Database Technologies

UE19CS344

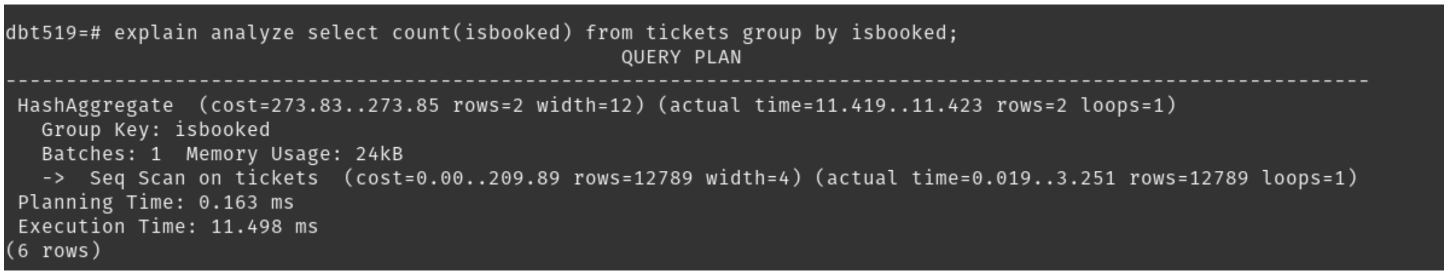
6th Semester, Academic Year 2021-22

Week #7: Query Optimization

Date: 8/3/2022

Name : SUMUKH RAJU BHAT	SRN : PES1UG19CS519	Section : H

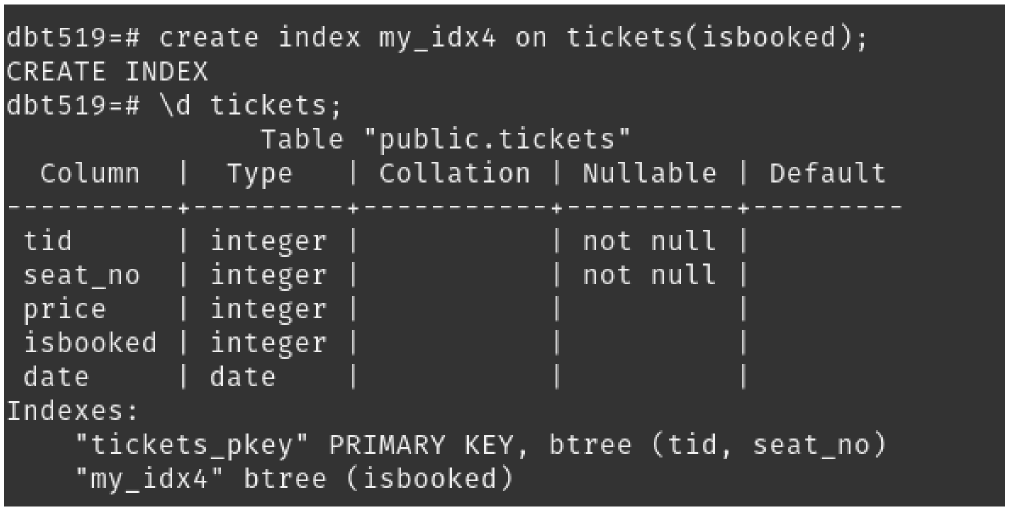
1. Queries containing group by:



```
dbt519=# explain analyze select count(isbooked) from tickets group by isbooked;
                                                   QUERY PLAN
-------------------------------------------------------------------------------------------------------------------
 HashAggregate  (cost=273.83..273.85 rows=2 width=12) (actual time=11.419..11.423 rows=2 loops=1)
   Group Key: isbooked
   Batches: 1  Memory Usage: 24kB
   ->  Seq Scan on tickets  (cost=0.00..209.89 rows=12789 width=4) (actual time=0.019..3.251 rows=12789 loops=1)
 Planning Time: 0.163 ms
 Execution Time: 11.498 ms
(6 rows)
```

