Database Technologies

UE19CS344

6th Semester, Academic Year 2021-22

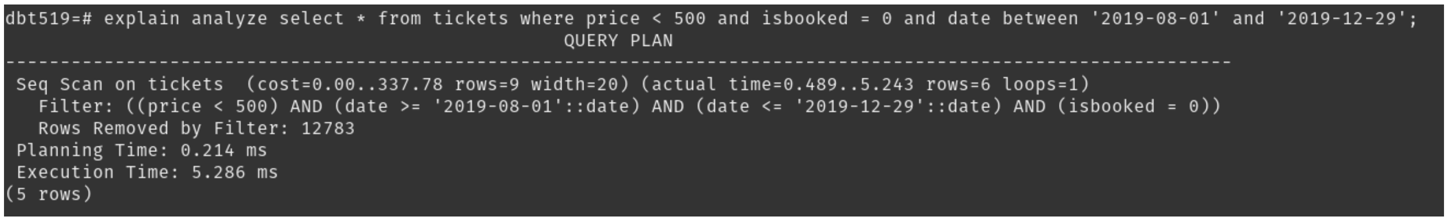
Week #5: Secondary Indexes

(A2)

Date: 15/2/2022

Name : SUMUKH RAJU BHAT	SRN : PES1UG19CS519	Section : H

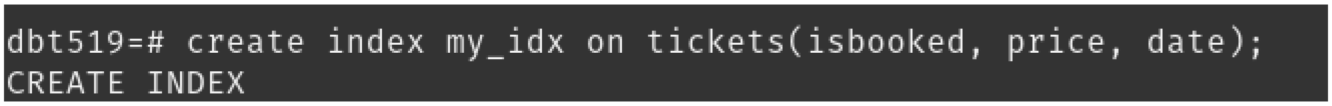
1. Consider the query,



```
dbt519=# explain analyze select * from tickets where price < 500 and isbooked = 0 and date between '2019-08-01' and '2019-12-29';
                                                 QUERY PLAN
------------------------------------------------------------------------------------------------------------------
 Seq Scan on tickets  (cost=0.00..337.78 rows=9 width=20) (actual time=0.489..5.243 rows=6 loops=1)
   Filter: ((price < 500) AND (date >= '2019-08-01'::date) AND (date <= '2019-12-29'::date) AND (isbooked = 0))
   Rows Removed by Filter: 12783
 Planning Time: 0.214 ms
 Execution Time: 5.286 ms
(5 rows)
```

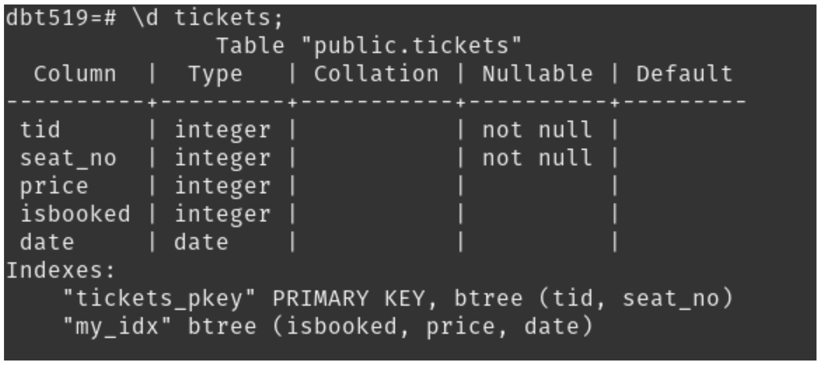
We notice that there is a sequential scan of the table.

If we create a index for the attributes in the where clause using:



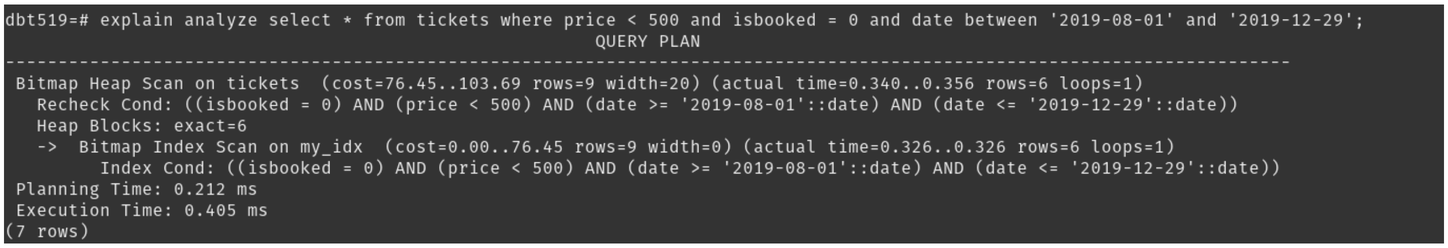
```
dbt519=# create index my_idx on tickets(isbooked, price, date);
CREATE INDEX
```

