개발자를 위한 Advanced SQL & SQL Tuning

Edition 1.0
October 2016

Author : Chongha Ryu
chongharyu@naver.com

목차

1. Introduction to the Optimizer

SQL 명령문은 결과만을 요청하는 명령문이다. 내부적으로 처리 절차(과정)에 대한 부분은 Oracle Database Server가 담당하게 되어 있으며 문장의 성능을 비교할 경우에는 실행 계획을 확인해야 한다.

1.1. SQL 문 처리 과정

1. 구문 및 의미(객체 및 권한) 확인

- 실행 가능한 문장 여부 확인

  : Dictionary Data는 Dictionary Cache, 또는 Recursive Call 사용

- Private SQL Area 에 저장 (Cursor)

2. 동일 문장 확인 (Library Cache 내에 존재 여부)

3. 동일 문장이 발견되면 Soft Parse, 발견되지 않으면 Hard Parse

4. Hard Parse 단계

- Query Transformer
- Estimator (실행 계획의 Cost 산출, Dictionary Data 이용)
- Plan Generator
- 생성된 실행 계획은 Library Cache에 저장 (공유 커서)

5. 명령문 확인 : FETCH 단계 준비 (실행 결과를 저장할 수 있는 영역 구성)

6. Bind : 호출 환경에서 선언된 변수에 값 입력

7. Parallelize : 병렬 작업 배분 (병렬 실행 시)

8. Execute : 실행 (실행 계획 사용)

9. FETCH : Query 결과 반환 (SELECT 문장일 경우에만 수행)

모든 컬럼에 인덱스가 존재할 때 사용된 인덱스는?

```
[orcl:adsql]$ sqlplus user01/oracle

# SAL 컬럼에 만족하는 행이 더 적다
SQL> SELECT (SELECT COUNT(*) FROM emp WHERE deptno = 10) AS cnt_deptno,
            (SELECT COUNT(*) FROM emp WHERE sal > 4000 ) AS cnt_sal
     FROM dual ;
CNT_DEPTNO    CNT_SAL
---------- ----------
         3          1

SQL> SELECT empno, ename, sal, deptno
     FROM emp
     WHERE deptno = 10
       AND sal > 4000 ;
     EMPNO ENAME            SAL     DEPTNO
---------- ---------- ---------- ----------
      7839 KING             5000         10

SQL> SELECT /*+ rule */ empno, ename, sal, deptno
     FROM emp
     WHERE deptno = 10
       AND sal    > 4000 ;
-----------------------------------------------------
| Id  | Operation                   | Name          |
-----------------------------------------------------
|   0 | SELECT STATEMENT            |               |
|*  1 |  TABLE ACCESS BY INDEX ROWID| EMP           |
|*  2 |   INDEX RANGE SCAN          | EMP_DEPTNO_IX |
-----------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   1 - filter("SAL">4000)
   2 - access("DEPTNO"=10)
```

RBO는 각 조건에 만족하는 행의 개수를 예상하지 않고 조건식에 사용되는 컬럼에 인덱스 존재 유무, 또는 비교 연산자의 종류 중 보다 범위가 작을 수 있는 연산식을 우선적으로 처리하는 규칙을 가지고 있다. 이는 교통 흐름 상태는 모르는 상태에서 원하는 목적지를 최소 거리를 기준으로 길 찾기를 하는 것과 별반 다르지 않다. 최상의 실행 계획을 만들기 위해서는 현재의 데이터의 상태(Optimizer Statistics)를 통해서 가장 적은 비용으로 처리가 가능한 실행 계획을 생성하는 것이 최적의 실행 계획이라 할 수 있다. (Oracle Databse 10g부터는 CBO가 기본적으로 사용된다.)

```
SQL> SELECT empno, ename, sal, deptno
     FROM emp
     WHERE deptno = 10
       AND sal    > 4000 ;
-----------------------------------------------------------------------------------------------
| Id  | Operation                   | Name       | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT            |            |     1 |    17 |     2   (0)| 00:00:01 |
|*  1 |  TABLE ACCESS BY INDEX ROWID| EMP        |     1 |    17 |     2   (0)| 00:00:01 |
|*  2 |   INDEX RANGE SCAN          | EMP_SAL_IX |     1 |       |     1   (0)| 00:00:01 |
-----------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   1 - filter("DEPTNO"=10)
   2 - access("SAL">4000)

SQL> SELECT column_name, num_distinct, low_value, high_value,
            density, num_nulls
     FROM user_tab_columns
     WHERE table_name = 'EMP'
     ORDER BY column_id ;
COLUMN_NAME     NUM_DISTINCT LOW_VALUE           HIGH_VALUE              DENSITY  NUM_NULLS
--------------- ------------ ------------------- ------------------- ---------- ----------
EMPNO                   14 C24A46              C25023              .071428571          0
ENAME                   14 4144414D53          57415244            .071428571          0
JOB                      5 414E414C595354      53414C45534D414E    .035714286          0
MGR                      6 C24C43              C25003              .166666667          1
HIREDATE                13 77B40C11010101      77BB0517010101      .076923077          0
SAL                     12 C209                C233                .035714286          0
COMM                     4 80                  C20F                       .25         10
DEPTNO                   3 C10B                C11F                .035714286          0
```

옵티마이저 통계(Optimizer Statistics)는 DBMS_STATS 패키지 또는 ANALYZE 명령문을 통해 수집 가능하며 수집된 통계 정보는 여러 Dictionary View 를 통해 그 내용을 확인할 수 있다. 이러한 통계는 실행 계획을 생성할 때 매우 중요한 지표로 사용되므로 현재 객체의 상태를 잘 반영하도록 수집되어 있어야 한다.

```
SQL> execute DBMS_STATS.SET_COLUMN_STATS(USER,'EMP','SAL', distcnt => 3)
PL/SQL procedure successfully completed.

SQL> execute DBMS_STATS.SET_COLUMN_STATS(USER,'EMP','DEPTNO', distcnt => 12)
PL/SQL procedure successfully completed.

SQL> SELECT column_name, num_distinct, low_value, high_value,
            density, num_nulls
     FROM user_tab_columns
     WHERE table_name = 'EMP'
     ORDER BY column_id ;
COLUMN_NAME     NUM_DISTINCT LOW_VALUE           HIGH_VALUE              DENSITY  NUM_NULLS
--------------- ------------ ------------------- ------------------- ---------- ----------
EMPNO                     14 C24A46              C25023              .071428571          0
ENAME                     14 4144414D53          57415244            .071428571          0
JOB                        5 414E414C595354      53414C45534D414E    .035714286          0
MGR                        6 C24C43              C25003              .166666667          1
HIREDATE                  13 77B40C11010101      77BB0517010101      .076923077          0
SAL                        3 C209                C233                .333333333          0
COMM                       4 80                  C20F                       .25         10
DEPTNO                    12 C10B                C11F                .083333333          0
```

옵티마이저 통계를 임의로 수정 후 동일한 SELECT 문장 실행 시 어떤 인덱스를 사용하는가?

```
SQL> SELECT empno, ename, sal, deptno
     FROM emp
     WHERE deptno = 10
       AND sal    > 4000 ;
-------------------------------------------------------------------------------------------
| Id  | Operation                   | Name          | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT            |               |     1 |    17 |     2   (0)| 00:00:01 |
|*  1 |  TABLE ACCESS BY INDEX ROWID| EMP           |     1 |    17 |     2   (0)| 00:00:01 |
|*  2 |   INDEX RANGE SCAN          | EMP_DEPTNO_IX |     1 |       |     1   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   1 - filter("SAL">4000)
   2 - access("DEPTNO"=10)
```

DEPTNO 컬럼의 구분 값이 더 많은 것으로 인식하고 있기 때문에 SAL 컬럼의 인덱스 대신 DEPTNO 컬럼의 인덱스를 사용하는 실행 계획이 선택되었다.

```
SQL> execute DBMS_STATS.GATHER_TABLE_STATS(USER,'EMP')
PL/SQL procedure successfully completed.

SQL> SELECT empno, ename, sal, deptno
     FROM emp
     WHERE deptno = 10
       AND sal    > 4000 ;
-------------------------------------------------------------------------------------------
| Id  | Operation                   | Name       | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT            |            |     1 |    17 |     2   (0)| 00:00:01 |
|*  1 |  TABLE ACCESS BY INDEX ROWID| EMP        |     1 |    17 |     2   (0)| 00:00:01 |
|*  2 |   INDEX RANGE SCAN          | EMP_SAL_IX |     1 |       |     1   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   1 - filter("DEPTNO"=10)
   2 - access("SAL">4000)
```

2. Optimizer Statistics

옵티마이저 통계

- 현재 데이터 및 시스템의 상태 정보 (Object 통계와 System 통계로 구분)
- cost 계산의 중요한 지표로 사용 됨
- 정적이며 데이터의 상태가 변경될 때마다 새로 수집 필요
- 비 대칭적인 데이터 존재 시 Histogram 필요 (데이터의 분포도)

Rule Based Optimizer (RBO)는 옵티마이저 통계를 사용하지 않는다.

```
SQL> SELECT channel_id, COUNT(*)
     FROM sales
     GROUP BY channel_id ;
CHANNEL_ID   COUNT(*)
---------- ----------
         2     258025
         4     118416
         3     540328
         9       2074

SQL> SELECT /*+ rule */ prod_id, COUNT(*)
     FROM sales
     WHERE channel_id = 9
     GROUP BY prod_id ;
------------------------------------------------------------
| Id  | Operation                    | Name             |
------------------------------------------------------------
|   0 | SELECT STATEMENT             |                  |
|   1 |  SORT GROUP BY               |                  |
|   2 |   TABLE ACCESS BY INDEX ROWID| SALES            |
|*  3 |    INDEX RANGE SCAN          | SALES_CHANNEL_IX |
------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   3 - access("CHANNEL_ID"=9)
```

```
SQL> SELECT /*+ rule */ prod_id, COUNT(*)
     FROM sales
     WHERE channel_id = 3
     GROUP BY prod_id ;
-----------------------------------------------------
| Id  | Operation                    | Name            |
-----------------------------------------------------
|   0 | SELECT STATEMENT             |                 |
|   1 |  SORT GROUP BY               |                 |
|   2 |   TABLE ACCESS BY INDEX ROWID| SALES           |
|*  3 |    INDEX RANGE SCAN          | SALES_CHANNEL_IX |
-----------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   3 - access("CHANNEL_ID"=3)
```

> 확인 사항

RBO는 인덱스가 존재하는 컬럼이 조건식에 사용되면 실제 데이터의 양과 상관없이 항상 인덱스 사용한다. 즉, RBO는 통계를 활용하지 않는다.

Cost Based Optimizer (CBO) 사용

```
SQL> SELECT prod_id, COUNT(*)
     FROM sales
     WHERE channel_id = 3
     GROUP BY prod_id ;
-------------------------------------------------------------------------------
| Id  | Operation          | Name  | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |       |    72 |   504 |  1238   (2)| 00:00:15 |
|   1 |  HASH GROUP BY     |       |    72 |   504 |  1238   (2)| 00:00:15 |
|*  2 |   TABLE ACCESS FULL| SALES |   229K|  1570K|  1231   (1)| 00:00:15 |
-------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   2 - filter("CHANNEL_ID"=3)
```

```
SQL> SELECT prod_id, COUNT(*)
     FROM sales
     WHERE channel_id = 9
     GROUP BY prod_id ;
-------------------------------------------------------------------------------
| Id  | Operation          | Name  | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |       |    72 |   504 |  1238   (2)| 00:00:15 |
|   1 |  HASH GROUP BY     |       |    72 |   504 |  1238   (2)| 00:00:15 |
|*  2 |   TABLE ACCESS FULL| SALES |   229K|  1570K|  1231   (1)| 00:00:15 |
-------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   2 - filter("CHANNEL_ID"=9)
```

> 확인 사항

CBO 가 사용되었으므로 예상 rows, bytes, cost 등의 값이 함께 출력되었다. 하지만 두 조건식 모두 예상되는 rows 가 229K이므로 인덱스를 사용하지 않고 Full Table Scan 이 사용되었다. 229K의 예상 행의 개수는 어떻게 계산되었을까?

```
SQL> SELECT table_name, num_rows, blocks
     FROM user_tables
     WHERE table_name = 'SALES' ;
TABLE_NAME             NUM_ROWS     BLOCKS
-------------------- ---------- ----------
SALES                    918843       4511

SQL> save tab replace
Wrote file tab.sql
```

```
SQL> SELECT column_name, num_distinct, density, num_buckets, histogram
     FROM user_tab_col_statistics
     WHERE table_name = 'SALES' ;
COLUMN_NAME                    NUM_DISTINCT    DENSITY NUM_BUCKETS HISTOGRAM
------------------------------ ------------ ---------- ----------- ---------------
PROD_ID                                  72 .013888889           1 NONE
CUST_ID                                7059 .000141663           1 NONE
TIME_ID                                1460 .000684932           1 NONE
CHANNEL_ID                                4         .25           1 NONE
PROMO_ID                                  4         .25           1 NONE
QUANTITY_SOLD                             1           1           1 NONE
AMOUNT_SOLD                            3586 .000278862           1 NONE

SQL> save col replace
Wrote file col.sql
```

통계 정보를 수집할 때 히스토그램을 수집하지 않으면 기본적으로 모든 데이터는 균등하게 분포하는 것으로 간주된다. 때문에 위의 결과를 보면 CHANNEL_ID의 구분 값은 4개이며 density는 25%를 가지고 있다. 이러한 통계 정보를 활용하여 예상되는 행의 개수는 다음과 같이 계산되었다.

```
SQL> SELECT ROUND(1/4 * 918843) FROM dual ;
ROUND(1/4*918843)
-----------------
           229711
```

비 대칭적인 데이터가 존재하는 경우에는 보다 정확한 분포도를 계산할 수 있도록 히스토그램을 수집한다.

```
SQL> execute DBMS_STATS.GATHER_TABLE_STATS(USER,'SALES', -
                                    method_opt=> 'for all columns size auto')
PL/SQL procedure successfully completed.
```

```
SQL> @col
COLUMN_NAME                    NUM_DISTINCT    DENSITY NUM_BUCKETS HISTOGRAM
------------------------------ ------------ ---------- ----------- ---------------
PROD_ID                                  72 5.3936E-07          72 FREQUENCY
CUST_ID                                7059 .000141663           1 NONE
TIME_ID                                1460 .000684932           1 NONE
CHANNEL_ID                                4 5.3936E-07           4 FREQUENCY
PROMO_ID                                  4         .25           1 NONE
QUANTITY_SOLD                             1           1           1 NONE
AMOUNT_SOLD                            3586 .000278862           1 NONE

SQL> SELECT column_name, endpoint_number, endpoint_value,
            endpoint_number - lag(endpoint_number,1,0)
                              OVER(ORDER BY endpoint_number) AS Diff
     FROM user_histograms
     WHERE table_name = 'SALES'
       AND column_name = 'CHANNEL_ID'
     ORDER BY endpoint_number ;
COLUMN_NAME          ENDPOINT_NUMBER ENDPOINT_VALUE       DIFF
-------------------- --------------- -------------- ----------
CHANNEL_ID                      1573              2       1573
CHANNEL_ID                      4806              3       3233
CHANNEL_ID                      5532              4        726
CHANNEL_ID                      5549              9         17
```

```
SQL> SELECT prod_id, COUNT(*)
     FROM sales
     WHERE channel_id = 3
     GROUP BY prod_id ;
-------------------------------------------------------------------------------
| Id  | Operation          | Name  | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |       |    72 |   504 |  1247   (2)| 00:00:15 |
|   1 |  HASH GROUP BY     |       |    72 |   504 |  1247   (2)| 00:00:15 |
|*  2 |   TABLE ACCESS FULL| SALES |   535K|  3659K|  1231   (1)| 00:00:15 |
-------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   2 - filter("CHANNEL_ID"=3)
```

```
SQL> SELECT prod_id, COUNT(*)
     FROM sales
     WHERE channel_id = 9
     GROUP BY prod_id ;
```

```
-------------------------------------------------------------------------------------------------
| Id  | Operation                    | Name             | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |                  |    72 |   504 |    46   (3)| 00:00:01 |
|   1 |  HASH GROUP BY               |                  |    72 |   504 |    46   (3)| 00:00:01 |
|   2 |   TABLE ACCESS BY INDEX ROWID| SALES            |  2732 | 19124 |    45   (0)| 00:00:01 |
|*  3 |    INDEX RANGE SCAN          | SALES_CHANNEL_IX |  2732 |       |     8   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   3 - access("CHANNEL_ID"=9)
```

> 결론

모든 컬럼에 히스토그램이 필요한 것은 아니다. 위의 예제처럼 인덱스의 사용 선택이 성능에 영향을 미치는 경우가 존재한다면 필요한 컬럼만을 대상으로 히스토그램을 생성하면 된다. 히스토그램이 존재하면 보다 정확한 예측 행을 계산할 수 있고 최적화된 실행 계획을 만들 수 있게 된다.

3. 실행 계획 (Execution Plans)

- SQL을 실행하기 위한 처리 순서도
- Hard Parsing 도중 생성되며, Shared Pool의 Library Cache에 저장
- 문장 실행 전 예측되는 실행 계획은 PLAN_TABLE에 저장

3.1. EXPLAIN PLAN

- 실제 사용된 실행 계획이 아닌 예상 실행 계획을 PLAN_TABLE에 저장 함
- 사용자 정의 PLAN_TABLE 생성 가능 ($ORACLE_HOME/rdbms/admin/utlxplan.sql)
- DBMS_XPLAN.DISPLAY 이용하여 결과 확인

```
SQL> EXPLAIN PLAN FOR
     SELECT * FROM emp
     WHERE job = 'CLERK' ;
Explained.
SQL> SELECT * FROM table(dbms_xplan.display) ;
-------------------------------------------------------------------------------------------
| Id  | Operation                   | Name       | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT            |            |     3 |   114 |     2   (0)| 00:00:01 |
|   1 |  TABLE ACCESS BY INDEX ROWID| EMP        |     3 |   114 |     2   (0)| 00:00:01 |
|* 2  |   INDEX RANGE SCAN          | EMP_JOB_IX |     3 |       |     1   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   2 - access("JOB"='CLERK')
```

3.2. AUTOTRACE

- SQL*Plus 및 SQL Developer 기능
- PLAN_TABLE 테이블 및 PLUSTRACE 롤 필요 (통계정보 출력 시)
- 문장의 예상 실행 계획 및 실행 결과, 실행 통계 출력 가능
- 실제 사용된 실행 계획과 다를 수 있음

일반 계정에서 실행 통계를 확인 하기 위해서는 PLUSTRACE 롤이 필요함.

```
SQL> conn user01/oracle
SQL> SET AUTOTRACE ON
SP2-0618: Cannot find the Session Identifier.  Check PLUSTRACE role is enabled
SP2-0611: Error enabling STATISTICS report

SQL> conn / as sysdba
SQL> @$ORACLE_HOME/sqlplus/admin/plustrce.sql

SQL> GRANT plustrace TO user01, user02 ;
SQL> CONN user01/oracle

SQL> SET AUTOTRACE ON
SQL> SELECT * FROM dept WHERE deptno = 30 ;
    DEPTNO DNAME          LOC
---------- -------------- -------------
        30 SALES          CHICAGO

------------------------------------------------------------------------------------------
| Id  | Operation                   | Name    | Rows  | Bytes | Cost (%CPU)| Time     |
------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT            |         |     1 |    20 |     1   (0)| 00:00:01 |
|   1 |  TABLE ACCESS BY INDEX ROWID| DEPT    |     1 |    20 |     1   (0)| 00:00:01 |
|*  2 |   INDEX UNIQUE SCAN         | PK_DEPT |     1 |       |     0   (0)| 00:00:01 |
------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   2 - access("DEPTNO"=30)
```

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

AUTOTRACE를 이용하여 문장 실행 결과, 실행 계획, 실행 통계를 확인한다. 단, 실행 계획은 실제 사용된 실행 계획이 아닐 수 있으며 실행 통계는 전체의 요약 정보이므로 각 단계별 실제 사용된 실행 통계를 확인할 수는 없다.

