개발자를 위한 Advanced SQL & SQL Tuning

Edition 1.0
July 2015

Author : Chongha Ryu
chongharyu@naver.com

Introduction to the Optimizer

SQL 문 처리 과정

1. Parse : 구문 분석 (실행 계획 확보)
2. Bind : 바인드 변수에 값 치환
3. Execute : 실행 계획대로 결과 도출
4. Fetch : Query 결과 반환 (SELECT 문장일 경우에만 수행)

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

```
[orcl:adsql]$ sqlplus user01/oracle
SQL> @idx
Enter value for tab_name: emp
INDEX_NAME                     INDEX_TYPE      UNIQUENESS      COLUMNS
------------------------------ --------------- --------------- ------------------------------
PK_EMP                         NORMAL          UNIQUE          EMPNO
EMP_JOB_IX                     NORMAL          NONUNIQUE       JOB
EMP_MGR_IX                     NORMAL          NONUNIQUE       MGR
EMP_SAL_IX                     NORMAL          NONUNIQUE       SAL
EMP_COMM_IX                    NORMAL          NONUNIQUE       COMM
EMP_HIRE_IX                    NORMAL          NONUNIQUE       HIREDATE
EMP_ENAME_IX                   NORMAL          NONUNIQUE       ENAME
EMP_DEPTNO_IX                  NORMAL          NONUNIQUE       DEPTNO

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

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
```

DEPTNO 칼럼의 조건식에 만족하는 행은 세 개이며 SAL 칼럼의 조건식에 만족하는 행은 한 개이다. 두 조건식을 동시에 만족하는 행을 찾을 때 SAL 칼럼의 인덱스를 이용하는 것이 보다 나은 실행 계획임을 확인할 수 있다. 그렇다면 실제 문장이 실행될 때 사용한 실행 계획은 어떤 인덱스를 사용했을까?

```
SQL> SET AUTOTRACE TRACEONLY EXPLAIN
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)
```

실행 계획을 생성하는 Optimizer 는 두 가지의 버전이 존재한다. 현재는 Cost Based Optimizer (CBO)가 사용되고 있으나 과거에는 Rule Based Optimizer(RBO)를 사용했었다. RBO 는 미리 지정된 규칙에 의해 실행 계획을 생성하며 그 기준 규칙은 다음의 링크를 통해 확인할 수 있다.

- Rule Based Optimizer : http://docs.oracle.com/cd/B10501_01/server.920/a96533/rbo.htm#38893

RBO 는 각 조건에 만족하는 행의 개수를 예상하지 않고 조건식에 사용되는 칼럼에 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 10g 부터는 CBO 가 기본적으로 사용된다.)

- Cost Based Optimizer : http://docs.oracle.com/cd/E11882_01/server.112/e41573/optimops.htm#i21299

```
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> SET AUTOTRACE OFF
SQL> column column_name format a15
SQL> column low_value format a20
SQL> column high_value format a20
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> SET AUTOTRACE TRACEONLY EXPLAIN
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 칼럼의 구분 값이 더 많은 것으로 인식하고 있기 때문에 SAL 칼럼의 인덱스 대신 DEPTNO 칼럼의 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)

SQL> SET AUTOTRACE OFF
```

실행 계획 (Execution Plans)

- SQL을 실행하기 위한 처리 순서도
- Optimizer에 의해 생성되며 Shared Pool의 Library Cache에 저장
- 문장 실행 전 예측되는 실행 계획은 PLAN_TABLE에 저장

1. EXPLAIN PLAN

- 실제 사용된 실행 계획이 아닌 현재 시점의 예상되는 실행 계획을 PLAN_TABLE에 저장 함
- $ORACLE_HOME/rdbms/admin/utlxplan.sql 파일 이용하여 PLAN_TABLE 생성 가능 (10g DB부터는 기본 제공)
- DBMS_XPLAN.DISPLAY 이용하여 결과 확인

```
[orcl:adsql]$ sqlplus user01/oracle
SQL> select * from plan_table ;
no rows selected

SQL> EXPLAIN PLAN FOR
     SELECT * FROM emp
     WHERE job = 'CLERK' ;
Explained.
SQL> select * from plan_table ;
...
```

분석된 실행 계획은 PLAN_TABLE을 검색하여 결과를 출력할 수 있으나 출력 결과를 조정하기가 불편하다.
DBMS_XPLAN 패키지를 사용하여 손쉽게 출력 결과를 생성한다.

```
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')
```

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> 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 옵션 확인

```
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. Library Cache

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

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

문장 실행 후 사용된 실행 계획은 V$SQL_PLAN 등을 직접 검색해서 확인할 수 있으나 원하는 내용만으로 검색하는 것이 까다롭다. 때문에 DBMS_XPLAN.DISPLAY_CURSOR를 이용한다.

```
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)
```

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 다음과 같이 에러가 발생한다.

```
SQL> conn user01/oracle
SQL> SELECT * FROM user01.dept WHERE deptno = 10;
...
SQL> SELECT * FROM table(dbms_xplan.display_cursor) ;
PLAN_TABLE_OUTPUT
----------------------------------------
User has no SELECT privilege on V$SESSION
```

일반적으로 개발자에게는 제한된 권한만을 부여하게된다. 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.

- **권한 획득**

```
SQL> conn / as sysdba
SQL> DROP ROLE sqlt ;
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) ;
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)
```

V$로 시작되는 뷰를 Dynamic Performance View라고 한다. 이러한 뷰들은 Database의 성능과 관련된 부분을 확인하거나 현재 시스템의 Workload를 분석하기 위해 DBA가 접근하는 뷰이다. 실제 사용된 문장과 실행 계획은 Library Cache라 불리는 메모리 영역에 저장되기 때문에 위와 같은 몇몇 V$ 뷰에 접근 권한을 획득해야만 일반 개발자도 실제 사용된 실행 계획과 성능에 관한 부분을 측정할 수 있다. 하지만 일반적인 경우 보안상의 이유로 V$의 뷰에 대한 접근 권한은 개발자에게 주지 않을 것이다. 만약 직접적인 권한 획득이 힘든 경우라면?

- **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 필자가 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> @myxplan
Package created.
Package body created.
Synonym created.
Grant succeeded.

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.)

MYXPLAN 패키지는 DISPLAY, DISPLAY_CURSOR의 함수를 사용 목적으로 생성하였고 사용 방법은 DBMS_XPLAN 패키지의 함수의 사용 방법도 동일하다. 다음의 실습은 MYXPLAN, DBMS_XPLAN 패키지를 이용하여 실제 사용된 실행 계획 및 추가적인 실행 통계 정보를 확인하는 방법을 학습한다.

- **실행 통계 정보 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, 'IOSTATS 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, 'IOSTATS 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, 'IOSTATS 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.

옵티마이저 연산자(Optimizer Operations)

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 **<= SYS 유저로 실행 (권한 필요)**

```
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 |
-----------------------------------------------------------------------------------------
```

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)
```

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. Subquery 종류

- Single row Subquery
- Multiple rows Subquery
- Multiple columns Subquery
- Inline View
- Scalar Subquery (Correlated 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 실행 된다. 하지만 둘 이상의 결과를 리턴하는 Multiple Rows Subquery는 Query Transformation을 통해 Join Operation를 이용하거나 Filter Operation을 사용한다. 이러한 Subquery의 Transformation은 차후 별도의 챕터를 통해 설명한다.

*** 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")
```

*** Scalar Subquery (Correlated 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.

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

- 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 ;
    DEPTNO      EMPNO ENAME                     SAL
---------- ---------- -------------------- ----------
        10       7782 CLARK                      2450
        30       7698 BLAKE                      2850
        20       7566 JONES                      2975
        20       7788 SCOTT                      3000
        20       7902 FORD                       3000
        10       7839 KING                       5000
        20       7369 SMITH                       800
        20       7566 JONES                      2975
        20       7788 SCOTT                      3000
        20       7876 ADAMS                      1100
        20       7902 FORD                       3000
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)
```

```
SQL> SELECT deptno, empno, ename, sal
     FROM emp
     WHERE sal > 2000
     UNION
     SELECT deptno, empno, ename, sal
     FROM emp
     WHERE deptno = 20 ;
SQL> @xplan
```

```
-------------------------------------------------------------------------------------------------------------------------------------
| 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)
```

```
SQL> SELECT deptno, empno, ename, sal
     FROM emp
     WHERE sal > 2000
     MINUS
     SELECT deptno, empno, ename, sal
     FROM emp
     WHERE deptno = 20 ;
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 |       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)
```

```
SQL> SELECT deptno, empno, ename, sal
     FROM emp
     WHERE sal > 2000
     INTERSECT
     SELECT deptno, empno, ename, sal
     FROM emp
     WHERE deptno = 20 ;
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 |       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)
```

SORT MERGE JOIN

- 조인의 방법에 따라 암시적인 정렬 수행 가능

```
SQL> SELECT /*+ use_merge(d e) no_index(e(deptno)) no_index(d(deptno)) */ *
     FROM dept d, emp e
     WHERE d.deptno = e.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 |       6 |       |       |          |
|   1 |  MERGE JOIN         |      |      1 |     14 |     14 |00:00:00.01 |       6 |       |       |          |
|   2 |   SORT JOIN         |      |      1 |      4 |      4 |00:00:00.01 |       3 |  2048 |  2048 | 2048  (0)|
|   3 |    TABLE ACCESS FULL| DEPT |      1 |      4 |      4 |00:00:00.01 |       3 |       |       |          |
|*  4 |   SORT JOIN         |      |      4 |     14 |     14 |00:00:00.01 |       3 |  2048 |  2048 | 2048  (0)|
|   5 |    TABLE ACCESS FULL| EMP  |      1 |     14 |     14 |00:00:00.01 |       3 |       |       |          |
-----------------------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   4 - access("D"."DEPTNO"="E"."DEPTNO")
       filter("D"."DEPTNO"="E"."DEPTNO")
```

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			

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)
```

Index 활용

〉 문제 1.

다음 문장의 실행 계획을 통해 문제점을 식별하고 성능을 향상화 시킨다.

```
SQL〉 SELECT /*+ index(c(cust_credit_limit)) */
            cust_id, cust_last_name, cust_year_of_birth, cust_city, country_id
     FROM customers c
     WHERE cust_credit_limit BETWEEN 1500 AND 3000
     AND cust_year_of_birth = 1990 ;
SQL〉 @xplan
```

```
-------------------------------------------------------------------------------------------------------------
| Id  | Operation                   | Name           | Starts | E-Rows | A-Rows |   A-Time   | Buffers |
-------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT            |                |      1 |        |      3 |00:00:00.03 |    2581 |
|*  1 |  TABLE ACCESS BY INDEX ROWID| CUSTOMERS      |      1 |      3 |      3 |00:00:00.03 |    2581 |
|*  2 |   INDEX RANGE SCAN          | CUST_CREDIT_IX |      1 |  19309 |  19309 |00:00:00.01 |      40 |
-------------------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   1 - filter("CUST_YEAR_OF_BIRTH"=1990)
   2 - access("CUST_CREDIT_LIMIT">=1500 AND "CUST_CREDIT_LIMIT"<=3000)
```

〉 문제 사항

cust_credit_limit 컬럼의 조건식에 만족하는 행의 개수가 19,309 개이나 테이블에서 나머지 조건식을 필터링 한 결과 3개의 행만이 남았다. 인덱스에서 조건에 만족하는 최소한의 행만을 찾지 못하였으므로 인덱스의 선택이 적합하지 않다.

> 답안 1. 필터링 조건이 뛰어난 컬럼의 인덱스 사용

```
SQL> SELECT COUNT(CASE WHEN cust_credit_limit BETWEEN 1500 AND 3000 THEN 1 END) AS limit,
            COUNT(CASE WHEN cust_year_of_birth = 1990              THEN 1 END) AS birth
     FROM customers ;
     LIMIT      BIRTH
---------- ----------
     19309         31

SQL> @idx
Enter value for tab_name: customers

INDEX_NAME                     INDEX_TYPE      UNIQUENESS      COLUMNS
------------------------------ --------------- --------------- ------------------------------
CUST_YOB_IX                    NORMAL          NONUNIQUE       CUST_YEAR_OF_BIRTH
CUSTOMERS_PK                   NORMAL          UNIQUE          CUST_ID
CUST_CITY_IX                   NORMAL          NONUNIQUE       CUST_CITY
CUST_FNAME_IX                  NORMAL          NONUNIQUE       CUST_FIRST_NAME
CUST_LNAME_IX                  NORMAL          NONUNIQUE       CUST_LAST_NAME
CUST_CREDIT_IX                 NORMAL          NONUNIQUE       CUST_CREDIT_LIMIT
CUST_COUNTRY_IX                NORMAL          NONUNIQUE       COUNTRY_ID

SQL> SELECT /*+ index(c(cust_year_of_birth)) */
            cust_id, cust_last_name, cust_year_of_birth, cust_city, country_id
     FROM customers c
     WHERE cust_credit_limit BETWEEN 1500 AND 3000
     AND cust_year_of_birth = 1990 ;
@xplan
-------------------------------------------------------------------------------------------------------------
| Id  | Operation                    | Name        | Starts | E-Rows | A-Rows |   A-Time   | Buffers |
-------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |             |      1 |        |      3 |00:00:00.01 |      32 |
|*  1 |  TABLE ACCESS BY INDEX ROWID| CUSTOMERS   |      1 |      3 |      3 |00:00:00.01 |      32 |
|*  2 |   INDEX RANGE SCAN           | CUST_YOB_IX |      1 |     31 |     31 |00:00:00.01 |       3 |
-------------------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   1 - filter(("CUST_CREDIT_LIMIT"<=3000 AND "CUST_CREDIT_LIMIT">=1500))
   2 - access("CUST_YEAR_OF_BIRTH"=1990)
```

조건식으로 사용되는 두 컬럼에 모두 인덱스는 존재한다. 조건에 만족하는 범위가 넓은 CUST_CREDIT_LIMIT 컬럼의 인덱스보다는 필터링 조건이 뛰어난 CUST_YEAR_OF_BIRTH 컬럼의 인덱스를 사용하여 인덱스에서의 최소한의 테이블 접근만을 시도하게 한다.

〉답안 2. 결합 인덱스 사용

```
SQL〉 CREATE INDEX custs_ix01 ON customers(cust_credit_limit, cust_year_of_birth) ;
Index created.

SQL〉 SELECT /*+ index(c custs_ix01) no_index_ss(c custs_ix01) */
            cust_id, cust_last_name, cust_year_of_birth, cust_city, country_id
     FROM customers c
     WHERE cust_credit_limit BETWEEN 1500 AND 3000
     AND cust_year_of_birth = 1990 ;
SQL〉 @xplan
-------------------------------------------------------------------------------------------------------
| Id  | Operation                   | Name       | Starts | E-Rows | A-Rows |   A-Time   | Buffers |
-------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT            |            |      1 |        |      3 |00:00:00.01 |      26 |
|   1 |  TABLE ACCESS BY INDEX ROWID| CUSTOMERS  |      1 |    175 |      3 |00:00:00.01 |      26 |
|*  2 |   INDEX RANGE SCAN          | CUSTS_IX01 |      1 |    175 |      3 |00:00:00.01 |      23 |
-------------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   2 - access("CUST_CREDIT_LIMIT">=1500 AND "CUST_YEAR_OF_BIRTH"=1990 AND
              "CUST_CREDIT_LIMIT"<=3000)
       filter("CUST_YEAR_OF_BIRTH"=1990)

SQL〉 DROP INDEX custs_ix01 ;
Index dropped.
```

조건식에 있는 두 컬럼이 결합 인덱스로 비교가 가능하면 인덱스에서 테이블의 접근이 최소화될 수 있다.
하지만 결합 인덱스는 인덱스의 컬럼 순서가 중요하며 조건에 만족하는 범위가 넓은 CUST_CREDIT_LIMIT 컬럼이 선행으로 만들어진 인덱스는 인덱스의 스캔 범위가 넓어진다. 인덱스에서 테이블에 접근 시도는 최소화되었지만 인덱스의 스캔 범위가 넓어지면서 I/O 개선 효과가 미흡하다.

```
SQL> CREATE INDEX custs_ix01 ON customers(cust_year_of_birth, cust_credit_limit) ;
Index created.

SQL> SELECT /*+ index(c custs_ix01) */
            cust_id, cust_last_name, cust_year_of_birth, cust_city, country_id
     FROM customers c
     WHERE cust_credit_limit BETWEEN 1500 AND 3000
     AND cust_year_of_birth = 1990 ;
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| CUSTOMERS  |      1 |    175 |      3 |00:00:00.01 |       6 |
|*  2 |   INDEX RANGE SCAN          | CUSTS_IX01 |      1 |    175 |      3 |00:00:00.01 |       3 |
-------------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   2 - access("CUST_YEAR_OF_BIRTH"=1990 AND "CUST_CREDIT_LIMIT">=1500 AND "CUST_CREDIT_LIMIT"<=3000)

SQL> DROP INDEX custs_ix01 ;
Index dropped.
```

필터링 조건이 뛰어난 CUST_YEAR_OF_BIRTH 컬럼이 선행으로 만들어진 결합 인덱스는 스캔 범위도 최적화되었고 테이블의 액세스 시도 역시 최소화되었다.

> 결론

테이블에 둘 이상의 조건이 존재할 때

1. 인덱스는 필터링이 가장 뛰어난 인덱스 사용
2. 인덱스의 Range Scan 범위가 임계값을 초과하면 인덱스와 테이블 사이의 랜덤 액세스가 증가되므로 필터링 조건이 좋은 컬럼들을 묶어서 결합 인덱스 생성 및 사용

› 문제 2.

다음 문장의 성능을 확인하고 인덱스를 사용할 수 있도록 수정하시오.

```
SQL> SELECT *
     FROM user02.emp
     WHERE empno = 7788 ;
SQL> @xplan
```

```
-------------------------------------------------------------------------------------
| Id  | Operation         | Name | Starts | E-Rows | A-Rows |   A-Time   | Buffers |
-------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT  |      |      1 |        |      1 |00:00:00.01 |       3 |
|*  1 |  TABLE ACCESS FULL| EMP  |      1 |      1 |      1 |00:00:00.01 |       3 |
-------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   1 - filter(TO_NUMBER("EMPNO")=7788)
```

› 문제 사항

Predicate Information을 확인하면 조건식에는 정의하지 않은 TO_NUMBER 함수가 사용되고 있다. 즉, 암시적인 형 변환 작업으로 인해 인덱스를 사용할 수 없는 상황이다.

〉답안 1. 명시적 형 변환

```
SQL〉 DESC user02.emp
 Name                              Null?    Type
 --------------------------------- -------- ------------------------
 EMPNO                                      VARCHAR2(4)
 ENAME                                      VARCHAR2(10)
 JOB                                        VARCHAR2(9)
 MGR                                        VARCHAR2(4)
 HIREDATE                                   DATE
 SAL                                        NUMBER(7,2)
 COMM                                       NUMBER(7,2)
 DEPTNO                                     VARCHAR2(2)

SQL〉 SELECT *
     FROM user02.emp
     WHERE empno = '7788' ;
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          | EMP_EMPNO_IX |      1 |      1 |      1 |00:00:00.01 |       1 |
-------------------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   2 - access("EMPNO"='7788')

SQL〉 SELECT *
     FROM user02.emp
     WHERE empno = TO_CHAR(7788) ;
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          | EMP_EMPNO_IX |      1 |      1 |      1 |00:00:00.01 |       1 |
-------------------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   2 - access("EMPNO"='7788')
```

문장을 수정할 경우 명시적인 형 변환이 가능 하도록 수정하거나 바인드 변수를 컬럼의 데이터 타입과 동일하게 선언한다.

> 답안 2. 함수 기반 인덱스 사용

```
SQL> CONN user02/oracle
SQL> CREATE INDEX emp_fix ON emp(TO_NUMBER(empno)) ;

SQL> SELECT *
     FROM user02.emp
     WHERE empno = 7788 ;
SQL> @xplan
```

```
-------------------------------------------------------------------------------------------------
| Id  | Operation                   | Name    | Starts | E-Rows | A-Rows |   A-Time   | Buffers |
-------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT            |         |      1 |        |      1 |00:00:00.01 |       3 |
|   1 |  TABLE ACCESS BY INDEX ROWID| EMP     |      1 |      1 |      1 |00:00:00.01 |       3 |
|*  2 |   INDEX RANGE SCAN          | EMP_FIX |      1 |      1 |      1 |00:00:00.01 |       2 |
-------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   2 - access("EMP"."SYS_NC00009$"=7788)
```

```
SQL> DROP INDEX emp_fix ;
SQL> CONN user01/oracle
```

문장을 수정할 수 없는 경우 암시적인 형 변환에 사용되는 함수를 이용하여 Function Based Index 생성

〉문제 3.

다음의 문장을 확인하고 성능을 최적화할 수 있도록 필요한 조치를 취하시오.

```
SQL〉 SELECT channel_id, COUNT(*)
     FROM sales s
     WHERE channel_id             <= 4
       AND TO_CHAR(time_id,'YYYY') >= '2000'
     GROUP BY channel_id ;
SQL〉 @xplan
```

```
-------------------------------------------------------------------------------------------------------------------------------
| Id  | Operation          | Name  | Starts | E-Rows | A-Rows |   A-Time   | Buffers | Reads  |  OMem |  1Mem | Used-Mem |
-------------------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |       |      1 |        |      3 |00:00:00.45 |    4434 |   4431 |       |       |          |
|   1 |  HASH GROUP BY     |       |      1 |      2 |      3 |00:00:00.45 |    4434 |   4431 |  888K |  888K |  655K (0)|
|*  2 |   TABLE ACCESS FULL| SALES |      1 |    492K|    492K|00:00:00.33 |    4434 |   4431 |       |       |          |
-------------------------------------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   2 - filter(("CHANNEL_ID"<=4 AND TO_CHAR(INTERNAL_FUNCTION("TIME_ID"),'YYYY')>='2000'))
```

〉문제 사항

- 컬럼의 가공 및 선분 조건이 넓어서 인덱스의 사용이 불가능
- 조건에 만족하는 행의 개수가 전체의 50% 를 넘기 때문에 인덱스 사용이 의미 없음

〉답안 1. 함수 사용의 최소화 및 인덱스의 활용

- I/O의 낭비를 최소화하려면 테이블의 액세스 없이 인덱스만으로 처리하도록 결합 인덱스 생성
- 인덱스에서도 액세스 범위는 넓기 때문에 Multi Block I/O 사용

```
SQL> CREATE INDEX sales_x01 ON sales(channel_id, time_id) ;
Index created.
SQL> SELECT /*+ index_ffs (s sales_x01) */ channel_id, COUNT(*)
     FROM sales s
     WHERE channel_id              <= 4
       AND TO_CHAR(time_id,'YYYY') >= '2000'
     GROUP BY channel_id ;
SQL> @xplan
-------------------------------------------------------------------------------------------------------------------------------
| Id  | Operation             | Name      | Starts | E-Rows | A-Rows |   A-Time   | Buffers |  OMem |  1Mem | Used-Mem |
-------------------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT      |           |      1 |        |      3 |00:00:00.45 |    2839 |       |       |          |
|   1 |  HASH GROUP BY        |           |      1 |      2 |      3 |00:00:00.45 |    2839 |   888K|   888K|  654K (0)|
|*  2 |   INDEX FAST FULL SCAN| SALES_X01 |      1 |  24612 |    492K|00:00:00.33 |    2839 |       |       |          |
-------------------------------------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   2 - filter(("CHANNEL_ID"<=4 AND TO_CHAR(INTERNAL_FUNCTION("TIME_ID"),'YYYY')>='2000'))
```

테이블의 액세스 없이 인덱스만으로 결과를 생성할 수 있다. 하지만 암시적인 형 변환 작업이 반복되므로 불필요한 시간 낭비가 여전하다.

```
SQL> SELECT /*+ index_ffs (s sales_x01) */ channel_id, COUNT(*)
     FROM sales s
     WHERE channel_id <= 4
       AND time_id    >= TO_DATE('2000/01/01','YYYY/MM/DD')
     GROUP BY channel_id ;
SQL> @xplan
-------------------------------------------------------------------------------------------------------------------------------
| Id  | Operation             | Name      | Starts | E-Rows | A-Rows |   A-Time   | Buffers |  OMem |  1Mem | Used-Mem |
-------------------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT      |           |      1 |        |      3 |00:00:00.28 |    2839 |       |       |          |
|   1 |  HASH GROUP BY        |           |      1 |      2 |      3 |00:00:00.28 |    2839 |   888K|   888K|  654K (0)|
|*  2 |   INDEX FAST FULL SCAN| SALES_X01 |      1 |    246K|    492K|00:00:00.16 |    2839 |       |       |          |
-------------------------------------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   2 - filter(("TIME_ID">=TO_DATE(' 2000-01-01 00:00:00', 'syyyy-mm-dd hh24:mi:ss') AND "CHANNEL_ID"<=4))

SQL> DROP INDEX sales_x01 ;
Index dropped.
```

› 문제 4.

다음 문장의 실행 계획을 확인하고, SORT 작업 없이 정렬된 결과를 검색할 수 있도록 수정하시오.

```
SQL> SELECT deptno, empno, ename, sal
     FROM emp
     WHERE deptno IN (10, 20)
     ORDER BY deptno, sal ;
    DEPTNO      EMPNO ENAME             SAL
---------- ---------- ---------- ----------
        10       7934 MILLER           1300
        10       7782 CLARK            2450
        10       7839 KING             5000
        20       7369 SMITH             800
        20       7876 ADAMS            1100
        20       7566 JONES            2975
        20       7788 SCOTT            3000
        20       7902 FORD             3000
SQL> @iostat
-------------------------------------------------------------------------------------------------------------
| Id  | Operation                     | Name          | Starts | E-Rows | A-Rows |   A-Time   | Buffers |
-------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT              |               |      1 |        |      8 |00:00:00.01 |       3 |
|   1 |  SORT ORDER BY                |               |      1 |      9 |      8 |00:00:00.01 |       3 |
|   2 |   INLIST ITERATOR             |               |      1 |        |      8 |00:00:00.01 |       3 |
|   3 |    TABLE ACCESS BY INDEX ROWID| EMP           |      2 |      9 |      8 |00:00:00.01 |       3 |
|*  4 |     INDEX RANGE SCAN          | EMP_DEPTNO_IX |      2 |      9 |      8 |00:00:00.01 |       2 |
-------------------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   4 - access(("DEPTNO"=10 OR "DEPTNO"=20))
```

› 문제 사항

정렬을 수행해야 하는 경우, 조건에 만족하는 모든 행이 탐색되어야 하고 추가적인 메모리를 사용한다. 때문에 처리된 결과를 우선적으로 화면에 출력해야 하는 Online 업무에서는 ORDER BY 절을 사용하는 것보다 인덱스를 활용하는 것이 좋을 수 있다.

› 답안. 인덱스를 활용한 정렬

```
SQL> @idx
Enter value for tab_name: emp
INDEX_NAME                     INDEX_TYPE      UNIQUENESS      COLUMNS
------------------------------ --------------- --------------- ------------------------------
PK_EMP                         NORMAL          UNIQUE          EMPNO
EMP_JOB_IX                     NORMAL          NONUNIQUE       JOB
EMP_MGR_IX                     NORMAL          NONUNIQUE       MGR
EMP_SAL_IX                     NORMAL          NONUNIQUE       SAL
EMP_COMM_IX                    NORMAL          NONUNIQUE       COMM
EMP_HIRE_IX                    NORMAL          NONUNIQUE       HIREDATE
EMP_ENAME_IX                   NORMAL          NONUNIQUE       ENAME
EMP_DEPTNO_IX                  NORMAL          NONUNIQUE       DEPTNO
```

조건식에 비교되는 컬럼은 DEPTNO 컬럼이며 정렬에는 DEPTNO, SAL 컬럼이 사용되고 있다. 때문에 해당 조건에 만족하는 작업을 인덱스로 처리하려면 결합 인덱스가 필요하다.

```
SQL> CREATE INDEX emp_x01 ON emp(deptno, sal) ;
Index created.

SQL> SELECT /*+ index(emp emp_x01) */ deptno, empno, ename, sal
     FROM emp
     WHERE deptno IN (10, 20)
     ORDER BY deptno, sal ;
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_X01 |      2 |      9 |      8 |00:00:00.01 |       3 |
-------------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   3 - access(("DEPTNO"=10 OR "DEPTNO"=20))

SQL> DROP INDEX emp_x01 ;
Index dropped.
```

〉추가 실습

인덱스의 검색 방향을 조절하면 오름차순, 내림차순 정렬 상태를 변경할 수 있다.

```
SQL〉 SELECT /*+ index_asc(emp(sal)) */ *
      FROM emp WHERE sal 〉 0
      ORDER BY sal ASC ;
     EMPNO ENAME      JOB              MGR HIREDATE         SAL       COMM     DEPTNO
---------- ---------- --------- ---------- -------- ---------- ---------- ----------
      7369 SMITH      CLERK           7902 80/12/17        800                    20
      7900 JAMES      CLERK           7698 81/12/03        950                    30
      7876 ADAMS      CLERK           7788 87/05/23       1100                    20
...
14 rows selected.

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 BY INDEX ROWID| EMP        |      1 |     14 |     14 |00:00:00.01 |       4 |
|*  2 |   INDEX RANGE SCAN           | EMP_SAL_IX |      1 |     14 |     14 |00:00:00.01 |       2 |
-------------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   2 - access("SAL">0)

SQL〉 SELECT /*+ index_desc(emp(sal)) */ *
      FROM emp WHERE sal 〉 0
      ORDER BY sal DESC ;
     EMPNO ENAME      JOB              MGR HIREDATE         SAL       COMM     DEPTNO
---------- ---------- --------- ---------- -------- ---------- ---------- ----------
      7839 KING       PRESIDENT            81/11/17       5000                    10
      7902 FORD       ANALYST         7566 81/12/03       3000                    20
      7788 SCOTT      ANALYST         7566 87/04/19       3000                    20
...
14 rows selected.

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 BY INDEX ROWID | EMP        |      1 |     14 |     14 |00:00:00.01 |       4 |
|*  2 |   INDEX RANGE SCAN DESCENDING| EMP_SAL_IX |      1 |     14 |     14 |00:00:00.01 |       2 |
-------------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   2 - access("SAL">0)
```

단, 인덱스는 NULL 값의 주소를 저장하지 않기 때문에, 인덱스를 사용하려면 반드시 NULL을 제외한 값이 검색될 수 있도록 조건식이 존재하거나 해당 컬럼에 Primary Key 또는 NOT NULL 제약 조건이 필요하다.

```
SQL> SELECT /*+ index_asc(emp(sal)) */ *
       FROM emp
     ORDER BY sal ASC ;
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 |       2 |       |       |          |
|   1 |  SORT ORDER BY     |      |      1 |     14 |     14 |00:00:00.01 |       2 |  2048 |  2048 | 2048  (0)|
|   2 |   TABLE ACCESS FULL| EMP  |      1 |     14 |     14 |00:00:00.01 |       2 |       |       |          |
-------------------------------------------------------------------------------------------------------------------
```

SAL 컬럼에는 제약 조건이 존재하지 않으므로 힌트만으로 인덱스 사용을 강제화 시킬 수 없다.

```
SQL> SELECT /*+ index_asc(emp(empno)) */ *
     FROM emp
     ORDER BY empno ASC ;
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 BY INDEX ROWID| EMP    |      1 |     14 |     14 |00:00:00.01 |       4 |
|   2 |   INDEX FULL SCAN           | PK_EMP |      1 |     14 |     14 |00:00:00.01 |       2 |
-----------------------------------------------------------------------------------------------
```

EMPNO 컬럼에는 Primary Key 제약 조건이 존재한다. 때문에 EMPNO 컬럼은 NULL을 가지고 있지 않다는 것이 제약 조건을 통해서 보장이 되고, 인덱스를 이용한 정렬 작업이 가능하다.

› 주의

ORDER BY를 대체하는 인덱스 사용 시 ORDER BY 절을 생략해도 정렬 상태는 보장받을 수 있다. 하지만 경우에 따라 해당 인덱스를 사용할 수 없는 경우가 발생하면 결과가 틀려질 수 있다. 때문에 불필요해 보이는 ORDER BY 절이라도 해도 생략하지 않는다.

인덱스를 사용할 경우 ORDER BY 절이 존재하더라도 Query Transformation을 통해 불필요한 ORDER BY 절은 제거된다. 만약 해당 인덱스의 사용이 불가능한 경우라면 ORDER BY 절을 통해 결과를 보장할 수 있다.

> 문제 5.

EMPLOYEES 테이블에서 급여를 가장 많이 받는 사원을 2명 검색하도록 문장을 작성하고, 성능을 최적화할 수 있도록 필요한 조치를 취하시오.

```
EMPLOYEE_ID LAST_NAME            JOB_ID                    SALARY DEPARTMENT_ID
----------- -------------------- -------------------- ---------- -------------
        100 King                 AD_PRES                   24000            90
        101 Kochhar              AD_VP                     17000            90
```

〉답안 1. TOP-n 질의 사용

```
SQL〉 SELECT employee_id, last_name, job_id, salary, department_id
     FROM ( SELECT * FROM employees ORDER BY salary DESC )
     WHERE rownum <= 2 ;
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 |  COUNT STOPKEY         |           |      1 |        |      2 |00:00:00.01 |       3 |       |       |          |
|   2 |   VIEW                 |           |      1 |    107 |      2 |00:00:00.01 |       3 |       |       |          |
|*  3 |    SORT ORDER BY STOPKEY|          |      1 |    107 |      2 |00:00:00.01 |       3 |  2048 |  2048 | 2048  (0)|
|   4 |     TABLE ACCESS FULL  | EMPLOYEES |      1 |    107 |    107 |00:00:00.01 |       3 |       |       |          |
-------------------------------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   1 - filter(ROWNUM<=2)
   3 - filter(ROWNUM<=2)
```

TOP-n 질의 문의 가장 큰 장점은 Sort Area의 크기를 작게 사용한다는 점이다. 검색되는 행이 2개밖에 없다면 정렬에 필요한 공간도 크게 사용할 이유가 없어진다. 다만 정렬에 필요한 테이블의 Full Scan 작업은 테이블의 크기가 커질수록 부담이 될 수 있다.

〉답안 2. 인덱스 사용

```
SQL〉 SELECT /*+ index_rs_desc(e(salary)) */ employee_id, last_name, job_id, salary, department_id
     FROM employees e WHERE salary 〉 0 AND 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 BY INDEX ROWID | EMPLOYEES   |      1 |      2 |      2 |00:00:00.01 |       4 |
|*  3 |    INDEX RANGE SCAN DESCENDING| EMPL_SAL_IX |      1 |    107 |      2 |00:00:00.01 |       2 |
-----------------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   1 - filter(ROWNUM<=2)
   3 - access("SALARY">0)
       filter("SALARY">0)
```

앞서 확인한 인덱스의 액세스 방향을 조절하여 TOP-n 질의와 동일한 결과를 만들 수 있으며 불필요한 정렬 작업도 제거할 수 있다. 단, 인덱스의 사용이 불가능한 경우 결과가 틀려질 수 있다. 위의 실행 계획은 반드시 인덱스를 이용해야만 급여를 가장 많이 받는 사원을 검색할 수 있다.

> 답안 3. TOP-n 질의 사용 (인덱스 활용)

```
SQL> SELECT employee_id, last_name, job_id, salary, department_id
     FROM ( SELECT /*+ index_rs_desc(e(salary)) */ *
            FROM employees
            WHERE salary > 0
            ORDER BY salary DESC )
     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	VIEW		1	2	2	00:00:00.01	4
3	TABLE ACCESS BY INDEX ROWID	EMPLOYEES	1	107	2	00:00:00.01	4
* 4	INDEX RANGE SCAN DESCENDING	EMPL_SAL_IX	1	2	2	00:00:00.01	2

```
Predicate Information (identified by operation id):
---------------------------------------------------
   1 - filter(ROWNUM<=2)
   4 - access("SALARY">0)
```

인덱스를 사용하여 별도의 정렬 작업 없이 동일한 결과를 검색할 수 있다. 또한 인덱스가 사용 불가능한 상황이 생겨도 항상 올바른 결과를 보장할 수 있다.

› 문제 6.

다음 문장의 실행 계획을 확인하고 인덱스를 최대한 활용할 수 있도록 필요한 사항을 구현하시오.

```
SQL> SELECT channel_id, MAX(time_id)
     FROM sales
     WHERE channel_id = 3
     GROUP BY channel_id ;
SQL> @xplan
```

```
-----------------------------------------------------------------------------------------------
| Id  | Operation            | Name  | Starts | E-Rows | A-Rows |   A-Time   | Buffers | Reads  |
-----------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT     |       |      1 |        |      1 |00:00:00.31 |    4434 |   4431 |
|   1 |  SORT GROUP BY NOSORT|       |      1 |      1 |      1 |00:00:00.31 |    4434 |   4431 |
|*  2 |   TABLE ACCESS FULL  | SALES |      1 |    229K|    540K|00:00:00.15 |    4434 |   4431 |
-----------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   2 - filter("CHANNEL_ID"=3)
```

› 문제 사항

MIN, MAX의 값은 컬럼에 저장된 값들을 계산하여 새로운 값을 생성하는 것이 아니라 기존의 값 중 가장 큰 값이나 작은 값을 가져온다. 저장되어 있는 하나의 컬럼의 값을 검색하고자 Table Full Scan이 사용된다면 불필요한 I/O가 증가된다.

〉답안 1. 결합 인덱스를 생성하고 인덱스만 액세스 하도록 문장 수정

```
SQL〉 CREATE INDEX sales_x01 ON sales(channel_id,time_id) ;
SQL〉 SELECT channel_id, MAX(time_id)
     FROM sales WHERE channel_id = 3
     GROUP BY channel_id ;
SQL〉 @xplan
```

```
-------------------------------------------------------------------------------------------
| Id  | Operation            | Name      | Starts | E-Rows | A-Rows |   A-Time   | Buffers |
-------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT     |           |      1 |        |      1 |00:00:00.28 |    1660 |
|   1 |  SORT GROUP BY NOSORT|           |      1 |      1 |      1 |00:00:00.28 |    1660 |
|*  2 |   INDEX RANGE SCAN   | SALES_X01 |      1 |    229K|    540K|00:00:00.13 |    1660 |
-------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   2 - access("CHANNEL_ID"=3)
```

인덱스로 필요한 범위만 스캔 후 결과를 만들었지만 인덱스의 스캔 범위가 넓다. 만약 조건에 만족하는 행이 많으면 더 넓은 범위를 액세스하며 성능이 저하될 것이다.

〉답안 2. 인덱스 사용 및 ROWNUM 활용

```
SQL〉 SELECT /*+ index_rs_desc(sales sales_x01) */ channel_id, time_id
       FROM sales
      WHERE channel_id = 3
        AND ROWNUM     = 1;
SQL〉 @xplan
```

```
-------------------------------------------------------------------------------------------------------
| Id  | Operation                    | Name      | Starts | E-Rows | A-Rows |   A-Time   | Buffers |
-------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |           |      1 |        |      1 |00:00:00.01 |       3 |
|*  1 |  COUNT STOPKEY               |           |      1 |        |      1 |00:00:00.01 |       3 |
|*  2 |   INDEX RANGE SCAN DESCENDING| SALES_X01 |      1 |      1 |      1 |00:00:00.01 |       3 |
-------------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   1 - filter(ROWNUM=1)
   2 - access("CHANNEL_ID"=3)
```

동일한 CHANNEL_ID 값의 개수가 증가하여도 항상 일정한 I/O로 결과를 검색할 수 있으나 해당 인덱스의 사용이 불가능한 경우가 발생하면 잘못된 결과를 검색할 수 있다. 또한 ROWNUM 의 조건식을 Subquery 안에서 사용하면 Query Transformation이 진행되지 못하는 경우도 발생하므로 전체 문장의 최적화가 힘들 수 있다.

> 답안 3. 인덱스를 활용한 MIN, MAX 함수의 사용

```
SQL> SELECT MAX(time_id)
     FROM sales
     WHERE channel_id = 3 ;
SQL> @xplan
```

```
-------------------------------------------------------------------------------------------------------
| Id  | Operation                    | Name      | Starts | E-Rows | A-Rows |   A-Time   | Buffers |
-------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |           |      1 |        |      1 |00:00:00.01 |       3 |
|   1 |  SORT AGGREGATE              |           |      1 |      1 |      1 |00:00:00.01 |       3 |
|   2 |   FIRST ROW                  |           |      1 |      1 |      1 |00:00:00.01 |       3 |
|*  3 |    INDEX RANGE SCAN (MIN/MAX)| SALES_X01 |      1 |      1 |      1 |00:00:00.01 |       3 |
-------------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   3 - access("CHANNEL_ID"=3)
```

```
SQL> SELECT 3 AS channel_id, MAX(time_id)
     FROM sales
     WHERE channel_id = 3 ;
SQL> @xplan
```

```
-------------------------------------------------------------------------------------------------------
| Id  | Operation                    | Name      | Starts | E-Rows | A-Rows |   A-Time   | Buffers |
-------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |           |      1 |        |      1 |00:00:00.01 |       3 |
|   1 |  SORT AGGREGATE              |           |      1 |      1 |      1 |00:00:00.01 |       3 |
|   2 |   FIRST ROW                  |           |      1 |      1 |      1 |00:00:00.01 |       3 |
|*  3 |    INDEX RANGE SCAN (MIN/MAX)| SALES_X01 |      1 |      1 |      1 |00:00:00.01 |       3 |
-------------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   3 - access("CHANNEL_ID"=3)
```

```
SQL> DROP INDEX sales_x01 ;
```

〉문제 7.

다음 문장이 기존의 인덱스를 최적화된 방법으로 사용할 수 있도록 튜닝하시오.

```
SQL〉 SELECT /*+ no_index(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 FULL| EMPLOYEES |      1 |      1 |      2 |00:00:00.01 |       4 |
-------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   1 - filter("FIRST_NAME"='Steven')
```

```
SQL〉 @idx
Enter value for tab_name: employees

INDEX_NAME                     INDEX_TYPE      UNIQUENESS      COLUMNS
------------------------------ --------------- --------------- ------------------------------
EMPL_JOB_IX                    NORMAL          NONUNIQUE       JOB_ID
EMPL_MGR_IX                    NORMAL          NONUNIQUE       MANAGER_ID
EMPL_SAL_IX                    NORMAL          NONUNIQUE       SALARY
EMPL_NAME_IX                   NORMAL          NONUNIQUE       LAST_NAME+FIRST_NAME
EMPL_EMAIL_IX                  NORMAL          NONUNIQUE       EMAIL
EMP_EMP_ID_PK                  NORMAL          UNIQUE          EMPLOYEE_ID
EMPL_DEPTNO_IX                 NORMAL          NONUNIQUE       DEPARTMENT_ID
```

〉문제 사항

FIRST_NAME 컬럼에 인덱스가 존재하지만 선행 컬럼으로 존재하는 것이 아니기 때문에 인덱스를 사용하지 못한다.