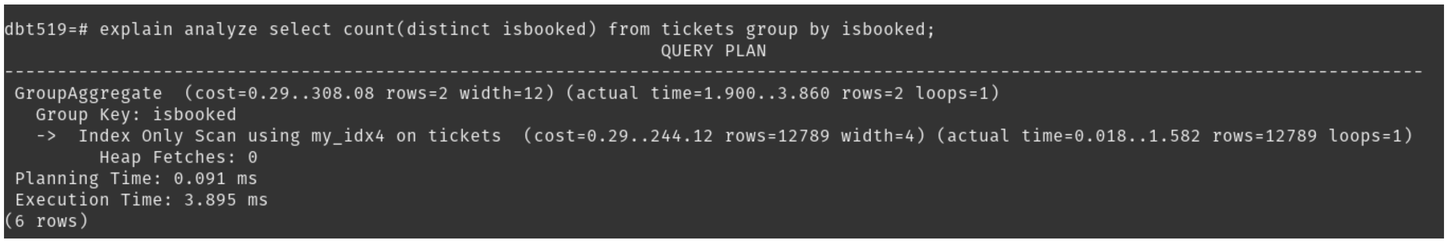
We can see that it hashes based on group key and then does a sequential scan on group key which is performance bottleneck for the query.

Hence, we add a secondary index on the group key:



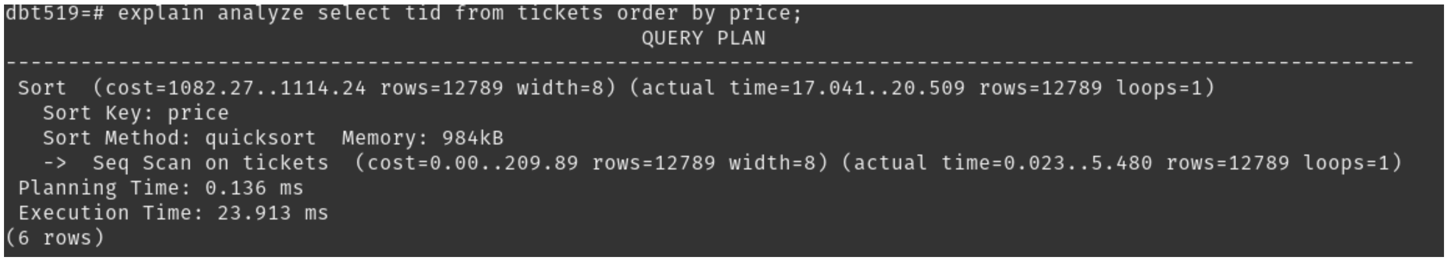
```
dbt519=# create index my_idx4 on tickets(isbooked);
CREATE INDEX
dbt519=# \d tickets;
               Table "public.tickets"
  Column  |  Type   | Collation | Nullable | Default
----------+---------+-----------+----------+---------
 tid      | integer |           | not null |
 seat_no  | integer |           | not null |
 price    | integer |           |          |
 isbooked | integer |           |          |
 date     | date    |           |          |
Indexes:
    "tickets_pkey" PRIMARY KEY, btree (tid, seat_no)
    "my_idx4" btree (isbooked)
```

We can now see that, index scan is performed using the new secondary index added. It results in approximately 4x faster execution:



```
dbt519=# explain analyze select count(distinct isbooked) from tickets group by isbooked;
                                                              QUERY PLAN
------------------------------------------------------------------------------------------------------------------------------------
 GroupAggregate  (cost=0.29..308.08 rows=2 width=12) (actual time=1.900..3.860 rows=2 loops=1)
   Group Key: isbooked
   ->  Index Only Scan using my_idx4 on tickets  (cost=0.29..244.12 rows=12789 width=4) (actual time=0.018..1.582 rows=12789 loops=1)
         Heap Fetches: 0
 Planning Time: 0.091 ms
 Execution Time: 3.895 ms
(6 rows)
```

