SQL 중·상급 활용

2021. 10. 20.

Oracle SQL

Contents

1. Hint

- 힌트는 SQL 튜닝의 핵심부분으로 일종의 '지시구문'임
- SQL에 포함되어 쓰여져 Optimizer의 실행 계획을 원하는 대로 바꿀 수 있게 해 줌
- 오라클 Optimizer가 항상 최선의 실행 계획을 수립할 수는 없으므로, 테이블이나 인덱스의 잘못된 실행 계획을 개발자가 직접 바꿀 수 있도록 도와 줌
- 즉, 오라클 Optimizer에 의존하여 나온 실행 계획보다 훨씬 효율적인 실행 계획을 사용자가 구사할 수 있음
- **실행계획이 바뀐다고 해서 산출물이 바뀌는 건 아님**
- Hint 규칙 :

1. /*+ Hint */
2. 여러 개의 Hint를 섞어 쓸 수 있고, 해당 Hint들을 '공백'으로 구분함

   (ex : /*+ LEADING(A B) USE_HASH(B) */)
3. Hint내 인자도 '공백'으로 구분함
4. Hint내 인자로 사용된 테이블은 ALIAS(별칭)으로 정의되어야 하며, 계정명까지 포함해서 작성하지 않음

   (**잘못된 Hint 작성** : /*+ LEADING(SYSTEM.EMP SYSTEM.DEPT) */

   올바른 Hint 작성 : /*+ LEADING(EMP DEPT) */

   /*+ LEADING(A B) */)

2. 자주 사용되는 Hint

- Access 방식

Hint 이름	설명
/*+ FULL */	INDEX를 사용하지 않고, TABLE FULL SCAN 실시
/*+ INDEX */	INDEX를 활용한 SCAN 실시 (INDEX UNIQUE SCAN, INDEX RANGE SCAN)
/*+ INDEX_FFS */	INDEX FAST FULL SCAN 실시

- Join 순서

Hint 이름	설명
/*+ ORDERED */	FROM절에 나열된 순서대로 Join 실시
/*+ LEADING */	해당 Hint에 열거한 테이블 순서대로 Join 실시 (/*+ LEADING(A B C) */ : A, B, C 순서로 Join 실시)

2. 자주 사용되는 Hint

- Join 방식

Hint 이름	설명
/*+ USE_NL */	NL Join 실시
/*+ USE_MERGE */	Sort Merge Join 실시
/*+ USE_HASH */	Hash Join 실시

- WITH 동작 방식

Hint 이름	설명
/*+ INLINE */	WITH을 INLINE VIEW로 처리
/*+ MATERIALIZE */	WITH을 임시테이블(임시로 생성되는 실제 테이블)로 처리

2. 자주 사용되는 Hint

- 쿼리 변환

Hint 이름	설명
/*+ MERGE */	VIEW MERGING 실시
/*+ NO_MERGE */	VIEW MERGING 방지
/*+ UNNEST */	NESTED SUBQUERY UNNESTING 실시
/*+ NO_UNNEST */	NESTED SUBQUERY UNNESTING 방지
/*+ PUSH_SUBQ */	NESTED SUBQUERY를 우선적으로 실시 일반적으로 '/*+ NO_UNNEST */'Hint와 같이 사용됨

- VIEW MERGE : VIEW를 통해 데이터를 가져오는 작업을 최적화하고자, 해당 VIEW와 MAIN QUERY를 병합하는 것
  즉, VIEW를 해체함
- UNNEST : NESTED SUBQUERY와 MAIN QUERY를 합쳐 JOIN형태로 실행 계획을 변경하는 것
  즉, NESTED SUBQUERY를 해체함

FULL Hint

```
EXPLAIN PLAN FOR
SELECT /*+ FULL(A) */
       A.*
FROM SYSTEM.STUDENT A
WHERE SNO <= '930000';

SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY);
```

```
-------------------------------------------------------------------------------
| Id  | Operation          | Name    | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |         |    46 |  2530 |     2   (0)| 00:00:01 |
|*  1 |  TABLE ACCESS FULL| STUDENT |    46 |  2530 |     2   (0)| 00:00:01 |
-------------------------------------------------------------------------------

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

   1 - filter("SNO"<='930000')
```

TABLE FULL SCAN

```
EXPLAIN PLAN FOR
SELECT /*+ FULL(A) */
       A.*
FROM SYSTEM.STUDENT A
WHERE SNO <= '930000';

SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY);
```

```
-------------------------------------------------------------------------------
| Id  | Operation          | Name    | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |         |    46 |  2530 |     2   (0)| 00:00:01 |
|*  1 |  TABLE ACCESS FULL| STUDENT |    46 |  2530 |     2   (0)| 00:00:01 |
-------------------------------------------------------------------------------

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

   1 - filter("SNO"<='930000')
```

INDEX RANGE SCAN으로
변경하는 방법은?

INDEX Hint

```
EXPLAIN PLAN FOR
SELECT /*+ INDEX(A STUDENT_SNO_PK) */
       A.*
FROM SYSTEM.STUDENT A
WHERE SNO <= '930000';

SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY);
```

```
-----------------------------------------------------------------------------------------------------
| Id  | Operation                           | Name           | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                    |                |    46 |  2530 |     7   (0)| 00:00:01 |
|   1 |  TABLE ACCESS BY INDEX ROWID BATCHED| STUDENT        |    46 |  2530 |     7   (0)| 00:00:01 |
|*  2 |   INDEX RANGE SCAN                  | STUDENT_SNO_PK |    46 |       |     2   (0)| 00:00:01 |
-----------------------------------------------------------------------------------------------------

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

   2 - access("SNO"<='930000')
```

5. USE_NL

Join 결과

```
EXPLAIN PLAN FOR
SELECT A.SNO,
       A.SNAME,
       B.CNO,
       B.RESULT
FROM SYSTEM.STUDENT A,
     SYSTEM.SCORE B
WHERE A.SNO = B.SNO;

SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY);
```

```
-------------------------------------------------------------------------------
| Id  | Operation          | Name    | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |         |  2601 |   109K|     7   (0)| 00:00:01 |
|*  1 |  HASH JOIN         |         |  2601 |   109K|     7   (0)| 00:00:01 |
|   2 |   TABLE ACCESS FULL| STUDENT |    87 |  1566 |     2   (0)| 00:00:01 |
|   3 |   TABLE ACCESS FULL| SCORE   |  2601 | 65025 |     5   (0)| 00:00:01 |
-------------------------------------------------------------------------------

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

   1 - access("A"."SNO"="B"."SNO")
```

Nested Loops Join으로 변경하는 방법은?

5. USE_NL

USE_NL Hint

```
EXPLAIN PLAN FOR
SELECT /*+ LEADING(A B) USE_NL(B) */
       A.SNO,
       A.SNAME,
       B.CNO,
       B.RESULT
FROM SYSTEM.STUDENT A,
     SYSTEM.SCORE B
WHERE A.SNO = B.SNO;

SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY);
```

```
-------------------------------------------------------------------------------------------------
| Id  | Operation                    | Name             | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |                  |  2601 |   109K|   263   (0)| 00:00:01 |
|   1 |  NESTED LOOPS                |                  |  2601 |   109K|   263   (0)| 00:00:01 |
|   2 |   NESTED LOOPS               |                  |  2610 |   109K|   263   (0)| 00:00:01 |
|   3 |    TABLE ACCESS FULL         | STUDENT          |    87 |  1566 |     2   (0)| 00:00:01 |
|*  4 |    INDEX RANGE SCAN          | SCORE_SNO_CNO_PK |    30 |       |     1   (0)| 00:00:01 |
|   5 |   TABLE ACCESS BY INDEX ROWID| SCORE            |    30 |   750 |     3   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------------

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

   4 - access("A"."SNO"="B"."SNO")
```

Join 방식의 Hint를 사용할 때,
'LEADING' Hint도 함께 사용할 것

5. USE_NL

Join 결과

```
EXPLAIN PLAN FOR
SELECT /*+ LEADING(A B) USE_NL(B) */
       A.SNO,
       A.SNAME,
       B.CNO,
       B.RESULT
FROM SYSTEM.STUDENT A,
     SYSTEM.SCORE B
WHERE A.SNO = B.SNO;

SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY);
```

```
-----------------------------------------------------------------------------------------------------
| Id  | Operation                    | Name             | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |                  |  2601 |   109K|   263   (0)| 00:00:01 |
|   1 |  NESTED LOOPS                |                  |  2601 |   109K|   263   (0)| 00:00:01 |
|   2 |   NESTED LOOPS               |                  |  2610 |   109K|   263   (0)| 00:00:01 |
|   3 |    TABLE ACCESS FULL         | STUDENT          |    87 |  1566 |     2   (0)| 00:00:01 |
|*  4 |    INDEX RANGE SCAN          | SCORE_SNO_CNO_PK |    30 |       |     1   (0)| 00:00:01 |
|   5 |   TABLE ACCESS BY INDEX ROWID| SCORE            |    30 |   750 |     3   (0)| 00:00:01 |
-----------------------------------------------------------------------------------------------------

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

   4 - access("A"."SNO"="B"."SNO")
```

- SCORE TABLE에 대해 INDEX SCAN이 발생함
- 두 테이블에 TABLE FULL SCAN을 발생시킨 후 NL Join을 실시하는 방법은?