〉답안

```
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 Skip Scan을 이용하면 선행 컬럼을 제외하고 인덱스의 액세스 성능을 최적화 시킬 수 있다. 다만, 선행 컬럼의 구분 값이 많아지면 실제 작업량이 증가되므로 FIRST_NAME 컬럼이 조건식에 많이 사용된다면 인덱스의 추가 생성을 고려한다.

```
SQL〉 SELECT /*+ no_index_ss(e empl_name_ix) index(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 FULL 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')
```

실행 계획 상에서 INDEX FULL SCAN은 인덱스의 모든 leaf 블록을 탐색했다는 의미이다. 경우에 따라 필요할 수도 있지만 일반적으로 INDEX FULL SCAN은 인덱스 블록의 액세스 증가로 성능이 저하되는 경우도 존재한다. 때문에 인덱스의 사용 유무만을 확인하지 말고 필요한 실행 계획이 선택되었는지, 해당 작업이 성능상 이점이 존재하는지를 확인한다.

> 인덱스의 올바른 사용

```
SQL> CREATE TABLE sales2 AS SELECT * FROM sales ORDER BY cust_id, time_id ;
SQL> CREATE INDEX s2_prod_ix ON sales2(prod_id) ;
SQL> CREATE INDEX s2_time_ix ON sales2(time_id) ;

SQL> SELECT *
     FROM sales2 s
     WHERE prod_id = 120
       AND time_id BETWEEN TO_DATE('2001/01/01','YYYY/MM/DD')
                       AND TO_DATE('2001/12/31','YYYY/MM/DD') ;
SQL> @xplan
-------------------------------------------------------------------------------------------------
| Id  | Operation         | Name   | Starts | E-Rows | A-Rows |   A-Time   | Buffers | Reads  |
-------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT  |        |      1 |        |   4912 |00:00:00.03 |    4485 |   4431 |
|*  1 |  TABLE ACCESS FULL| SALES2 |      1 |   5299 |   4912 |00:00:00.03 |    4485 |   4431 |
-------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   1 - filter(("PROD_ID"=120 AND "TIME_ID">=TO_DATE(' 2001-01-01 00:00:00', 'syyyy-mm-dd hh24:mi:ss')
                             AND "TIME_ID"<=TO_DATE(' 2001-12-31 00:00:00', 'syyyy-mm-dd hh24:mi:ss')))
```

인덱스를 사용할 수 있는 상황이지만 사용된 실행 계획은 "FULL TABLE SCAN"이 사용되었다. 힌트를 이용하여 인덱스 사용을 유도하면?

```
SQL> SELECT /*+ index(s(prod_id)) */ *
     FROM sales2 s
     WHERE prod_id = 120
       AND time_id BETWEEN TO_DATE('2001/01/01','YYYY/MM/DD')
                       AND TO_DATE('2001/12/31','YYYY/MM/DD') ;
SQL> @xplan
-------------------------------------------------------------------------------------------------------
| Id  | Operation                    | Name       | Starts | E-Rows | A-Rows |   A-Time   | Buffers |
-------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |            |      1 |        |   4912 |00:00:00.04 |    4361 |
|*  1 |  TABLE ACCESS BY INDEX ROWID| SALES2     |      1 |   5299 |   4912 |00:00:00.04 |    4361 |
|*  2 |   INDEX RANGE SCAN           | S2_PROD_IX |      1 |  17471 |  19403 |00:00:00.01 |      94 |
-------------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   1 - filter(("TIME_ID">=TO_DATE(' 2001-01-01 00:00:00', 'syyyy-mm-dd hh24:mi:ss') AND
              "TIME_ID"<=TO_DATE(' 2001-12-31 00:00:00', 'syyyy-mm-dd hh24:mi:ss')))
   2 - access("PROD_ID"=120)
```

```
SQL> SELECT /*+ index(s(time_id)) */ *
     FROM sales2 s
     WHERE prod_id = 120
       AND time_id BETWEEN TO_DATE('2001/01/01','YYYY/MM/DD')
                       AND TO_DATE('2001/12/31','YYYY/MM/DD') ;
SQL> @xplan
-------------------------------------------------------------------------------------------------------
| Id  | Operation                   | Name       | Starts | E-Rows | A-Rows |   A-Time   | Buffers |
-------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT            |            |      1 |        |   4912 |00:00:00.43 |   30789 |
|*  1 |  TABLE ACCESS BY INDEX ROWID| SALES2     |      1 |   5299 |   4912 |00:00:00.43 |   30789 |
|*  2 |   INDEX RANGE SCAN          | S2_TIME_IX |      1 |    233K|    259K|00:00:00.10 |     741 |
-------------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   1 - filter("PROD_ID"=120)
   2 - access("TIME_ID">=TO_DATE(' 2001-01-01 00:00:00', 'syyyy-mm-dd hh24:mi:ss') AND
              "TIME_ID"<=TO_DATE(' 2001-12-31 00:00:00', 'syyyy-mm-dd hh24:mi:ss'))
```

조건으로 사용되는 두 개의 컬럼의 각 인덱스를 사용했지만 성능 개선 효과는 없다. 오히려 TIME_ID 컬럼의 인덱스를 사용한 실행 계획은 메모리(Buffer Cache)에서의 I/O가 폭발적으로 증가된 것을 확인할 수 있다.
인덱스는 무조건 사용한다고 최적의 성능을 보장하지는 않는다. 인덱스에서 조건에 만족하는 범위가 넓어질수록 ROWID를 통한 테이블의 반복적인 접근이 많아지고, 이는 성능상 문제점을 발생시킨다. 필터링 조건이 뛰어난 최소한의 인덱스의 사용이 불가능한 경우라면 차라리 "TABLE FULL SCAN"이 최적이라고 할 수도 있다.
단, 현재의 경우는 두 조건을 동시에 만족하는 행이 많은 것은 아니므로 결합 인덱스의 추가 생성을 고려한다.

```
SQL> CREATE INDEX sales2_prod_time_ix ON sales2(prod_id, time_id) ;
SQL> SELECT /*+ index(s sales2_prod_time_ix) */ *
     FROM sales2 s
     WHERE prod_id = 120
       AND time_id BETWEEN TO_DATE('2001/01/01','YYYY/MM/DD')
                       AND TO_DATE('2001/12/31','YYYY/MM/DD') ;
SQL> @xplan
-------------------------------------------------------------------------------------------------------
| Id  | Operation                   | Name               | Starts | E-Rows | A-Rows |   A-Time   | Buffers |
-------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT            |                    |      1 |        |   4912 |00:00:00.01 |    3545 |
|   1 |  TABLE ACCESS BY INDEX ROWID| SALES2             |      1 |   4583 |   4912 |00:00:00.01 |    3545 |
|*  2 |   INDEX RANGE SCAN          | SALES2_PROD_TIME_IX |      1 |   4583 |   4912 |00:00:00.01 |      68 |
-------------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   2 - access("PROD_ID"=120 AND "TIME_ID">=TO_DATE(' 2001-01-01 00:00:00', 'syyyy-mm-dd hh24:mi:ss')
                            AND "TIME_ID"<=TO_DATE(' 2001-12-31 00:00:00', 'syyyy-mm-dd hh24:mi:ss'))
```

추가 생성된 결합 인덱스는 두 조건식을 하나의 인덱스로 처리할 수 있다. 하지만 Buffers의 I/O 개수는 "FULL TABLE SCAN" 실행 계획에 비해 획기적으로 개선되었다고 볼 수는 없다. 이는 생성된 SALES2 테이블이 CUST_ID, TIME_ID 컬럼을 기준을 정렬하여 저장하였기 때문에 발생한 현상이다. 지금과 같은 현상을 'Clustering Factor가 나쁘다."라고 표현하는데, 인덱스에서 발견한 4,912개의 행이 동일한 블록에 가급적 함께 저장되지 않고, 테이블이 가지고 있는 여러 블록에 흩어져 저장되어 있기 때문에 발생한 현상이다. 이렇게 Clustering Factor가 나쁜 경우라면 결합 인덱스를 가지고 있더라도 "FULL TABLE SCAN"이 오히려 성능상 유리할 수도 있다.

```
SQL> SELECT * FROM sales2 s
     WHERE prod_id = 120
       AND time_id BETWEEN TO_DATE('2001/01/01','YYYY/MM/DD')
                       AND TO_DATE('2001/12/31','YYYY/MM/DD') ;
SQL> @xplan
-------------------------------------------------------------------------------------------------
| Id  | Operation         | Name   | Starts | E-Rows | A-Rows |   A-Time   | Buffers | Reads  |
-------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT  |        |      1 |        |   4912 |00:00:00.03 |    4485 |   4431 |
|*  1 |  TABLE ACCESS FULL| SALES2 |      1 |   5299 |   4912 |00:00:00.03 |    4485 |   4431 |
-------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   1 - filter(("PROD_ID"=120 AND "TIME_ID">=TO_DATE(' 2001-01-01 00:00:00', 'syyyy-mm-dd hh24:mi:ss')
                             AND "TIME_ID"<=TO_DATE(' 2001-12-31 00:00:00', 'syyyy-mm-dd hh24:mi:ss')))
```

힌트를 제거하고 동일 문장을 재실행 했더니, 결합 인덱스가 존재하는 상황에서도 인덱스의 사용을 안 했다.
인덱스 사용이 크게 효율적이지 않기 때문에 옵티마이저가 선택한 사항이다.
만약 동일한 조건에서 SALES2 테이블 보다는 동일한 PROD_ID 끼리 모여 저장되어 있다면?

```
SQL> CREATE INDEX sales_prod_time_ix ON sales(prod_id, time_id) ;
SQL> SELECT /*+ index(s sales_prod_time_ix) */ * FROM sales s
     WHERE prod_id = 120
       AND time_id BETWEEN TO_DATE('2001/01/01','YYYY/MM/DD') AND TO_DATE('2001/12/31','YYYY/MM/DD') ;
SQL> @xplan
-------------------------------------------------------------------------------------------------------------------
| Id  | Operation                    | Name               | Starts | E-Rows | A-Rows |   A-Time   | Buffers |
-------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |                    |      1 |        |   4912 |00:00:00.01 |     488 |
|   1 |  TABLE ACCESS BY INDEX ROWID| SALES              |      1 |   3190 |   4912 |00:00:00.01 |     488 |
|*  2 |   INDEX RANGE SCAN           | SALES_PROD_TIME_IX |      1 |   3190 |   4912 |00:00:00.01 |      68 |
-------------------------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   2 - access("PROD_ID"=120 AND "TIME_ID">=TO_DATE(' 2001-01-01 00:00:00', 'syyyy-mm-dd hh24:mi:ss') AND
                            "TIME_ID"<=TO_DATE(' 2001-12-31 00:00:00', 'syyyy-mm-dd hh24:mi:ss'))
```

동일한 행의 개수를 검색하는 문장이지만 앞서 확인했던 실행 계획보다는 효율적인 I/O가 수행된 것을 확인할 수 있다. 동일한 조건에, 동일한 데이터를 가지고 있더라도 저장된 주소가 얼마나 하나의 블록에 모여서 저장되어 있느냐는 성능상 매우 중요한 부분이다. 때문에 테이블의 재구성도 튜닝의 한 방법으로 사용된다. (또는 Partitioned Table을 사용하여 특정 값이 모여 저장될 수 있도록 할 수도 있다.)

Clustering Factor 가 좋은 상황에서 다음과 같은 문장의 성능은 어떠한가?

```
SQL> SELECT /*+ index_rs_asc(s sales_prod_time_ix) no_index_ss(s) */ * FROM sales s
     WHERE prod_id BETWEEN 120 AND 130 AND time_id = TO_DATE('2001/01/27','YYYY/MM/DD') ;
SQL> @xplan
-------------------------------------------------------------------------------------------------------------------
| Id  | Operation                    | Name               | Starts | E-Rows | A-Rows |   A-Time   | Buffers |
-------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |                    |      1 |        |    380 |00:00:00.01 |     473 |
|   1 |  TABLE ACCESS BY INDEX ROWID| SALES              |      1 |    380 |    380 |00:00:00.01 |     473 |
|*  2 |   INDEX RANGE SCAN           | SALES_PROD_TIME_IX |      1 |    380 |    380 |00:00:00.01 |     439 |
-------------------------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   2 - access("PROD_ID">=120 AND "TIME_ID"=TO_DATE(' 2001-01-27 00:00:00', 'syyyy-mm-dd
              hh24:mi:ss') AND "PROD_ID"<=130)
       filter("TIME_ID"=TO_DATE(' 2001-01-27 00:00:00', 'syyyy-mm-dd hh24:mi:ss'))
```

결과에 비해 인덱스 내에서의 I/O가 상대적으로 크다. 이는 인덱스의 스캔 범위가 넓어져서 발생한 현상이다. SALES_PROD_TIME_IX 인덱스는 (PROD_ID, TIME_ID) 순으로 생성했고 다음의 내용이 저장되어 있다. (현재 상황에 필요한 부분만을 검색했다.)

```
SQL> SET PAUSE ON
SQL> SELECT prod_id, time_id, rowid FROM sales
     WHERE prod_id BETWEEN 120 AND 130 ORDER BY 1, 2, 3 ;
    ROWNUM    PROD_ID TIME_ID  ROWID
---------- ---------- -------- ------------------
...
     14888        120 01/01/26 AAASfoAAFAAABE2ABS
     14889        120 01/01/27 AAASfoAAFAAABE2ABT
     14890        120 01/01/27 AAASfoAAFAAABE2ABU
...
     14987        120 01/01/27 AAASfoAAFAAABG/AAg
     14988        120 01/01/28 AAASfoAAFAAABE2AB3
...
     26073        121 01/01/26 AAASfoAAFAAABG/ACD
     26074        121 01/01/27 AAASfoAAFAAABE7AAY
     26075        121 01/01/27 AAASfoAAFAAABE7AAZ
...
[Ctrl]+C
SQL> SET PAUSE OFF
```

두 조건을 동시에 만족하는 첫 번째 행을(PROD_ID = 120 AND TIME_ID = '01/01/27') 시작으로 스캔이 시작되고, 조건에 만족하지 않는 행(rownum:14,988)이 발견되면 PROD_ID 121이면서 TIME_ID 조건에 만족하는 데이터를 찾아야 한다. 하지만 조건식이 선분 조건을 사용하고 있다면 14,988행의 위치에 PROD_ID 121이 시작될 수도 있으므로 14,988부터 26,073 범위의 불필요한 인덱스 엔트리를 모두 탐색해야 한다. 이러한 인덱스 스캔 작업은 인덱스 내에서의 작업 범위를 넓히므로 인덱스를 효율적으로 사용했다고 볼 수는 없다.

조건식을 정의할 때 점 조건을(비교 연산자: '=', 'IN') 사용하면 특정 값의 범위가 제한된다. 하지만 선분 조건을(비교 연산자: '>', '<', 'BETWEEN', 'LIKE') 사용하면 조건에 참이 아닌 경우가 발견될 때까지 인덱스의 항목들을 모두 비교해야 한다. 특히 결합 인덱스의 선행 컬럼의 조건식이 선분 조건을 사용하고 있다면 이 문제는 더욱더 커질 수 있다. 때문에 조건식을 정의할 때 가급적 점 조건을 이용하는 것이 효율적인 인덱스의 사용 방법이 된다.

만약 위의 문장만을 위해서 인덱스 생성을 고려한다면?

```
SQL> CREATE INDEX sales_time_prod_ix ON sales(time_id, prod_id) ;
SQL> SELECT /*+ index_rs_asc(s sales_time_prod_ix) */ * FROM sales s
     WHERE prod_id BETWEEN 120 AND 130
       AND time_id = TO_DATE('2001/01/27','YYYY/MM/DD') ;
SQL> @xplan
-------------------------------------------------------------------------------------------------------------
| Id  | Operation                   | Name               | Starts | E-Rows | A-Rows |   A-Time   | Buffers |
-------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT            |                    |      1 |        |    380 |00:00:00.01 |      42 |
|   1 |  TABLE ACCESS BY INDEX ROWID| SALES              |      1 |    380 |    380 |00:00:00.01 |      42 |
|*  2 |   INDEX RANGE SCAN          | SALES_TIME_PROD_IX |      1 |    380 |    380 |00:00:00.01 |       8 |
-------------------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   2 - access("TIME_ID"=TO_DATE(' 2001-01-27 00:00:00', 'syyyy-mm-dd hh24:mi:ss') AND
              "PROD_ID">=120 AND "PROD_ID"<=130)
```

확실히 개선된 결과를 확인할 수 있다. 다만, WHERE 절에 정의되는 수많은 조건식을 연산자의 종류에 따라 컬럼 순서를 달리하여 인덱스를 생성한다면 시스템 전체를 관리하는데 있어서는 오히려 악영향을 미칠 수 있다. 인덱스의 설계는 해당 테이블의 접근되는 SQL 문장 전체를 고려하여 최소한으로 생성되어야 한다. 때문에 SQL 작성 시 현재 존재하는 인덱스를 고려하여 최적화된 실행 계획을 사용할 수 있는 문장의 작성 방법이 필요하다.

```
SQL> SELECT /*+ index_rs_asc(s sales_prod_time_ix) */ *
     FROM sales s
     WHERE prod_id IN (120, 121, 122, 123, 124, 125, 126, 127, 128, 129, 130)
       AND time_id = TO_DATE('2001/01/27','YYYY/MM/DD') ;
SQL> @xplan
-------------------------------------------------------------------------------------------------------------------
| Id  | Operation                    | Name               | Starts | E-Rows | A-Rows |   A-Time   | Buffers |
-------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |                    |      1 |        |    380 |00:00:00.01 |      64 |
|   1 |  INLIST ITERATOR             |                    |      1 |        |    380 |00:00:00.01 |      64 |
|   2 |   TABLE ACCESS BY INDEX ROWID| SALES              |     11 |    278 |    380 |00:00:00.01 |      64 |
|*  3 |    INDEX RANGE SCAN          | SALES_PROD_TIME_IX |     11 |    278 |    380 |00:00:00.01 |      30 |
-------------------------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   3 - access((("PROD_ID"=120 OR "PROD_ID"=121 OR "PROD_ID"=122 OR "PROD_ID"=123 OR
              "PROD_ID"=124 OR "PROD_ID"=125 OR "PROD_ID"=126 OR "PROD_ID"=127 OR "PROD_ID"=128 OR
              "PROD_ID"=129 OR "PROD_ID"=130)) AND "TIME_ID"=TO_DATE(' 2001-01-27 00:00:00', 'syyyy-mm-dd
              hh24:mi:ss'))
```

PROD_ID 컬럼의 조건식을 IN 연산자로 변경 후 문장의 성능은 최적화되었다. IN 연산자는 Predicate Information에서 확인되듯이 각 조건을 OR 연산으로 비교한다. 수정하면 다음과 같은 WHERE 절이 사용되었다.

```
('PROD_ID = 120 AND TIME_ID = '01/01/27') OR ('PROD_ID = 121 AND TIME_ID = '01/01/27') OR
('PROD_ID = 122 AND TIME_ID = '01/01/27') OR ('PROD_ID = 123 AND TIME_ID = '01/01/27') OR ..
```

위와 같은 조건식은 인덱스의 불필요한 스캔을 진행하지 않고 필요한 범위만을 스캔할 수 있기 때문에 동일한 결과를 검색할 때 보다 효율적인 I/O가 가능해진다. 물론 IN 연산자에 입력할 값의 종류가 많아진다면 Subquery 등을 이용할 수도 있다. 또는 앞서 확인했던 'INDEX SKIP SCAN' 실행 계획을 사용하는 것도 가능하다.

```
SQL> SELECT /*+ index_ss(s sales_prod_time_ix) */ * FROM sales s
     WHERE prod_id BETWEEN 120 AND 130 AND time_id = TO_DATE('2001/01/27','YYYY/MM/DD') ;
SQL> @xplan
-------------------------------------------------------------------------------------------------------------------
| Id  | Operation                    | Name               | Starts | E-Rows | A-Rows |   A-Time   | Buffers |
-------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |                    |      1 |        |    380 |00:00:00.01 |      70 |
|   1 |  TABLE ACCESS BY INDEX ROWID| SALES              |      1 |    380 |    380 |00:00:00.01 |      70 |
|*  2 |   INDEX SKIP SCAN            | SALES_PROD_TIME_IX |      1 |     64 |    380 |00:00:00.01 |      36 |
-------------------------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   2 - access("PROD_ID">=120 AND "TIME_ID"=TO_DATE(' 2001-01-27 00:00:00', 'syyyy-mm-dd
              hh24:mi:ss') AND "PROD_ID"<=130)
       filter("TIME_ID"=TO_DATE(' 2001-01-27 00:00:00', 'syyyy-mm-dd hh24:mi:ss'))
```

"INDEX SKIP SCAN" 실행 계획은 조건식을 수정하지 않고 실행 계획으로만 성능을 최적화 시킬 수 있기 때문에 결합 인덱스를 사용하는 상황에서 다양하게 응용이 가능하다. 단, 현재 예제처럼 PROD_ID의 구분 값의 종류가 많지 않을 때 의미가 있다. 만약 구분 값의 종류가 많아지면 INDEX SKIP SCAN으로도 성능의 최적화는 불가능하다. 경우에 따라서는 추가적인 인덱스 생성을 고려해야 한다.

```
SQL> SELECT /*+ index(s sales_time_prod_ix) no_index_ss(s) */ *  FROM sales s
     WHERE prod_id = 120
       AND time_id BETWEEN TO_DATE('2001/01/01','YYYY/MM/DD') AND TO_DATE('2001/12/31','YYYY/MM/DD') ;
SQL> @xplan
-------------------------------------------------------------------------------------------------------------------
| Id  | Operation                   | Name               | Starts | E-Rows | A-Rows |   A-Time   | Buffers |
-------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT            |                    |      1 |        |   4912 |00:00:00.03 |    1286 |
|   1 |  TABLE ACCESS BY INDEX ROWID| SALES              |      1 |   3190 |   4912 |00:00:00.03 |    1286 |
|*  2 |   INDEX RANGE SCAN          | SALES_TIME_PROD_IX |      1 |   3190 |   4912 |00:00:00.02 |     866 |
-------------------------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   2 - access("TIME_ID">=TO_DATE(' 2001-01-01 00:00:00', 'syyyy-mm-dd hh24:mi:ss') AND
              "PROD_ID"=120 AND "TIME_ID"<=TO_DATE(' 2001-12-31 00:00:00', 'syyyy-mm-dd hh24:mi:ss'))
       filter("PROD_ID"=120)
```

TIME_ID 컬럼이 선행으로 있을 때 선분 조건을 사용하고 있다. 때문에 인덱스 스캔 범위가 넓고, 불필요한 I/O가 증가된다.

```
SQL> SELECT /*+ index_ss(s sales_time_prod_ix) */ * FROM sales s
     WHERE prod_id = 120
       AND time_id BETWEEN TO_DATE('2001/01/01','YYYY/MM/DD') AND TO_DATE('2001/12/31','YYYY/MM/DD') ;
SQL> @xplan
-------------------------------------------------------------------------------------------------------------------
| Id  | Operation                   | Name               | Starts | E-Rows | A-Rows |   A-Time   | Buffers |
-------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT            |                    |      1 |        |   4912 |00:00:00.01 |    1023 |
|   1 |  TABLE ACCESS BY INDEX ROWID| SALES              |      1 |   3190 |   4912 |00:00:00.01 |    1023 |
|*  2 |   INDEX SKIP SCAN           | SALES_TIME_PROD_IX |      1 |   3190 |   4912 |00:00:00.01 |     603 |
-------------------------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   2 - access("TIME_ID">=TO_DATE(' 2001-01-01 00:00:00', 'syyyy-mm-dd hh24:mi:ss') AND
              "PROD_ID"=120 AND "TIME_ID"<=TO_DATE(' 2001-12-31 00:00:00', 'syyyy-mm-dd hh24:mi:ss'))
       filter("PROD_ID"=120)
```

선행 컬럼의 구분 값이 많으면 SKIP 연산이 많아지므로 "INDEX SKIP SCAN"이 효율적이지 못하다. 차라리 SALES_PROD_TIME_IX 인덱스를 사용하는 것이 올바른 인덱스의 선택이다.

```
SQL> SELECT /*+ index(s sales_prod_time_ix) */ * FROM sales s
     WHERE prod_id = 120
       AND time_id BETWEEN TO_DATE('2001/01/01','YYYY/MM/DD') AND TO_DATE('2001/12/31','YYYY/MM/DD') ;
SQL> @xplan
-------------------------------------------------------------------------------------------------------------
| Id  | Operation                   | Name               | Starts | E-Rows | A-Rows |   A-Time   | Buffers |
-------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT            |                    |      1 |        |   4912 |00:00:00.01 |     488 |
|   1 |  TABLE ACCESS BY INDEX ROWID| SALES              |      1 |   3190 |   4912 |00:00:00.01 |     488 |
|*  2 |   INDEX RANGE SCAN          | SALES_PROD_TIME_IX |      1 |   3190 |   4912 |00:00:00.01 |      68 |
-------------------------------------------------------------------------------------------------------------
...
SQL> DROP TABLE sales2 PURGE ;
SQL> CREATE TABLE sales2 AS SELECT * FROM sales ORDER BY prod_id, time_id ;
SQL> CREATE INDEX sales2_prod_time_ix ON sales2(prod_id, time_id) ;

SQL> SELECT /*+ index(s sales2_prod_time_ix) */ * FROM sales2 s
     WHERE prod_id = 120
       AND time_id BETWEEN TO_DATE('2001/01/01','YYYY/MM/DD') AND TO_DATE('2001/12/31','YYYY/MM/DD') ;
SQL> @xplan
-------------------------------------------------------------------------------------------------------------
| Id  | Operation                   | Name               | Starts | E-Rows | A-Rows |   A-Time   | Buffers |
-------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT            |                    |      1 |        |   4912 |00:00:00.01 |     143 |
|   1 |  TABLE ACCESS BY INDEX ROWID| SALES2             |      1 |   2500 |   4912 |00:00:00.01 |     143 |
|*  2 |   INDEX RANGE SCAN          | SALES2_PROD_TIME_IX |      1 |   2500 |   4912 |00:00:00.01 |      68 |
-------------------------------------------------------------------------------------------------------------
...
SQL> DROP TABLE sales2 PURGE ;
SQL> DROP INDEX sales_prod_time_ix ;
SQL> DROP INDEX sales_time_prod_ix ;
```

› 결론

1. 결합 인덱스 생성 시 컬럼의 순서는 매우 중요하다.
   - 조건식에 의해 항상 사용되는 컬럼을 후보군으로 선정
   - 필터링이 뛰어난 컬럼, 점 조건식에 자주 사용되는 컬럼, 정렬에 사용되는 컬럼등을 선행 컬럼으로 배치
2. 문장마다 필요한 인덱스를 추가할 수 없으므로 조건식은 가급적 점 조건식을 최대한 많이 사용

실행 계획의 변경만으로 튜닝 목표에 도달된다면 추가적인 튜닝 기법은 고민할 필요가 없다. 실행 계획의 변경만으로 해결이 불가능하다면 문장의 수정을 고민하고 최후의 선택으로 조건식에 최적화된 인덱스를 구성한다.

〉문제 8.

다음의 문장을 튜닝 하시오.

```
SQL〉 SELECT /*+ index(s(time_id)) */ *
     FROM sales s
     WHERE cust_id = 19010
       AND channel_id = 3
       AND time_id IS NOT NULL
       AND rownum <= 20 ;

   PROD_ID    CUST_ID TIME_ID  CHANNEL_ID   PROMO_ID QUANTITY_SOLD AMOUNT_SOLD
---------- ---------- -------- ---------- ---------- ------------- -----------
        18      19010 98/11/20          3        999             1     1655.65
        31      19010 00/10/17          3        999             1        9.33
        31      19010 00/12/17          3        999             1        8.86
        20      19010 01/02/17          3        351             1      628.89
        32      19010 01/02/17          3        351             1       71.29
        33      19010 01/02/17          3        351             1       46.22
        35      19010 01/02/17          3        351             1       54.21
        36      19010 01/02/17          3        351             1       49.26
        37      19010 01/02/17          3        351             1        59.6
        38      19010 01/02/17          3        351             1       27.18
        39      19010 01/02/17          3        351             1       31.39
        42      19010 01/02/17          3        351             1       48.19
        43      19010 01/02/17          3        351             1       48.22
        45      19010 01/02/17          3        351             1       47.17
        47      19010 01/02/17          3        351             1       29.98
        48      19010 01/02/17          3        351             1       10.36
        14      19010 01/02/21          3        351             1     1181.36
        15      19010 01/02/21          3        351             1      969.01
        16      19010 01/02/21          3        351             1      304.26
        17      19010 01/02/21          3        351             1     1289.12

20 rows selected.

SQL〉 @xplan
-------------------------------------------------------------------------------------------------------------------
| Id  | Operation                     | Name          | Starts | E-Rows | A-Rows |   A-Time   | Buffers |
-------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT              |               |      1 |        |     20 |00:00:00.43 |   40121 |
|*  1 |  COUNT STOPKEY                |               |      1 |        |     20 |00:00:00.43 |   40121 |
|*  2 |   TABLE ACCESS BY INDEX ROWID| SALES         |      1 |     20 |     20 |00:00:00.43 |   40121 |
|   3 |    INDEX FULL SCAN            | SALES_TIME_IX |      1 |    694K|    694K|00:00:00.19 |    1845 |
-------------------------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   1 - filter(ROWNUM<=20)
   2 - filter(("CUST_ID"=19010 AND "CHANNEL_ID"=3))
```

› 문제 사항

```
SQL› SELECT COUNT(*) AS cnt,
            COUNT(CASE WHEN time_id IS NOT NULL THEN 1 END) AS time_id,
            COUNT(CASE WHEN channel_id = 3      THEN 1 END) AS channel_id,
            COUNT(CASE WHEN cust_id = 19010     THEN 1 END) AS cust_id,
            COUNT(CASE WHEN time_id IS NOT NULL
                       AND channel_id = 3
                       AND cust_id = 19010      THEN 1 END ) AS condition
     FROM sales ;

       CNT    TIME_ID CHANNEL_ID    CUST_ID  CONDITION
---------- ---------- ---------- ---------- ----------
    918843     918843     540328        283         89
```

3개의 조건 중 필터링이 가장 뛰어난 조건은 cust_id 컬럼의 조건절이나 힌트를 통해서 필터링이 가장 낮은 time_id 컬럼의 인덱스를 사용하게 함으로 불필요한 I/O가 증가되었다. 또한 IS NOT NULL 조건식은 인덱스의 Full Scan을 사용할 수밖에 없으므로 필터링이 아주 뛰어날 경우를 제외하면 불필요한 실행 계획일 수 있다.

〉답안 1. 필터링이 뛰어난 SALES_CUST_IX 인덱스를 사용함으로 작업량을 줄임

```
SQL〉 SELECT /*+ index(s(cust_id)) */ *
     FROM sales s
     WHERE cust_id = 19010
       AND channel_id = 3
       AND time_id IS NOT NULL
       AND rownum <= 20 ;

   PROD_ID    CUST_ID TIME_ID   CHANNEL_ID   PROMO_ID QUANTITY_SOLD AMOUNT_SOLD
---------- ---------- --------- ---------- ---------- ------------- -----------
        18      19010 20-NOV-98          3        999             1     1655.65
        31      19010 17-OCT-00          3        999             1        9.33
        31      19010 17-DEC-00          3        999             1        8.86
        14      19010 21-FEB-01          3        351             1     1181.36
        15      19010 21-FEB-01          3        351             1      969.01
        15      19010 24-FEB-01          3        351             1      969.01
        16      19010 21-FEB-01          3        351             1      304.26
        17      19010 21-FEB-01          3        351             1     1289.12
        17      19010 24-FEB-01          3        351             1     1289.12
...
20 rows selected.
SQL〉 @xplan
-------------------------------------------------------------------------------------------------------
| Id  | Operation                    | Name          | Starts | E-Rows | A-Rows |   A-Time   | Buffers |
-------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |               |      1 |        |     20 |00:00:00.01 |      40 |
|*  1 |  COUNT STOPKEY               |               |      1 |        |     20 |00:00:00.01 |      40 |
|*  2 |   TABLE ACCESS BY INDEX ROWID| SALES         |      1 |     20 |     20 |00:00:00.01 |      40 |
|*  3 |    INDEX RANGE SCAN          | SALES_CUST_IX |      1 |     62 |     62 |00:00:00.01 |       4 |
-------------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   1 - filter(ROWNUM<=20)
   2 - filter("CHANNEL_ID"=3)
   3 - access("CUST_ID"=19010)
```

필터링이 가장 좋은 컬럼의 인덱스를 사용함으로 I/O 의 횟수가 최적화되고 작업량의 감소로 처리 시간 역시 단축됨을 확인 할수 있다. 하지만, 튜닝 전/후의 문장이 실행된 결과를 보면 두 문장은 동일한 결과를 가져오는 문장은 아니다.

〉주의 : SQL 문장의 의도 파악

튜닝 전 문장을 확인하면 필터링 효과가 전혀 없는 time_id 컬럼의 인덱스를 사용하도록 유도하였다. 이는 문장을 작성한 이의 실수일 수도 있겠지만 Sort Operation을 제거하기 위한 인덱스 사용일 수도 있다. 즉, 튜닝 대상의 문장은 cust_id와 channel_id의 조건식에 만족하는 행들 중에 구매일자를 기준으로 정렬하여 가장 빠른 구매 내역 20개를 검색하는 문장이다.

〉답안 2. 조건과 정렬상태를 보장할 수 있는 결합 인덱스 사용

인덱스를 추가하여 튜닝을 수행할 경우에는 가급적 불필요한 컬럼의 추가는 최대한 자제한다. channel_id 컬럼 역시 필터링이 뛰어나다고 할 순 없으므로 다음과 같은 인덱스를 생성한다.

```
SQL〉 CREATE INDEX sales_x01 ON sales(cust_id, time_id) ;
Index created.
SQL〉 SELECT /*+ index(s sales_x01) */ *
     FROM sales s
     WHERE cust_id = 19010
       AND channel_id = 3
       AND time_id IS NOT NULL
       AND rownum <= 20 ;

   PROD_ID    CUST_ID TIME_ID   CHANNEL_ID   PROMO_ID QUANTITY_SOLD AMOUNT_SOLD
---------- ---------- --------- ---------- ---------- ------------- -----------
        18      19010 20-NOV-98          3        999             1     1655.65
        31      19010 17-OCT-00          3        999             1        9.33
        31      19010 17-DEC-00          3        999             1        8.86
        20      19010 17-FEB-01          3        351             1      628.89
        32      19010 17-FEB-01          3        351             1       71.29
        33      19010 17-FEB-01          3        351             1       46.22
        35      19010 17-FEB-01          3        351             1       54.21
        36      19010 17-FEB-01          3        351             1       49.26
        37      19010 17-FEB-01          3        351             1        59.6
...
20 rows selected.

SQL〉 @xplan
-------------------------------------------------------------------------------------------------------------
| Id  | Operation                    | Name      | Starts | E-Rows | A-Rows |   A-Time   | Buffers |
-------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |           |      1 |        |     20 |00:00:00.01 |      47 |
|*  1 |  COUNT STOPKEY               |           |      1 |        |     20 |00:00:00.01 |      47 |
|*  2 |   TABLE ACCESS BY INDEX ROWID| SALES     |      1 |     20 |     20 |00:00:00.01 |      47 |
|*  3 |    INDEX RANGE SCAN          | SALES_X01 |      1 |     67 |     67 |00:00:00.01 |       4 |
-------------------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   1 - filter(ROWNUM<=20)
   2 - filter("CHANNEL_ID"=3)
   3 - access("CUST_ID"=19010)

SQL〉 DROP INDEX sales_x01 ;
Index dropped.
```

› 답안 3.

문장이 의도하는 바를 파악하고 동일한 결과를 검색할 수 있도록 SELECT 문장을 수정한다.

```
SQL> SELECT *
     FROM ( SELECT /*+ index(s(cust_id)) */ *
              FROM sales s
             WHERE cust_id    = 19010
               AND channel_id = 3
             ORDER BY time_id, rowid )
     WHERE rownum <= 20 ;

   PROD_ID    CUST_ID TIME_ID   CHANNEL_ID   PROMO_ID QUANTITY_SOLD AMOUNT_SOLD
---------- ---------- --------- ---------- ---------- ------------- -----------
        18      19010 20-NOV-98          3        999             1     1655.65
        31      19010 17-OCT-00          3        999             1        9.33
        31      19010 17-DEC-00          3        999             1        8.86
        20      19010 17-FEB-01          3        351             1      628.89
        32      19010 17-FEB-01          3        351             1       71.29
        33      19010 17-FEB-01          3        351             1       46.22
        35      19010 17-FEB-01          3        351             1       54.21
        36      19010 17-FEB-01          3        351             1       49.26
        37      19010 17-FEB-01          3        351             1        59.6
...
20 rows selected.

SQL> @xplan
-------------------------------------------------------------------------------------------------------------------------------------------
| Id  | Operation                     | Name          | Starts | E-Rows | A-Rows |   A-Time   | Buffers |  OMem |  1Mem | Used-Mem |
-------------------------------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT              |               |      1 |        |     20 |00:00:00.01 |     168 |       |       |          |
|*  1 |  COUNT STOPKEY                |               |      1 |        |     20 |00:00:00.01 |     168 |       |       |          |
|   2 |   VIEW                        |               |      1 |     89 |     20 |00:00:00.01 |     168 |       |       |          |
|*  3 |    SORT ORDER BY STOPKEY      |               |      1 |     89 |     20 |00:00:00.01 |     168 |  2048 |  2048 | 2048  (0)|
|*  4 |     TABLE ACCESS BY INDEX ROWID| SALES         |      1 |     89 |     89 |00:00:00.01 |     168 |       |       |          |
|*  5 |      INDEX RANGE SCAN         | SALES_CUST_IX |      1 |    283 |    283 |00:00:00.01 |       4 |       |       |          |
-------------------------------------------------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   1 - filter(ROWNUM<=20)
   3 - filter(ROWNUM<=20)
   4 - filter("CHANNEL_ID"=3)
   5 - access("CUST_ID"=19010)
```

› 결론

인덱스의 추가는 차후 DBA의 관리 요소가 증가될 수 있다. 다른 문장에서도 사용 가능한 인덱스라면 인덱스를 추가하는 것도 좋은 방법일 수 있으나 기존의 인덱스를 활용할 수 있는 방법이 있다면 기존 인덱스를 이용하여 최적화를 시도한다. 튜닝은 최대한의 작업량을 떨어뜨리기보단 정확한 목표에 근접시키는 것이 중요하다.

Join Tuning

Join 문은 두 집합의 행을 조인 조건에 따라 연결된 행 집합을 생성한다. 이러한 조인을 수행할 때 선택되는 조인의 종류는 튜닝 시 매우 중요할 수 있으므로 그 종류의 특징과 장단점을 확인한다.

```
[orcl:adsql]$ sqlplus user01/oracle
SQL> CREATE TABLE cp_emp   AS SELECT * FROM emp ;
SQL> CREATE TABLE cp_dept  AS SELECT * FROM dept ;
```

※ Nested Loops Join

- 선행 테이블(Outer Table) 접근하여 후보 행을 선택하고, 조인 조건에 만족하는 후행 테이블(Inner Table)을 반복적으로 접근하여 조인 결과 생성
- Inner Table의 조인 컬럼에 인덱스가 존재하지 않으면 "FULL TABLE SCAN"이 사용되므로 인덱스 필요
- FIRST_ROWS 상황에 최적화 가능하며, 많은 양의 조인 결과 생성 시 반복 접근으로 인한 성능 저하 발생

```
SQL> SELECT /*+ leading(d) use_nl(e) */ d.deptno, d.dname, e.empno, e.ename, e.sal
     FROM cp_dept d, cp_emp e
     WHERE d.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 |      17 |
|   1 |  NESTED LOOPS      |         |      1 |     14 |     14 |00:00:00.01 |      17 |
|   2 |   TABLE ACCESS FULL| CP_DEPT |      1 |      4 |      4 |00:00:00.01 |       4 |
|*  3 |   TABLE ACCESS FULL| CP_EMP  |      4 |      4 |     14 |00:00:00.01 |      13 |
-------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   3 - filter("D"."DEPTNO"="E"."DEPTNO")
```

ID: 2 에서 CP_DEPT 테이블을 먼저 액세스하고, 발견된 후보 행이 4개이다. 때문에 CP_EMP 테이블은 4번의 "FULL TABLE SCAN"이 발생하였다. 조인 컬럼에 인덱스가 없으므로 조건식에 만족하는 행을 찾기 위한 유일한 방법인 "FULL TABLE SCAN"이 사용된 것이다. 때문에 이러한 Nested Loops Join을 최적화하기 위해서는 Inner Table에 인덱스가 반드시 필요하다.

```
SQL> CREATE INDEX cp_dept_deptno ON cp_dept(deptno) ;
SQL> CREATE INDEX cp_emp_deptno ON cp_emp(deptno) ;
```

```
SQL> SELECT /*+ leading(d) use_nl(e) */ d.deptno, d.dname, e.empno, e.ename, e.sal
     FROM cp_dept d, cp_emp e
     WHERE d.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 |       9 |
|   1 |  NESTED LOOPS                |               |      1 |        |     14 |00:00:00.01 |       9 |
|   2 |   NESTED LOOPS               |               |      1 |     14 |     14 |00:00:00.01 |       7 |
|   3 |    TABLE ACCESS FULL         | CP_DEPT       |      1 |      4 |      4 |00:00:00.01 |       4 |
|*  4 |    INDEX RANGE SCAN          | CP_EMP_DEPTNO |      4 |      5 |     14 |00:00:00.01 |       3 |
|   5 |   TABLE ACCESS BY INDEX ROWID| CP_EMP        |     14 |      4 |     14 |00:00:00.01 |       2 |
-------------------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   4 - access("D"."DEPTNO"="E"."DEPTNO")
```

조인에 사용되는 두 컬럼에 각각 인덱스를 생성했지만 CP_DEPT 테이블은 여전히 "FULL TABLE SCAN"이 사용된다. CP_DEPT 테이블은 후보 행을 공급해야 하는 집합이므로 별도의 조건이 존재하지 않는다. (Predicate Information 존재 안 함) 때문에 Outer Table의 조인 컬럼은 인덱스가 반드시 필요한 것은 아니다.