We can see an index added to the table. Index for primary key already set by the DBMS when it was defined:



```
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_idx" btree (isbooked, price, date)
```

We get the following:

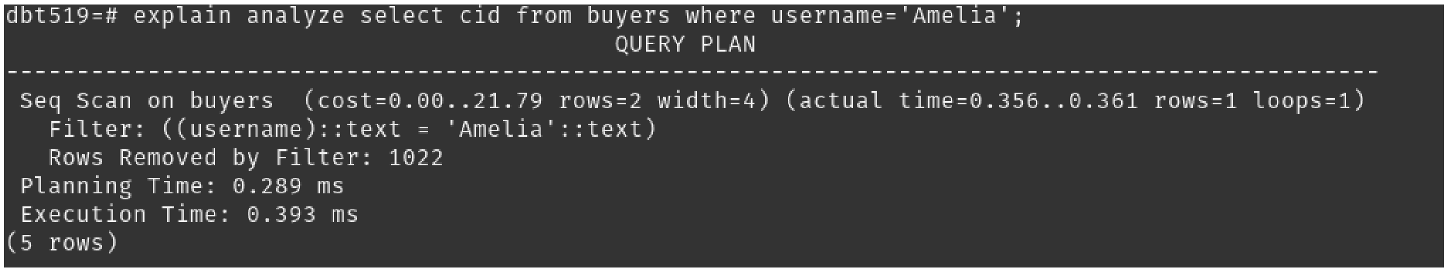


```
dbt519=# explain analyze select * from tickets where price < 500 and isbooked = 0 and date between '2019-08-01' and '2019-12-29';
                                                        QUERY PLAN
------------------------------------------------------------------------------------------------------------------------
 Bitmap Heap Scan on tickets  (cost=76.45..103.69 rows=9 width=20) (actual time=0.340..0.356 rows=6 loops=1)
   Recheck Cond: ((isbooked = 0) AND (price < 500) AND (date >= '2019-08-01'::date) AND (date <= '2019-12-29'::date))
   Heap Blocks: exact=6
   ->  Bitmap Index Scan on my_idx  (cost=0.00..76.45 rows=9 width=0) (actual time=0.326..0.326 rows=6 loops=1)
         Index Cond: ((isbooked = 0) AND (price < 500) AND (date >= '2019-08-01'::date) AND (date <= '2019-12-29'::date))
 Planning Time: 0.212 ms
 Execution Time: 0.405 ms
(7 rows)
```

We can notice now that it does an index scan using the secondary index setup. The execution time drops more than 10x than earlier.

---

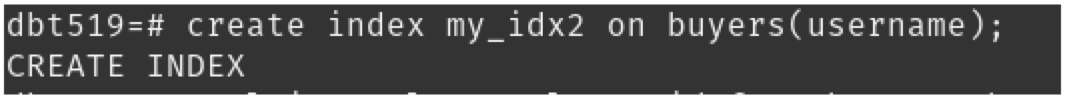
2. Consider another query,



```
dbt519=# explain analyze select cid from buyers where username='Amelia';
                                                QUERY PLAN
---------------------------------------------------------------------------------------------------
 Seq Scan on buyers  (cost=0.00..21.79 rows=2 width=4) (actual time=0.356..0.361 rows=1 loops=1)
   Filter: ((username)::text = 'Amelia'::text)
   Rows Removed by Filter: 1022
 Planning Time: 0.289 ms
 Execution Time: 0.393 ms
(5 rows)
```

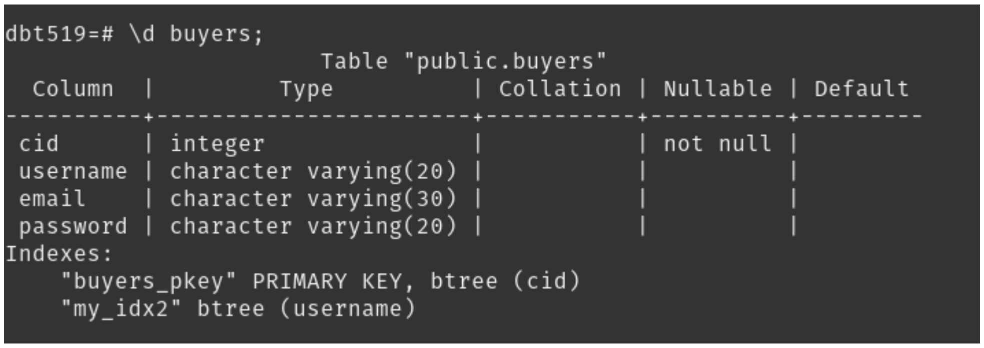
It does a sequential scan to watch the string values in its columns with the given target.

