SWE3003-41: Database

Homework #2

2017313107 이승태

Deadline: Nov. 24. 11:59PM, 2021 (firm deadline, no late submission is allowed)

Submission: icampus; **(submit in PDF form)**

1. Ch9.example.sql에서 사용된 다음 Test 테이블에 대해,

```
CREATE TABLE test (a int, b int, c varchar2(650)) NOLOGGING TABLESPACE users;
-- Insert 1M tuples into TEST table (approximately 664 bytes per tuple)

BEGIN
    FOR i IN 1..1000000 LOOP
        INSERT INTO TEST (a, b, c) values (i, i, rpad('X', 650, 'X'));
    END LOOP;
END;
-- Oracle19c default block Size = 8K --> 10 tuples / block
-- ==> total block # of TEST table = 100000 blocks (= 800M)
```

a) Select sum(b) from test where a between 1 and 1000; 질의를 Oracle 이 full table scan 방식으로 처리하는 과정을 상세히 기술하시오.

이 구문은 range query 이므로, OLTP 쿼리에 해당된다. 10 tuple 당 1 개의 block 이 생성되므로, 10 만개의 block 이 생성된다. 그러므로 oracle 은 full table scan 을 할 때, 10 만개의 block 을 스캐닝 하면서 b 의 값이 1 에서 1000 사이의 값인지 판단하고 맞으면 값에 더해주게 된다.

b) 다음 index 가 존재할 때, index-based access method 로 위 질의를 처리하는 과정을 상세히 기술하시오.

create index test a on test(a);

만약 B+ index tree 를 사용한다고 한다면, leaf node 에서 record 에 관련된 정보들을 보고 해당 page 에 접근한다. range query 이므로 root node 에서 a = 1 인 leaf node 에 접근하고 그 node 부터 leaf block 의 linked list 를 타면서 a = 1000 인 부분까지 접근을 한다. node 의 정보를 보고 page 에 접근하며, a 의 값을 더해주게 된다. 또한, 이 쿼리문을 여러 번 사용하게 되면 buffer 에 정보가 남아 있게 되어서 더 빨리 실행될 수 있다.

2. Oracle clustering factor 관련해서,

a) 다음 명령을 사용해서 위 test_a 인덱스를 analyze하고, ch10.example.sql의 clustering factor (CF)을 구하는 질의 수행해서 얼마인지 제시하시오.

analyze index TEST_A compute statistics;

```
SQL> select blevel, leaf_blocks, distinct_keys, avg_leaf_blocks_per_key, avg_data_blocks_per_key,
  2         clustering_factor, last_analyzed
  3  from user_indexes
  4  where index_name = 'TEST_A';

    BLEVEL LEAF_BLOCKS DISTINCT_KEYS AVG_LEAF_BLOCKS_PER_KEY AVG_DATA_BLOCKS_PER_KEY CLUSTERING_FACTOR LAST_ANA
---------- ----------- ------------- ------------------------ ------------------------ ----------------- --------
         2        2226       1000000                        1                        1            100000 21/11/24
```

Clucstering_factor는 100000이 나온다.(well-clustered되어있다.)

b) Oracle이 analyze 명령 수행 시 CF값을 구하는 과정을 설명하시오.

B+ index tree를 사용한다면, 첫번째 leaf block부터 마지막 leaf block까지 linked list로 scan을 하면서 page에 접근을 시도한다. 이 때, page가 바뀌게 되면, clustering factor를 1을 추가하게 된다. leaf node의 scan이 끝났을 때, 최종적으로 clustering factor를 반환하게 된다.

c) 그리고, 1. Q1이 주어졌을 때, CF값이 어떻게 사용되는 사용되는지 "set autotrace on" 설정 후 Q1수행시, execution plan의 Cost에 어떻게 반영되는지 설명하시오.

```
SQL> set lines 200;
SQL> Select sum(b) from test where a between 1 and 1000;

    SUM(B)
----------
    500500

Execution Plan
----------------------------------------------------------
Plan hash value: 3058376349

-------------------------------------------------------------------------------------------------
| Id  | Operation                           | Name   | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                    |        |     1 |    26 |   103   (0)| 00:00:01 |
|   1 |  SORT AGGREGATE                     |        |     1 |    26 |            |          |
|   2 |   TABLE ACCESS BY INDEX ROWID BATCHED| TEST   |  1000 | 26000 |   103   (0)| 00:00:01 |
|*  3 |    INDEX RANGE SCAN                 | TEST_A |  1000 |       |     5   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   3 - access("A">=1 AND "A"<=1000)

Note
-----
   - dynamic statistics used: dynamic sampling (level=2)

Statistics
----------------------------------------------------------
          0  recursive calls
          0  db block gets
        105  consistent gets
          0  physical reads
          0  redo size
        573  bytes sent via SQL*Net to client
        412  bytes received via SQL*Net from client
          2  SQL*Net roundtrips to/from client
          0  sorts (memory)
          0  sorts (disk)
          1  rows processed
```

1000000개의 tuple이 존재하고 range가 1000이므로, selectivity는 0.001이다. Clustering factor는 100000이므로, 약 100개의 page접근이 일어난다. 또, index에 접근하는 비용인 index block traversals는 약 5이므로 이를 더하면 105가 나온다. 실제로는 그와 비슷한 값인 103이 나오게 된다.