```
SQL> SELECT /*+ leading(e) use_nl(d) */ d.deptno, d.dname, e.empno, e.ename, e.sal
     FROM cp_dept d, cp_emp e
     WHERE d.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 |      10 |
|   1 |  NESTED LOOPS                |               |      1 |        |     14 |00:00:00.01 |      10 |
|   2 |   NESTED LOOPS               |               |      1 |     14 |     14 |00:00:00.01 |       8 |
|   3 |    TABLE ACCESS FULL         | CP_EMP        |      1 |     14 |     14 |00:00:00.01 |       4 |
|*  4 |    INDEX RANGE SCAN          | CP_DEPT_DEPTNO |     14 |      1 |     14 |00:00:00.01 |       4 |
|   5 |   TABLE ACCESS BY INDEX ROWID| CP_DEPT       |     14 |      1 |     14 |00:00:00.01 |       2 |
-------------------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   4 - access("D"."DEPTNO"="E"."DEPTNO")
```

조인의 순서를 변경하여 CP_EMP 테이블을 Outer Table로 사용할 수도 있다. 하지만 후보 행이 상대적으로 많아져서 오히려 작업량은 증가하는 것을 확인할 수 있다. 어떠한 조인의 종류를 사용하더라도 Outer Table은 더 적은 후보 행을 가지고 있는 쪽을 사용하는 것이 좋다. 단, 추가적인 비 조인 술어를 통해 추가적인 조건이 정의될 수도 있으므로 특정 조인 순서를 고정화 시키는 것은 피해야 하며, 어느 쪽이 Outer Table로 결정되더라도 Nested Loops Join의 최적화를 위해 조인 컬럼에는 인덱스가 반드시 존재해야 한다.

※ Sort Merge Join

- 각각의 집합을 한 번씩 액세스하여 조인 컬럼을 기준으로 정렬하고 병합함
- 정렬 작업이 완료되기 전까지는 조인 결과의 검색 불가능
- ALL_ROWS 상황에 최적화 가능하고, 정렬 작업의 부담이 클수록 성능 저하
- Sort Merge Join 보다는 Hash Join을 사용하고, Hash Join이 불가능할 경우에 Sort Merge Join을 고려

```
SQL> SELECT /*+ leading(d) use_merge(e) no_index(d) */ d.deptno, d.dname, e.empno, e.ename, e.sal
     FROM cp_dept d, cp_emp e
     WHERE d.deptno = e.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 |       6 |       |       |          |
|   1 |  MERGE JOIN         |         |      1 |     14 |     14 |00:00:00.01 |       6 |       |       |          |
|   2 |   SORT JOIN         |         |      1 |      4 |      4 |00:00:00.01 |       3 |  2048 |  2048 | 2048  (0)|
|   3 |    TABLE ACCESS FULL| CP_DEPT |      1 |      4 |      4 |00:00:00.01 |       3 |       |       |          |
|*  4 |   SORT JOIN         |         |      4 |     14 |     14 |00:00:00.01 |       3 |  2048 |  2048 | 2048  (0)|
|   5 |    TABLE ACCESS FULL| CP_EMP  |      1 |     14 |     14 |00:00:00.01 |       3 |       |       |          |
-------------------------------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   4 - access("D"."DEPTNO"="E"."DEPTNO")
       filter("D"."DEPTNO"="E"."DEPTNO")
```

조인의 순서를 변경하더라도 각 집합의 접근은 한 번씩 수행하기 때문에 I/O의 변화는 없다. 단, 정렬이 끝나기 전까지는 조인 결과를 만들 수 없기 때문에 다른 조인 종류에 비해 자원 소모가 가장 클 수 있다. 때문에 Sort Merge Join은 ALL_ROWS 상황에서 Hash Join을 사용할 수 없는 경우에만 사용을 고려한다. (만약 조인 컬럼을 기준으로 반드시 정렬된 결과를 확인해야 하는 경우라면 Sort Merge Join을 사용할 수도 있다.)

또한 NO_INDEX 힌트를 제거하면 위의 실행 계획에서 CP_DEPT 테이블의 접근은 DEPTNO 컬럼의 인덱스를 "INDEX FULL SCAN"으로 접근하는 실행 계획을 확인할 수도 있다. 이는 인덱스를 통해 정렬된 상태를 얻고자 함이며 Optimizer가 실행 계획을 생성할 때 결정한 사항이다. 경우에 따라서는 정렬 작업을 실제 수행하는 것이 더 좋은 경우도 있으므로 필요에 따라 NO_INDEX 힌트의 사용도 가능하다.

※ Hash Join

- Outer Table을 먼저 액세스하여 Hash Table을 생성하고 생성하고, Inner Table을 접근하여 동일한 Hash key를 갖는 행끼리 조인 결과 생성
- ALL_ROWS 상황에 최적이지만 Euqi-Join 에서만 사용이 가능

```
SQL> SELECT /*+ leading(d) use_hash(e) */ d.deptno, d.dname, e.empno, e.ename, e.sal
     FROM cp_dept d, cp_emp e WHERE d.deptno = e.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 |  HASH JOIN         |         |      1 |     14 |     14 |00:00:00.01 |       7 |   888K|   888K|  728K (0)|
|   2 |   TABLE ACCESS FULL| CP_DEPT |      1 |      4 |      4 |00:00:00.01 |       3 |       |       |          |
|   3 |   TABLE ACCESS FULL| CP_EMP  |      1 |     14 |     14 |00:00:00.01 |       4 |       |       |          |
-------------------------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   1 - access("D"."DEPTNO"="E"."DEPTNO")
```

조인 컬럼에 인덱스가 있더라도 "FULL TABLE SCAN"을 선호한다. (추가 조건식이 존재할 경우 인덱스 사용 가능)
각 집합은 한 번씩 액세스 되며 먼저 액세스 된 테이블의 조인 컬럼의 구분 값에 따라 Hash Table을 생성한다. 때문에 후보 행이 더 적은 집합을 먼저 액세스하는 것이 Hash Table의 크기를 줄이고, Hash Join의 성능을 향상화 시키는 방법이다. 하지만 Non-Equi Join을 사용할 경우에는 Hash Join이 불가능하다.

```
SQL> SELECT /*+ leading(e) use_hash(s) */ e.empno, e.ename, e.sal, s.grade
     FROM cp_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 |      33 |
|   1 |  NESTED LOOPS      |          |      1 |      1 |     14 |00:00:00.01 |      33 |
|   2 |   TABLE ACCESS FULL| CP_EMP   |      1 |     14 |     14 |00:00:00.01 |       4 |
|*  3 |   TABLE ACCESS FULL| SALGRADE |     14 |      1 |     14 |00:00:00.01 |      29 |
-----------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   3 - filter(("E"."SAL">="S"."LOSAL" AND "E"."SAL"<="S"."HISAL"))
```

Hash Join을 유도하도록 힌트를 지정했으나 Non-Equi Join 문이기 때문에 Hash Join을 사용하지 못했다.

```
SQL> DROP TABLE cp_dept PURGE ;
SQL> DROP TABLE cp_emp PURGE ;
```

※ Join 순서 및 종류에 따른 튜닝 방법

문제. 다음 문장을 Nested Loops Join을 사용하여 성능을 최적화 시키시오.

```
SQL> SELECT e.employee_id, e.last_name, e.job_id, d.department_id, d.department_name
     FROM employees e, departments d
     WHERE e.department_id = d.department_id
       AND d.location_id   = 1700
       AND e.job_id IN ('FI_ACCOUNT','PU_CLERK');
...
10 rows selected.
```

일반적으로 조인 문장에는 조인 술어 이외에도 비조인 술어가 함께 사용된다. 때문에 Outer Table을 반드시 "FULL TABLE SCAN"으로 접근할 필요는 없다. 두 집합에 어느 쪽이 더 적은 후보행이 있는지 확인하여 조인 순서를 결정한다.

```
SQL> SELECT COUNT(*)
     FROM employees
     WHERE job_id IN ('FI_ACCOUNT','PU_CLERK') ;
  COUNT(*)
----------
        10

SQL> SELECT COUNT(*)
     FROM departments
     WHERE location_id = 1700 ;
  COUNT(*)
----------
        21
```

조인 결과는 10개의 행이며 EMPLOYEES 테이블의 일반 조건은 10개, DEPARTMENTS 테이블의 일반 조건은 21개의 행이 조건에 만족한다. 후보행이 더 적은 EMPLOYEES 테이블을 Driving Table로 사용하는 것이 좋을까?

```
SQL> SELECT /*+ leading(e) use_nl(d) index(e(job_id)) index(d(department_id)) */
            e.employee_id, e.last_name, e.job_id, d.department_id, d.department_name
     FROM employees e, departments d
     WHERE e.department_id = d.department_id
       AND d.location_id   = 1700
       AND e.job_id IN ('FI_ACCOUNT','PU_CLERK') ;
SQL> @xplan
-------------------------------------------------------------------------------------------------------------
| Id  | Operation                     | Name        | Starts | E-Rows | A-Rows |   A-Time   | Buffers |
-------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT              |             |      1 |        |     10 |00:00:00.01 |      19 |
|   1 |  NESTED LOOPS                 |             |      1 |        |     10 |00:00:00.01 |      19 |
|   2 |   NESTED LOOPS                |             |      1 |      5 |     10 |00:00:00.01 |       9 |
|   3 |    INLIST ITERATOR            |             |      1 |        |     10 |00:00:00.01 |       5 |
|   4 |     TABLE ACCESS BY INDEX ROWID| EMPLOYEES   |      2 |     11 |     10 |00:00:00.01 |       5 |
|*  5 |      INDEX RANGE SCAN         | EMPL_JOB_IX |      2 |     11 |     10 |00:00:00.01 |       3 |
|*  6 |    INDEX UNIQUE SCAN          | DEPT_ID_PK  |     10 |      1 |     10 |00:00:00.01 |       4 |
|*  7 |   TABLE ACCESS BY INDEX ROWID | DEPARTMENTS |     10 |      1 |     10 |00:00:00.01 |      10 |
-------------------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   5 - access(("E"."JOB_ID"='FI_ACCOUNT' OR "E"."JOB_ID"='PU_CLERK'))
   6 - access("E"."DEPARTMENT_ID"="D"."DEPARTMENT_ID")
   7 - filter("D"."LOCATION_ID"=1700)


SQL> SELECT /*+ leading(d) use_nl(e) index(d(location_id)) */
            e.employee_id, e.last_name, e.job_id, d.department_id, d.department_name
     FROM employees e, departments d
     WHERE e.department_id = d.department_id
       AND d.location_id   = 1700
       AND e.job_id IN ('FI_ACCOUNT','PU_CLERK');
SQL> @xplan
-------------------------------------------------------------------------------------------------------------
| Id  | Operation                     | Name            | Starts | E-Rows | A-Rows |   A-Time   | Buffers |
-------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT              |                 |      1 |        |     10 |00:00:00.01 |      11 |
|   1 |  NESTED LOOPS                 |                 |      1 |        |     10 |00:00:00.01 |      11 |
|   2 |   NESTED LOOPS                |                 |      1 |      5 |     18 |00:00:00.01 |       9 |
|   3 |    TABLE ACCESS BY INDEX ROWID| DEPARTMENTS     |      1 |      4 |     21 |00:00:00.01 |       4 |
|*  4 |     INDEX RANGE SCAN          | DEPT_LOCATION_IX|      1 |      4 |     21 |00:00:00.01 |       2 |
|*  5 |    INDEX RANGE SCAN           | EMPL_DEPTNO_IX  |     21 |     10 |     18 |00:00:00.01 |       5 |
|*  6 |   TABLE ACCESS BY INDEX ROWID | EMPLOYEES       |     18 |      1 |     10 |00:00:00.01 |       2 |
-------------------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   4 - access("D"."LOCATION_ID"=1700)
   5 - access("E"."DEPARTMENT_ID"="D"."DEPARTMENT_ID")
   6 - filter(("E"."JOB_ID"='FI_ACCOUNT' OR "E"."JOB_ID"='PU_CLERK'))
```

› 확인 사항

첫 번째 문장은 총 19개의 I/O가 수행되었고, 두 번째 문장은 11개의 I/O가 수행되었다. Looping의 횟수를 보면 첫 번째 문장이 더 적은 Looping을 하였으나 오히려 작업량은 더 크게 나타난다. 이유는?

해당 작업은 Oracle Database의 버전에 따라서 약간은 다른 결과를 보여줄 수도 있다. 9i 버전부터 Nested Loops Join 의 성능을 높이고자 Table Prefetching, 11g에서는 Batching NLJ 기능이 추가되었다. 때문에 어떠한 Nested Loops Join이 사용되었는지에 따라서 그 결과는 성능 차이를 보여주기도 한다.

다음의 결과는 동일한 문장을 Oracle Database 10g 버전에서 수행한 결과이다.

```
-------------------------------------------------------------------------------------------------------------
| Id  | Operation                     | Name        | Starts | E-Rows | A-Rows |   A-Time   | Buffers |
-------------------------------------------------------------------------------------------------------------
|   1 |  NESTED LOOPS                 |             |      1 |     11 |     10 |00:00:00.01 |      26 |
|   2 |   INLIST ITERATOR             |             |      1 |        |     10 |00:00:00.01 |       6 |
|   3 |    TABLE ACCESS BY INDEX ROWID| EMPLOYEES   |      2 |     11 |     10 |00:00:00.01 |       6 |
|*  4 |     INDEX RANGE SCAN          | EMP_JOB_IX  |      2 |     11 |     10 |00:00:00.01 |       3 |
|*  5 |   TABLE ACCESS BY INDEX ROWID | DEPARTMENTS |     10 |      1 |     10 |00:00:00.01 |      20 |
|*  6 |    INDEX UNIQUE SCAN          | DEPT_ID_PK  |     10 |      1 |     10 |00:00:00.01 |      10 |
-------------------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   4 - access(("E"."JOB_ID"='FI_ACCOUNT' OR "E"."JOB_ID"='PU_CLERK'))
   5 - filter("D"."LOCATION_ID"=1700)
   6 - access("E"."DEPARTMENT_ID"="D"."DEPARTMENT_ID")

-------------------------------------------------------------------------------------------------------------
| Id  | Operation                     | Name              | Starts | E-Rows | A-Rows |   A-Time   | Buffers |
-------------------------------------------------------------------------------------------------------------
|*  1 |  TABLE ACCESS BY INDEX ROWID  | EMPLOYEES         |      1 |      1 |     10 |00:00:00.01 |      30 |
|   2 |   NESTED LOOPS                |                   |      1 |     11 |     40 |00:00:00.01 |      26 |
|   3 |    TABLE ACCESS BY INDEX ROWID| DEPARTMENTS       |      1 |     21 |     21 |00:00:00.01 |       4 |
|*  4 |     INDEX RANGE SCAN          | DEPT_LOCATION_IX  |      1 |     21 |     21 |00:00:00.01 |       2 |
|*  5 |    INDEX RANGE SCAN           | EMP_DEPARTMENT_IX |     21 |     10 |     18 |00:00:00.01 |      22 |
-------------------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   1 - filter(("E"."JOB_ID"='FI_ACCOUNT' OR "E"."JOB_ID"='PU_CLERK'))
   4 - access("D"."LOCATION_ID"=1700)
   5 - access("E"."DEPARTMENT_ID"="D"."DEPARTMENT_ID")
```

Oracle 10g DB에서는 EMPLOYEES 테이블이 먼저 Driving 된 실행 계획이 더 좋은 성능을 보여준다. 즉, 버전에 따라 동일한 조인의 종류 및 순서에서도 성능은 다를 수 있다. 때문에 올바른 조인 종류 및 순서를 상황에 맞게 선택해야 한다. 그렇다면 이러한 차이는 왜 생길까?

- Oracle 10g DB 실행 계획 (EMPLOYEES -> DEPARTMENTS)

```
-------------------------------------------------------------------------------------------------------
| Id  | Operation                     | Name        | Starts | E-Rows | A-Rows |   A-Time   | Buffers |
-------------------------------------------------------------------------------------------------------
|   1 |  NESTED LOOPS                 |             |      1 |     11 |     10 |00:00:00.01 |      26 |
|   2 |   INLIST ITERATOR             |             |      1 |        |     10 |00:00:00.01 |       6 |
|   3 |    TABLE ACCESS BY INDEX ROWID| EMPLOYEES   |      2 |     11 |     10 |00:00:00.01 |       6 |
|*  4 |     INDEX RANGE SCAN          | EMP_JOB_IX  |      2 |     11 |     10 |00:00:00.01 |       3 |
|*  5 |   TABLE ACCESS BY INDEX ROWID | DEPARTMENTS |     10 |      1 |     10 |00:00:00.01 |      20 |
|*  6 |    INDEX UNIQUE SCAN          | DEPT_ID_PK  |     10 |      1 |     10 |00:00:00.01 |      10 |
-------------------------------------------------------------------------------------------------------
```

위의 실행 계획에서 Driving 테이블은 EMPLOYEES이고 Inner Table 인덱스를 후보 행의 개수만큼 10번의 반복 접근이 일어났다. 이때 사용된 I/O의 개수가 10개인 것을 확인할 수 있다. 즉, 후보 행이 하나씩 공급될 때마다 10번의 인덱스 블록을 접근한 상황이다. 그렇게 확보된 10개의 ROWID를 이용하여 DEPARTMENTS 테이블에 접근했고, DEPARTMENTS 테이블에서 수행된 I/O 횟수가 10개이다. 즉, 인덱스를 포함하여 DEPARTMENTS 테이블에 접근하는데 20개의 I/O가 발생했다.

- Oracle 11g DB 실행 계획 (EMPLOYEES -> DEPARTMENTS)

```
-------------------------------------------------------------------------------------------------------
| Id  | Operation                     | Name        | Starts | E-Rows | A-Rows |   A-Time   | Buffers |
-------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT              |             |      1 |        |     10 |00:00:00.01 |      19 |
|   1 |  NESTED LOOPS                 |             |      1 |        |     10 |00:00:00.01 |      19 |
|   2 |   NESTED LOOPS                |             |      1 |      5 |     10 |00:00:00.01 |       9 |
|   3 |    INLIST ITERATOR            |             |      1 |        |     10 |00:00:00.01 |       5 |
|   4 |     TABLE ACCESS BY INDEX ROWID| EMPLOYEES   |      2 |     11 |     10 |00:00:00.01 |       5 |
|*  5 |      INDEX RANGE SCAN         | EMPL_JOB_IX |      2 |     11 |     10 |00:00:00.01 |       3 |
|*  6 |    INDEX UNIQUE SCAN          | DEPT_ID_PK  |     10 |      1 |     10 |00:00:00.01 |       4 |
|*  7 |   TABLE ACCESS BY INDEX ROWID | DEPARTMENTS |     10 |      1 |     10 |00:00:00.01 |      10 |
-------------------------------------------------------------------------------------------------------
```

Oracle 11g DB부터는 Batching NLJ를 이용하여 Outer Table과 Inner Table의 조인 컬럼의 인덱스를 먼저 조인한다. 이때 DEPT_ID_PK 인덱스를 액세스하면서 발생한 I/O의 수는 4개이다. Oracle 10g 버전보다 6개의 I/O 횟수가 감소한 것을 확인할 수 있다. 인덱스에서 확보된 10개의 ROWID를 이용하여 DEPARTMENTS 테이블의 액세스에는 10개의 I/O가 사용되었다. 앞서 확인한 10g 버전의 실행 계획은 Inner Table에 접근할 때 20개의 I/O가 사용되었지만, 11g 버전에서는 14개의 I/O가 사용되었다. 이는 인덱스의 블록을 Pinning 하여 반복적인 블록의 접근 수를 줄일 수 있기 때문에 일어난 현상이다. 보다 자세한 Batching NLJ의 처리 방식 및 효과는 상위 과정에서 더 확인할 수 있다. 여기서는 동일한 문장이라도 Oracle DB 버전에 따라 다른 성능을 보일 수 있다는 것을 확인하자.

DEPARTMENTS 테이블을 먼저 Driving한 실행 계획은 어떨까?

- Oracle 10g DB 실행 계획 (DEPARTMENTS -> EMPLOYEES)

```
-------------------------------------------------------------------------------------------------------------
| Id  | Operation                    | Name             | Starts | E-Rows | A-Rows |   A-Time   | Buffers |
-------------------------------------------------------------------------------------------------------------
|*  1 |  TABLE ACCESS BY INDEX ROWID | EMPLOYEES        |      1 |      1 |     10 |00:00:00.01 |      30 |
|   2 |   NESTED LOOPS               |                  |      1 |     11 |     40 |00:00:00.01 |      26 |
|   3 |    TABLE ACCESS BY INDEX ROWID| DEPARTMENTS      |      1 |     21 |     21 |00:00:00.01 |       4 |
|*  4 |     INDEX RANGE SCAN         | DEPT_LOCATION_IX |      1 |     21 |     21 |00:00:00.01 |       2 |
|*  5 |    INDEX RANGE SCAN          | EMP_DEPARTMENT_IX|     21 |     10 |     18 |00:00:00.01 |      22 |
-------------------------------------------------------------------------------------------------------------
```

Inner Table의 인덱스를 액세스하면서 22개의 I/O가 수행되었고, EMPLOYEES 테이블에서는 4개의 I/O가 수행되었다. 그래서 Inner Table에 접근하면서 사용한 I/O의 합은 26개이다. Outer Table에 접근하면서 사용된 4개의 I/O를 합하면 전체 30개의 I/O를 수행하여 실행된 문장이다. 인덱스 블록의 Pinning 효과를 활용하지 못한 부분이 있기 때문에 후보 행이 많은 경우 반복적인 인덱스의 접근으로 성능이 저하될 수 있다.

- Oracle 11g DB 실행 계획 (DEPARTMENTS -> EMPLOYEES)

```
-------------------------------------------------------------------------------------------------------------
| Id  | Operation                    | Name             | Starts | E-Rows | A-Rows |   A-Time   | Buffers |
-------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |                  |      1 |        |     10 |00:00:00.01 |      11 |
|   1 |  NESTED LOOPS                |                  |      1 |        |     10 |00:00:00.01 |      11 |
|   2 |   NESTED LOOPS               |                  |      1 |      5 |     18 |00:00:00.01 |       9 |
|   3 |    TABLE ACCESS BY INDEX ROWID| DEPARTMENTS      |      1 |      4 |     21 |00:00:00.01 |       4 |
|*  4 |     INDEX RANGE SCAN         | DEPT_LOCATION_IX |      1 |      4 |     21 |00:00:00.01 |       2 |
|*  5 |    INDEX RANGE SCAN          | EMPL_DEPTNO_IX   |     21 |     10 |     18 |00:00:00.01 |       5 |
|*  6 |   TABLE ACCESS BY INDEX ROWID| EMPLOYEES        |     18 |      1 |     10 |00:00:00.01 |       2 |
-------------------------------------------------------------------------------------------------------------
```

Inner Table의 인덱스(EMPL_DEPTNO_IX)를 액세스하면서 5개의 I/O가 수행되었고, EMPLOYEES 테이블에서는 2개의 I/O가 수행되었다. Outer Table에 접근하면서 사용한 4개의 I/O를 합하면, 전체 11개의 I/O가 수행되었다. DEPARTMENTS 테이블을 Driving 했을 때 더 많은 후보 행이 발견되지만 Inner Table의 인덱스의 Pinning 효과로 인해 I/O가 증가하지 않았고, 오히려 최상의 실행 계획이 되었다.

이렇듯 Nested Loops Join은 동일한 결과를 검색하더라도 Oracle 버전, 조인의 순서에 따라 서로 다른 성능을 가질 수 있으므로 현재 환경을 고려하여 필요한 실행 계획을 유도해야 한다.

Nested Loops Join을 사용할 때 조인 순서에 따라 성능 개선 효과를 확인했다. 만약 이러한 Nested Loops Join의 성능을 좀 더 개선하려면?

Nested Loops Join 실행 계획은 Inner Table에 접근할 때 조인 컬럼의 인덱스를 이용하여 액세스되고 추가적인 비 조인 술어가 있다면 테이블에서 필터링하게 된다. 이때 Inner Table의 비 조인 술어의 필터링이 뛰어나다면 Inner Table의 비 조인 술어 컬럼도 인덱스를 이용할 수 있도록 조인 컬럼과 함께 결합 인덱스를 생성하는 것이 좋다. 이때 주의할 사항은 조인 컬럼을 선행 컬럼으로 생성해야 한다.

```
SQL> CREATE INDEX empl_deptno_job_ix ON employees(department_id, job_id) ;

SQL> SELECT /*+ leading(d) use_nl(e) index(d(location_id)) index(e empl_deptno_job_ix) */
            e.employee_id, e.last_name, e.job_id, d.department_id, d.department_name
     FROM employees e, departments d
     WHERE e.department_id = d.department_id
       AND d.location_id   = 1700
       AND e.job_id IN ('FI_ACCOUNT','PU_CLERK');
SQL> @xplan
```

```
-------------------------------------------------------------------------------------------------------------------
| Id  | Operation                     | Name               | Starts | E-Rows | A-Rows |   A-Time   | Buffers |
-------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT              |                    |      1 |        |     10 |00:00:00.01 |      11 |
|   1 |  NESTED LOOPS                 |                    |      1 |        |     10 |00:00:00.01 |      11 |
|   2 |   NESTED LOOPS                |                    |      1 |      5 |     10 |00:00:00.01 |       9 |
|   3 |    TABLE ACCESS BY INDEX ROWID| DEPARTMENTS        |      1 |      4 |     21 |00:00:00.01 |       4 |
|*  4 |     INDEX RANGE SCAN          | DEPT_LOCATION_IX   |      1 |      4 |     21 |00:00:00.01 |       2 |
|   5 |    INLIST ITERATOR            |                    |     21 |        |     10 |00:00:00.01 |       5 |
|*  6 |     INDEX RANGE SCAN          | EMPL_DEPTNO_JOB_IX |     42 |     11 |     10 |00:00:00.01 |       5 |
|   7 |   TABLE ACCESS BY INDEX ROWID | EMPLOYEES          |     10 |      1 |     10 |00:00:00.01 |       2 |
-------------------------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   4 - access("D"."LOCATION_ID"=1700)
   6 - access("E"."DEPARTMENT_ID"="D"."DEPARTMENT_ID" AND (("E"."JOB_ID"='FI_ACCOUNT' OR
              "E"."JOB_ID"='PU_CLERK')))
```

앞선 실행 계획에 비해 성능 개선 효과는 없다. 하지만 EMPL_DEPTNO_JOB_IX 인덱스에서 검색해야 하는 모든 조건을 비교할 수 있으므로 불필요한 테이블의 접근은 제거되었다. (차후 더 많은 데이터를 가진 테이블에서 실습하면 그 효과를 확인할 수 있다.)

```
SQL> DROP INDEX empl_deptno_job_ix ;
```

문제. Hash Join을 사용하여 성능을 최적화 시키시오.

```
SQL> SELECT /*+ leading(c) use_hash(s) */ count(*)
     FROM customers c, sales s
     WHERE c.cust_id = s.cust_id ;
SQL> @xplan
```

```
-------------------------------------------------------------------------------------------------------------------------------
| Id  | Operation              | Name          | Starts | E-Rows | A-Rows |   A-Time   | Buffers |  OMem |  1Mem | Used-Mem |
-------------------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT       |               |      1 |        |      1 |00:00:01.25 |    2084 |       |       |          |
|   1 |  SORT AGGREGATE        |               |      1 |      1 |      1 |00:00:01.25 |    2084 |       |       |          |
|*  2 |   HASH JOIN            |               |      1 |    918K|    918K|00:00:01.08 |    2084 |  1812K|  1381K| 2261K (0)|
|   3 |    INDEX FAST FULL SCAN| CUSTOMERS_PK  |      1 |  55500 |  55500 |00:00:00.01 |     122 |       |       |          |
|   4 |    INDEX FAST FULL SCAN| SALES_CUST_IX |      1 |    918K|    918K|00:00:00.22 |    1962 |       |       |          |
-------------------------------------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   2 - access("C"."CUST_ID"="S"."CUST_ID")
```

```
SQL> SELECT /*+ leading(s) use_hash(c) */ count(*)
     FROM customers c, sales s
     WHERE c.cust_id = s.cust_id ;
SQL> @xplan
```

```
-------------------------------------------------------------------------------------------------------------------------------
| Id  | Operation              | Name          | Starts | E-Rows | A-Rows |   A-Time   | Buffers |  OMem |  1Mem | Used-Mem |
-------------------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT       |               |      1 |        |      1 |00:00:01.30 |    2084 |       |       |          |
|   1 |  SORT AGGREGATE        |               |      1 |      1 |      1 |00:00:01.30 |    2084 |       |       |          |
|*  2 |   HASH JOIN            |               |      1 |    918K|    918K|00:00:01.13 |    2084 |    16M|  2762K|   36M (0)|
|   3 |    INDEX FAST FULL SCAN| SALES_CUST_IX |      1 |    918K|    918K|00:00:00.21 |    1962 |       |       |          |
|   4 |    INDEX FAST FULL SCAN| CUSTOMERS_PK  |      1 |  55500 |  55500 |00:00:00.01 |     122 |       |       |          |
-------------------------------------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   2 - access("C"."CUST_ID"="S"."CUST_ID")
```

Hash Join은 대량의 조인을 수행할 때 사용되므로 가급적 Multi Block I/O를 지원하는 액세스 방법을 사용한다. Looping 작업이 없기 때문에 조인 순서에 따라 I/O의 변화는 없으나 사용된 메모리의 크기가 눈에 띄게 증가된 것을 확인할 수 있다. 데이터가 더 많은 집합을 먼저 액세스하여 Hash Table을 생성하려면 그에 따른 부담이 더 커질 수 있으므로 Hash Join은 후보 행의 개수가 적은 집합을 먼저 액세스하는 것이 좋다.

만약 비 조인 술어가 포함되었다면?

```
SQL> SELECT c.cust_id, c.cust_first_name, c.cust_last_name, s.prod_id, s.amount_sold
     FROM sales s, customers c
     WHERE s.cust_id = c.cust_id
     AND c.country_id = 52790
     AND s.time_id BETWEEN TO_DATE('1999/01/01','YYYY/MM/DD')
                       AND TO_DATE('1999/12/31','YYYY/MM/DD') ;
SQL> @xplan
-------------------------------------------------------------------------------------------------------------------
| Id  | Operation          | Name      | Starts | E-Rows | A-Rows |   A-Time   | Buffers |  OMem |  1Mem | Used-Mem |
-------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |           |      1 |        |    141K|00:00:00.35 |    7309 |       |       |          |
|*  1 |  HASH JOIN         |           |      1 |  95316 |    141K|00:00:00.35 |    7309 |  1323K|  1114K| 1592K (0)|
|*  2 |   TABLE ACCESS FULL| CUSTOMERS |      1 |   2921 |  18520 |00:00:00.01 |    1456 |       |       |          |
|*  3 |   TABLE ACCESS FULL| SALES     |      1 |    230K|    247K|00:00:00.10 |    5853 |       |       |          |
-------------------------------------------------------------------------------------------------------------------
...
```

대량의 데이터를 검색하는 경우 Table Full Scan을 이용한 Hash Join은 올바른 실행 계획일 수 있다. 다만, 인덱스를 활용할 수 있도록 하려면 조인 컬럼이 아닌 비 조인 술어에 사용된 컬럼에 인덱스가 필요하며 가급적 인덱스만 읽고 처리될 수 있도록 결합 인덱스를 활용할 수 있다.

```
SQL> SELECT /*+ leading(c) use_hash(s) index(s(time_id)) index(c(country_id)) */
            c.cust_id, c.cust_first_name, c.cust_last_name, s.prod_id, s.amount_sold
     FROM sales s, customers c
     WHERE s.cust_id = c.cust_id
     AND c.country_id = 52790
     AND s.time_id BETWEEN TO_DATE('1999/01/01','YYYY/MM/DD')
                       AND TO_DATE('1999/12/31','YYYY/MM/DD') ;
SQL> @iostat
-------------------------------------------------------------------------------------------------------------
| Id  | Operation                    | Name            | Starts | E-Rows | A-Rows |   A-Time   | Buffers |
-------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |                 |      1 |        |    141K|00:00:00.48 |   17800 |
|*  1 |  HASH JOIN                   |                 |      1 |  95316 |    141K|00:00:00.48 |   17800 |
|   2 |   TABLE ACCESS BY INDEX ROWID| CUSTOMERS       |      1 |   2921 |  18520 |00:00:00.03 |    1450 |
|*  3 |    INDEX RANGE SCAN          | CUST_COUNTRY_IX |      1 |   2921 |  18520 |00:00:00.01 |      43 |
|   4 |   TABLE ACCESS BY INDEX ROWID| SALES           |      1 |    230K|    247K|00:00:00.23 |   16350 |
|*  5 |    INDEX RANGE SCAN          | SALES_TIME_IX   |      1 |    230K|    247K|00:00:00.08 |    2074 |
-------------------------------------------------------------------------------------------------------------
...
```

비 조인 술어에 만족하는 행이 많기 때문에 각 테이블의 인덱스와 테이블 사이에서의 I/O가 오히려 증가되었다. Hash Join에서도 인덱스를 사용할 수 있지만 지금과 같이 처리 범위가 넓으면 오히려 성능은 저하될 수 있다.

```
SQL> CREATE INDEX cust_ix01  ON customers(country_id, cust_id, cust_first_name, cust_last_name) ;
SQL> CREATE INDEX sales_ix01 ON sales(time_id, cust_id, prod_id, amount_sold) ;
```

결합 인덱스 생성 시 일반 조건식에 사용되는 컬럼을 선행으로 두고, 필요한 모든 컬럼을 포함시킨다. 만약 일부 컬럼이 제외되면 인덱스 액세스 후 테이블에 대한 액세스가 오히려 성능을 저하 시킬 수도 있다. 이러한 결합 인덱스로도 성능 개선이 불가능하면 테이블의 저장 구조를 변경하거나 Materialized View를 이용하는 것도 고려할 수 있다.

```
SQL> SELECT /*+ leading(c) use_hash(s) index(c cust_ix01) index(s sales_ix01) */
            c.cust_id, c.cust_first_name, c.cust_last_name, s.prod_id, s.amount_sold
     FROM sales s, customers c
     WHERE s.cust_id = c.cust_id
     AND c.country_id = 52790
     AND s.time_id BETWEEN TO_DATE('1999/01/01','YYYY/MM/DD')
                       AND TO_DATE('1999/12/31','YYYY/MM/DD') ;
SQL> @xplan
```

Id	Operation	Name	Starts	E-Rows	A-Rows	A-Time	Buffers	OMem	1Mem	Used-Mem
0	SELECT STATEMENT		1		141K	00:00:00.30	2572			
* 1	HASH JOIN		1	95316	141K	00:00:00.30	2572	1323K	1114K	1577K (0)
* 2	INDEX RANGE SCAN	CUST_IX01	1	2921	18520	00:00:00.01	92			
* 3	INDEX RANGE SCAN	SALES_IX01	1	230K	247K	00:00:00.07	2480			

```
Predicate Information (identified by operation id):
---------------------------------------------------
   1 - access("S"."CUST_ID"="C"."CUST_ID")
   2 - access("C"."COUNTRY_ID"=52790)
   3 - access("S"."TIME_ID">=TO_DATE(' 1999-01-01 00:00:00', 'syyyy-mm-dd hh24:mi:ss') AND
              "S"."TIME_ID"<=TO_DATE(' 1999-12-31 00:00:00', 'syyyy-mm-dd hh24:mi:ss'))
```

Hash Join의 튜닝은 반드시 위와 같이 하자는 의미는 아니다. 테이블에 대한 액세스가 부담되면 상대적으로 크기가 작은 인덱스만 액세스하는 방법도 있다는 것을 확인한다.

```
SQL> DROP INDEX cust_ix01 ;
SQL> DROP INDEX sales_ix01 ;
```

› 결론

조인 순서 결정시 Driving Table은 후보 행이 더 적은 집합을 선택한다. 단, 실행 통계를 확인하여 예외 상황도 대비한다. 조인 컬럼에 인덱스가 없으면 Hash Join 또는 Sort Merge Join이 사용되므로 상황에 맞는 실행 계획을 위해 조인 컬럼에 인덱스를 생성한다.

Nested Loops Join 사용

- 소량의 데이터 조인 (First Rows 환경에서 유리)
- Inner Table의 조인 컬럼을 선행으로 하는 인덱스 필요 (경우에 따라 결합 인덱스)

Hash Join 사용

- 대량의 데이터 조인 (All Rows 환경에서 유리)
- 각 집합의 일반 조건 컬럼을 선행으로 하는 결합 인덱스 사용 고려
- 주로 Multi Block I/O 가 사용 됨

Sort Merge Join 사용

- All Rows 환경에서 유리
- 조인 컬럼을 기준으로 정렬된 결과 필요 시 사용
- Non-Equi Join 사용 & 대량의 데이터 조인
- Equi Join 사용 & 소량의 데이터 조인

※ Outer Join 정의

- 조인을 수행하는 두 집합 중 어느 한쪽에만 조인 컬럼의 값을 더 가지고 있는 경우 수행
- INNER JOIN의 결과와 한 쪽 집합에만 더 있는 행 정보를 함께 출력

```
SQL> SELECT /*+ leading(d) use_nl(e) */ d.deptno, d.dname, e.ename, e.sal, e.deptno
     FROM dept d, emp e
     WHERE d.deptno = e.deptno (+) ;
SQL> @xplan
```

```
-------------------------------------------------------------------------------------------------------------
| Id  | Operation                    | Name          | Starts | E-Rows | A-Rows |   A-Time   | Buffers |
-------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |               |      1 |        |     15 |00:00:00.01 |       8 |
|   1 |  NESTED LOOPS OUTER          |               |      1 |     14 |     15 |00:00:00.01 |       8 |
|   2 |   TABLE ACCESS FULL          | DEPT          |      1 |      4 |      4 |00:00:00.01 |       3 |
|   3 |   TABLE ACCESS BY INDEX ROWID| EMP           |      4 |      4 |     14 |00:00:00.01 |       5 |
|*  4 |    INDEX RANGE SCAN          | EMP_DEPTNO_IX |      4 |      5 |     14 |00:00:00.01 |       3 |
-------------------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   4 - access("D"."DEPTNO"="E"."DEPTNO")
```

Hash Join 사용

```
-----------------------------------------------------------------------------------------------------------------------
| Id  | Operation          | Name | Starts | E-Rows | A-Rows |   A-Time   | Buffers |  OMem |  1Mem | Used-Mem |
-----------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |      |      1 |        |     15 |00:00:00.01 |       5 |       |       |          |
|*  1 |  HASH JOIN OUTER   |      |      1 |     14 |     15 |00:00:00.01 |       5 |  888K |  888K |  739K (0)|
|   2 |   TABLE ACCESS FULL| DEPT |      1 |      4 |      4 |00:00:00.01 |       2 |       |       |          |
|   3 |   TABLE ACCESS FULL| EMP  |      1 |     14 |     14 |00:00:00.01 |       3 |       |       |          |
-----------------------------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   1 - access("D"."DEPTNO"="E"."DEPTNO")
```

Sort Merge Join 사용

```
-----------------------------------------------------------------------------------------------------------------------
| Id  | Operation           | Name | Starts | E-Rows | A-Rows |   A-Time   | Buffers |  OMem |  1Mem | Used-Mem |
-----------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT    |      |      1 |        |     15 |00:00:00.01 |       4 |       |       |          |
|   1 |  MERGE JOIN OUTER   |      |      1 |     14 |     15 |00:00:00.01 |       4 |       |       |          |
|   2 |   SORT JOIN         |      |      1 |      4 |      4 |00:00:00.01 |       2 |  2048 |  2048 | 2048  (0)|
|   3 |    TABLE ACCESS FULL| DEPT |      1 |      4 |      4 |00:00:00.01 |       2 |       |       |          |
|*  4 |   SORT JOIN         |      |      4 |     14 |     14 |00:00:00.01 |       2 |  2048 |  2048 | 2048  (0)|
|   5 |    TABLE ACCESS FULL| EMP  |      1 |     14 |     14 |00:00:00.01 |       2 |       |       |          |
-----------------------------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   4 - access("D"."DEPTNO"="E"."DEPTNO")
       filter("D"."DEPTNO"="E"."DEPTNO")
```

※ Outer Join 순서

- OUTER JOIN시 데이터를 더 가지고 있는 쪽이 Driving Table로 선택 됨
- Hash Join 사용 시 SWAP_JOIN_INPUTS 힌트를 사용하면 Hash Table 변경 가능 (10gNF)

```
SQL> SELECT /*+ leading(d) use_nl(e) */ d.department_id, d.department_name, e.last_name, e.salary
     FROM departments d, employees e
     WHERE d.department_id (+) = e.department_id AND d.location_id (+) = 1700 ;
SQL> @iostat
```

```
-------------------------------------------------------------------------------------------------------------
| Id  | Operation                    | Name             | Starts | E-Rows | A-Rows |   A-Time   | Buffers |
-------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |                  |      1 |        |    107 |00:00:00.01 |       7 |
|*  1 |  HASH JOIN OUTER             |                  |      1 |    107 |    107 |00:00:00.01 |       7 |
|   2 |   TABLE ACCESS FULL          | EMPLOYEES        |      1 |    107 |    107 |00:00:00.01 |       3 |
|   3 |   TABLE ACCESS BY INDEX ROWID| DEPARTMENTS      |      1 |      4 |     21 |00:00:00.01 |       4 |
|*  4 |    INDEX RANGE SCAN          | DEPT_LOCATION_IX |      1 |      4 |     21 |00:00:00.01 |       2 |
-------------------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   1 - access("D"."DEPARTMENT_ID"="E"."DEPARTMENT_ID")
   4 - access("D"."LOCATION_ID"=1700)
```

LEADING 힌트를 사용했지만 DEPARTMENTS 테이블이 먼저 액세스 되지 않았다. 필터링 조건이 뛰어난 조건식이 있을 때 DEPARTMENTS 테이블이 먼저 액세스 되면 성능상 유리한 경우도 있지 않을까? Oracle Database 10g 버전부터는 SWAP_JOIN_INPUTS 힌트를 사용하여 Outer Join에서 데이터가 없는 쪽을 Driving Table로 사용 가능하게 한다. 단, Hash Join에서 사용 가능하다.

```
SQL> SELECT /*+ swap_join_inputs(d) use_hash(e) */
            d.department_id, d.department_name, e.last_name, e.salary
     FROM departments d, employees e
     WHERE d.department_id (+) = e.department_id AND d.location_id (+) = 1700 ;
SQL> @iostat
```

```
-------------------------------------------------------------------------------------------------------------
| Id  | Operation                    | Name             | Starts | E-Rows | A-Rows |   A-Time   | Buffers |
-------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |                  |      1 |        |    107 |00:00:00.01 |       7 |
|*  1 |  HASH JOIN RIGHT OUTER       |                  |      1 |    107 |    107 |00:00:00.01 |       7 |
|   2 |   TABLE ACCESS BY INDEX ROWID| DEPARTMENTS      |      1 |      4 |     21 |00:00:00.01 |       2 |
|*  3 |    INDEX RANGE SCAN          | DEPT_LOCATION_IX |      1 |      4 |     21 |00:00:00.01 |       1 |
|   4 |   TABLE ACCESS FULL          | EMPLOYEES        |      1 |    107 |    107 |00:00:00.01 |       5 |
-------------------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   1 - access("D"."DEPARTMENT_ID"="E"."DEPARTMENT_ID")
   3 - access("D"."LOCATION_ID"=1700)
```

※ FULL OUTER JOIN의 성능

- LEFT OUTER JOIN, RIGHT OUTER JOIN의 결과의 합집합 (중복 행 제외)
- Native Full Outer Join을 활용하여 성능 향상 가능 (10gNF, Hash Join만 가능)

• Oracle 10g DB에서의 FULL OUTER JOIN

```
SQL> ALTER SESSION SET optimizer_features_enable = '10.2.0.3' ;
Session altered.
SQL> SELECT e.empno, e.ename, e.sal, d.deptno, d.dname, d.loc
     FROM emp e  FULL OUTER JOIN dept  d
       ON ( e.deptno = d.deptno );
SQL> @xplan
```

Id	Operation	Name	Starts	E-Rows	A-Rows	A-Time	Buffers	OMem	1Mem	Used-Mem
0	SELECT STATEMENT		1		15	00:00:00.01	9			
1	VIEW		1	15	15	00:00:00.01	9			
2	UNION-ALL		1		15	00:00:00.01	9			
* 3	HASH JOIN OUTER		1	14	14	00:00:00.01	5	842K	842K	613K (0)
4	TABLE ACCESS FULL	EMP	1	14	14	00:00:00.01	2			
5	TABLE ACCESS FULL	DEPT	1	4	4	00:00:00.01	3			
6	NESTED LOOPS ANTI		1	1	1	00:00:00.01	4			
7	TABLE ACCESS FULL	DEPT	1	4	4	00:00:00.01	2			
* 8	INDEX RANGE SCAN	EMP_DEPTNO_IX	4	9	3	00:00:00.01	2			

```
Predicate Information (identified by operation id):
---------------------------------------------------
   3 - access("E"."DEPTNO"="D"."DEPTNO")
   8 - access("E"."DEPTNO"="D"."DEPTNO")
```

> 확인 사항

OUTER JOIN은 데이터를 더 가지고 있는 쪽을 먼저 액세스하게 되어 있으나 FULL OUTER JOIN 의 경우 서로 데이터를 더 가지고 있으므로 각각의 집합을 한 번씩 Driving Table로 사용하여 두 번의 조인 결과를 함께 보여준다. 이러한 실행 계획은 동일 테이블의 반복적인 접근으로 성능이 저하될 수 있다.
위의 실행 계획은 다음 문장의 실행 계획과 동일하다.