AUTOTRACE 의 추가 옵션

```
SQL> SET AUTOTRACE ON EXPLAIN
SQL> /

SQL> SET AUTOTRACE ON STATISTICS
SQL> /

SQL> SET AUTOTRACE TRACEONLY
SQL> /
------------------------------------------------------------------------------------------
| Id  | Operation                    | Name    | Rows  | Bytes | Cost (%CPU)| Time     |
------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |         |     1 |    20 |     1   (0)| 00:00:01 |
|   1 |  TABLE ACCESS BY INDEX ROWID| DEPT    |     1 |    20 |     1   (0)| 00:00:01 |
|*  2 |   INDEX UNIQUE SCAN          | PK_DEPT |     1 |       |     0   (0)| 00:00:01 |
------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   2 - access("DEPTNO"=30)
```

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

SQL> SET AUTOTRACE TRACEONLY EXPLAIN
SQL> /

SQL> SET AUTOTRACE TRACEONLY STATISTICS
SQL> /

SQL> SET AUTOTRACE OFF
```

3.3. Library Cache

- 최근에 실행된 실행 계획을 검사할 때 VSQL, VSQL_PLAN 뷰를 검색하여 확인한다.
- V$SQL_PLAN_STATISTICS_ALL 뷰를 통해 실행 통계 검색 가능
- DBMS_XPLAN.DISPLAY_CURSOR 이용하여 결과 확인

```
SQL> conn user01/oracle
SQL> SELECT * FROM user01.dept WHERE deptno = 10;
    DEPTNO DNAME          LOC
---------- -------------- -------------
        10 ACCOUNTING     NEW YORK

SQL> SELECT * FROM table(dbms_xplan.display_cursor) ;
PLAN_TABLE_OUTPUT
----------------------------------------
User has no SELECT privilege on V$SESSION                              -- ERROR

SQL> conn system/oracle
SQL> SELECT * FROM user01.dept WHERE deptno = 10;
    DEPTNO DNAME          LOC
---------- -------------- -------------
        10 ACCOUNTING     NEW YORK

SQL> SELECT * FROM table(dbms_xplan.display_cursor) ;
PLAN_TABLE_OUTPUT
---------------------------------
SQL_ID  50ch2pbjnvdau, child number 0
-------------------------------------
SELECT * FROM user01.dept WHERE deptno = 10

Plan hash value: 2852011669
-------------------------------------------------------------------------------------------
| Id  | Operation                   | Name    | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT            |         |       |       |     1 (100)|          |
|   1 |  TABLE ACCESS BY INDEX ROWID| DEPT    |     1 |    20 |     1   (0)| 00:00:01 |
|*  2 |   INDEX UNIQUE SCAN         | PK_DEPT |     1 |       |     0   (0)|          |
-------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   2 - access("DEPTNO"=10)
```

VSQL, VSQL_PLAN 뷰는 DBMS_XPLAN.DISPLAY_CURSOR를 이용하여 쉽게 검색할 수 있다. 단, 실제 사용된 SQL의 실행 계획 정보를 검색하려면 V$SESSION, V$SQL과 같은 Dynamic Performance View에 대한 접근 권한이 있어야 한다. 때문에 해당 권한이 없는 유저가 DBMS_XPLAN.DISPLAY_CURSOR 함수를 사용하면 에러가 발생한다.

일반적으로 개발자에게는 제한된 권한만을 부여하게 된다. 때문에 개발자가 가지고 있는 권한 내에서 문장의 성능을 측정하고 테스트하기는 매우 어렵다. 문장의 성능을 측정할 수 있도록, 그리고 튜닝을 수행할 수 있도록 하려면 최소한의 추가 권한을 획득해야만 제대로 된 튜닝을 진행할 수 있을 것이다.

- **필요 권한**

```
SQL> conn / as sysdba
SQL> CREATE ROLE sqlt ;
SQL> GRANT SELECT ON V_$SESSION TO sqlt ;
SQL> GRANT SELECT ON V_$SQL TO sqlt ;
SQL> GRANT SELECT ON V_$SQL_PLAN TO sqlt ;
SQL> GRANT SELECT ON V_$SQL_PLAN_STATISTICS_ALL TO sqlt ;
SQL> GRANT sqlt TO user01 ;

SQL> conn user01/oracle
SQL> SELECT * FROM user01.dept WHERE deptno = 10;
SQL> SELECT * FROM table(dbms_xplan.display_cursor) ;
------------------------------------------------------------------------------------------
| Id  | Operation                   | Name    | Rows  | Bytes | Cost (%CPU)| Time     |
------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT            |         |       |       |     1 (100)|          |
|   1 |  TABLE ACCESS BY INDEX ROWID| DEPT    |     1 |    20 |     1   (0)| 00:00:01 |
|*  2 |   INDEX UNIQUE SCAN         | PK_DEPT |     1 |       |     0   (0)|          |
------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   2 - access("DEPTNO"=10)
```

• **생성자 권한으로 실행되는 패키지 생성**

실제 사용된 실행 계획은 DBMS_XPLAN 패키지를 이용하여 출력할 수 있다. 이때 사용되는 DBMS_XPLAN은 호출자 권한으로 실행되는 패키지이다. 때문에 실행 계획 등을 확인하기 위해 V$ 뷰에 대한 접근 권한이 없으면 DBMS_XPLAN 패키지를 통해서도 조회가 불가능하다. 이러한 DBMS_XPLAN 패키지를 생성자 권한으로 실행할 수 있도록 한다면 직접적인 V$ 뷰에 대한 접근 권한이 없어도 실행 계획은 조회가 가능하다. 다음의 스크립트는 저자가 생성자 권한으로 DBMS_XPLAN 패키지와 동일 작업을 수행할 수 있도록 수정한 MYXPLAN 패키지를 생성할 수 있는 스크립트이다. 해당 스크립트를 실행하여 MYXPLAN 패키지를 생성하고, V$ 뷰에 접근 권한이 없는 상태에서도 실행 계획의 확인 방법을 테스트한다.

```
SQL> conn / as sysdba
SQL> REVOKE sqlt FROM user01 ;
SQL> @/home/oracle/adsql/myxplan
Package created.
Package body created.
...

SQL> conn user01/oracle
SQL> SELECT * FROM v$sql_plan ;
ERROR at line 1:
ORA-00942: table or view does not exist

SQL> SELECT * FROM dept ;
    DEPTNO DNAME          LOC
---------- -------------- -------------
        10 ACCOUNTING     NEW YORK
        20 RESEARCH       DALLAS
        30 SALES          CHICAGO
        40 OPERATIONS     BOSTON
```

```
SQL> SELECT * FROM table(myxplan.display_cursor) ;
PLAN_TABLE_OUTPUT
------------------------------------
SQL_ID  2rbmqdmt9aj4m, child number 0
------------------------------------
SELECT * FROM dept

Plan hash value: 3383998547
-------------------------------------------------------------------------
| Id  | Operation         | Name | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------
|   0 | SELECT STATEMENT  |      |       |       |     3 (100)|          |
|   1 |  TABLE ACCESS FULL| DEPT |     4 |    80 |     3   (0)| 00:00:01 |
-------------------------------------------------------------------------


SQL> SELECT * FROM table(dbms_xplan.display_cursor) ;
PLAN_TABLE_OUTPUT
----------------------------------------
User has no SELECT privilege on V$SESSION
```

SQLT 롤을 회수 했기 때문에 V$ 뷰에 대한 접근 권한은 없다. 때문에 DBMS_XPLAN 패키지를 사용하는 것도 불가능하다. 하지만 MYXPLAN 패키지는 뷰에 접근 권한이 없어도, 해당 패키지를 생성한 SYS 유저의 소유권으로 실행되므로 실제 사용된 실행 계획을 확인할 수 있게 된다. 해당 패키지는 PUBLIC SYNONYM을 통해 DB의 모든 유저가 사용 가능하도록 필요한 명령어를 정의했으므로 필요시 특정 유저만이 사용 가능하도록 수정하여 사용한다. (해당 패키지는 반드시 SYS 유저가 생성해야 한다.)

• **실행 통계 확인**

```
SQL> SELECT * FROM dept WHERE deptno = 10 ;
    DEPTNO DNAME          LOC
---------- -------------- -------------
        10 ACCOUNTING     NEW YORK

SQL> SELECT * FROM table(myxplan.display_cursor(null, null, 'IOSTATS LAST')) ;
------------------------------------------------------------
| Id  | Operation                   | Name    | E-Rows |
------------------------------------------------------------
|   0 | SELECT STATEMENT            |         |        |
|   1 |  TABLE ACCESS BY INDEX ROWID| DEPT    |      1 |
|*  2 |   INDEX UNIQUE SCAN         | PK_DEPT |      1 |
------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   2 - access("DEPTNO"=10)

Note
-----
   - Warning: basic plan statistics not available. These are only collected when:
       * hint 'gather_plan_statistics' is used for the statement or
       * parameter 'statistics_level' is set to 'ALL', at session or system level
```

예상 실행 계획이 아닌 실제 사용된 실행 계획은 각 단계별 사용된 리소스 정보를 추가로 확인할 수 있다. 다만, 모든 문장이 실행될 때 실행 통계를 항상 수집하는 것은 아니다. 때문에 위의 노트와 같이 실행 통계를 수행하기 위해서는 추가적인 설정이 필요하다.

- 문장 레벨의 실행 통계 수집 (힌트 사용)

```
SQL> SELECT /*+ gather_plan_statistics */ * FROM dept WHERE deptno = 10 ;
...
SQL> SELECT * FROM table(myxplan.display_cursor(null, null, 'ALLSTATS LAST')) ;
------------------------------------------------------------------------------------------------------
| Id  | Operation                   | Name    | Starts | E-Rows | A-Rows |   A-Time   | Buffers |
------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT            |         |      1 |        |      1 |00:00:00.01 |       2 |
|   1 |  TABLE ACCESS BY INDEX ROWID| DEPT    |      1 |      1 |      1 |00:00:00.01 |       2 |
|*  2 |   INDEX UNIQUE SCAN         | PK_DEPT |      1 |      1 |      1 |00:00:00.01 |       1 |
------------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   2 - access("DEPTNO"=10)
```

- 세션 레벨의 실행 통계 수집 (파라미터 수정, ALTER SESSION 권한 필요)

```
SQL> ALTER SESSION SET statistics_level = all ;


SQL> SELECT * FROM dept WHERE deptno = 10 ;
...
SQL> SELECT * FROM table(myxplan.display_cursor(null, null, 'ALLSTATS LAST')) ;
------------------------------------------------------------------------------------------------------
| Id  | Operation                   | Name    | Starts | E-Rows | A-Rows |   A-Time   | Buffers |
------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT            |         |      1 |        |      1 |00:00:00.01 |       2 |
|   1 |  TABLE ACCESS BY INDEX ROWID| DEPT    |      1 |      1 |      1 |00:00:00.01 |       2 |
|*  2 |   INDEX UNIQUE SCAN         | PK_DEPT |      1 |      1 |      1 |00:00:00.01 |       1 |
------------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   2 - access("DEPTNO"=10)
```

- 시스템 레벨의 실행 통계 수집 (파라미터 수정, ALTER SYSTEM 권한 필요)

```
SQL> conn / as sysdba
SQL> ALTER SYSTEM SET statistics_level = all ;

SQL> conn user01/oracle
SQL> SELECT * FROM dept WHERE deptno = 10 ;
...
SQL> SELECT * FROM table(myxplan.display_cursor(null, null, 'ALLSTATS LAST')) ;
------------------------------------------------------------------------------------------------------
| Id  | Operation                   | Name    | Starts | E-Rows | A-Rows |   A-Time   | Buffers |
------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT            |         |      1 |        |      1 |00:00:00.01 |       2 |
|   1 |  TABLE ACCESS BY INDEX ROWID| DEPT    |      1 |      1 |      1 |00:00:00.01 |       2 |
|*  2 |   INDEX UNIQUE SCAN         | PK_DEPT |      1 |      1 |      1 |00:00:00.01 |       1 |
------------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   2 - access("DEPTNO"=10)
```

• 출력 포맷 설정

DISPLAY, DISPLAY_CURSOR 함수는 다양한 출력 포맷을 지원한다. 본 실습에서는 그중 일부 포맷의 차이점을 확인하고, 차후 과정이 진행되는 도중 추가적인 부분은 필요시 추가적으로 확인한다.

```
SQL> SELECT deptno, SUM(sal)
     FROM emp
     GROUP BY deptno ;
    DEPTNO   SUM(SAL)
---------- ----------
        30       9400
        20      10875
        10       8750
```

```
SQL> SELECT * FROM table(myxplan.display_cursor(null, null, 'MEMSTATS LAST')) ;
PLAN_TABLE_OUTPUT
------------------------------------
SQL_ID  bq6u6kyt27kyw, child number 0
------------------------------------
SELECT deptno, SUM(sal) FROM emp GROUP BY deptno
Plan hash value: 4067220884
-----------------------------------------------------------------------------------------------------------
| Id  | Operation          | Name | Starts | E-Rows | A-Rows |   A-Time   |  OMem |  1Mem | Used-Mem |
-----------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |      |      1 |        |      3 |00:00:00.01 |       |       |          |
|   1 |  HASH GROUP BY     |      |      1 |      3 |      3 |00:00:00.01 |   801K|   801K|  668K (0)|
|   2 |   TABLE ACCESS FULL| EMP  |      1 |     14 |     14 |00:00:00.01 |       |       |          |
-----------------------------------------------------------------------------------------------------------
```

MEMSTATS는 PGA 영역의 추가적으로 사용한 메모리의 사용량을 확인할 수 있다. 하지만 단계별 I/O의 횟수는 확인이 불가능하다. 이럴땐 ALLSTATS를 이용한다. 또한 한 번 수행된 SQL의 SQL_ID, CHILD_NUMBER를 알고 있다면 다음과 같이 실행하는 것도 가능하다.

```
SQL> SELECT *
     FROM table(myxplan.display_cursor('bq6u6kyt27kyw', 0, 'ALLSTATS LAST')) ;
-----------------------------------------------------------------------------------------------------------------
| Id  | Operation          | Name | Starts | E-Rows | A-Rows |   A-Time   | Buffers |  OMem |  1Mem | Used-Mem |
-----------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |      |      1 |        |      3 |00:00:00.01 |       3 |       |       |          |
|   1 |  HASH GROUP BY     |      |      1 |      3 |      3 |00:00:00.01 |       3 |   801K|   801K|  668K (0)|
|   2 |   TABLE ACCESS FULL| EMP  |      1 |     14 |     14 |00:00:00.01 |       3 |       |       |          |
-----------------------------------------------------------------------------------------------------------------
```

어떤 문장이든 I/O는 수행된다. 하지만 모든 문장이 PGA에 추가 메모리를 사용하는 것은 아니다. 본 과정에서는 지면상 컬럼의 개수를 줄이기 위해 I/O의 통계를 주로 확인할 것이며, 필요시 MEMSTATS를 함께 호출할 것이다.

SQL*Plus를 이용하고 있다면 adsql/iostat.sql, memstat.sql, xplan.sql, xplan_all.sql 스크립트를 호출하여 실습을 진행하며, SQL Developer를 사용할 경우에는 해당 스크립트의 명령문을 확인하여 필요한 문장을 따로 실행한다. 또한 각 스크립트의 MYXPLAN 패키지의 이름은 DBMS_XPLAN 패키지로 변경하여도 동일 결과를 확인할 수 있다.

3.4. SQL Trace

- Server Process 가 수행하는 모든 작업 내용 확인 가능
- 파라미터를 통해 생성되는 파일의 위치 설정 가능
- 실제 사용된 실행 계획 및 실행 도중 사용한 자원의 사용량 확인 가능
- TKPROF 유틸리티를 이용하여 보고서 파일을 생성
- DB Server 가 위치한 OS에 접근 권한 필요

```
SQL> show parameter sql_trace
NAME                                 TYPE        VALUE
------------------------------------ ----------- ------------------------------
sql_trace                            boolean     FALSE

SQL> ALTER SESSION SET sql_trace = true ;
Session altered.

SQL> SELECT * FROM dept WHERE deptno = 10 ;
    DEPTNO DNAME                        LOC
---------- ---------------------------- --------------------------
        10 ACCOUNTING                   NEW YORK

SQL> ALTER SESSION SET sql_trace = false ;
Session altered.

- Oracle Database 11g New Features
SQL> SELECT value
     FROM v$diag_info
     WHERE name = 'Default Trace File' ;
VALUE
--------------------------------------------------------------------------------
/u01/app/oracle/diag/rdbms/orcl/orcl/trace/orcl_ora_8426.trc

SQL> host tkprof /u01/app/oracle/diag/rdbms/orcl/orcl/trace/orcl_ora_8426.trc
output.txt sys=no
TKPROF: Release 11.2.0.1.0 - Development on Sun Dec 7 18:01:15 2014
Copyright (c) 1982, 2009, Oracle and/or its affiliates.  All rights reserved.
```

```
SQL> edit output.txt
********************************************************************************
call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.00       0.00          0          0          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch        2      0.00       0.00          0          2          0           1
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total        4      0.00       0.00          0          2          0           1

Misses in library cache during parse: 1
Optimizer mode: ALL_ROWS
Parsing user id: 153

Rows     Row Source Operation
-------  ---------------------------------------------------
      1  TABLE ACCESS BY INDEX ROWID DEPT (cr=2 pr=0 pw=0 time=0 us cost=1 size=20 card=1)
      1   INDEX UNIQUE SCAN DEPT_DEPTNO_IX (cr=1 pr=0 pw=0 time=0 us cost=0 size=0 card=1)
********************************************************************************
```

4. 옵티마이저 연산자(Optimizer Operations)

4.1. Table Access

- Full Table Scan
- Rowid Scan
- Sample Table Scan

*** Full Table Scan**

- 많은 양의 데이터 검색 시 유용함
- High Water Mark 아래의 모든 블록 I/O
- Multi Block I/O 수행 (db_file_multiblock_read_count)

```
SQL> show parameter db_file_multiblock_read_count
NAME                                 TYPE        VALUE
------------------------------------ ----------- ------------------------------
db_file_multiblock_read_count        integer     128

SQL> SELECT /*+ full(emp) */ *
     FROM emp ;
SQL> @xplan
-------------------------------------------------------------------------------------------
| Id  | Operation         | Name | Starts | E-Rows | A-Rows |   A-Time   | Buffers |
-------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT  |      |      1 |        |     14 |00:00:00.01 |       4 |
|   1 |  TABLE ACCESS FULL| EMP  |      1 |     14 |     14 |00:00:00.01 |       4 |
-------------------------------------------------------------------------------------------
```

*** Rowid Scan**

- User Rowid, Index Rowid를 이용하여 소량의 데이터 검색 시 유용함
- Single Block I/O 수행

```
SQL> SELECT /*+ rowid */ *
     FROM emp
     WHERE rowid = 'AAAUn7AAIAAAl/bAAA' ;
SQL> @xplan
-------------------------------------------------------------------------------------------------
| Id  | Operation                  | Name | Starts | E-Rows | A-Rows |   A-Time   | Buffers |
-------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT           |      |      1 |        |      1 |00:00:00.01 |       1 |
|   1 |  TABLE ACCESS BY USER ROWID| EMP  |      1 |      1 |      1 |00:00:00.01 |       1 |
-------------------------------------------------------------------------------------------------


SQL> SELECT /*+ index (emp(empno)) */ *
     FROM emp
     WHERE empno = 7782 ;
SQL> @xplan
-------------------------------------------------------------------------------------------------------
| Id  | Operation                   | Name   | Starts | E-Rows | A-Rows |   A-Time   | Buffers |
-------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT            |        |      1 |        |      1 |00:00:00.01 |       2 |
|   1 |  TABLE ACCESS BY INDEX ROWID| EMP    |      1 |      1 |      1 |00:00:00.01 |       2 |
|*  2 |   INDEX UNIQUE SCAN         | PK_EMP |      1 |      1 |      1 |00:00:00.01 |       1 |
-------------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   2 - access("EMPNO"=7782)
```

*** Sample Table Scan**

- 무작위 샘플 데이터 검색
- 테스트용 데이터 생성 시 사용
- 샘플링 값은 0.000001 ~ 99.999999 범위 사용
- SAMPLE, SAMPLE BLOCK 사용 가능

