2022-Fall
Database Systems

학번	2015150220
이름	이윤수

Lab - Query Processing

[Exercise 1]

- Consider two join cases below:

a. Equi-join two tables of “supplier” and “nation” with “s_nationkey” and “n_nationkey” as join keys.

1) Estimation

: 두 테이블 전체를 완전 탐색하면서 Equi-join 을 수행하는 것이기 때문에 Nested Loop Join, Merge Join, Hash Join 중에서 Hash-join 이 가장 빠를 것이다.

2) Verification

```
postgres=# EXPLAIN ANALYZE SELECT * FROM supplier join nation on s_nationkey = n_nationkey;
                                                          QUERY PLAN
-------------------------------------------------------------------------------------------------------------------------------
 Hash Join  (cost=442.00..1024.40 rows=27000 width=261) (actual time=7.204..10.596 rows=9580 loops=1)
   Hash Cond: (nation.n_nationkey = supplier.s_nationkey)
   ->  Seq Scan on nation  (cost=0.00..15.40 rows=540 width=122) (actual time=0.030..0.036 rows=26 loops=1)
   ->  Hash  (cost=317.00..317.00 rows=10000 width=139) (actual time=7.107..7.108 rows=10000 loops=1)
         Buckets: 16384  Batches: 1  Memory Usage: 1848kB
         ->  Seq Scan on supplier  (cost=0.00..317.00 rows=10000 width=139) (actual time=0.008..2.029 rows=10000 loops=1)
 Planning Time: 0.278 ms
 Execution Time: 12.173 ms
(8 rows)
```

실행결과 실제로 Hash Join 이 이루어짐을 확인할 수 있다. 구체적으로는 supplier 테이블을 Seq scan 하면서 해시 함수를 사용해 Bucket 을 만든 다음, nation 테이블에 대해 seq scan 을 하면서 역시나 해시 함수를 적용한다. 이때 같은 해시 값으로 매치되는 supplier 테이블의 레코드가 있는지 probe 가 이루어진다 (Hash Cond 부분).

b. Attach the above SQL statement with “ORDER BY s_nationkey”, and test again.

1) Estimation

두 테이블을 equi-join 한 결과를 sorting 해서 나타내야 하므로, 이러한 경우에는 애초에 join 하는 과정에서 각 테이블에 대한 sorting 을 먼저 실시하고 join 하는 Merge join 이, join 한 결과를 sorting 하는 다른 알고리즘에 비해, 더 빠를 것이다.

2) Verification

```
postgres=# EXPLAIN ANALYZE SELECT * FROM supplier join nation on s_nationkey = n_nationkey ORDER BY s_nationkey;
                                                                 QUERY PLAN
--------------------------------------------------------------------------------------------------------------------------------------------
 Merge Join  (cost=1021.29..1428.99 rows=27000 width=261) (actual time=9.572..20.470 rows=9580 loops=1)
   Merge Cond: (nation.n_nationkey = supplier.s_nationkey)
   ->  Sort  (cost=39.91..41.26 rows=540 width=122) (actual time=0.067..0.082 rows=25 loops=1)
         Sort Key: nation.n_nationkey
         Sort Method: quicksort  Memory: 26kB
         ->  Seq Scan on nation  (cost=0.00..15.40 rows=540 width=122) (actual time=0.020..0.026 rows=26 loops=1)
   ->  Sort  (cost=981.39..1006.39 rows=10000 width=139) (actual time=8.452..11.941 rows=10000 loops=1)
         Sort Key: supplier.s_nationkey
         Sort Method: quicksort  Memory: 2873kB
         ->  Seq Scan on supplier  (cost=0.00..317.00 rows=10000 width=139) (actual time=0.007..1.991 rows=10000 loops=1)
 Planning Time: 0.262 ms
 Execution Time: 24.857 ms
(12 rows)
```

각 테이블에 대해 join attribute 로 sort 를 실행한 다음 Merge 가 이루어진다. Sorting 방식으로는 quicksort 가 사용됐다.

[Exercise 2]

Consider two self-join cases below:

a. Equi-join of the table “nation” with “n_nationkey” as a join key

1) Estimation

Equi-join 이므로 Hash Join 이 사용 가능하다. 일부 값만을 찾는 것이 아니라 테이블 전체를 탐색하는 것이므로 다른 join 알고리즘 보다는 hash join 이 더 빠를 것이다.