```
SQL> SELECT *
     FROM (SELECT a.empno, a.ename, b.dname FROM emp a, dept b
           WHERE a.deptno = b.deptno(+)
           UNION ALL
           SELECT null, null, a.dname FROM dept a
           WHERE NOT EXISTS (SELECT 1 FROM emp b
                             WHERE b.deptno = a.deptno)) ;
```

• native_full_outer_join 활용 (from v10.2.0.3)

```
SQL> SELECT /*+ native_full_outer_join */
            e.empno, e.ename, e.sal, d.deptno, d.dname, d.loc
     FROM emp e  FULL OUTER JOIN dept  d
       ON ( e.deptno = d.deptno );
SQL> @xplan
-------------------------------------------------------------------------------------------------------------------------
| Id  | Operation             | Name     | Starts | E-Rows | A-Rows |   A-Time   | Buffers |  OMem |  1Mem | Used-Mem |
-------------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT      |          |      1 |        |     15 |00:00:00.01 |       5 |       |       |          |
|   1 |  VIEW                 | VW_FOJ_0 |      1 |     15 |     15 |00:00:00.01 |       5 |       |       |          |
|*  2 |   HASH JOIN FULL OUTER|          |      1 |     15 |     15 |00:00:00.01 |       5 |   825K|   825K|  725K (0)|
|   3 |    TABLE ACCESS FULL  | DEPT     |      1 |      4 |      4 |00:00:00.01 |       2 |       |       |          |
|   4 |    TABLE ACCESS FULL  | EMP      |      1 |     14 |     14 |00:00:00.01 |       3 |       |       |          |
-------------------------------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   2 - access("E"."DEPTNO"="D"."DEPTNO")
```

Oracle Database 11g부터는 별도의 힌트 없이도 사용 가능하다.

```
SQL> ALTER SESSION SET optimizer_features_enable = '11.2.0.1';

SQL> SELECT e.empno, e.ename, e.sal, d.deptno, d.dname, d.loc
     FROM emp e  FULL OUTER JOIN dept  d
       ON ( e.deptno = d.deptno );
SQL> @xplan
-------------------------------------------------------------------------------------------------------------------------
| Id  | Operation             | Name     | Starts | E-Rows | A-Rows |   A-Time   | Buffers |  OMem |  1Mem | Used-Mem |
-------------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT      |          |      1 |        |     15 |00:00:00.01 |       5 |       |       |          |
|   1 |  VIEW                 | VW_FOJ_0 |      1 |     15 |     15 |00:00:00.01 |       5 |       |       |          |
|*  2 |   HASH JOIN FULL OUTER|          |      1 |     15 |     15 |00:00:00.01 |       5 |   825K|   825K|  706K (0)|
|   3 |    TABLE ACCESS FULL  | DEPT     |      1 |      4 |      4 |00:00:00.01 |       2 |       |       |          |
|   4 |    TABLE ACCESS FULL  | EMP      |      1 |     14 |     14 |00:00:00.01 |       3 |       |       |          |
-------------------------------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   2 - access("E"."DEPTNO"="D"."DEPTNO")
```

Full Outer Join은 반드시 필요한 경우에만 사용하고 가급적 사용을 자제하는 것이 좋다. 꼭 써야 한다면 Native Full Outer Join을 활용하도록 한다. 필요시 히든 파라미터를 이용하여 비활성 가능하다.

```
SQL> ALTER SESSION SET "_optimizer_native_full_outer_join"= off ;
SQL> ALTER SESSION SET "_optimizer_native_full_outer_join"= force ;
```

〉문제 1.

다음 문장의 실행 계획을 확인하고 Nested Loops Join의 성능을 최적화 시키시오.

```
SQL〉 SELECT *
     FROM customers c, sales s
     WHERE c.cust_id    = s.cust_id
       AND c.cust_city  = 'Los Angeles'
       AND c.cust_credit_limit 〉 3000
       AND s.time_id BETWEEN TO_DATE('1999/01/01','YYYY/MM/DD')
                         AND TO_DATE('1999/12/31','YYYY/MM/DD') ;
SQL〉 @xplan
```

```
-------------------------------------------------------------------------------------------------------------
| Id  | Operation                     | Name          | Starts | E-Rows | A-Rows |   A-Time   | Buffers |
-------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT              |               |      1 |        |   1936 |00:00:00.03 |    9232 |
|   1 |  NESTED LOOPS                 |               |      1 |        |   1936 |00:00:00.03 |    9232 |
|   2 |   NESTED LOOPS                |               |      1 |   2596 |   8044 |00:00:00.01 |    1902 |
|*  3 |    TABLE ACCESS BY INDEX ROWID| CUSTOMERS     |      1 |     80 |    629 |00:00:00.01 |     684 |
|*  4 |     INDEX RANGE SCAN          | CUST_CITY_IX  |      1 |     90 |    932 |00:00:00.01 |      25 |
|*  5 |    INDEX RANGE SCAN           | SALES_CUST_IX |    629 |    130 |   8044 |00:00:00.01 |    1218 |
|*  6 |   TABLE ACCESS BY INDEX ROWID | SALES         |   8044 |     33 |   1936 |00:00:00.01 |    7330 |
-------------------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   3 - filter("C"."CUST_CREDIT_LIMIT">3000)
   4 - access("C"."CUST_CITY"='Los Angeles')
   5 - access("C"."CUST_ID"="S"."CUST_ID")
   6 - filter(("S"."TIME_ID"<=TO_DATE(' 1999-12-31 00:00:00', 'syyyy-mm-dd hh24:mi:ss') AND
              "S"."TIME_ID">=TO_DATE(' 1999-01-01 00:00:00', 'syyyy-mm-dd hh24:mi:ss')))
```

〉문제 사항

인덱스에서 충분한 필터링이 진행되지 못하였다. 때문에 테이블에 대한 I/O가 증가되었다.

〉답안. Nested Loops Join의 최적화 방법

1. Looping을 최소화시킬 수 있도록 Driving Table을 선택한다.
2. Inner Table에는 조인 컬럼을 선행 컬럼으로 포함한 결합 인덱스를 준비한다.
3. 최대한 인덱스에서 모든 조건을 비교 가능하게 하여 테이블에 대한 Random Access를 최대한 줄인다.

```
SQL〉 CREATE INDEX sales_x01 ON sales(cust_id, time_id) ;

SQL〉 SELECT /*+ leading(c) use_nl(s) index_rs_asc (s sales_x01) */ *
     FROM customers c, sales s
     WHERE c.cust_id    = s.cust_id
       AND c.cust_city  = 'Los Angeles'
       AND c.cust_credit_limit 〉 3000
       AND s.time_id BETWEEN TO_DATE('1999/01/01','YYYY/MM/DD')
                         AND TO_DATE('1999/12/31','YYYY/MM/DD') ;
SQL〉 @xplan
-------------------------------------------------------------------------------------------------------
| Id  | Operation                     | Name         | Starts | E-Rows | A-Rows |   A-Time   | Buffers |
-------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT              |              |      1 |        |   1936 |00:00:00.01 |    3749 |
|   1 |  NESTED LOOPS                 |              |      1 |        |   1936 |00:00:00.01 |    3749 |
|   2 |   NESTED LOOPS                |              |      1 |   2596 |   1936 |00:00:00.01 |    1896 |
|*  3 |    TABLE ACCESS BY INDEX ROWID| CUSTOMERS    |      1 |     80 |    629 |00:00:00.01 |     684 |
|*  4 |     INDEX RANGE SCAN          | CUST_CITY_IX |      1 |     90 |    932 |00:00:00.01 |      25 |
|*  5 |    INDEX RANGE SCAN           | SALES_X01    |    629 |     33 |   1936 |00:00:00.01 |    1212 |
|   6 |   TABLE ACCESS BY INDEX ROWID | SALES        |   1936 |     33 |   1936 |00:00:00.01 |    1853 |
-------------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   3 - filter("C"."CUST_CREDIT_LIMIT"〉3000)
   4 - access("C"."CUST_CITY"='Los Angeles')
   5 - access("C"."CUST_ID"="S"."CUST_ID" AND "S"."TIME_ID"〉=TO_DATE(' 1999-01-01 00:00:00',
              'syyyy-mm-dd hh24:mi:ss') AND "S"."TIME_ID"〈=TO_DATE(' 1999-12-31 00:00:00', 'syyyy-mm-dd
              hh24:mi:ss'))

SQL〉 DROP INDEX sales_x01 ;
```

〉문제 2.

다음 문장의 실행 계획을 확인하고 성능을 최적화 시키시오.

```
SQL〉 SELECT /*+ leading(c) use_nl(s) */ COUNT(c.cust_email)
     FROM sales s, customers c
     WHERE s.cust_id = c.cust_id
     AND c.country_id = 52790
     AND s.time_id BETWEEN TO_DATE('1999/01/01','YYYY/MM/DD')
                       AND TO_DATE('1999/12/31','YYYY/MM/DD') ;
SQL〉 @xplan
```

```
-------------------------------------------------------------------------------------------------------------
| Id  | Operation                     | Name          | Starts | E-Rows | A-Rows |   A-Time   | Buffers |
-------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT              |               |      1 |        |      1 |00:00:01.49 |     457K|
|   1 |  SORT AGGREGATE               |               |      1 |      1 |      1 |00:00:01.49 |     457K|
|   2 |   NESTED LOOPS                |               |      1 |        |    141K|00:00:01.46 |     457K|
|   3 |    NESTED LOOPS               |               |      1 |  95316 |    526K|00:00:00.43 |   37522 |
|*  4 |     TABLE ACCESS FULL         | CUSTOMERS     |      1 |   2921 |  18520 |00:00:00.01 |    1456 |
|*  5 |     INDEX RANGE SCAN          | SALES_CUST_IX |  18520 |    130 |    526K|00:00:00.19 |   36066 |
|*  6 |    TABLE ACCESS BY INDEX ROWID| SALES         |    526K|     33 |    141K|00:00:00.66 |     419K|
-------------------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   4 - filter("C"."COUNTRY_ID"=52790)
   5 - access("S"."CUST_ID"="C"."CUST_ID")
   6 - filter(("S"."TIME_ID"<=TO_DATE(' 1999-12-31 00:00:00', 'syyyy-mm-dd hh24:mi:ss') AND
              "S"."TIME_ID">=TO_DATE(' 1999-01-01 00:00:00', 'syyyy-mm-dd hh24:mi:ss')))
```

〉문제 사항

- Nested Loops Join을 수행하기에 적절하지 않은 조인 결과가 생성됨 (141K)
- Outer Table의 후보 행이 많기 때문에 Looping 횟수 증가되고 Inner Table의 I/O 증가

> 답안. Hash Join 사용

```
SQL> SELECT /*+ leading(c) use_hash(s) */ COUNT(c.cust_email)
     FROM sales s, customers c
     WHERE s.cust_id = c.cust_id
     AND c.country_id = 52790
     AND s.time_id BETWEEN TO_DATE('1999/01/01','YYYY/MM/DD')
                       AND TO_DATE('1999/12/31','YYYY/MM/DD') ;
SQL> @xplan
```

```
-----------------------------------------------------------------------------------------------------------------------------
| Id  | Operation           | Name      | Starts | E-Rows | A-Rows |   A-Time   | Buffers |  OMem |  1Mem | Used-Mem |
-----------------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT    |           |      1 |        |      1 |00:00:00.35 |    5894 |       |       |          |
|   1 |  SORT AGGREGATE     |           |      1 |      1 |      1 |00:00:00.35 |    5894 |       |       |          |
|*  2 |   HASH JOIN         |           |      1 |  95316 |    141K|00:00:00.32 |    5894 |  1353K|  1082K| 1579K (0)|
|*  3 |    TABLE ACCESS FULL| CUSTOMERS |      1 |   2921 |  18520 |00:00:00.01 |    1456 |       |       |          |
|*  4 |    TABLE ACCESS FULL| SALES     |      1 |    230K|    247K|00:00:00.09 |    4438 |       |       |          |
-----------------------------------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   2 - access("S"."CUST_ID"="C"."CUST_ID")
   3 - filter("C"."COUNTRY_ID"=52790)
   4 - filter(("S"."TIME_ID"<=TO_DATE(' 1999-12-31 00:00:00', 'syyyy-mm-dd hh24:mi:ss') AND
              "S"."TIME_ID">=TO_DATE(' 1999-01-01 00:00:00', 'syyyy-mm-dd hh24:mi:ss')))
```

› 문제 3.

다음 문장의 실행 계획을 확인하고 성능을 최적화 시킨다.

```
SQL> SELECT /*+ leading(c) use_nl(s) */
            c.cust_id, c.cust_last_name, COUNT(s.prod_id)
     FROM customers c, sales s
     WHERE s.cust_id    (+) = c.cust_id
       AND s.channel_id (+) = 9
       AND c.cust_gender = 'F'
       AND c.cust_year_of_birth BETWEEN 1940 AND 1949
     GROUP BY c.cust_id, c.cust_last_name ;
SQL> @xplan
```

```
-------------------------------------------------------------------------------------------------------------------------------------
| Id  | Operation                     | Name          | Starts | E-Rows | A-Rows |   A-Time   | Buffers |  OMem |  1Mem | Used-Mem |
-------------------------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT              |               |      1 |        |   4141 |00:00:00.11 |   66039 |       |       |          |
|   1 |  HASH GROUP BY                |               |      1 |   4141 |   4141 |00:00:00.11 |   66039 |   801K|   801K| 2456K (0)|
|   2 |   NESTED LOOPS OUTER          |               |      1 |    129K|   4179 |00:00:00.11 |   66039 |       |       |          |
|*  3 |    TABLE ACCESS FULL          | CUSTOMERS     |      1 |   3984 |   4141 |00:00:00.01 |    1456 |       |       |          |
|*  4 |    TABLE ACCESS BY INDEX ROWID| SALES         |   4141 |     33 |     59 |00:00:00.10 |   64583 |       |       |          |
|*  5 |     INDEX RANGE SCAN          | SALES_CUST_IX |   4141 |    130 |  69606 |00:00:00.03 |    8414 |       |       |          |
-------------------------------------------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   3 - filter(("C"."CUST_YEAR_OF_BIRTH"<=1949 AND "C"."CUST_YEAR_OF_BIRTH">=1940 AND "C"."CUST_GENDER"='F'))
   4 - filter("S"."CHANNEL_ID"=9)
   5 - access("S"."CUST_ID"="C"."CUST_ID")
```

› 문제 사항

CUSTOMERS 테이블의 결과 중 조건식에 만족하는 행은 4141개이며, SALES 테이블은 조인된 이후 59개의 행이 조건에 만족하고 있다. 하지만 OUTER JOIN을 수행하고 있기 때문에 CUSTOMERS 테이블이 Driving Table로 선택되어 4141 번의 Looping 수행 됨.

```
SQL> SELECT (SELECT COUNT(CASE WHEN cust_gender = 'F' AND cust_year_of_birth BETWEEN 1940 AND 1949
                               THEN 1 END)
             FROM customers) AS custs,
            (SELECT COUNT(CASE WHEN channel_id = 9 THEN 1 END)
             FROM sales)     AS sals
     FROM dual ;
     CUSTS       SALS
---------- ----------
      4141       2074
```

> 답안 1. Hash Join 사용

```
SQL> SELECT /*+ leading(c) use_hash(s) */
            c.cust_id, c.cust_last_name, COUNT(s.prod_id)
      FROM customers c, sales s
      WHERE s.cust_id    (+) = c.cust_id
        AND s.channel_id (+) = 9
        AND c.cust_gender = 'F'
        AND c.cust_year_of_birth BETWEEN 1940 AND 1949
      GROUP BY c.cust_id, c.cust_last_name ;
SQL> @xplan
```

```
-------------------------------------------------------------------------------------------------------------------------
| Id  | Operation           | Name      | Starts | E-Rows | A-Rows |   A-Time   | Buffers |  OMem |  1Mem | Used-Mem |
-------------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT    |           |      1 |        |   4141 |00:00:00.04 |    5894 |       |       |          |
|   1 |  HASH GROUP BY      |           |      1 |   3984 |   4141 |00:00:00.04 |    5894 |   801K|   801K| 1304K (0)|
|*  2 |   HASH JOIN OUTER   |           |      1 |   3984 |   4179 |00:00:00.04 |    5894 |   888K|   888K| 1340K (0)|
|*  3 |    TABLE ACCESS FULL| CUSTOMERS |      1 |   3984 |   4141 |00:00:00.01 |    1456 |       |       |          |
|*  4 |    TABLE ACCESS FULL| SALES     |      1 |   2074 |   2074 |00:00:00.03 |    4438 |       |       |          |
-------------------------------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   2 - access("S"."CUST_ID"="C"."CUST_ID")
   3 - filter(("C"."CUST_YEAR_OF_BIRTH"<=1949 AND "C"."CUST_YEAR_OF_BIRTH">=1940 AND "C"."CUST_GENDER"='F'))
   4 - filter("S"."CHANNEL_ID"=9)
```

Hash Join이 사용되면 인덱스 사용 없이 각각의 조건에 만족하는 행 집합을 생성하여 Join 수행.
단, 필터링이 더 뛰어난 SALES 테이블을 Hash Table로 사용하지는 못하였음.

〉답안 2. Hash Join 사용 (swap_join_inputs)

```
SQL〉 SELECT /*+ swap_join_inputs(s) use_hash(c) */
             c.cust_id, c.cust_last_name, COUNT(s.prod_id)
     FROM customers c, sales s
     WHERE s.cust_id    (+) = c.cust_id
       AND s.channel_id (+) = 9
       AND c.cust_gender = 'F'
       AND c.cust_year_of_birth BETWEEN 1940 AND 1949
     GROUP BY c.cust_id, c.cust_last_name ;
SQL〉 @xplan
```

```
-------------------------------------------------------------------------------------------------------------------------
| Id  | Operation              | Name      | Starts | E-Rows | A-Rows |   A-Time   | Buffers |  OMem |  1Mem | Used-Mem |
-------------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT       |           |      1 |        |   4141 |00:00:00.05 |    5894 |       |       |          |
|   1 |  HASH GROUP BY         |           |      1 |   3984 |   4141 |00:00:00.05 |    5894 |   801K|   801K| 1304K (0)|
|*  2 |   HASH JOIN RIGHT OUTER|           |      1 |   3984 |   4179 |00:00:00.04 |    5894 |   935K|   935K| 1295K (0)|
|*  3 |    TABLE ACCESS FULL   | SALES     |      1 |   2074 |   2074 |00:00:00.03 |    4438 |       |       |          |
|*  4 |    TABLE ACCESS FULL   | CUSTOMERS |      1 |   3984 |   4141 |00:00:00.01 |    1456 |       |       |          |
-------------------------------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   2 - access("S"."CUST_ID"="C"."CUST_ID")
   3 - filter("S"."CHANNEL_ID"=9)
   4 - filter(("C"."CUST_YEAR_OF_BIRTH"<=1949 AND "C"."CUST_YEAR_OF_BIRTH">=1940 AND "C"."CUST_GENDER"='F'))
```

HASH JOIN RIGHT OUTER는 Probe Table을 먼저 액세스 했다는 실행 계획이다. 위의 실습에서 실제 작업량은 앞서 진행했던 결과와 큰 차이는 없으나 추가적인 조건을 통해 SALES 테이블이 더 작은 Hash Table을 생성할 수 있다면 SALES 테이블이 선행 테이블로 액세스 되는 것이 좋을 수 있다.

> 추가 확인

```
SQL> SELECT c.cust_id, c.cust_last_name, COUNT(s.prod_id)
       FROM customers c, sales s
       WHERE s.cust_id    (+) = c.cust_id
        AND s.channel_id  (+) = 9
        AND c.cust_gender = 'F'
        AND c.cust_year_of_birth BETWEEN 1940 AND 1949
      GROUP BY c.cust_id, c.cust_last_name ;
SQL> @xplan
```

```
-------------------------------------------------------------------------------------------------------------------------------
| Id  | Operation              | Name      | Starts | E-Rows | A-Rows |   A-Time   | Buffers |  OMem |  1Mem | Used-Mem |
-------------------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT       |           |      1 |        |   4141 |00:00:00.04 |    5894 |       |       |          |
|   1 |  HASH GROUP BY         |           |      1 |   3984 |   4141 |00:00:00.04 |    5894 |   801K|   801K| 1304K (0)|
|*  2 |   HASH JOIN RIGHT OUTER|           |      1 |   3984 |   4179 |00:00:00.04 |    5894 |   935K|   935K| 1303K (0)|
|*  3 |    TABLE ACCESS FULL   | SALES     |      1 |   2074 |   2074 |00:00:00.03 |    4438 |       |       |          |
|*  4 |    TABLE ACCESS FULL   | CUSTOMERS |      1 |   3984 |   4141 |00:00:00.01 |    1456 |       |       |          |
-------------------------------------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   2 - access("S"."CUST_ID"="C"."CUST_ID")
   3 - filter("S"."CHANNEL_ID"=9)
   4 - filter(("C"."CUST_YEAR_OF_BIRTH"<=1949 AND "C"."CUST_YEAR_OF_BIRTH">=1940 AND "C"."CUST_GENDER"='F'))
```

Oracle Database 11g에서는 별도의 힌트 없이도 Hash Table 의 생성 시 조건에 만족하는 행이 더 적은 쪽을 선행 테이블로 사용 함. 즉, SWAP_JOIN_INPUTS 이 자동으로 사용 됨.

› 문제 4.

다음 문장의 실행 계획을 확인하고 성능을 최적화 시킨다.

```
SQL> SELECT /*+ leading(p) use_nl(s) */
            p.prod_id, p.prod_name, SUM(s.quantity_sold) AS sold_sum
     FROM products p, sales s
     WHERE p.prod_id = s.prod_id (+)
     GROUP BY p.prod_id, p.prod_name ;
SQL> @xplan
```

```
-------------------------------------------------------------------------------------------------------------------------------------------
| Id  | Operation                     | Name          | Starts | E-Rows | A-Rows |   A-Time   | Buffers |  OMem |  1Mem | Used-Mem |
-------------------------------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT              |               |      1 |        |     72 |00:00:01.38 |    9439 |       |       |          |
|   1 |  HASH GROUP BY                |               |      1 |     72 |     72 |00:00:01.38 |    9439 |   729K|   729K| 2806K (0)|
|   2 |   NESTED LOOPS OUTER          |               |      1 |   918K |   918K|00:00:01.13 |    9439 |       |       |          |
|   3 |    TABLE ACCESS FULL          | PRODUCTS      |      1 |     72 |     72 |00:00:00.01 |       3 |       |       |          |
|   4 |    TABLE ACCESS BY INDEX ROWID| SALES         |     72 |  12762 |   918K|00:00:00.73 |    9436 |       |       |          |
|*  5 |     INDEX RANGE SCAN          | SALES_PROD_IX |     72 |  12762 |   918K|00:00:00.26 |    2001 |       |       |          |
-------------------------------------------------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   5 - access("P"."PROD_ID"="S"."PROD_ID")
```

› 문제 사항

- Nested Loops Join은 반복적인 액세스로 인한 I/O가 증가
- 조인 후 그룹을 생성하기 때문에 1:M의 조인이 수행 됨

> 답안 1. Hash Join 이용

```
SQL> SELECT /*+ leading(p) use_hash(s) */
            p.prod_id, p.prod_name, SUM(s.quantity_sold) AS sold_sum
     FROM products p, sales s
     WHERE p.prod_id = s.prod_id (+)
     GROUP BY p.prod_id, p.prod_name ;
SQL> @xplan
```

```
-------------------------------------------------------------------------------------------------------------------------
| Id  | Operation           | Name     | Starts | E-Rows | A-Rows |   A-Time   | Buffers |  OMem |  1Mem | Used-Mem |
-------------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT    |          |      1 |        |     72 |00:00:01.33 |    4441 |       |       |          |
|   1 |  HASH GROUP BY      |          |      1 |     72 |     72 |00:00:01.33 |    4441 |   729K|   729K| 2806K (0)|
|*  2 |   HASH JOIN OUTER   |          |      1 |    918K|    918K|00:00:01.05 |    4441 |   794K|   794K| 1217K (0)|
|   3 |    TABLE ACCESS FULL| PRODUCTS |      1 |     72 |     72 |00:00:00.01 |       3 |       |       |          |
|   4 |    TABLE ACCESS FULL| SALES    |      1 |    918K|    918K|00:00:00.23 |    4438 |       |       |          |
-------------------------------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   2 - access("P"."PROD_ID"="S"."PROD_ID")
```

> 문제 사항

Nested Loops Join의 반복적인 Looping이 제거되면서 I/O는 개선되었지만 1:M Join 작업과 Grouping 작업량이 동일하기 때문에 자원 소모가 크다.

> 답안 2. GROUPING 작업 후 Join 진행

```
SQL> SELECT p.prod_id, p.prod_name, s.sold_sum
     FROM products p, ( SELECT prod_id, SUM(quantity_sold) AS sold_sum
                        FROM sales
                        GROUP BY prod_id ) s
     WHERE p.prod_id = s.prod_id (+) ;
SQL> @xplan
```

```
-------------------------------------------------------------------------------------------------------------------------
| Id  | Operation            | Name     | Starts | E-Rows | A-Rows |   A-Time   | Buffers |  OMem |  1Mem | Used-Mem |
-------------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT     |          |      1 |        |     72 |00:00:00.47 |    4441 |       |       |          |
|*  1 |  HASH JOIN OUTER     |          |      1 |     72 |     72 |00:00:00.47 |    4441 |   794K|   794K| 1209K (0)|
|   2 |   TABLE ACCESS FULL  | PRODUCTS |      1 |     72 |     72 |00:00:00.01 |       3 |       |       |          |
|   3 |   VIEW               |          |      1 |     72 |     72 |00:00:00.47 |    4438 |       |       |          |
|   4 |    HASH GROUP BY     |          |      1 |     72 |     72 |00:00:00.47 |    4438 |   798K|   798K| 1292K (0)|
|   5 |     TABLE ACCESS FULL| SALES    |      1 |    918K|    918K|00:00:00.22 |    4438 |       |       |          |
-------------------------------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   1 - access("P"."PROD_ID"="S"."PROD_ID")
```

SALES 테이블의 Grouping을 통해 조인에 참여할 행의 수를 PRODUCTS 테이블의 수와 동일하게 만들었다.
I/O의 횟수는 동일하지만 Join의 수행 시간은 더 빠르게 진행 가능하다. (1:1 조인 수행됨)

› 답안 3. Index 이용

SALES 테이블의 Full Scan 이 부담 된다면 Index 만 읽고 작업할 수 있도록 한다.

```
SQL> CREATE INDEX sales_x01 ON sales(prod_id, quantity_sold) ;

SQL> SELECT /*+ leading(p) use_hash(s) */
            p.prod_id, p.prod_name, s.sold_sum
     FROM products p, ( SELECT /*+ index_ffs(sales sales_x01) */
                               prod_id, SUM(quantity_sold) AS sold_sum
                        FROM sales
                        WHERE prod_id IS NOT NULL
                        GROUP BY prod_id ) s
     WHERE p.prod_id = s.prod_id (+) ;
SQL> @xplan
-------------------------------------------------------------------------------------------------------------------------------
| Id  | Operation              | Name      | Starts | E-Rows | A-Rows |   A-Time   | Buffers |  OMem |  1Mem | Used-Mem |
-------------------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT       |           |      1 |        |     72 |00:00:00.46 |    2261 |       |       |          |
|*  1 |  HASH JOIN OUTER       |           |      1 |     72 |     72 |00:00:00.46 |    2261 |   794K|   794K| 1211K (0)|
|   2 |   TABLE ACCESS FULL    | PRODUCTS  |      1 |     72 |     72 |00:00:00.01 |       3 |       |       |          |
|   3 |   VIEW                 |           |      1 |     72 |     72 |00:00:00.46 |    2258 |       |       |          |
|   4 |    HASH GROUP BY       |           |      1 |     72 |     72 |00:00:00.46 |    2258 |   798K|   798K| 1291K (0)|
|   5 |     INDEX FAST FULL SCAN| SALES_X01 |      1 |    918K|    918K|00:00:00.21 |    2258 |       |       |          |
-------------------------------------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   1 - access("P"."PROD_ID"="S"."PROD_ID")

SQL> DROP INDEX sales_x01 ;
```

〉문제 5.

다음 문장의 실행 계획을 확인하고 성능을 최적화 시키시오.

```
SQL〉 SELECT DISTINCT c.channel_id, c.channel_desc, s.prod_id, s.promo_id
     FROM channels c, sales s
     WHERE c.channel_id = s.channel_id
       AND s.prod_id BETWEEN 10 AND 20 ;
SQL〉 @xplan
```

```
-------------------------------------------------------------------------------------------------------------------------------------------
| Id  | Operation                     | Name          | Starts | E-Rows | A-Rows |   A-Time   | Buffers |  OMem |  1Mem | Used-Mem |
-------------------------------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT              |               |      1 |        |     53 |00:00:00.13 |     798 |       |       |          |
|   1 |  HASH UNIQUE                  |               |      1 |    142 |     53 |00:00:00.13 |     798 |   808K|   808K| 1285K (0)|
|*  2 |   HASH JOIN                   |               |      1 |  60405 |  61791 |00:00:00.11 |     798 |   878K|   878K|  731K (0)|
|   3 |    TABLE ACCESS FULL          | CHANNELS      |      1 |      5 |      5 |00:00:00.01 |       2 |       |       |          |
|   4 |    TABLE ACCESS BY INDEX ROWID| SALES         |      1 |  60405 |  61791 |00:00:00.05 |     796 |       |       |          |
|*  5 |     INDEX RANGE SCAN          | SALES_PROD_IX |      1 |  60405 |  61791 |00:00:00.02 |     123 |       |       |          |
-------------------------------------------------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   2 - access("C"."CHANNEL_ID"="S"."CHANNEL_ID")
   5 - access("S"."PROD_ID">=10 AND "S"."PROD_ID"<=20)
```

〉문제 사항

JOIN 작업 후 DISTINCT 작업 수행됨

> 답안. DISTINCT 후 Join 수행

```
SQL> SELECT /*+ leading(c) use_hash(s) */
            c.channel_id, c.channel_desc, s.prod_id, s.promo_id
      FROM  channels c, ( SELECT DISTINCT channel_id, prod_id, promo_id
                          FROM sales
                          WHERE prod_id BETWEEN 10 AND 20 ) s
      WHERE c.channel_id = s.channel_id ;
SQL> @xplan
```

```
-------------------------------------------------------------------------------------------------------------------------------------
| Id  | Operation                     | Name          | Starts | E-Rows | A-Rows |   A-Time   | Buffers |  OMem |  1Mem | Used-Mem |
-------------------------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT              |               |      1 |        |     53 |00:00:00.07 |     798 |       |       |          |
|*  1 |  HASH JOIN                    |               |      1 |     40 |     53 |00:00:00.07 |     798 |   878K|   878K|  730K (0)|
|   2 |   TABLE ACCESS FULL           | CHANNELS      |      1 |      5 |      5 |00:00:00.01 |       2 |       |       |          |
|   3 |   VIEW                        |               |      1 |     40 |     53 |00:00:00.07 |     796 |       |       |          |
|   4 |    HASH UNIQUE                |               |      1 |     40 |     53 |00:00:00.07 |     796 |   898K|   898K| 1291K (0)|
|   5 |     TABLE ACCESS BY INDEX ROWID| SALES         |      1 |  60405 |  61791 |00:00:00.05 |     796 |       |       |          |
|*  6 |      INDEX RANGE SCAN         | SALES_PROD_IX |      1 |  60405 |  61791 |00:00:00.02 |     123 |       |       |          |
-------------------------------------------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   1 - access("C"."CHANNEL_ID"="S"."CHANNEL_ID")
   6 - access("PROD_ID">=10 AND "PROD_ID"<=20)
```

두 테이블의 관계는 1:M 관계를 가지고 있으며 DISTINCT를 어디에서 사용하던 동일한 결과를 갖는 문장이다. JOIN 작업의 성능을 위해서라면 JOIN에 참여하는 행의 개수를 줄이는 것이 성능상 유리하다.

› 문제 6.

다음 문장의 실행 계획을 확인하고 성능을 최적화 시키시오.

```
SQL› SELECT c.cust_city, t.calendar_quarter_desc, SUM(s.amount_sold) AS sales_amount
     FROM sales s, times t, customers c
     WHERE s.time_id = t.time_id
       AND s.cust_id = c.cust_id
       AND c.cust_state_province    ='CA'
       AND t.calendar_quarter_desc IN ('1999-01','1999-02')
     GROUP BY c.cust_city, t.calendar_quarter_desc ;
SQL› @xplan
```

Id	Operation	Name	Starts	E-Rows	A-Rows	A-Time	Buffers	OMem	1Mem	Used-Mem
0	SELECT STATEMENT		1		22	00:00:00.93	5948			
1	HASH GROUP BY		1	22	22	00:00:00.93	5948	756K	756K	2437K (0)
*2	HASH JOIN		1	54390	8618	00:00:00.93	5948	862K	862K	1277K (0)
*3	TABLE ACCESS FULL	CUSTOMERS	1	3341	3341	00:00:00.01	1456			
*4	HASH JOIN		1	114K	118K	00:00:00.84	4492	855K	855K	1222K (0)
*5	TABLE ACCESS FULL	TIMES	1	183	181	00:00:00.01	54			
6	TABLE ACCESS FULL	SALES	1	918K	918K	00:00:00.23	4438			

```
Predicate Information (identified by operation id):
---------------------------------------------------
   2 - access("S"."CUST_ID"="C"."CUST_ID")
   3 - filter("C"."CUST_STATE_PROVINCE"='CA')
   4 - access("S"."TIME_ID"="T"."TIME_ID")
   5 - filter(("T"."CALENDAR_QUARTER_DESC"='1999-01' OR "T"."CALENDAR_QUARTER_DESC"='1999-02'))
```

〉답안 1. 조인 종류 및 순서 조절

```
SQL> SELECT COUNT(CASE WHEN c.cust_state_province = 'CA'                     THEN 1 END ) AS custs,
            COUNT(CASE WHEN t.calendar_quarter_desc IN ('1999-01','1999-02') THEN 1 END ) AS times
     FROM sales s, times t, customers c
     WHERE s.time_id = t.time_id
       AND s.cust_id = c.cust_id ;
     CUSTS      TIMES
---------- ----------
     67470     118419
```

조인 방법 : Nested Loops Join (times - sales - customers)

```
SQL> SELECT /*+  leading(t s c) use_nl(s) use_nl(c) */
             c.cust_city, t.calendar_quarter_desc, SUM(s.amount_sold) AS sales_amount
     FROM sales s, times t, customers c
     WHERE s.time_id = t.time_id
       AND s.cust_id = c.cust_id
       AND c.cust_state_province    ='CA'
       AND t.calendar_quarter_desc IN ('1999-01','1999-02')
     GROUP BY c.cust_city, t.calendar_quarter_desc ;
SQL> @xplan
```

```
-------------------------------------------------------------------------------------------------------------------------------------------
| Id  | Operation                      | Name          | Starts | E-Rows | A-Rows |   A-Time   | Buffers |  OMem |  1Mem | Used-Mem |
-------------------------------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT               |               |      1 |        |     22 |00:00:00.62 |     245K|       |       |          |
|   1 |  HASH GROUP BY                 |               |      1 |     22 |     22 |00:00:00.62 |     245K|   756K|   756K| 1288K (0)|
|   2 |   NESTED LOOPS                 |               |      1 |        |   8618 |00:00:00.61 |     245K|       |       |          |
|   3 |    NESTED LOOPS                |               |      1 |   6231 |    118K|00:00:00.44 |     126K|       |       |          |
|   4 |     NESTED LOOPS               |               |      1 |    114K|    118K|00:00:00.16 |    7420 |       |       |          |
|*  5 |      TABLE ACCESS FULL         | TIMES         |      1 |    183 |    181 |00:00:00.01 |      54 |       |       |          |
|   6 |      TABLE ACCESS BY INDEX ROWID| SALES         |    181 |    629 |    118K|00:00:00.11 |    7366 |       |       |          |
|*  7 |       INDEX RANGE SCAN         | SALES_TIME_IX |    181 |    629 |    118K|00:00:00.04 |     683 |       |       |          |
|*  8 |     INDEX UNIQUE SCAN          | CUSTOMERS_PK  |    118K|      1 |    118K|00:00:00.15 |     119K|       |       |          |
|*  9 |    TABLE ACCESS BY INDEX ROWID | CUSTOMERS     |    118K|      1 |   8618 |00:00:00.10 |     118K|       |       |          |
-------------------------------------------------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   5 - filter(("T"."CALENDAR_QUARTER_DESC"='1999-01' OR "T"."CALENDAR_QUARTER_DESC"='1999-02'))
   7 - access("S"."TIME_ID"="T"."TIME_ID")
   8 - access("S"."CUST_ID"="C"."CUST_ID")
   9 - filter("C"."CUST_STATE_PROVINCE"='CA')
```

조인 방법 : Nested Loops Join (customers - sales - times)

```
SQL> SELECT /*+  leading(c s t) use_nl(s) use_nl(t) */
            c.cust_city, t.calendar_quarter_desc, SUM(s.amount_sold) AS sales_amount
     FROM sales s, times t, customers c
     WHERE s.time_id = t.time_id
       AND s.cust_id = c.cust_id
       AND c.cust_state_province    ='CA'
       AND t.calendar_quarter_desc IN ('1999-01','1999-02')
     GROUP BY c.cust_city, t.calendar_quarter_desc ;
SQL> @xplan
```

Id	Operation	Name	Starts	E-Rows	A-Rows	A-Time	Buffers	OMem	1Mem	Used-Mem
0	SELECT STATEMENT		1		22	00:00:00.41	108K			
1	HASH GROUP BY		1	22	22	00:00:00.41	108K	756K	756K	2437K (0)
2	NESTED LOOPS		1		8618	00:00:00.41	108K			
3	NESTED LOOPS		1	54390	67470	00:00:00.30	78922			
4	NESTED LOOPS		1	434K	67470	00:00:00.15	63705			
* 5	TABLE ACCESS FULL	CUSTOMERS	1	3341	3341	00:00:00.01	1456			
6	TABLE ACCESS BY INDEX ROWID	SALES	3341	130	67470	00:00:00.12	62249			
* 7	INDEX RANGE SCAN	SALES_CUST_IX	3341	130	67470	00:00:00.03	6393			
* 8	INDEX UNIQUE SCAN	TIMES_PK	67470	1	67470	00:00:00.07	15217			
* 9	TABLE ACCESS BY INDEX ROWID	TIMES	67470	1	8618	00:00:00.06	29180			

```
Predicate Information (identified by operation id):
---------------------------------------------------
   5 - filter("C"."CUST_STATE_PROVINCE"='CA')
   7 - access("S"."CUST_ID"="C"."CUST_ID")
   8 - access("S"."TIME_ID"="T"."TIME_ID")
   9 - filter(("T"."CALENDAR_QUARTER_DESC"='1999-01' OR "T"."CALENDAR_QUARTER_DESC"='1999-02'))
```

전체 조인된 결과 행은 테이블의 크기에 비해 상대적으로 크지 않다. 때문에 Nested Loops Join을 진행했고, 필터링 조건이 더 뛰어난 CUSTOMERS 테이블을 먼저 액세스하여 조인의 부담을 최소화시켰다. 작업의 시간은 Hash Join보다는 좋아졌지만 Nested Loops Join을 사용하여 I/O는 증가된 것을 확인할 수 있다. 이러한 I/O의 횟수가 튜닝의 목표를 벗어난다면 위의 실행 계획은 최적화된 실행 계획이라고 볼 수 없다.

조인 방법 : Hash Join(Nested Loops Join (customers - sales) - times)

```
SQL> SELECT /*+  leading(c) use_nl(s) use_hash(t) */
            c.cust_city, t.calendar_quarter_desc, SUM(s.amount_sold) AS sales_amount
     FROM sales s, times t, customers c
     WHERE s.time_id = t.time_id
       AND s.cust_id = c.cust_id
       AND c.cust_state_province    ='CA'
       AND t.calendar_quarter_desc IN ('1999-01','1999-02')
     GROUP BY c.cust_city, t.calendar_quarter_desc ;
SQL> @xplan
```

```
-------------------------------------------------------------------------------------------------------------------------------------
| Id  | Operation                      | Name          | Starts | E-Rows | A-Rows |   A-Time   | Buffers |  OMem |  1Mem | Used-Mem |
-------------------------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT               |               |      1 |        |     22 |00:00:00.26 |   63754 |       |       |          |
|   1 |  HASH GROUP BY                 |               |      1 |     22 |     22 |00:00:00.26 |   63754 |   756K|   756K| 2439K (0)|
|*  2 |   HASH JOIN                    |               |      1 |  54390 |   8618 |00:00:00.26 |   63754 |   855K|   855K| 1215K (0)|
|*  3 |    TABLE ACCESS FULL           | TIMES         |      1 |    183 |    181 |00:00:00.01 |      54 |       |       |          |
|   4 |    NESTED LOOPS                |               |      1 |        |  67470 |00:00:00.21 |   63700 |       |       |          |
|   5 |     NESTED LOOPS               |               |      1 |   434K |  67470 |00:00:00.06 |    7844 |       |       |          |
|*  6 |      TABLE ACCESS FULL         | CUSTOMERS     |      1 |   3341 |   3341 |00:00:00.01 |    1456 |       |       |          |
|*  7 |      INDEX RANGE SCAN          | SALES_CUST_IX |   3341 |    130 |  67470 |00:00:00.02 |    6388 |       |       |          |
|   8 |     TABLE ACCESS BY INDEX ROWID| SALES         |  67470 |    130 |  67470 |00:00:00.09 |   55856 |       |       |          |
-------------------------------------------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   2 - access("S"."TIME_ID"="T"."TIME_ID")
   3 - filter(("T"."CALENDAR_QUARTER_DESC"='1999-01' OR "T"."CALENDAR_QUARTER_DESC"='1999-02'))
   6 - filter("C"."CUST_STATE_PROVINCE"='CA')
   7 - access("S"."CUST_ID"="C"."CUST_ID")
```

SALES 테이블에서 CUSTOMERS 테이블과 조인 결과는 많은 양의 데이터가 아니므로 Nested Loops Join으로 수행하고 TIMES 테이블과는 Hash Join으로 처리하여 작업 시간을 단축 시켰다. 단, SALES 테이블의 데이터가 CUST_ID 컬럼을 기준으로 흩어져 있으므로 SALES 테이블에 대한 접근 I/O가 크게 증가되어 있는 모습을 확인할 수 있다.

만약 SALES 테이블이 CUST_ID 컬럼을 기준으로 정렬된 상태의 값을 저장하고 있다면?

