

CS349

Lab7 Indexing and Query Plans

Saksham Rathi

22B1003

28th March 2025

1

The relation was created as follows: “CREATE TABLE player(A INT, B INT);”.

- (a) Here is how when I run explain analyze on the insertion query:

```
QUERY PLAN

-----
Insert on player  (cost=2.97..3.59 rows=0 width=0) (actual time=19.140..19.143
rows=0 loops=1)
  CTE generateplayers
    -> Recursive Union  (cost=0.00..2.97 rows=31 width=8) (actual time=0.011..
8.463 rows=10000 loops=1)
      -> Result  (cost=0.00..0.01 rows=1 width=8) (actual time=0.003..0.00
4 rows=1 loops=1)
      -> WorkTable Scan on generateplayers generateplayers_1  (cost=0.00..
0.27 rows=3 width=8) (actual time=0.000..0.000 rows=1 loops=10000)
        Filter: ((1 + a) < 10001)
        Rows Removed by Filter: 0
      -> CTE Scan on generateplayers  (cost=0.00..0.62 rows=31 width=8) (actual ti
me=0.014..10.814 rows=10000 loops=1)
  Planning Time: 0.263 ms
  Execution Time: 20.710 ms
(10 rows)
```

Execution Time: **20.710 ms**

- (b) Here is how to retrieve the record with A=1000: “EXPLAIN ANALYZE SELECT * FROM player WHERE A = 1000;”.

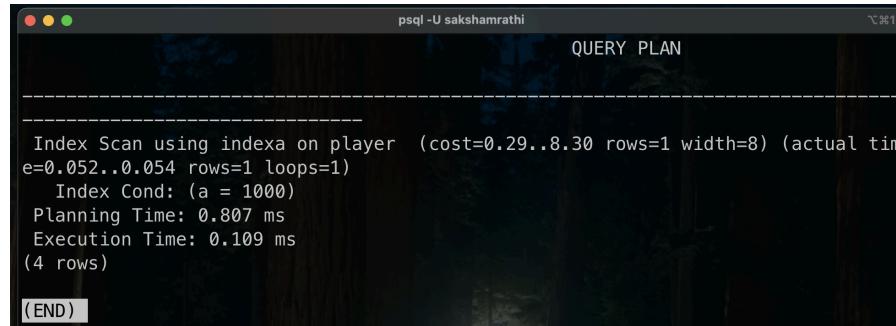
```
QUERY PLAN

-----
Seq Scan on player  (cost=0.00..170.00 rows=1 width=8) (actual time=0.099..0.74
5 rows=1 loops=1)
  Filter: (a = 1000)
  Rows Removed by Filter: 9999
  Planning Time: 0.695 ms
  Execution Time: 0.772 ms
(5 rows)
```

The plan shows that a **sequential scan** has been performed. The planning time was **0.695 ms** and the execution time was **0.772 ms**. The table is relatively small and there is no index on attribute A, that's why psql has chosen sequential scan for this query.

(c) The query used to create the index was: “CREATE INDEX indexA ON player(A);”.