5. USE_NL

FULL Hint & USE_NL Hint

```
EXPLAIN PLAN FOR
SELECT /*+ LEADING(A B) USE_NL(B) FULL(A) FULL(B) */
       A.SNO,
       A.SNAME,
       B.CNO,
       B.RESULT
FROM SYSTEM.STUDENT A,
     SYSTEM.SCORE B
WHERE A.SNO = B.SNO;

SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY);
```

```
-------------------------------------------------------------------------------
| Id  | Operation          | Name    | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |         |  2601 |   109K|   292   (2)| 00:00:01 |
|   1 |  NESTED LOOPS      |         |  2601 |   109K|   292   (2)| 00:00:01 |
|   2 |   TABLE ACCESS FULL| STUDENT |    87 |  1566 |     2   (0)| 00:00:01 |
|*  3 |   TABLE ACCESS FULL| SCORE   |    30 |   750 |     3   (0)| 00:00:01 |
-------------------------------------------------------------------------------

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

   3 - filter("A"."SNO"="B"."SNO")
```

5. USE_NL

네 테이블 Join 결과

```
EXPLAIN PLAN FOR
SELECT A.SNAME,
       B.CNAME,
       C.RESULT,
       D.GRADE
FROM SYSTEM.STUDENT A,
     SYSTEM.COURSE B,
     SYSTEM.SCORE C,
     SYSTEM.SCGRADE D
WHERE A.SNO = C.SNO
      AND B.CNO = C.CNO
      AND C.RESULT <= D.HISCORE
      AND C.RESULT >= D.LOSCORE
      AND A.SNO <= '930000';

SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY);
```

```
-------------------------------------------------------------------------------------
| Id  | Operation             | Name    | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT      |         |     9 |   891 |    12   (0)| 00:00:01 |
|*  1 |  HASH JOIN            |         |     9 |   891 |    12   (0)| 00:00:01 |
|*  2 |   HASH JOIN           |         |     9 |   639 |    10   (0)| 00:00:01 |
|   3 |    MERGE JOIN CARTESIAN|        |   230 | 10580 |     5   (0)| 00:00:01 |
|   4 |     TABLE ACCESS FULL | SCGRADE |     5 |   140 |     2   (0)| 00:00:01 |
|   5 |     BUFFER SORT       |         |    46 |   828 |     3   (0)| 00:00:01 |
|*  6 |      TABLE ACCESS FULL| STUDENT |    46 |   828 |     1   (0)| 00:00:01 |
|*  7 |    TABLE ACCESS FULL  | SCORE   |  1367 | 34175 |     5   (0)| 00:00:01 |
|   8 |   TABLE ACCESS FULL   | COURSE  |    39 |  1092 |     2   (0)| 00:00:01 |
-------------------------------------------------------------------------------------

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

   1 - access("B"."CNO"="C"."CNO")
   2 - access("A"."SNO"="C"."SNO")
       filter("C"."RESULT"<="D"."HISCORE" AND "C"."RESULT">="D"."LOSCORE")
   6 - filter("A"."SNO"<='930000')
   7 - filter("C"."SNO"<='930000')
```

- Hash Join과 Sort Merge Join을 활용하여, 네 테이블에 대한 Join 계획을 세웠음
- Nested Loops Join과 Sort Merge Join을 활용하여, 네 테이블에 대한 Join을 실시하도록 변경하는 방법은?

(조건 : 1. Join 순서 : STUDENT -> SCORE -> COURSE -> SCGRADE

2. INDEX SCAN 필수 테이블 : STUDENT, COURSE

3. SCGRADE와 Join시에 Sort Merge Join을 발생시키고, 나머지 Join은 NL Join으로 수행)

5. USE_NL

네 테이블 Join에 대한 실행 계획 수정

```
EXPLAIN PLAN FOR
SELECT /*+ LEADING(A C B D) USE_NL(C) USE_NL(B) USE_MERGE(D) INDEX(A STUDENT_SNO_PK) INDEX(B COURSE_CNO_PK) */
       A.SNAME,
       B.CNAME,
       C.RESULT,
       D.GRADE
FROM SYSTEM.STUDENT A,
     SYSTEM.COURSE B,
     SYSTEM.SCORE C,
     SYSTEM.SCGRADE D
WHERE A.SNO = C.SNO
      AND B.CNO = C.CNO
      AND C.RESULT <= D.HISCORE
      AND C.RESULT >= D.LOSCORE
      AND A.SNO <= '930000';

SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY);
```

```
-----------------------------------------------------------------------------------------------------------
| Id  | Operation                              | Name            | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                       |                 |     9 |   891 |   827   (1)| 00:00:01 |
|   1 |  MERGE JOIN                            |                 |     9 |   891 |   827   (1)| 00:00:01 |
|   2 |   SORT JOIN                            |                 |   723 | 51333 |   824   (1)| 00:00:01 |
|   3 |    NESTED LOOPS                        |                 |   723 | 51333 |   823   (1)| 00:00:01 |
|   4 |     NESTED LOOPS                       |                 |   723 | 51333 |   823   (1)| 00:00:01 |
|   5 |      NESTED LOOPS                      |                 |   723 | 31089 |    99   (0)| 00:00:01 |
|   6 |       TABLE ACCESS BY INDEX ROWID BATCHED| STUDENT       |    46 |   828 |     7   (0)| 00:00:01 |
|*  7 |        INDEX RANGE SCAN                | STUDENT_SNO_PK  |    46 |       |     2   (0)| 00:00:01 |
|   8 |       TABLE ACCESS BY INDEX ROWID BATCHED| SCORE         |    16 |   400 |     2   (0)| 00:00:01 |
|*  9 |        INDEX RANGE SCAN                | SCORE_SNO_CNO_PK|     1 |       |     1   (0)| 00:00:01 |
|* 10 |      INDEX UNIQUE SCAN                 | COURSE_CNO_PK   |     1 |       |     0   (0)| 00:00:01 |
|  11 |     TABLE ACCESS BY INDEX ROWID        | COURSE          |     1 |    28 |     1   (0)| 00:00:01 |
|* 12 |   FILTER                               |                 |       |       |            |          |
|* 13 |    SORT JOIN                           |                 |     5 |   140 |     3  (34)| 00:00:01 |
|  14 |     TABLE ACCESS FULL                  | SCGRADE         |     5 |   140 |     2   (0)| 00:00:01 |
-----------------------------------------------------------------------------------------------------------

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

   7 - access("A"."SNO"<='930000')
   9 - access("A"."SNO"="C"."SNO")
       filter("C"."SNO"<='930000')
  10 - access("B"."CNO"="C"."CNO")
  12 - filter("C"."RESULT">="D"."LOSCORE")
  13 - access("C"."RESULT"<="D"."HISCORE")
       filter("C"."RESULT"<="D"."HISCORE")
```

- 이 실행계획을 통해,
  세 개 이상의 테이블을 조인할 때 해당 조인이
  한 번에 수행되지 않는 것을 알 수 있음
- 두 테이블을 조인하고,
  해당 조인 결과와 나머지 테이블을 조인하는 식으로
  수행됨

6. USE_HASH

Join 결과

```
EXPLAIN PLAN FOR
SELECT /*+ LEADING(A B) USE_NL(B) FULL(A) FULL(B) */
       A.SNO,
       A.SNAME,
       B.CNO,
       B.RESULT
FROM SYSTEM.STUDENT A,
     SYSTEM.SCORE B
WHERE A.SNO = B.SNO;

SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY);
```

```
-------------------------------------------------------------------------------
| Id  | Operation          | Name    | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |         |  2601 |   109K|   292   (2)| 00:00:01 |
|   1 |  NESTED LOOPS      |         |  2601 |   109K|   292   (2)| 00:00:01 |
|   2 |   TABLE ACCESS FULL| STUDENT |    87 |  1566 |     2   (0)| 00:00:01 |
|*  3 |   TABLE ACCESS FULL| SCORE   |    30 |   750 |     3   (0)| 00:00:01 |
-------------------------------------------------------------------------------

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

   3 - filter("A"."SNO"="B"."SNO")
```

Hash Join으로 변경하는 방법은?

6. USE_HASH

USE_HASH Hint

```
EXPLAIN PLAN FOR
SELECT /*+ LEADING(A B) USE_HASH(B) */
       A.SNO,
       A.SNAME,
       B.CNO,
       B.RESULT
FROM SYSTEM.STUDENT A,
     SYSTEM.SCORE B
WHERE A.SNO = B.SNO;

SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY);
```

```
-------------------------------------------------------------------------------
| Id  | Operation          | Name    | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |         |  2601 |   109K|     7   (0)| 00:00:01 |
|*  1 |  HASH JOIN         |         |  2601 |   109K|     7   (0)| 00:00:01 |
|   2 |   TABLE ACCESS FULL| STUDENT |    87 |  1566 |     2   (0)| 00:00:01 |
|   3 |   TABLE ACCESS FULL| SCORE   |  2601 | 65025 |     5   (0)| 00:00:01 |
-------------------------------------------------------------------------------

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

   1 - access("A"."SNO"="B"."SNO")
```

6. USE_HASH

Join 결과

```
EXPLAIN PLAN FOR
SELECT A.SNO,
       A.SNAME,
       B.CNO,
       B.RESULT
FROM SYSTEM.STUDENT A,
     SYSTEM.SCORE B
WHERE A.SNO = B.SNO
      AND A.SNO <= '930000';

SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY);
```

```
-------------------------------------------------------------------------------
| Id  | Operation          | Name    | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |         |   723 | 31089 |     7   (0)| 00:00:01 |
|*  1 |  HASH JOIN         |         |   723 | 31089 |     7   (0)| 00:00:01 |
|*  2 |   TABLE ACCESS FULL| STUDENT |    46 |   828 |     2   (0)| 00:00:01 |
|*  3 |   TABLE ACCESS FULL| SCORE   |  1367 | 34175 |     5   (0)| 00:00:01 |
-------------------------------------------------------------------------------

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

   1 - access("A"."SNO"="B"."SNO")
   2 - filter("A"."SNO"<='930000')
   3 - filter("B"."SNO"<='930000')
```

STUDENT TABLE에 INDEX RANGE SCAN을 발생시킨 후, 해당 결과를 Build Input으로 하여 Hash Join을 실시하는 방법은?