```
SQL> DROP TABLE sales2 PURGE ;
SQL> CREATE TABLE sales2
     AS SELECT * FROM sales ORDER BY cust_id ;

SQL> CREATE INDEX sales2_cust_ix ON sales2(cust_id) ;
```

```
SQL> SELECT /*+  leading(c) use_nl(s) use_hash(t) */
            c.cust_city, t.calendar_quarter_desc, SUM(s.amount_sold) AS sales_amount
     FROM sales2 s, times t, customers c
     WHERE s.time_id = t.time_id
       AND s.cust_id = c.cust_id
       AND c.cust_state_province    ='CA'
       AND t.calendar_quarter_desc IN ('1999-01','1999-02')
     GROUP BY c.cust_city, t.calendar_quarter_desc ;
SQL> @xplan
```

Id	Operation	Name	Starts	E-Rows	A-Rows	A-Time	Buffers	OMem	1Mem	Used-Mem
0	SELECT STATEMENT		1		22	00:00:00.23	8920			
1	HASH GROUP BY		1	22	22	00:00:00.23	8920	756K	756K	2567K (0)
* 2	HASH JOIN		1	149K	8618	00:00:00.23	8920	855K	855K	1217K (0)
* 3	TABLE ACCESS FULL	TIMES	1	183	181	00:00:00.01	54			
4	NESTED LOOPS		1		67470	00:00:00.18	8866			
5	NESTED LOOPS		1	830K	67470	00:00:00.06	7844			
* 6	TABLE ACCESS FULL	CUSTOMERS	1	3341	3341	00:00:00.01	1456			
* 7	INDEX RANGE SCAN	SALES2_CUST_IX	3341	6647	67470	00:00:00.03	6388			
8	TABLE ACCESS BY INDEX ROWID	SALES2	67470	249	67470	00:00:00.05	1022			

```
Predicate Information (identified by operation id):
---------------------------------------------------
   2 - access("S"."TIME_ID"="T"."TIME_ID")
   3 - filter(("T"."CALENDAR_QUARTER_DESC"='1999-01' OR "T"."CALENDAR_QUARTER_DESC"='1999-02'))
   6 - filter("C"."CUST_STATE_PROVINCE"='CA')
   7 - access("S"."CUST_ID"="C"."CUST_ID")
```

하나의 데이터 블록에 동일한 CUST_ID가 모여 저장되어 있으면 최소한의 I/O로 필요한 행들을 검색할 수 있다. 다만 특정 컬럼을 기준으로 테이블의 데이터를 매번 재구성을 할 수는 없기 때문에 항상 사용할 수는 없다.

```
SQL> DROP TABLE sales2 PURGE ;
Table dropped.
```

조인 방법 : Hash Join(Nested Loops Join (times - sales) - customers)

```
SQL> SELECT /*+  leading(t) use_nl(s) use_hash(c) */
            c.cust_city, t.calendar_quarter_desc, SUM(s.amount_sold) AS sales_amount
     FROM sales s, times t, customers c
     WHERE s.time_id = t.time_id
       AND s.cust_id = c.cust_id
       AND c.cust_state_province    ='CA'
       AND t.calendar_quarter_desc IN ('1999-01','1999-02')
     GROUP BY c.cust_city, t.calendar_quarter_desc ;
SQL> @xplan
```

```
-------------------------------------------------------------------------------------------------------------------------------------
| Id  | Operation                      | Name          | Starts | E-Rows | A-Rows |   A-Time   | Buffers |  OMem |  1Mem | Used-Mem |
-------------------------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT               |               |      1 |        |     22 |00:00:00.39 |    8871 |       |       |          |
|   1 |  HASH GROUP BY                 |               |      1 |     22 |     22 |00:00:00.39 |    8871 |   756K|   756K| 2433K (0)|
|*  2 |   HASH JOIN                    |               |      1 |  54390 |   8618 |00:00:00.39 |    8871 |   862K|   862K| 1300K (0)|
|*  3 |    TABLE ACCESS FULL           | CUSTOMERS     |      1 |   3341 |   3341 |00:00:00.01 |    1456 |       |       |          |
|   4 |    NESTED LOOPS                |               |      1 |        |   118K|00:00:00.30 |    7415 |       |       |          |
|   5 |     NESTED LOOPS               |               |      1 |   114K|   118K|00:00:00.09 |     732 |       |       |          |
|*  6 |      TABLE ACCESS FULL         | TIMES         |      1 |    183 |    181 |00:00:00.01 |      54 |       |       |          |
|*  7 |      INDEX RANGE SCAN          | SALES_TIME_IX |    181 |    629 |   118K|00:00:00.04 |     678 |       |       |          |
|   8 |     TABLE ACCESS BY INDEX ROWID| SALES         |   118K|    629 |   118K|00:00:00.09 |    6683 |       |       |          |
-------------------------------------------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   2 - access("S"."CUST_ID"="C"."CUST_ID")
   3 - filter("C"."CUST_STATE_PROVINCE"='CA')
   6 - filter(("T"."CALENDAR_QUARTER_DESC"='1999-01' OR "T"."CALENDAR_QUARTER_DESC"='1999-02'))
   7 - access("S"."TIME_ID"="T"."TIME_ID")
```

앞선 실행 계획에 비해 I/O는 감소되었다. 단, 수행 시간은 약간 증가된 것을 확인할 수 있다. 특정 조인 순서, 조인 종류가 항상 조인의 정답이 될 수도 없다. 현재 튜닝의 목표, 기준이 무엇인지 정확하게 정의되어 있어야 하며 데이터의 상태는 항상 변할 수 있으므로 힌트의 남용은 가급적 피하는 것이 좋다. 최적화된 테이블의 구조 및 인덱스의 도움, 옵티마이저 통계 등도 최적화가 되어야 시스템 전체 성능이 향상된다.

위의 문장이 최적의 실행 계획을 의미하지는 않는다. 기준에 부합한다면 사용 가능할 것이다.

〉답안 2. Star Transformation Join 활용

```
SQL〉 SELECT /*+  star_transformation */
             c.cust_city, t.calendar_quarter_desc, SUM(s.amount_sold) AS sales_amount
     FROM sales s, times t, customers c
     WHERE s.time_id = t.time_id
       AND s.cust_id = c.cust_id
       AND c.cust_state_province    ='CA'
       AND t.calendar_quarter_desc IN ('1999-01','1999-02')
     GROUP BY c.cust_city, t.calendar_quarter_desc ;
SQL〉 @iostat
```

```
-------------------------------------------------------------------------------------------------------------------------------------------
| Id  | Operation                              | Name                      | Starts | E-Rows | A-Rows |   A-Time   | Buffers | Reads  | Writes |
-------------------------------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                       |                           |      1 |        |     22 |00:00:00.17 |    9262 |     10 |     10 |
|   1 |  TEMP TABLE TRANSFORMATION             |                           |      1 |        |     22 |00:00:00.17 |    9262 |     10 |     10 |
|   2 |   LOAD AS SELECT                       |                           |      1 |        |      0 |00:00:00.01 |    1469 |      0 |     10 |
|*  3 |    TABLE ACCESS FULL                   | CUSTOMERS                 |      1 |    383 |   3341 |00:00:00.01 |    1456 |      0 |      0 |
|   4 |   HASH GROUP BY                        |                           |      1 |    542 |     22 |00:00:00.16 |    7790 |     10 |      0 |
|*  5 |    HASH JOIN                           |                           |      1 |  48247 |   8618 |00:00:00.16 |    7790 |     10 |      0 |
|   6 |     TABLE ACCESS FULL                  | SYS_TEMP_0FD9D6613_E5DF4  |      1 |   3341 |   3341 |00:00:00.01 |      14 |     10 |      0 |
|*  7 |     HASH JOIN                          |                           |      1 |   5531 |   8618 |00:00:00.14 |    7776 |      0 |      0 |
|*  8 |      TABLE ACCESS FULL                 | TIMES                     |      1 |    183 |    181 |00:00:00.01 |      54 |      0 |      0 |
|   9 |      VIEW                              | VW_ST_62EEF96F            |      1 |   5543 |   8618 |00:00:00.13 |    7722 |      0 |      0 |
|  10 |       NESTED LOOPS                     |                           |      1 |   5543 |   8618 |00:00:00.13 |    7722 |      0 |      0 |
|  11 |        BITMAP CONVERSION TO ROWIDS     |                           |      1 |   5543 |   8618 |00:00:00.11 |    7151 |      0 |      0 |
|  12 |         BITMAP AND                     |                           |      1 |        |      1 |00:00:00.11 |    7151 |      0 |      0 |
|  13 |          BITMAP MERGE                  |                           |      1 |        |      2 |00:00:00.05 |    6399 |      0 |      0 |
|  14 |           BITMAP KEY ITERATION         |                           |      1 |        |    732 |00:00:00.05 |    6399 |      0 |      0 |
|  15 |            TABLE ACCESS FULL           | SYS_TEMP_0FD9D6613_E5DF4  |      1 |   3341 |   3341 |00:00:00.01 |      11 |      0 |      0 |
|  16 |            BITMAP CONVERSION FROM ROWIDS|                          |   3341 |        |    732 |00:00:00.04 |    6388 |      0 |      0 |
|* 17 |             INDEX RANGE SCAN           | SALES_CUST_IX             |   3341 |        |  67470 |00:00:00.02 |    6388 |      0 |      0 |
|  18 |          BITMAP MERGE                  |                           |      1 |        |      2 |00:00:00.06 |     752 |      0 |      0 |
|  19 |           BITMAP KEY ITERATION         |                           |      1 |        |    181 |00:00:00.06 |     752 |      0 |      0 |
|* 20 |            TABLE ACCESS FULL           | TIMES                     |      1 |    183 |    181 |00:00:00.01 |      54 |      0 |      0 |
|  21 |            BITMAP CONVERSION FROM ROWIDS|                          |    181 |        |    181 |00:00:00.06 |     698 |      0 |      0 |
|* 22 |             INDEX RANGE SCAN           | SALES_TIME_IX             |    181 |        |   118K|00:00:00.03 |     698 |      0 |      0 |
|  23 |        TABLE ACCESS BY USER ROWID      | SALES                     |   8618 |      1 |   8618 |00:00:00.01 |     571 |      0 |      0 |
-------------------------------------------------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   3 - filter("C"."CUST_STATE_PROVINCE"='CA')
   5 - access("ITEM_1"="C0")
   7 - access("ITEM_2"="T"."TIME_ID")
   8 - filter(("T"."CALENDAR_QUARTER_DESC"='1999-01' OR "T"."CALENDAR_QUARTER_DESC"='1999-02'))
  17 - access("S"."CUST_ID"="C0")
  20 - filter(("T"."CALENDAR_QUARTER_DESC"='1999-01' OR "T"."CALENDAR_QUARTER_DESC"='1999-02'))
  22 - access("S"."TIME_ID"="T"."TIME_ID")
```

동일 문장의 재실행 이후 @memstat.sql 파일을 실행하면 사용된 메모리 정보를 확인할 수 있다.

Subquery Tuning

다음의 두 문장은 동일한 결과를 검색한다. 그렇다면 실제 업무에서 어떠한 문장을 사용하는가?

```
SQL> SELECT *
     FROM emp
     WHERE deptno IN ( SELECT deptno FROM dept ) ;
     EMPNO ENAME      JOB              MGR HIREDATE         SAL       COMM     DEPTNO
---------- ---------- --------- ---------- --------- ---------- ---------- ----------
      7369 SMITH      CLERK           7902 17-DEC-80        800                    20
...
14 rows selected.

SQL> SELECT e.*
     FROM emp e, dept d
     WHERE e.deptno = d.deptno ;
     EMPNO ENAME      JOB              MGR HIREDATE         SAL       COMM     DEPTNO
---------- ---------- --------- ---------- --------- ---------- ---------- ----------
      7369 SMITH      CLERK           7902 17-DEC-80        800                    20
...
14 rows selected.
```

많은 사용자들이 조인과 서브 쿼리 중 선호하는 문장의 형태가 존재한다. 경우에 따라서는 위의 예제처럼 동일한 결과를 검색하기도 한다. 때문에 특별한 구분 없이 사용하는 경우도 있다. 하지만 두 문장의 형태는 반드시 구분해야 하고, 경우에 따라 어떠한 문장을 사용해야 하는지를 확인한다.

SQL을 처음 공부할 때 서브 쿼리는 메인 쿼리보다 먼저 실행되고, 그 결과를 메인 쿼리에서 사용한다고 배운다. 실제로 그러한가?

```
SQL> SELECT * FROM emp
     WHERE deptno IN ( SELECT 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 |  NESTED LOOPS      |         |      1 |     14 |     14 |00:00:00.01 |       7 |
|   2 |   TABLE ACCESS FULL| EMP     |      1 |     14 |     14 |00:00:00.01 |       3 |
|*  3 |   INDEX UNIQUE SCAN| PK_DEPT |     14 |      1 |     14 |00:00:00.01 |       4 |
-------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   3 - access("DEPTNO"="DEPTNO")
```

사용한 문장은 분명 Subquery이지만 실행 계획은 Join의 실행 계획을 보여준다. Optimizer는 문장의 실행 계획을 생성할 때 Query Transformer를 통해 사용자가 실행한 문장을 변경하여 최적화를 시도한다. 이때 Subquery의 문장은 다음과 같은 Join 문으로 변경이 되므로 실제 사용된 실행 계획은 Join의 실행 계획을 보여줄 수 있다. 즉, Subquery 문장을 실행해도 Oracle DB에서 Join 문으로 변경하여 실행한다.

변경 전		변경 후
SELECT *		SELECT e.*
FROM emp	=>	FROM emp e, dept d
WHERE deptno IN (SELECT deptno FROM dept) ;		WHERE e.deptno = d.deptno ;

그렇다면 어차피 Subquery 문장이 Join 문장으로 바뀌는데 모든 문장을 Join 문으로만 작성해도 괜찮을까?

문제. 근무하는 사원이 존재하는 부서 정보만을 검색하시오.
만약 위와 같은 문제를 해결해야 할 경우 Join과 Subquery 중 어떠한 형식을 사용할 것인가?

```
SQL> SELECT *
     FROM dept
     WHERE deptno IN ( SELECT deptno FROM emp ) ;
    DEPTNO DNAME          LOC
---------- -------------- -------------
        10 ACCOUNTING     NEW YORK
        20 RESEARCH       DALLAS
        30 SALES          CHICAGO

SQL> SELECT d.*
     FROM emp e, dept d
     WHERE e.deptno = d.deptno ;
    DEPTNO DNAME                        LOC
---------- ---------------------------- ----------------------
        10 ACCOUNTING                   NEW YORK
        10 ACCOUNTING                   NEW YORK
        10 ACCOUNTING                   NEW YORK
        20 RESEARCH                     DALLAS
...
14 rows selected.
```

Subquery를 사용한 문장은 제대로 된 결과를 검색했으나 Join 문장은 중복되는 행이 검색된다. Join 문장은 조건식에 만족하는 하나의 행 소스를 생성하므로 항상 M 쪽의 행만큼 검색되게 된다. 중복을 제거하는 DISTINCT를 사용하면 결과를 검색할 수는 있겠지만 불필요한 Join으로 중복 값을 생성하고 다시 그 중복을 제거하는 작업을 수행한다면 성능은 최적화될 수 없다. 즉, 위의 결과는 Subquery 문장을 사용하는 것이 정답이다. 그렇다면 Subquery 문장의 실행 계획은 어떨까?

```
SQL> SELECT *
     FROM dept
     WHERE deptno IN ( SELECT deptno FROM emp ) ;
SQL> @xplan
```

```
-------------------------------------------------------------------------------------------------
| Id  | Operation          | Name          | Starts | E-Rows | A-Rows |   A-Time   | Buffers |
-------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |               |      1 |        |      3 |00:00:00.01 |       5 |
|   1 |  NESTED LOOPS SEMI |               |      1 |      3 |      3 |00:00:00.01 |       5 |
|   2 |   TABLE ACCESS FULL| DEPT          |      1 |      4 |      4 |00:00:00.01 |       3 |
|*  3 |   INDEX RANGE SCAN | EMP_DEPTNO_IX |      4 |      9 |      3 |00:00:00.01 |       2 |
-------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   3 - access("DEPTNO"="DEPTNO")
```

여전히 Join의 실행 계획을 보여준다. 그렇다면 Subquery 문장을 Join으로 수행했다는 의미일까?
아니다. 만약 Join 문으로 변경하고 실행했다면 중복 값이 발견되었을 것이다. 하지만 중복 값이 출력되지 않은 점은 Join 문장이 실행된 것은 아니라는 의미다.

위의 실행 계획은 NESTED LOOPS **SEMI** 조인을 수행한 것으로 표시되고 있다. Semi 조인은 Join Operation은 사용하지만 Join 된 결과를 출력하지는 않고 Join Operation을 조건식의 비교 목적으로 사용하는 Join의 한 종류이다. 하지만 변경된 Join 문을 그대로 출력해도 결과가 같은 상황이면 "SEMI" 키워드는 안 보일 수 있다.

그렇다면 왜 조인일까?
조인 연산은 Optimizer Goal, 인덱스 유/무, 처리될 데이터의 양에 따라 다양한 조인 방법과(Nested Loops, Hash, Sort Merge) 조인의 순서를 변경할 수 있다. 상황에 만족하는 유연성을 제공하고 문장의 성능 최적화에 더 큰 도움이 되기 때문에 Semi Join 역시 필요하다. 때문에 상황에 맞게 Join, Subquery 문장을 작성하고, 그 실행 계획을 확인하여 올바른 Join 또는 Semi Join의 종류를 선택할 수 있어야 한다.

> 결론

- 둘 이상의 행을 출력하는 조건절의 Subquery는 Join Operation으로 변경될 수 있다.
- WHERE 절의 Subquery 문장이 Join의 처리 방식으로 변경되는 것을 "Semi Join"이라 한다.
- Semi Join 은 다양한 조인 방법 및 조인 순서를 사용하여 문장의 최적화에 도움을 줄 수 있다.
- Subuqery 문장을 Semi Join 방식으로 변경할 경우 "UNNEST" 힌트 사용
- Subuqery 문장을 FILTER 방식으로 처리할 경우 "NO_UNNEST" 힌트 사용

*** FILTER 사용**

```
SQL> SELECT *
     FROM dept d
     WHERE EXISTS ( SELECT /*+ no_unnest */ 1
                    FROM emp
                    WHERE deptno = d.deptno ) ;
SQL> @xplan
```

```
-------------------------------------------------------------------------------------------------
| Id  | Operation          | Name          | Starts | E-Rows | A-Rows |   A-Time   | Buffers |
-------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |               |      1 |        |      3 |00:00:00.01 |       7 |
|*  1 |  FILTER            |               |      1 |        |      3 |00:00:00.01 |       7 |
|   2 |   TABLE ACCESS FULL| DEPT          |      1 |      4 |      4 |00:00:00.01 |       3 |
|*  3 |   INDEX RANGE SCAN | EMP_DEPTNO_IX |      4 |      2 |      3 |00:00:00.01 |       4 |
-------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   1 - filter( IS NOT NULL)
   3 - access("DEPTNO"=:B1)
```

```
SQL> SELECT /*+ no_query_transformation */ *
     FROM dept
     WHERE deptno IN ( SELECT deptno FROM emp ) ;
SQL> @xplan
```

```
-------------------------------------------------------------------------------------------------
| Id  | Operation          | Name          | Starts | E-Rows | A-Rows |   A-Time   | Buffers |
-------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |               |      1 |        |      3 |00:00:00.01 |       7 |
|*  1 |  FILTER            |               |      1 |        |      3 |00:00:00.01 |       7 |
|   2 |   TABLE ACCESS FULL| DEPT          |      1 |      4 |      4 |00:00:00.01 |       3 |
|*  3 |   INDEX RANGE SCAN | EMP_DEPTNO_IX |      4 |      2 |      3 |00:00:00.01 |       4 |
-------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   1 - filter( IS NOT NULL)
   3 - access("DEPTNO"=:B1)
```

> 확인 사항

Subquery 영역에 NO_UNNEST 힌트 사용 시 FILTER 연산을 수행한다. 즉, Join 연산을 수행하지 않고 DEPT 테이블의 후보 행을 하나씩 선택하여 EMP_DEPTNO_IX 인덱스에 존재 유무를 확인한다. 때문에 인덱스의 액세스 작업은 4번의 반복 작업이 필요하다. 때문에 NO_UNNEST 힌트 사용 시 Subquery 집합을 반복적으로 접근하며 값의 존재 유무를 확인하므로 후보 행의 개수가 적을 경우 유리하며 후보 행의 개수가 많은 경우에는 Join 연산이 유리할 수 있다.

*** ANTI Join 확인**

```
SQL> SELECT *
     FROM dept d
     WHERE NOT EXISTS ( SELECT *
                        FROM emp
                        WHERE deptno = d.deptno ) ;
    DEPTNO DNAME          LOC
---------- -------------- -------------
        40 OPERATIONS     BOSTON
SQL> @xplan
-------------------------------------------------------------------------------------------------
| Id  | Operation          | Name          | Starts | E-Rows | A-Rows |   A-Time   | Buffers |
-------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |               |      1 |        |      1 |00:00:00.01 |       4 |
|   1 |  NESTED LOOPS ANTI |               |      1 |      1 |      1 |00:00:00.01 |       4 |
|   2 |   TABLE ACCESS FULL| DEPT          |      1 |      4 |      4 |00:00:00.01 |       2 |
|*  3 |   INDEX RANGE SCAN | EMP_DEPTNO_IX |      4 |      9 |      3 |00:00:00.01 |       2 |
-------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   3 - access("DEPTNO"="D"."DEPTNO")

SQL> SELECT *
     FROM dept
     WHERE deptno NOT IN ( SELECT deptno FROM emp ) ;
SQL> @xplan
-----------------------------------------------------------------------------------------------------------------------------------
| Id  | Operation                     | Name    | Starts | E-Rows | A-Rows |   A-Time   | Buffers |  OMem |  1Mem | Used-Mem |
-----------------------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT              |         |      1 |        |      1 |00:00:00.01 |       4 |       |       |          |
|   1 |  MERGE JOIN ANTI NA           |         |      1 |      1 |      1 |00:00:00.01 |       4 |       |       |          |
|   2 |   SORT JOIN                   |         |      1 |      4 |      4 |00:00:00.01 |       0 |  2048 |  2048 | 2048  (0)|
|   3 |    TABLE ACCESS BY INDEX ROWID| DEPT    |      1 |      4 |      4 |00:00:00.01 |       2 |       |       |          |
|   4 |     INDEX FULL SCAN           | PK_DEPT |      1 |      4 |      4 |00:00:00.01 |       1 |       |       |          |
|*  5 |   SORT UNIQUE                 |         |      6 |     14 |      3 |00:00:00.01 |       0 |  2048 |  2048 | 2048  (0)|
|   6 |    TABLE ACCESS FULL          | EMP     |      1 |     14 |     14 |00:00:00.01 |       2 |       |       |          |
-----------------------------------------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   5 - access("DEPTNO"="DEPTNO")
       filter("DEPTNO"="DEPTNO")
```

> 확인 사항

- WHERE 절의 NOT EXISTS, NOT IN 연산자를 사용할 경우 Anti Join으로 실행 가능
- 값의 존재 유무를 Join 연산을 통해서 확인하며 실제 조인된 결과를 출력하는 것은 아님
- 단, NOT IN 연산 사용 시 Oracle Database 11g부터 NULL AWARE ANTI JOIN 사용되고, 이하 버전에서는 NOT NULL이 보장될 때만 Anti Join 가능

*** Query Block 확인**

- Subquery를 사용할 때 QB_NAME 힌트를 이용하여 각각의 Query Block의 식별 및 제어 가능

```
SQL> SELECT /*+ qb_name(main) */ *
     FROM emp
     WHERE sal > ( SELECT /*+ qb_name(sub) */ sal
                   FROM emp
                   WHERE empno = 7566 ) ;
SQL> SELECT *
      FROM table(myxplan.display_cursor(null,null,'IOSTATS LAST +alias')) ;
-------------------------------------------------------------------------------------------------------------
| 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 |      3 |      3 |00:00:00.01 |       6 |
|*  2 |   INDEX RANGE SCAN            | EMP_SAL_IX |      1 |      3 |      3 |00:00:00.01 |       4 |
|   3 |    TABLE ACCESS BY INDEX ROWID| EMP        |      1 |      1 |      1 |00:00:00.01 |       2 |
|*  4 |     INDEX UNIQUE SCAN         | PK_EMP     |      1 |      1 |      1 |00:00:00.01 |       1 |
-------------------------------------------------------------------------------------------------------------
Query Block Name / Object Alias (identified by operation id):
-------------------------------------------------------------
   1 - MAIN / EMP@MAIN
   2 - MAIN / EMP@MAIN
   3 - SUB  / EMP@SUB
   4 - SUB  / EMP@SUB

Predicate Information (identified by operation id):
---------------------------------------------------
   2 - access("SAL">)
   4 - access("EMPNO"=7566)
```

Subquery의 사용이 많아지고 Query Transformation으로 문장이 변화되면 사용자가 원하는 힌트가 제대로 적용되지 않는 경우가 발생할 수 있다. 이러한 Subquery의 각 영역을 보다 명확히 식별하고 사용자 제어의 힌트를 적용하기 위해서 Oracle Database 10g 버전부터 QB_NAME 힌트를 제공한다.

*** Nested Loops Semi Join 사용**

```
SQL> SELECT *
     FROM dept d
     WHERE EXISTS ( SELECT /*+ unnest nl_sj */ *
                    FROM emp
                    WHERE deptno = d.deptno ) ;
SQL> @xplan
```

```
-------------------------------------------------------------------------------------------------
| Id  | Operation          | Name          | Starts | E-Rows | A-Rows |   A-Time   | Buffers |
-------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |               |      1 |        |      3 |00:00:00.01 |       5 |
|   1 |  NESTED LOOPS SEMI |               |      1 |      3 |      3 |00:00:00.01 |       5 |
|   2 |   TABLE ACCESS FULL| DEPT          |      1 |      4 |      4 |00:00:00.01 |       3 |
|*  3 |   INDEX RANGE SCAN | EMP_DEPTNO_IX |      4 |      9 |      3 |00:00:00.01 |       2 |
-------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   3 - access("DEPTNO"="D"."DEPTNO")
```

```
SQL> SELECT *
     FROM dept
     WHERE deptno IN ( SELECT /*+ unnest nl_sj */ deptno
                       FROM emp ) ;
SQL> @xplan
```

```
-------------------------------------------------------------------------------------------------
| Id  | Operation          | Name          | Starts | E-Rows | A-Rows |   A-Time   | Buffers |
-------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |               |      1 |        |      3 |00:00:00.01 |       5 |
|   1 |  NESTED LOOPS SEMI |               |      1 |      3 |      3 |00:00:00.01 |       5 |
|   2 |   TABLE ACCESS FULL| DEPT          |      1 |      4 |      4 |00:00:00.01 |       3 |
|*  3 |   INDEX RANGE SCAN | EMP_DEPTNO_IX |      4 |      9 |      3 |00:00:00.01 |       2 |
-------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   3 - access("DEPTNO"="DEPTNO")
```

> 확인 사항

1의 집합인 DEPT 테이블을 Main Query에서 검색하며 EXISTS, IN 연산자 모두 동일한 실행 계획을 사용하였다. 이때 조인된 결과를 검색하는 것은 아니며 조인 연산을 수행하여 값 비교를 진행하였다.

```
SQL> SELECT *
     FROM emp e
     WHERE EXISTS ( SELECT /*+ unnest nl_sj */ *
                    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 |       5 |
|   1 |  NESTED LOOPS SEMI |         |      1 |     14 |     14 |00:00:00.01 |       5 |
|   2 |   TABLE ACCESS FULL| EMP     |      1 |     14 |     14 |00:00:00.01 |       3 |
|*  3 |   INDEX UNIQUE SCAN| PK_DEPT |      3 |      4 |      3 |00:00:00.01 |       2 |
-------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   3 - access("DEPTNO"="E"."DEPTNO")
```

```
SQL> SELECT *
     FROM emp e
     WHERE deptno IN ( SELECT /*+ unnest nl_sj */ 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 |  NESTED LOOPS      |         |      1 |     14 |     14 |00:00:00.01 |       7 |
|   2 |   TABLE ACCESS FULL| EMP     |      1 |     14 |     14 |00:00:00.01 |       3 |
|*  3 |   INDEX UNIQUE SCAN| PK_DEPT |     14 |      1 |     14 |00:00:00.01 |       4 |
-------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   3 - access("DEPTNO"="DEPTNO")
```

> 확인 사항

M의 집합인 EMP 테이블을 Main Query에서 검색하며 Join 연산을 이용하여 검색되었다. IN 연산자는 Join의 결과를 그대로 검색했으며 EXISTS 연산자는 Semi Join을 수행하였다. 한가지 확인할 사항은 **EXISTS 연산은 Semi Join을 수행하면서 DEPT 인덱스의 반복 접근이 3회밖에 이루어지지 않았다.** 즉, 반복되는 동일 부서 번호에 대한 검색은 최소화되었으므로 성능상 이점이 발생했다. (I/O 횟수가 적음)

이러한 Subquery의 Unnesting 시 조인의 순서는 변경 불가능할까? 앞선 내용을 확인하면 항상 Main Query의 테이블이 Driving 되었고, 경우에 따라서 Main Query의 후보 행 개수가 더 많을 수도 있다. 때문에 순서를 바꿀 수 있다면 보다 나은 실행 계획을 생성할 수 있을 것이다. Subquery의 Unnesting 시 조인의 순서를 변경하려면?

```
SQL> SELECT /*+ qb_name(main) unnest(@sub) leading(e@sub) use_nl(d@main) */ *
     FROM dept d
     WHERE EXISTS ( SELECT /*+ qb_name(sub) */ *
                    FROM emp e
                    WHERE deptno = d.deptno ) ;

SQL> SELECT /*+ qb_name(main) unnest(@sub) leading(e@sub) use_nl(d@main) */ *
     FROM dept d
     WHERE deptno IN ( SELECT /*+ qb_name(sub) */ deptno
                       FROM emp e ) ;
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 |       6 |       |       |          |
|   1 |  NESTED LOOPS                 |               |      1 |        |      3 |00:00:00.01 |       6 |       |       |          |
|   2 |   NESTED LOOPS                |               |      1 |      3 |      3 |00:00:00.01 |       3 |       |       |          |
|   3 |    SORT UNIQUE                |               |      1 |     14 |      3 |00:00:00.01 |       1 |  2048 |  2048 | 2048  (0)|
|   4 |     INDEX FULL SCAN           | EMP_DEPTNO_IX |      1 |     14 |     14 |00:00:00.01 |       1 |       |       |          |
|*  5 |    INDEX UNIQUE SCAN          | PK_DEPT       |      3 |      1 |      3 |00:00:00.01 |       2 |       |       |          |
|   6 |   TABLE ACCESS BY INDEX ROWID | DEPT          |      3 |      1 |      3 |00:00:00.01 |       3 |       |       |          |
-------------------------------------------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   5 - access("DEPTNO"="DEPTNO")
```

> 확인 사항

Subquery가 Unnesting 되면 조인 문장으로 바뀌어 하나의 쿼리 블록으로 변경된다. 쿼리 블록의 이름은 QB_NAME 힌트를 사용하여 지정하고 지정된 쿼리 블록의 이름을 사용하면 조인의 종류 및 순서 등을 결정할 수 있다.

위의 문장은 1의 집합인 DEPT 테이블의 결과를 검색해야 한다. 또한 후보 행의 개수가 적은 Main Query의 DEPT 테이블이 먼저 Driving 되는 것이 맞는 답이겠지만 여기서는 조인의 순서를 변경할 수 있다는 사실과 변경된 조인 순서로 인해 어떠한 실행 계획이 만들어지는지 확인한다.

EMP_DEPTNO_IX 인덱스의 Full Scan 후 SORT UNIQUE 작업이 수행되고 있다. 이는 EMP 테이블의 DEPTNO 컬럼의 중복 값으로 인한 반복적인 Nested Loops Join을 피하기 위해서 진행된 작업이다. Subquery의 Unnesting 시 중복 값이 발생 가능한 경우 Join 연산의 결과를 그대로 보여주면 잘못된 결과를 보여줄 수 있다. 때문에 자동으로 필요에 따라 중복 값을 제거하는 SORT UNIQUE 또는 HASH UNIQUE 연산이 수행되어 중복을 제거하고 조인 연산을 수행한다.

```
SQL> SELECT /*+ qb_name(main) unnest(@sub) leading(d@sub) use_nl(e@main) */ *
     FROM emp e
     WHERE EXISTS ( SELECT /*+ qb_name(sub) */ *
                    FROM dept d
                    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		14	00:00:00.01	6			
1	NESTED LOOPS		1		14	00:00:00.01	6			
2	NESTED LOOPS		1	14	14	00:00:00.01	4			
3	SORT UNIQUE		1	4	4	00:00:00.01	1	2048	2048	2048 (0)
4	INDEX FULL SCAN	PK_DEPT	1	4	4	00:00:00.01	1			
* 5	INDEX RANGE SCAN	EMP_DEPTNO_IX	4	5	14	00:00:00.01	3			
6	TABLE ACCESS BY INDEX ROWID	EMP	14	4	14	00:00:00.01	2			

```
Predicate Information (identified by operation id):
---------------------------------------------------
   5 - access("DEPTNO"="E"."DEPTNO")
```

```
SQL> SELECT /*+ qb_name(main) unnest(@sub) leading(d@sub) use_nl(e@main) */ *
     FROM emp e
     WHERE deptno IN ( SELECT /*+ qb_name(sub) */ deptno FROM dept d ) ;
SQL> @xplan
```

Id	Operation	Name	Starts	E-Rows	A-Rows	A-Time	Buffers
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	14	00:00:00.01	5
3	INDEX FULL SCAN	PK_DEPT	1	4	4	00:00:00.01	2
* 4	INDEX RANGE SCAN	EMP_DEPTNO_IX	4	5	14	00:00:00.01	3
5	TABLE ACCESS BY INDEX ROWID	EMP	14	4	14	00:00:00.01	2

```
Predicate Information (identified by operation id):
---------------------------------------------------
   4 - access("DEPTNO"="DEPTNO")
```

› 확인 사항

Subquery의 테이블이 먼저 Driving 가능하도록 실행 계획이 생성되었다. 단, EXISTS 연산자는 PK_DEPT 인덱스의 결과를 SORT UNIQUE로 중복을 제거하는 작업을 수행하고, IN 연산자를 사용하는 문장은 인덱스의 값을 그냥 사용하고 있다. 기본적으로 EXISTS 연산은 Semi Join을 사용한다. Semi Join은 Join과 유사한 방법으로 실행이 되는 것이지 실제 Join의 결과를 검색하지는 않는다. 때문에 앞선 실행 계획에서도 EXISTS의 반복 횟수가 적었던 것을 확인할 수 있었다. 비록 현재 상황에서는 불필요한 SORT UNIQUE 작업으로 성능이 저하되었다고 할 수도 있지만 상황에 따라서는 필요한 작업일 수도 있다. 그리고 PK_DEPT 인덱스가 Non-Unique 인덱스였다면 IN 연산자의 실행 계획도 중복을 제거하는 작업이 수행된다.

*** Hash Semi Join 사용**

```
SQL> SELECT *
     FROM dept d
     WHERE EXISTS ( SELECT /*+ unnest hash_sj */ *
                    FROM emp
                    WHERE deptno = d.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 |       4 |       |       |          |
|*  1 |  HASH JOIN SEMI    |               |      1 |      3 |      3 |00:00:00.01 |       4 |   825K|   825K|  702K (0)|
|   2 |   TABLE ACCESS FULL| DEPT          |      1 |      4 |      4 |00:00:00.01 |       2 |       |       |          |
|   3 |   INDEX FULL SCAN  | EMP_DEPTNO_IX |      1 |     14 |     14 |00:00:00.01 |       2 |       |       |          |
-------------------------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   1 - access("DEPTNO"="D"."DEPTNO")
```

```
SQL> SELECT *
     FROM emp e
     WHERE EXISTS ( SELECT /*+ unnest hash_sj */ *
                    FROM dept
                    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 |        |     14 |00:00:00.01 |       4 |       |       |          |
|*  1 |  HASH JOIN SEMI    |         |      1 |     14 |     14 |00:00:00.01 |       4 |   762K|   762K|  614K (0)|
|   2 |   TABLE ACCESS FULL| EMP     |      1 |     14 |     14 |00:00:00.01 |       2 |       |       |          |
|   3 |   INDEX FULL SCAN  | PK_DEPT |      1 |      4 |      3 |00:00:00.01 |       2 |       |       |          |
-------------------------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   1 - access("DEPTNO"="E"."DEPTNO")
```

> 확인 사항

HASH_SJ 힌트를 통해 Hash Semi Join을 유도할 수 있다. 위의 예제에서는 EXISTS 연산자만 사용했지만 IN 연산자를 사용하는 상황에서도 가능하다.

*** Hash Join의 순서 변경**

```
SQL> SELECT /*+ qb_name(main) unnest(@sub) leading(e@sub) use_hash(d@main) */ *
     FROM dept d
     WHERE EXISTS ( SELECT /*+ qb_name(sub) */ *
                    FROM emp e
                    WHERE deptno = d.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 |       4 |       |       |          |
|*  1 |  HASH JOIN         |               |      1 |      3 |      3 |00:00:00.01 |       4 |  1066K|  1066K|  606K (0)|
|   2 |   SORT UNIQUE      |               |      1 |     14 |      3 |00:00:00.01 |       1 |  2048 |  2048 | 2048  (0)|
|   3 |    INDEX FULL SCAN | EMP_DEPTNO_IX |      1 |     14 |     14 |00:00:00.01 |       1 |       |       |          |
|   4 |   TABLE ACCESS FULL| DEPT          |      1 |      4 |      4 |00:00:00.01 |       3 |       |       |          |
-----------------------------------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   1 - access("DEPTNO"="D"."DEPTNO")
```

중복 값이 존재하는 EMP 테이블의 인덱스를 사용하므로 중복 제거 후 조인 수행한다. **동일 상황에서 Right Outer Join을 활용하면 다음과 같이 수행할 수도 있다.**

```
SQL> SELECT /*+ qb_name(main) unnest(@sub) swap_join_inputs(e@sub) use_hash(e@sub d@main)  */ *
     FROM dept d
     WHERE EXISTS ( SELECT /*+ qb_name(sub) */ *
                    FROM emp e
                    WHERE deptno = d.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 |       4 |       |       |          |
|*  1 |  HASH JOIN RIGHT SEMI|               |      1 |      3 |      3 |00:00:00.01 |       4 |  1066K|  1066K|  582K (0)|
|   2 |   INDEX FULL SCAN    | EMP_DEPTNO_IX |      1 |     14 |     14 |00:00:00.01 |       1 |       |       |          |
|   3 |   TABLE ACCESS FULL  | DEPT          |      1 |      4 |      4 |00:00:00.01 |       3 |       |       |          |
-----------------------------------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   1 - access("DEPTNO"="D"."DEPTNO")
```

*** Sort Merge Semi Join 사용**

```
SQL> SELECT *
     FROM dept d
     WHERE EXISTS ( SELECT /*+ unnest merge_sj */ *
                    FROM emp
                    WHERE deptno = d.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	MERGE JOIN SEMI		1	3	3	00:00:00.01	5			
2	TABLE ACCESS BY INDEX ROWID	DEPT	1	4	4	00:00:00.01	4			
3	INDEX FULL SCAN	PK_DEPT	1	4	4	00:00:00.01	2			
* 4	SORT UNIQUE		4	14	3	00:00:00.01	1	2048	2048	2048 (0)
5	INDEX FULL SCAN	EMP_DEPTNO_IX	1	14	14	00:00:00.01	1			

```
Predicate Information (identified by operation id):
---------------------------------------------------
   4 - access("DEPTNO"="D"."DEPTNO")
       filter("DEPTNO"="D"."DEPTNO")
```

*** Sort Merge Join 의 순서 변경**

```
SQL> SELECT /*+ qb_name(main) unnest(@sub) leading(e@sub) use_merge(d@main) */ *
     FROM dept d
     WHERE EXISTS ( SELECT /*+ qb_name(sub) */ *
                    FROM emp e
                    WHERE deptno = d.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	MERGE JOIN		1	3	3	00:00:00.01	3			
2	SORT UNIQUE		1	14	3	00:00:00.01	1	2048	2048	2048 (0)
3	INDEX FULL SCAN	EMP_DEPTNO_IX	1	14	14	00:00:00.01	1			
* 4	SORT JOIN		3	4	3	00:00:00.01	2	2048	2048	2048 (0)
5	TABLE ACCESS FULL	DEPT	1	4	4	00:00:00.01	2			

```
Predicate Information (identified by operation id):
---------------------------------------------------
   4 - access("DEPTNO"="D"."DEPTNO")
       filter("DEPTNO"="D"."DEPTNO")
```

*** Nested Loops Anti Join 사용**

```
SQL SELECT *
     FROM dept d
     WHERE NOT EXISTS ( SELECT /*+ unnest nl_aj */ 1
                        FROM emp
                        WHERE deptno = d.deptno ) ;
SQL> @xplan
```

```
-------------------------------------------------------------------------------------------------
| Id  | Operation          | Name          | Starts | E-Rows | A-Rows |   A-Time   | Buffers |
-------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |               |      1 |        |      1 |00:00:00.01 |       4 |
|   1 |  NESTED LOOPS ANTI |               |      1 |      1 |      1 |00:00:00.01 |       4 |
|   2 |   TABLE ACCESS FULL| DEPT          |      1 |      4 |      4 |00:00:00.01 |       2 |
|*  3 |   INDEX RANGE SCAN | EMP_DEPTNO_IX |      4 |      9 |      3 |00:00:00.01 |       2 |
-------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   3 - access("DEPTNO"="D"."DEPTNO")
```

*** Hash Anti Join 사용**

```
SQL> SELECT *
     FROM dept d
     WHERE NOT EXISTS ( SELECT /*+ unnest hash_aj */ 1
                        FROM emp
                        WHERE deptno = d.deptno ) ;
SQL> @xplan
```

```
-------------------------------------------------------------------------------------------------------------------------------
| Id  | Operation          | Name          | Starts | E-Rows | A-Rows |   A-Time   | Buffers |  OMem |  1Mem | Used-Mem |
-------------------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |               |      1 |        |      1 |00:00:00.01 |       3 |       |       |          |
|*  1 |  HASH JOIN ANTI    |               |      1 |      1 |      1 |00:00:00.01 |       3 |   825K|   825K|  710K (0)|
|   2 |   TABLE ACCESS FULL| DEPT          |      1 |      4 |      4 |00:00:00.01 |       2 |       |       |          |
|   3 |   INDEX FULL SCAN  | EMP_DEPTNO_IX |      1 |     14 |     14 |00:00:00.01 |       1 |       |       |          |
-------------------------------------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   1 - access("DEPTNO"="D"."DEPTNO")
```

*** Sort Merge Anti Join 사용**

```
SQL> SELECT *
     FROM dept d
     WHERE NOT EXISTS ( SELECT /*+ unnest merge_aj */ *
                        FROM emp
                        WHERE deptno = d.deptno ) ;
SQL> @xplan
```

Id	Operation	Name	Starts	E-Rows	A-Rows	A-Time	Buffers	OMem	1Mem	Used-Mem
0	SELECT STATEMENT		1		1	00:00:00.01	3			
1	MERGE JOIN ANTI		1	1	1	00:00:00.01	3			
2	TABLE ACCESS BY INDEX ROWID	DEPT	1	4	4	00:00:00.01	2			
3	INDEX FULL SCAN	PK_DEPT	1	4	4	00:00:00.01	1			
* 4	SORT UNIQUE		4	14	3	00:00:00.01	1	2048	2048	2048 (0)
5	INDEX FULL SCAN	EMP_DEPTNO_IX	1	14	14	00:00:00.01	1			

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

   4 - access("DEPTNO"="D"."DEPTNO")
       filter("DEPTNO"="D"."DEPTNO")
```

*** Anti Join 에서 Join 순서 변경**

```
SQL SELECT /*+ qb_name(main) leading(e@sub) use_nl(d@main) */ *
    FROM dept d
    WHERE NOT EXISTS ( SELECT /*+ qb_name(sub) */ *
                       FROM emp e
                       WHERE deptno = d.deptno ) ;
SQL> @xplan
```

```
-------------------------------------------------------------------------------------------------
| Id  | Operation          | Name          | Starts | E-Rows | A-Rows |   A-Time   | Buffers |
-------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |               |      1 |        |      1 |00:00:00.01 |       4 |
|   1 |  NESTED LOOPS ANTI |               |      1 |      1 |      1 |00:00:00.01 |       4 |
|   2 |   TABLE ACCESS FULL| DEPT          |      1 |      4 |      4 |00:00:00.01 |       2 |
|*  3 |   INDEX RANGE SCAN | EMP_DEPTNO_IX |      4 |      9 |      3 |00:00:00.01 |       2 |
-------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   3 - access("DEPTNO"="D"."DEPTNO")
```

> 확인 사항

NOT EXISTS 나 NOT IN 연산은 Subquery 결과에 없는 Main Query 값을 검색하는 문장이다. 즉, 조인 문으로 변경되면 Outer Join의 결과이다. 때문에 값이 존재하지 않는 Subquery를 먼저 Driving 할 수 없다. 때문에 SWAP_JOIN_INPUTS 힌트를 이용하여 Hash Join 으로만 조인 순서를 변경할 수 있다.