```
SQL> SELECT *
     FROM emp SAMPLE (50) ;
 SQL> @xplan
-------------------------------------------------------------------------------------------
| Id  | Operation           | Name | Starts | E-Rows | A-Rows |   A-Time   | Buffers |
-------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT    |      |      1 |        |      6 |00:00:00.01 |       4 |
|   1 |  TABLE ACCESS SAMPLE| EMP  |      1 |      7 |      6 |00:00:00.01 |       4 |
-------------------------------------------------------------------------------------------

SQL> SELECT *
     FROM emp SAMPLE BLOCK (10) ;
SQL> @xplan
-------------------------------------------------------------------------------------------
| Id  | Operation           | Name | Starts | E-Rows | A-Rows |   A-Time   | Buffers |
-------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT    |      |      1 |        |     14 |00:00:00.01 |       4 |
|   1 |  TABLE ACCESS SAMPLE| EMP  |      1 |      1 |     14 |00:00:00.01 |       4 |
-------------------------------------------------------------------------------------------
```

4.2. Index Scan

- Index Unique Scan
- Index Range Scan
- Index Range Scan Descending
- Index Full Scan
- Index Fast Full Scan
- Index Skip Scan
- Index Join

*** Index Unique Scan**

- Unique Index 필요
  - Single Block I/O 사용

```
SQL> SELECT /*+ index (e(employee_id)) */
            employee_id, last_name, salary, department_id
     FROM employees e
     WHERE employee_id = 100 ;


SQL> @xplan
-------------------------------------------------------------------------------------------------------------
| Id  | Operation                   | Name          | Starts | E-Rows | A-Rows |   A-Time   | Buffers | Reads  |
-------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT            |               |      1 |        |      1 |00:00:00.01 |       2 |      1 |
|   1 |  TABLE ACCESS BY INDEX ROWID| EMPLOYEES     |      1 |      1 |      1 |00:00:00.01 |       2 |      1 |
|*  2 |   INDEX UNIQUE SCAN         | EMP_EMP_ID_PK |      1 |      1 |      1 |00:00:00.01 |       1 |      1 |
-------------------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   2 - access("EMPLOYEE_ID"=100)
```

*** Index Range Scan**

- 가장 일반적인 접근 방법이며 기본적으로 Ascending 하게 접근 됨
- Single Block I/O 사용

```
SQL> SELECT /*+ index_rs_asc (e(employee_id)) */
            employee_id, last_name, salary, department_id
     FROM employees e
     WHERE employee_id > 100 ;
```

```
SQL> @xplan
-------------------------------------------------------------------------------------------------------------
| Id  | Operation                    | Name          | Starts | E-Rows | A-Rows |   A-Time   | Buffers |
-------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |               |      1 |        |    106 |00:00:00.01 |       8 |
|   1 |  TABLE ACCESS BY INDEX ROWID| EMPLOYEES     |      1 |    106 |    106 |00:00:00.01 |       8 |
|*  2 |   INDEX RANGE SCAN           | EMP_EMP_ID_PK |      1 |    106 |    106 |00:00:00.01 |       3 |
-------------------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   2 - access("EMPLOYEE_ID">100)
```

*** Index Range Scan Descending**

- 인덱스에서 조건에 만족하는 가장 큰 값부터 작은 값으로 접근
- Single Block I/O 사용

```
SQL> SELECT /*+ index_rs_desc (e(employee_id)) */
            employee_id, last_name, salary, department_id
     FROM employees e
     WHERE employee_id > 100 ;
SQL> @xplan
-------------------------------------------------------------------------------------------------------------
| Id  | Operation                    | Name          | Starts | E-Rows | A-Rows |   A-Time   | Buffers |
-------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |               |      1 |        |    106 |00:00:00.01 |       8 |
|   1 |  TABLE ACCESS BY INDEX ROWID | EMPLOYEES     |      1 |    106 |    106 |00:00:00.01 |       8 |
|*  2 |   INDEX RANGE SCAN DESCENDING| EMP_EMP_ID_PK |      1 |    106 |    106 |00:00:00.01 |       3 |
-------------------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   2 - access("EMPLOYEE_ID">100)
       filter("EMPLOYEE_ID">100)
```

*** Index Full Scan**

- 사용되는 컬럼에 NOT NULL 제약조건이 존재하거나 조건식 필요
- Single Block I/O 사용

```
SQL> SELECT /*+ index (e(employee_id)) */ COUNT(*)
     FROM employees e
     WHERE employee_id IS NOT NULL ;
SQL> @xplan
-------------------------------------------------------------------------------------------
| Id  | Operation        | Name          | Starts | E-Rows | A-Rows |   A-Time   | Buffers |
-------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT |               |      1 |        |      1 |00:00:00.01 |       1 |
|   1 |  SORT AGGREGATE  |               |      1 |      1 |      1 |00:00:00.01 |       1 |
|   2 |   INDEX FULL SCAN| EMP_EMP_ID_PK |      1 |    107 |    107 |00:00:00.01 |       1 |
-------------------------------------------------------------------------------------------
```

*** Index Fast Full Scan**

- 사용되는 컬럼에 NOT NULL 제약조건이 존재하거나 조건식 필요
- Multi Block I/O 사용

```
SQL> SELECT /*+ index_ffs (e(employee_id)) */ COUNT(*)
     FROM employees e
     WHERE employee_id IS NOT NULL ;
SQL> @xplan
-------------------------------------------------------------------------------------------
| Id  | Operation             | Name          | Starts | E-Rows | A-Rows |   A-Time   | Buffers |
-------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT      |               |      1 |        |      1 |00:00:00.01 |       4 |
|   1 |  SORT AGGREGATE       |               |      1 |      1 |      1 |00:00:00.01 |       4 |
|   2 |   INDEX FAST FULL SCAN| EMP_EMP_ID_PK |      1 |    107 |    107 |00:00:00.01 |       4 |
-------------------------------------------------------------------------------------------
```

*** Index Skip Scan**

- 결합 인덱스의 선행 컬럼이 조건식에 없어도 Index 사용 가능
- Single Block I/O 사용

```
SQL> SELECT /*+ rule */
     employee_id, first_name, last_name, salary, department_id
     FROM employees e
     WHERE first_name = 'Steven' ;
SQL> @xplan
-------------------------------------------------------------------------------------
| Id  | Operation         | Name      | Starts | A-Rows |   A-Time   | Buffers |
-------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT  |           |      1 |      2 |00:00:00.01 |       5 |
|*  1 |  TABLE ACCESS FULL| EMPLOYEES |      1 |      2 |00:00:00.01 |       5 |
-------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   1 - filter("FIRST_NAME"='Steven')

SQL> SELECT /*+ index_ss (e empl_name_ix) */
           employee_id, first_name, last_name, salary, department_id
     FROM employees e
     WHERE first_name = 'Steven' ;
SQL> @xplan
-----------------------------------------------------------------------------------------------------
| Id  | Operation                   | Name         | Starts | E-Rows | A-Rows |   A-Time   | Buffers |
-----------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT            |              |      1 |        |      2 |00:00:00.01 |       4 |
|   1 |  TABLE ACCESS BY INDEX ROWID| EMPLOYEES    |      1 |      1 |      2 |00:00:00.01 |       4 |
|*  2 |   INDEX SKIP SCAN           | EMPL_NAME_IX |      1 |      1 |      2 |00:00:00.01 |       2 |
-----------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   2 - access("FIRST_NAME"='Steven')
       filter("FIRST_NAME"='Steven')
```

*** Index Join**

- 테이블의 접근 없이 인덱스만 이용 가능할 경우 사용 됨
- Single Block I/O 사용

```
SQL> SELECT  department_id, job_id
     FROM employees e
     WHERE department_id = 80
       AND job_id        = 'SA_REP'  ;
SQL> @xplan
-------------------------------------------------------------------------------------------------------------------
| Id  | Operation                    | Name        | Starts | E-Rows | A-Rows |   A-Time   | Buffers |
-------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |             |      1 |        |     29 |00:00:00.01 |       4 |
|*  1 |  TABLE ACCESS BY INDEX ROWID| EMPLOYEES   |      1 |     29 |     29 |00:00:00.01 |       4 |
|*  2 |   INDEX RANGE SCAN           | EMPL_JOB_IX |      1 |     30 |     30 |00:00:00.01 |       2 |
-------------------------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   1 - filter("DEPARTMENT_ID"=80)
   2 - access("JOB_ID"='SA_REP')

SQL> SELECT  /*+ index_join (e empl_department_ix empl_job_ix) */
             department_id, job_id
     FROM employees e
     WHERE department_id = 80
       AND job_id        = 'SA_REP'  ;
SQL> @xplan
-----------------------------------------------------------------------------------------------------------------------------------
| Id  | Operation          | Name             | Starts | E-Rows | A-Rows |   A-Time   | Buffers |  OMem |  1Mem | Used-Mem |
-----------------------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |                  |      1 |        |     29 |00:00:00.01 |       3 |       |       |          |
|*  1 |  VIEW              | index$_join$_001 |      1 |      2 |     29 |00:00:00.01 |       3 |       |       |          |
|*  2 |   HASH JOIN        |                  |      1 |        |     29 |00:00:00.01 |       3 |   842K|   842K| 1214K (0)|
|*  3 |    INDEX RANGE SCAN| EMPL_JOB_IX      |      1 |      2 |     30 |00:00:00.01 |       1 |       |       |          |
|*  4 |    INDEX RANGE SCAN| EMPL_DEPTNO_IX   |      1 |      2 |     34 |00:00:00.01 |       2 |       |       |          |
-----------------------------------------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   1 - filter(("JOB_ID"='SA_REP' AND "DEPARTMENT_ID"=80))
   2 - access(ROWID=ROWID)
   3 - access("JOB_ID"='SA_REP')
   4 - access("DEPARTMENT_ID"=80)
```

4.3. Join Operations

- Nested Loops Join
- Sort Merge Join
- Hash Join

*** Nested Loops Join**

- 선행 테이블 (Outer Table) 결정 후 후행 테이블(Inner Table)에 반복적인 접근
- 후행 테이블의 조인 컬럼에 인덱스 필요
- 소량의 데이터를 조인 시 사용
- 부분 범위 (first_rows) 처리에 최적화
- 많은 양의 데이터 조인 시 Random Access 증가
- DB 버전에 따라 Nested Loops Join의 처리 절차는 개선된 부분이 존재함

```
SQL> SELECT /*+ optimizer_features_enable('8.1.7')
                use_nl(d e)
                index(e empl_deptno_ix)
                no_nlj_prefetch(e) */
            d.department_id, d.department_name, e.last_name, e.salary
     FROM departments d, employees e
     WHERE d.department_id = e.department_id ;
SQL> @xplan
-------------------------------------------------------------------------------------------------------------
| Id  | Operation                    | Name           | Starts | E-Rows | A-Rows |   A-Time   | Buffers |
-------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |                |      1 |        |    106 |00:00:00.01 |      16 |
|   1 |  NESTED LOOPS                |                |      1 |    106 |    106 |00:00:00.01 |      16 |
|   2 |   TABLE ACCESS FULL          | DEPARTMENTS    |      1 |     27 |     27 |00:00:00.01 |       5 |
|   3 |   TABLE ACCESS BY INDEX ROWID| EMPLOYEES      |     27 |      4 |    106 |00:00:00.01 |      11 |
|*  4 |    INDEX RANGE SCAN          | EMPL_DEPTNO_IX |     27 |     10 |    106 |00:00:00.01 |       6 |
-------------------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   4 - access("D"."DEPARTMENT_ID"="E"."DEPARTMENT_ID")
```

```
SQL> SELECT /*+ optimizer_features_enable('9.2.0')
                leading(d) use_nl(e) index(e empl_deptno_ix)
                nlj_prefetch(e) */
           d.department_id, d.department_name, e.last_name, e.salary
     FROM departments d, employees e
     WHERE d.department_id = e.department_id ;
SQL> @xplan
-------------------------------------------------------------------------------------------------------
| Id  | Operation                   | Name           | Starts | E-Rows | A-Rows |   A-Time   | Buffers |
-------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT            |                |      0 |        |      0 |00:00:00.01 |       0 |
|   1 |  TABLE ACCESS BY INDEX ROWID| EMPLOYEES      |      1 |      4 |    106 |00:00:00.01 |      16 |
|   2 |   NESTED LOOPS              |                |      1 |    106 |    134 |00:00:00.01 |      11 |
|   3 |    TABLE ACCESS FULL        | DEPARTMENTS    |      1 |     27 |     27 |00:00:00.01 |       5 |
|*  4 |    INDEX RANGE SCAN         | EMPL_DEPTNO_IX |     27 |     10 |    106 |00:00:00.01 |       6 |
-------------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   4 - access("D"."DEPARTMENT_ID"="E"."DEPARTMENT_ID")

SQL> SELECT /*+ leading(d) use_nl(e) nlj_batching(e)
                no_index(d(department_id)) */
           d.department_id, d.department_name, e.last_name, e.salary
     FROM departments d, employees e
     WHERE d.department_id = e.department_id ;
SQL> @xplan
-------------------------------------------------------------------------------------------------------
| Id  | Operation                   | Name           | Starts | E-Rows | A-Rows |   A-Time   | Buffers |
-------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT            |                |      1 |        |    106 |00:00:00.01 |      16 |
|   1 |  NESTED LOOPS               |                |      1 |        |    106 |00:00:00.01 |      16 |
|   2 |   NESTED LOOPS              |                |      1 |    106 |    106 |00:00:00.01 |      11 |
|   3 |    TABLE ACCESS FULL        | DEPARTMENTS    |      1 |     27 |     27 |00:00:00.01 |       5 |
|*  4 |    INDEX RANGE SCAN         | EMPL_DEPTNO_IX |     27 |     10 |    106 |00:00:00.01 |       6 |
|   5 |   TABLE ACCESS BY INDEX ROWID| EMPLOYEES      |    106 |      4 |    106 |00:00:00.01 |       5 |
-------------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   4 - access("D"."DEPARTMENT_ID"="E"."DEPARTMENT_ID")
```

*** Sort Merge Join**

- 각각의 집합을 조인 컬럼으로 정렬 후 조인 수행
- 정렬 작업이 완료되기 전 부분적인 조인 결과 도출 불가능
- 정렬해야 할 데이터가 많은 경우 부담이 가장 큰 조인 방법
- Non equi join 시 사용 가능

```
SQL> SELECT /*+ leading(d) use_merge(e) no_index(d) */
            d.department_id, d.department_name, e.last_name, e.salary
     FROM departments d, employees e
     WHERE d.department_id = e.department_id ;
SQL> @xplan
-------------------------------------------------------------------------------------------------------------------
| Id  | Operation           | Name        | Starts | E-Rows | A-Rows |   A-Time   | Buffers |  OMem |  1Mem | Used-Mem |
-------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT    |             |      1 |        |    106 |00:00:00.01 |       7 |       |       |          |
|   1 |  MERGE JOIN         |             |      1 |    106 |    106 |00:00:00.01 |       7 |       |       |          |
|   2 |   SORT JOIN         |             |      1 |     27 |     27 |00:00:00.01 |       3 |  2048 |  2048 | 2048  (0)|
|   3 |    TABLE ACCESS FULL| DEPARTMENTS |      1 |     27 |     27 |00:00:00.01 |       3 |       |       |          |
|*  4 |   SORT JOIN         |             |     27 |    107 |    106 |00:00:00.01 |       4 |  6144 |  6144 | 6144  (0)|
|   5 |    TABLE ACCESS FULL| EMPLOYEES   |      1 |    107 |    107 |00:00:00.01 |       4 |       |       |          |
-------------------------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   4 - access("D"."DEPARTMENT_ID"="E"."DEPARTMENT_ID")
       filter("D"."DEPARTMENT_ID"="E"."DEPARTMENT_ID")


SQL> SELECT /*+ leading(e) use_merge(s) no_index(e) */
                e.empno, e.ename, e.sal, s.grade
     FROM emp e, salgrade s
     WHERE e.sal BETWEEN s.losal AND s.hisal ;
SQL> @xplan
-------------------------------------------------------------------------------------------------------------------------
| Id  | Operation           | Name     | Starts | E-Rows | A-Rows |   A-Time   | Buffers | Reads  |  OMem |  1Mem | Used-Mem |
-------------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT    |          |      1 |        |     14 |00:00:00.01 |       6 |      2 |       |       |          |
|   1 |  MERGE JOIN         |          |      1 |     42 |     14 |00:00:00.01 |       6 |      2 |       |       |          |
|   2 |   SORT JOIN         |          |      1 |     14 |     14 |00:00:00.01 |       3 |      0 |  2048 |  2048 | 2048  (0)|
|   3 |    TABLE ACCESS FULL| EMP      |      1 |     14 |     14 |00:00:00.01 |       3 |      0 |       |       |          |
|*  4 |   FILTER            |          |     14 |        |     14 |00:00:00.01 |       3 |      2 |       |       |          |
|*  5 |    SORT JOIN        |          |     14 |      5 |     40 |00:00:00.01 |       3 |      2 |  2048 |  2048 | 2048  (0)|
|   6 |     TABLE ACCESS FULL| SALGRADE |      1 |      5 |      5 |00:00:00.01 |       3 |      2 |       |       |          |
-------------------------------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   4 - filter("E"."SAL"<="S"."HISAL")
   5 - access(INTERNAL_FUNCTION("E"."SAL")>=INTERNAL_FUNCTION("S"."LOSAL"))
       filter(INTERNAL_FUNCTION("E"."SAL")>=INTERNAL_FUNCTION("S"."LOSAL"))
```

*** Hash Join**

- 두 집합 중 크기가 작은 테이블을 선행 테이블(hash table)로 결정 후,
  후행 테이블(Probe Table)을 엑세스하여 조인
- Equi Join 에서만 사용 가능
- 대량의 데이터를 조인 시 사용
- 전체 범위 (all_rows)에 최적화
- 소량의 데이터를 조인할 때 Hash Join 이 사용되면 불필요한 I/O 증가

```
SQL> SELECT /*+ leading(d) use_hash(e) no_index(d(department_id)) */
            d.department_id, d.department_name, e.last_name, e.salary
     FROM departments d, employees e
     WHERE d.department_id = e.department_id ;
SQL> @xplan
-------------------------------------------------------------------------------------------------------------------
| Id  | Operation          | Name        | Starts | E-Rows | A-Rows |   A-Time   | Buffers |  OMem |  1Mem | Used-Mem |
-------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |             |      1 |        |    106 |00:00:00.01 |       9 |       |       |          |
|*  1 |  HASH JOIN         |             |      1 |    106 |    106 |00:00:00.01 |       9 |   862K|   862K| 1190K (0)|
|   2 |   TABLE ACCESS FULL| DEPARTMENTS |      1 |     27 |     27 |00:00:00.01 |       3 |       |       |          |
|   3 |   TABLE ACCESS FULL| EMPLOYEES   |      1 |    107 |    107 |00:00:00.01 |       6 |       |       |          |
-------------------------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   1 - access("D"."DEPARTMENT_ID"="E"."DEPARTMENT_ID")

SQL> SELECT /*+ leading(e) use_hash(s) */
            e.empno, e.ename, e.sal, s.grade
     FROM emp e, salgrade s
     WHERE e.sal BETWEEN s.losal AND s.hisal ;
SQL> @xplan
-----------------------------------------------------------------------------------------
| Id  | Operation          | Name     | Starts | E-Rows | A-Rows |   A-Time   | Buffers |
-----------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |          |      1 |        |     14 |00:00:00.01 |      47 |
|   1 |  NESTED LOOPS      |          |      1 |     42 |     14 |00:00:00.01 |      47 |
|   2 |   TABLE ACCESS FULL| EMP      |      1 |     14 |     14 |00:00:00.01 |       4 |
|*  3 |   TABLE ACCESS FULL| SALGRADE |     14 |      3 |     14 |00:00:00.01 |      43 |
-----------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   3 - filter(("E"."SAL">="S"."LOSAL" AND "E"."SAL"<="S"."HISAL"))
```

*** Join 순서 조정**

- ORDERED 힌트 사용 : FROM 절의 나열된 테이블의 순서대로 조인 수행
- LEADING 힌트 사용 : 선행 테이블 지정