6. USE_HASH

INDEX Hint & USE_HASH Hint

```
EXPLAIN PLAN FOR
SELECT /*+ LEADING(A B) USE_HASH(B) INDEX(A STUDENT_SNO_PK) */
       A.SNO,
       A.SNAME,
       B.CNO,
       B.RESULT
FROM SYSTEM.STUDENT A,
     SYSTEM.SCORE B
WHERE A.SNO = B.SNO
      AND A.SNO <= '930000';

SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY);
```

```
-----------------------------------------------------------------------------------------------------
| Id  | Operation                           | Name           | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                    |                |   723 | 31089 |    12   (0)| 00:00:01 |
|*  1 |  HASH JOIN                          |                |   723 | 31089 |    12   (0)| 00:00:01 |
|   2 |   TABLE ACCESS BY INDEX ROWID BATCHED| STUDENT        |    46 |   828 |     7   (0)| 00:00:01 |
|*  3 |    INDEX RANGE SCAN                 | STUDENT_SNO_PK |    46 |       |     2   (0)| 00:00:01 |
|*  4 |   TABLE ACCESS FULL                 | SCORE          |  1367 | 34175 |     5   (0)| 00:00:01 |
-----------------------------------------------------------------------------------------------------

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

   1 - access("A"."SNO"="B"."SNO")
   3 - access("A"."SNO"<='930000')
   4 - filter("B"."SNO"<='930000')
```

7. NO_MERGE

INLINE VIEW 두 개 사용

```
EXPLAIN PLAN FOR
SELECT SNAME,
       SUM(A_CNT) AS A_CNT,
       SUM(B_CNT) AS B_CNT,
       SUM(C_CNT) AS C_CNT,
       SUM(D_CNT) AS D_CNT,
       SUM(F_CNT) AS F_CNT
FROM (
    SELECT SNAME,
           CASE WHEN GRADE = 'A' THEN CNT ELSE 0 END AS A_CNT,
           CASE WHEN GRADE = 'B' THEN CNT ELSE 0 END AS B_CNT,
           CASE WHEN GRADE = 'C' THEN CNT ELSE 0 END AS C_CNT,
           CASE WHEN GRADE = 'D' THEN CNT ELSE 0 END AS D_CNT,
           CASE WHEN GRADE = 'F' THEN CNT ELSE 0 END AS F_CNT
    FROM (
        SELECT A.SNAME,
               D.GRADE,
               COUNT(*) AS CNT
        FROM SYSTEM.STUDENT A,
             SYSTEM.COURSE B,
             SYSTEM.SCORE C,
             SYSTEM.SCGRADE D
        WHERE A.SNO = C.SNO
          AND B.CNO = C.CNO
          AND C.RESULT <= D.HISCORE
          AND C.RESULT >= D.LOSCORE
          AND A.SNO IN ('944503', '925602')
        GROUP BY A.SNAME, D.GRADE
    )
)
GROUP BY SNAME;

SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY);
```

① ②

```
-------------------------------------------------------------------------------------------
| Id  | Operation                | Name    | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT         |         |     1 |    77 |    10  (10)| 00:00:01 |
|   1 |  HASH GROUP BY           |         |     1 |    77 |    10  (10)| 00:00:01 |
|   2 |   VIEW                   |         |     1 |    77 |    10  (10)| 00:00:01 |
|   3 |    HASH GROUP BY         |         |     1 |    65 |    10  (10)| 00:00:01 |
|*  4 |     HASH JOIN            |         |     1 |    65 |     9   (0)| 00:00:01 |
|   5 |      MERGE JOIN CARTESIAN|         |    10 |   460 |     4   (0)| 00:00:01 |
|*  6 |       TABLE ACCESS FULL  | STUDENT |     2 |    36 |     2   (0)| 00:00:01 |
|   7 |       BUFFER SORT        |         |     5 |   140 |     2   (0)| 00:00:01 |
|   8 |        TABLE ACCESS FULL | SCGRADE |     5 |   140 |     1   (0)| 00:00:01 |
|*  9 |      TABLE ACCESS FULL   | SCORE   |    62 |  1178 |     5   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------

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

   4 - access("A"."SNO"="C"."SNO")
       filter("C"."RESULT"<="D"."HISCORE" AND "C"."RESULT">="D"."LOSCORE")
   6 - filter("A"."SNO"='925602' OR "A"."SNO"='944503')
   9 - filter("C"."SNO"='925602' OR "C"."SNO"='944503')
```

VIEW Operation :
Oracle이 연산 작업을 일시중지 한 후,
VEIW 내부 연산들에 의해 생성된
중간 결과 집합을 메모리에 생성함

- 실행계획상 나타난 VIEW Operation은 '**1번**'또는 '**2번**'INLINE VIEW에 대한 Operation임
- 만약 두 INLINE VIEW 결과 모두 메모리에 생성하고 싶다면, 어떻게 해야 할까?
  (즉, VIEW Operation이 두 번 발생해야 함)

7. NO_MERGE

NO_MERGE Hint

```
EXPLAIN PLAN FOR
SELECT SNAME,
       SUM(A_CNT) AS A_CNT,
       SUM(B_CNT) AS B_CNT,
       SUM(C_CNT) AS C_CNT,
       SUM(D_CNT) AS D_CNT,
       SUM(F_CNT) AS F_CNT
FROM (
   SELECT /*+ NO_MERGE */
          SNAME,
          CASE WHEN GRADE = 'A' THEN CNT ELSE 0 END AS A_CNT,
          CASE WHEN GRADE = 'B' THEN CNT ELSE 0 END AS B_CNT,
          CASE WHEN GRADE = 'C' THEN CNT ELSE 0 END AS C_CNT,
          CASE WHEN GRADE = 'D' THEN CNT ELSE 0 END AS D_CNT,
          CASE WHEN GRADE = 'F' THEN CNT ELSE 0 END AS F_CNT
   FROM (
      SELECT /*+ NO_MERGE */
             A.SNAME,
             D.GRADE,
             COUNT(*) AS CNT
      FROM SYSTEM.STUDENT A,
           SYSTEM.COURSE B,
           SYSTEM.SCORE C,
           SYSTEM.SCGRADE D
      WHERE A.SNO = C.SNO
            AND B.CNO = C.CNO
            AND C.RESULT <= D.HISCORE
            AND C.RESULT >= D.LOSCORE
            AND A.SNO IN ('944503', '925602')
      GROUP BY A.SNAME, D.GRADE
   )
)
GROUP BY SNAME;

SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY);
```

```
-------------------------------------------------------------------------------------------
| Id  | Operation                | Name    | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT         |         |     1 |    77 |    10  (10)| 00:00:01 |
|   1 |  HASH GROUP BY           |         |     1 |    77 |    10  (10)| 00:00:01 |
|   2 |   VIEW                   |         |     1 |    77 |    10  (10)| 00:00:01 |
|   3 |    VIEW                  |         |     1 |    27 |    10  (10)| 00:00:01 |
|   4 |     HASH GROUP BY        |         |     1 |    65 |    10  (10)| 00:00:01 |
|*  5 |      HASH JOIN           |         |     1 |    65 |     9   (0)| 00:00:01 |
|   6 |       MERGE JOIN CARTESIAN|        |    10 |   460 |     4   (0)| 00:00:01 |
|*  7 |        TABLE ACCESS FULL | STUDENT |     2 |    36 |     2   (0)| 00:00:01 |
|   8 |        BUFFER SORT       |         |     5 |   140 |     2   (0)| 00:00:01 |
|   9 |         TABLE ACCESS FULL| SCGRADE |     5 |   140 |     1   (0)| 00:00:01 |
|* 10 |       TABLE ACCESS FULL  | SCORE   |    62 |  1178 |     5   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------

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

   5 - access("A"."SNO"="C"."SNO")
       filter("C"."RESULT"<="D"."HISCORE" AND "C"."RESULT">="D"."LOSCORE")
   7 - filter("A"."SNO"='925602' OR "A"."SNO"='944503')
  10 - filter("C"."SNO"='925602' OR "C"."SNO"='944503')
```

8. INLINE

INLINE Hint

```
WITH COUNT_BY_STDNT_N_GRADE AS (
    SELECT /*+ INLINE */
           A.SNAME,
           D.GRADE,
           COUNT(*) AS CNT
    FROM SYSTEM.STUDENT A,
         SYSTEM.COURSE B,
         SYSTEM.SCORE C,
         SYSTEM.SCGRADE D
    WHERE A.SNO = C.SNO
          AND B.CNO = C.CNO
          AND C.RESULT <= D.HISCORE
          AND C.RESULT >= D.LOSCORE
          AND A.SNO IN ('944503', '925602')
    GROUP BY A.SNAME, D.GRADE
),

TMP_PIVOT_TABLE AS (
    SELECT /*+ INLINE */
           SNAME,
           CASE WHEN GRADE = 'A' THEN CNT ELSE 0 END AS A_CNT,
           CASE WHEN GRADE = 'B' THEN CNT ELSE 0 END AS B_CNT,
           CASE WHEN GRADE = 'C' THEN CNT ELSE 0 END AS C_CNT,
           CASE WHEN GRADE = 'D' THEN CNT ELSE 0 END AS D_CNT,
           CASE WHEN GRADE = 'F' THEN CNT ELSE 0 END AS F_CNT
    FROM COUNT_BY_STDNT_N_GRADE
)

SELECT SNAME,
       SUM(A_CNT) AS A_CNT,
       SUM(B_CNT) AS B_CNT,
       SUM(C_CNT) AS C_CNT,
       SUM(D_CNT) AS D_CNT,
       SUM(F_CNT) AS F_CNT
FROM TMP_PIVOT_TABLE
GROUP BY SNAME;

SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY);
```

```
-------------------------------------------------------------------------------------
| Id  | Operation                | Name    | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT         |         |     1 |    77 |    10  (10)| 00:00:01 |
|   1 |  HASH GROUP BY           |         |     1 |    77 |    10  (10)| 00:00:01 |
|   2 |   VIEW                   |         |     1 |    77 |    10  (10)| 00:00:01 |
|   3 |    HASH GROUP BY         |         |     1 |    65 |    10  (10)| 00:00:01 |
|*  4 |     HASH JOIN            |         |     1 |    65 |     9   (0)| 00:00:01 |
|   5 |      MERGE JOIN CARTESIAN|         |    10 |   460 |     4   (0)| 00:00:01 |
|*  6 |       TABLE ACCESS FULL  | STUDENT |     2 |    36 |     2   (0)| 00:00:01 |
|   7 |       BUFFER SORT        |         |     5 |   140 |     2   (0)| 00:00:01 |
|   8 |        TABLE ACCESS FULL | SCGRADE |     5 |   140 |     1   (0)| 00:00:01 |
|*  9 |      TABLE ACCESS FULL   | SCORE   |    62 |  1178 |     5   (0)| 00:00:01 |
-------------------------------------------------------------------------------------

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

   4 - access("A"."SNO"="C"."SNO")
       filter("C"."RESULT"<="D"."HISCORE" AND "C"."RESULT">="D"."LOSCORE")
   6 - filter("A"."SNO"='925602' OR "A"."SNO"='944503')
   9 - filter("C"."SNO"='925602' OR "C"."SNO"='944503')
```

