Query plans in PostgreSQL

- - 140050002, 140050007

Q.1

explain select * from takes where id = '1111';

```
postgres=# explain select * from takes where id = '1111';
                                QUERY PLAN
---------------------------------------------------------------------------
 Bitmap Heap Scan on takes  (cost=4.40..52.84 rows=15 width=24)
   Recheck Cond: ((id)::text = '1111'::text)
   ->  Bitmap Index Scan on takes_pkey  (cost=0.00..4.40 rows=15 width=0)
         Index Cond: ((id)::text = '1111'::text)
(4 rows)
```

Q.2

explain select * from student natural join takes where (name<'deep') AND (course_id='CS-101');

```
postgres=# explain select * from student natural join takes where (name<'deep') AND (course_id='CS-101');
                                QUERY PLAN
---------------------------------------------------------------------------
 Nested Loop  (cost=0.28..603.31 rows=1 width=43)
   ->  Seq Scan on takes  (cost=0.00..595.00 rows=1 width=24)
         Filter: ((course_id)::text = 'CS-101'::text)
   ->  Index Scan using student_pkey on student  (cost=0.28..8.30 rows=1 width=24)
         Index Cond: ((id)::text = (takes.id)::text)
         Filter: ((name)::text < 'deep'::text)
(6 rows)
```

Q.3

explain select * from student,takes where student.dept_name=takes.course_id order by dept_name;

```
postgres=# explain select * from student,takes where student.dept_name=takes.course_id order by dept_name;
                              QUERY PLAN
-----------------------------------------------------------------------
 Merge Join  (cost=3039.48..3049.49 rows=1 width=48)
   Merge Cond: ((student.dept_name)::text = (takes.course_id)::text)
   ->  Sort  (cost=144.66..149.66 rows=2000 width=24)
         Sort Key: student.dept_name
         ->  Seq Scan on student  (cost=0.00..35.00 rows=2000 width=24)
   ->  Sort  (cost=2745.13..2819.97 rows=29937 width=24)
         Sort Key: takes.course_id
         ->  Seq Scan on takes  (cost=0.00..519.37 rows=29937 width=24)
(8 rows)
```

Q.4

explain select * from student natural join department order by id limit 10;

```
postgres=#  explain select * from student natural join department order by id;
                                  QUERY PLAN
-------------------------------------------------------------------------------
 Sort  (cost=194.31..199.31 rows=2000 width=88)
   Sort Key: student.id
   ->  Hash Join  (cost=22.15..84.65 rows=2000 width=88)
         Hash Cond: ((student.dept_name)::text = (department.dept_name)::text)
         ->  Seq Scan on student  (cost=0.00..35.00 rows=2000 width=24)
         ->  Hash  (cost=15.40..15.40 rows=540 width=122)
               ->  Seq Scan on department  (cost=0.00..15.40 rows=540 width=122)
(7 rows)

postgres=#  explain select * from student natural join department order by id limit 10;
                                       QUERY PLAN
-----------------------------------------------------------------------------------------
 Limit  (cost=0.43..3.09 rows=10 width=88)
   ->  Nested Loop  (cost=0.43..533.27 rows=2000 width=88)
         ->  Index Scan using student_pkey on student  (cost=0.28..130.27 rows=2000 width=24)
         ->  Index Scan using department_pkey on department  (cost=0.15..0.19 rows=1 width=122)
               Index Cond: ((dept_name)::text = (student.dept_name)::text)
(5 rows)
```

Explanation :

For **not using limit**, the planner has chosen to use a hash join, in which rows of *'student'* are entered into an in-memory hash table, after which *'department'* is scanned and the hash table is probed for matches to each row. The sequential scan on department is the input to the Hash node, which constructs the hash table. Finally we sort the whole result based on the key of student id.

For **using limit,** we have a nested-loop join node with two table scans as inputs. The join's "outer" child is an Index scan. The nested-loop join node will run its "inner" child once for each row obtained from the outer child. The inner child is an Index scan which has the conditions for the natural join in it.

Cause of Change of planning :

Since after adding limit of 10 we have to get the top 10 id, we do not have to sort the whole result as nested looping is a less costly way.

Q.5

explain select * from student where exists (select * from takes where takes.id = student.id AND takes.grade < 'B');

```
postgres=# explain select * from student where exists (select * from takes where takes.id = student.id AND takes.grade < 'B');
                              QUERY PLAN
-----------------------------------------------------------------------
 Hash Join  (cost=634.63..683.74 rows=2000 width=24)
   Hash Cond: ((student.id)::text = (takes.id)::text)
   ->  Seq Scan on student  (cost=0.00..35.00 rows=2000 width=24)
   ->  Hash  (cost=626.37..626.37 rows=661 width=5)
         ->  HashAggregate  (cost=619.76..626.37 rows=661 width=5)
               Group Key: (takes.id)::text
               ->  Seq Scan on takes  (cost=0.00..595.00 rows=9904 width=5)
                     Filter: ((grade)::text < 'B'::text)
(8 rows)
```

Explanation :

Here the planner has chosen to use a hash join, in which rows of *'student'* are entered into an in-memory hash table, after which *'takes'* is scanned and the hash table is probed for matches to each row. The sequential scan on student is the input to the Hash node, which constructs the hash table. The Hash uses the HashAggregate to group the ids of takes and student and there is a Sequential scan over takes to get the condition applied on grade.

Q.6

explain select * from student where not exists (select * from takes where takes.id = student.id AND takes.grade < 'B');

```
postgres=# explain select * from student where not exists (select * from takes where takes.id = student.id AND takes.grade < 'B');
                             QUERY PLAN
--------------------------------------------------------------------
 Hash Anti Join  (cost=718.80..781.30 rows=1 width=24)
   Hash Cond: ((student.id)::text = (takes.id)::text)
   ->  Seq Scan on student  (cost=0.00..35.00 rows=2000 width=24)
   ->  Hash  (cost=595.00..595.00 rows=9904 width=5)
         ->  Seq Scan on takes  (cost=0.00..595.00 rows=9904 width=5)
               Filter: ((grade)::text < 'B'::text)
(6 rows)
```

Explanation :

Here the planner has chosen to use a hash Anti Join. Anti-join is preferred as we don't really need to join; we only check if a join would NOT yield results for any given tuple. In Hash Anti Join rows of *'student'* are entered into an in-memory hash table, after which *'department'* is scanned and the hash table is probed for matches to each row. The sequential scan on department is the input to the Hash node, which constructs the hash table.