```
SQL> SELECT /*+ ordered use_nl(d e) */
            d.department_id, d.department_name, e.last_name, e.salary
     FROM employees e, departments d
     WHERE d.department_id = e.department_id ;
SQL> @xplan
-------------------------------------------------------------------------------------------------------------
| Id  | Operation                    | Name        | Starts | E-Rows | A-Rows |   A-Time   | Buffers |
-------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |             |      1 |        |    106 |00:00:00.01 |     117 |
|   1 |  NESTED LOOPS                |             |      1 |        |    106 |00:00:00.01 |     117 |
|   2 |   NESTED LOOPS               |             |      1 |    106 |    106 |00:00:00.01 |      11 |
|   3 |    TABLE ACCESS FULL         | EMPLOYEES   |      1 |    107 |    107 |00:00:00.01 |       6 |
|*  4 |    INDEX UNIQUE SCAN         | DEPT_ID_PK  |    107 |      1 |    106 |00:00:00.01 |       5 |
|   5 |   TABLE ACCESS BY INDEX ROWID| DEPARTMENTS |    106 |      1 |    106 |00:00:00.01 |     106 |
-------------------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   4 - access("D"."DEPARTMENT_ID"="E"."DEPARTMENT_ID")

SQL> SELECT /*+ ordered use_nl(e d) no_index(d(department_id)) */
            d.department_id, d.department_name, e.last_name, e.salary
     FROM departments d, employees e
     WHERE d.department_id = e.department_id ;
SQL> @xplan
-------------------------------------------------------------------------------------------------------------
| Id  | Operation                    | Name           | Starts | E-Rows | A-Rows |   A-Time   | Buffers |
-------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |                |      1 |        |    106 |00:00:00.01 |      16 |
|   1 |  NESTED LOOPS                |                |      1 |        |    106 |00:00:00.01 |      16 |
|   2 |   NESTED LOOPS               |                |      1 |    106 |    106 |00:00:00.01 |      11 |
|   3 |    TABLE ACCESS FULL         | DEPARTMENTS    |      1 |     27 |     27 |00:00:00.01 |       5 |
|*  4 |    INDEX RANGE SCAN          | EMPL_DEPTNO_IX |     27 |     10 |    106 |00:00:00.01 |       6 |
|   5 |   TABLE ACCESS BY INDEX ROWID| EMPLOYEES      |    106 |      4 |    106 |00:00:00.01 |       5 |
-------------------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   4 - access("D"."DEPARTMENT_ID"="E"."DEPARTMENT_ID")
```

```
SQL> SELECT /*+ leading(e) use_nl(d) */
            d.department_id, d.department_name, e.last_name, e.salary
     FROM departments d, employees e
     WHERE d.department_id = e.department_id ;
SQL> @xplan
-------------------------------------------------------------------------------------------------------------
| Id  | Operation                    | Name        | Starts | E-Rows | A-Rows |   A-Time   | Buffers |
-------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |             |      1 |        |    106 |00:00:00.01 |     117 |
|   1 |  NESTED LOOPS                |             |      1 |        |    106 |00:00:00.01 |     117 |
|   2 |   NESTED LOOPS               |             |      1 |    106 |    106 |00:00:00.01 |      11 |
|   3 |    TABLE ACCESS FULL         | EMPLOYEES   |      1 |    107 |    107 |00:00:00.01 |       6 |
|*  4 |    INDEX UNIQUE SCAN         | DEPT_ID_PK  |    107 |      1 |    106 |00:00:00.01 |       5 |
|   5 |   TABLE ACCESS BY INDEX ROWID| DEPARTMENTS |    106 |      1 |    106 |00:00:00.01 |     106 |
-------------------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   4 - access("D"."DEPARTMENT_ID"="E"."DEPARTMENT_ID")
```

4.4. Subquery 종류

- Single row Subquery
- Multiple rows Subquery
- Multiple columns Subquery
- Inline View
- Correlated Subquery (Scalar Subquery)

*** Single Row Subquery**

```
SQL> SELECT *
     FROM emp
     WHERE deptno = ( SELECT deptno
                      FROM dept
                      WHERE dname = 'ACCOUNTING' ) ;
SQL> @xplan
-------------------------------------------------------------------------------------------------------------
| Id  | Operation                    | Name          | Starts | E-Rows | A-Rows |   A-Time   | Buffers |
-------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |               |      1 |        |      3 |00:00:00.01 |       6 |
|   1 |  TABLE ACCESS BY INDEX ROWID | EMP           |      1 |      5 |      3 |00:00:00.01 |       6 |
|*  2 |   INDEX RANGE SCAN           | EMP_DEPTNO_IX |      1 |      5 |      3 |00:00:00.01 |       4 |
|   3 |    TABLE ACCESS BY INDEX ROWID| DEPT          |      1 |      1 |      1 |00:00:00.01 |       2 |
|*  4 |     INDEX RANGE SCAN         | DEPT_DNAME_IX |      1 |      1 |      1 |00:00:00.01 |       1 |
-------------------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   2 - access("DEPTNO"=)
   4 - access("DNAME"='ACCOUNTING')
```

*** Multiple Rows Subquery**

```
SQL> SELECT *
     FROM emp
     WHERE deptno IN ( SELECT /*+ unnest */
                              deptno
                       FROM dept ) ;
SQL> @xplan
-------------------------------------------------------------------------------------------
| Id  | Operation          | Name    | Starts | E-Rows | A-Rows |   A-Time   | Buffers |
-------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |         |      1 |        |     14 |00:00:00.01 |       8 |
|   1 |  NESTED LOOPS      |         |      1 |     14 |     14 |00:00:00.01 |       8 |
|   2 |   TABLE ACCESS FULL| EMP     |      1 |     14 |     14 |00:00:00.01 |       4 |
|*  3 |   INDEX UNIQUE SCAN| PK_DEPT |     14 |      1 |     14 |00:00:00.01 |       4 |
-------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   3 - access("DEPTNO"="DEPTNO")

SQL> SELECT *
     FROM emp
     WHERE deptno IN ( SELECT /*+ no_unnest */
                              deptno
                       FROM dept ) ;
SQL> @xplan
-------------------------------------------------------------------------------------------
| Id  | Operation          | Name    | Starts | E-Rows | A-Rows |   A-Time   | Buffers |
-------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |         |      1 |        |     14 |00:00:00.01 |       7 |
|*  1 |  FILTER            |         |      1 |        |     14 |00:00:00.01 |       7 |
|   2 |   TABLE ACCESS FULL| EMP     |      1 |     14 |     14 |00:00:00.01 |       4 |
|*  3 |   INDEX UNIQUE SCAN| PK_DEPT |      3 |      1 |      3 |00:00:00.01 |       3 |
-------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   1 - filter( IS NOT NULL)
   3 - access("DEPTNO"=:B1)
```

```
SQL> SELECT *
     FROM emp e
     WHERE deptno = ( SELECT deptno
                      FROM dept
                      WHERE deptno = e.deptno) ;
SQL> @xplan
-------------------------------------------------------------------------------------------
| Id  | Operation          | Name    | Starts | E-Rows | A-Rows |   A-Time   | Buffers |
-------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |         |      1 |        |     14 |00:00:00.01 |       6 |
|*  1 |  FILTER            |         |      1 |        |     14 |00:00:00.01 |       6 |
|   2 |   TABLE ACCESS FULL| EMP     |      1 |     14 |     14 |00:00:00.01 |       4 |
|*  3 |   INDEX UNIQUE SCAN| PK_DEPT |      3 |      1 |      3 |00:00:00.01 |       2 |
-------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   1 - filter("DEPTNO"=)
   3 - access("DEPTNO"=:B1)
```

Single Row Subquery는 Main Query 보다 먼저 실행되고 그 결과를 Main Query에 리턴하여 실행할 수 있다. 하지만 둘 이상의 결과를 리턴하는 Multiple Rows Subquery는 Query Transformation을 통해 Join Operation를 이용하거나 Filter Operation을 사용한다.

*** Multiple Columns Subquery**

```
SQL> SELECT * FROM emp
     WHERE (deptno,sal) IN ( SELECT deptno, MIN(sal)
                             FROM emp
                             GROUP BY deptno ) ;
SQL> @xplan
-----------------------------------------------------------------------------------------------------------------------------
| Id  | Operation                     | Name          | Starts | E-Rows | A-Rows |   A-Time   | Buffers |  OMem |  1Mem | Used-Mem |
-----------------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT              |               |      1 |        |      3 |00:00:00.01 |       5 |       |       |          |
|*  1 |  FILTER                       |               |      1 |        |      3 |00:00:00.01 |       5 |       |       |          |
|   2 |   HASH GROUP BY               |               |      1 |      6 |     14 |00:00:00.01 |       5 |   721K|   721K| 1154K (0)|
|   3 |    MERGE JOIN                 |               |      1 |     71 |     70 |00:00:00.01 |       5 |       |       |          |
|   4 |     TABLE ACCESS BY INDEX ROWID| EMP          |      1 |     14 |     14 |00:00:00.01 |       2 |       |       |          |
|   5 |      INDEX FULL SCAN          | EMP_DEPTNO_IX |      1 |     14 |     14 |00:00:00.01 |       1 |       |       |          |
|*  6 |     SORT JOIN                 |               |     14 |     14 |     70 |00:00:00.01 |       3 |  2048 |  2048 | 2048  (0)|
|   7 |      TABLE ACCESS FULL        | EMP           |      1 |     14 |     14 |00:00:00.01 |       3 |       |       |          |
-----------------------------------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   1 - filter("EMP"."SAL"=MIN("SAL"))
   6 - access("DEPTNO"="DEPTNO")
       filter("DEPTNO"="DEPTNO")
```

*** Inline View**

```
SQL> SELECT e.deptno, e.empno, e.ename, e.sal, a.avg
     FROM emp e, ( SELECT deptno, AVG(sal) AS AVG
                   FROM emp
                   GROUP BY deptno ) a
     WHERE e.deptno = a.deptno ;
SQL> @xplan
-------------------------------------------------------------------------------------------------------------------------
| Id  | Operation                    | Name          | Starts | E-Rows | A-Rows |   A-Time   | Buffers |  OMem |  1Mem | Used-Mem |
-------------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |               |      1 |        |     14 |00:00:00.01 |       7 |       |       |          |
|   1 |  MERGE JOIN                  |               |      1 |     15 |     14 |00:00:00.01 |       7 |       |       |          |
|   2 |   TABLE ACCESS BY INDEX ROWID| EMP           |      1 |     14 |     14 |00:00:00.01 |       4 |       |       |          |
|   3 |    INDEX FULL SCAN           | EMP_DEPTNO_IX |      1 |     14 |     14 |00:00:00.01 |       2 |       |       |          |
|*  4 |   SORT JOIN                  |               |     14 |      3 |     14 |00:00:00.01 |       3 |  2048 |  2048 | 2048  (0)|
|   5 |    VIEW                      |               |      1 |      3 |      3 |00:00:00.01 |       3 |       |       |          |
|   6 |     HASH GROUP BY            |               |      1 |      3 |      3 |00:00:00.01 |       3 |   778K|   778K| 1161K (0)|
|   7 |      TABLE ACCESS FULL       | EMP           |      1 |     14 |     14 |00:00:00.01 |       3 |       |       |          |
-------------------------------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   4 - access("E"."DEPTNO"="A"."DEPTNO")
       filter("E"."DEPTNO"="A"."DEPTNO")
```

*** Correlated Subquery (Scalar Subquery)**

```
SQL> SELECT e.empno, e.ename, e.sal, e.deptno
     FROM emp e
     WHERE e.sal > ( SELECT /*+ no_unnest */ AVG(SAL)
                     FROM emp
                     WHERE deptno = e.deptno ) ;
SQL> @xplan
-----------------------------------------------------------------------------------------------------------
| Id  | Operation                    | Name          | Starts | E-Rows | A-Rows |   A-Time   | Buffers |
-----------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |               |      1 |        |      6 |00:00:00.01 |       9 |
|*  1 |  FILTER                      |               |      1 |        |      6 |00:00:00.01 |       9 |
|   2 |   TABLE ACCESS FULL          | EMP           |      1 |     14 |     14 |00:00:00.01 |       4 |
|   3 |   SORT AGGREGATE             |               |      3 |      1 |      3 |00:00:00.01 |       5 |
|   4 |    TABLE ACCESS BY INDEX ROWID| EMP          |      3 |      5 |     14 |00:00:00.01 |       5 |
|*  5 |     INDEX RANGE SCAN         | EMP_DEPTNO_IX |      3 |      5 |     14 |00:00:00.01 |       3 |
-----------------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   1 - filter("E"."SAL">)
   5 - access("DEPTNO"=:B1)
```

```
SQL> SELECT e.empno, e.ename, e.sal, e.deptno
     FROM emp e
     WHERE e.sal > ( SELECT /*+ unnest */ AVG(SAL)
                     FROM emp
                     WHERE deptno = e.deptno ) ;
SQL> @xplan
-------------------------------------------------------------------------------------------------------------------------
| Id  | Operation                    | Name          | Starts | E-Rows | A-Rows |   A-Time   | Buffers |  OMem |  1Mem | Used-Mem |
-------------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |               |      1 |        |      6 |00:00:00.01 |       7 |       |       |          |
|   1 |  MERGE JOIN                  |               |      1 |      1 |      6 |00:00:00.01 |       7 |       |       |          |
|   2 |   TABLE ACCESS BY INDEX ROWID| EMP           |      1 |     14 |     14 |00:00:00.01 |       4 |       |       |          |
|   3 |    INDEX FULL SCAN           | EMP_DEPTNO_IX |      1 |     14 |     14 |00:00:00.01 |       2 |       |       |          |
|*  4 |   FILTER                     |               |     14 |        |      6 |00:00:00.01 |       3 |       |       |          |
|*  5 |    SORT JOIN                 |               |     14 |      3 |     14 |00:00:00.01 |       3 |  2048 |  2048 | 2048  (0)|
|   6 |     VIEW                     | VW_SQ_1       |      1 |      3 |      3 |00:00:00.01 |       3 |       |       |          |
|   7 |      SORT GROUP BY           |               |      1 |      3 |      3 |00:00:00.01 |       3 |  2048 |  2048 | 2048  (0)|
|   8 |       TABLE ACCESS FULL      | EMP           |      1 |     14 |     14 |00:00:00.01 |       3 |       |       |          |
-------------------------------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   4 - filter("E"."SAL">"AVG(SAL)")
   5 - access("ITEM_1"="E"."DEPTNO")
       filter("ITEM_1"="E"."DEPTNO")


SQL> SELECT e.empno, e.ename, e.sal, e.deptno, ( SELECT AVG(sal)
                                                 FROM emp
                                                WHERE deptno = e.deptno ) AS avg
     FROM emp e ;
SQL> @xplan
-----------------------------------------------------------------------------------------------------------
| Id  | Operation                    | Name          | Starts | E-Rows | A-Rows |   A-Time   | Buffers |
-----------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |               |      1 |        |     14 |00:00:00.01 |       4 |
|   1 |  SORT AGGREGATE              |               |      3 |      1 |      3 |00:00:00.01 |       4 |
|   2 |   TABLE ACCESS BY INDEX ROWID| EMP           |      3 |      5 |     14 |00:00:00.01 |       4 |
|*  3 |    INDEX RANGE SCAN          | EMP_DEPTNO_IX |      3 |      5 |     14 |00:00:00.01 |       2 |
|   4 |  TABLE ACCESS FULL           | EMP           |      1 |     14 |     14 |00:00:00.01 |       4 |
-----------------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   3 - access("DEPTNO"=:B1)
```

SELECT 절에 포함된 Correlated Subquery는 Join Operation으로 수행은 불가능하다. (상황에 따라 Oracle 12c부터는 가능) 때문에 Main Query에서 하나의 후보행 별로 반복 실행되는 Subquery이며 Oracle Database 9i부터 반복되는 Subquery의 호출을 막고자 동일한 후보값에 대해선 한 번만 실행 가능하도록 실행 결과를 버퍼링 한다.

4.5. Sort 연산자

- ORDER BY
- DISTINCT
- UNION, MINUS, INTERSECT
- BUFFER SORT
- AGGREGATE
- GROUP BY
- SORT MERGE JOIN

ORDER BY

- 사용자 정의의 정렬 수행

```
SQL> SELECT *
       FROM emp
       ORDER BY sal ;
SQL> @xplan
-------------------------------------------------------------------------------------------------------------
| Id  | Operation          | Name | Starts | E-Rows | A-Rows |   A-Time   | Buffers |  OMem |  1Mem | Used-Mem |
-------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |      |      1 |        |     14 |00:00:00.01 |       3 |       |       |          |
|   1 |  SORT ORDER BY     |      |      1 |     14 |     14 |00:00:00.01 |       3 |  2048 |  2048 | 2048  (0)|
|   2 |   TABLE ACCESS FULL| EMP  |      1 |     14 |     14 |00:00:00.01 |       3 |       |       |          |
-------------------------------------------------------------------------------------------------------------
```

AGGREGATE

- 그룹 함수를 이용한 단일 행 추출
- 실제 정렬 작업이 수행되지 않음

```
SQL> SELECT SUM(sal)
     FROM emp ;
SQL> @xplan
-----------------------------------------------------------------------------------------------
| Id  | Operation        | Name       | Starts | E-Rows | A-Rows |   A-Time   | Buffers |
-----------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT |            |      1 |        |      1 |00:00:00.01 |       1 |
|   1 |  SORT AGGREGATE  |            |      1 |      1 |      1 |00:00:00.01 |       1 |
|   2 |   INDEX FULL SCAN| EMP_SAL_IX |      1 |     14 |     14 |00:00:00.01 |       1 |
-----------------------------------------------------------------------------------------------
```

DISTINCT

- 사용자 정의의 중복 행 제거
- v10g 부터는 HASH UNIQUE 사용 됨
- use_hash_aggregation, no_use_hash_aggregation 힌트 사용 가능

```
SQL> SELECT /*+ optimizer_features_enable('9.2.0') */
            DISTINCT deptno
     FROM emp ;
SQL> @xplan
-------------------------------------------------------------------------------------------------------------------
| Id  | Operation          | Name | Starts | E-Rows | A-Rows |   A-Time   | Buffers |  OMem |  1Mem | Used-Mem |
-------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |      |      1 |        |      3 |00:00:00.01 |       3 |       |       |          |
|   1 |  SORT UNIQUE       |      |      1 |      3 |      3 |00:00:00.01 |       3 |  2048 |  2048 | 2048  (0)|
|   2 |   TABLE ACCESS FULL| EMP  |      1 |     14 |     14 |00:00:00.01 |       3 |       |       |          |
-------------------------------------------------------------------------------------------------------------------

SQL> SELECT DISTINCT deptno
     FROM emp ;
SQL> @xplan
-------------------------------------------------------------------------------------------------------------------
| Id  | Operation          | Name | Starts | E-Rows | A-Rows |   A-Time   | Buffers |  OMem |  1Mem | Used-Mem |
-------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |      |      1 |        |      3 |00:00:00.01 |       3 |       |       |          |
|   1 |  HASH UNIQUE       |      |      1 |      3 |      3 |00:00:00.01 |       3 |  1067K|  1067K|  670K (0)|
|   2 |   TABLE ACCESS FULL| EMP  |      1 |     14 |     14 |00:00:00.01 |       3 |       |       |          |
-------------------------------------------------------------------------------------------------------------------

SQL> SELECT DISTINCT deptno
     FROM emp
     ORDER BY deptno ;
SQL> @xplan
-------------------------------------------------------------------------------------------------------------------
| Id  | Operation          | Name | Starts | E-Rows | A-Rows |   A-Time   | Buffers |  OMem |  1Mem | Used-Mem |
-------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |      |      1 |        |      3 |00:00:00.01 |       3 |       |       |          |
|   1 |  SORT UNIQUE       |      |      1 |      3 |      3 |00:00:00.01 |       3 |  2048 |  2048 | 2048  (0)|
|   2 |   TABLE ACCESS FULL| EMP  |      1 |     14 |     14 |00:00:00.01 |       3 |       |       |          |
-------------------------------------------------------------------------------------------------------------------
```