- WITH은 기본적으로 'INLINE VIEW'형식으로 동작함
- 또한, 특정 WITH이 두 번 이상 참조되면, 해당 WITH은 'MATERIALIZE'형식으로 동작함
- WITH이 두 번 이상 참조되더라도 해당 WITH을 INLINE VIEW형식으로 동작하게 만들고 싶다면, INLINE Hint를 WITH 내부에 기입하면 됨

8. INLINE

INLINE Hint

```
WITH COUNT_BY_STDNT_N_GRADE AS (
    SELECT /*+ INLINE */
           A.SNAME,
           D.GRADE,
           COUNT(*) AS CNT
    FROM SYSTEM.STUDENT A,
         SYSTEM.COURSE B,
         SYSTEM.SCORE C,
         SYSTEM.SCGRADE D
    WHERE A.SNO = C.SNO
          AND B.CNO = C.CNO
          AND C.RESULT <= D.HISCORE
          AND C.RESULT >= D.LOSCORE
          AND A.SNO IN ('944503', '925602')
    GROUP BY A.SNAME, D.GRADE
),

TMP_PIVOT_TABLE AS (
    SELECT /*+ INLINE */
           SNAME,
           CASE WHEN GRADE = 'A' THEN CNT ELSE 0 END AS A_CNT,
           CASE WHEN GRADE = 'B' THEN CNT ELSE 0 END AS B_CNT,
           CASE WHEN GRADE = 'C' THEN CNT ELSE 0 END AS C_CNT,
           CASE WHEN GRADE = 'D' THEN CNT ELSE 0 END AS D_CNT,
           CASE WHEN GRADE = 'F' THEN CNT ELSE 0 END AS F_CNT
    FROM COUNT_BY_STDNT_N_GRADE
)

SELECT SNAME,
       SUM(A_CNT) AS A_CNT,
       SUM(B_CNT) AS B_CNT,
       SUM(C_CNT) AS C_CNT,
       SUM(D_CNT) AS D_CNT,
       SUM(F_CNT) AS F_CNT
FROM TMP_PIVOT_TABLE
GROUP BY SNAME;

SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY);
```

```
-------------------------------------------------------------------------------------------
| Id  | Operation                | Name    | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT         |         |     1 |    77 |    10  (10)| 00:00:01 |
|   1 |  HASH GROUP BY           |         |     1 |    77 |    10  (10)| 00:00:01 |
|   2 |   VIEW                   |         |     1 |    77 |    10  (10)| 00:00:01 |
|   3 |    HASH GROUP BY         |         |     1 |    65 |    10  (10)| 00:00:01 |
|*  4 |     HASH JOIN            |         |     1 |    65 |     9   (0)| 00:00:01 |
|   5 |      MERGE JOIN CARTESIAN|         |    10 |   460 |     4   (0)| 00:00:01 |
|*  6 |       TABLE ACCESS FULL  | STUDENT |     2 |    36 |     2   (0)| 00:00:01 |
|   7 |       BUFFER SORT        |         |     5 |   140 |     2   (0)| 00:00:01 |
|   8 |        TABLE ACCESS FULL | SCGRADE |     5 |   140 |     1   (0)| 00:00:01 |
|*  9 |      TABLE ACCESS FULL   | SCORE   |    62 |  1178 |     5   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------

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

   4 - access("A"."SNO"="C"."SNO")
       filter("C"."RESULT"<="D"."HISCORE" AND "C"."RESULT">="D"."LOSCORE")
   6 - filter("A"."SNO"='925602' OR "A"."SNO"='944503')
   9 - filter("C"."SNO"='925602' OR "C"."SNO"='944503')
```

- 실행계획상 나와있는 VIEW Operation은 'COUNT_BY_STDNT_N_GRADE'WITH에 대한 연산임
- INLINE VIEW로 동작하는 'TMP_PIVOT_TABLE'WITH이 해체되어, 해당 WITH에 대한 VIEW Operation이 발생하지 않음
- 만약 'TMP_PIVOT_TABLE'WITH에 대해서도 VIEW Operation을 발생시키고 싶다면, 어떻게 해야 할까?

8. INLINE

INLINE Hint & NO_MERGE Hint

```
EXPLAIN PLAN FOR
WITH COUNT_BY_STDNT_N_GRADE AS (
    SELECT /*+ INLINE NO_MERGE */
           A.SNAME,
           D.GRADE,
           COUNT(*) AS CNT
    FROM SYSTEM.STUDENT A,
         SYSTEM.COURSE B,
         SYSTEM.SCORE C,
         SYSTEM.SCGRADE D
    WHERE A.SNO = C.SNO
          AND B.CNO = C.CNO
          AND C.RESULT <= D.HISCORE
          AND C.RESULT >= D.LOSCORE
          AND A.SNO IN ('944503', '925602')
    GROUP BY A.SNAME, D.GRADE
),
TMP_PIVOT_TABLE AS (
    SELECT /*+ INLINE NO_MERGE */
           SNAME,
           CASE WHEN GRADE = 'A' THEN CNT ELSE 0 END AS A_CNT,
           CASE WHEN GRADE = 'B' THEN CNT ELSE 0 END AS B_CNT,
           CASE WHEN GRADE = 'C' THEN CNT ELSE 0 END AS C_CNT,
           CASE WHEN GRADE = 'D' THEN CNT ELSE 0 END AS D_CNT,
           CASE WHEN GRADE = 'F' THEN CNT ELSE 0 END AS F_CNT
    FROM COUNT_BY_STDNT_N_GRADE
)
SELECT SNAME,
       SUM(A_CNT) AS A_CNT,
       SUM(B_CNT) AS B_CNT,
       SUM(C_CNT) AS C_CNT,
       SUM(D_CNT) AS D_CNT,
       SUM(F_CNT) AS F_CNT
FROM TMP_PIVOT_TABLE
GROUP BY SNAME;

SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY);
```

```
-------------------------------------------------------------------------------------
| Id  | Operation                    | Name    | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |         |     1 |    77 |    10  (10)| 00:00:01 |
|   1 |  HASH GROUP BY               |         |     1 |    77 |    10  (10)| 00:00:01 |
|   2 |   VIEW                       |         |     1 |    77 |    10  (10)| 00:00:01 |
|   3 |    VIEW                      |         |     1 |    27 |    10  (10)| 00:00:01 |
|   4 |     HASH GROUP BY            |         |     1 |    65 |    10  (10)| 00:00:01 |
|*  5 |      HASH JOIN               |         |     1 |    65 |     9   (0)| 00:00:01 |
|   6 |       MERGE JOIN CARTESIAN   |         |    10 |   460 |     4   (0)| 00:00:01 |
|*  7 |        TABLE ACCESS FULL     | STUDENT |     2 |    36 |     2   (0)| 00:00:01 |
|   8 |        BUFFER SORT           |         |     5 |   140 |     2   (0)| 00:00:01 |
|   9 |         TABLE ACCESS FULL    | SCGRADE |     5 |   140 |     1   (0)| 00:00:01 |
|* 10 |       TABLE ACCESS FULL      | SCORE   |    62 |  1178 |     5   (0)| 00:00:01 |
-------------------------------------------------------------------------------------

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

   5 - access("A"."SNO"="C"."SNO")
       filter("C"."RESULT"<="D"."HISCORE" AND "C"."RESULT">="D"."LOSCORE")
   7 - filter("A"."SNO"='925602' OR "A"."SNO"='944503')
  10 - filter("C"."SNO"='925602' OR "C"."SNO"='944503')
```