(d) The query used was: “EXPLAIN ANALYZE SELECT * FROM player WHERE A = 1000;”.



```
psql -U sakshamrathi
QUERY PLAN

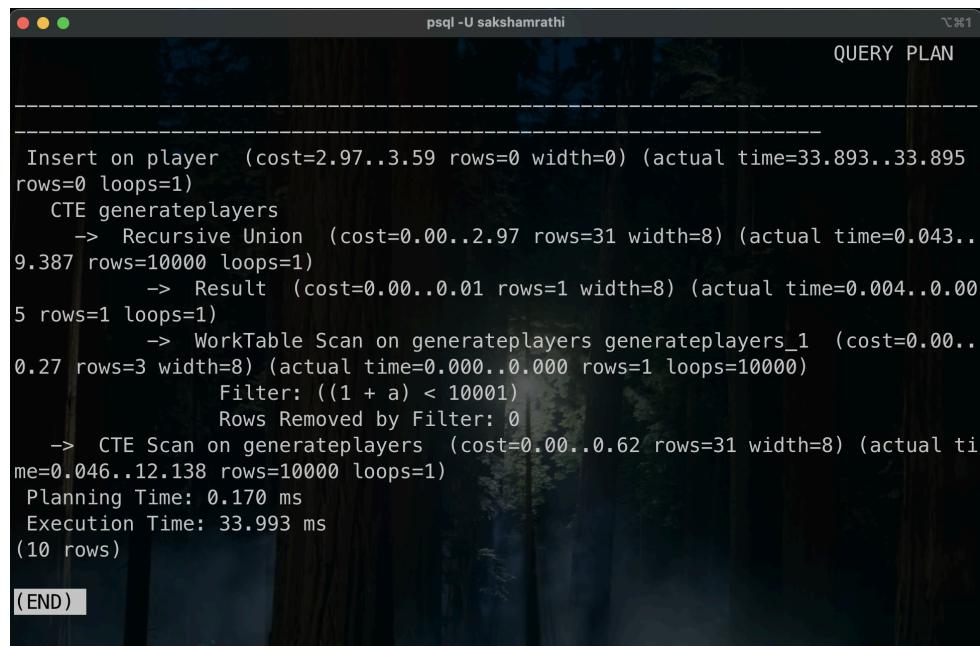
Index Scan using indexA on player  (cost=0.29..8.30 rows=1 width=8) (actual time=0.052..0.054 rows=1 loops=1)
  Index Cond: (a = 1000)
Planning Time: 0.807 ms
Execution Time: 0.109 ms
(4 rows)

(END)
```

The planning time is **0.807 ms** and the execution time is **0.109 ms**. The query plan shows that the **index** which was created in the previous part, has been used to answer the query. This is evident from the lower execution time as compared to the previous execution. However, it is worth noting that the planning time has increased.

(e) The query used was: “DELETE FROM player;”.

(f) The query was the same as part (a). The query plan:



```
psql -U sakshamrathi
QUERY PLAN

Insert on player  (cost=2.97..3.59 rows=0 width=0) (actual time=33.893..33.895
rows=0 loops=1)
  CTE generateplayers
    -> Recursive Union  (cost=0.00..2.97 rows=31 width=8) (actual time=0.043..9.387
rows=10000 loops=1)
      -> Result  (cost=0.00..0.01 rows=1 width=8) (actual time=0.004..0.005
rows=1 loops=1)
        -> WorkTable Scan on generateplayers generateplayers_1  (cost=0.00..0.27
rows=3 width=8) (actual time=0.000..0.000 rows=1 loops=10000)
          Filter: ((1 + a) < 10001)
          Rows Removed by Filter: 0
      -> CTE Scan on generateplayers  (cost=0.00..0.62 rows=31 width=8) (actual time=0.046..12.138
rows=10000 loops=1)
Planning Time: 0.170 ms
Execution Time: 33.993 ms
(10 rows)

(END)
```

When we had deleted the table in one of the previous parts, the index didn't get deleted (because the relation remained, and only the tuples got deleted). So, in each insertion, which we perform in this step, leads to an update to the index too. This causes a higher execution time as compared to when we had run the query previously.

2

- (a) The query used was: “EXPLAIN SELECT * FROM student WHERE tot_cred = 100;”. Here is the output:

```
sakshamrathi=> select * from student;
sakshamrathi=> EXPLAIN SELECT * FROM student WHERE tot_cred = 100;
                  QUERY PLAN
-----
Seq Scan on student  (cost=0.00..40.00 rows=19 width=24)
    Filter: (tot_cred = '100'::numeric)
(2 rows)
```

As we can see from the query plan, a sequential file scan has been performed to get the student whose tot_cred is equal to 100.