```
SQL> SELECT /*+ no_use_hash_aggregation */  -- 10gNF
            DISTINCT deptno
     FROM emp ;
SQL> @xplan
-------------------------------------------------------------------------------------------------------------
| Id  | Operation          | Name | Starts | E-Rows | A-Rows |   A-Time   | Buffers |  OMem |  1Mem | Used-Mem |
-------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |      |      1 |        |      3 |00:00:00.01 |       3 |       |       |          |
|   1 |  SORT UNIQUE       |      |      1 |      3 |      3 |00:00:00.01 |       3 |  2048 |  2048 | 2048  (0)|
|   2 |   TABLE ACCESS FULL| EMP  |      1 |     14 |     14 |00:00:00.01 |       3 |       |       |          |
-------------------------------------------------------------------------------------------------------------


SQL> SELECT /*+ use_hash_aggregation */
            DISTINCT deptno
     FROM emp ;
SQL> @xplan
-------------------------------------------------------------------------------------------------------------
| Id  | Operation          | Name | Starts | E-Rows | A-Rows |   A-Time   | Buffers |  OMem |  1Mem | Used-Mem |
-------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |      |      1 |        |      3 |00:00:00.01 |       3 |       |       |          |
|   1 |  HASH UNIQUE       |      |      1 |      3 |      3 |00:00:00.01 |       3 |  1067K|  1067K|  667K (0)|
|   2 |   TABLE ACCESS FULL| EMP  |      1 |     14 |     14 |00:00:00.01 |       3 |       |       |          |
-------------------------------------------------------------------------------------------------------------
```

GROUP BY

- 사용자 정의의 그룹 생성
- v10g 부터는 HASH GROUP BY 사용 됨
- use_hash_aggregation, no_use_hash_aggregation 힌트 사용 가능

```
SQL> SELECT /*+ optimizer_features_enable('9.2.0') */
            deptno, SUM(sal)
     FROM emp
     GROUP BY deptno ;
SQL> @xplan
-------------------------------------------------------------------------------------------------------------
| Id  | Operation          | Name | Starts | E-Rows | A-Rows |   A-Time   | Buffers |  OMem |  1Mem | Used-Mem |
-------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |      |      1 |        |      3 |00:00:00.01 |       3 |       |       |          |
|   1 |  SORT GROUP BY     |      |      1 |      3 |      3 |00:00:00.01 |       3 |  2048 |  2048 | 2048  (0)|
|   2 |   TABLE ACCESS FULL| EMP  |      1 |     14 |     14 |00:00:00.01 |       3 |       |       |          |
-------------------------------------------------------------------------------------------------------------
```

```
SQL> SELECT deptno, SUM(sal)
     FROM emp
     GROUP BY deptno ;
SQL> @xplan
-------------------------------------------------------------------------------------------------------------------
| Id  | Operation          | Name | Starts | E-Rows | A-Rows |   A-Time   | Buffers |  OMem |  1Mem | Used-Mem |
-------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |      |      1 |        |      3 |00:00:00.01 |       3 |       |       |          |
|   1 |  HASH GROUP BY     |      |      1 |      3 |      3 |00:00:00.01 |       3 |   801K|   801K|  649K (0)|
|   2 |   TABLE ACCESS FULL| EMP  |      1 |     14 |     14 |00:00:00.01 |       3 |       |       |          |
-------------------------------------------------------------------------------------------------------------------

SQL> SELECT deptno, SUM(sal)
     FROM emp
     GROUP BY deptno
     ORDER BY deptno ;
SQL> @xplan
-------------------------------------------------------------------------------------------------------------------
| Id  | Operation          | Name | Starts | E-Rows | A-Rows |   A-Time   | Buffers |  OMem |  1Mem | Used-Mem |
-------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |      |      1 |        |      3 |00:00:00.01 |       3 |       |       |          |
|   1 |  SORT GROUP BY     |      |      1 |      3 |      3 |00:00:00.01 |       3 |  2048 |  2048 | 2048  (0)|
|   2 |   TABLE ACCESS FULL| EMP  |      1 |     14 |     14 |00:00:00.01 |       3 |       |       |          |
-------------------------------------------------------------------------------------------------------------------

SQL> SELECT /*+ no_use_hash_aggregation */
            deptno, SUM(sal)
     FROM emp
     GROUP BY deptno ;
SQL> @xplan
-------------------------------------------------------------------------------------------------------------------
| Id  | Operation          | Name | Starts | E-Rows | A-Rows |   A-Time   | Buffers |  OMem |  1Mem | Used-Mem |
-------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |      |      1 |        |      3 |00:00:00.01 |       3 |       |       |          |
|   1 |  SORT GROUP BY     |      |      1 |      3 |      3 |00:00:00.01 |       3 |  2048 |  2048 | 2048  (0)|
|   2 |   TABLE ACCESS FULL| EMP  |      1 |     14 |     14 |00:00:00.01 |       3 |       |       |          |
-------------------------------------------------------------------------------------------------------------------

SQL> SELECT /*+ use_hash_aggregation */
            deptno, SUM(sal)
     FROM emp
     GROUP BY deptno ;
SQL> @xplan
-------------------------------------------------------------------------------------------------------------------
| Id  | Operation          | Name | Starts | E-Rows | A-Rows |   A-Time   | Buffers |  OMem |  1Mem | Used-Mem |
-------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |      |      1 |        |      3 |00:00:00.01 |       3 |       |       |          |
|   1 |  HASH GROUP BY     |      |      1 |      3 |      3 |00:00:00.01 |       3 |   801K|   801K|  649K (0)|
|   2 |   TABLE ACCESS FULL| EMP  |      1 |     14 |     14 |00:00:00.01 |       3 |       |       |          |
-------------------------------------------------------------------------------------------------------------------

SQL> ALTER SESSION SET "_gby_hash_aggregation_enabled" = false ;
```

```
SQL> SELECT DISTINCT deptno
     FROM emp ;
SQL> @xplan
-----------------------------------------------------------------------------------------------------------
| Id  | Operation          | Name | Starts | E-Rows | A-Rows |   A-Time   | Buffers |  OMem |  1Mem | Used-Mem |
-----------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |      |      1 |        |      3 |00:00:00.01 |       3 |       |       |          |
|   1 |  SORT UNIQUE       |      |      1 |      3 |      3 |00:00:00.01 |       3 |  2048 |  2048 | 2048  (0)|
|   2 |   TABLE ACCESS FULL| EMP  |      1 |     14 |     14 |00:00:00.01 |       3 |       |       |          |
-----------------------------------------------------------------------------------------------------------

SQL> SELECT deptno, SUM(sal)
     FROM emp
     GROUP BY deptno ;
SQL> @xplan
-----------------------------------------------------------------------------------------------------------
| Id  | Operation          | Name | Starts | E-Rows | A-Rows |   A-Time   | Buffers |  OMem |  1Mem | Used-Mem |
-----------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |      |      1 |        |      3 |00:00:00.01 |       3 |       |       |          |
|   1 |  SORT GROUP BY     |      |      1 |      3 |      3 |00:00:00.01 |       3 |  2048 |  2048 | 2048  (0)|
|   2 |   TABLE ACCESS FULL| EMP  |      1 |     14 |     14 |00:00:00.01 |       3 |       |       |          |
-----------------------------------------------------------------------------------------------------------

SQL> ALTER SESSION SET "_gby_hash_aggregation_enabled" = true ;
```

UNION, MINUS, INTERSECT, (UNION ALL)

- 사용자 정의의 합집합, 차집합, 교집합 생성
- 첫 번째 컬럼을 기준으로 자동 정렬, 중복되는 행 제거

```
SQL> SELECT deptno, empno, ename, sal
     FROM emp WHERE sal > 2000
     UNION ALL
     SELECT deptno, empno, ename, sal
     FROM emp WHERE deptno = 20 ;
SQL> @xplan
-----------------------------------------------------------------------------------------------------------
| Id  | Operation                    | Name          | Starts | E-Rows | A-Rows |   A-Time   | Buffers |
-----------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |               |      1 |        |     11 |00:00:00.01 |       6 |
|   1 |  UNION-ALL                   |               |      1 |        |     11 |00:00:00.01 |       6 |
|   2 |   TABLE ACCESS BY INDEX ROWID| EMP           |      1 |      6 |      6 |00:00:00.01 |       4 |
|*  3 |    INDEX RANGE SCAN          | EMP_SAL_IX    |      1 |      6 |      6 |00:00:00.01 |       2 |
|   4 |   TABLE ACCESS BY INDEX ROWID| EMP           |      1 |      5 |      5 |00:00:00.01 |       2 |
|*  5 |    INDEX RANGE SCAN          | EMP_DEPTNO_IX |      1 |      5 |      5 |00:00:00.01 |       1 |
-----------------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------

   3 - access("SAL">2000)
   5 - access("DEPTNO"=20)
```

 # UNION

```
-----------------------------------------------------------------------------------------------------------------------------------
| Id  | Operation                     | Name          | Starts | E-Rows | A-Rows |   A-Time   | Buffers |  OMem |  1Mem | Used-Mem |
-----------------------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT              |               |      1 |        |      8 |00:00:00.01 |       4 |       |       |          |
|   1 |  SORT UNIQUE                  |               |      1 |     11 |      8 |00:00:00.01 |       4 |  2048 |  2048 | 2048  (0)|
|   2 |   UNION-ALL                   |               |      1 |        |     11 |00:00:00.01 |       4 |       |       |          |
|   3 |    TABLE ACCESS BY INDEX ROWID| EMP           |      1 |      6 |      6 |00:00:00.01 |       2 |       |       |          |
|*  4 |     INDEX RANGE SCAN          | EMP_SAL_IX    |      1 |      6 |      6 |00:00:00.01 |       1 |       |       |          |
|   5 |    TABLE ACCESS BY INDEX ROWID| EMP           |      1 |      5 |      5 |00:00:00.01 |       2 |       |       |          |
|*  6 |     INDEX RANGE SCAN          | EMP_DEPTNO_IX |      1 |      5 |      5 |00:00:00.01 |       1 |       |       |          |
-----------------------------------------------------------------------------------------------------------------------------------

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

   4 - access("SAL">2000)
   6 - access("DEPTNO"=20)
```

 # MINUS

```
-----------------------------------------------------------------------------------------------------------------------------------
| Id  | Operation                     | Name          | Starts | E-Rows | A-Rows |   A-Time   | Buffers |  OMem |  1Mem | Used-Mem |
-----------------------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT              |               |      1 |        |      3 |00:00:00.01 |       4 |       |       |          |
|   1 |  MINUS                        |               |      1 |        |      3 |00:00:00.01 |       4 |       |       |          |
|   2 |   SORT UNIQUE                 |               |      1 |      6 |      6 |00:00:00.01 |       2 |  2048 |  2048 | 2048  (0)|
|   3 |    TABLE ACCESS BY INDEX ROWID| EMP           |      1 |      6 |      6 |00:00:00.01 |       2 |       |       |          |
|*  4 |     INDEX RANGE SCAN          | EMP_SAL_IX    |      1 |      6 |      6 |00:00:00.01 |       1 |       |       |          |
|   5 |   SORT UNIQUE                 |               |      1 |      5 |      5 |00:00:00.01 |       2 |  2048 |  2048 | 2048  (0)|
|   6 |    TABLE ACCESS BY INDEX ROWID| EMP           |      1 |      5 |      5 |00:00:00.01 |       2 |       |       |          |
|*  7 |     INDEX RANGE SCAN          | EMP_DEPTNO_IX |      1 |      5 |      5 |00:00:00.01 |       1 |       |       |          |
-----------------------------------------------------------------------------------------------------------------------------------

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

   4 - access("SAL">2000)
   7 - access("DEPTNO"=20)
```

 # INTERSECT

```
-----------------------------------------------------------------------------------------------------------------------------------
| Id  | Operation                     | Name          | Starts | E-Rows | A-Rows |   A-Time   | Buffers |  OMem |  1Mem | Used-Mem |
-----------------------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT              |               |      1 |        |      3 |00:00:00.01 |       4 |       |       |          |
|   1 |  INTERSECTION                 |               |      1 |        |      3 |00:00:00.01 |       4 |       |       |          |
|   2 |   SORT UNIQUE                 |               |      1 |      6 |      6 |00:00:00.01 |       2 |  2048 |  2048 | 2048  (0)|
|   3 |    TABLE ACCESS BY INDEX ROWID| EMP           |      1 |      6 |      6 |00:00:00.01 |       2 |       |       |          |
|*  4 |     INDEX RANGE SCAN          | EMP_SAL_IX    |      1 |      6 |      6 |00:00:00.01 |       1 |       |       |          |
|   5 |   SORT UNIQUE                 |               |      1 |      5 |      5 |00:00:00.01 |       2 |  2048 |  2048 | 2048  (0)|
|   6 |    TABLE ACCESS BY INDEX ROWID| EMP           |      1 |      5 |      5 |00:00:00.01 |       2 |       |       |          |
|*  7 |     INDEX RANGE SCAN          | EMP_DEPTNO_IX |      1 |      5 |      5 |00:00:00.01 |       1 |       |       |          |
-----------------------------------------------------------------------------------------------------------------------------------

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

   4 - access("SAL">2000)
   7 - access("DEPTNO"=20)
```

BUFFER SORT

- 임시 테이블 또는 UGA의 정렬 영역을 사용하여 중간 데이터 저장
- 실제 정렬 작업이 수행되지 않음

```
SQL> SELECT *
     FROM dept d, emp e ;

SQL> SELECT *
     FROM dept d CROSS JOIN emp e ;

SQL> @xplan
-------------------------------------------------------------------------------------------------------------------
| Id  | Operation            | Name | Starts | E-Rows | A-Rows |   A-Time   | Buffers |  OMem |  1Mem | Used-Mem |
-------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT     |      |      1 |        |     56 |00:00:00.01 |       7 |       |       |          |
|   1 |  MERGE JOIN CARTESIAN|      |      1 |     56 |     56 |00:00:00.01 |       7 |       |       |          |
|   2 |   TABLE ACCESS FULL  | DEPT |      1 |      4 |      4 |00:00:00.01 |       4 |       |       |          |
|   3 |   BUFFER SORT        |      |      4 |     14 |     56 |00:00:00.01 |       3 |  2048 |  2048 | 2048  (0)|
|   4 |    TABLE ACCESS FULL | EMP  |      1 |     14 |     14 |00:00:00.01 |       3 |       |       |          |
-------------------------------------------------------------------------------------------------------------------
```

4.6. 기타 옵티마이저 연산

- FILTER
- CONCATENATION
- INLIST ITERATOR
- COUNT STOPKEY

FILTER

- 선행 단계에서 반환된 행을 걸러냄

```
SQL> SELECT deptno, SUM(sal)
     FROM emp
     GROUP BY deptno
     HAVING deptno IN (10,30) ;
SQL> @xplan
-------------------------------------------------------------------------------------------------------------------
| Id  | Operation           | Name | Starts | E-Rows | A-Rows |   A-Time   | Buffers |  OMem |  1Mem | Used-Mem |
-------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT    |      |      1 |        |      2 |00:00:00.01 |       3 |       |       |          |
|*  1 |  FILTER             |      |      1 |        |      2 |00:00:00.01 |       3 |       |       |          |
|   2 |   HASH GROUP BY     |      |      1 |      1 |      3 |00:00:00.01 |       3 |   801K|   801K|  667K (0)|
|   3 |    TABLE ACCESS FULL| EMP  |      1 |     14 |     14 |00:00:00.01 |       3 |       |       |          |
-------------------------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   1 - filter(("DEPTNO"=10 OR "DEPTNO"=30))
```

INLIST ITERATOR

- 리스트의 값을 각각 별도로 검색

```
SQL> SELECT empno, ename, sal, deptno
     FROM emp
     WHERE deptno IN (10,20) ;
SQL> @xplan
-----------------------------------------------------------------------------------------------------------------
| Id  | Operation                    | Name          | Starts | E-Rows | A-Rows |   A-Time   | Buffers |
-----------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |               |      1 |        |      8 |00:00:00.01 |       5 |
|   1 |  INLIST ITERATOR             |               |      1 |        |      8 |00:00:00.01 |       5 |
|   2 |   TABLE ACCESS BY INDEX ROWID| EMP           |      2 |      9 |      8 |00:00:00.01 |       5 |
|*  3 |    INDEX RANGE SCAN          | EMP_DEPTNO_IX |      2 |      9 |      8 |00:00:00.01 |       3 |
-----------------------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   3 - access(("DEPTNO"=10 OR "DEPTNO"=20))
```

CONCATENATION

- 둘 이상의 행 집합에서 반환된 행의 합집합 (UNION ALL 연산)

```
SQL> SELECT /*+ use_concat(1) */
            empno, ename, sal, deptno
     FROM emp
     WHERE deptno IN (10,20) ;
SQL> @xplan
-------------------------------------------------------------------------------------------------------
| Id  | Operation                    | Name          | Starts | E-Rows | A-Rows |   A-Time   | Buffers |
-------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |               |      1 |        |      8 |00:00:00.01 |       6 |
|   1 |  CONCATENATION               |               |      1 |        |      8 |00:00:00.01 |       6 |
|   2 |   TABLE ACCESS BY INDEX ROWID| EMP           |      1 |      5 |      5 |00:00:00.01 |       4 |
|*  3 |    INDEX RANGE SCAN          | EMP_DEPTNO_IX |      1 |      5 |      5 |00:00:00.01 |       2 |
|   4 |   TABLE ACCESS BY INDEX ROWID| EMP           |      1 |      5 |      3 |00:00:00.01 |       2 |
|*  5 |    INDEX RANGE SCAN          | EMP_DEPTNO_IX |      1 |      5 |      3 |00:00:00.01 |       1 |
-------------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   3 - access("DEPTNO"=20)
   5 - access("DEPTNO"=10)
```

COUNT STOPKEY

- ROWNUM을 이용하여 행 제한 시 사용
- 지정된 카운트에 도달되면 작업 종료

```
SQL> SELECT *
     FROM emp
     WHERE ROWNUM <= 2 ;
SQL> @xplan
-----------------------------------------------------------------------------------------
| Id  | Operation          | Name | Starts | E-Rows | A-Rows |   A-Time   | Buffers |
-----------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |      |      1 |        |      2 |00:00:00.01 |       4 |
|*  1 |  COUNT STOPKEY     |      |      1 |        |      2 |00:00:00.01 |       4 |
|   2 |   TABLE ACCESS FULL| EMP  |      1 |      2 |      2 |00:00:00.01 |       4 |
-----------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   1 - filter(ROWNUM<=2)
```

5. Optimizer Goal

SQL 명령문은 First Rows, All Rows 의 두 가지 목표를 갖는다. 기본 설정은 All Rows 이지만 상황에 따라 First Rows 의 설정이 필요한 경우도 있으므로 그 차이점을 구분하고 실행 계획의 올바른 선택을 확인한다.

문제. EMP 테이블에서 사원 정보를 출력하며 해당 부서의 급여 합계를 함께 검색하시오.