INLINE VIEW WITH 사용

```
EXPLAIN PLAN FOR
WITH COUNT_BY_STDNT_N_GRADE AS (
    SELECT /*+ INLINE NO_MERGE */
           A.SNAME,
           D.GRADE,
           COUNT(*) AS CNT
    FROM SYSTEM.STUDENT A,
         SYSTEM.COURSE B,
         SYSTEM.SCORE C,
         SYSTEM.SCGRADE D
    WHERE A.SNO = C.SNO
          AND B.CNO = C.CNO
          AND C.RESULT <= D.HISCORE
          AND C.RESULT >= D.LOSCORE
          AND A.SNO IN ('944503', '925602')
    GROUP BY A.SNAME, D.GRADE
),
TMP_PIVOT_TABLE AS (
    SELECT /*+ INLINE NO_MERGE */
           SNAME,
           CASE WHEN GRADE = 'A' THEN CNT ELSE 0 END AS A_CNT,
           CASE WHEN GRADE = 'B' THEN CNT ELSE 0 END AS B_CNT,
           CASE WHEN GRADE = 'C' THEN CNT ELSE 0 END AS C_CNT,
           CASE WHEN GRADE = 'D' THEN CNT ELSE 0 END AS D_CNT,
           CASE WHEN GRADE = 'F' THEN CNT ELSE 0 END AS F_CNT
    FROM COUNT_BY_STDNT_N_GRADE
)
SELECT SNAME,
       SUM(A_CNT) AS A_CNT,
       SUM(B_CNT) AS B_CNT,
       SUM(C_CNT) AS C_CNT,
       SUM(D_CNT) AS D_CNT,
       SUM(F_CNT) AS F_CNT
FROM TMP_PIVOT_TABLE
GROUP BY SNAME;

SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY);
```

```
-------------------------------------------------------------------------------------
| Id  | Operation                    | Name    | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |         |     1 |    77 |    10  (10)| 00:00:01 |
|   1 |  HASH GROUP BY               |         |     1 |    77 |    10  (10)| 00:00:01 |
|   2 |   VIEW                       |         |     1 |    77 |    10  (10)| 00:00:01 |
|   3 |    VIEW                      |         |     1 |    27 |    10  (10)| 00:00:01 |
|   4 |     HASH GROUP BY            |         |     1 |    65 |    10  (10)| 00:00:01 |
|*  5 |      HASH JOIN               |         |     1 |    65 |     9   (0)| 00:00:01 |
|   6 |       MERGE JOIN CARTESIAN   |         |    10 |   460 |     4   (0)| 00:00:01 |
|*  7 |        TABLE ACCESS FULL     | STUDENT |     2 |    36 |     2   (0)| 00:00:01 |
|   8 |        BUFFER SORT           |         |     5 |   140 |     2   (0)| 00:00:01 |
|   9 |         TABLE ACCESS FULL    | SCGRADE |     5 |   140 |     1   (0)| 00:00:01 |
|* 10 |       TABLE ACCESS FULL      | SCORE   |    62 |  1178 |     5   (0)| 00:00:01 |
-------------------------------------------------------------------------------------

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

   5 - access("A"."SNO"="C"."SNO")
       filter("C"."RESULT"<="D"."HISCORE" AND "C"."RESULT">="D"."LOSCORE")
   7 - filter("A"."SNO"='925602' OR "A"."SNO"='944503')
  10 - filter("C"."SNO"='925602' OR "C"."SNO"='944503')
```

- 앞서 봤듯이, 두 개의 WITH은 INLINE VIEW로 동작하고 있음
- 해당 WITH들을 VIEW가 아닌 임시테이블로 만드는 방법은?

9. MATERIALIZE

MATERIALIZE Hint

```
EXPLAIN PLAN FOR
WITH COUNT_BY_STDNT_N_GRADE AS (
    SELECT /*+ MATERIALIZE */
           A.SNAME,
           D.GRADE,
           COUNT(*) AS CNT
    FROM SYSTEM.STUDENT A,
         SYSTEM.COURSE B,
         SYSTEM.SCORE C,
         SYSTEM.SCGRADE D
    WHERE A.SNO = C.SNO
          AND B.CNO = C.CNO
          AND C.RESULT <= D.HISCORE
          AND C.RESULT >= D.LOSCORE
          AND A.SNO IN ('944503', '925602')
    GROUP BY A.SNAME, D.GRADE
),
TMP_PIVOT_TABLE AS (
    SELECT /*+ MATERIALIZE */
           SNAME,
           CASE WHEN GRADE = 'A' THEN CNT ELSE 0 END AS A_CNT,
           CASE WHEN GRADE = 'B' THEN CNT ELSE 0 END AS B_CNT,
           CASE WHEN GRADE = 'C' THEN CNT ELSE 0 END AS C_CNT,
           CASE WHEN GRADE = 'D' THEN CNT ELSE 0 END AS D_CNT,
           CASE WHEN GRADE = 'F' THEN CNT ELSE 0 END AS F_CNT
    FROM COUNT_BY_STDNT_N_GRADE
)
SELECT SNAME,
       SUM(A_CNT) AS A_CNT,
       SUM(B_CNT) AS B_CNT,
       SUM(C_CNT) AS C_CNT,
       SUM(D_CNT) AS D_CNT,
       SUM(F_CNT) AS F_CNT
FROM TMP_PIVOT_TABLE
GROUP BY SNAME;

SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY);
```

```
-----------------------------------------------------------------------------------------------------------------------
| Id  | Operation                              | Name                 | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                       |                      |     1 |    77 |    15  (14)| 00:00:01 |
|   1 |  TEMP TABLE TRANSFORMATION             |                      |       |       |            |          |
|   2 |   LOAD AS SELECT (CURSOR DURATION MEMORY)| SYS_TEMP_0FD9D6614_75D5F7 |       |       |            |          |
|   3 |    HASH GROUP BY                       |                      |     1 |    65 |    10  (10)| 00:00:01 |
|*  4 |     HASH JOIN                          |                      |     1 |    65 |     9   (0)| 00:00:01 |
|   5 |      MERGE JOIN CARTESIAN              |                      |    10 |   460 |     4   (0)| 00:00:01 |
|*  6 |       TABLE ACCESS FULL                | STUDENT              |     2 |    36 |     2   (0)| 00:00:01 |
|   7 |       BUFFER SORT                      |                      |     5 |   140 |     2   (0)| 00:00:01 |
|   8 |        TABLE ACCESS FULL               | SCGRADE              |     5 |   140 |     1   (0)| 00:00:01 |
|*  9 |      TABLE ACCESS FULL                 | SCORE                |    62 |  1178 |     5   (0)| 00:00:01 |
|  10 |   LOAD AS SELECT (CURSOR DURATION MEMORY)| SYS_TEMP_0FD9D6615_75D5F7 |       |       |            |          |
|  11 |    VIEW                                |                      |     1 |    27 |     2   (0)| 00:00:01 |
|  12 |     TABLE ACCESS FULL                  | SYS_TEMP_0FD9D6614_75D5F7 |     1 |    27 |     2   (0)| 00:00:01 |
|  13 |   HASH GROUP BY                        |                      |     1 |    77 |     3  (34)| 00:00:01 |
|  14 |    VIEW                                |                      |     1 |    77 |     2   (0)| 00:00:01 |
|  15 |     TABLE ACCESS FULL                  | SYS_TEMP_0FD9D6615_75D5F7 |     1 |    27 |     2   (0)| 00:00:01 |
-----------------------------------------------------------------------------------------------------------------------

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

   4 - access("A"."SNO"="C"."SNO")
       filter("C"."RESULT"<="D"."HISCORE" AND "C"."RESULT">="D"."LOSCORE")
   6 - filter("A"."SNO"='925602' OR "A"."SNO"='944503')
   9 - filter("C"."SNO"='925602' OR "C"."SNO"='944503')
```

9. WITH INLINE VS WITH MATERIALIZE

WITH INLINE

```
EXPLAIN PLAN FOR
WITH TARGET_EMP_INFO AS (
    SELECT /*+ INLINE NO_MERGE */
           A.ENO,
           A.ENAME,
           A.JOB,
           A.SAL,
           A.DNO
    FROM SYSTEM.EMP A,
         SYSTEM.BEST_EMP B
    WHERE A.ENO = B.ENO
),

TMP_JOIN_WITH_DEPT_TBL AS (
    SELECT /*+ INLINE NO_MERGE */
           D.DNO,
           D.DNAME,
           D.LOC
    FROM TARGET_EMP_INFO C,
         SYSTEM.DEPT D
    WHERE C.DNO = D.DNO
)

SELECT /*+ LEADING(X1 X2) */
       X1.ENAME,
       X1.JOB,
       X1.SAL,
       X2.DNAME,
       X2.LOC
FROM TARGET_EMP_INFO X1,
     TMP_JOIN_WITH_DEPT_TBL X2
WHERE X1.DNO = X2.DNO;

SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY);
```

```
-------------------------------------------------------------------------------------------
| Id  | Operation                    | Name         | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |              |     2 |   110 |     6   (0)| 00:00:01 |
|*  1 |  HASH JOIN                   |              |     2 |   110 |     6   (0)| 00:00:01 |
|   2 |   VIEW                       |              |     2 |    80 |     2   (0)| 00:00:01 |
|   3 |    NESTED LOOPS              |              |     2 |    72 |     2   (0)| 00:00:01 |
|   4 |     TABLE ACCESS FULL        | EMP          |    22 |   682 |     2   (0)| 00:00:01 |
|*  5 |     INDEX UNIQUE SCAN        | BEST_EMP_PK  |     1 |     5 |     0   (0)| 00:00:01 |
|   6 |   VIEW                       |              |     2 |    30 |     4   (0)| 00:00:01 |
|*  7 |    HASH JOIN                 |              |     2 |    34 |     4   (0)| 00:00:01 |
|   8 |     VIEW                     |              |     2 |     6 |     2   (0)| 00:00:01 |
|   9 |      NESTED LOOPS            |              |     2 |    26 |     2   (0)| 00:00:01 |
|  10 |       TABLE ACCESS FULL      | EMP          |    22 |   176 |     2   (0)| 00:00:01 |
|* 11 |       INDEX UNIQUE SCAN      | BEST_EMP_PK  |     1 |     5 |     0   (0)| 00:00:01 |
|  12 |     TABLE ACCESS FULL        | DEPT         |     7 |    98 |     2   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------

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

   1 - access("X1"."DNO"="X2"."DNO")
   5 - access("A"."ENO"="B"."ENO")
   7 - access("C"."DNO"="D"."DNO")
  11 - access("A"."ENO"="B"."ENO")
```