2. Queries containing Order by:

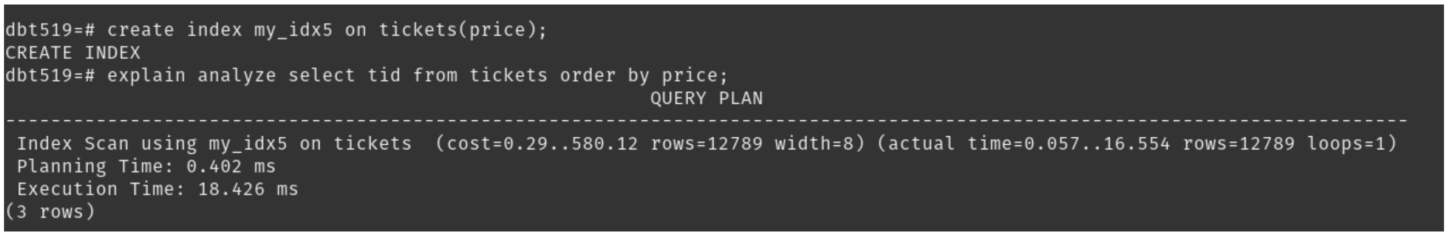


```
dbt519=# explain analyze select tid from tickets order by price;
                                                        QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------
 Sort  (cost=1082.27..1114.24 rows=12789 width=8) (actual time=17.041..20.509 rows=12789 loops=1)
   Sort Key: price
   Sort Method: quicksort  Memory: 984kB
   ->  Seq Scan on tickets  (cost=0.00..209.89 rows=12789 width=8) (actual time=0.023..5.480 rows=12789 loops=1)
 Planning Time: 0.136 ms
 Execution Time: 23.913 ms
(6 rows)
```

We can see that it does quicksort using sequential scan on the sort key, which is definetely a performance bottleneck for the query.

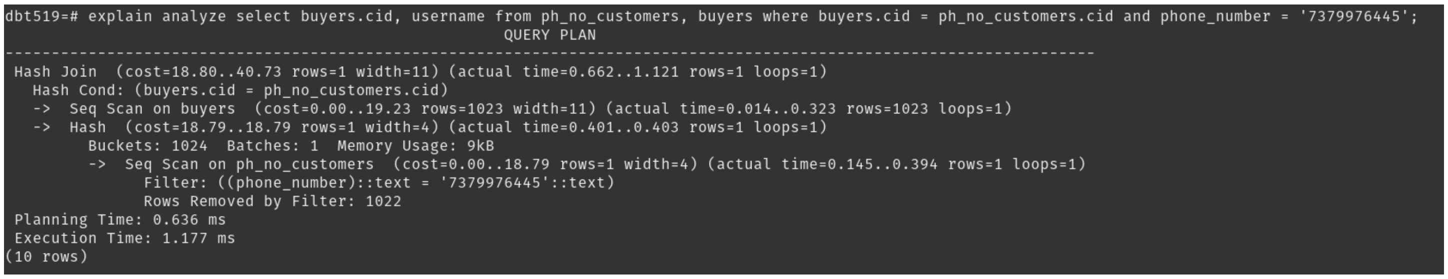
Hence we add a secondary index on the sort key.

Now we can see index scan being performed and decrease in execution time:



```
dbt519=# create index my_idx5 on tickets(price);
CREATE INDEX
dbt519=# explain analyze select tid from tickets order by price;
                                                            QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------------------
 Index Scan using my_idx5 on tickets  (cost=0.29..580.12 rows=12789 width=8) (actual time=0.057..16.554 rows=12789 loops=1)
 Planning Time: 0.402 ms
 Execution Time: 18.426 ms
(3 rows)
```

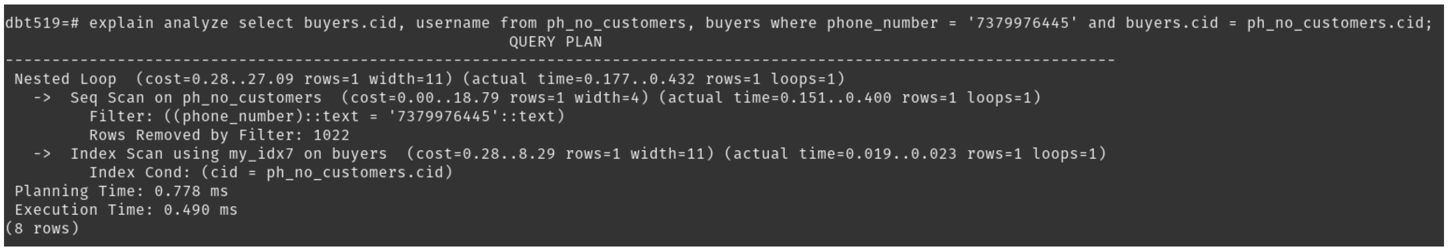
3. Queries containing equijoin:



```
dbt519=# explain analyze select buyers.cid, username from ph_no_customers, buyers where buyers.cid = ph_no_customers.cid and phone_number = '7379976445';
                                                      QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------
 Hash Join  (cost=18.80..40.73 rows=1 width=11) (actual time=0.662..1.121 rows=1 loops=1)
   Hash Cond: (buyers.cid = ph_no_customers.cid)
   ->  Seq Scan on buyers  (cost=0.00..19.23 rows=1023 width=11) (actual time=0.014..0.323 rows=1023 loops=1)
   ->  Hash  (cost=18.79..18.79 rows=1 width=4) (actual time=0.401..0.403 rows=1 loops=1)
         Buckets: 1024  Batches: 1  Memory Usage: 9kB
         ->  Seq Scan on ph_no_customers  (cost=0.00..18.79 rows=1 width=4) (actual time=0.145..0.394 rows=1 loops=1)
               Filter: ((phone_number)::text = '7379976445'::text)
               Rows Removed by Filter: 1022
 Planning Time: 0.636 ms
 Execution Time: 1.177 ms
(10 rows)
```