- (b) We need to get index scan, so one of our attribute conditions should be on the primary key of the relation: “EXPLAIN SELECT * FROM student where dept_name = 'Math' AND id = '557';”.

```
sakshamrathi=> EXPLAIN SELECT * FROM student where dept_name = 'Math' AND id = '557';
                  QUERY PLAN
-----
Index Scan using student_pkey on student  (cost=0.28..8.30 rows=1 width=24)
    Index Cond: ((id)::text = '557'::text)
    Filter: ((dept_name)::text = 'Math'::text)
(3 rows)
```

As we can see from the query plan, an index scan was performed.

(c) Here is the query used: “EXPLAIN SELECT * FROM takes where id='636' AND grade='F';”.

```
sakshamrathi=> EXPLAIN SELECT * FROM takes where id='636' AND grade='F';
                                         QUERY PLAN
-----
Bitmap Heap Scan on takes  (cost=4.40..52.88 rows=1 width=24)
  Recheck Cond: ((id)::text = '636'::text)
  Filter: ((grade)::text = 'F'::text)
    -> Bitmap Index Scan on takes_pkey  (cost=0.00..4.40 rows=15 width=0)
      Index Cond: ((id)::text = '636'::text)
(5 rows)

sakshamrathi=>
```

As we can see, a bitmap index scan was performed on the takes relation (id is one of the attributes present in the primary key set of takes, whereas grade is not a primary key). It also minimizes the number of records fetched from the disk, thus reducing the number of random I/O operations.

(d) Here is the query which was used: “EXPLAIN SELECT s.name, t.course_id FROM student s JOIN takes t ON s.id = t.id where grade='F';”.

```
psql -U sakshamrathi
                                         QUERY PLAN
-----
-----
Nested Loop  (cost=0.28..603.30 rows=1 width=10)
  -> Seq Scan on takes t  (cost=0.00..595.00 rows=1 width=9)
      Filter: ((grade)::text = 'F'::text)
  -> Index Scan using student_pkey on student s  (cost=0.28..8.29 rows=1 width=11)
      Index Cond: ((id)::text = (t.id)::text)
(5 rows)

(END)
```

For the outer child, a seq scan was performed (for relation takes), whereas for the inner child, an index scan was performed (for relation student).

(e) Here is how the relation was constructed: “create table takes2 (id varchar(5), course_id varchar(8), sec_id varchar(8), semester varchar(6), year numeric(4,0), grade varchar(2));”

Here is how the other query was run: “explain analyse insert into takes2 select * from takes”.

```
psql -U sakshamrathi
                                         QUERY PLAN
-----
Insert on takes2  (cost=0.00..520.00 rows=0 width=0) (actual time=29.794..29.79
6 rows=0 loops=1)
    -> Seq Scan on takes  (cost=0.00..520.00 rows=30000 width=24) (actual time=0
.031..4.717 rows=30000 loops=1)
Planning Time: 0.155 ms
Execution Time: 29.838 ms
(4 rows)

(END)
```

The plan chosen is seq scan. The planning time is 0.155 ms, and the execution time is 29.838 ms.

(f) The table was dropped: “drop table takes2;”. Here is how the table was created: “create table takes2 (id varchar(5), course_id varchar(8), sec_id varchar(8), semester varchar(6), year numeric(4,0), grade varchar(2), primary key (ID, course_id, sec_id, semester, year));”. Here is how the tuples were inserted: “EXPLAIN ANALYSE insert into takes2 select * from takes;”.

```
psql -U sakshamrathi
                                         QUERY PLAN
-----
```

Search

```
Insert on takes2  (cost=0.00..520.00 rows=0 width=0) (actual time=112.327..112.
328 rows=0 loops=1) > lab7 > DDL.sql
    -> Seq Scan on takes  (cost=0.00..520.00 rows=30000 width=24) (actual time=0
.016..2.266 rows=30000 loops=1)
Planning Time: 0.153 ms
Execution Time: 112.360 ms
(4 rows)
```

The planning time is 0.153 ms, and the execution time is 112.360 ms. So, the planning time roughly remains the same as before, however the execution time increases drastically. There are multiple reasons for this:

- Primary key has default constraints such as that of uniqueness and not being null. Each time a tuple is being inserted, these constraints are checked for, and thus it takes more time.\
- Psql creates indices on the primary key attributes, this index creation time is also included in this execution time, increasing it even further.