```
SQL SELECT *
    FROM dept d
    WHERE NOT EXISTS ( SELECT /*+ unnest hash_aj swap_join_inputs(e) */ *
                       FROM emp e
                       WHERE deptno = d.deptno ) ;
SQL> @xplan
```

```
-----------------------------------------------------------------------------------------------------------------------
| Id  | Operation           | Name          | Starts | E-Rows | A-Rows |   A-Time   | Buffers |  OMem |  1Mem | Used-Mem |
-----------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT    |               |      1 |        |      1 |00:00:00.01 |       3 |       |       |          |
|*  1 |  HASH JOIN RIGHT ANTI|               |      1 |      1 |      1 |00:00:00.01 |       3 |  1066K|  1066K|  622K (0)|
|   2 |   INDEX FULL SCAN   | EMP_DEPTNO_IX |      1 |     14 |     14 |00:00:00.01 |       1 |       |       |          |
|   3 |   TABLE ACCESS FULL | DEPT          |      1 |      4 |      4 |00:00:00.01 |       2 |       |       |          |
-----------------------------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   1 - access("DEPTNO"="D"."DEPTNO")
```

*** NOT IN 연산자와 Subquery**

```
SQL> SELECT *
     FROM departments
     WHERE department_id NOT IN ( SELECT department_id FROM employees ) ;

no rows selected
```

위의 문장은 검색된 결과가 존재하지 않는다. 하지만 다음과 같이 문장을 수정하면?

```
SQL> SELECT *
     FROM departments d
     WHERE NOT EXISTS ( SELECT *
                        FROM employees
                        WHERE department_id = d.department_id ) ;
DEPARTMENT_ID DEPARTMENT_NAME                MANAGER_ID LOCATION_ID
------------- ------------------------------ ---------- -----------
          120 Treasury                                         1700
          130 Corporate Tax                                    1700
          140 Control And Credit                               1700
...
16 rows selected.
```

IN, EXISTS 연산은 앞서 확인 했듯이 동일한 결과를 검색한다. 하지만 NOT IN 과 NOT EXISTS 는 분명 다른 결과를 가져올 수 있다. (동일한 결과를 검색할 수도 있다.) 때문에 NOT IN 연산의 정확한 의미를 파악하고 제대로 된 문장을 작성할 필요가 있다.

```
SQL> SELECT *
      FROM emp
      WHERE deptno IN (10,20, NULL) ;
     EMPNO ENAME      JOB              MGR HIREDATE         SAL       COMM     DEPTNO
---------- ---------- --------- ---------- --------- ---------- ---------- ----------
      7782 CLARK      MANAGER         7839 09-JUN-81       2450                    10
      7839 KING       PRESIDENT            17-NOV-81       5000                    10
      7934 MILLER     CLERK           7782 23-JAN-82       1300                    10
...
8 rows selected.
```

NULL은 IN 연산자의 목록에 포함되어도 결과를 검색할 때 영향을 미치지 않는다. 그 이유는 IN 연산자가 OR 연산을 수행하기 때문이다.

```
WHERE deptno IN (10,20,NULL)       =>      WHERE deptno = 10  OR  deptno = 20  OR  deptno = NULL
```

하지만 문장이 다음과 같이 변경되면?

```
SQL> SELECT *
      FROM emp
      WHERE deptno NOT IN (10,20, NULL) ;

no rows selected
```

NOT IN 연산자는 다음과 같다. 주어진 목록 중 어느 하나라도 DEPTNO 의 값과 동일하면 안되고 모든 조건을 만족해야 한다. 문제는 AND 연산으로 조건식을 비교하기 때문에 NULL 값은 전체 결과에 영향을 미친다.

```
SQL> SELECT *
     FROM emp
     WHERE deptno != 10 AND deptno != 20 AND deptno != NULL ;
```

일반적인 문장 작성 시 NOT IN 연산자의 목록에 NULL 값을 사용자가 직접 입력하는 경우는 없다. 하지만 Subquery를 이용하고 있다면 문장 실행 시 NULL 값은 얼마든지 검색될 수 있으며 전체 문장에 영향을 미치게 된다. 앞선 문장에서 검색된 행이 없는 것 역시 EMPLOYEES 테이블의 DEPARTMENT_ID 컬럼에 존재하는 NULL 때문이다.

```
SQL> SELECT COUNT(*) FROM employees WHERE department_id IS NULL ;
  COUNT(*)
----------
         1
```

때문에 Subquery를 NOT IN 과 함께 사용할 때는 반드시 NULL 값이 리턴되지 않도록 문장이 수정되어야 한다.

```
SQL> SELECT *
     FROM departments
     WHERE department_id NOT IN ( SELECT NVL(department_id,-1)
                                  FROM employees
                                  WHERE department_id IS NOT NULL ) ;
DEPARTMENT_ID DEPARTMENT_NAME                MANAGER_ID LOCATION_ID
------------- ------------------------------ ---------- -----------
          220 NOC                                             1700
          170 Manufacturing                                   1700
          240 Government Sales                                1700
...
16 rows selected.
```

※ NVL 함수나 IS NOT NULL 조건식 중 하나만 사용해도 된다.

*** NOT IN 연산자와 Anti Join (Null Aware Anti Join 확인)**

```
SQL> SELECT deptno, dname, loc
     FROM dept
     WHERE deptno NOT IN (SELECT deptno FROM emp);
SQL> @xplan
```

```
-----------------------------------------------------------------------------------------------------------------------------------
| Id  | Operation                     | Name    | Starts | E-Rows | A-Rows |   A-Time   | Buffers |  OMem |  1Mem | Used-Mem |
-----------------------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT              |         |      1 |        |      1 |00:00:00.01 |       4 |       |       |          |
|   1 |  MERGE JOIN ANTI NA           |         |      1 |      1 |      1 |00:00:00.01 |       4 |       |       |          |
|   2 |   SORT JOIN                   |         |      1 |      4 |      4 |00:00:00.01 |       0 |  2048 |  2048 | 2048  (0)|
|   3 |    TABLE ACCESS BY INDEX ROWID| DEPT    |      1 |      4 |      4 |00:00:00.01 |       2 |       |       |          |
|   4 |     INDEX FULL SCAN           | PK_DEPT |      1 |      4 |      4 |00:00:00.01 |       1 |       |       |          |
|*  5 |   SORT UNIQUE                 |         |      6 |     14 |      3 |00:00:00.01 |       0 |  2048 |  2048 | 2048  (0)|
|   6 |    TABLE ACCESS FULL          | EMP     |      1 |     14 |     14 |00:00:00.01 |       2 |       |       |          |
-----------------------------------------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   5 - access("DEPTNO"="DEPTNO")
       filter("DEPTNO"="DEPTNO")
```

또는 다음의 실행 계획이 사용될 수도 있다. (DEPT.DEPTNO 컬럼에 Primary Key 제약 조건이 존재하기 때문에 현재는 위의 실행 계획이 사용됨)

```
SQL> ALTER TABLE dept DROP PRIMARY KEY ;
SQL> SELECT deptno, dname, loc
     FROM dept
     WHERE deptno NOT IN (SELECT deptno FROM emp);
SQL> @xplan
```

```
-----------------------------------------------------------------------------------------------------------------------------------
| Id  | Operation          | Name | Starts | E-Rows | A-Rows |   A-Time   | Buffers |  OMem |  1Mem | Used-Mem |
-----------------------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |      |      1 |        |      1 |00:00:00.01 |       4 |       |       |          |
|*  1 |  HASH JOIN ANTI NA |      |      1 |      1 |      1 |00:00:00.01 |       4 |   825K|   825K|  702K (0)|
|   2 |   TABLE ACCESS FULL| DEPT |      1 |      4 |      4 |00:00:00.01 |       2 |       |       |          |
|   3 |   TABLE ACCESS FULL| EMP  |      1 |     14 |     14 |00:00:00.01 |       2 |       |       |          |
-----------------------------------------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   1 - access("DEPTNO"="DEPTNO")
```

SQL> ALTER TABLE dept ADD CONSTRAINT pk_dept PRIMARY KEY(deptno) ;

실행 계획에 ANTI NA라고 표시 되는 것은 Oracle Database 11g 부터 지원하는 Null Aware Anti Join을 수행했음을 표시한다. 만약 Oracle Database 10g 버전이라면?

```
SQL> ALTER SESSION SET optimizer_features_enable = '10.2.0.1';

SQL> SELECT deptno, dname, loc FROM dept
     WHERE deptno NOT IN (SELECT deptno FROM emp);
SQL> @xplan
```

```
-------------------------------------------------------------------------------------------
| Id  | Operation          | Name | Starts | E-Rows | A-Rows |   A-Time   | Buffers |
-------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |      |      1 |        |      1 |00:00:00.01 |      10 |
|*  1 |  FILTER            |      |      1 |        |      1 |00:00:00.01 |      10 |
|   2 |   TABLE ACCESS FULL| DEPT |      1 |      4 |      4 |00:00:00.01 |       2 |
|*  3 |   TABLE ACCESS FULL| EMP  |      4 |      2 |      3 |00:00:00.01 |       8 |
-------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   1 - filter( IS NULL)
   3 - filter(LNNVL("DEPTNO"<>:B1))
```

Subquery가 Unnesting 되지 못하고 FILTER를 사용했다. NULL 값의 존재 유무를 찾기 위해 추가 필터링도 추가되었고 인덱스를 사용하지도 못하고 Full Table Scan의 반복 작업이 성능을 저하시킨다. (NULL 값의 유무를 확인하는 이유는 앞서 확인한 NOT IN 연산자와 Subquery 참고)

```
SQL> SELECT deptno, dname, loc
     FROM dept
     WHERE deptno NOT IN (SELECT /*+ unnest */ deptno FROM emp);
SQL> @xplan
```

```
-------------------------------------------------------------------------------------------
| Id  | Operation          | Name | Starts | E-Rows | A-Rows |   A-Time   | Buffers |
-------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |      |      1 |        |      1 |00:00:00.01 |      10 |
|*  1 |  FILTER            |      |      1 |        |      1 |00:00:00.01 |      10 |
|   2 |   TABLE ACCESS FULL| DEPT |      1 |      4 |      4 |00:00:00.01 |       2 |
|*  3 |   TABLE ACCESS FULL| EMP  |      4 |      2 |      3 |00:00:00.01 |       8 |
-------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   1 - filter( IS NULL)
   3 - filter(LNNVL("DEPTNO"<>:B1))
```

힌트 역시 반영되지 않는다. Oracle 10g DB까지는 NOT IN 연산자 사용 시 Subquery의 Unnesting은 Primary Key - Foreign Key 의 관계가 성립되고, 제약 조건이나 조건식을 통해 NULL 값이 Subquery에서 리턴 되지 않을 것이라는 것이 보장되어야만 Anti Join을 수행할 수 있었다.

```
SQL> ALTER TABLE emp ADD CONSTRAINT EMP_FK FOREIGN KEY(deptno) REFERENCES dept(deptno) ;

SQL> SELECT deptno, dname, loc
     FROM dept
     WHERE deptno NOT IN (SELECT /*+ unnest */ NVL(deptno,-1)
                          FROM emp);
SQL> @xplan
-------------------------------------------------------------------------------------------------------------------------------
| Id  | Operation                     | Name    | Starts | E-Rows | A-Rows |   A-Time   | Buffers |  OMem |  1Mem | Used-Mem |
-------------------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT              |         |      1 |        |      1 |00:00:00.01 |       4 |       |       |          |
|   1 |  MERGE JOIN ANTI              |         |      1 |      1 |      1 |00:00:00.01 |       4 |       |       |          |
|   2 |   TABLE ACCESS BY INDEX ROWID| DEPT    |      1 |      4 |      4 |00:00:00.01 |       2 |       |       |          |
|   3 |    INDEX FULL SCAN            | PK_DEPT |      1 |      4 |      4 |00:00:00.01 |       1 |       |       |          |
|*  4 |   SORT UNIQUE                 |         |      4 |     14 |      3 |00:00:00.01 |       2 |  2048 |  2048 | 2048  (0)|
|   5 |    TABLE ACCESS FULL          | EMP     |      1 |     14 |     14 |00:00:00.01 |       2 |       |       |          |
-------------------------------------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   4 - access("DEPTNO"=NVL("DEPTNO",(-1)))
       filter("DEPTNO"=NVL("DEPTNO",(-1)))

SQL> SELECT deptno, dname, loc
     FROM dept
     WHERE deptno NOT IN (SELECT /*+ unnest */ deptno
                          FROM emp
                          WHERE deptno 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 |  NESTED LOOPS ANTI |               |      1 |      1 |      1 |00:00:00.01 |       4 |
|   2 |   TABLE ACCESS FULL| DEPT          |      1 |      4 |      4 |00:00:00.01 |       2 |
|*  3 |   INDEX RANGE SCAN | EMP_DEPTNO_IX |      4 |      9 |      3 |00:00:00.01 |       2 |
-----------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   3 - access("DEPTNO"="DEPTNO")
       filter("DEPTNO" IS NOT NULL)

SQL> ALTER TABLE emp DROP CONSTRAINT EMP_FK ;
```

```
SQL> SELECT deptno, dname, loc FROM dept d
     WHERE NOT EXISTS (SELECT /*+ unnest */ * FROM emp WHERE deptno = d.deptno) ;
SQL> @xplan
```

```
-----------------------------------------------------------------------------------------------
| Id  | Operation          | Name          | Starts | E-Rows | A-Rows |   A-Time   | Buffers |
-----------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |               |      1 |        |      1 |00:00:00.01 |       4 |
|   1 |  NESTED LOOPS ANTI |               |      1 |      1 |      1 |00:00:00.01 |       4 |
|   2 |   TABLE ACCESS FULL| DEPT          |      1 |      4 |      4 |00:00:00.01 |       2 |
|*  3 |   INDEX RANGE SCAN | EMP_DEPTNO_IX |      4 |      9 |      3 |00:00:00.01 |       2 |
-----------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   3 - access("DEPTNO"="D"."DEPTNO")
```

NOT EXISTS 연산자는 NULL의 존재 유무와 상관없기 때문에 Oracle 10g DB에서도 제약 조건이 없어도 Anti Join 수행 가능하며 11g부터는 NOT IN을 사용할 때 Null Aware Anti Join을 사용한다.

```
SQL> ALTER SESSION SET optimizer_features_enable = '11.2.0.1';
SQL> UPDATE emp SET deptno = NULL WHERE empno = 7369 ;
SQL> SELECT deptno, dname, loc
     FROM dept WHERE deptno NOT IN (SELECT deptno FROM emp);
SQL> @iostat
```

```
-----------------------------------------------------------------------------------------------
| Id  | Operation                    | Name    | Starts | E-Rows | A-Rows |   A-Time   | Buffers |
-----------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |         |      1 |        |      0 |00:00:00.01 |       4 |
|   1 |  MERGE JOIN ANTI NA          |         |      1 |      1 |      0 |00:00:00.01 |       4 |
|   2 |   SORT JOIN                  |         |      1 |      4 |      0 |00:00:00.01 |       0 |
|   3 |    TABLE ACCESS BY INDEX ROWID| DEPT    |      1 |      4 |      4 |00:00:00.01 |       2 |
|   4 |     INDEX FULL SCAN          | PK_DEPT |      1 |      4 |      4 |00:00:00.01 |       1 |
|*  5 |   SORT UNIQUE                |         |      1 |     14 |      0 |00:00:00.01 |       0 |
|   6 |    TABLE ACCESS FULL         | EMP     |      1 |     14 |      1 |00:00:00.01 |       2 |
-----------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   5 - access("DEPTNO"="DEPTNO")
       filter("DEPTNO"="DEPTNO")
```

```
SQL> ROLLBACK ;
```

실행 계획 상에서 특이점을 발견했는가? Null Aware Anti Join은 Subquery의 테이블을 검색하는 도중 NULL 값이 발견되면 연산을 중단하고 "no rows selected"를 처리한다. NOT IN 연산을 수행할 때 Subquery에서 NULL이 발견되면 어차피 수행된 결과는 리턴할 수 없다. 때문에 Subquery의 테이블을 계속 액세스할 필요가 없다. 만약 Unnesting이 불가능하면 FILTER 연산을 수행해야 하고, 이는 성능상 더 큰 부담이 될 수 있다. 때문에 Null Aware Anti Join을 이용하여 Unnesting이 가능하게 해준다. **하지만 NOT EXISTS를 사용하면 버전에 상관없이 ANTI JOIN을 이용할 수 있으므로 NOT IN의 결과가 필요한 상황이 아니라면 NOT EXISTS의 사용을 권장한다.**

*** View Merge**

```
SQL> SELECT /*+ leading(d) use_hash(e) */ *
     FROM dept d, (SELECT deptno, COUNT(ename) AS CNT
                   FROM emp GROUP BY deptno) e
     WHERE d.deptno = e.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 |       4 |       |       |          |
|*  1 |  HASH JOIN           |      |      1 |      3 |      3 |00:00:00.01 |       4 |   825K|   825K|  712K (0)|
|   2 |   TABLE ACCESS FULL  | DEPT |      1 |      4 |      4 |00:00:00.01 |       2 |       |       |          |
|   3 |   VIEW               |      |      1 |      3 |      3 |00:00:00.01 |       2 |       |       |          |
|   4 |    HASH GROUP BY     |      |      1 |      3 |      3 |00:00:00.01 |       2 |   888K|   888K|  658K (0)|
|   5 |     TABLE ACCESS FULL| EMP  |      1 |     14 |     14 |00:00:00.01 |       2 |       |       |          |
-------------------------------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   1 - access("D"."DEPTNO"="E"."DEPTNO")
```

VIEW 또는 Inline View를(FROM절의 서브 쿼리) 사용할 때 메인 쿼리와 별도로 해당 쿼리 블록을 실행되면, 실행 계획에는 VIEW Operation을 확인할 수 있다. 하지만 서브 쿼리는 메인 쿼리와 하나의 Query 블록으로 결합되는 실행 계획을 선호한다. 다음과 같이 추가 조건이 존재하면 불필요한 GROUPING을 피할 수도 있고, 상황에 최적화 가능한 조인 종류를 선택할 수 있기 때문이다.

```
SQL> SELECT /*+ leading(d) use_nl(e) */ *
     FROM dept d, (SELECT /*+ merge */ deptno, COUNT(ename) AS CNT
                   FROM emp GROUP BY deptno) e
     WHERE d.deptno = e.deptno AND d.dname = 'ACCOUNTING' ;
SQL> @iostat
```

```
-------------------------------------------------------------------------------------------------------------
| Id  | Operation                     | Name          | Starts | E-Rows | A-Rows |   A-Time   | Buffers |
-------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT              |               |      1 |        |      1 |00:00:00.01 |       4 |
|   1 |  HASH GROUP BY                |               |      1 |      1 |      1 |00:00:00.01 |       4 |
|   2 |   NESTED LOOPS                |               |      1 |        |      3 |00:00:00.01 |       4 |
|   3 |    NESTED LOOPS               |               |      1 |      5 |      3 |00:00:00.01 |       3 |
|   4 |     TABLE ACCESS BY INDEX ROWID| DEPT          |      1 |      1 |      1 |00:00:00.01 |       2 |
|*  5 |      INDEX RANGE SCAN         | DEPT_DNAME_IX |      1 |      1 |      1 |00:00:00.01 |       1 |
|*  6 |     INDEX RANGE SCAN          | EMP_DEPTNO_IX |      1 |      5 |      3 |00:00:00.01 |       1 |
|   7 |    TABLE ACCESS BY INDEX ROWID| EMP           |      3 |      5 |      3 |00:00:00.01 |       1 |
-------------------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   5 - access("D"."DNAME"='ACCOUNTING')
   6 - access("D"."DEPTNO"="DEPTNO")
```

```
SQL> SELECT *
     FROM dept d, (SELECT /*+ no_merge */ deptno, COUNT(ename) AS CNT
                   FROM emp GROUP BY deptno) e
     WHERE d.deptno = e.deptno AND d.dname = 'ACCOUNTING' ;
SQL> @xplan
```

Id	Operation	Name	Starts	E-Rows	A-Rows	A-Time	Buffers
0	SELECT STATEMENT		1		1	00:00:00.01	5
1	NESTED LOOPS		1	1	1	00:00:00.01	5
2	TABLE ACCESS BY INDEX ROWID	DEPT	1	1	1	00:00:00.01	3
* 3	INDEX RANGE SCAN	DEPT_DNAME_IX	1	1	1	00:00:00.01	2
4	VIEW PUSHED PREDICATE		1	1	1	00:00:00.01	2
* 5	FILTER		1		1	00:00:00.01	2
6	SORT AGGREGATE		1	1	1	00:00:00.01	2
7	TABLE ACCESS BY INDEX ROWID	EMP	1	5	3	00:00:00.01	2
* 8	INDEX RANGE SCAN	EMP_DEPTNO_IX	1	5	3	00:00:00.01	1

```
Predicate Information (identified by operation id):
---------------------------------------------------
   3 - access("D"."DNAME"='ACCOUNTING')
   5 - filter(COUNT(*)>0)
   8 - access("DEPTNO"="D"."DEPTNO")
```

Inline View에서 View Merging을 막기 위해 NO_MERGE 힌트를 사용했다. 때문에 실행 계획에서도 VIEW Operation이 확인된다. 한가지 특이한 부분은 "VIEW PUSHED PREDICATE"이다. NO_MERGE 힌트를 통해 Merging을 하지 못했지만 조인 컬럼으로 사용하는 컬럼이 GROUP BY에 사용되기 때문에 모든 DEPTNO 값을 그룹핑 할 필요는 없다. 때문에 DEPT 테이블을 먼저 액세스하여 필요한 후보행 집합을 선택하여 ID: 8에서 확인되듯이 DEPTNO 컬럼의 조건식을 추가하였다. Oracle DB의 버전이 올라가면서 이러한 Query의 변형 작업은 매우 활발하게 진행된다. 문제는 경우에 따라 필요한 변형일 수도 있고 아닐 수도 있기 때문에, 이를 제어할 수 있는 방법을 숙지해야 한다. 또한 성능 최적화에 매우 큰 역할을 수행하는 Query Transformation이 불가능한 상황들을 확인한다.

› 문제 1.

다음 문장의 실행 계획을 확인하고 성능을 최적화 시키시오.

```
SQL> SELECT cust_last_name, cust_postal_code
     FROM customers c
     WHERE EXISTS (SELECT 1
                   FROM sales
                   WHERE cust_id = c.cust_id)
       AND cust_year_of_birth = 1980 ;
SQL> @xplan
```

```
-------------------------------------------------------------------------------------------------------------------------------------
| Id  | Operation             | Name          | Starts | E-Rows | A-Rows |   A-Time   | Buffers |  OMem |  1Mem | Used-Mem |
-------------------------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT      |               |      1 |        |    129 |00:00:00.81 |    3420 |       |       |          |
|*  1 |  HASH JOIN SEMI       |               |      1 |    715 |    129 |00:00:00.81 |    3420 |   842K|   842K| 1211K (0)|
|*  2 |   TABLE ACCESS FULL   | CUSTOMERS     |      1 |    740 |    891 |00:00:00.01 |    1456 |       |       |          |
|   3 |   INDEX FAST FULL SCAN| SALES_CUST_IX |      1 |    918K|    918K|00:00:00.22 |    1964 |       |       |          |
-------------------------------------------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   1 - access("CUST_ID"="C"."CUST_ID")
   2 - filter("CUST_YEAR_OF_BIRTH"=1980)
```

› 문제 사항

상대적으로 CUSTOMERS 테이블의 후보행이 많지 않은 상황에서 Hash Join을 이용하였기 때문에 SALES_CUST_IX 인덱스를 FULL SCAN 하였다.

> 답안 1. FILTER 사용

```
SQL> SELECT /*+ index (c(cust_year_of_birth)) */
            cust_last_name, cust_postal_code
     FROM customers c
     WHERE EXISTS (SELECT /*+ no_unnest */ 1
                   FROM sales
                   WHERE cust_id = c.cust_id)
       AND cust_year_of_birth = 1980 ;
SQL> @xplan
```

Id	Operation	Name	Starts	E-Rows	A-Rows	A-Time	Buffers
0	SELECT STATEMENT		1		129	00:00:00.01	3402
*1	FILTER		1		129	00:00:00.01	3402
2	TABLE ACCESS BY INDEX ROWID	CUSTOMERS	1	740	891	00:00:00.01	770
*3	INDEX RANGE SCAN	CUST_YOB_IX	1	740	891	00:00:00.01	6
*4	INDEX RANGE SCAN	SALES_CUST_IX	891	2	129	00:00:00.01	2632

```
Predicate Information (identified by operation id):
---------------------------------------------------
   1 - filter( IS NOT NULL)
   3 - access("CUST_YEAR_OF_BIRTH"=1980)
   4 - access("CUST_ID"=:B1)
```

> 답안 2. Nested Loops Semi Join 사용

```
SQL> SELECT /*+ index (c(cust_year_of_birth)) */
            cust_last_name, cust_postal_code
     FROM customers c
     WHERE EXISTS (SELECT /*+ unnest nl_sj */ 1
                   FROM sales
                   WHERE cust_id = c.cust_id)
       AND cust_year_of_birth = 1980 ;
SQL> @xplan
```

Id	Operation	Name	Starts	E-Rows	A-Rows	A-Time	Buffers
0	SELECT STATEMENT		1		129	00:00:00.01	2515
1	NESTED LOOPS SEMI		1	715	129	00:00:00.01	2515
2	TABLE ACCESS BY INDEX ROWID	CUSTOMERS	1	740	891	00:00:00.01	770
*3	INDEX RANGE SCAN	CUST_YOB_IX	1	740	891	00:00:00.01	6
*4	INDEX RANGE SCAN	SALES_CUST_IX	891	888K	129	00:00:00.01	1745

```
Predicate Information (identified by operation id):
---------------------------------------------------
   3 - access("CUST_YEAR_OF_BIRTH"=1980)
   4 - access("CUST_ID"="C"."CUST_ID")
```

〉확인 사항 : Filter vs Nested Loops Join 비교

```
SQL〉 CREATE TABLE t_emp AS
     SELECT *
     FROM emp, (SELECT rownum AS no
                FROM dual
                CONNECT BY level <= 100) ;

SQL〉 SELECT *
     FROM t_emp t
     WHERE EXISTS ( SELECT /*+ no_unnest */ *
                    FROM dept
                    WHERE deptno = t.deptno
                      AND loc IS NOT NULL ) ;
SQL〉 @xplan
```

Id	Operation	Name	Starts	E-Rows	A-Rows	A-Time	Buffers
0	SELECT STATEMENT		1		1400	00:00:00.01	32
*1	FILTER		1		1400	00:00:00.01	32
2	TABLE ACCESS FULL	T_EMP	1	1400	1400	00:00:00.01	26
*3	TABLE ACCESS BY INDEX ROWID	DEPT	3	1	3	00:00:00.01	6
*4	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 - filter("LOC" IS NOT NULL)
   4 - access("DEPTNO"=:B1)
```

FILTER 연산을 수행하더라도 후보행의 중복이 많으면 실제 후보행의 개수만큼 Subquery의 접근이 발생하지는 않는다. 위의 결과를 확인하더라도 Subquery의 수행 횟수는 3회이다. 즉, Main Query의 동일한 입력값에 대해 결과를 캐싱 하여 사용하면서 Main Query의 중복 값이 많은 경우 보다 나은 실행 계획을 생성할 수 있게 된다.

```
SQL> SELECT *
     FROM t_emp t
     WHERE EXISTS ( SELECT /*+ unnest nl_sj */ *
                    FROM dept
                    WHERE deptno = t.deptno
                      AND loc IS NOT NULL ) ;
SQL> @xplan
```

```
-------------------------------------------------------------------------------------------------
| Id  | Operation                    | Name    | Starts | E-Rows | A-Rows |   A-Time   | Buffers |
-------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |         |      1 |        |   1400 |00:00:00.01 |      31 |
|   1 |  NESTED LOOPS SEMI           |         |      1 |   1400 |   1400 |00:00:00.01 |      31 |
|   2 |   TABLE ACCESS FULL          | T_EMP   |      1 |   1400 |   1400 |00:00:00.01 |      26 |
|*  3 |   TABLE ACCESS BY INDEX ROWID| DEPT    |      3 |      4 |      3 |00:00:00.01 |       5 |
|*  4 |    INDEX UNIQUE SCAN         | PK_DEPT |      3 |      1 |      3 |00:00:00.01 |       2 |
-------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   3 - filter("LOC" IS NOT NULL)
   4 - access("DEPTNO"="T"."DEPTNO")
```

Oracle Database 10g 부터는 Nested Loops Semi Join 에서도 동일한 Input 에 결과를 캐싱 하므로 실제 Subquery 의 접근 횟수가 3 회이다. 만약 Oracle Database 9i 버전에서 동일 실습을 진행한다면 Nested Loops Semi Join 보다 Filter 연산이 보다 나은 실행 계획이 될 수도 있다.

```
SQL> DROP TABLE t_emp PURGE ;
```

〉문제 2.

다음 문장의 실행 계획을 확인하고 성능을 최적화 시키시오.

```
SQL〉 SELECT /*+ no_query_transformation */
            prod_id, cust_id, channel_id, amount_sold
     FROM sales s
     WHERE EXISTS (SELECT 1
                   FROM customers
                   WHERE cust_id = s.cust_id
                     AND cust_year_of_birth = 1980)
       AND channel_id 〉 2 ;
SQL〉 @xplan
```

```
-------------------------------------------------------------------------------------------------------------
¦ Id  ¦ Operation                    ¦ Name         ¦ Starts ¦ E-Rows ¦ A-Rows ¦   A-Time   ¦ Buffers ¦
-------------------------------------------------------------------------------------------------------------
¦   0 ¦ SELECT STATEMENT             ¦              ¦      1 ¦        ¦  10037 ¦00:00:02.16 ¦    1322K¦
¦*  1 ¦  FILTER                      ¦              ¦      1 ¦        ¦  10037 ¦00:00:02.16 ¦    1322K¦
¦*  2 ¦   TABLE ACCESS FULL          ¦ SALES        ¦      1 ¦    689K¦    660K¦00:00:00.22 ¦    4538 ¦
¦*  3 ¦   TABLE ACCESS BY INDEX ROWID¦ CUSTOMERS    ¦    439K¦      1 ¦   6325 ¦00:00:01.41 ¦    1318K¦
¦*  4 ¦    INDEX UNIQUE SCAN         ¦ CUSTOMERS_PK ¦    439K¦      1 ¦    439K¦00:00:00.72 ¦     878K¦
-------------------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   1 - filter( IS NOT NULL)
   2 - filter("CHANNEL_ID">2)
   3 - filter("CUST_YEAR_OF_BIRTH"=1980)
   4 - access("CUST_ID"=:B1)
```

〉문제 사항

- 660,000개의 후보행의 CUST_ID 값의 존재 유무를 확인하기 위해 CUSTOMERS 테이블의 반복적인 접근 발생
- 반복적인 CUSTOMERS 테이블에 접근으로 인해 I/O 증가, 속도 저하
- Query Transformation을 방해함으로 성능 저하 발생

> 답안 1. Semi Join 사용

```
SQL> SELECT prod_id, cust_id, channel_id, amount_sold
     FROM sales s
     WHERE EXISTS (SELECT /*+ unnest */ 1
                   FROM customers
                   WHERE cust_id = s.cust_id
                     AND cust_year_of_birth = 1980)
       AND channel_id > 2 ;
SQL> @xplan
```

```
-----------------------------------------------------------------------------------------------------------------------------------
| Id  | Operation               | Name             | Starts | E-Rows | A-Rows |   A-Time   | Buffers |  OMem |  1Mem | Used-Mem |
-----------------------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT        |                  |      1 |        |  10037 |00:00:00.69 |    4664 |       |       |          |
|*  1 |  HASH JOIN RIGHT SEMI   |                  |      1 |  72242 |  10037 |00:00:00.69 |    4664 |  1036K|  1036K| 1226K (0)|
|*  2 |   VIEW                  | index$_join$_002 |      1 |    740 |    891 |00:00:00.05 |     126 |       |       |          |
|*  3 |    HASH JOIN            |                  |      1 |        |    891 |00:00:00.05 |     126 |   862K|   862K| 1224K (0)|
|*  4 |     INDEX RANGE SCAN    | CUST_YOB_IX      |      1 |    740 |    891 |00:00:00.01 |       4 |       |       |          |
|   5 |     INDEX FAST FULL SCAN| CUSTOMERS_PK     |      1 |    740 |  55500 |00:00:00.01 |     122 |       |       |          |
|*  6 |   TABLE ACCESS FULL     | SALES            |      1 |    689K|    660K|00:00:00.18 |    4538 |       |       |          |
-----------------------------------------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   1 - access("CUST_ID"="S"."CUST_ID")
   2 - filter("CUST_YEAR_OF_BIRTH"=1980)
   3 - access(ROWID=ROWID)
   4 - access("CUST_YEAR_OF_BIRTH"=1980)
   6 - filter("CHANNEL_ID">2)
```

```
SQL> SELECT prod_id, cust_id, channel_id, amount_sold
     FROM sales s
     WHERE EXISTS (SELECT /*+ unnest hash_sj no_index(customers(cust_id)) */ 1
                   FROM customers
                   WHERE cust_id = s.cust_id
                     AND cust_year_of_birth = 1980)
       AND channel_id > 2 ;
SQL> @xplan
```

```
-----------------------------------------------------------------------------------------------------------------------------
| Id  | Operation            | Name      | Starts | E-Rows | A-Rows |   A-Time   | Buffers |  OMem |  1Mem | Used-Mem |
-----------------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT     |           |      1 |        |  10037 |00:00:00.64 |    5994 |       |       |          |
|*  1 |  HASH JOIN RIGHT SEMI|           |      1 |  72242 |  10037 |00:00:00.64 |    5994 |  1036K|  1036K| 1214K (0)|
|*  2 |   TABLE ACCESS FULL  | CUSTOMERS |      1 |    740 |    891 |00:00:00.01 |    1456 |       |       |          |
|*  3 |   TABLE ACCESS FULL  | SALES     |      1 |    689K|    660K|00:00:00.19 |    4538 |       |       |          |
-----------------------------------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   1 - access("CUST_ID"="S"."CUST_ID")
   2 - filter("CUST_YEAR_OF_BIRTH"=1980)
   3 - filter("CHANNEL_ID">2)
```

〉답안 2. Subquery 를 Join 문장으로 변경

```
SQL〉 SELECT s.prod_id, s.cust_id, s.channel_id, s.amount_sold
     FROM sales s, customers c
     WHERE s.cust_id = c.cust_id
       AND c.cust_year_of_birth = 1980
       AND s.channel_id 〉 2 ;
SQL〉 @xplan
```

```
-------------------------------------------------------------------------------------------------------------------------------------------
| Id  | Operation               | Name             | Starts | E-Rows | A-Rows |   A-Time   | Buffers |  OMem |  1Mem | Used-Mem |
-------------------------------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT        |                  |      1 |        |  10037 |00:00:00.71 |    4664 |       |       |          |
|*  1 |  HASH JOIN              |                  |      1 |  72242 |  10037 |00:00:00.71 |    4664 |  1036K|  1036K| 1215K (0)|
|*  2 |   VIEW                  | index$_join$_002 |      1 |    740 |    891 |00:00:00.05 |     126 |       |       |          |
|*  3 |    HASH JOIN            |                  |      1 |        |    891 |00:00:00.05 |     126 |   862K|   862K| 1223K (0)|
|*  4 |     INDEX RANGE SCAN    | CUST_YOB_IX      |      1 |    740 |    891 |00:00:00.01 |       4 |       |       |          |
|   5 |     INDEX FAST FULL SCAN| CUSTOMERS_PK     |      1 |    740 |  55500 |00:00:00.01 |     122 |       |       |          |
|*  6 |   TABLE ACCESS FULL     | SALES            |      1 |    689K|    660K|00:00:00.19 |    4538 |       |       |          |
-------------------------------------------------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   1 - access("S"."CUST_ID"="C"."CUST_ID")
   2 - filter("C"."CUST_YEAR_OF_BIRTH"=1980)
   3 - access(ROWID=ROWID)
   4 - access("C"."CUST_YEAR_OF_BIRTH"=1980)
   6 - filter("S"."CHANNEL_ID"〉2)
```

› 문제 3.

소속 부서의 최대 급여를 받는 사원 정보를 검색하는 문장이다. 실행 계획을 확인하고 성능을 최적화 시킨다.

```
SQL> CREATE INDEX empl_dept_sal_ix ON employees(department_id, salary) ;

SQL> SELECT last_name, salary, job_id, department_id
     FROM employees e
     WHERE salary = ( SELECT /*+ index_rs_desc(se empl_dept_sal_ix) */ salary
                      FROM employees se
                     WHERE department_id = e.department_id
                       AND ROWNUM        = 1 ) ;
SQL> @xplan
-------------------------------------------------------------------------------------------------------------------
| Id  | Operation                     | Name             | Starts | E-Rows | A-Rows |   A-Time   | Buffers |
-------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT              |                  |      1 |        |     11 |00:00:00.01 |      15 |
|*  1 |  FILTER                       |                  |      1 |        |     11 |00:00:00.01 |      15 |
|   2 |   TABLE ACCESS FULL           | EMPLOYEES        |      1 |    107 |    107 |00:00:00.01 |       4 |
|*  3 |   COUNT STOPKEY               |                  |     12 |        |     11 |00:00:00.01 |      11 |
|*  4 |    INDEX RANGE SCAN DESCENDING| EMPL_DEPT_SAL_IX |     12 |      2 |     11 |00:00:00.01 |      11 |
-------------------------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   1 - filter("SALARY"=)
   3 - filter(ROWNUM=1)
   4 - access("DEPARTMENT_ID"=:B1)
```

› 문제 사항

Subquery 에서의 ROWNUM 사용으로 Unnesting 이 진행되지 못하고 FILTER 를 진행한다. 후보행이 많은 경우 작업량은 증가하게 된다.

> 답안 1. MIN, MAX 함수 사용

```
SQL> SELECT e.last_name, e.salary, e.job_id, e.department_id
     FROM employees e
     WHERE e.salary = ( SELECT MAX(salary)
                        FROM employees se
                       WHERE se.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 |        |     11 |00:00:00.01 |       4 |       |       |          |
|*  1 |  FILTER              |                  |      1 |        |     11 |00:00:00.01 |       4 |       |       |          |
|   2 |   HASH GROUP BY      |                  |      1 |     18 |    106 |00:00:00.01 |       4 |   732K|   732K| 1288K (0)|
|*  3 |    HASH JOIN         |                  |      1 |   1021 |   3298 |00:00:00.01 |       4 |   968K|   968K|  837K (0)|
|   4 |     INDEX FULL SCAN  | EMPL_DEPT_SAL_IX |      1 |    107 |    107 |00:00:00.01 |       1 |       |       |          |
|   5 |     TABLE ACCESS FULL| EMPLOYEES        |      1 |    107 |    107 |00:00:00.01 |       3 |       |       |          |
-------------------------------------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   1 - filter("E"."SALARY"=MAX("SALARY"))
   3 - access("SE"."DEPARTMENT_ID"="E"."DEPARTMENT_ID")
```

해당 실행 계획은 아래와 같이 Query Transformation 이 진행되면서 실행된 것을 확인할 수 있다.

```
SQL> SELECT e.last_name, e.salary, e.job_id, e.department_id
     FROM employees e, employees se
     WHERE e.department_id = se.department_id
     GROUP BY se.department_id, e.rowid, e.department_id, e.job_id, e.salary, e.last_name
     HAVING e.salary = MAX(se.salary) ;

SQL> @xplan
-------------------------------------------------------------------------------------------------------------------------------
| Id  | Operation            | Name             | Starts | E-Rows | A-Rows |   A-Time   | Buffers |  OMem |  1Mem | Used-Mem |
-------------------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT     |                  |      1 |        |     11 |00:00:00.01 |       4 |       |       |          |
|*  1 |  FILTER              |                  |      1 |        |     11 |00:00:00.01 |       4 |       |       |          |
|   2 |   HASH GROUP BY      |                  |      1 |     18 |    106 |00:00:00.01 |       4 |   732K|   732K| 1288K (0)|
|*  3 |    HASH JOIN         |                  |      1 |   1021 |   3298 |00:00:00.01 |       4 |   778K|   778K|  837K (0)|
|   4 |     TABLE ACCESS FULL| EMPLOYEES        |      1 |    107 |    107 |00:00:00.01 |       3 |       |       |          |
|   5 |     INDEX FULL SCAN  | EMPL_DEPT_SAL_IX |      1 |    107 |    107 |00:00:00.01 |       1 |       |       |          |
-------------------------------------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   1 - filter("E"."SALARY"=MAX("SE"."SALARY"))
   3 - access("E"."DEPARTMENT_ID"="SE"."DEPARTMENT_ID")
```

과거의 튜닝 방법론으로 MIN, MAX 함수 대신 인덱스의 활용을 권장하던 시절이 있다. 지금도 경우에 따라 사용이 가능하다. 다만, 서브 쿼리에서의 ROWNUM 사용은 Query Transformation 을 불가능하게 한다. 즉, 서브 쿼리에서 ROWNUM 을 사용하면 Unnesting 이 안된다. 때문에 실행 계획은 FILTER 연산으로만 처리되고 이는 메인 쿼리의 후보행의 개수에 따라 성능상 취약점이 발생할 수 있다.