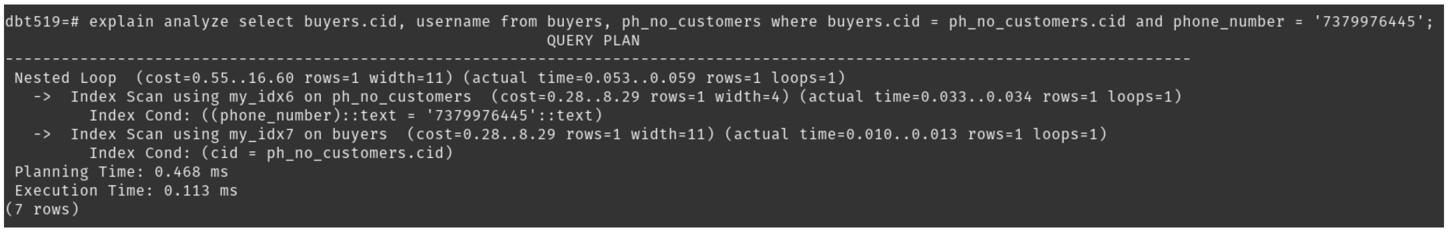
We can see that it does a hash join using sequential scan on the join attribute on both tables, which is a performance bottleneck. (Primary key constraint for cid is removed for both tables for demonstration of secondary indices)

Hence we add secondary indices on join condition and we get 2x improvement in execution speed:



```
dbt519=# explain analyze select buyers.cid, username from ph_no_customers, buyers where phone_number = '7379976445' and buyers.cid = ph_no_customers.cid;
                                                      QUERY PLAN
------------------------------------------------------------------------------------------------------------------------
 Nested Loop  (cost=0.28..27.09 rows=1 width=11) (actual time=0.177..0.432 rows=1 loops=1)
   ->  Seq Scan on ph_no_customers  (cost=0.00..18.79 rows=1 width=4) (actual time=0.151..0.400 rows=1 loops=1)
         Filter: ((phone_number)::text = '7379976445'::text)
         Rows Removed by Filter: 1022
   ->  Index Scan using my_idx7 on buyers  (cost=0.28..8.29 rows=1 width=11) (actual time=0.019..0.023 rows=1 loops=1)
         Index Cond: (cid = ph_no_customers.cid)
 Planning Time: 0.778 ms
 Execution Time: 0.490 ms
(8 rows)
```

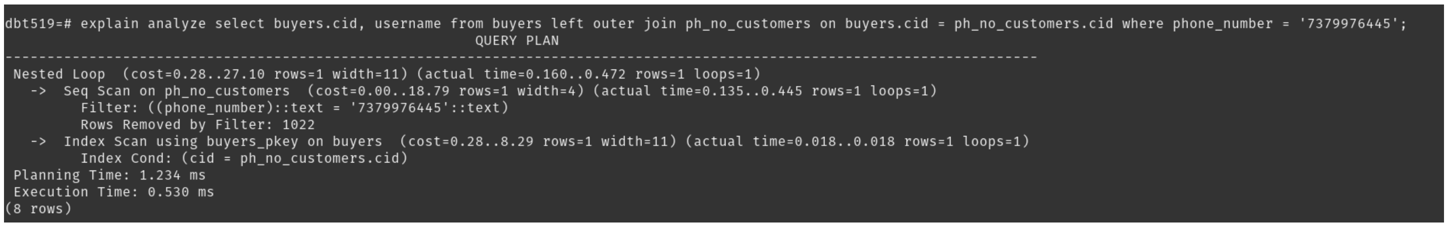
Eventhough index scan is performed on join attribute, there is a sequential scan on phone_number attribute. We can also add a secondary index on it and we see a 10x improvement in execution speed:



```
dbt519=# explain analyze select buyers.cid, username from buyers, ph_no_customers where buyers.cid = ph_no_customers.cid and phone_number = '7379976445';
                                                          QUERY PLAN
--------------------------------------------------------------------------------------------------------------------------------
 Nested Loop  (cost=0.55..16.60 rows=1 width=11) (actual time=0.053..0.059 rows=1 loops=1)
   ->  Index Scan using my_idx6 on ph_no_customers  (cost=0.28..8.29 rows=1 width=4) (actual time=0.033..0.034 rows=1 loops=1)
         Index Cond: ((phone_number)::text = '7379976445'::text)
   ->  Index Scan using my_idx7 on buyers  (cost=0.28..8.29 rows=1 width=11) (actual time=0.010..0.013 rows=1 loops=1)
         Index Cond: (cid = ph_no_customers.cid)
 Planning Time: 0.468 ms
 Execution Time: 0.113 ms
(7 rows)
```