- 'TARGET_EMP_INFO' WITH 내부 Operation이 두 번 실행됨

9. WITH INLINE VS WITH MATERIALIZE

WITH MATERIALIZE

```
EXPLAIN PLAN FOR
WITH TARGET_EMP_INFO AS (
    SELECT /*+ MATERIALIZE */
           A.ENO,
           A.ENAME,
           A.JOB,
           A.SAL,
           A.DNO
    FROM SYSTEM.EMP A,
         SYSTEM.BEST_EMP B
    WHERE A.ENO = B.ENO
),

TMP_JOIN_WITH_DEPT_TBL AS (
    SELECT /*+ INLINE NO_MERGE */
           D.DNO,
           D.DNAME,
           D.LOC
    FROM TARGET_EMP_INFO C,
         SYSTEM.DEPT D
    WHERE C.DNO = D.DNO
)

SELECT /*+ LEADING(X1 X2) */
       X1.ENAME,
       X1.JOB,
       X1.SAL,
       X2.DNAME,
       X2.LOC
FROM TARGET_EMP_INFO X1,
     TMP_JOIN_WITH_DEPT_TBL X2
WHERE X1.DNO = X2.DNO;

SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY);
```

```
-----------------------------------------------------------------------------------------------------------------------
| Id  | Operation                          | Name                     | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                   |                          |     2 |   110 |     8   (0)| 00:00:01 |
|   1 |  TEMP TABLE TRANSFORMATION         |                          |       |       |            |          |
|   2 |   LOAD AS SELECT (CURSOR DURATION MEMORY)| SYS_TEMP_0FD9D66A7_392E62 |       |       |            |          |
|   3 |    NESTED LOOPS                    |                          |     2 |    72 |     2   (0)| 00:00:01 |
|   4 |     TABLE ACCESS FULL              | EMP                      |    22 |   682 |     2   (0)| 00:00:01 |
|*  5 |     INDEX UNIQUE SCAN              | BEST_EMP_PK              |     1 |     5 |     0   (0)| 00:00:01 |
|*  6 |   HASH JOIN                        |                          |     2 |   110 |     6   (0)| 00:00:01 |
|   7 |    VIEW                            |                          |     2 |    80 |     2   (0)| 00:00:01 |
|   8 |     TABLE ACCESS FULL              | SYS_TEMP_0FD9D66A7_392E62 |     2 |    62 |     2   (0)| 00:00:01 |
|   9 |    VIEW                            |                          |     2 |    30 |     4   (0)| 00:00:01 |
|* 10 |     HASH JOIN                      |                          |     2 |    34 |     4   (0)| 00:00:01 |
|  11 |      VIEW                          |                          |     2 |     6 |     2   (0)| 00:00:01 |
|  12 |       TABLE ACCESS FULL            | SYS_TEMP_0FD9D66A7_392E62 |     2 |    62 |     2   (0)| 00:00:01 |
|  13 |      TABLE ACCESS FULL             | DEPT                     |     7 |    98 |     2   (0)| 00:00:01 |
-----------------------------------------------------------------------------------------------------------------------

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

   5 - access("A"."ENO"="B"."ENO")
   6 - access("X1"."DNO"="X2"."DNO")
  10 - access("C"."DNO"="D"."DNO")
```

- 'TARGET_EMP_INFO'WITH 내부 Operation이 한 번 실행됨
- 해당 Operation에 의해 생성된 결과를 'SYS_TEMP_OFD9D66A7_392E62'라는 임시테이블로 생성
- 추후 'TARGET_EMP_INFO'WITH 참조에 대해선, 'SYS_TEMP_OFD9D66A7_392E62'임시테이블을 조회함

NESTED SUBQ

```
EXPLAIN PLAN FOR
SELECT A.ENAME,
       A.JOB,
       A.SAL,
       B.DNAME
FROM SYSTEM.EMP A,
     SYSTEM.DEPT B
WHERE A.DNO = B.DNO
      AND A.ENO IN (SELECT C.ENO FROM SYSTEM.BEST_EMP C);

SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY);
```

```
-------------------------------------------------------------------------------------
| Id  | Operation           | Name        | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT    |             |     2 |    88 |     4   (0)| 00:00:01 |
|*  1 |  HASH JOIN          |             |     2 |    88 |     4   (0)| 00:00:01 |
|   2 |   NESTED LOOPS      |             |     2 |    72 |     2   (0)| 00:00:01 |
|   3 |    TABLE ACCESS FULL| EMP         |    22 |   682 |     2   (0)| 00:00:01 |
|*  4 |    INDEX UNIQUE SCAN| BEST_EMP_PK |     1 |     5 |     0   (0)| 00:00:01 |
|   5 |   TABLE ACCESS FULL | DEPT        |     7 |    56 |     2   (0)| 00:00:01 |
-------------------------------------------------------------------------------------

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

   1 - access("A"."DNO"="B"."DNO")
   4 - access("A"."ENO"="C"."ENO")
```

- 앞서 봤었던 NESTED SUBQUERY를 활용한 로직임
- 해당 NESTED SUBQ는 CORRELATED SUBQ가 아니기 때문에, 해당 NESTED SUBQ가 MAIN QUERY보다 먼저 실행되야 함

- 결국, 다음과 같은 순서로 해당 로직이 실행될 것이라 예상됨
  1. 'BEST_EMP'TABLE을 SCAN한 후, 해당 테이블에 있는 레코드를 추출
  2. 직전 단계에서 추출된 결과를 ENO에 대한 조건으로 활용하여, 'EMP'TABLE SCAN 실시
  3. 'DEPT'TABLE을 SCAN한 후, 직전 SCAN 결과와 Join 실시

- 실행계획을 확인하면, 위에서 예상한 로직처럼 계획을 세우지 않음
- 'EMP'TABLE, 'DEPT'TABLE, 'NESTED SUBQ' 세 개를 Join하는 형식으로 계획을 세움

- 우리가 예상한 로직처럼 실행계획을 변경하려면, 어떻게 해야 할까?

UNNEST Hint & PUSH_SUBQ Hint

```
EXPLAIN PLAN FOR
SELECT /*+ LEADING(A B) USE_NL(B) */
       A.ENAME,
       A.JOB,
       A.SAL,
       B.DNAME
FROM SYSTEM.EMP A,
     SYSTEM.DEPT B
WHERE A.DNO = B.DNO
      AND A.ENO IN (SELECT /*+ NO_UNNEST PUSH_SUBQ */ C.ENO FROM SYSTEM.BEST_EMP C);

SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY);
```

```
-------------------------------------------------------------------------------------
| Id  | Operation           | Name         | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT    |              |     1 |    39 |     4   (0)| 00:00:01 |
|   1 |  NESTED LOOPS       |              |     1 |    39 |     4   (0)| 00:00:01 |
|*  2 |   TABLE ACCESS FULL | EMP          |     1 |    31 |     2   (0)| 00:00:01 |
|*  3 |    INDEX UNIQUE SCAN| BEST_EMP_PK  |     1 |     5 |     0   (0)| 00:00:01 |
|*  4 |   TABLE ACCESS FULL | DEPT         |     1 |     8 |     2   (0)| 00:00:01 |
-------------------------------------------------------------------------------------

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

   2 - filter( EXISTS (SELECT /*+ PUSH_SUBQ NO_UNNEST */ 0 FROM
              "SYSTEM"."BEST_EMP" "C" WHERE "C"."ENO"=:B1))
   3 - access("C"."ENO"=:B1)
   4 - filter("A"."DNO"="B"."DNO")
```

- UNNEST : NESTED SUBQ와 MAIN Q를 합쳐 Join형태로 실행 계획을 변경하는 것

- NO_UNNEST : NESTED SUBQ UNNESTING 방지
- PUSH_SUBQ : NESTED SUBQ를 우선적으로 실시

- 기본적으로 Optimizer는 NESTED SUBQ에 대해 Join을 수행하도록 실행계획을 세움

11. 실행계획을 참고하여 작성한 쿼리 수정

NESTED SUBQ

```
EXPLAIN PLAN FOR
SELECT A.ENAME,
       A.JOB,
       A.SAL,
       B.DNAME
FROM SYSTEM.EMP A,
     SYSTEM.DEPT B
WHERE A.DNO = B.DNO
      AND A.ENO IN (SELECT C.ENO FROM SYSTEM.BEST_EMP C);

SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY);
```

```
-------------------------------------------------------------------------------------
| Id  | Operation           | Name         | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT    |              |     2 |    88 |     4   (0)| 00:00:01 |
|*  1 |  HASH JOIN          |              |     2 |    88 |     4   (0)| 00:00:01 |
|   2 |   NESTED LOOPS      |              |     2 |    72 |     2   (0)| 00:00:01 |
|   3 |    TABLE ACCESS FULL| EMP          |    22 |   682 |     2   (0)| 00:00:01 |
|*  4 |    INDEX UNIQUE SCAN| BEST_EMP_PK  |     1 |     5 |     0   (0)| 00:00:01 |
|   5 |   TABLE ACCESS FULL | DEPT         |     7 |    56 |     2   (0)| 00:00:01 |
-------------------------------------------------------------------------------------

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

   1 - access("A"."DNO"="B"."DNO")
   4 - access("A"."ENO"="C"."ENO")
```

- 앞서 NESTED SUBQ를 포함한 QUERY를 작성했을 때,
  Optimizer가 해당 NESTED SUBQ를 Join으로 해결하는 것을 확인했음

- 실행계획을 참고하여, NESTED SUBQ를 사용하지 않고 동일한 결과를 산출하도록 기존의 QUERY를 수정하는 방법은?