```
SQL> SELECT a.empno, a.ename, a.sal, a.deptno, b.sum_sal
     FROM emp a, (SELECT deptno, SUM(sal) AS sum_sal
                  FROM emp
                  GROUP BY deptno) b
     WHERE a.deptno = b.deptno ;
SQL> @xplan
```

```
-------------------------------------------------------------------------------------------------------------------
| Id  | Operation                    | Name          | Starts | A-Rows |   A-Time   | Buffers | Used-Mem |
-------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |               |      1 |     14 |00:00:00.01 |       6 |          |
|   1 |  MERGE JOIN                  |               |      1 |     14 |00:00:00.01 |       6 |          |
|   2 |   TABLE ACCESS BY INDEX ROWID| EMP           |      1 |     14 |00:00:00.01 |       4 |          |
|   3 |    INDEX FULL SCAN           | EMP_DEPTNO_IX |      1 |     14 |00:00:00.01 |       2 |          |
|*  4 |   SORT JOIN                  |               |     14 |     14 |00:00:00.01 |       2 | 2048  (0)|
|   5 |    VIEW                      |               |      1 |      3 |00:00:00.01 |       2 |          |
|   6 |     HASH GROUP BY            |               |      1 |      3 |00:00:00.01 |       2 |  641K (0)|
|   7 |      TABLE ACCESS FULL       | EMP           |      1 |     14 |00:00:00.01 |       2 |          |
-------------------------------------------------------------------------------------------------------------------
```

```
SQL> SELECT a.empno, a.ename, a.sal, a.deptno,
            (SELECT SUM(sal)
             FROM emp
             WHERE deptno = a.deptno) AS sum_sal
     FROM emp a ;
SQL> @xplan
```

```
-------------------------------------------------------------------------------------------------
| Id  | Operation                    | Name          | Starts | A-Rows |   A-Time   | Buffers |
-------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |               |      1 |     14 |00:00:00.01 |       3 |
|   1 |  SORT AGGREGATE              |               |      3 |      3 |00:00:00.01 |       4 |
|   2 |   TABLE ACCESS BY INDEX ROWID| EMP           |      3 |     14 |00:00:00.01 |       4 |
|*  3 |    INDEX RANGE SCAN          | EMP_DEPTNO_IX |      3 |     14 |00:00:00.01 |       2 |
|   4 |  TABLE ACCESS FULL           | EMP           |      1 |     14 |00:00:00.01 |       3 |
-------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------

   3 - access("DEPTNO"=:B1)
```

두 문장 모두 문제를 해결할 수 있는 문장이다. 하지만 문장의 형태 및 실행 계획은 전혀 다르게 접근한다. 만약 두 문장 중 하나를 선택하여 사용한다면 어떤 문장을 사용하겠는가?

지금 상황에서 고민해야 하는 것이 바로 "Optimizer Mode (Goal)" 이다.

- 첫 번째 문장의 실행 계획

```
-------------------------------------------------------------------------------------------------------
| Id  | Operation                    | Name          | Starts | A-Rows |   A-Time   | Buffers | Used-Mem |
-------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |               |      1 |     14 |00:00:00.01 |       6 |          |
|   1 |  MERGE JOIN                  |               |      1 |     14 |00:00:00.01 |       6 |          |
|   2 |   TABLE ACCESS BY INDEX ROWID| EMP           |      1 |     14 |00:00:00.01 |       4 |          |
|   3 |    INDEX FULL SCAN           | EMP_DEPTNO_IX |      1 |     14 |00:00:00.01 |       2 |          |
|*  4 |   SORT JOIN                  |               |     14 |     14 |00:00:00.01 |       2 | 2048  (0)|
|   5 |    VIEW                      |               |      1 |      3 |00:00:00.01 |       2 |          |
|   6 |     HASH GROUP BY            |               |      1 |      3 |00:00:00.01 |       2 |  641K (0)|
|   7 |      TABLE ACCESS FULL       | EMP           |      1 |     14 |00:00:00.01 |       2 |          |
-------------------------------------------------------------------------------------------------------
```

첫 번째 문장은 Inline View 의 결과를 계산하고 Sort Merge Join을 이용하여 최종 결과를 만들었다. 즉, Inline View 의 결과가 생성되기 전에는 조인의 결과를 만들 수 없다. 만약 EMP 테이블에 대량의 데이터가 저장되어 있고, 좀 더 복잡한 형식의 Grouping을 수행해야 한다면 조인을 수행하기 전 많은 대기 시간이 필요할 수 있다. 때문에 조건에 만족하는 첫 번째 행의 결과는 먼저 검색 될 수 없다. 하지만 실행 계획의 각 단계는 각각 한 번씩만 수행되고 단계별 반복 실행이 없으므로 모든 행이 최소한의 작업량을 통해 빨리 나올 수 있는 실행 계획이다.

- 두 번째 문장의 실행 계획

```
-----------------------------------------------------------------------------------------------------
| Id  | Operation                    | Name          | Starts | A-Rows |   A-Time   | Buffers |
-----------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |               |      1 |     14 |00:00:00.01 |       3 |
|   1 |  SORT AGGREGATE              |               |      3 |      3 |00:00:00.01 |       4 |
|   2 |   TABLE ACCESS BY INDEX ROWID| EMP           |      3 |     14 |00:00:00.01 |       4 |
|*  3 |    INDEX RANGE SCAN          | EMP_DEPTNO_IX |      3 |     14 |00:00:00.01 |       2 |
|   4 |  TABLE ACCESS FULL           | EMP           |      1 |     14 |00:00:00.01 |       3 |
-----------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------

   3 - access("DEPTNO"=:B1)
```

두 번째 문장은 SELECT 절의 Correlated Subquery를 사용한 문장인데 이러한 문장의 실행 계획은 일반적인 실행 계획의 읽는 순서와 좀 다르게 4-Loop(3-2-1) 순으로 실행된다. 즉, 메인 쿼리의 테이블을 먼저 검색하여 하나씩 후보행의 부서 번호를 서브쿼리에 공급하고 3-2-1 단계를 총 3번의 반복 실행을 통해서 최종 결과를 검색하였다. 이때 4번 단계의 작업에 3개의 I/O 가 수행되었고 3-2-1 의 반복 처리를 진행하기 위해 4개의 I/O 가 수행되어 총 7개의 I/O 가 발생하였다. 해당 문장은 전체 부서별 급여의 합계를 계산하지 않고, 메인 쿼리의 테이블을 검색하면서 필요한 후보 값 만의 급여 합계를 계산하기 때문에 첫 번째 행의 검색 결과는 더 빨리 만들어 낼 수 있다. 하지만 후보행이 많고, 공급되는 후보 값의 중복이 거의 없을 때는 서브쿼리의 반복 실행이 많아지므로 전체 작업량은 오히려 늘어날 수 있다.

SELECT 문장의 결과는 두 가지 목적으로 사용될 수 있다. 하나는 검색 결과를 화면으로 가져오는 것이고 또 하나는 내부적인 작업 처리를 위해 테이블을 검색하는 경우이다.

즉, 검색 결과가 일괄처리를 위해 프로시저와 같은 서브프로그램 내에서만 사용되고 그 결과를 화면에 가져올 필요가 없는 문장은 검색된 행의 개수와 상관없이 최소한의 작업량을 가지며 가능한 한 빨리 모든 결과가 함께 검색 될 수 있도록 해야 한다. 하지만 검색 결과를 화면에 가져와야 할 경우에는 페이지 단위 처리에 만족하는 문장의 실행 계획이 필요하다.

예를 들어 쿼리 결과가 100,000개의 행을 갖는다고 가정할 때 그 모든 행이 하나의 페이지에 한 번에 검색하는 것이 의미가 있을까? SQL Developer 와 같은 툴은 기본적으로 검색 결과를 모두 가져오는 것이 아니고 한 번에 50개의 행씩 보여주게 된다. SQL*Plus 도 PAUSE 설정을 통해 페이지 단위의 검색을 지원한다.

SELECT 문장을 실행하는 환경에 따라서 조건에 만족하는 모든 행을 빠르게 가져와야 할지, 아니면 조건에 만족하는 최초 n개의 행을 먼저 가져와야 할지를 결정해야 한다. 이러한 설정은 파라미터 optimizer_mode 를 통해서 실행 계획을 생성할 때도 영향을 줄 수 있다.

```
SQL> ALTER SESSION SET optimizer_mode = first_rows ;
SQL> SELECT a.empno, a.ename, a.sal, a.deptno, b.sum_sal
     FROM emp a, (SELECT deptno, SUM(sal) AS sum_sal
                  FROM emp
                  GROUP BY deptno) b
     WHERE a.deptno = b.deptno ;
SQL> @xplan
```

```
-------------------------------------------------------------------------------------------------------------------
| Id  | Operation                    | Name          | Starts | A-Rows |   A-Time   | Buffers | Used-Mem |
-------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |               |      1 |     14 |00:00:00.01 |       7 |          |
|   1 |  NESTED LOOPS                |               |      1 |     14 |00:00:00.01 |       7 |          |
|   2 |   NESTED LOOPS               |               |      1 |     14 |00:00:00.01 |       5 |          |
|   3 |    VIEW                      |               |      1 |      3 |00:00:00.01 |       2 |          |
|   4 |     HASH GROUP BY            |               |      1 |      3 |00:00:00.01 |       2 |  675K (0)|
|   5 |      TABLE ACCESS FULL       | EMP           |      1 |     14 |00:00:00.01 |       2 |          |
|*  6 |    INDEX RANGE SCAN          | EMP_DEPTNO_IX |      3 |     14 |00:00:00.01 |       3 |          |
|   7 |   TABLE ACCESS BY INDEX ROWID| EMP           |     14 |     14 |00:00:00.01 |       2 |          |
-------------------------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   6 - access("A"."DEPTNO"="B"."DEPTNO")
```

ALL_ROWS 설정 때와는 다르게 Nested Loops Join을 수행 한 결과를 확인할 수 있다. Sort Merge Join에 비해 작업량은 늘어났으나 Nested Loops Join을 통해 조인된 첫 번째 행은 더 빠르게 검색할 수 있게 했다. 하지만 여전히 Inline View의 결과를 먼저 검색하고 있기 때문에 Inline View 의 작업량이 많으면 조인 수행 전 대기 시간이 필요한 상황이다.

```
SQL> SELECT a.empno, a.ename, a.sal, a.deptno,
           (SELECT SUM(sal)
            FROM emp
            WHERE deptno = a.deptno) AS sum_sal
     FROM emp a ;
SQL> @xplan
```

```
-----------------------------------------------------------------------------------------------------
| Id  | Operation                    | Name          | Starts | A-Rows |   A-Time   | Buffers |
-----------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |               |      1 |     14 |00:00:00.01 |       3 |
|   1 |  SORT AGGREGATE              |               |      3 |      3 |00:00:00.01 |       4 |
|   2 |   TABLE ACCESS BY INDEX ROWID| EMP           |      3 |     14 |00:00:00.01 |       4 |
|*  3 |    INDEX RANGE SCAN          | EMP_DEPTNO_IX |      3 |     14 |00:00:00.01 |       2 |
|   4 |  TABLE ACCESS FULL           | EMP           |      1 |     14 |00:00:00.01 |       3 |
-----------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   3 - access("DEPTNO"=:B1)
```

```
SQL> ALTER SESSION SET optimizer_mode = all_rows ;
```

두 번째 문장은 문장 자체가 First Rows 상황에 최적화된 문장이므로 처음의 실행 계획과 변동 사항이 없다. 즉, 항상 조건에 만족하는 첫 번째 행은 먼저 검색 가능한 상황이다. (Oracle 12c 부터는 SELECT 리스트의 Subquery도 Unnesting이 가능하다. 때문에 OPTIMIZER_MODE의 설정에 따라 조인의 종류가 바뀔 수도 있으므로 실행 계획의 확인이 필요하다.)

SQL 문장 작성 시 항상 어느 특정 패턴의 문장 작성 방법을 고집하는 것은 올바른 문장 작성 방법이 될 수 없다. 현재 SQL 문장이 실행되는 환경이 어떤 환경인지, 파라미터의 설정이 어떤 값을 가지고 있는지, 실행 계획이 어떻게 만들어졌는지 확인하지 않는다면 결과는 나올지 몰라도 성능은 보장할 수 없다.

다음의 문장은 동일한 결과를 검색하지만 ALL_ROWS, FIRST_ROWS 설정에 따라 다른 실행 계획, 다른 속도를 확인할 수 있다. SQL Developer와 같은 환경에서 각 문장을 실행하면서 첫 번째 페이지를 검색하는데 걸리는 시간을 비교하고, 사용된 실행 계획을 확인한다. 또한 SQL*Plus에서도 동일 문장을 실행하면서 전체 결과를 검색하는데 걸리는 시간과 성능 차이도 비교한다.

```
SQL> SELECT c.cust_id, c.cust_last_name, s.sum_amount
     FROM customers c, (SELECT cust_id, SUM(amount_sold) AS sum_amount
                        FROM sales
                        GROUP BY cust_id) s
     WHERE c.cust_id = s.cust_id (+) ;

SQL> SELECT c.cust_id, c.cust_last_name, (SELECT SUM(amount_sold)
                                          FROM sales
                                          WHERE  cust_id  =  c.cust_id)  AS
sum_amount
     FROM customers c ;
```

> 결론

First Rows 환경 : 조건에 만족하는 최초 n개의 행을 먼저 검색 (Page 단위 처리)
All Rows 환경 : 조건에 만족하는 모든 행을 검색

First Rows 환경

- 조인 수행 시 Nested Loops Join을 우선 고려
- 집계성 데이터를 검색할 경우 Correlated Subquery 사용 고려
- Subquery 사용 시 Query Transformation 여부 확인하여 최적화

All Rows 환경

- 조인 수행 시 Hash Join을 우선 고려 (작업량이 많을 경우)
- 집계성 데이터를 검색할 경우 Inline View 또는 분석 함수 사용 고려
- Subquery 사용 시 Query Transformation 여부 확인하여 최적화

6. 인덱스 사용

6.1. 인덱스 이해

테이블에 저장된 데이터는 특정 컬럼의 값을 기준으로 정렬되어 저장되지도 않습니다. 때문에 다음과 같이 특정 조건에 만족하는 행을 검색하고자 할 때 인덱스를 사용하지 못하면 "TABLE FULL SCAN"이라는 실행 계획이 사용됩니다.

```
SQL> DROP INDEX emp_ename_ix ;
SQL> SELECT empno, ename, sal, deptno, ROWID
     FROM emp
     WHERE ename = 'SCOTT' ;
     EMPNO ENAME             SAL     DEPTNO ROWID
---------- ---------- ---------- ---------- ------------------
      7788 SCOTT            3000         20 AAASaMAAAAAAACmAAA

-------------------------------------------------------------------------------
| Id  | Operation         | Name | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT  |      |     1 |    58 |     3   (0)| 00:00:01 |
|*  1 |  TABLE ACCESS FULL| EMP  |     1 |    58 |     3   (0)| 00:00:01 |
-------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   1 - filter("ENAME"='SCOTT')

SQL> SELECT empno, ename, sal, deptno, ROWID
     FROM emp
     WHERE ROWID = 'AAASaMAAAAAAACmAAA' ;             -- 확인한 ROWID 입력
--------------------------------------------------------------------------------------
| Id  | Operation                   | Name | Rows  | Bytes | Cost (%CPU)| Time     |
--------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT            |      |     1 |    58 |     1   (0)| 00:00:01 |
|   1 |  TABLE ACCESS BY USER ROWID| EMP  |     1 |    58 |     1   (0)| 00:00:01 |
--------------------------------------------------------------------------------------
```

"TABLE FULL SCAN"은 테이블의 모든 행을 액세스하여 조건식에 TRUE인 행을 필터링하여 검색하는 방법으로 대량의 데이터를 검색할 경우에는 유리한 실행 계획이지만 소량의 데이터를 검색할 때는 성능 저하의 원인이 되기도 합니다. 만약 두 번째 문장처럼, 행이 저장된 주소(ROWID)를 조건으로 사용할 수 있다면 불필요한 "TABLE FULL SCAN"을 피하고 하나의 행만을 읽을 수 있으므로 보다 빠른 검색이 가능해집니다. 하지만 ROWID를 직접 외워서 조건식에 사용하기는 매우 어렵습니다. 때문에 조건식에 자주 사용되는 컬럼에는 인덱스를 생성하고, 생성된 인덱스를 이용하여 보다 빠른 검색을 가능하게 합니다.

인덱스는 NULL을 제외한 컬럼에 저장된 값과 주소(ROWID)를 저장하고 있는 데이터베이스의 객체입니다. 또한 일반 테이블과는 다르게 항상 정렬된 상태를 유지하고 있어서 부분 검색만으로 원하는 데이터를 찾을 수 있습니다.

```
SQL> CREATE INDEX emp_ename_ix ON emp(ename) ;
SQL> SELECT ename, rowid
     FROM emp
     WHERE ename IS NOT NULL
     ORDER BY ename, rowid ;                          -- 인덱스에 저장된 데이터
ENAME      ROWID
---------- ------------------
ADAMS      AAASaMAAAAAAACmAAD
ALLEN      AAASaMAAAAAAACmAAE
BLAKE      AAASaMAAAAAAACmAAH
CLARK      AAASaMAAAAAAACmAAI
FORD       AAASaMAAAAAAACmAAL
JAMES      AAASaMAAAAAAACmAAK
JONES      AAASaMAAAAAAACmAAB
KING       AAASaMAAAAAAACmAAN
MARTIN     AAASaMAAAAAAACmAAG
MILLER     AAASaMAAAAAAACmAAM
SCOTT      AAASaMAAAAAAACmAAA
...
14 rows selected.
```

```
SQL> SELECT empno, ename, sal, deptno, ROWID
     FROM emp
     WHERE ename = 'SCOTT' ;
-----------------------------------------------------------------------------------------
¦ Id  ¦ Operation                   ¦ Name         ¦ Rows  ¦ Bytes ¦ Cost (%CPU)¦ Time     ¦
-----------------------------------------------------------------------------------------
¦   0 ¦ SELECT STATEMENT            ¦              ¦     1 ¦    58 ¦     2   (0)¦ 00:00:01 ¦
¦   1 ¦  TABLE ACCESS BY INDEX ROWID¦ EMP          ¦     1 ¦    58 ¦     2   (0)¦ 00:00:01 ¦
¦* 2 ¦   INDEX RANGE SCAN           ¦ EMP_ENAME_IX ¦     1 ¦       ¦     1   (0)¦ 00:00:01 ¦
-----------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   2 - access("ENAME"='SCOTT')
```

앞서 실행했던 문장과 동일한 문장이지만 생성된 인덱스를 이용하여 "TABLE FULL SCAN" 없이도 조건에 만족하는 행을 검색할 수 있습니다. 이렇게 인덱스는 조건식에 만족하는 데이터를 보다 빠르게 검색할 수 있도록 도움을 주는 객체이며 많은 업무에서 다양하게 사용되고 있습니다. 하지만 인덱스가 존재하더라도 SQL의 작성 방법에 따라 인덱스를 사용할 수 없거나, 사용하더라도 사용 방법이 적절하지 못할 수 있습니다. 때문에 다음 몇 가지 상황들을 확인하여 인덱스를 사용할 수 있는 문장 작성 방법을 숙지한다면 개발 단계에서 일어날 수 있는 성능상 실수를 조금이나마 줄일 수 있게 될 것입니다.

6.2. 인덱스를 사용할 수 없는 SQL

인덱스 사용이 반드시 필요한 것은 아닙니다. 하지만 일반적으로 인덱스 사용을 원하는 경우가 많습니다. 만약 작성하는 문장이 인덱스를 사용할 수 있어야 한다면 다음의 주의 사항을 확인하시기 바랍니다.

1. IS NULL은 항상 인덱스를 사용할 수 없습니다.
2. IS NOT NULL은 인덱스 사용 시 "INDEX FULL SCAN"만 가능합니다.
3. 인덱스가 존재하는 컬럼이 조건식에 존재해야 합니다.
4. 명시적, 암시적 컬럼 가공이 있어서는 안됩니다.
5. 부정형 비교보다는 긍정형 조건식을 사용합니다.
6. "%"를 앞에 사용한 LIKE는 인덱스 사용 시 "INDEX FULL SCAN"만 가능합니다.

개발 경험이 있는 분들은 한 번쯤 들어봤을 얘기들입니다. 하지만 약간 다른 부분도 있을 수 있습니다. 예전에는 무조건 인덱스를 사용하지 못한다고 했던 부분들이 오라클 버전이 올라가면서 인덱스를 사용할 수 있는 상황으로 바뀐 부분도 있기 때문입니다. 때문에 무조건적인 암기가 아닌 실행 계획을 확인하여 현재 상황에 맞는 문장의 작성 방법을 숙지하셔야 합니다.

그럼 실제 문장과 함께 인덱스를 사용하는 문장과 사용하지 못하는 문장의 차이점을 구분해 볼까요?

6.2.1. IS NULL, IS NOT NULL 비교

```
SQL> SELECT empno, ename, mgr, deptno
     FROM emp
     WHERE mgr IS NULL ;
-------------------------------------------------------------------------------
| Id  | Operation         | Name | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT  |      |     1 |    17 |     3   (0)| 00:00:01 |
|*  1 |  TABLE ACCESS FULL| EMP  |     1 |    17 |     3   (0)| 00:00:01 |
-------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   1 - filter("MGR" IS NULL)
```

인덱스는 NULL을 제외한 컬럼 값과 ROWID를 저장하고 있습니다. 그래서 "IS NULL" 비교를 사용할 경우에는 항상 인덱스를 사용할 수 없습니다. 다음과 같이 힌트를[1] 이용하더라도 인덱스를 이용할 수 있는 방법은 없습니다. NULL의 주소가 저장되어 있지 않기 때문에 인덱스 사용은 항상 불가능합니다.