We can setup an index for string attributes as well:

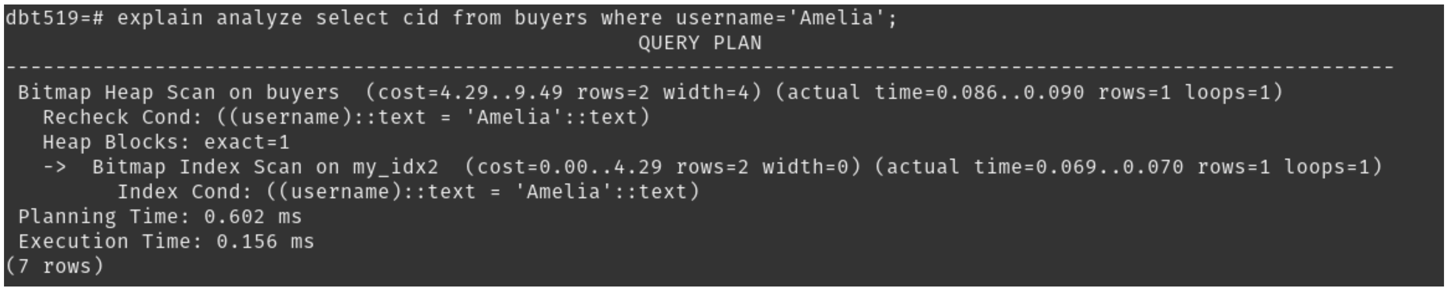


```
dbt519=# create index my_idx2 on buyers(username);
CREATE INDEX
```

We can see the index being added to the table:



```
dbt519=# \d buyers;
                     Table "public.buyers"
  Column  |         Type          | Collation | Nullable | Default
----------+-----------------------+-----------+----------+---------
 cid      | integer               |           | not null |
 username | character varying(20) |           |          |
 email    | character varying(30) |           |          |
 password | character varying(20) |           |          |
Indexes:
    "buyers_pkey" PRIMARY KEY, btree (cid)
    "my_idx2" btree (username)
```

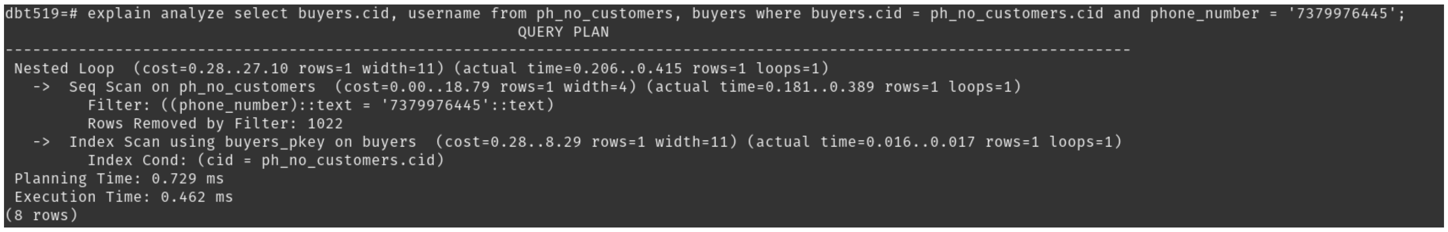


```
dbt519=# explain analyze select cid from buyers where username='Amelia';
                                                    QUERY PLAN
--------------------------------------------------------------------------------------------------------------------
 Bitmap Heap Scan on buyers  (cost=4.29..9.49 rows=2 width=4) (actual time=0.086..0.090 rows=1 loops=1)
   Recheck Cond: ((username)::text = 'Amelia'::text)
   Heap Blocks: exact=1
   ->  Bitmap Index Scan on my_idx2  (cost=0.00..4.29 rows=2 width=0) (actual time=0.069..0.070 rows=1 loops=1)
         Index Cond: ((username)::text = 'Amelia'::text)
 Planning Time: 0.602 ms
 Execution Time: 0.156 ms
(7 rows)
```

Now we can see an index scan being performed. We can see the execution speed drop 2x than before.