11. 실행계획을 참고하여 작성한 쿼리 수정

NESTED SUBQ를 Join으로 수정

```
EXPLAIN PLAN FOR
SELECT /*+ LEADING(A B C) */
       B.ENAME,
       B.JOB,
       B.SAL,
       C.DNAME
FROM  SYSTEM.BEST_EMP A,
      SYSTEM.EMP B,
      SYSTEM.DEPT C
WHERE A.ENO = B.ENO
      AND B.DNO = C.DNO;

SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY);
```

```
-------------------------------------------------------------------------------------
| Id  | Operation           | Name         | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT    |              |     2 |    88 |     5   (0)| 00:00:01 |
|*  1 |  HASH JOIN          |              |     2 |    88 |     5   (0)| 00:00:01 |
|*  2 |   HASH JOIN         |              |     2 |    72 |     3   (0)| 00:00:01 |
|   3 |    INDEX FULL SCAN  | BEST_EMP_PK  |     2 |    10 |     1   (0)| 00:00:01 |
|   4 |    TABLE ACCESS FULL| EMP          |    22 |   682 |     2   (0)| 00:00:01 |
|   5 |   TABLE ACCESS FULL | DEPT         |     7 |    56 |     2   (0)| 00:00:01 |
-------------------------------------------------------------------------------------

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

   1 - access("B"."DNO"="C"."DNO")
   2 - access("A"."ENO"="B"."ENO")
```

11. PARALLEL

- ‘병렬 처리’란 하나의 작업을 처리하기 위해 동시에 여러 개의 프로세스를 띄워 분할해서 처리하는 것을 의미함
- 데이터 처리량이 그다지 많지 않거나 작은 규모의 DB 서버를 운용할 때는 굳이 병렬로 처리할 필요가 없음
- 그러나 현재 기업에서 사용하고 있는 대부분의 DB 서버는 용량도 크고 처리하는 데이터 양도 매우 많아 병렬로 수행한다면 매우 큰 성능 향상 효과를 볼 수 있다.
- 오라클에서도 SQL문을 병렬로 처리할 수 있는데, SQL문을 작성해 실행하면 내부적으로 Optimizer가 해당 문장을 처리해 그 결과를 반환함.
- 보통은 SQL문을 실행하면 하나의 프로세스가 이를 처리하는데, 병렬로 처리하면 여러 개의 프로세스가 SQL문을 분석하고 실행해 데이터를 처리한 뒤 그 결과를 반환함

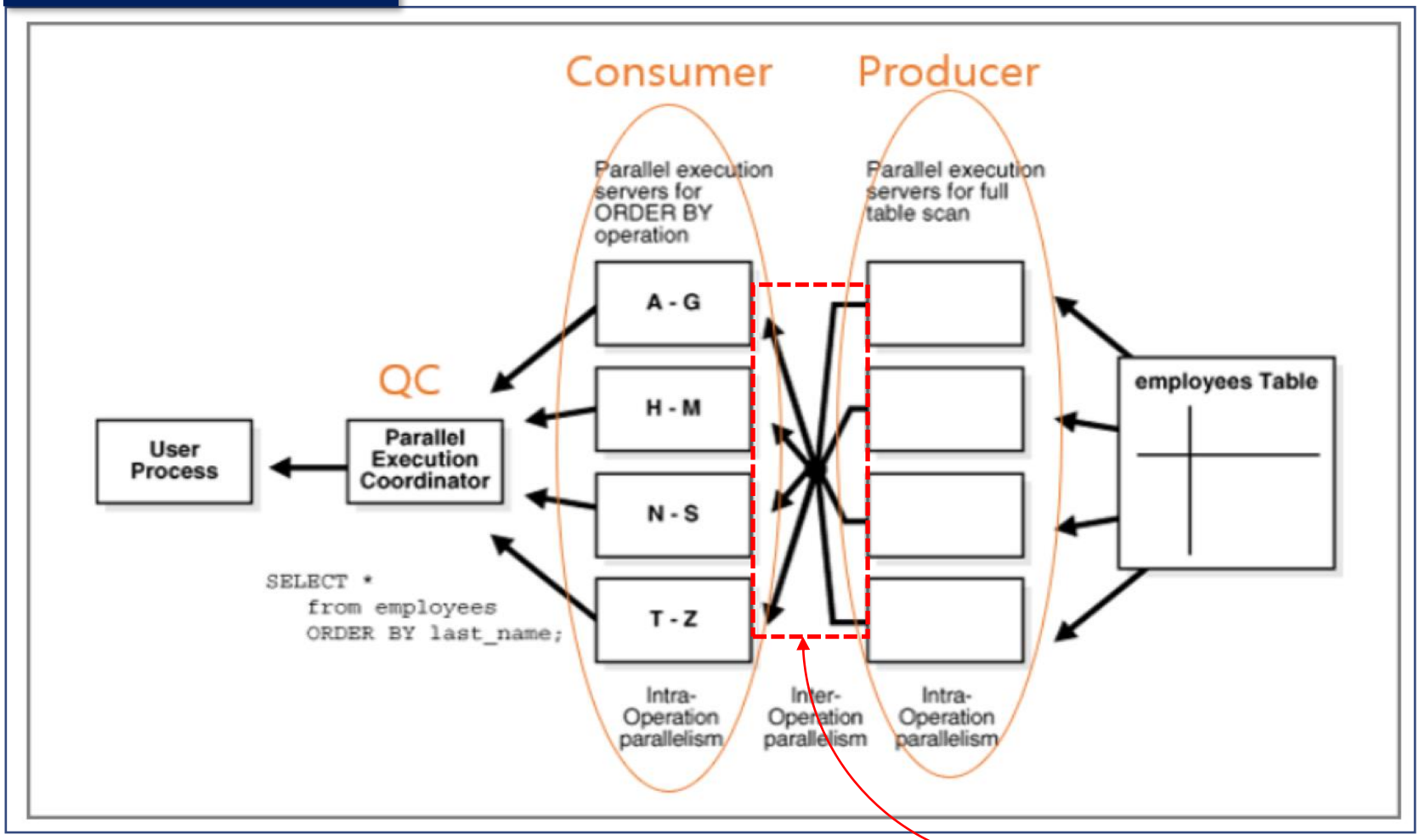
- ‘병렬 처리’에 가담하는 요소

  1. Query Coordinator(QC) : 병렬 SQL문을 발생한 세션
  2. Parallel Process : 실제 작업을 수행하는 Process

- Slave : 특정 작업을 수행하는 Parallel Process 집합
- Slave 종류

  1. Producer Slave : Disk로 부터 데이터를 읽는 Parallel Process 집합
  2. Consumer Slave : 이전 Slave에 의해 만들어진 결과를 수신한 뒤, DML·DDL·Join 등을 수행하는 Parallel Process 집합
     (이전 Slave에 의해 만들어진 결과를 Consumer Slave가 수신할 때, Redistribution이 발생됨)

병렬 처리 도식화


Consumer
Producer
Parallel execution servers for ORDER BY operation
Parallel execution servers for full table scan
A - G
H - M
N - S
T - Z
QC
User Process
Parallel Execution Coordinator
employees Table
SELECT *
from employees
ORDER BY last_name;
Intra-Operation parallelism
Inter-Operation parallelism
Intra-Operation parallelism
Redistribution

11. PARALLEL

- 병렬 처리 수행 과정

1. 병렬 SQL이 시작되면, QC는 사용자가 지정한 병렬도(DOP[Degree of Parallelism], Parallel Process 개수)와 오퍼레이션 종류에 따라 하나 또는 두 개의 Parallel Process Set(Slave)를 할당함
2. QC는 각 Parallel Process에게 작업을 할당하고, 자기자신은 작업을 지시하고 일이 잘 진행되는지 관리·감독하는 역할을 수행
3. 병렬로 처리하도록 사용자가 지시하지 않은 테이블은 QC가 직접 처리함
   (예를 들어, 'EMP'TABLE에는 PARALLEL SCAN을 발생시켰는데 'DEPT'TABLE에는 PARALLEL SCAN을 발생시키지 않았다면, 해당 'DEPT'TABLE에 대한 SCAN은 QC가 실시함)
4. 전체 Parallel Process가 마무리 되면, QC가 각 Parallel Process로 부터의 산출물을 통합하는 작업을 수행함
   (예를 들어, 집계함수[SUM, AVG, MIN, MAX 등]가 사용된 병렬 쿼리를 수행할 때, 각 Parallel Process가 자신의 처리범위 내에서 집계한 값을 QC에 전송하면 QC가 최종 집계 작업을 수행함)
5. 마지막으로, QC는 쿼리의 최종 결과를 사용자에게 전송함

PARALLEL SCAN

```
EXPLAIN PLAN FOR
SELECT /*+ FULL(A) PARALLEL(A 4) */
       *
FROM SYSTEM.SCORE A;

SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY);
```

DOP

Parallel Scan 대상 테이블

```
-------------------------------------------------------------------------------------------------------------
| Id  | Operation            | Name     | Rows  | Bytes | Cost (%CPU)| Time     |    TQ  |IN-OUT| PQ Distrib |
-------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT     |          |  2601 | 39015 |     2   (0)| 00:00:01 |        |      |            |
|   1 |  PX COORDINATOR      |          |       |       |            |          |        |      |            |
|   2 |   PX SEND QC (RANDOM)| :TQ10000 |  2601 | 39015 |     2   (0)| 00:00:01 |  Q1,00 | P->S | QC (RAND)  |
|   3 |    PX BLOCK ITERATOR |          |  2601 | 39015 |     2   (0)| 00:00:01 |  Q1,00 | PCWC |            |
|   4 |     TABLE ACCESS FULL| SCORE    |  2601 | 39015 |     2   (0)| 00:00:01 |  Q1,00 | PCWP |            |
-------------------------------------------------------------------------------------------------------------
```

- 'PX BLOCK ITERATOR', 'PX PARTITION RANGE ITERATOR'등의 Operation이 발생해야 Parallel Process들이 병렬 처리를 수행하는 것임
- 'PX COORDINATOR'Operation이 발생하면, 해당 Operation 위로 등장하는 Operation들은 QC에 의해 수행됨
- 'PX SEND'Operation : 다음 Slave 또는 QC에 결과를 전달하는 Operation
- 'PX RECEIVE'Operation : 이전 Slave로부터 결과를 전달받는 Operation
- 'PX SEND'Operation에 'RANDOM', 'HASH', 'BROADCAST'등의 용어가 붙어있고, 이들은 Redistribution 종류를 의미함
- 각 Redistribution의 동작 방식을 이해하고 싶다면, 아래 영상을 참고할 것

  (Oracle Parallel Process Explanation :

  https://www.youtube.com/watch?v=abEEvZhKk-w)