```
SQL> SELECT /*+ index(emp emp_mgr_ix) */ empno, ename, mgr, deptno
     FROM emp
     WHERE mgr IS NULL ;
-------------------------------------------------------------------------------
| Id  | Operation         | Name | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT  |      |     1 |    17 |     3   (0)| 00:00:01 |
|*  1 |  TABLE ACCESS FULL| EMP  |     1 |    17 |     3   (0)| 00:00:01 |
-------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   1 - filter("MGR" IS NULL)
```

하지만 "IS NULL" 조건에 만족하는 행이 소량일 경우에는 인덱스를 사용하는 것이 성

[1] 힌트: 실행 계획에 영향을 주기 위해 사용되는 사용자 정의의 지시어

능상 유리할 수 있습니다. 때문에 성능을 위해서라면 NULL의 개수가 많지 않은 컬럼은 NULL 대신 별도의 대체 값을 저장하고, 대체 값을 검색할 수 있도록 조건식을 작성하는 것이 필요합니다.

```
SQL> UPDATE emp
     SET mgr = 0
     WHERE mgr IS NULL ;
1 row updated.

SQL> SELECT empno, ename, sal, comm, deptno
     FROM emp
     WHERE mgr = 0 ;
-------------------------------------------------------------------------------
| Id  | Operation                   | Name       | Rows  | Bytes | Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT            |            |     1 |    23 | 00:00:01 |
|   1 |  TABLE ACCESS BY INDEX ROWID| EMP        |     1 |    23 | 00:00:01 |
|*  2 |   INDEX RANGE SCAN          | EMP_MGR_IX |     1 |       | 00:00:01 |
-------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   2 - access("MGR"=0)

SQL> ROLLBACK ;
```

NULL 값을 0으로 UPDATE 후에 "MGR = 0" 조건식을 사용하니까 인덱스를 사용할 수 있는 실행 계획이 생성되었습니다. 해당 컬럼의 인덱스를 이용할 필요가 없다면 "IS NULL" 조건은 그대로 사용하셔도 됩니다. 하지만 인덱스 사용이 필요하다면 "IS NULL" 조건은 반드시 제거되어야 합니다.

그렇다면 반대로 "IS NOT NULL" 비교를 수행할 때는 어떨까요?

```
SQL> SELECT empno, ename, comm, deptno
     FROM emp
     WHERE comm IS NOT NULL ;
-------------------------------------------------------------------------------
| Id  | Operation                   | Name        | Rows  | Bytes | Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT            |             |     4 |    60 | 00:00:01 |
|   1 |  TABLE ACCESS BY INDEX ROWID| EMP         |     4 |    60 | 00:00:01 |
|*  2 |   INDEX FULL SCAN           | EMP_COMM_IX |     4 |       | 00:00:01 |
-------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   2 - filter("COMM" IS NOT NULL)
```

인덱스를 사용하는 실행 계획이 확인됩니다. 단, "INDEX RANGE SCAN"이 아닌 "INDEX FULL SCAN"을 이용했다는 것에 주의하세요. NULL의 주소는 인덱스에 저장되어 있지 않지만 NULL이 아닌 값을 비교하는 것은 인덱스를 사용할 수 있습니다. 대신 인덱스에 저장된 모든 값을 검색해야 하므로 부분적 탐색이 아닌 "INDEX FULL SCAN"을 사용할 수밖에 없습니다. 여기서 "INDEX RANGE SCAN"과 "INDEX FULL SCAN" 의 성능상 차이점은 언급하지 않겠습니다. 다만 "INDEX FULL SCAN" 작업이 대량의 데이터를 검색할 때 사용된다면 "TABLE FULL SCAN" 보다 성능은 떨어질 수 있습니다. 그래서 "IS NOT NULL" 조건에 만족하는 데이터가 많을 경우에는 "TABLE FULL SCAN"이 자동으로 사용되기도 합니다.

"IS NULL" 조건은 인덱스를 사용하지 못 합니다. 하지만 "IS NOT NULL"은 인덱스를 사용할 수 있습니다. "INDEX FULL SCAN"이 성능상 유리하다면 자동으로 선택되기도 하고, 힌트를 통해 유도할 수 있습니다.

6.2.2. WHERE 절 부재

```
SQL> SELECT deptno, SUM(sal)
     FROM emp
     GROUP BY deptno
     HAVING deptno = 10 ;
-------------------------------------------------------------
| Id  | Operation           | Name | Rows  | Bytes | Time     |
-------------------------------------------------------------
|   0 | SELECT STATEMENT    |      |     1 |     7 | 00:00:01 |
|*  1 |  FILTER             |      |       |       |          |
|   2 |   HASH GROUP BY     |      |     1 |     7 | 00:00:01 |
|   3 |    TABLE ACCESS FULL| EMP  |    14 |    98 | 00:00:01 |
-------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   1 - filter("DEPTNO"=10)
```

HAVING 절은 일반적으로 그룹 함수 결과를 비교할 때 사용합니다. 물론 일반 컬럼의 조건도 비교할 수 있지만 인덱스를 사용할 수는 없습니다. 이는 문장의 수행 순서와 관계가 있습니다. 오라클은 GROUP BY 절을 수행하고, HAVING 절을 비교합니다. 이때 GROUP BY를 수행하기 위한 행의 제한이 없다면(WHERE 절이 없다면) 테이블의 전체 데이터를 모두 액세스하고, HAVING 절의 조건 비교는 마지막에 수행할 수밖에 없습니다. 이는 불필요한 데이터까지 GROUP BY 절에 포함되면서 성능 저하의 원인이 됩니다. 때문에 문장은 다음과 같이 작성되어야 합니다.

```
SQL> SELECT deptno, SUM(sal)
     FROM emp
     WHERE deptno = 10
     GROUP BY deptno ;
```

```
-------------------------------------------------------------------------------
| Id  | Operation                    | Name          | Rows  | Bytes | Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |               |     1 |     7 | 00:00:01 |
|   1 |  SORT GROUP BY NOSORT        |               |     1 |     7 | 00:00:01 |
|   2 |   TABLE ACCESS BY INDEX ROWID| EMP           |     5 |    35 | 00:00:01 |
|*  3 |    INDEX RANGE SCAN          | EMP_DEPTNO_IX |     5 |       | 00:00:01 |
-------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   3 - access("DEPTNO"=10)
```

WHERE 절을 사용하는 문장은 조건에 만족하는 행을 먼저 비교하여 GROUP BY를 수행할 수 있으므로 인덱스를 사용하는 실행 계획이 사용되었습니다. 즉, HAVING 절은 그룹 함수의 조건 비교를 수행할 경우에만 사용하는 것이 좋습니다.

또한 경우에 따라 힌트를 이용하여 인덱스 사용을 강제화시키기도 합니다. 아래 예제는 정렬 상태를 유지하는 인덱스를 이용하여 ORDER BY 절을 대체하기 위한 상황입니다. 하지만 실행 계획은 지정된 인덱스를 사용하지 않고 "TABLE FULL SCAN"을 사용하였습니다.

```
SQL> SELECT /*+ index(e emp_sal_ix) */ empno, ename, sal
     FROM emp e
     ORDER BY sal ;
-------------------------------------------------------------------
| Id  | Operation          | Name | Rows  | Bytes | Time     |
-------------------------------------------------------------------
|   0 | SELECT STATEMENT   |      |    14 |   196 | 00:00:01 |
|   1 |  SORT ORDER BY     |      |    14 |   196 | 00:00:01 |
|   2 |   TABLE ACCESS FULL| EMP  |    14 |   196 | 00:00:01 |
-------------------------------------------------------------------
```

힌트까지 사용했지만 인덱스를 사용하지 않은 이유는 무엇일까요?

원인은 SAL 컬럼에 존재할 수 있는 NULL 때문에 그렇습니다. 인덱스에는 NULL의 주소가 저장되어 있지 않다고 했습니다. 때문에 인덱스를 이용하면 NULL을 가지고 있는 데이터는 검색이 될 수 없습니다. SAL 컬럼에 조건이 없다면 혹시나 존재할 수 있는 NULL도 검색이 가능해야 하므로 지정한 힌트는 사용되지 못하고 "TABLE FULL SCAN"을 사용하였습니다. 만약 정렬 작업을 대신할 인덱스 사용을 원하는 경우라면 다음과 같이 모든 행이 만족할 수 있는 조건식이 필요합니다. (인덱스 사용을 위해 모든 행을 검색 가능하게 하는 조건을 더미 조건이라고도 합니다.)

```
SQL> SELECT /*+ index(e emp_sal_ix) */ empno, ename, sal
     FROM emp e
     WHERE sal > 0
     ORDER BY sal ;
-------------------------------------------------------------------------------
| Id  | Operation                   | Name       | Rows  | Bytes | Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT            |            |    14 |   196 | 00:00:01 |
|   1 |  TABLE ACCESS BY INDEX ROWID| EMP        |    14 |   196 | 00:00:01 |
|*  2 |   INDEX RANGE SCAN          | EMP_SAL_IX |    14 |       | 00:00:01 |
-------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   2 - access("SAL">0)
```

정렬 상태를 이미 보장하는 인덱스를 사용했기 때문에 실행 계획상 "SORT ORDER BY"가 사라진 것을 확인할 수 있습니다.

하지만 해당 문장을 실제 실행하셨다면 SAL 컬럼에는 NULL을 가지고 있는 행은 없다는 것을 알 수 있습니다. 그렇다면 위와 같은 조건식을 추가하지 않아도 인덱스는 사용할 수 있어야 하는 것이 아닐까요?

제약 조건이 필요한 상황이 바로 이때입니다. SAL 컬럼에 제약 조건 "NOT NULL"을 사용하고 있다면 NULL은 절대 입력될 수 없습니다. 하지만 제약 조건이 없는 상황이라면 DML 명령문을 통해 언제든지 SAL 컬럼 값은 NULL로 바뀔 수도 있습니다. 결국 WHERE 절을 지정하지 않으면, 지금은 NULL이 없어도 NULL이 저장될 수 있다는 가정이 존재하므로 지정한 인덱스 사용은 불가능합니다.

```
SQL> ALTER TABLE emp MODIFY sal NOT NULL ;
SQL> SELECT /*+ index(e emp_sal_ix) */ empno, ename, sal
     FROM emp e
     ORDER BY sal ;
-------------------------------------------------------------------------------
| Id  | Operation                   | Name       | Rows  | Bytes | Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT            |            |    14 |   196 | 00:00:01 |
|   1 |  TABLE ACCESS BY INDEX ROWID| EMP        |    14 |   196 | 00:00:01 |
|   2 |   INDEX FULL SCAN           | EMP_SAL_IX |    14 |       | 00:00:01 |
-------------------------------------------------------------------------------
```

컬럼에 "NOT NULL" 제약 조건을 정의한 후로는 WHERE 절의 더미 조건을 제거해도 인덱스를 사용할 수 있습니다. 하지만 모든 컬럼에 제약 조건을 정의할 수는 없습니다. 때문에 필요에 따라 WHERE 절의 더미 조건을 사용하는 것이 실제로는 더 많이 사용됩니다. 즉, 특정 컬럼의 인덱스를 사용하고자 한다면 WHERE 절에 조건식을 지정하는 것이 제약 조건과 상관없이 인덱스를 사용할 수 있습니다.

힌트는 반드시 지켜야 하는 규칙은 아닙니다. 만약 정상적인 결과를 보장할 수 없는 실행 계획을 요구하고 있다면 오라클 서버는 해당 힌트를 무시하고 결과를 검색할 수 있는 실행 계획을 사용합니다. 때문에 힌트를 지정하는 문장을 사용하신다면 항상 원하는 실행 계획이 사용되었는지를 확인해야 합니다.

6.2.3. 컬럼 가공 (표현식 일부로 사용된 컬럼)

```
SQL> SELECT empno, ename, sal, deptno
     FROM emp
     WHERE SUBSTR(ename, 2, 1) = 'A' ;
------------------------------------------------------------
| Id  | Operation         | Name | Rows  | Bytes | Time     |
------------------------------------------------------------
|   0 | SELECT STATEMENT  |      |     1 |    17 | 00:00:01 |
|*  1 |  TABLE ACCESS FULL| EMP  |     1 |    17 | 00:00:01 |
------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   1 - filter(SUBSTR("ENAME",2,1)='A')
```

컬럼이 표현식 일부로 사용된다면 "INDEX RANGE SCAN"을 수행할 수는 없습니다. 인덱스는 컬럼에 저장된 값이 그대로 저장되어 있기 때문에 가공된 값을 비교할 때는 인덱스 사용을 못 합니다. 만약 컬럼에 "NOT NULL" 제약 조건이 있다면 "INDEX FULL SCAN" 실행 계획이 사용될 수도 있지만, 오히려 작업량이 늘어나는 경우가 많아 "TABLE FULL SCAN"이 더 많이 사용됩니다.

때문에 컬럼을 가공하지 않고 동일 결과를 검색할 수 있는 경우라면 조건식을 수정하여 인덱스를 사용할 수 있는 문장을 작성해야 합니다.

```
SQL> SELECT empno, ename, sal
     FROM emp
     WHERE SUBSTR(ename,1,1) ='A' ;
```

```
SQL> SELECT empno, ename, sal
     FROM emp
     WHERE ename LIKE 'A%' ;
```

```
SQL> SELECT empno, ename, sal
     FROM emp
     WHERE sal * 12 > 20000 ;
```

```
SQL> SELECT empno, ename, sal
     FROM emp
     WHERE sal > 20000 / 12 ;
```

하지만 문장을 수정할 수 없는 상황이라면 어떻게 해야 할까요?

```
SQL> CREATE INDEX emp_ename_fbi ON emp(SUBSTR(ename,2,1));

SQL> SELECT SUBSTR(ename,2,1), rowid
     FROM emp
     WHERE SUBSTR(ename,2,1) IS NOT NULL
     ORDER BY 1, 2;
SUBS ROWID
---- ------------------
A    AAASUpAAAAAAACbAAC
A    AAASUpAAAAAAACbAAE
A    AAASUpAAAAAAACbAAL
C    AAASUpAAAAAAACbAAH
...
14 rows selected.

SQL> SELECT empno, ename, sal, deptno
     FROM emp
     WHERE SUBSTR(ename, 2, 1) = 'A' ;
-------------------------------------------------------------------------------
| Id  | Operation                   | Name          | Rows  | Bytes | Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT            |               |     1 |    21 | 00:00:01 |
|   1 |  TABLE ACCESS BY INDEX ROWID| EMP           |     1 |    21 | 00:00:01 |
|*  2 |   INDEX RANGE SCAN          | EMP_ENAME_FBI |     1 |       | 00:00:01 |
-------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   2 - access(SUBSTR("ENAME",2,1)='A')

SQL> DROP INDEX emp_ename_fbi ;
```

이때 필요한 것이 Function based index(함수 기반 인덱스) 입니다. 표현식의 결과를 정렬하여 저장하고 있다면 컬럼의 가공이 있어도 "INDEX RANGE SCAN"을 사용할 수 있습니다. 하지만 추가적인 저장 공간이 필요하고, 유지 관리 비용이 늘어나므로 최후의 방법으로 사용해야 합니다.

또한 사용자가 컬럼을 가공하진 않아도 암시적인 형 변환 때문에 인덱스를 사용하지 못하는 경우도 존재합니다.

```
SQL> SELECT empno, ename, sal, deptno
     FROM emp
     WHERE empno LIKE '77%' ;
-----------------------------------------------------------
| Id  | Operation         | Name | Rows  | Bytes | Time     |
-----------------------------------------------------------
|   0 | SELECT STATEMENT  |      |     1 |    17 | 00:00:01 |
|*  1 |  TABLE ACCESS FULL| EMP  |     1 |    17 | 00:00:01 |
-----------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   1 - filter(TO_CHAR("EMPNO") LIKE '77%')

SQL> SELECT empno, ename, sal, deptno
     FROM emp
     WHERE empno BETWEEN 7700 AND 7799 ;
-------------------------------------------------------------------------------------
| Id  | Operation                   | Name         | Rows  | Bytes | Time     |
-------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT            |              |     4 |    68 | 00:00:01 |
|   1 |  TABLE ACCESS BY INDEX ROWID| EMP          |     4 |    68 | 00:00:01 |
|*  2 |   INDEX RANGE SCAN          | EMP_EMPNO_IX |     4 |       | 00:00:01 |
-------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   2 - access("EMPNO">=7700 AND "EMPNO"<=7799)
```

LIKE는 문자의 패턴을 비교할 때 사용합니다. 이때 비교되는 컬럼 타입이 문자형이 아닐 경우에는 형 변환 함수를 사용하여 문자형으로 변경합니다. ("Predicate Information" 부분에 TO_CHAR 함수가 사용된 것을 확인할 수 있습니다.) 암시적인 형 변환은 다양한 상황에서 발생하므로, 예상했던 인덱스를 사용하지 못한다면 사용자가 지정하지 않은 컬럼 가공이 없는지 확인하고 인덱스를 사용할 수 있도록 조건식을 수정합니다.

6.2.4. INDEX RANGE SCAN 이 불가능한 조건

```
SQL> SELECT  empno, ename, sal, deptno
     FROM emp
     WHERE deptno != 20 ;
-----------------------------------------------------------
| Id  | Operation         | Name | Rows  | Bytes | Time     |
-----------------------------------------------------------
|   0 | SELECT STATEMENT  |      |     9 |   153 | 00:00:01 |
|*  1 |  TABLE ACCESS FULL| EMP  |     9 |   153 | 00:00:01 |
-----------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   1 - filter("DEPTNO"<>20)

SQL> SELECT /*+ index(emp(deptno)) */ empno, ename, sal, deptno
     FROM emp
     WHERE deptno != 20 ;
-------------------------------------------------------------------------------------
| Id  | Operation                   | Name          | Rows  | Bytes | Time     |
-------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT            |               |     9 |   153 | 00:00:01 |
|   1 |  TABLE ACCESS BY INDEX ROWID| EMP           |     9 |   153 | 00:00:01 |
|*  2 |   INDEX FULL SCAN           | EMP_DEPTNO_IX |     9 |       | 00:00:01 |
-------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   2 - filter("DEPTNO"<>20)
```

부정형 비교는 특정 값을 제외하고 나머지 모든 값을 검색해야 합니다. 만약 조건에 만족하는 행이 적을 경우 힌트 없이도 인덱스를 사용하는 실행 계획이 생성되기도 하지만 "INDEX RANGE SCAN"을 사용하지는 않습니다. 이때 사용되는 "INDEX FULL SCAN"의 비용이 크다면 인덱스 사용은 불가능할 수 있습니다. 때문에 조건식은 가급적 긍정형으로 작성하는 것이 좋습니다.

```
SQL> SELECT empno, ename, sal, deptno
     FROM emp
     WHERE deptno IN (10, 30) ;
-------------------------------------------------------------------------------
| Id  | Operation                    | Name          | Rows  | Bytes | Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |               |     9 |   153 | 00:00:01 |
|   1 |  INLIST ITERATOR             |               |       |       |          |
|   2 |   TABLE ACCESS BY INDEX ROWID| EMP           |     9 |   153 | 00:00:01 |
|*  3 |    INDEX RANGE SCAN          | EMP_DEPTNO_IX |     9 |       | 00:00:01 |
-------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   3 - access("DEPTNO"=10 OR "DEPTNO"=30)
```

조건식을 변경하니까 힌트 없이도 "INDEX RANGE SCAN"이 사용되는 것을 확인할 수 있습니다.