2) Verification

```
postgres=# EXPLAIN ANALYZE SELECT * FROM nation as A join nation as B on A.n_nationkey = B.n_nationkey;
                                                        QUERY PLAN
---------------------------------------------------------------------------------------------------------------------------
 Hash Join  (cost=22.15..89.93 rows=1458 width=244) (actual time=0.097..0.128 rows=26 loops=1)
   Hash Cond: (a.n_nationkey = b.n_nationkey)
   ->  Seq Scan on nation a  (cost=0.00..15.40 rows=540 width=122) (actual time=0.020..0.027 rows=26 loops=1)
   ->  Hash  (cost=15.40..15.40 rows=540 width=122) (actual time=0.030..0.031 rows=26 loops=1)
         Buckets: 1024  Batches: 1  Memory Usage: 10kB
         ->  Seq Scan on nation b  (cost=0.00..15.40 rows=540 width=122) (actual time=0.006..0.013 rows=26 loops=1)
 Planning Time: 0.217 ms
 Execution Time: 0.211 ms
(8 rows)
```

실제로 Hash Join 이 실행됐다. 특히 앞서 서로 다른 두 테이블에 대해 Hash Join 을 한 것과 비교했을 때,Self-jion 의 경우가 훨씬 더 빠른 것을 확인할 수 있다. supplier 테이블보다 nation 테이블의 크기가 더 작기 때문에 이러한 차이가 발생했다고 생각한다

.

b. Non equi-join of the table “nation” with “n_nationkey” as a join key

1) Estimation

Non-equii-join 이므로 Hash Join 사용이 불가능하다. 이 경우 Nested Loop 와 Merge Join 중 더 빠른 Merge Join 이 적용될 것이라 생각한다.

2) Verification

```
postgres=# EXPLAIN ANALYZE SELECT * FROM nation as A join nation as B on A.n_nationkey < B.n_nationkey;
                                                      QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------
 Nested Loop  (cost=0.00..4406.15 rows=97200 width=244) (actual time=0.086..0.581 rows=325 loops=1)
   Join Filter: (a.n_nationkey < b.n_nationkey)
   Rows Removed by Join Filter: 351
   ->  Seq Scan on nation a  (cost=0.00..15.40 rows=540 width=122) (actual time=0.020..0.028 rows=26 loops=1)
   ->  Materialize  (cost=0.00..18.10 rows=540 width=122) (actual time=0.002..0.008 rows=26 loops=26)
         ->  Seq Scan on nation b  (cost=0.00..15.40 rows=540 width=122) (actual time=0.006..0.017 rows=26 loops=1)
 Planning Time: 0.225 ms
 Execution Time: 0.690 ms
(8 rows)
```

예측과는 다르게 Nested Loop Join 이 실행됐다. 그 이유를 추측해보면 모든 데이터를 nation key 값에 따라 정렬을 하고 merge 하는 것보다 테이블 b 의 모든 레코드에 대해서 그보다 작은 테이블 a 의 레코드를 찾아 join 하는 방식이 더 빠르게 수행된 것으로 보인다.

[Exercise 3]

Consider two self-join cases below:

a. Equi-join two tables of “supplier” and “table1” with “s_suppkey” and “sorted” as join keys.

b. Equi-join two tables of “supplier” and “table1” with “s_suppkey” and “unsorted” as join keys

1) Estimation

두 경우 모두 Hash Join 이 사용될 것이다. 두 테이블 모두 따로 정렬돼 있는 상황이 아니며, 현재 index scan 을 사용하지 않기 때문에 Hash join 알고리즘을 사용하는 것이 가장 빠를 것이다.

2) Verification

```
postgres=# EXPLAIN ANALYZE SELECT * FROM supplier, table1 WHERE s_suppkey = sorted;
                                                                  QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------------------------------
 Hash Join  (cost=442.00..229785.43 rows=145625 width=192) (actual time=6.612..2203.427 rows=50000 loops=1)
   Hash Cond: (table1.sorted = supplier.s_suppkey)
   ->  Seq Scan on table1  (cost=0.00..203093.21 rows=10000021 width=53) (actual time=0.026..1098.840 rows=10000000 loops=1)
   ->  Hash  (cost=317.00..317.00 rows=10000 width=139) (actual time=6.557..6.558 rows=10000 loops=1)
         Buckets: 16384  Batches: 1  Memory Usage: 1848kB
         ->  Seq Scan on supplier  (cost=0.00..317.00 rows=10000 width=139) (actual time=0.005..2.039 rows=10000 loops=1)
 Planning Time: 0.266 ms
 Execution Time: 2209.289 ms
(8 rows)

postgres=# EXPLAIN ANALYZE SELECT * FROM supplier, table1 WHERE s_suppkey = unsorted;
                                                                  QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------------------------------
 Hash Join  (cost=442.00..229785.33 rows=64136 width=192) (actual time=11.802..2364.509 rows=50164 loops=1)
   Hash Cond: (table1.unsorted = supplier.s_suppkey)
   ->  Seq Scan on table1  (cost=0.00..203093.21 rows=10000021 width=53) (actual time=0.012..1095.532 rows=10000000 loops=1)
   ->  Hash  (cost=317.00..317.00 rows=10000 width=139) (actual time=6.090..6.091 rows=10000 loops=1)
         Buckets: 16384  Batches: 1  Memory Usage: 1848kB
         ->  Seq Scan on supplier  (cost=0.00..317.00 rows=10000 width=139) (actual time=0.006..1.855 rows=10000 loops=1)
 Planning Time: 0.246 ms
 Execution Time: 2368.748 ms
(8 rows)
```