(g) The query which was run was: “EXPLAIN select count(*) from course c where exists (select * from takes t where t.course_id < c.course_id)”. The plan chosen is:

The screenshot shows a terminal window titled "psql -U sakshamrathi". The title bar also includes "QUERY PLAN". The main content is a query plan for an aggregate function. The plan starts with an "Aggregate" node with a cost of 67.95..67.96, followed by a "Nested Loop Semi Join" with a cost of 0.29..67.79. This join consists of a "Seq Scan on course c" with a cost of 0.00..4.00 and an "Index Only Scan using takes_pkey on takes t" with a cost of 0.29..328.93. The join condition is "(course_id < (c.course_id)::text)". The total cost for the plan is 67.95..67.96. The output indicates there are 5 rows. At the bottom of the terminal window, there is a button labeled "(END)".

```
psql -U sakshamrathi
QUERY PLAN

Aggregate  (cost=67.95..67.96 rows=1 width=8)
  -> Nested Loop Semi Join  (cost=0.29..67.79 rows=67 width=0)
      -> Seq Scan on course c  (cost=0.00..4.00 rows=200 width=4)
      -> Index Only Scan using takes_pkey on takes t  (cost=0.29..328.93 rows=10000 width=4)
          Index Cond: (course_id < (c.course_id)::text)
(5 rows)

(END)
```

The plan chosen by this query was nested loop semi join, with seq scan on course c and index only scan on takes t. Firstly, we want to convert this query to some join operation, so that we can perform optimizations over it. Also, we need to preserve duplicates, so psql uses semi-join operation. (Hash and merge joins are costly). The course size is small, so a sequential scan was performed, index only scan avoids unnecessary table lookups.

(h) The query which was run was: “EXPLAIN select count(*) from course c where c.course_id in (select course_id from takes t)”. The plan chosen is:

```
psql -U sakshamrathi
QUERY PLAN

-----
Aggregate  (cost=250.56..250.57 rows=1 width=8)
  -> Nested Loop Semi Join  (cost=0.29..250.34 rows=85 width=0)
      -> Seq Scan on course c  (cost=0.00..4.00 rows=200 width=4)
      -> Index Only Scan using takes_pkey on takes t  (cost=0.29..232.46 rows=353 width=4)
          Index Cond: (course_id = (c.course_id)::text)
(5 rows)

(END)
```

The plan chosen by this query was nested loop semi join, with seq scan on course c and index only scan on takes t.

The query of this part is similar to that of the previous part. So, this query is converted to a relational expression similar as the previous query, and thus the plan remains the same.

(i) The query which was run was: “EXPLAIN select count(*) from course c where c.course_id not in (select course_id from takes t)”. The plan chosen is:

```
psql -U sakshamrathi
QUERY PLAN

-----
Aggregate  (cost=599.75..599.76 rows=1 width=8)
  -> Seq Scan on course c  (cost=595.00..599.50 rows=100 width=0)
      Filter: (NOT (ANY ((course_id)::text = ((hashed SubPlan 1).col1)::text)))
)
SubPlan 1
  -> Seq Scan on takes t  (cost=0.00..520.00 rows=30000 width=4)
(5 rows)

(END)
```

The plan chosen by SQL is seq scan on course c (with a filter), and another subplan1 which has seq scan on takes t.

For not in, we need to search over all the records (as it does not allow early termination). An index scan would require more lookups, which will be slower than hashing. A hashed subquery still requires a full scan.