```
SQL> SELECT empno, ename, sal, deptno
     FROM emp
     WHERE ename LIKE '%S' ;
-------------------------------------------------------------------------------
| Id  | Operation                    | Name          | Rows  | Bytes | Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |               |     1 |    17 | 00:00:01 |
|   1 |  TABLE ACCESS BY INDEX ROWID| EMP           |     1 |    17 | 00:00:01 |
|*  2 |   INDEX FULL SCAN            | EMP_ENAME_IX  |     1 |       | 00:00:01 |
-------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   2 - filter("ENAME" LIKE '%S' AND "ENAME" IS NOT NULL)
```

LIKE 비교는 매우 많이 사용되는 조건식입니다. 하지만 위와 같이 "%"가 앞에 있으면 "INDEX RANGE SCAN"은 불가능합니다. 영어 사전에서 "S"로 끝나는 문자를 찾으려면 특정 부분 검색만으로 찾을 수 없는 것과 동일한 이유입니다.

예전에는 부정형 조건을 사용하거나, LIKE 비교를 할 때 "%"를 앞에 붙이면 인덱스를 사용하지 못 했습니다. 하지만 현재는 상황에 따라 인덱스를 사용하는 실행 계획이 생성되기도 합니다. 물론 "INDEX FULL SCAN"을 사용하기 때문에 성능상 항상 좋다고 할 수는 없습니다.

LIKE 비교 시 "%"를 앞에 붙여야 한다면 Function based index를 이용하여 "INDEX RANGE SCAN"을 사용할 수 있도록 변경할 수 있습니다. 하지만 앞, 뒤에 "%"를 붙이는 경우에는 사용이 불가능하므로 상황에 맞게 응용하시기 바랍니다.

```
SQL> SELECT ename, REVERSE(ename)
     FROM emp ;
ENAME      REVERSE(ENAME)                          -- 문자열의 순서를 뒤집는다.
---------- ---------------
SCOTT      TTOCS
JONES      SENOJ
SMITH      HTIMS
...
14 rows selected.

SQL> CREATE INDEX emp_ename_fbi ON emp(REVERSE(ename)) ;
SQL> SELECT empno, ename, sal, deptno
     FROM emp
     WHERE REVERSE(ename) LIKE 'S%' ;
-------------------------------------------------------------------------------
| Id  | Operation                    | Name          | Rows  | Bytes | Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |               |     1 |    24 | 00:00:01 |
|   1 |  TABLE ACCESS BY INDEX ROWID| EMP           |     1 |    24 | 00:00:01 |
|*  2 |   INDEX RANGE SCAN           | EMP_ENAME_FBI |     1 |       | 00:00:01 |
-------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   2 - access(REVERSE("ENAME") LIKE 'S%')
       filter(REVERSE("ENAME") LIKE 'S%')
SQL> DROP INDEX emp_ename_fbi ;
```

6.3. 인덱스를 사용하는 SQL

개발을 하다 보면 SQL의 성능을 고민하는 경우가 종종 생깁니다. 이때 많은 분들이 실행 계획을 확인하고 인덱스의 사용 유무를 체크해서 힌트들을 지정합니다. 하지만 정의된 조건식에 따라 항상 인덱스를 사용할 수 없다면 힌트를 지정하는 작업도 소용없습니다. 인덱스는 반드시 사용돼야 하는 것은 아닙니다. 하지만 항상 사용을 못한다면 그 또한 좋은 문장이 될 수는 없습니다. 인덱스를 사용할 수 있는 SQL 작성이 우선이고, 필요에 따라 인덱스 사용은 취사선택돼야 합니다.

SQL 튜닝이 필요한 경우, 참고되는 많은 서적들이 조건식은 어떻게 작성해야 한다고 얘기합니다. 하지만 오라클 서버도 버전업을 하면서 과거보다는 인덱스를 사용할 수 있는 실행 계획을 보다 폭넓게 지원하고 있습니다. 그래서 SQL 튜닝 서적은 가급적 현재 사용하는 데이터베이스 버전을 기준으로 소개된 방법론을 공부하셔야 합니다.

SQL 튜닝에 반드시 필요한 실행 계획 분석 및 인덱스의 보다 자세한 내용들은 이 책에서 소개하려는 주제와는 조금 벗어나기 때문에 어떤 실행 계획이 좋다, 나쁘다를 평가하지는 않겠습니다. 다만 작성된 명령문이 인덱스를 사용하는지 확인하려면 실행 계획 검토가 반드시 필요하고, 인덱스를 사용할 수 없는 상황이 있는 건 아닌지 꼭 확인하시기 바랍니다.

현재 오라클 데이터베이스는 컬럼에 저장된 NULL이 없다는 것이 보장만 될 수 있다면 "INDEX [RANGE | FULL] SCAN" 실행 계획을 이용하여 어느 상황에서든 인덱스를 사용할 수는 있습니다. 다만, "INDEX FULL SCAN"은 인덱스의 모든 항목을 검색하는 것이 "TABLE FULL SCAN"보다 이점이 있을 때 유리합니다.

```
SQL> CREATE INDEX emp_idx ON emp(mgr, empno) ;        -- 결합 인덱스 생성
Index created.
SQL> SELECT mgr, empno, rowid
     FROM emp
     WHERE NOT (mgr IS NULL AND empno IS NULL)
     ORDER BY mgr, empno, rowid ;                     -- 인덱스에 저장된 값
       MGR      EMPNO ROWID
---------- ---------- ------------------
      7566       7788 AAASVDAAAAAAACjAAA
      7566       7902 AAASVDAAAAAAACjAAL
...
    (null)       7839 AAASVDAAAAAAACjAAN
14 rows selected.
```

결합 인덱스는 모든 구성 컬럼이 NULL 일 때, 해당 행의 주소를 저장하지 않습니다. 둘 중 어느 한쪽이라도 값을 가지고 있다면 인덱스에도 NULL의 주소가 저장됩니다. 그렇다면 "IS NULL" 조건도 인덱스를 사용할 수 있지 않을까요?

```
SQL> SELECT /*+ index(emp emp_idx) */ empno, ename, mgr, deptno
     FROM emp
     WHERE mgr IS NULL ;
     EMPNO ENAME             MGR     DEPTNO
---------- ---------- ---------- ----------
      7839 KING           (null)         10
-------------------------------------------------------------------------------
| Id  | Operation         | Name | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT  |      |     1 |    17 |     3   (0)| 00:00:01 |
|*  1 |  TABLE ACCESS FULL| EMP  |     1 |    17 |     3   (0)| 00:00:01 |
-------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   1 - filter("MGR" IS NULL)
```

EMPNO 컬럼은 NULL을 가지고 있는 행도 없기 때문에 모든 행의 주소는 인덱스에 저장되어 있습니다. 하지만 "TABLE FULL SCAN"이 사용되었습니다. 그 이유는 무엇일까요?

제약 조건이 없기 때문입니다. 현재 NULL을 가지고 있지 않다고 해도, 제약 조건이 정의되어 있지 않으면 언제든지 EMPNO 컬럼은 NULL을 가질 수 있습니다. 두 컬럼이 모두 NULL 상태를 가지면 인덱스에 행의 ROWID를 저장하지 않습니다. 그러면 NULL의 주소는 인덱스에서 찾을 수 없으므로 "TABLE FULL SCAN"을 이용합니다.

```
SQL> ALTER TABLE emp ADD PRIMARY KEY (empno) ;
     또는
SQL> ALTER TABLE emp MODIFY empno NOT NULL ;

SQL> SELECT /*+ index(emp emp_idx) */ empno, ename, mgr, deptno
     FROM emp
     WHERE mgr IS NULL ;

-------------------------------------------------------------------------------------------
| Id  | Operation                   | Name    | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT            |         |     1 |    17 |     2   (0)| 00:00:01 |
|   1 |  TABLE ACCESS BY INDEX ROWID| EMP     |     1 |    17 |     2   (0)| 00:00:01 |
|*  2 |   INDEX RANGE SCAN          | EMP_IDX |     1 |       |     1   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   2 - access("MGR" IS NULL)

SQL> DROP INDEX emp_idx ;
```

MGR 컬럼에는 NULL이 존재하지만 제약조건을 통해 EMPNO에 NULL이 없다는 것이 보장된다면 "mgr IS NULL" 조건도 "INDEX RANGE SCAN"을 사용할 수 있습니다. ROWID가 저장되어 있으니 사용 못할 이유는 없습니다. 단, 결합 인덱스이기 때문에 가능했고 제약조건을 통해 최소 하나 이상의 컬럼은 NULL이 없다는 것이 보장돼야 합니다.

```
SQL> SELECT /*+ index(emp emp_empno_ix) */ empno, ename, mgr, deptno
     FROM emp
     WHERE empno LIKE '77%'  ;

-----------------------------------------------------------------------------------------
| Id  | Operation                   | Name         | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT            |              |     1 |    17 |     2   (0)| 00:00:01 |
|   1 |  TABLE ACCESS BY INDEX ROWID| EMP          |     1 |    17 |     2   (0)| 00:00:01 |
|*  2 |   INDEX FULL SCAN           | EMP_EMPNO_IX |     1 |       |     1   (0)| 00:00:01 |
-----------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   2 - filter(TO_CHAR("EMPNO") LIKE '77%')
```

명시적, 암시적 컬럼 가공이 생겨도 컬럼에 NOT NULL 상태가 보장된다면 "INDEX FULL SCAN"을 사용할 수 있습니다.

실행 계획이 생성될 때 테이블 및 인덱스 구조, 그리고 적절한 제약조건 정의는 매우 중요합니다. 가급적 전문가의 도움을 받아 설계가 이루어져야 할 것이고, 여러분은 실행 계획을 확인하면서 인덱스를 사용하지 못하는 문장보다는 사용할 수 있는 문장을 작성하도록 노력하는 것이 필요합니다.

7. PGA Tuning

PGA (Program Global Area)

- Server Process, Background Process 에게 할당된 Private Memory
- User Process의 명령문 수행 시 UGA (User Global Area) 사용
- PGA는 각 Process 별로 할당된 메모리 영역이므로 그 크기의 제한이 SGA에 비해 상대적으로 어렵다.
- Oracle Database 9i부터는 PGA 영역을 자동 관리하는 것을 권장

```
SQL> DROP TABLE t1 PURGE ;
SQL> CREATE TABLE t1
     AS
     SELECT dbms_random.string('a',100) AS c1
     FROM dual
     CONNECT BY level <= 500000 ;
Table created.

SQL> SELECT segment_name, ROUND(bytes/1024/1024,1) AS MB, blocks
     FROM user_segments
     WHERE segment_name = 'T1'  ;
SEGMENT_NAME                MB     BLOCKS
--------------------- ---------- ----------
T1                          59       7552

SQL> ALTER SYSTEM FLUSH SHARED_POOL ;
System altered.
```

Optimal

- 문장 실행 시 사용된 UGA 가 충분한 메모리를 확보하여 추가적인 Temporary Data를 생성하지 않음
- 최상의 실행 환경이긴 하나 모든 작업이 Optimal 하게 수행될 수는 없다.

```
SQL> ALTER SYSTEM FLUSH SHARED_POOL ;
SQL> ALTER SYSTEM FLUSH BUFFER_CACHE ;
SQL> ALTER SESSION SET workarea_size_policy = manual ;

# 60MB 설정
SQL> ALTER SESSION SET sort_area_size = 62914560 ;
Session altered.

SQL> SET AUTOTRACE TRACEONLY STATISTICS
SQL> SELECT /*+ pga */ * FROM t1 ORDER BY 1 ;
500000 rows selected.

Elapsed: 00:00:01.71

Statistics
----------------------------------------------------------
          0  recursive calls
          0  db block gets
       7357  consistent gets
       7353  physical reads
          0  redo size
   51650712  bytes sent via SQL*Net to client
      55409  bytes received via SQL*Net from client
       5001  SQL*Net roundtrips to/from client
          1  sorts (memory)
          0  sorts (disk)
     500000  rows processed

SQL> SET AUTOTRACE OFF
```

```
SQL> COLUMN sql_id NEW_VALUE sqlid
SQL> SELECT sql_id
     FROM v$sql
     WHERE sql_text LIKE 'SELECT /*+ pga%' ;
SQL_ID
-------------
gcuq4gpbt73kj

SQL> SELECT *
     FROM table(dbms_xplan.display_cursor('&sqlid',0,
                                          'ALLSTATS LAST -rows'));
-------------------------------------------------------------------------------
| Id  | Operation          | Name |   A-Time   | Buffers | Reads  | Used-Mem |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |      |00:00:00.73 |    7357 |   7353 |          |
|   1 |  SORT ORDER BY     |      |00:00:00.73 |    7357 |   7353 |   53M (0)|
|   2 |   TABLE ACCESS FULL| T1   |00:00:00.17 |    7357 |   7353 |          |
-------------------------------------------------------------------------------
# 출력 포맷은 추가 조정함

SQL> save xplan_pga replace
Wrote file xplan_pga.sql
```

One Pass

- 문장 실행 시 사용된 UGA 가 충분하지 않아 추가적인 Temporary Data를 생성
- 생성된 Temporary Data는 Temporary Tablespace에 Physical I/O를 수행 함
- PGA의 자동 관리 사용 시 One Pass 작입이 튜닝의 목표로 사용 됨

```
SQL> ALTER SYSTEM FLUSH SHARED_POOL ;
SQL> ALTER SYSTEM FLUSH BUFFER_CACHE ;
SQL> ALTER SESSION SET workarea_size_policy = manual ;

# 10MB 설정
SQL> ALTER SESSION SET sort_area_size = 10485760 ;
Session altered.

SQL> SET AUTOTRACE TRACEONLY STATISTICS
SQL> SELECT /*+ pga */ * FROM t1 ORDER BY 1 ;
500000 rows selected.

Elapsed: 00:00:02.31

Statistics
----------------------------------------------------------
         53  recursive calls
          7  db block gets
       7357  consistent gets
      14025  physical reads
          0  redo size
   51650712  bytes sent via SQL*Net to client
      55409  bytes received via SQL*Net from client
       5001  SQL*Net roundtrips to/from client
          0  sorts (memory)
          1  sorts (disk)
     500000  rows processed

SQL> SET AUTOTRACE OFF
```

```
SQL> @xplan_pga
-------------------------------------------------------------------------------------------------
| Id  | Operation          | Name |   A-Time   | Buffers | Reads  | Writes | Used-Mem | Used-Tmp|
-------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |      |00:00:01.36 |    7364 |  14025 |   6672 |          |         |
|   1 |  SORT ORDER BY     |      |00:00:01.36 |    7364 |  14025 |   6672 | 9754K (1)|   54272 |
|   2 |   TABLE ACCESS FULL| T1   |00:00:00.17 |    7357 |   7353 |      0 |          |         |
-------------------------------------------------------------------------------------------------
```

> 확인 사항

One Pass 작업으로 수행된 내용을 확인해 보면 Temporary Data 가 테이블의 크기만큼 생성된 것을 확인할 수 있고 물리적인 writes 와 reads 가 Optimal 한 작업에 비해 늘어난 것을 확인할 수 있다.

Multi Pass

- 문장 실행 시 사용된 UGA 가 충분하지 않아 추가적인 Temporary Data를 생성
- 생성된 Temporary Data는 Temporary Tablespace에 Physical I/O를 수행 함
- UGA 공간이 부족하여 한 번의 I/O로 Temporary Data를 처리하지 못하고 반복적인 I/O 발생 됨
- 모든 작업이 Optimal로 수행되는 것이 최상이겠으나 현실적으로 불가능하다. 때문에 Multi Pass 작업이 일어나지 않도록 최소한의 메모리를 확보

```
SQL> ALTER SYSTEM FLUSH SHARED_POOL ;
SQL> ALTER SYSTEM FLUSH BUFFER_CACHE ;
SQL> ALTER SESSION SET workarea_size_policy = manual ;

# 32KB 설정
SQL> ALTER SESSION SET sort_area_size = 32768 ;
Session altered.
SQL> SET AUTOTRACE TRACEONLY STATISTICS
SQL> SELECT /*+ pga */ * FROM t1 ORDER BY 1 ;
500000 rows selected.

Elapsed: 00:00:25.82

Statistics
----------------------------------------------------------
        105  recursive calls
       3262  db block gets
       7357  consistent gets
      51812  physical reads
          0  redo size
   51650712  bytes sent via SQL*Net to client
      55409  bytes received via SQL*Net from client
       5001  SQL*Net roundtrips to/from client
          0  sorts (memory)
          1  sorts (disk)
     500000  rows processed

SQL> SET AUTOTRACE OFF
```

```
SQL> @xplan_pga
-------------------------------------------------------------------------------------------------------
| Id  | Operation          | Name |   A-Time   | Buffers | Reads  | Writes | Used-Mem | Used-Tmp|
-------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |      |00:00:21.37 |   10617 |  45145 |  37792 |          |         |
|   1 |  SORT ORDER BY     |      |00:00:21.37 |   10617 |  45145 |  37792 |41984  (58|     105K|
|   2 |   TABLE ACCESS FULL| T1   |00:00:00.17 |    7357 |   7353 |      0 |          |         |
-------------------------------------------------------------------------------------------------------
```

> 확인 사항

Multi Pass 작업으로 수행된 내용을 확인해 보면 Temporary Data가 테이블의 크기 이상으로 생성되었고 I/O 의 반복 작업이 매우 큰 것을 확인할 수 있다. 이러한 불필요한 I/O로 인해 총 경과 시간도 앞선 작업의 경과 시간에 비해 크게 늘어난 것을 확인할 수 있다.

PGA 자동 관리 (pga_aggregate_target 사용)

- pga_aggregate_target 및 workarea_size_policy 설정
- 시스템 차원에서 전체 메모리의 요구 사항을 확인하여 Multi Pass 의 작업이 수행되지 않도록 최소한의 메모리를 할당
- 사용된 메모리는 시스템 차원에서 반납 후 다른 작업에 재 사용 가능하도록 자원 관리 가능

```
SQL> conn user01/oracle

SQL> show parameter memory_target
NAME                                 TYPE        VALUE
------------------------------------ ----------- --------------------
memory_target                        big integer 808M

SQL> show parameter pga_aggregate_target
NAME                                 TYPE        VALUE
------------------------------------ ----------- --------------------
pga_aggregate_target                 big integer 0

SQL> show parameter workarea_size_policy
NAME                                 TYPE        VALUE
------------------------------------ ----------- --------------------
workarea_size_policy                 string      AUTO
```

```
SQL> ALTER SYSTEM FLUSH SHARED_POOL ;
SQL> ALTER SYSTEM FLUSH BUFFER_CACHE ;

# 32KB 설정
SQL> ALTER SESSION SET sort_area_size = 32768 ;

SQL> SET AUTOTRACE TRACEONLY STATISTICS
SQL> SELECT /*+ pga */ * FROM t1 ORDER BY 1 ;
500000 rows selected.

Elapsed: 00:00:02.67

Statistics
----------------------------------------------------------
         53  recursive calls
          4  db block gets
       7357  consistent gets
      14021  physical reads
          0  redo size
   51650712  bytes sent via SQL*Net to client
      55409  bytes received via SQL*Net from client
       5001  SQL*Net roundtrips to/from client
          0  sorts (memory)
          1  sorts (disk)
     500000  rows processed

SQL> SET AUTOTRACE OFF
```

```
SQL> @xplan_pga
-------------------------------------------------------------------------------------------------------
| Id  | Operation          | Name |   A-Time   | Buffers | Reads  | Writes | Used-Mem | Used-Tmp|
-------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |      |00:00:01.70 |    7361 |  14021 |   6668 |          |         |
|   1 |  SORT ORDER BY     |      |00:00:01.70 |    7361 |  14021 |   6668 |   34M (1)|   54272 |
|   2 |   TABLE ACCESS FULL| T1   |00:00:00.17 |    7357 |   7353 |      0 |          |         |
-------------------------------------------------------------------------------------------------------
```

> 확인 사항

workarea 의 공간이 자동 관리되므로 sort_area_size 의 설정은 적용되지 않고 34M 의 공간이 사용된 것을 확인할 수 있다. 자동 관리에 사용되는 메모리 크기는 히든 파마리터를 통해 관리된다.

```
SQL> @hidden
Enter value for parm_name: smm_max_size
Parameter                      Value           SES_MODIF  SYS_MODIF  Description
------------------------------ --------------- ---------- ---------- ------------------------------
_smm_max_size                  35225           true       immediate  maximum work area size in auto
                                                                     mode (serial)


SQL> @hidden
Enter value for parm_name: smm_px_max_size
Parameter                      Value           SES_MODIF  SYS_MODIF  Description
------------------------------ --------------- ---------- ---------- ------------------------------
_smm_px_max_size               88064           true       immediate  maximum work area size in auto
                                                                     mode (global)


SQL> DROP TABLE t1 PURGE ;
Table dropped.
```

위의 파라미터는 pga_aggregate_target 의 크기가 변경될 때마다 그 크기가 변경될 수 있으며 필요시 각 히든 파라미터를 조정하여 원하는 설정을 할 수도 있다.