> 답안 2. 분석 함수 이용

```
SQL> SELECT last_name, salary, job_id, department_id
     FROM ( SELECT /*+ no_index(e(department_id)) */
                   last_name, salary, job_id, department_id,
                   MAX(salary) OVER(PARTITION BY department_id) AS max
            FROM employees e
            WHERE department_id IS NOT NULL)
     WHERE salary = max ;
SQL> @xplan
```

```
-------------------------------------------------------------------------------------------------------------------------------
| Id  | Operation           | Name      | Starts | E-Rows | A-Rows |   A-Time   | Buffers |  OMem |  1Mem | Used-Mem |
-------------------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT    |           |      1 |        |     11 |00:00:00.01 |       3 |       |       |          |
|*  1 |  VIEW               |           |      1 |    106 |     11 |00:00:00.01 |       3 |       |       |          |
|   2 |   WINDOW SORT       |           |      1 |    106 |    106 |00:00:00.01 |       3 |  9216 |  9216 | 8192  (0)|
|*  3 |    TABLE ACCESS FULL| EMPLOYEES |      1 |    106 |    106 |00:00:00.01 |       3 |       |       |          |
-------------------------------------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   1 - filter("SALARY"="MAX")
   3 - filter("DEPARTMENT_ID" IS NOT NULL)
```

일반적으로 테이블의 "FULL TABLE SCAN"을 피하기 위해 인덱스를 사용한다. 하지만 앞선 결과에서는 이미 EMPLOYEES 테이블을 FULL SCAN 을 수행했는데 MAX 값을 비교하기 위해 인덱스를 추가로 사용한다. 어차피 테이블의 전체를 읽었다면, 그 안에서 모든 비교를 가능하게 할 수 있도록 문장을 작성하는 것도 성능을 위한 문장 작성 방법이 된다.

〉문제 4.

소속 부서의 최대 급여를 받는 사원 정보를 검색하는 문장이다. 실행 계획을 확인하고 성능을 최적화 시킨다.

```
SQL〉 SELECT e.last_name, e.salary, d.department_id, d.department_name, d.location_id
     FROM employees e, departments d
     WHERE e.department_id = d.department_id
       AND e.salary = ( SELECT /*+ index_rs_desc(se empl_dept_sal_ix) */ salary
                        FROM employees se
                        WHERE se.department_id = e.department_id
                          AND ROWNUM           = 1 ) ;

SQL〉 @xplan
```

Id	Operation	Name	Starts	E-Rows	A-Rows	A-Time	Buffers	OMem	1Mem	Used-Mem
0	SELECT STATEMENT		1		11	00:00:00.01	18			
*1	FILTER		1		11	00:00:00.01	18			
2	MERGE JOIN		1	106	106	00:00:00.01	7			
3	TABLE ACCESS BY INDEX ROWID	DEPARTMENTS	1	27	27	00:00:00.01	4			
4	INDEX FULL SCAN	DEPT_ID_PK	1	27	27	00:00:00.01	2			
*5	SORT JOIN		27	107	106	00:00:00.01	3	6144	6144	6144 (0)
6	TABLE ACCESS FULL	EMPLOYEES	1	107	107	00:00:00.01	3			
*7	COUNT STOPKEY		11		11	00:00:00.01	11			
*8	INDEX RANGE SCAN DESCENDING	EMPL_DEPT_SAL_IX	11	2	11	00:00:00.01	11			

```
Predicate Information (identified by operation id):
---------------------------------------------------
   1 - filter("E"."SALARY"=)
   5 - access("E"."DEPARTMENT_ID"="D"."DEPARTMENT_ID")
       filter("E"."DEPARTMENT_ID"="D"."DEPARTMENT_ID")
   7 - filter(ROWNUM=1)
   8 - access("SE"."DEPARTMENT_ID"=:B1)
```

> 답안 1. 분석 함수 이용

```
SQL> SELECT e.last_name, e.salary, d.department_id, d.department_name, d.location_id
     FROM departments d, (SELECT last_name, salary, job_id, department_id
                          FROM ( SELECT last_name, salary, job_id, department_id,
                                        MAX(salary) OVER(PARTITION BY department_id) AS max
                                 FROM employees e )
                          WHERE salary = max ) 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		11	00:00:00.01	7			
1	MERGE JOIN		1	106	11	00:00:00.01	7			
2	TABLE ACCESS BY INDEX ROWID	DEPARTMENTS	1	27	27	00:00:00.01	4			
3	INDEX FULL SCAN	DEPT_ID_PK	1	27	27	00:00:00.01	2			
* 4	SORT JOIN		27	107	11	00:00:00.01	3	2048	2048	2048 (0)
* 5	VIEW		1	107	12	00:00:00.01	3			
6	WINDOW SORT		1	107	107	00:00:00.01	3	6144	6144	6144 (0)
7	TABLE ACCESS FULL	EMPLOYEES	1	107	107	00:00:00.01	3			

```
Predicate Information (identified by operation id):
---------------------------------------------------
   4 - access("D"."DEPARTMENT_ID"="DEPARTMENT_ID")
       filter("D"."DEPARTMENT_ID"="DEPARTMENT_ID")
   5 - filter("SALARY"="MAX")
```

```
SQL> SELECT last_name, salary, department_id, department_name, location_id
     FROM ( SELECT e.last_name, e.salary, d.department_id, d.department_name, d.location_id,
                   MAX(e.salary) OVER(PARTITION BY d.department_id) AS max
            FROM departments d, employees e
            WHERE d.department_id = e.department_id )
     WHERE salary = max ;
SQL> @xplan
```

Id	Operation	Name	Starts	E-Rows	A-Rows	A-Time	Buffers	OMem	1Mem	Used-Mem
0	SELECT STATEMENT		1		11	00:00:00.01	5			
* 1	VIEW		1	106	11	00:00:00.01	5			
2	WINDOW BUFFER		1	106	106	00:00:00.01	5	9216	9216	8192 (0)
3	MERGE JOIN		1	106	106	00:00:00.01	5			
4	TABLE ACCESS BY INDEX ROWID	DEPARTMENTS	1	27	27	00:00:00.01	2			
5	INDEX FULL SCAN	DEPT_ID_PK	1	27	27	00:00:00.01	1			
* 6	SORT JOIN		27	107	106	00:00:00.01	3	6144	6144	6144 (0)
7	TABLE ACCESS FULL	EMPLOYEES	1	107	107	00:00:00.01	3			

```
Predicate Information (identified by operation id):
---------------------------------------------------
   1 - filter("SALARY"="MAX")
   6 - access("D"."DEPARTMENT_ID"="E"."DEPARTMENT_ID")
       filter("D"."DEPARTMENT_ID"="E"."DEPARTMENT_ID")
```

> 답안 2. CRSW(Correlated Removal Subquery Using Window Function)

```
SQL> SELECT e.last_name, e.salary, d.department_id, d.department_name, d.location_id
     FROM employees e, departments d
     WHERE e.department_id = d.department_id
       AND e.salary = ( SELECT MAX(salary)
                        FROM employees se
                       WHERE se.department_id = d.department_id ) ;
SQL> @xplan
```

```
-------------------------------------------------------------------------------------------------------------------------------------------
| Id  | Operation                     | Name        | Starts | E-Rows | A-Rows |   A-Time   | Buffers |  OMem |  1Mem | Used-Mem |
-------------------------------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT              |             |      1 |        |     11 |00:00:00.01 |       5 |       |       |          |
|*  1 |  VIEW                         | VW_WIF_1    |      1 |    106 |     11 |00:00:00.01 |       5 |       |       |          |
|   2 |   WINDOW BUFFER               |             |      1 |    106 |    106 |00:00:00.01 |       5 |  9216 |  9216 | 8192  (0)|
|   3 |    MERGE JOIN                 |             |      1 |    106 |    106 |00:00:00.01 |       5 |       |       |          |
|   4 |     TABLE ACCESS BY INDEX ROWID| DEPARTMENTS |      1 |     27 |     27 |00:00:00.01 |       2 |       |       |          |
|   5 |      INDEX FULL SCAN          | DEPT_ID_PK  |      1 |     27 |     27 |00:00:00.01 |       1 |       |       |          |
|*  6 |     SORT JOIN                 |             |     27 |    107 |    106 |00:00:00.01 |       3 |  9216 |  9216 | 8192  (0)|
|   7 |      TABLE ACCESS FULL        | EMPLOYEES   |      1 |    107 |    107 |00:00:00.01 |       3 |       |       |          |
-------------------------------------------------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   1 - filter("VW_COL_6" IS NOT NULL)
   6 - access("E"."DEPARTMENT_ID"="D"."DEPARTMENT_ID")
       filter("E"."DEPARTMENT_ID"="D"."DEPARTMENT_ID")
```

Oracle Database 10g 부터 지원하는 RSW 를 이용할 수 있도록 MAX 함수를 사용하도록 문장을 수정한다. RSW 는 Group 함수 사용 시 동일 테이블의 반복을 최소화하고자 내부적으로 분석 함수 (MAX(salary) OVER(PARTITION BY department_id)를 이용하여 문장을 처리한다. 때문에 실행 계획상에서 EMPLOYEES 테이블의 접근은 한 번만 수행된다.

› 주의 사항

단, 다음 예문처럼 Subquery 에 공급되는 컬럼이 분석함수를 이용해야 하는 테이블과 같은 테이블의 컬럼일 경우 RSW 는 수행되지 않는다.

```
SQL> SELECT e.last_name, e.salary, d.department_id, d.department_name, d.location_id
     FROM employees e, departments d
     WHERE e.department_id = d.department_id
       AND e.employee_id   = ( SELECT MAX(se.employee_id)
                               FROM employees se
                              WHERE se.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 |        |     11 |00:00:00.01 |      22 |       |       |          |
|   1 |  NESTED LOOPS                 |             |      1 |        |     11 |00:00:00.01 |      22 |       |       |          |
|   2 |   NESTED LOOPS                |             |      1 |      1 |     11 |00:00:00.01 |      11 |       |       |          |
|*  3 |    HASH JOIN                  |             |      1 |      1 |     11 |00:00:00.01 |       7 |  951K |  951K |  871K (0)|
|   4 |     VIEW                      | VW_SQ_1     |      1 |     11 |     12 |00:00:00.01 |       3 |       |       |          |
|   5 |      HASH GROUP BY            |             |      1 |     11 |     12 |00:00:00.01 |       3 |  795K |  795K |  891K (0)|
|   6 |       TABLE ACCESS FULL       | EMPLOYEES   |      1 |    107 |    107 |00:00:00.01 |       3 |       |       |          |
|   7 |     TABLE ACCESS FULL         | EMPLOYEES   |      1 |    107 |    107 |00:00:00.01 |       4 |       |       |          |
|*  8 |    INDEX UNIQUE SCAN          | DEPT_ID_PK  |     11 |      1 |     11 |00:00:00.01 |       4 |       |       |          |
|   9 |   TABLE ACCESS BY INDEX ROWID | DEPARTMENTS |     11 |      1 |     11 |00:00:00.01 |      11 |       |       |          |
-------------------------------------------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   3 - access("E"."EMPLOYEE_ID"="MAX(SE.EMPLOYEE_ID)" AND "ITEM_1"="E"."DEPARTMENT_ID")
   8 - access("E"."DEPARTMENT_ID"="D"."DEPARTMENT_ID")
```

```
SQL> DROP INDEX empl_dept_sal_ix ;
```

› 문제 5.

다음 문장의 성능을 최적화 시키시오.

```
SQL> SELECT prod_id, channel_id, SUM(quantity_sold) AS qty, SUM(amount_sold) AS amt
     FROM sales
     WHERE prod_id IN ( SELECT prod_id
                        FROM products
                        WHERE prod_category_desc = 'Software/Other' )
     GROUP BY prod_id, channel_id ;
SQL> @xplan
```

```
-------------------------------------------------------------------------------------------------------------------------------------
| Id  | Operation           | Name     | Starts | E-Rows | A-Rows |   A-Time   | Buffers | Reads  |  OMem |  1Mem | Used-Mem |
-------------------------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT    |          |      1 |        |     82 |00:00:01.04 |    4441 |    126 |       |       |          |
|   1 |  HASH GROUP BY      |          |      1 |     82 |     82 |00:00:01.04 |    4441 |    126 |   746K|   746K| 2415K (0)|
|*  2 |   HASH JOIN         |          |      1 |    183K|    405K|00:00:00.91 |    4441 |    126 |  1066K|  1066K| 1187K (0)|
|*  3 |    TABLE ACCESS FULL| PRODUCTS |      1 |     14 |     26 |00:00:00.01 |       3 |      0 |       |       |          |
|   4 |    TABLE ACCESS FULL| SALES    |      1 |    918K|    918K|00:00:00.23 |    4438 |    126 |       |       |          |
-------------------------------------------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   2 - access("PROD_ID"="PROD_ID")
   3 - filter("PROD_CATEGORY_DESC"='Software/Other')
```

› 문제 사항

Subquery 가 Unnesting 되면서 조인문으로 변경되었고 Hash Join 을 수행하여 조인이 수행된다. 하지만 후보행이 많지 않은 관계에서의 Hash Join 은 작업의 낭비가 존재한다.

> 확인 사항 : Nested Loops Join 유도

```
SQL> SELECT /*+ qb_name(main) leading(p@sub s@main) use_nl(s@main) */
            prod_id, channel_id, SUM(quantity_sold) AS qty, SUM(amount_sold) AS amt
     FROM sales s
     WHERE prod_id IN ( SELECT /*+ qb_name(sub) */ prod_id
                        FROM products p
                        WHERE prod_category_desc = 'Software/Other' )
     GROUP BY prod_id, channel_id ;
SQL> @xplan
```

```
-------------------------------------------------------------------------------------------------------------------------------------
| Id  | Operation                     | Name          | Starts | E-Rows | A-Rows |   A-Time   | Buffers |  OMem |  1Mem | Used-Mem |
-------------------------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT              |               |      1 |        |     82 |00:00:01.11 |    3943 |       |       |          |
|   1 |  HASH GROUP BY                |               |      1 |     82 |     82 |00:00:01.11 |    3943 |   746K|   746K| 1278K (0)|
|   2 |   NESTED LOOPS                |               |      1 |        |   405K|00:00:00.98 |    3943 |       |       |          |
|   3 |    NESTED LOOPS               |               |      1 |   183K|   405K|00:00:00.28 |     877 |       |       |          |
|*  4 |     TABLE ACCESS FULL         | PRODUCTS      |      1 |     14 |     26 |00:00:00.01 |       3 |       |       |          |
|*  5 |     INDEX RANGE SCAN          | SALES_PROD_IX |     26 |  12762 |   405K|00:00:00.12 |     874 |       |       |          |
|   6 |    TABLE ACCESS BY INDEX ROWID| SALES         |   405K|  12762 |   405K|00:00:00.31 |    3066 |       |       |          |
-------------------------------------------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   4 - filter("PROD_CATEGORY_DESC"='Software/Other')
   5 - access("PROD_ID"="PROD_ID")
```

후보 행이 적으므로 Nested Loops Join 을 유도했으나 조건에 만족하는 SALES 테이블의 데이터가 많아서 성능 개선 효과가 없다.

> 답안. DSJ (Driving Semi Join) 사용

```
SQL> SELECT /*+ qb_name(main) semijoin_driver(@sub) */
            prod_id, channel_id, SUM(quantity_sold) AS qty, SUM(amount_sold) AS amt
     FROM sales s
     WHERE prod_id IN ( SELECT /*+ qb_name(sub) */ prod_id
                        FROM products p
                        WHERE prod_category_desc = 'Software/Other' )
     GROUP BY prod_id, channel_id ;
SQL> @xplan
```

```
-------------------------------------------------------------------------------------------------------------------------------------------
| Id  | Operation                          | Name          | Starts | E-Rows | A-Rows |   A-Time   | Buffers |  OMem |  1Mem | Used-Mem |
-------------------------------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                   |               |      1 |        |     82 |00:00:00.62 |    3056 |       |       |          |
|   1 |  HASH GROUP BY                     |               |      1 |     82 |     82 |00:00:00.62 |    3056 |   746K|   746K| 2409K (0)|
|   2 |   TABLE ACCESS BY INDEX ROWID      | SALES         |      1 |  78816 |    405K|00:00:00.49 |    3056 |       |       |          |
|   3 |    BITMAP CONVERSION TO ROWIDS     |               |      1 |        |    405K|00:00:00.28 |     877 |       |       |          |
|   4 |     BITMAP MERGE                   |               |      1 |        |      4 |00:00:00.20 |     877 |  1024K|   512K|80896  (0)|
|   5 |      BITMAP KEY ITERATION          |               |      1 |        |     26 |00:00:00.20 |     877 |       |       |          |
|*  6 |       TABLE ACCESS FULL            | PRODUCTS      |      1 |     14 |     26 |00:00:00.01 |       3 |       |       |          |
|   7 |       BITMAP CONVERSION FROM ROWIDS|               |     26 |        |     26 |00:00:00.20 |     874 |       |       |          |
|*  8 |        INDEX RANGE SCAN            | SALES_PROD_IX |     26 |        |    405K|00:00:00.11 |     874 |       |       |          |
-------------------------------------------------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   6 - filter("PROD_CATEGORY_DESC"='Software/Other')
   8 - access("PROD_ID"="PROD_ID")
```

Driving Semi Join 은 후보행이 적은 Subquery 를 먼저 Driving 하면서 Nested Loops Join 의 효과를 극대화 시킨다. Star Transformation Join 역시 DSJ 를 응용한다. 때문에 PRODUCTS 테이블과 SALES 테이블의 Bitmap Index 를 활용하여 먼저 조건에 만족하는 행을 추출하고 SALES 테이블에 접근하므로 불필요한 SALES 테이블의 Full Scan 을 막을 수 있다. 단, Bitmap Index 가 존재하지 않으면 B-Tree Index 를 활용하기 때문에 약간의 성능 저하는 발생할 수 있다.

> Bitmap Index 활용

```
SQL> DROP INDEX sales_prod_ix ;
SQL> CREATE BITMAP INDEX sales_prod_ix ON sales(prod_id) ;

SQL> SELECT /*+ qb_name(main) semijoin_driver(@sub) */
            prod_id, channel_id, SUM(quantity_sold) AS qty, SUM(amount_sold) AS amt
     FROM sales s
     WHERE prod_id IN ( SELECT /*+ qb_name(sub) */ prod_id
                        FROM products p
                        WHERE prod_category_desc = 'Software/Other' )
     GROUP BY prod_id, channel_id ;
SQL> @xplan
```

Id	Operation	Name	Starts	E-Rows	A-Rows	A-Time	Buffers	OMem	1Mem	Used-Mem
0	SELECT STATEMENT		1		82	00:00:00.42	2241			
1	HASH GROUP BY		1	82	82	00:00:00.42	2241	746K	746K	2409K (0)
2	TABLE ACCESS BY INDEX ROWID	SALES	1	78816	405K	00:00:00.29	2241			
3	BITMAP CONVERSION TO ROWIDS		1		405K	00:00:00.09	62			
4	BITMAP MERGE		1		4	00:00:00.01	62	1024K	512K	77824 (0)
5	BITMAP KEY ITERATION		1		30	00:00:00.01	62			
* 6	TABLE ACCESS FULL	PRODUCTS	1	14	26	00:00:00.01	3			
* 7	BITMAP INDEX RANGE SCAN	SALES_PROD_IX	26		30	00:00:00.01	59			

```
Predicate Information (identified by operation id):
---------------------------------------------------
   6 - filter("PROD_CATEGORY_DESC"='Software/Other')
   7 - access("PROD_ID"="PROD_ID")

SQL> DROP INDEX sales_prod_ix ;
SQL> CREATE INDEX sales_prod_ix ON sales(prod_id) ;
```

› 문제 6.

다음 문장의 성능을 인덱스를 사용하지 않고 최적화 시키시오.

```
SQL> SELECT prod_id, SUM(amount_sold)
     FROM sales
     GROUP BY prod_id
     ORDER BY prod_id ;
SQL> @xplan
```

```
-------------------------------------------------------------------------------------------------------------------
| Id  | Operation          | Name  | Starts | E-Rows | A-Rows |   A-Time   | Buffers |  OMem |  1Mem | Used-Mem |
-------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |       |      1 |        |     72 |00:00:00.55 |    4438 |       |       |          |
|   1 |  SORT GROUP BY     |       |      1 |     72 |     72 |00:00:00.55 |    4438 |  4096 |  4096 | 4096  (0)|
|   2 |   TABLE ACCESS FULL| SALES |      1 |    918K|    918K|00:00:00.23 |    4438 |       |       |          |
-------------------------------------------------------------------------------------------------------------------
```

› 문제 사항

```
SQL> SELECT prod_id, SUM(amount_sold)
     FROM sales
     GROUP BY prod_id ;
SQL> @xplan
```

```
-------------------------------------------------------------------------------------------------------------------
| Id  | Operation          | Name  | Starts | E-Rows | A-Rows |   A-Time   | Buffers |  OMem |  1Mem | Used-Mem |
-------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |       |      1 |        |     72 |00:00:00.47 |    4438 |       |       |          |
|   1 |  HASH GROUP BY     |       |      1 |     72 |     72 |00:00:00.47 |    4438 |   798K|   798K| 1257K (0)|
|   2 |   TABLE ACCESS FULL| SALES |      1 |    918K|    918K|00:00:00.22 |    4438 |       |       |          |
-------------------------------------------------------------------------------------------------------------------
```

Oracle 10g 부터 GROUP BY, DISTINCT 문장은 Hash 연산을 사용하여 수행된다. Sort 보다 메모리는 더 사용할 수 있으나 대량의 데이터의 작업에서는 보다 빠른 속도를 처리할 수 있다. 단, 정렬된 결과를 얻고자 ORDER BY 를 함께 사용한다면 불필요한 추가 정렬 작업을 제거하고자 예전 방식인 Sort 연산을 수행하여 처리된다.

> 답안.

```
SQL> SELECT *
     FROM (SELECT /*+ no_merge */ prod_id, SUM(amount_sold)
           FROM sales
           GROUP BY prod_id)
     ORDER BY prod_id ;
SQL> @xplan
```

Id	Operation	Name	Starts	E-Rows	A-Rows	A-Time	Buffers	OMem	1Mem	Used-Mem
0	SELECT STATEMENT		1		72	00:00:00.48	4438			
1	SORT ORDER BY		1	72	72	00:00:00.48	4438	6144	6144	6144 (0)
2	VIEW		1	72	72	00:00:00.48	4438			
3	HASH GROUP BY		1	72	72	00:00:00.48	4438	798K	798K	1265K (0)
4	TABLE ACCESS FULL	SALES	1	918K	918K	00:00:00.21	4438			

Inline View 를 이용하여 HASH GROUP BY 된 결과를 대상으로 정렬을 수행하면 정렬 작업의 부담을 최소화시킬 수 있다. 단, Inline View 는 메인 쿼리와 하나의 쿼리 블록으로 Transformation 이 수행될 수 있기 때문에 NO_MERGE 힌트를 사용한다.

```
SQL> SELECT *
     FROM (SELECT prod_id, SUM(amount_sold)
           FROM sales
           GROUP BY prod_id)
     ORDER BY prod_id ;
SQL> @xplan
```

Id	Operation	Name	Starts	E-Rows	A-Rows	A-Time	Buffers	OMem	1Mem	Used-Mem
0	SELECT STATEMENT		1		72	00:00:00.55	4438			
1	SORT GROUP BY		1	72	72	00:00:00.55	4438	4096	4096	4096 (0)
2	TABLE ACCESS FULL	SALES	1	918K	918K	00:00:00.23	4438			

힌트를 제거하면 Transformation 이 수행되면서 SORT ORDER BY 를 사용한다.

> 문제 7.

다음 문장의 성능을 최적화 시키시오.

```
SQL> SELECT prod_id, MAX(amount_sold) AS max_amt
     FROM sales
     GROUP BY prod_id
     HAVING MAX(amount_sold) > 1000 ;
SQL> @xplan
```

```
-------------------------------------------------------------------------------------------------------------------
| Id  | Operation           | Name  | Starts | E-Rows | A-Rows |   A-Time   | Buffers |  OMem |  1Mem | Used-Mem |
-------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT    |       |      1 |        |     10 |00:00:00.50 |    4438 |       |       |          |
|*  1 |  FILTER             |       |      1 |        |     10 |00:00:00.50 |    4438 |       |       |          |
|   2 |   HASH GROUP BY     |       |      1 |      4 |     72 |00:00:00.50 |    4438 |   795K|   795K| 1286K (0)|
|   3 |    TABLE ACCESS FULL| SALES |      1 |    918K|    918K|00:00:00.23 |    4438 |       |       |          |
-------------------------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   1 - filter(MAX("AMOUNT_SOLD")>1000)
```

> 답안 1. Index 사용 가능하도록 문장 수정

```
SQL> CREATE INDEX sales_amt_ix ON sales(amount_sold) ;
SQL> SELECT /*+ index(sales(amount_sold)) */ prod_id, MAX(amount_sold) AS max_amt
     FROM sales
     WHERE amount_sold > 1000
     GROUP BY prod_id
     HAVING MAX(amount_sold) > 1000 ;
SQL> @xplan
```

```
-----------------------------------------------------------------------------------------------------------------------------------
| Id  | Operation                     | Name         | Starts | E-Rows | A-Rows |   A-Time   | Buffers |  OMem |  1Mem | Used-Mem |
-----------------------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT              |              |      1 |        |     10 |00:00:00.05 |    2051 |       |       |          |
|*  1 |  FILTER                       |              |      1 |        |     10 |00:00:00.05 |    2051 |       |       |          |
|   2 |   HASH GROUP BY               |              |      1 |      4 |     10 |00:00:00.05 |    2051 |   795K|   795K| 1138K (0)|
|   3 |    TABLE ACCESS BY INDEX ROWID| SALES        |      1 |    404K|  32640 |00:00:00.04 |    2051 |       |       |          |
|*  4 |     INDEX RANGE SCAN          | SALES_AMT_IX |      1 |    404K|  32640 |00:00:00.01 |      75 |       |       |          |
-----------------------------------------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   1 - filter(MAX("AMOUNT_SOLD")>1000)
   4 - access("AMOUNT_SOLD">1000)
```

MAX, MIN 함수는 기존의 값을 수정하지 않는다. 때문에 "MAX(amount_sold) > 1000"의 조건식은 "amount_sold > 1000"의 범위 안에서 Grouping 을 수행해도 결과가 틀리지 않다. 조건식에 만족하는 작업량이 Full Scan 보다 적은 비용으로 처리가 가능하다면 위와 같이 문장을 수정할 수 있다.

하지만 조건식의 범위가 넓어지면?

```
SQL> SELECT /*+ index(sales(amount_sold)) */
            prod_id, MAX(amount_sold) AS max_amt
     FROM sales
     WHERE amount_sold > 10
     GROUP BY prod_id
     HAVING MAX(amount_sold) > 10 ;
SQL> @xplan
```

```
-----------------------------------------------------------------------------------------------------------------------------------
| Id  | Operation                     | Name         | Starts | E-Rows | A-Rows |   A-Time   | Buffers | Reads  |  OMem |  1Mem | Used-Mem |
-----------------------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT              |              |      1 |        |     71 |00:00:01.24 |   26124 |    156 |       |       |          |
|*  1 |  FILTER                       |              |      1 |        |     71 |00:00:01.24 |   26124 |    156 |       |       |          |
|   2 |   HASH GROUP BY               |              |      1 |      4 |     71 |00:00:01.24 |   26124 |    156 |   795K|   795K| 1257K (0)|
|   3 |    TABLE ACCESS BY INDEX ROWID| SALES        |      1 |    916K|    820K|00:00:00.99 |   26124 |    156 |       |       |          |
|*  4 |     INDEX RANGE SCAN          | SALES_AMT_IX |      1 |    916K|    820K|00:00:00.23 |    1736 |      0 |       |       |          |
-----------------------------------------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   1 - filter(MAX("AMOUNT_SOLD")>10)
   4 - access("AMOUNT_SOLD">10)
```

〉답안 2. CBPPD (Cost Based Predicate Push Down) 사용

```
SQL〉 SELECT *
     FROM (SELECT /*+ no_merge index(sales(amount_sold)) */
                 prod_id, MAX(amount_sold) AS max_amt
           FROM sales
           GROUP BY prod_id)
     WHERE max_amt 〉 1000 ;
SQL〉 @xplan
```

```
-------------------------------------------------------------------------------------------------------------------------------------
| Id  | Operation                     | Name         | Starts | E-Rows | A-Rows |   A-Time   | Buffers |  OMem |  1Mem | Used-Mem |
-------------------------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT              |              |      1 |        |     10 |00:00:00.05 |    2051 |       |       |          |
|   1 |  VIEW                         |              |      1 |      4 |     10 |00:00:00.05 |    2051 |       |       |          |
|*  2 |   FILTER                      |              |      1 |        |     10 |00:00:00.05 |    2051 |       |       |          |
|   3 |    HASH GROUP BY              |              |      1 |      4 |     10 |00:00:00.05 |    2051 |   795K|   795K| 1135K (0)|
|   4 |     TABLE ACCESS BY INDEX ROWID| SALES        |      1 |    404K|  32640 |00:00:00.04 |    2051 |       |       |          |
|*  5 |      INDEX RANGE SCAN         | SALES_AMT_IX |      1 |    404K|  32640 |00:00:00.01 |      75 |       |       |          |
-------------------------------------------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   2 - filter(MAX("AMOUNT_SOLD")>1000)
   5 - access("AMOUNT_SOLD">1000)
```

Complex View 사용 시 Cost Based Predicate Push Down 은 비용을 기반으로 Inline View 안으로 Main Query 의 조건절을 Push Down 여부를 결정짓는 Query Transformation 의 한 종류이다. 위의 문장에서는 힌트를 이용하여 작업했지만 더 많은 데이터가 존재한다고 가정하면 경우에 따라 NO_MERGE, INDEX 힌트 없이도 상황에 만족하는 최적화된 실행 계획이 생성될 수 있다. (추가 실습 확인)

〉추가 실습 : 값의 범위에 따라 인덱스 사용 유무 확인

```
SQL〉 SELECT *
      FROM (SELECT /*+ no_merge */
                   prod_id, MAX(amount_sold) AS max_amt
            FROM sales
            GROUP BY prod_id)
      WHERE max_amt 〉 10000 ;
SQL〉 @xplan
```

```
-------------------------------------------------------------------------------------------------------------------------------
| Id  | Operation                     | Name         | Starts | E-Rows | A-Rows |   A-Time   | Buffers |  OMem |  1Mem | Used-Mem |
-------------------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT              |              |      1 |        |      0 |00:00:00.01 |       3 |       |       |          |
|   1 |  VIEW                         |              |      1 |      1 |      0 |00:00:00.01 |       3 |       |       |          |
|*  2 |   FILTER                      |              |      1 |        |      0 |00:00:00.01 |       3 |       |       |          |
|   3 |    HASH GROUP BY              |              |      1 |      1 |      0 |00:00:00.01 |       3 |   848K|   848K|          |
|   4 |     TABLE ACCESS BY INDEX ROWID| SALES        |      1 |      1 |      0 |00:00:00.01 |       3 |       |       |          |
|*  5 |      INDEX RANGE SCAN         | SALES_AMT_IX |      1 |      1 |      0 |00:00:00.01 |       3 |       |       |          |
-------------------------------------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   2 - filter(MAX("AMOUNT_SOLD")>10000)
   5 - access("AMOUNT_SOLD">10000)
```

```
SQL〉 SELECT *
      FROM (SELECT /*+ no_merge */
                   prod_id, MAX(amount_sold) AS max_amt
            FROM sales
            GROUP BY prod_id)
      WHERE max_amt 〉 10 ;
SQL〉 @xplan
```

```
-------------------------------------------------------------------------------------------------------------------------------
| Id  | Operation            | Name  | Starts | E-Rows | A-Rows |   A-Time   | Buffers |  OMem |  1Mem | Used-Mem |
-------------------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT     |       |      1 |        |     71 |00:00:00.46 |    4438 |       |       |          |
|   1 |  VIEW                |       |      1 |      4 |     71 |00:00:00.46 |    4438 |       |       |          |
|*  2 |   FILTER             |       |      1 |        |     71 |00:00:00.46 |    4438 |       |       |          |
|   3 |    HASH GROUP BY     |       |      1 |      4 |     71 |00:00:00.46 |    4438 |   795K|   795K| 1275K (0)|
|*  4 |     TABLE ACCESS FULL| SALES |      1 |    916K|    820K|00:00:00.21 |    4438 |       |       |          |
-------------------------------------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   2 - filter(MAX("AMOUNT_SOLD")>10)
   4 - filter("AMOUNT_SOLD">10)
```

```
SQL〉 DROP INDEX sales_amt_ix ;
```

> 추가 실습 : CBPPD (Cost Based Predicate Push Down) 활용

```
SQL> SELECT d.deptno, d.dname, e.sum_sal
     FROM dept d, (SELECT deptno, SUM(sal) AS sum_sal
                   FROM emp
                   GROUP BY deptno) e
     WHERE d.deptno = e.deptno
       AND d.deptno = 30 ;
SQL> @xplan
```

```
-------------------------------------------------------------------------------------------------------------------------------------------
| Id  | Operation                     | Name          | Starts | E-Rows | A-Rows |   A-Time   | Buffers |  OMem |  1Mem | Used-Mem |
-------------------------------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT              |               |      1 |        |      1 |00:00:00.01 |       4 |       |       |          |
|   1 |  NESTED LOOPS                 |               |      1 |      1 |      1 |00:00:00.01 |       4 |       |       |          |
|   2 |   TABLE ACCESS BY INDEX ROWID | DEPT          |      1 |      1 |      1 |00:00:00.01 |       2 |       |       |          |
|*  3 |    INDEX UNIQUE SCAN          | PK_DEPT       |      1 |      1 |      1 |00:00:00.01 |       1 |       |       |          |
|   4 |   VIEW                        |               |      1 |      1 |      1 |00:00:00.01 |       2 |       |       |          |
|   5 |    SORT GROUP BY              |               |      1 |      1 |      1 |00:00:00.01 |       2 |  2048 |  2048 | 2048  (0)|
|   6 |     TABLE ACCESS BY INDEX ROWID| EMP          |      1 |      5 |      6 |00:00:00.01 |       2 |       |       |          |
|*  7 |      INDEX RANGE SCAN         | EMP_DEPTNO_IX |      1 |      5 |      6 |00:00:00.01 |       1 |       |       |          |
-------------------------------------------------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   3 - access("D"."DEPTNO"=30)
   7 - access("DEPTNO"=30)
```

위 문장은 NO_MERGE 힌트를 사용한 것도 아니지만 View Merging 이 수행되지 않고 Main Query 의 조건절이 Push Down 되어 EMP 테이블의 전체가 아닌 30 번 부서번호에 대해서만 그룹함수를 계산하고 있다. 이렇듯 CBPPD 는 비용을 통해 필요에 따라 최적화를 시도한다.

> 추가 실습 : FPD (Filter Push Down) 확인

```
SQL> SELECT d.deptno, d.dname, e.empno, e.ename, e.job, e.sal
     FROM dept d, (SELECT /*+ no_merge */ *
                   FROM emp
                   WHERE TO_CHAR(hiredate,'YYYY') = '1981') e
     WHERE d.deptno = e.deptno
       AND e.job = 'CLERK' ;
SQL> @xplan
```

```
-------------------------------------------------------------------------------------------------------------
| Id  | Operation                     | Name       | Starts | E-Rows | A-Rows |   A-Time   | Buffers |
-------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT              |            |      1 |        |      1 |00:00:00.01 |       6 |
|   1 |  NESTED LOOPS                 |            |      1 |        |      1 |00:00:00.01 |       6 |
|   2 |   NESTED LOOPS                |            |      1 |      1 |      1 |00:00:00.01 |       5 |
|   3 |    VIEW                       |            |      1 |      1 |      1 |00:00:00.01 |       4 |
|*  4 |     TABLE ACCESS BY INDEX ROWID| EMP        |      1 |      1 |      1 |00:00:00.01 |       4 |
|*  5 |      INDEX RANGE SCAN         | EMP_JOB_IX |      1 |      4 |      4 |00:00:00.01 |       2 |
|*  6 |    INDEX UNIQUE SCAN          | PK_DEPT    |      1 |      1 |      1 |00:00:00.01 |       1 |
|   7 |   TABLE ACCESS BY INDEX ROWID | DEPT       |      1 |      1 |      1 |00:00:00.01 |       1 |
-------------------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   4 - filter(TO_CHAR(INTERNAL_FUNCTION("HIREDATE"),'YYYY')='1981')
   5 - access("EMP"."JOB"='CLERK')
   6 - access("D"."DEPTNO"="E"."DEPTNO")
```

Simple View 를 사용하는 경우 View Merging 이 수행되지 않고 Main Query 에서 Inline View 의 컬럼에 조건식이 정의되면 별도의 힌트 없이 Push Down 이 수행된다. 때문에 View 집합에 접근 시 EMP_JOB_IX 인덱스를 사용하는 실행 계획을 확인할 수 있다. 이러한 FPD 는 항상 더 나은 성능을 보장하므로 별도의 힌트를 제공하지 않는다.

만약 위의 문장에서 View Merging 이 수행되면 ID.3 의 VIEW 집합 생성만 제거된다. 즉, View Merge 가 수행되면 다음과 같은 문장이 수행된다.

```
SQL> SELECT d.deptno, d.dname, e.empno, e.ename, e.job, e.sal
     FROM dept d, emp e
     WHERE d.deptno = e.deptno
       AND e.job = 'CLERK'
       AND TO_CHAR(hiredate,'YYYY') = '1981' ;
...
```

단, ROWNUM, RANK 등의 분석 함수 사용시 Push Down 은 수행 안 된다.

```
SQL> SELECT d.deptno, d.dname, e.empno, e.ename, e.job, e.sal, e.rk
     FROM dept d, (SELECT /*+ no_merge */
                          empno, ename, job, sal, deptno, RANK() OVER(ORDER BY sal DESC) AS rk
                   FROM emp
                   WHERE TO_CHAR(hiredate,'YYYY') = '1981' ) e
     WHERE d.deptno = e.deptno
       AND e.job = 'CLERK' ;
SQL> @xplan
```

```
-------------------------------------------------------------------------------------------------------------------------------------
| Id  | Operation                     | Name    | Starts | E-Rows | A-Rows |   A-Time   | Buffers |  OMem |  1Mem | Used-Mem |
-------------------------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT              |         |      1 |        |      1 |00:00:00.01 |       6 |       |       |          |
|   1 |  MERGE JOIN                   |         |      1 |     10 |      1 |00:00:00.01 |       6 |       |       |          |
|   2 |   TABLE ACCESS BY INDEX ROWID| DEPT    |      1 |      4 |      4 |00:00:00.01 |       4 |       |       |          |
|   3 |    INDEX FULL SCAN            | PK_DEPT |      1 |      4 |      4 |00:00:00.01 |       2 |       |       |          |
|*  4 |   SORT JOIN                   |         |      4 |     10 |      1 |00:00:00.01 |       2 |  2048 |  2048 | 2048  (0)|
|*  5 |    VIEW                       |         |      1 |     10 |      1 |00:00:00.01 |       2 |       |       |          |
|   6 |     WINDOW SORT               |         |      1 |     10 |     10 |00:00:00.01 |       2 |  2048 |  2048 | 2048  (0)|
|*  7 |      TABLE ACCESS FULL        | EMP     |      1 |     10 |     10 |00:00:00.01 |       2 |       |       |          |
-------------------------------------------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   4 - access("D"."DEPTNO"="E"."DEPTNO")
       filter("D"."DEPTNO"="E"."DEPTNO")
   5 - filter("E"."JOB"='CLERK')
   7 - filter(TO_CHAR(INTERNAL_FUNCTION("HIREDATE"),'YYYY')='1981')
```

〉문제 8.

다음 문장의 실행 계획을 확인하고 성능을 최적화 시키시오.