---

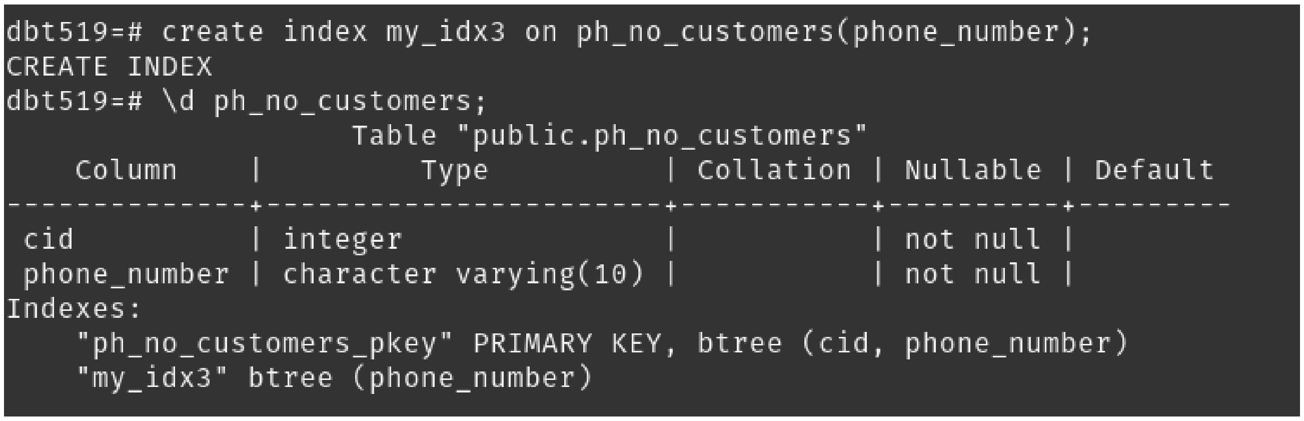
3. Consider another query,



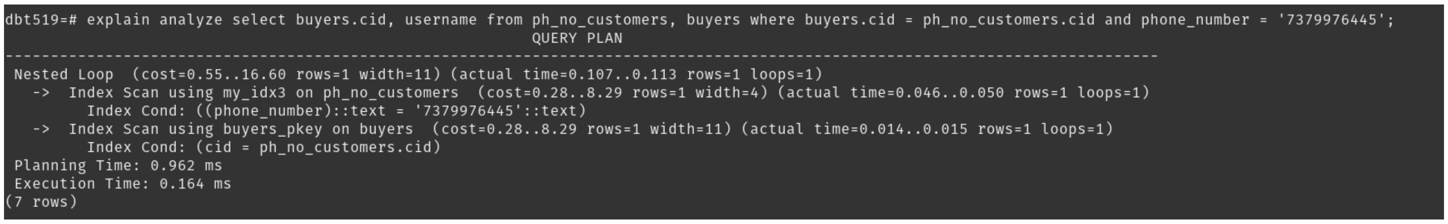
```
dbt519=# explain analyze select buyers.cid, username from ph_no_customers, buyers where buyers.cid = ph_no_customers.cid and phone_number = '7379976445';
                                                       QUERY PLAN
--------------------------------------------------------------------------------------------------------------------------
 Nested Loop  (cost=0.28..27.10 rows=1 width=11) (actual time=0.206..0.415 rows=1 loops=1)
   ->  Seq Scan on ph_no_customers  (cost=0.00..18.79 rows=1 width=4) (actual time=0.181..0.389 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=0.016..0.017 rows=1 loops=1)
         Index Cond: (cid = ph_no_customers.cid)
 Planning Time: 0.729 ms
 Execution Time: 0.462 ms
(8 rows)
```

Since the join condition attribute happens to be a primary key, it does a index scan for that, but a sequential scan for other attributes in the where condition of the query.

Therefore we setup the following secondary indexes:



```
dbt519=# create index my_idx3 on ph_no_customers(phone_number);
CREATE INDEX
dbt519=# \d ph_no_customers;
                    Table "public.ph_no_customers"
    Column    |         Type          | Collation | Nullable | Default
--------------+-----------------------+-----------+----------+---------
 cid          | integer               |           | not null |
 phone_number | character varying(10) |           | not null |
Indexes:
    "ph_no_customers_pkey" PRIMARY KEY, btree (cid, phone_number)
    "my_idx3" btree (phone_number)
```



```
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
-----------------------------------------------------------------------------------------------------------------------------
 Nested Loop  (cost=0.55..16.60 rows=1 width=11) (actual time=0.107..0.113 rows=1 loops=1)
   ->  Index Scan using my_idx3 on ph_no_customers  (cost=0.28..8.29 rows=1 width=4) (actual time=0.046..0.050 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=0.014..0.015 rows=1 loops=1)
         Index Cond: (cid = ph_no_customers.cid)
 Planning Time: 0.962 ms
 Execution Time: 0.164 ms
(7 rows)
```

Now it does index scans for all attributes. Execution time drops 2x than earlier.