두 경우 모두 Hash Join 이 실행됐다. 또한 Join key 가 sorted 인지 unsorted 인지에 상관 없이 비슷한 실행시간이 소요됐다.

[Exercise 4]

Create three indexes:

- CREATE INDEX sorted_idx on table1(sorted);
- CREATE INDEX unsorted_idx on table1(unsorted);
- CREATE INDEX suppkey_idx on supplier(s_suppkey);

```
postgres=# CREATE INDEX sorted_idx on table1(sorted);
CREATE INDEX
postgres=# CREATE INDEX unsorted_idx on table1(unsorted);
CREATE INDEX
postgres=# CREATE INDEX suppkey_idx on supplier(s_suppkey);
CREATE INDEX
```

Consider two self-join cases below:

a. Equi-join two tables of “supplier” and “table1” with “s_suppkey” and “sorted” as join keys

b. Equi-join two tables of “supplier” and “table1” with “s_suppkey” and “unsorted” as join keys

1) Estimation

Sorted의 경우는 cluster index, Unsorted의 경우는 Secondary index가 만들어져 있다. 따라서 이 경우에는 Hash Join 보다 index scan을 활용한 Nested Loop Join 내지는 Merge Join이 더 빠를 수 있다고 생각한다. 특히 sorted의 경우는 이미 정렬이 돼 있기 때문에 merge join을 하기 더 유용할 것이다.

2) Verification

```
postgres=# EXPLAIN ANALYZE SELECT * FROM supplier, table1 WHERE s_suppkey = sorted;
                                                                QUERY PLAN
--------------------------------------------------------------------------------------------------------------------------------------------
 Merge Join  (cost=1.96..3882.36 rows=145624 width=192) (actual time=0.068..34.410 rows=50000 loops=1)
   Merge Cond: (supplier.s_suppkey = table1.sorted)
   ->  Index Scan using suppkey_idx on supplier  (cost=0.29..491.40 rows=10000 width=139) (actual time=0.012..2.874 rows=10000 loops=1)
   ->  Index Scan using sorted_idx on table1  (cost=0.43..310044.43 rows=10000000 width=53) (actual time=0.045..13.198 rows=50006 loops=1)
 Planning Time: 0.318 ms
 Execution Time: 39.336 ms
(6 rows)

postgres=# EXPLAIN ANALYZE SELECT * FROM supplier, table1 WHERE s_suppkey = unsorted;
                                                                  QUERY PLAN
------------------------------------------------------------------------------------------------------------------------------------------------
 Merge Join  (cost=3.71..4465.25 rows=64136 width=192) (actual time=0.189..2210.449 rows=50164 loops=1)
   Merge Cond: (supplier.s_suppkey = table1.unsorted)
   ->  Index Scan using suppkey_idx on supplier  (cost=0.29..491.40 rows=10000 width=139) (actual time=0.012..8.848 rows=10000 loops=1)
   ->  Index Scan using unsorted_idx on table1  (cost=0.43..619964.37 rows=10000000 width=53) (actual time=0.078..2153.825 rows=50168 loops=1)
 Planning Time: 1.860 ms
 Execution Time: 2221.450 ms
(6 rows)
```

두 경우 모두 Merge Join 이 실행됐다. 이때 각 테이블을 먼저 Sorting 하는 과정 대신 Index scan 으로 두 테이블간 key 값 비교가 이루어졌다. 이때 수행시간 면에서는 sorted 가 unsorted 에 비해 훨씬 빨랐다. 이는 앞서 언급한 것처럼 sorted 는 이미 정렬이 된 상태에서 merge Join 이 이루어지는 것이라 seek 와 transfer 횟수가 최적화된 반면에 unsorted 는 index scan 으로 원하는 레코드 근처로 접근한 뒤 다시 seek 하고 transfer 하는 작업이 추가로 진행됐을 것이다. 이러한 차이가 두 쿼리의 execution time 차이를 만들었다고 생각한다.