CS349

Lab7 Indexing and Query Plans

Saksham Rathi

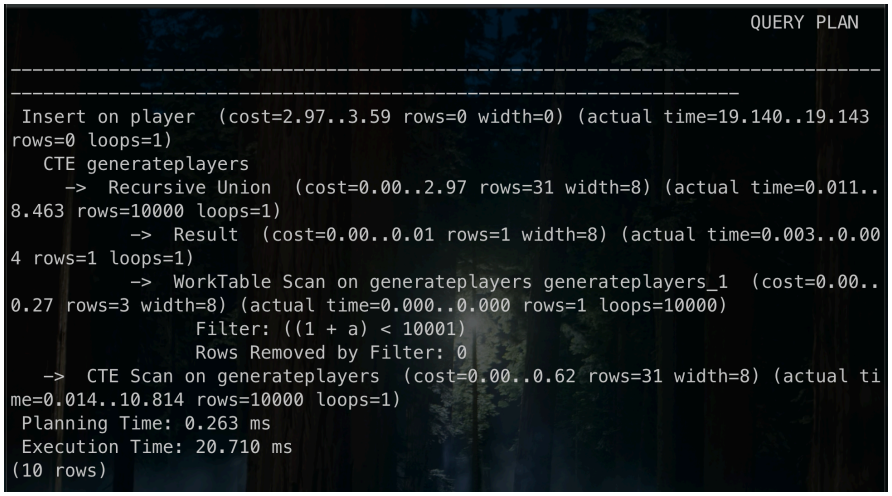
22B1003

28th March 2025

1

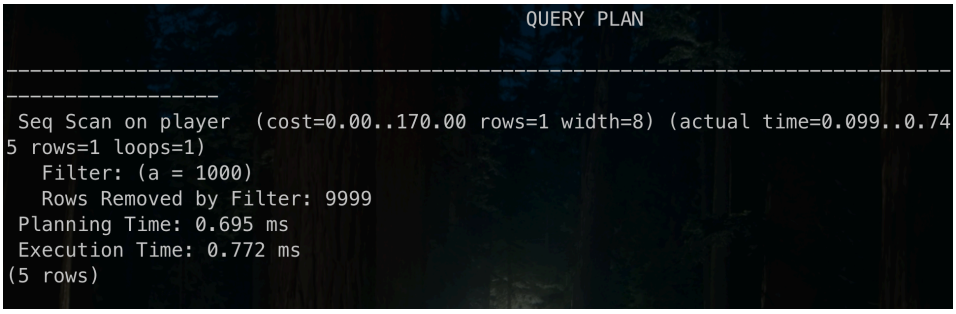
The relation was created as follows: “CREATE TABLE player(A INT, B INT);”.

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



Execution Time: **20.710 ms**

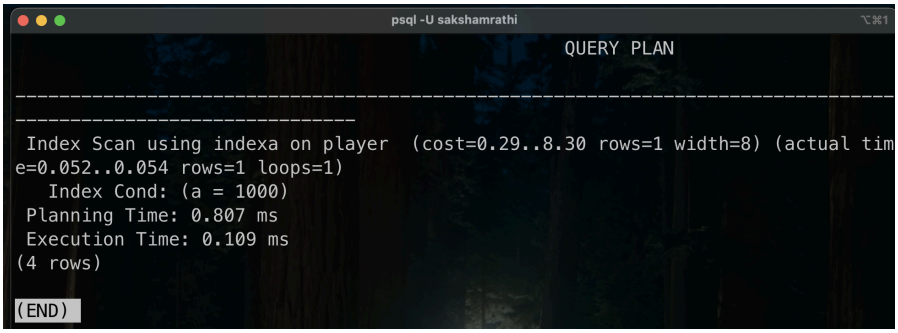
(b) Here is how to retrieve the record with A=1000: “EXPLAIN ANALYZE SELECT * FROM player WHERE A = 1000;”.



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);".