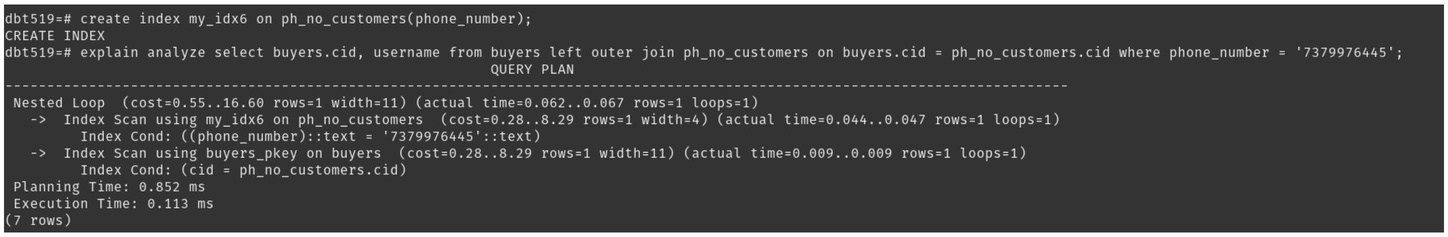
---

4. Queries containing outer joins:



```
dbt519=# explain analyze select buyers.cid, username from buyers left outer join ph_no_customers on buyers.cid = ph_no_customers.cid where phone_number = '7379976445';
                                                        QUERY PLAN
-------------------------------------------------------------------------------------------------------------------------
 Nested Loop  (cost=0.28..27.10 rows=1 width=11) (actual time=0.160..0.472 rows=1 loops=1)
   ->  Seq Scan on ph_no_customers  (cost=0.00..18.79 rows=1 width=4) (actual time=0.135..0.445 rows=1 loops=1)
         Filter: ((phone_number)::text = '7379976445'::text)
         Rows Removed by Filter: 1022
   ->  Index Scan using buyers_pkey on buyers  (cost=0.28..8.29 rows=1 width=11) (actual time=0.018..0.018 rows=1 loops=1)
         Index Cond: (cid = ph_no_customers.cid)
 Planning Time: 1.234 ms
 Execution Time: 0.530 ms
(8 rows)
```

Similar to inner joins, we add secondary indices on join attribute and get 5x improvement in execution time:



```
dbt519=# create index my_idx6 on ph_no_customers(phone_number);
CREATE INDEX
dbt519=# explain analyze select buyers.cid, username from buyers left outer join ph_no_customers on buyers.cid = ph_no_customers.cid where phone_number = '7379976445';
                                                           QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------------
 Nested Loop  (cost=0.55..16.60 rows=1 width=11) (actual time=0.062..0.067 rows=1 loops=1)
   ->  Index Scan using my_idx6 on ph_no_customers  (cost=0.28..8.29 rows=1 width=4) (actual time=0.044..0.047 rows=1 loops=1)
         Index Cond: ((phone_number)::text = '7379976445'::text)
   ->  Index Scan using buyers_pkey on buyers  (cost=0.28..8.29 rows=1 width=11) (actual time=0.009..0.009 rows=1 loops=1)
         Index Cond: (cid = ph_no_customers.cid)
 Planning Time: 0.852 ms
 Execution Time: 0.113 ms
(7 rows)
```