PARALLEL SCAN & NL Join

```
EXPLAIN PLAN FOR
SELECT /*+ LEADING(A B) USE_NL(B) FULL(A) PARALLEL(A 4) INDEX(B PROFESSOR_PNO_PK) */
       *
FROM SYSTEM.COURSE A,
     SYSTEM.PROFESSOR B
WHERE A.PNO = B.PNO;

SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY);
```

```
-------------------------------------------------------------------------------------------------------------------------------------
| Id  | Operation                      | Name              | Rows  | Bytes | Cost (%CPU)| Time     |    TQ  |IN-OUT| PQ Distrib |
-------------------------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT               |                   |    38 |  2052 |    13   (0)| 00:00:01 |        |      |            |
|   1 |  PX COORDINATOR                |                   |       |       |            |          |        |      |            |
|   2 |   PX SEND QC (RANDOM)          | :TQ10000          |    38 |  2052 |    13   (0)| 00:00:01 | Q1,00  | P->S | QC (RAND)  |
|   3 |    NESTED LOOPS                |                   |    38 |  2052 |    13   (0)| 00:00:01 | Q1,00  | PCWP |            |
|   4 |     NESTED LOOPS               |                   |    39 |  2052 |    13   (0)| 00:00:01 | Q1,00  | PCWP |            |
|   5 |      PX BLOCK ITERATOR         |                   |       |       |            |          | Q1,00  | PCWC |            |
|   6 |       TABLE ACCESS FULL        | COURSE            |    39 |   858 |     2   (0)| 00:00:01 | Q1,00  | PCWP |            |
|*  7 |      INDEX UNIQUE SCAN         | PROFESSOR_PNO_PK  |     1 |       |     0   (0)| 00:00:01 | Q1,00  | PCWP |            |
|   8 |     TABLE ACCESS BY INDEX ROWID| PROFESSOR         |     1 |    32 |     1   (0)| 00:00:01 | Q1,00  | PCWP |            |
-------------------------------------------------------------------------------------------------------------------------------------

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

   7 - access("A"."PNO"="B"."PNO")
```

- 작성된 Hint를 보면, 'COURSE'TABLE에 대해선 PARALLEL SCAN을 실시하고 'PROFESSOR'TABLE에 대해선 INDEX SNAN을 실시함
- 실행계획을 보면, 'COURSE'TABLE SCAN에 대해 'PX BLOCK ITERATOR'Operation이 발생했음
- 그리고, 'PROFESSOR'TABLE SCAN에 대해선 'PX BLOCK ITERATOR'Operation이 발생하지 않음
- 네 개의 Parallel Process가 'COURSE'TABLE을 SCAN했기 때문에, 'PROFESSOR'TABLE과의 NL Join이 네 번 발생함

  (Process 1 – 'PROFESSOR'TABLE, Process 2 – 'PROFESSOR'TABLE, Process 3 – 'PROFESSOR'TABLE, Process 4 – 'PROFESSOR'TABLE)

PARALLEL SCAN & Hash Join

```
EXPLAIN PLAN FOR
SELECT /*+ LEADING(A B) USE_HASH(B) FULL(A) PARALLEL(A 2) FULL(B) PARALLEL(B 2) */
       *
FROM SYSTEM.COURSE A,
     SYSTEM.PROFESSOR B
WHERE A.PNO = B.PNO;

SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY);
```

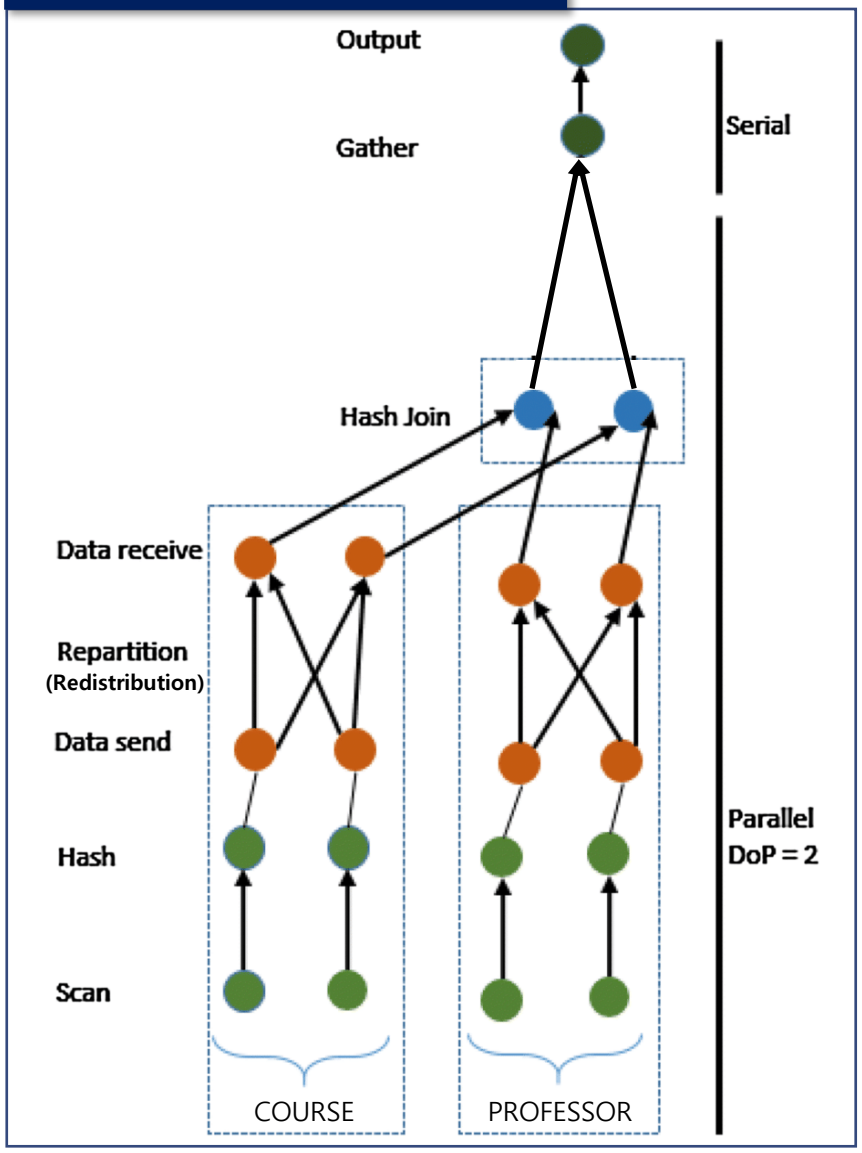
```
-----------------------------------------------------------------------------------------------------------------
| Id  | Operation              | Name      | Rows  | Bytes | Cost (%CPU)| Time     |    TQ  |IN-OUT| PQ Distrib |
-----------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT       |           |    38 |  2052 |     4   (0)| 00:00:01 |        |      |            |
|   1 |  PX COORDINATOR        |           |       |       |            |          |        |      |            |
|   2 |   PX SEND QC (RANDOM)  | :TQ10002  |    38 |  2052 |     4   (0)| 00:00:01 |  Q1,02 | P->S | QC (RAND)  |
|*  3 |    HASH JOIN BUFFERED  |           |    38 |  2052 |     4   (0)| 00:00:01 |  Q1,02 | PCWP |            |
|   4 |     PX RECEIVE         |           |    39 |   858 |     2   (0)| 00:00:01 |  Q1,02 | PCWP |            |
|   5 |      PX SEND HASH      | :TQ10000  |    39 |   858 |     2   (0)| 00:00:01 |  Q1,00 | P->P | HASH       |
|   6 |       PX BLOCK ITERATOR|           |    39 |   858 |     2   (0)| 00:00:01 |  Q1,00 | PCWC |            |
|   7 |        TABLE ACCESS FULL| COURSE   |    39 |   858 |     2   (0)| 00:00:01 |  Q1,00 | PCWP |            |
|   8 |     PX RECEIVE         |           |    24 |   768 |     2   (0)| 00:00:01 |  Q1,02 | PCWP |            |
|   9 |      PX SEND HASH      | :TQ10001  |    24 |   768 |     2   (0)| 00:00:01 |  Q1,01 | P->P | HASH       |
|  10 |       PX BLOCK ITERATOR|           |    24 |   768 |     2   (0)| 00:00:01 |  Q1,01 | PCWC |            |
|  11 |        TABLE ACCESS FULL| PROFESSOR|    24 |   768 |     2   (0)| 00:00:01 |  Q1,01 | PCWP |            |
-----------------------------------------------------------------------------------------------------------------

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

   3 - access("A"."PNO"="B"."PNO")
```

- Hint를 보면, 두 테이블에 대해 PARALLEL SCAN 실시
- 실행계획을 보면, 두 테이블 SCAN에 대해 'PX BLOCK ITERATOR'Operation이 발생함
- 'PX SEND'Operation과 'PX RECEIVE'Operation 둘 다 발생했기 때문에, Producer Slave와 Consumer Slave가 구성된 것을 확인할 수 있음
- 'PX SEND HASH'Operation을 통해, HASH Redistribution이 발생된 것을 확인할 수 있음

PARALLEL SCAN & Hash Join 도식화



- COURSE TABLE
  1. 'Hash'단계에서 Producer Slave가 Scan한 결과에 Hash Function 적용
  2. 'Data send'단계에서 Producer Slave가 Hash Function을 통과한 레코드를 Redistribution 실시
  3. 'Data receive'단계에서 Consumer Slave가 Redistribution된 레코드를 수신
     (ex : Consumer Slave의 첫 번째 Parallel Process는 hash key가 '0 ~ 4'인 레코드를 수신하고, 두 번째 Parallel Process는 hash key가 '5 ~ 9'인 레코드를 수신)
  4. Consumer Slave내 Parallel Process들은 각자 보유하고 있는 레코드를 기반으로 Hash Table을 생성
     (첫 번째 Parallel Process - Hash Table1 생성
     두 번째 Parallel Process2 – Hash Table2 생성)

- PROFESSOR TABLE
  1. COURSE TABLE이 거친 '1 ~ 3단계'를 수행
  2. Consumer Slave내 첫 번째 Parallel Process가 보유한 레코드를 Hash Table1에 송신하여 Join 실시하고, 두 번째 Parallel Process가 보유한 레코드를 Hash Table2에 송신하여 Join 실시

- Join결과를 QC에게 송신 후, 결과 반환