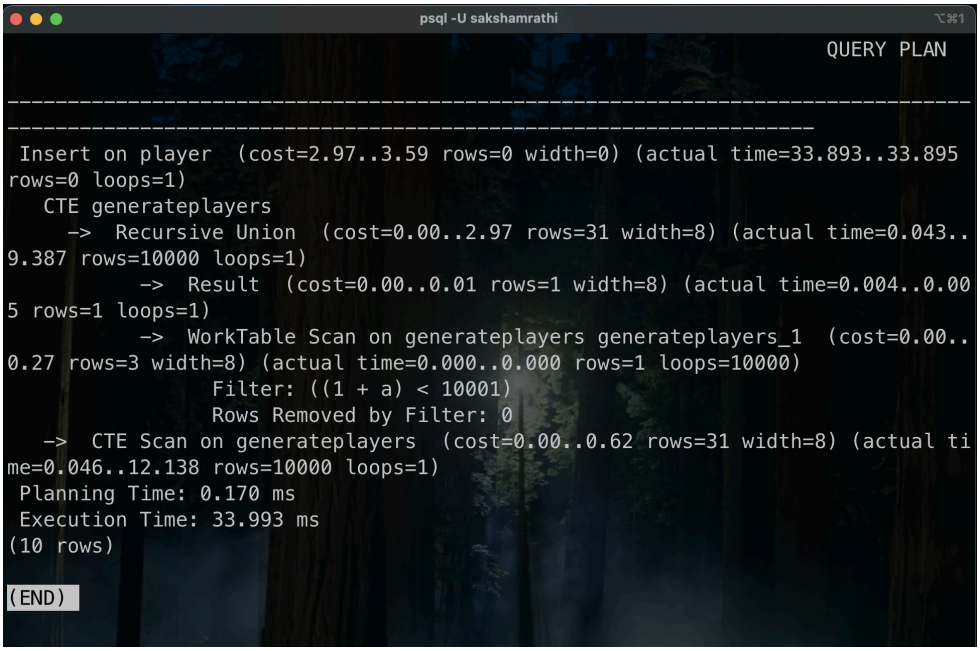
(d) The query used was: "EXPLAIN ANALYZE SELECT * FROM player WHERE A = 1000;".



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:



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)

sakshamrathi=>
```

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.

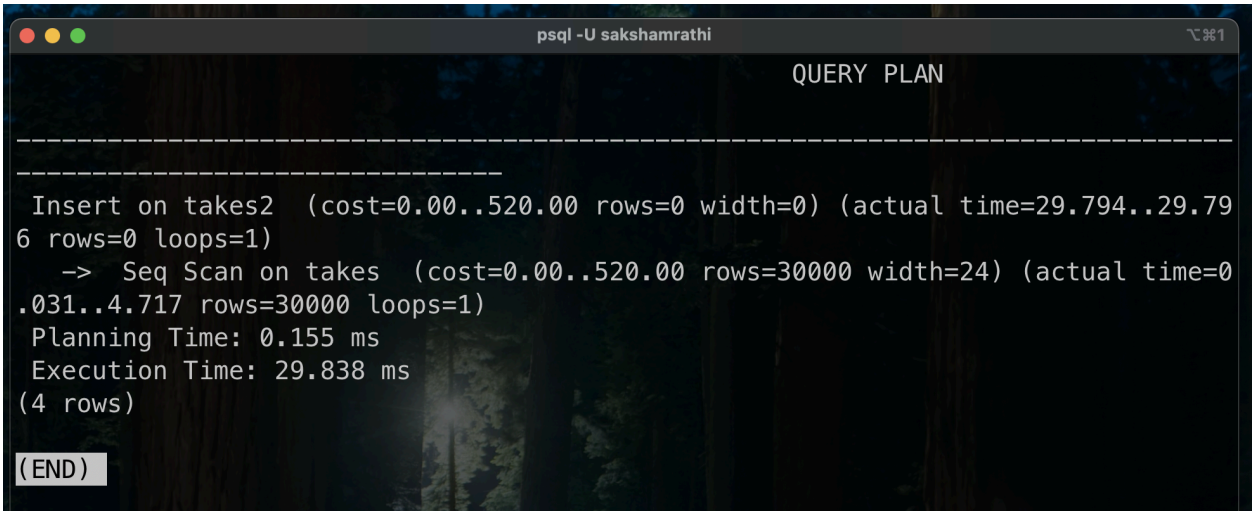
(d) Here is the query which was used: “EXPLAIN SELECT s.name, t.course_id FROM student s JOIN takes t ON s.id = t.id where grade='F';”.

```
psql -U sakshamrathi                                                ⌥⌘1
                                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).