3. 다음 질의를 full table scan, index로 각각 처리하는 과정을 설명하시오. 그리고, 각자 Oracle로 수행했을 때, oracle은 두 가지 방법 중 어느 방법을 사용하는지 그리고 왜 그 방법을 택하는지 설명하시오.

Select a, b from test order by a;

1) full table scan: buffer cache의 크기에 비해 데이터의 용량이 매우 크므로, external merge sort를 수행할 것이다.

2) index: B+ index tree를 사용하는 경우, root node에서 left-leaf node를 먼저 접근한 후, linked list를 이용하여 leaf block을 차례대로 접근한다. 접근한 node들에서 실제 정보가 들어간 page를 접근하여 값을 출력해준다. B+ index tree는 이미 sort가 되어있으므로, 별도의 sort과정이 필요 없다. 또한 well-clustered되어있지 않다면 page를 많이 접근하게 되어 별로 좋지 못한 방법이다.

Oracle은 full table scan을 이용하여 sort를 수행했다. Index는 index 접근비용, 실제 페이지의 접근 비용이 들게 되므로, sort가 미리 되어있지만 cost가 높게 나올 수 있다. 하지만 full table scan을 할 경우, page만 접근하고 sort를 진행하므로, 이 값이 더 적게 나올 수 있다. 이 상황에서는 oracle이 sort를 하고 page를 접근 하는게 더 빠르다고 판단하여 이러한 결정을 내렸다.

4. 다음 두 테이블 test, test2에 대해 Oracle에서 아래 조인 질의 Q2를 "set autotrace traceonly" 설정 후 수행 시 (Q2에서는 USE_NL 힌트를 통해서 오라클로 하여금 Nested Loop 조인을 사용하도록 강제하였음; NOTE: Oracle의 경우 tuple-oriented nested loop 조인만 지원함.) 동작과정을 상세히 설명하시오. 그리고, 이 과정과 set autotrace에서는 나오는 Statistics "consistent gets" 값과 관련해서 설명하시오. 해당 통계치는 간단히 buffer cache에서 page를 접근(즉, 9장에서 배운 내용 중 page PINNING에 해당) 하는 횟수를 나타냄.

```
Drop table test;
CREATE TABLE test (a int, b int, c varchar2(650)) NOLOGGING TABLESPACE
users;

BEGIN
    FOR i IN 1..1000 LOOP
        INSERT INTO TEST (a, b, c) values (i, i, rpad('X', 650, 'X'));
```

```
    END LOOP;
END;

CREATE TABLE test2 (a int, b int, c varchar2(650)) NOLOGGING TABLESPACE
users;

BEGIN
    FOR i IN 1..1000 LOOP
        INSERT INTO TEST2 (a, b, c) values (i, i, rpad('X', 650, 'X'));
    END LOOP;
END;

-- Oracle19c default block Size = 8K --> 10 tuples / block
-- ==> total block # of TEST and TEST2 tables = 100 blocks
```

Q2: select /*+ use_nl(t1,t2) */ t1.b, t2.b from test t1, test2 t2 where t1.a = t2.a;

```
SQL> select /*+ use_nl(t1,t2) */ t1.b, t2.b from test t1, test2 t2 where t1.a = t2.a;

1000 행이 선택되었습니다.

Execution Plan
----------------------------------------------------------
Plan hash value: 74026472

-------------------------------------------------------------------------------
| Id  | Operation          | Name  | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |       |   965 | 50180 | 27021   (1)| 00:00:02 |
|   1 |  NESTED LOOPS      |       |   965 | 50180 | 27021   (1)| 00:00:02 |
|   2 |   TABLE ACCESS FULL| TEST2 |   965 | 25090 |    30   (0)| 00:00:01 |
|*  3 |   TABLE ACCESS FULL| TEST  |     1 |    26 |    28   (0)| 00:00:01 |
-------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   3 - filter("T1"."A"="T2"."A")

Hint Report (identified by operation id / Query Block Name / Object Alias):
Total hints for statement: 1 (U - Unused (1))
---------------------------------------------------------------------------

   2 -  SEL$1 / T2@SEL$1
         U -  use_nl(t1,t2)

Note
-----
   - dynamic statistics used: dynamic sampling (level=2)


Statistics
----------------------------------------------------------
         15  recursive calls
          0  db block gets
     107386  consistent gets
          0  physical reads
          0  redo size
      22326  bytes sent via SQL*Net to client
       1167  bytes received via SQL*Net from client
         68  SQL*Net roundtrips to/from client
          2  sorts (memory)
          0  sorts (disk)
       1000  rows processed
```

2 중 for 문을 돌면서 t1.a 와 t2.a 의 값을 비교하고 같으면 결과 값에 추가해주는 방식으로 full table scan 과 nested loop 를 사용한다. Test2 의 tuple 의 개수가 1000 개, test 의 block 의 개수는 100 개로, test2 의 각 tuple(1000 개)에 대해 test 의 block 100 개를 접근한다. 그러므로 1000 * 100 의 consistent gets 가 발생하게 된다. 또 추가적으로, test table 에 접근하는 overhead 도 발생하므로 100000 보다 조금 더 큰 107386 의 consistent gets 가 발생하게 되었다.