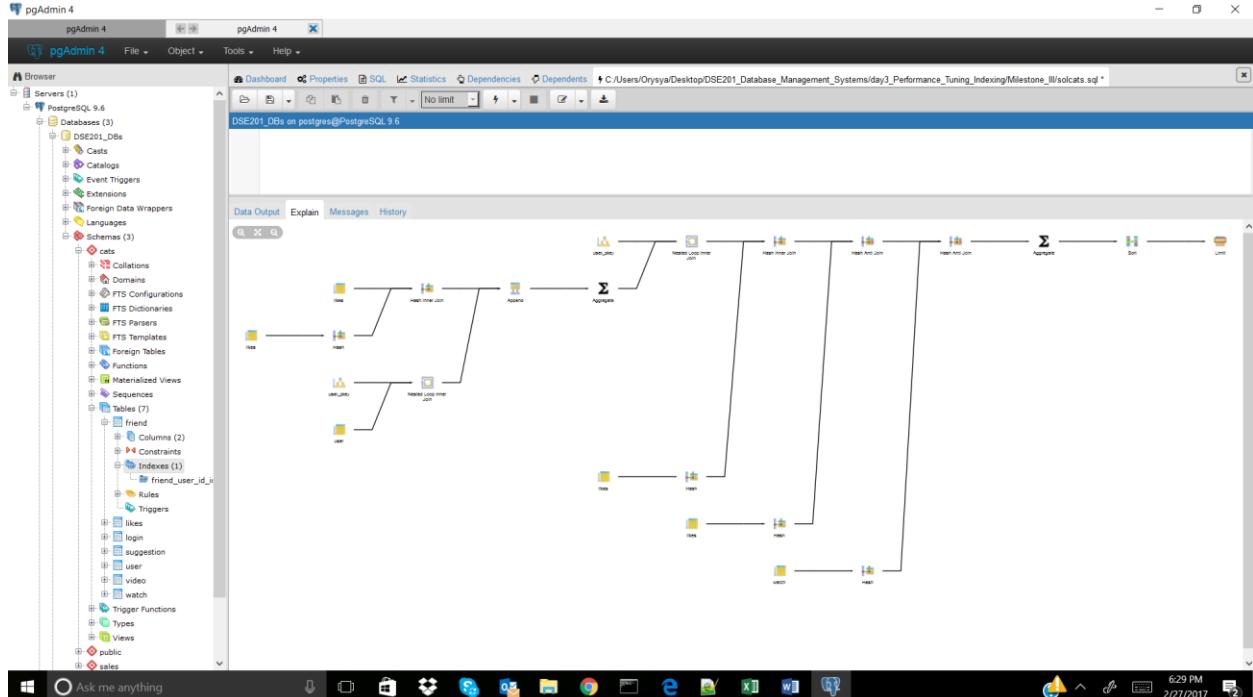
6:23 PM 2/27/2017

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`

pgAdmin 4

pgAdmin 4 File Object Tools Help

Browser

Servers (1)

PostgreSQL 9.6

Databases (3)

DSE201_0bs

Casts Catalogs Event Triggers Extensions Foreign Data Wrappers Languages Schemas (3)

Tables (7)

friend login likes suggestion user video watch

Trigger Functions Types Views

Sequences

DSE201_enrollment

Dashboard Properties SQL Statistics Dependencies Dependents C:/Users/Orysya/Desktop/DSE201_Database_Management_Systems/day3_Performance_Tuning_Indexing/Milestone_III/solcats.sql

No limit

1 SET search_path TO cats;

2

3

4 -- CREATE INDEX login_user_id_index ON cats.login(user_id);

5 CREATE INDEX friend_user_id_index ON cats.friend(user_id);

Data Output Explain Messages History

6:31 PM 2/27/2017

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

pgAdmin 4

pgAdmin 4 File Object Tools Help

Browser

Servers (1)

PostgreSQL 9.6

Databases (3)

DSE201_0bs

Casts Catalogs Event Triggers Extensions Foreign Data Wrappers Languages Schemas (3)

Tables (7)

friend login likes suggestion user video watch

Trigger Functions Types Views

Sequences

DSE201_enrollment

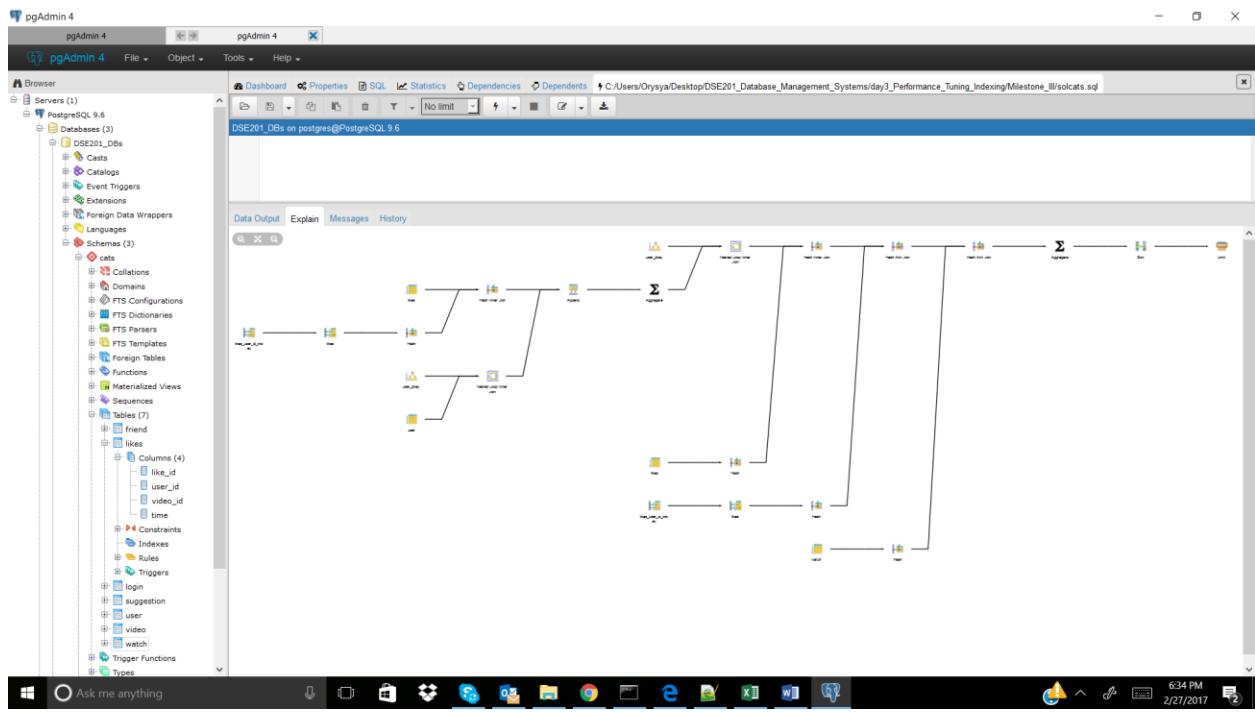
Dashboard Properties SQL Statistics Dependencies Dependents C:/Users/Orysya/Desktop/DSE201_Database_Management_Systems/day3_Performance_Tuning_Indexing/Milestone_III/solcats.sql

No limit

6:32 PM 2/27/2017

Total query runtime: 244 ms
1 rows retrieved

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



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