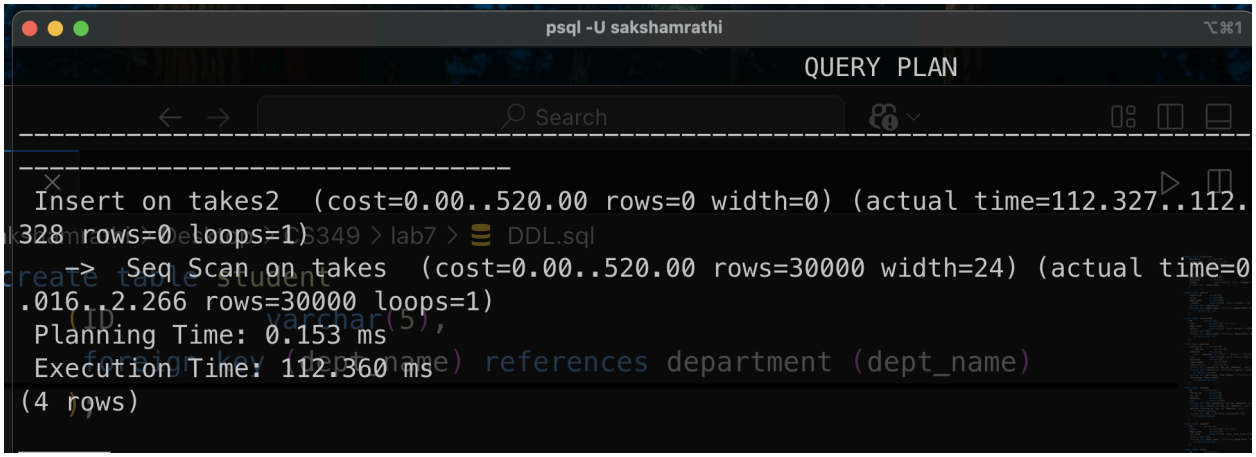
(e) Here is how the relation was constructed: “create table takes2 (id varchar(5), course_id varchar(8), sec_id varchar(8), semester varchar(6), year numeric(4,0), grade varchar(2));”

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



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

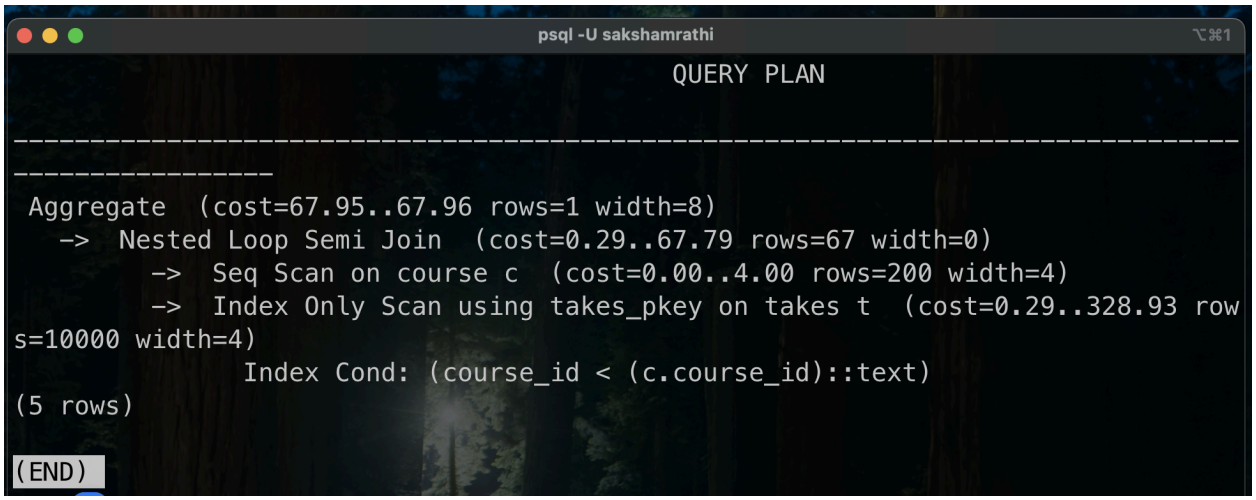
(f) The table was dropped: “drop table takes2;”. Here is how the table was created: “create table takes2 (id varchar(5), course_id varchar(8), sec_id varchar(8), semester varchar(6), year numeric(4,0), grade varchar(2), primary key (ID, course_id, sec_id, semester, year));”. Here is how the tuples were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



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:

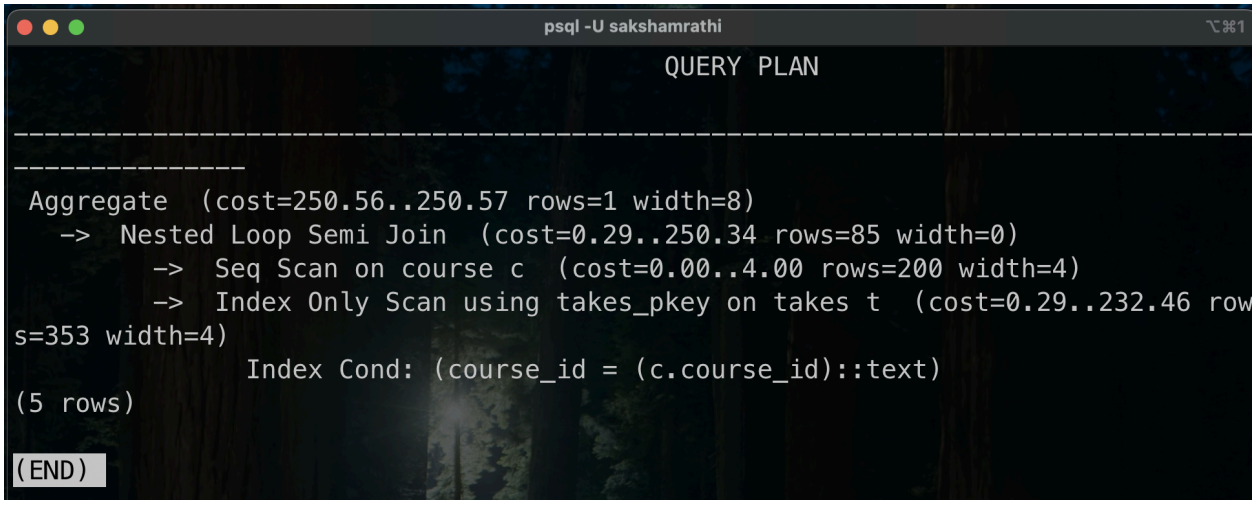
- Primary key has default constraints such as that of uniqueness and not being null. Each time a tuple is being inserted, these constraints are checked for, and thus it takes more time.\
- Psql creates indices on the primary key attributes, this index creation time is also included in this execution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

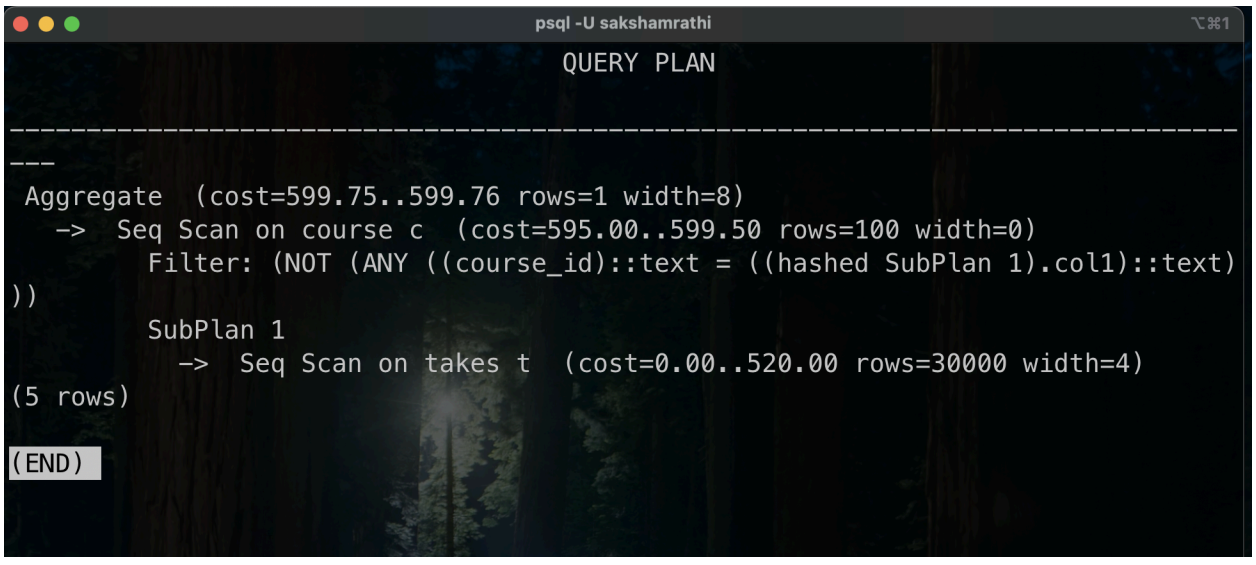
(h) The query which was run was: “EXPLAIN s*elect count(*) from course c where c.course_id in (select course_id from takes t)”.* The plan chosen is:



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 s*elect count(*) from course c where c.course_id not in (select course_id from takes t)”.* The plan chosen is:



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.