```
SQL〉 SELECT cust_id, cust_last_name, cust_year_of_birth, cust_city,
            (SELECT COUNT(cust_id)
             FROM sales s
             WHERE s.cust_id = c.cust_id
               AND TO_CHAR(time_id,'YYYYMM') = '199810' ) AS CNT
      FROM customers c
      WHERE ( country_id = 52778
              AND NOT EXISTS ( SELECT 1 FROM sales
                               WHERE cust_id = c.cust_id ) )
         OR ( country_id = 52778
              AND cust_id IN ( SELECT cust_id
                               FROM sales
                               WHERE TO_CHAR(time_id,'YYYYMM') = '199810'
                               GROUP BY cust_id
                               HAVING COUNT(cust_id) 〉= 10))
      ORDER BY cnt DESC ;
SQL〉 @xplan
```

```
-------------------------------------------------------------------------------------------------------------------------------------
| Id  | Operation                    | Name          | Starts | E-Rows | A-Rows |   A-Time   | Buffers |  OMem |  1Mem | Used-Mem |
-------------------------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |               |      1 |        |   1950 |00:00:24.33 |     484K|       |       |          |
|   1 |  SORT AGGREGATE              |               |   1950 |      1 |   1950 |00:00:00.01 |    6439 |       |       |          |
|*  2 |   TABLE ACCESS BY INDEX ROWID| SALES         |   1950 |      1 |    316 |00:00:00.01 |    6439 |       |       |          |
|*  3 |    INDEX RANGE SCAN          | SALES_CUST_IX |   1950 |    130 |   3008 |00:00:00.01 |    3906 |       |       |          |
|   4 |  SORT ORDER BY               |               |      1 |   2921 |   1950 |00:00:24.33 |     484K|   106K|   106K|96256  (0)|
|*  5 |   FILTER                     |               |      1 |        |   1950 |00:00:24.31 |     478K|       |       |          |
|*  6 |    TABLE ACCESS FULL         | CUSTOMERS     |      1 |   2921 |   2039 |00:00:00.01 |    1456 |       |       |          |
|*  7 |    INDEX RANGE SCAN          | SALES_CUST_IX |   2039 |      2 |    106 |00:00:00.01 |    6119 |       |       |          |
|*  8 |    FILTER                    |               |    106 |        |     17 |00:00:24.30 |     470K|       |       |          |
|   9 |     HASH GROUP BY            |               |    106 |      1 |    199K|00:00:24.25 |     470K|   870K|   870K| 1283K (0)|
|* 10 |      TABLE ACCESS FULL       | SALES         |    106 |   9188 |   2107K|00:00:23.48 |     470K|       |       |          |
-------------------------------------------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   2 - filter(TO_CHAR(INTERNAL_FUNCTION("TIME_ID"),'YYYYMM')='199810')
   3 - access("S"."CUST_ID"=:B1)
   5 - filter(( IS NULL OR  IS NOT NULL))
   6 - filter("COUNTRY_ID"=52778)
   7 - access("CUST_ID"=:B1)
   8 - filter(("CUST_ID"=:B1 AND COUNT(*)〉=10))
  10 - filter(TO_CHAR(INTERNAL_FUNCTION("TIME_ID"),'YYYYMM')='199810')
```

〉문제 사항

- Subquery Unnest, View Merge 가 수행되지 않음
- SALES 테이블이 3개의 Query Block을 통해 반복적인 액세스 진행

〉답안 1. UNION ALL 사용

- 배타적인 조건절을 분리하여 UNION ALL로 연결 (각각의 Query Block 의 최적화 가능)
- 중복되는 Query Block을 최소화

```
SQL〉 SELECT c.cust_id,c.cust_last_name,c.cust_year_of_birth,c.cust_city, COUNT(s.cust_id) CNT
     FROM customers c , sales s
     WHERE c.cust_id = s.cust_id
       AND c.country_id = 52778
       AND TO_CHAR(s.time_id,'YYYYMM') = '199810'
     GROUP BY c.cust_id,c.cust_last_name,c.cust_year_of_birth,c.cust_city
     HAVING COUNT(s.cust_id) 〉= 10
     UNION ALL
     SELECT cust_id, cust_last_name, cust_year_of_birth, cust_city, 0
     FROM customers c
     WHERE country_id = 52778
      AND NOT EXISTS ( SELECT 1
                       FROM sales
                       WHERE cust_id = c.cust_id )
     ORDER BY 5 DESC ;
SQL〉 @xplan
```

```
-------------------------------------------------------------------------------------------------------------------------------------
| Id  | Operation               | Name          | Starts | E-Rows | A-Rows |   A-Time   | Buffers |  OMem |  1Mem | Used-Mem |
-------------------------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT        |               |      1 |        |   1950 |00:00:01.05 |    9312 |       |       |          |
|   1 |  SORT ORDER BY          |               |      1 |    466 |   1950 |00:00:01.05 |    9312 |   106K|   106K|96256  (0)|
|   2 |   UNION-ALL             |               |      1 |        |   1950 |00:00:01.05 |    9312 |       |       |          |
|*  3 |    FILTER               |               |      1 |        |     17 |00:00:00.25 |    5894 |       |       |          |
|   4 |     HASH GROUP BY       |               |      1 |    437 |     34 |00:00:00.24 |    5894 |   764K|   764K| 1325K (0)|
|*  5 |      HASH JOIN          |               |      1 |   8732 |    377 |00:00:00.24 |    5894 |   801K|   801K| 1219K (0)|
|*  6 |       TABLE ACCESS FULL | CUSTOMERS     |      1 |   2921 |   2039 |00:00:00.01 |    1456 |       |       |          |
|*  7 |       TABLE ACCESS FULL | SALES         |      1 |  19878 |  19878 |00:00:00.22 |    4438 |       |       |          |
|*  8 |    HASH JOIN ANTI       |               |      1 |     29 |   1933 |00:00:00.80 |    3418 |   801K|   801K| 1247K (0)|
|*  9 |     TABLE ACCESS FULL   | CUSTOMERS     |      1 |   2921 |   2039 |00:00:00.01 |    1456 |       |       |          |
|  10 |     INDEX FAST FULL SCAN| SALES_CUST_IX |      1 |    918K|    918K|00:00:00.22 |    1962 |       |       |          |
-------------------------------------------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   3 - filter(COUNT(*)>=10)
   5 - access("C"."CUST_ID"="S"."CUST_ID")
   6 - filter("C"."COUNTRY_ID"=52778)
   7 - filter(TO_CHAR(INTERNAL_FUNCTION("S"."TIME_ID"),'YYYYMM')='199810')
   8 - access("CUST_ID"="C"."CUST_ID")
   9 - filter("COUNTRY_ID"=52778)
```

> 답안 2. 전체 범위 처리의 최적화

```
SQL> SELECT c.cust_id,c.cust_last_name,c.cust_year_of_birth,c.cust_city, COUNT(s.cust_id) AS CNT
     FROM sales s,
         (SELECT cust_id,cust_last_name,cust_year_of_birth,cust_city
          FROM customers c
          WHERE country_id = 52778) c
     WHERE c.cust_id = s.cust_id (+)
       AND DECODE(s.cust_id,NULL,'199810',TO_CHAR(time_id,'YYYYMM')) = '199810'
     GROUP BY c.cust_id,c.cust_last_name,c.cust_year_of_birth,c.cust_city
     HAVING COUNT(s.cust_id) >= 10
         OR COUNT(s.cust_id) = 0
     ORDER BY cnt DESC ;
SQL> @xplan
```

```
-------------------------------------------------------------------------------------------------------------------------
| Id  | Operation             | Name      | Starts | E-Rows | A-Rows |   A-Time   | Buffers |  OMem |  1Mem | Used-Mem |
-------------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT      |           |      1 |        |   1950 |00:00:00.84 |    5894 |       |       |          |
|   1 |  SORT ORDER BY        |           |      1 |  22624 |   1950 |00:00:00.84 |    5894 |   106K|   106K|96256  (0)|
|*  2 |   FILTER              |           |      1 |        |   1950 |00:00:00.84 |    5894 |       |       |          |
|   3 |    HASH GROUP BY      |           |      1 |  22624 |   1967 |00:00:00.84 |    5894 |   747K|   747K| 6754K (0)|
|*  4 |     FILTER            |           |      1 |        |   2310 |00:00:00.83 |    5894 |       |       |          |
|*  5 |      HASH JOIN OUTER  |           |      1 |    380K|  19069 |00:00:00.82 |    5894 |   801K|   801K| 1239K (0)|
|*  6 |       TABLE ACCESS FULL| CUSTOMERS |      1 |   2921 |   2039 |00:00:00.01 |    1456 |       |       |          |
|   7 |       TABLE ACCESS FULL| SALES     |      1 |    918K|    918K|00:00:00.22 |    4438 |       |       |          |
-------------------------------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   2 - filter((COUNT("S"."CUST_ID")>=10 OR COUNT("S"."CUST_ID")=0))
   4 - filter(DECODE(TO_CHAR("S"."CUST_ID"),NULL,'199810',TO_CHAR(INTERNAL_FUNCTION("TIME_ID"),'YYYYMM'))='199810')
   5 - access("CUST_ID"="S"."CUST_ID")
   6 - filter("COUNTRY_ID"=52778)
```

> 답안 3. 부분 범위 처리의 최적화

```
SQL> CREATE INDEX custs_x01 ON customers(country_id, cust_id) ;
SQL> CREATE INDEX sales_x01 ON sales(cust_id, time_id) ;
SQL> SELECT /*+ index(c custs_x01) */
            cust_id, cust_last_name, cust_year_of_birth, cust_city,
           (SELECT COUNT(cust_id) FROM sales s
             WHERE s.cust_id = c.cust_id
               AND TO_CHAR(time_id,'YYYYMM') = '199810' ) AS CNT
     FROM customers c
     WHERE (NOT EXISTS ( SELECT /*+ no_unnest push_subq */ 1 FROM sales
                         WHERE cust_id = c.cust_id )
            OR EXISTS ( SELECT 1 FROM sales
                        WHERE cust_id = c.cust_id
                          AND TO_CHAR(time_id,'YYYYMM') = '199810'
                        GROUP BY cust_id
                       HAVING COUNT(cust_id) >= 10))
       AND country_id = 52778
    ORDER BY cnt DESC ;
SQL> @xplan
```

Id	Operation	Name	Starts	E-Rows	A-Rows	A-Time	Buffers	OMem	1Mem	Used-Mem
0	SELECT STATEMENT		1		1950	00:00:00.03	8840			
1	SORT AGGREGATE		1950	1	1950	00:00:00.01	2212			
*2	INDEX RANGE SCAN	SALES_X01	1950	1	316	00:00:00.01	2212			
3	SORT ORDER BY		1	1950	1950	00:00:00.03	8840	106K	106K	96256 (0)
*4	TABLE ACCESS BY INDEX ROWID	CUSTOMERS	1	1950	1950	00:00:00.02	6628			
*5	INDEX RANGE SCAN	CUSTS_X01	1	2039	2039	00:00:00.01	8			
*6	INDEX RANGE SCAN	SALES_CUST_IX	2039	2	106	00:00:00.01	4233			
*7	FILTER		106		17	00:00:00.01	368			
8	SORT GROUP BY NOSORT		106	1	34	00:00:00.01	368			
*9	INDEX RANGE SCAN	SALES_X01	106	1	377	00:00:00.01	368			

```
Predicate Information (identified by operation id):
---------------------------------------------------
   2 - access("S"."CUST_ID"=:B1)
       filter(TO_CHAR(INTERNAL_FUNCTION("TIME_ID"),'YYYYMM')='199810')
   4 - filter(( IS NULL OR  IS NOT NULL))
   5 - access("COUNTRY_ID"=52778)
   6 - access("CUST_ID"=:B1)
   7 - filter(COUNT(*)>=10)
   9 - access("CUST_ID"=:B1)
       filter(TO_CHAR(INTERNAL_FUNCTION("TIME_ID"),'YYYYMM')='199810')

SQL> DROP INDEX custs_x01 ;
SQL> DROP INDEX sales_x01 ;
```

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 | E-Rows | A-Rows |   A-Time   | Buffers |  OMem |  1Mem | Used-Mem |
-------------------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |               |      1 |        |     14 |00:00:00.01 |       6 |       |       |          |
|   1 |  MERGE JOIN                  |               |      1 |     14 |     14 |00:00:00.01 |       6 |       |       |          |
|   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 |       2 |  2048 |  2048 | 2048  (0)|
|   5 |    VIEW                      |               |      1 |      3 |      3 |00:00:00.01 |       2 |       |       |          |
|   6 |     HASH GROUP BY            |               |      1 |      3 |      3 |00:00:00.01 |       2 |   801K|   801K|  641K (0)|
|   7 |      TABLE ACCESS FULL       | EMP           |      1 |     14 |     14 |00:00:00.01 |       2 |       |       |          |
-------------------------------------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   4 - access("A"."DEPTNO"="B"."DEPTNO")
       filter("A"."DEPTNO"="B"."DEPTNO")
```

```
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 | E-Rows | A-Rows |   A-Time   | Buffers |
-----------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |               |      1 |        |     14 |00:00:00.01 |       3 |
|   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 |       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 | E-Rows | A-Rows |   A-Time   | Buffers |  OMem |  1Mem | Used-Mem |
-----------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |               |      1 |        |     14 |00:00:00.01 |       6 |       |       |          |
|   1 |  MERGE JOIN                  |               |      1 |     14 |     14 |00:00:00.01 |       6 |       |       |          |
|   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 |       2 |  2048 |  2048 | 2048  (0)|
|   5 |    VIEW                      |               |      1 |      3 |      3 |00:00:00.01 |       2 |       |       |          |
|   6 |     HASH GROUP BY            |               |      1 |      3 |      3 |00:00:00.01 |       2 |   801K|   801K|  641K (0)|
|   7 |      TABLE ACCESS FULL       | EMP           |      1 |     14 |     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 | E-Rows | A-Rows |   A-Time   | Buffers |
-----------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |               |      1 |        |     14 |00:00:00.01 |       3 |
|   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 |       3 |
-----------------------------------------------------------------------------------------------------------------
```

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 | E-Rows | A-Rows |   A-Time   | Buffers |  OMem |  1Mem | 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 |     14 |00:00:00.01 |       5 |       |       |          |
|   3 |    VIEW                       |               |      1 |      3 |      3 |00:00:00.01 |       2 |       |       |          |
|   4 |     HASH GROUP BY             |               |      1 |      3 |      3 |00:00:00.01 |       2 |   801K|   801K|  675K (0)|
|   5 |      TABLE ACCESS FULL        | EMP           |      1 |     14 |     14 |00:00:00.01 |       2 |       |       |          |
|*  6 |    INDEX RANGE SCAN           | EMP_DEPTNO_IX |      3 |      5 |     14 |00:00:00.01 |       3 |       |       |          |
|   7 |   TABLE ACCESS BY INDEX ROWID | EMP           |     14 |      5 |     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 | E-Rows | A-Rows |   A-Time   | Buffers |
-------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT              |               |      1 |        |     14 |00:00:00.01 |       3 |
|   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 |       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

Aggregate, Analytic Functions

> 문제 1.

EMP 테이블에서 부서별 급여의 합계를 검색하면서 전체 급여의 합계도 함께 검색하시오.

> 출력 결과

DEPTNO	SUM_SAL
10	8750
20	10875
30	9400
	29025

> 답안 1. UNION ALL 사용

```
SQL> SELECT deptno, SUM(sal) AS sum_sal
     FROM emp
     GROUP BY deptno
     UNION ALL
     SELECT NULL, SUM(sal)
     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		4	00:00:00.01	4			
1	SORT ORDER BY		1	4	4	00:00:00.01	4	2048	2048	2048 (0)
2	UNION-ALL		1		4	00:00:00.01	4			
3	HASH GROUP BY		1	3	3	00:00:00.01	2	801K	801K	638K (0)
4	TABLE ACCESS FULL	EMP	1	14	14	00:00:00.01	2			
5	SORT AGGREGATE		1	1	1	00:00:00.01	2			
6	TABLE ACCESS FULL	EMP	1	14	14	00:00:00.01	2			

다양한 Grouping이 필요할 때 가장 손쉬운 방법이 UNION ALL을 이용하는 문장이지만 각각의 집합의 반복 접근이 성능을 저하시킬 수 있다. 하지만 경우에 따라 각 집합을 개별적으로 최적화하는 것도 가능하므로 필요하다면 사용한다.

> 답안 2. ROLLUP 사용

```
SQL> SELECT deptno, SUM(sal) AS sum_sal
     FROM emp
     GROUP BY ROLLUP(deptno) ;
SQL> @xplan
```

Id	Operation	Name	Starts	E-Rows	A-Rows	A-Time	Buffers	OMem	1Mem	Used-Mem
0	SELECT STATEMENT		1		4	00:00:00.01	2			
1	SORT GROUP BY ROLLUP		1	3	4	00:00:00.01	2	2048	2048	2048 (0)
2	TABLE ACCESS FULL	EMP	1	14	14	00:00:00.01	2			

GROUP BY의 확장된 사용 방법이며 ROLLUP, CUBE를 이용하면 하나의 쿼리 집합을 통해 다양한 Grouping 결과가 검색 가능하다. 테이블의 반복 접근은 줄기 때문에 성능 향상이 있을 수도 있지만 아닌 경우도 존재한다.
때문에 실제 사용된 실행 계획 및 성능 측정이 필요하다.

> 답안 3. Join 활용

```
SQL> CREATE INDEX emp_x01 ON emp(deptno, sal) ;

SQL> SELECT DECODE(no,1,deptno) AS deptno, SUM(sum_sal) AS sum_sal
     FROM ( SELECT /*+ index(emp(deptno,sal)) */ deptno, SUM(sal) AS sum_sal
            FROM emp
            WHERE deptno IS NOT NULL
            GROUP BY deptno ) a,
          ( SELECT level AS no
            FROM dual
            CONNECT BY level <= 2 ) b
    GROUP BY DECODE(no,1,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 |        |      4 |00:00:00.01 |       1 |       |       |          |
|   1 |  SORT GROUP BY                 |         |      1 |      3 |      4 |00:00:00.01 |       1 |  2048 |  2048 | 2048  (0)|
|   2 |   MERGE JOIN CARTESIAN         |         |      1 |      3 |      6 |00:00:00.01 |       1 |       |       |          |
|   3 |    VIEW                        |         |      1 |      1 |      2 |00:00:00.01 |       0 |       |       |          |
|   4 |     CONNECT BY WITHOUT FILTERING|         |      1 |        |      2 |00:00:00.01 |       0 |       |       |          |
|   5 |      FAST DUAL                 |         |      1 |      1 |      1 |00:00:00.01 |       0 |       |       |          |
|   6 |    BUFFER SORT                 |         |      2 |      3 |      6 |00:00:00.01 |       1 |  2048 |  2048 | 2048  (0)|
|   7 |     VIEW                       |         |      1 |      3 |      3 |00:00:00.01 |       1 |       |       |          |
|   8 |      HASH GROUP BY             |         |      1 |      3 |      3 |00:00:00.01 |       1 |   801K|   801K|  661K (0)|
|*  9 |       INDEX FULL SCAN          | EMP_X01 |      1 |     14 |     14 |00:00:00.01 |       1 |       |       |          |
-------------------------------------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   9 - filter("DEPTNO" IS NOT NULL)
```

실행 계획은 가장 복잡해 보일 수 있으나 DUAL 테이블을 이용하였기 때문에 별도의 I/O가 증가되지는 않는다. ROLLUP을 사용하지 않고도 Cartesian Join을 이용하여 결과를 얻을 수 있다. 한번 계산된 A 집합만을 이용하여 결과를 만들었기 때문에 필요한 테이블의 접근은 최소화되었다. 다만, 복잡한 식의 Grouping이 필요할 경우에는 문장 작성이 용이하지 않다.

```
SQL> DROP INDEX emp_x01 ;
```

> 문제 2.

다음 문장을 최적화 시키시오.

```
SQL> SELECT /*+ index(sales(channel_id)) */ prod_id, SUM(amount_sold)
     FROM sales
     WHERE channel_id IN (2,3,4)
     GROUP BY prod_id ;
SQL> @xplan
```

```
-------------------------------------------------------------------------------------------------------------------------------------
| Id  | Operation                     | Name             | Starts | E-Rows | A-Rows |   A-Time   | Buffers |  OMem |  1Mem | Used-Mem |
-------------------------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT              |                  |      1 |        |     72 |00:00:01.74 |   14223 |       |       |          |
|   1 |  HASH GROUP BY                |                  |      1 |     72 |     72 |00:00:01.74 |   14223 |   798K|   798K| 1269K (0)|
|   2 |   INLIST ITERATOR             |                  |      1 |        |    916K|00:00:01.48 |   14223 |       |       |          |
|   3 |    TABLE ACCESS BY INDEX ROWID| SALES            |      3 |    689K|    916K|00:00:01.11 |   14223 |       |       |          |
|*  4 |     INDEX RANGE SCAN          | SALES_CHANNEL_IX |      3 |    689K|    916K|00:00:00.26 |    1798 |       |       |          |
-------------------------------------------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   4 - access(("CHANNEL_ID"=2 OR "CHANNEL_ID"=3 OR "CHANNEL_ID"=4))
```

〉답안 1. FULL TABLE SCAN 활용

```
SQL〉 SELECT /*+ full(sales) */ prod_id, SUM(amount_sold)
     FROM sales
     WHERE channel_id IN (2,3,4)
     GROUP BY prod_id ;
SQL〉 @xplan
```

```
-------------------------------------------------------------------------------------------------------------------------
| Id  | Operation          | Name  | Starts | E-Rows | A-Rows |   A-Time   | Buffers |  OMem |  1Mem | Used-Mem |
-------------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |       |      1 |        |     72 |00:00:00.52 |    4438 |       |       |          |
|   1 |  HASH GROUP BY     |       |      1 |     72 |     72 |00:00:00.52 |    4438 |   798K|   798K| 1267K (0)|
|*  2 |   TABLE ACCESS FULL| SALES |      1 |    689K|    916K|00:00:00.29 |    4438 |       |       |          |
-------------------------------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   2 - filter(("CHANNEL_ID"=2 OR "CHANNEL_ID"=3 OR "CHANNEL_ID"=4))
```

대량의 데이터를 검색하고 있을 경우 잘못된 인덱스의 사용은 오히려 성능을 저하시킨다. 조건의 범위가 넓을 경우에는 인덱스의 사용 없이 FULL TABLE SCAN으로 접근하는 것이 더 좋을 수도 있다.

〉답안 2. Index 활용

```
SQL〉 CREATE INDEX sales_x01 ON sales(channel_id, prod_id, amount_sold) ;
SQL〉 SELECT prod_id, SUM(amount_sold)
     FROM sales
     WHERE channel_id IN (2,3,4)
     GROUP BY prod_id ;
SQL〉 @xplan
```

```
-------------------------------------------------------------------------------------------------------------------------
| Id  | Operation            | Name      | Starts | E-Rows | A-Rows |   A-Time   | Buffers |  OMem |  1Mem | Used-Mem |
-------------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT     |           |      1 |        |     72 |00:00:00.54 |    2791 |       |       |          |
|   1 |  HASH GROUP BY       |           |      1 |     72 |     72 |00:00:00.54 |    2791 |   798K|   798K| 1274K (0)|
|*  2 |   INDEX FAST FULL SCAN| SALES_X01 |      1 |    689K|    916K|00:00:00.28 |    2791 |       |       |          |
-------------------------------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   2 - filter(("CHANNEL_ID"=2 OR "CHANNEL_ID"=3 OR "CHANNEL_ID"=4))
```

현재의 실행 계획은 "FULL TABLE SCAN"과 속도 차이는 없다. 단, 결합 인덱스를 생성하였기 때문에 테이블의 접근이 없었고, 때문에 I/O의 수는 감속한 것을 확인할 수 있다. 만약 더 많은 데이터가 저장된 테이블에서의 계산을 수행하고 있다면 FULL TABLE SCAN보다는 상대적으로 테이블보다는 크기가 작은 인덱스만 읽고 끝낼 수 있도록 하는 것이 필요하다.

〉답안 3. 병렬 처리

```
SQL〉 SELECT /*+ parallel_index(s,sales_x01,2) */ prod_id, SUM(amount_sold)
     FROM sales s
     WHERE channel_id IN (2,3,4)
     GROUP BY prod_id ;
SQL〉 @xplan
```

```
-----------------------------------------------------------------------------------------------------------------------------------
| Id  | Operation                  | Name      | Starts | E-Rows | A-Rows |   A-Time   | Buffers |  OMem |  1Mem | Used-Mem |
-----------------------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT           |           |      1 |        |     72 |00:00:00.47 |       5 |       |       |          |
|   1 |  PX COORDINATOR            |           |      1 |        |     72 |00:00:00.47 |       5 |       |       |          |
|   2 |   PX SEND QC (RANDOM)      | :TQ10001  |      0 |     72 |      0 |00:00:00.01 |       0 |       |       |          |
|   3 |    HASH GROUP BY           |           |      0 |     72 |      0 |00:00:00.01 |       0 |   798K|   798K| 1233K (0)|
|   4 |     PX RECEIVE             |           |      0 |     72 |      0 |00:00:00.01 |       0 |       |       |          |
|   5 |      PX SEND HASH          | :TQ10000  |      0 |     72 |      0 |00:00:00.01 |       0 |       |       |          |
|   6 |       HASH GROUP BY        |           |      0 |     72 |      0 |00:00:00.01 |       0 |   798K|   798K| 1237K (0)|
|   7 |        PX BLOCK ITERATOR   |           |      0 |    689K|      0 |00:00:00.01 |       0 |       |       |          |
|*  8 |         INDEX FAST FULL SCAN| SALES_X01 |      0 |    689K|      0 |00:00:00.01 |       0 |       |       |          |
-----------------------------------------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   8 - access(:Z>=:Z AND :Z<=:Z)
       filter(("CHANNEL_ID"=2 OR "CHANNEL_ID"=3 OR "CHANNEL_ID"=4))
```

인덱스만 액세스하는 것으로 원하는 성능이 보장되지 않는다면 이미 하나의 프로세스가 처리하기에 데이터의 양이 너무 많아졌을 수 있다. 대용량 데이터를 이용한 작업 시 사용이 가능하다면 병렬 처리 또한 튜닝 방법 중 하나이다. 적절한 병렬도를 설정할 수 없다면 Oracle DB가 자동으로 선정한 병렬도를 사용할 수도 있다. 병렬 처리에 대한 튜닝 방법은 좀 더 추가적인 설명이 필요하다. 상세한 내용은 상위 과정을 참고한다.

〉답안 4. Materialized VIEW 사용

```
SQL〉 conn / as sysdba
SQL〉 GRANT CREATE MATERIALIZED VIEW TO user01 ;
SQL〉 CREATE MATERIALIZED VIEW user01.mv_sales
     ENABLE QUERY REWRITE
     AS SELECT channel_id, prod_id, SUM(amount_sold) AS sum_amt, COUNT(amount_sold) AS cnt_amt
     FROM user01.sales s
     GROUP BY channel_id, prod_id ;
```

병렬 처리로도 원하는 성능이 보장되지 않을 경우 계산된 결과를 저장하는 Materialized View 생성을 고려할 수 있다. 일반적인 View와는 다르게 MView는 계산 결과를 저장하고 있으며, Query Rewrite 기능을 이용하여 하나

이상의 SQL 문을 지원할 수 있다. 단, 생성하려면 적절한 권한이 필요하고, 유지 관리 비용이 추가되기 때문에 우선적으로 고려하기 보다는 DBA와의 협업을 통해 사용을 고려한다.

```
SQL> conn user01/oracle
SQL> SELECT prod_id, SUM(amount_sold)
     FROM sales s
     WHERE channel_id IN (2,3,4)
     GROUP BY prod_id ;
SQL> @xplan
```

```
-----------------------------------------------------------------------------------------------------------------------------
| Id  | Operation                     | Name     | Starts | E-Rows | A-Rows |   A-Time   | Buffers |  OMem |  1Mem | Used-Mem |
-----------------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT              |          |      1 |        |     72 |00:00:00.01 |       3 |       |       |          |
|   1 |  HASH GROUP BY                |          |      1 |    216 |     72 |00:00:00.01 |       3 |   798K|   798K| 1281K (0)|
|*  2 |   MAT_VIEW REWRITE ACCESS FULL| MV_SALES |      1 |    216 |    216 |00:00:00.01 |       3 |       |       |          |
-----------------------------------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   2 - filter(("MV_SALES"."CHANNEL_ID"=2 OR "MV_SALES"."CHANNEL_ID"=3 OR "MV_SALES"."CHANNEL_ID"=4))
```

원본 문장을 실행 했지만 이미 계산된 결과를 가지고 있는 MView를 이용하여 검색하였다. 이때 Query Rewrite된 Query는 다음과 같다.

```
SQL> SELECT prod_id, SUM(sum_amt)
     FROM mv_sales
     WHERE channel_id IN (2,3,4)
     GROUP BY prod_id ;
SQL> @xplan
```

```
-----------------------------------------------------------------------------------------------------------------------------
| Id  | Operation             | Name     | Starts | E-Rows | A-Rows |   A-Time   | Buffers |  OMem |  1Mem | Used-Mem |
-----------------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT      |          |      1 |        |     72 |00:00:00.01 |       3 |       |       |          |
|   1 |  HASH GROUP BY        |          |      1 |    216 |     72 |00:00:00.01 |       3 |   798K|   798K| 1267K (0)|
|*  2 |   MAT_VIEW ACCESS FULL| MV_SALES |      1 |    216 |    216 |00:00:00.01 |       3 |       |       |          |
-----------------------------------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   2 - filter(("CHANNEL_ID"=2 OR "CHANNEL_ID"=3 OR "CHANNEL_ID"=4))
```

MView는 직접적인 조회도 가능하며, 생성할 때 하나 이상의 Query를 지원할 수 있도록 하면 다양한 문장에서 활용될 수 있다. 때문에 어느 특정 SQL 문을 위해서 생성하지 말고 다양한 문장을 지원할 수 있도록 한다. 또한 SALES 테이블의 수정 사항이 있을 경우 MView의 결과는 과거 시점을 갖는다. 때문에 MView는 Refresh 정책을 수립하는 등 추가적인 유지 관리가 필요하다.

```
SQL> SELECT prod_id, SUM(amount_sold)
     FROM sales s
     GROUP BY prod_id ;
SQL> @xplan
```

```
-----------------------------------------------------------------------------------------------------------------------------
| Id  | Operation                     | Name     | Starts | E-Rows | A-Rows |   A-Time   | Buffers |  OMem |  1Mem | Used-Mem |
-----------------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT              |          |      1 |        |     72 |00:00:00.01 |       3 |       |       |          |
|   1 |  HASH GROUP BY                |          |      1 |    228 |     72 |00:00:00.01 |       3 |   798K|   798K| 1289K (0)|
|   2 |   MAT_VIEW REWRITE ACCESS FULL| MV_SALES |      1 |    228 |    228 |00:00:00.01 |       3 |       |       |          |
-----------------------------------------------------------------------------------------------------------------------------
```

```
SQL> SELECT prod_id, SUM(amount_sold), AVG(amount_sold)
     FROM sales s
     GROUP BY prod_id ;
SQL> @xplan
```

```
-----------------------------------------------------------------------------------------------------------------------------
| Id  | Operation                     | Name     | Starts | E-Rows | A-Rows |   A-Time   | Buffers |  OMem |  1Mem | Used-Mem |
-----------------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT              |          |      1 |        |     72 |00:00:00.01 |       3 |       |       |          |
|   1 |  HASH GROUP BY                |          |      1 |    228 |     72 |00:00:00.01 |       3 |   751K|   751K| 1286K (0)|
|   2 |   MAT_VIEW REWRITE ACCESS FULL| MV_SALES |      1 |    228 |    228 |00:00:00.01 |       3 |       |       |          |
-----------------------------------------------------------------------------------------------------------------------------
```

MView를 생성할 때 COUNT된 결과도 저장하고 있기 때문에 AVG 함수의 결과도 검색할 수 있다.

```
SQL> DROP INDEX sales_x01 ;
SQL> DROP MATERIALIZED VIEW mv_sales ;
```

› 문제 3.

다음 문장의 성능을 최적화 시키시오.

```
SQL› SELECT product,
            NVL(SUM(DECODE(channel_id,2,DECODE(quarter,'01',quantity_sold,0))),0) AS "Part_Q1",
            NVL(SUM(DECODE(channel_id,2,DECODE(quarter,'02',quantity_sold,0))),0) AS "Part_Q2",
            NVL(SUM(DECODE(channel_id,2,DECODE(quarter,'03',quantity_sold,0))),0) AS "Part_Q3",
            NVL(SUM(DECODE(channel_id,2,DECODE(quarter,'04',quantity_sold,0))),0) AS "Part_Q4",
            NVL(SUM(DECODE(channel_id,3,DECODE(quarter,'01',quantity_sold,0))),0) AS "Dir_Q1",
            NVL(SUM(DECODE(channel_id,3,DECODE(quarter,'02',quantity_sold,0))),0) AS "Dir_Q2",
            NVL(SUM(DECODE(channel_id,3,DECODE(quarter,'03',quantity_sold,0))),0) AS "Dir_Q3",
            NVL(SUM(DECODE(channel_id,3,DECODE(quarter,'04',quantity_sold,0))),0) AS "Dir_Q4",
            NVL(SUM(DECODE(channel_id,4,DECODE(quarter,'01',quantity_sold,0))),0) AS "Int_Q1",
            NVL(SUM(DECODE(channel_id,4,DECODE(quarter,'02',quantity_sold,0))),0) AS "Int_Q2",
            NVL(SUM(DECODE(channel_id,4,DECODE(quarter,'03',quantity_sold,0))),0) AS "Int_Q3",
            NVL(SUM(DECODE(channel_id,4,DECODE(quarter,'04',quantity_sold,0))),0) AS "Int_Q4",
            NVL(SUM(DECODE(channel_id,9,DECODE(quarter,'01',quantity_sold,0))),0) AS "Tel_Q1",
            NVL(SUM(DECODE(channel_id,9,DECODE(quarter,'02',quantity_sold,0))),0) AS "Tel_Q2",
            NVL(SUM(DECODE(channel_id,9,DECODE(quarter,'03',quantity_sold,0))),0) AS "Tel_Q3",
            NVL(SUM(DECODE(channel_id,9,DECODE(quarter,'04',quantity_sold,0))),0) AS "Tel_Q4"
     FROM (SELECT p.prod_name AS product, s.channel_id, s.quantity_sold,
                  LPAD(TO_CHAR(s.time_id,'Q'),2,'0') AS quarter
           FROM products p, sales s
           WHERE p.prod_id = s.prod_id)
     GROUP BY product ;
SQL› @xplan
```

Id	Operation	Name	Starts	E-Rows	A-Rows	A-Time	Buffers	OMem	1Mem	Used-Mem
0	SELECT STATEMENT		1		71	00:00:02.39	4441			
1	HASH GROUP BY		1	71	71	00:00:02.39	4441	685K	685K	1233K (0)
* 2	HASH JOIN		1	72	72	00:00:02.38	4441	791K	791K	1222K (0)
3	TABLE ACCESS FULL	PRODUCTS	1	72	72	00:00:00.01	3			
4	VIEW	VW_GBC_5	1	72	72	00:00:02.38	4438			
5	HASH GROUP BY		1	72	72	00:00:02.38	4438	686K	686K	1272K (0)
6	TABLE ACCESS FULL	SALES	1	918K	918K	00:00:00.24	4438			

```
Predicate Information (identified by operation id):
---------------------------------------------------
   2 - access("P"."PROD_ID"="ITEM_1")
```

〉답안 1. 인덱스 활용

```
SQL〉 CREATE INDEX sales_x01 ON sales(prod_id, channel_id, time_id, quantity_sold) ;
SQL〉 SELECT product,
            NVL(SUM(DECODE(channel_id,2,DECODE(quarter,'01',quantity_sold,0))),0) AS "Part_Q1",
            NVL(SUM(DECODE(channel_id,2,DECODE(quarter,'02',quantity_sold,0))),0) AS "Part_Q2",
            NVL(SUM(DECODE(channel_id,2,DECODE(quarter,'03',quantity_sold,0))),0) AS "Part_Q3",
            NVL(SUM(DECODE(channel_id,2,DECODE(quarter,'04',quantity_sold,0))),0) AS "Part_Q4",
            NVL(SUM(DECODE(channel_id,3,DECODE(quarter,'01',quantity_sold,0))),0) AS "Dir_Q1",
            NVL(SUM(DECODE(channel_id,3,DECODE(quarter,'02',quantity_sold,0))),0) AS "Dir_Q2",
            NVL(SUM(DECODE(channel_id,3,DECODE(quarter,'03',quantity_sold,0))),0) AS "Dir_Q3",
            NVL(SUM(DECODE(channel_id,3,DECODE(quarter,'04',quantity_sold,0))),0) AS "Dir_Q4",
            NVL(SUM(DECODE(channel_id,4,DECODE(quarter,'01',quantity_sold,0))),0) AS "Int_Q1",
            NVL(SUM(DECODE(channel_id,4,DECODE(quarter,'02',quantity_sold,0))),0) AS "Int_Q2",
            NVL(SUM(DECODE(channel_id,4,DECODE(quarter,'03',quantity_sold,0))),0) AS "Int_Q3",
            NVL(SUM(DECODE(channel_id,4,DECODE(quarter,'04',quantity_sold,0))),0) AS "Int_Q4",
            NVL(SUM(DECODE(channel_id,9,DECODE(quarter,'01',quantity_sold,0))),0) AS "Tel_Q1",
            NVL(SUM(DECODE(channel_id,9,DECODE(quarter,'02',quantity_sold,0))),0) AS "Tel_Q2",
            NVL(SUM(DECODE(channel_id,9,DECODE(quarter,'03',quantity_sold,0))),0) AS "Tel_Q3",
            NVL(SUM(DECODE(channel_id,9,DECODE(quarter,'04',quantity_sold,0))),0) AS "Tel_Q4"
     FROM (SELECT p.prod_name AS product, s.channel_id, s.quantity_sold,
                  LPAD(TO_CHAR(s.time_id,'Q'),2,'0') AS quarter
           FROM products p, sales s
           WHERE p.prod_id = s.prod_id)
     GROUP BY product ;
SQL〉 @xplan
```

Id	Operation	Name	Starts	E-Rows	A-Rows	A-Time	Buffers	OMem	1Mem	Used-Mem
0	SELECT STATEMENT		1		71	00:00:02.38	3682			
1	HASH GROUP BY		1	71	71	00:00:02.38	3682	685K	685K	1285K (0)
* 2	HASH JOIN		1	72	72	00:00:02.38	3682	791K	791K	1190K (0)
3	TABLE ACCESS FULL	PRODUCTS	1	72	72	00:00:00.01	3			
4	VIEW	VW_GBC_5	1	72	72	00:00:02.38	3679			
5	HASH GROUP BY		1	72	72	00:00:02.38	3679	686K	686K	1285K (0)
6	INDEX FAST FULL SCAN	SALES_X01	1	918K	918K	00:00:00.26	3679			

```
Predicate Information (identified by operation id):
---------------------------------------------------
   2 - access("P"."PROD_ID"="ITEM_1")
```

인덱스만 액세스하였지만 현재 상황에서는 최적화에 도움이 크지는 않다.

> 답안 2. 불필요한 연산 제거

```
SQL> SELECT product,
            NVL(SUM(DECODE(channel_id,2,DECODE(quarter,'01',quantity_sold))),0) AS "Part_Q1",
            NVL(SUM(DECODE(channel_id,2,DECODE(quarter,'02',quantity_sold))),0) AS "Part_Q2",
            NVL(SUM(DECODE(channel_id,2,DECODE(quarter,'03',quantity_sold))),0) AS "Part_Q3",
            NVL(SUM(DECODE(channel_id,2,DECODE(quarter,'04',quantity_sold))),0) AS "Part_Q4",
            NVL(SUM(DECODE(channel_id,3,DECODE(quarter,'01',quantity_sold))),0) AS "Dir_Q1",
            NVL(SUM(DECODE(channel_id,3,DECODE(quarter,'02',quantity_sold))),0) AS "Dir_Q2",
            NVL(SUM(DECODE(channel_id,3,DECODE(quarter,'03',quantity_sold))),0) AS "Dir_Q3",
            NVL(SUM(DECODE(channel_id,3,DECODE(quarter,'04',quantity_sold))),0) AS "Dir_Q4",
            NVL(SUM(DECODE(channel_id,4,DECODE(quarter,'01',quantity_sold))),0) AS "Int_Q1",
            NVL(SUM(DECODE(channel_id,4,DECODE(quarter,'02',quantity_sold))),0) AS "Int_Q2",
            NVL(SUM(DECODE(channel_id,4,DECODE(quarter,'03',quantity_sold))),0) AS "Int_Q3",
            NVL(SUM(DECODE(channel_id,4,DECODE(quarter,'04',quantity_sold))),0) AS "Int_Q4",
            NVL(SUM(DECODE(channel_id,9,DECODE(quarter,'01',quantity_sold))),0) AS "Tel_Q1",
            NVL(SUM(DECODE(channel_id,9,DECODE(quarter,'02',quantity_sold))),0) AS "Tel_Q2",
            NVL(SUM(DECODE(channel_id,9,DECODE(quarter,'03',quantity_sold))),0) AS "Tel_Q3",
            NVL(SUM(DECODE(channel_id,9,DECODE(quarter,'04',quantity_sold))),0) AS "Tel_Q4"
     FROM (SELECT p.prod_name AS product, s.channel_id, s.quantity_sold,
                  LPAD(TO_CHAR(s.time_id,'Q'),2,'0') AS quarter
           FROM products p, sales s
           WHERE p.prod_id = s.prod_id)
     GROUP BY product ;
SQL> @xplan
```

Id	Operation	Name	Starts	E-Rows	A-Rows	A-Time	Buffers	OMem	1Mem	Used-Mem
0	SELECT STATEMENT		1		71	00:00:02.27	3682			
1	HASH GROUP BY		1	71	71	00:00:02.27	3682	685K	685K	1277K (0)
* 2	HASH JOIN		1	72	72	00:00:02.27	3682	791K	791K	1240K (0)
3	TABLE ACCESS FULL	PRODUCTS	1	72	72	00:00:00.01	3			
4	VIEW	VW_GBC_5	1	72	72	00:00:02.27	3679			
5	HASH GROUP BY		1	72	72	00:00:02.27	3679	686K	686K	1265K (0)
6	INDEX FAST FULL SCAN	SALES_X01	1	918K	918K	00:00:00.24	3679			

```
Predicate Information (identified by operation id):
---------------------------------------------------
   2 - access("P"."PROD_ID"="ITEM_1")
```

DECODE 함수에서 불필요한 ELSE 절의 '0'을 제거하였다. DECODE 함수 사용 시 ELSE 절이 필요한 경우도 있지만 현재 상황은 불필요한 계산 작업의 증가로 성능만 저하된다.

> 답안 3. DECODE 함수를 CASE 식으로 변경

```
SQL> SELECT product,
         NVL(SUM(CASE WHEN channel_id = 2 AND quarter = '01' THEN quantity_sold END),0) AS "Part_Q1",
         NVL(SUM(CASE WHEN channel_id = 2 AND quarter = '02' THEN quantity_sold END),0) AS "Part_Q2",
         NVL(SUM(CASE WHEN channel_id = 2 AND quarter = '03' THEN quantity_sold END),0) AS "Part_Q3",
         NVL(SUM(CASE WHEN channel_id = 2 AND quarter = '04' THEN quantity_sold END),0) AS "Part_Q4",
         NVL(SUM(CASE WHEN channel_id = 3 AND quarter = '01' THEN quantity_sold END),0) AS "Dir_Q1",
         NVL(SUM(CASE WHEN channel_id = 3 AND quarter = '02' THEN quantity_sold END),0) AS "Dir_Q2",
         NVL(SUM(CASE WHEN channel_id = 3 AND quarter = '03' THEN quantity_sold END),0) AS "Dir_Q3",
         NVL(SUM(CASE WHEN channel_id = 3 AND quarter = '04' THEN quantity_sold END),0) AS "Dir_Q4",
         NVL(SUM(CASE WHEN channel_id = 4 AND quarter = '01' THEN quantity_sold END),0) AS "Int_Q1",
         NVL(SUM(CASE WHEN channel_id = 4 AND quarter = '02' THEN quantity_sold END),0) AS "Int_Q2",
         NVL(SUM(CASE WHEN channel_id = 4 AND quarter = '03' THEN quantity_sold END),0) AS "Int_Q3",
         NVL(SUM(CASE WHEN channel_id = 4 AND quarter = '04' THEN quantity_sold END),0) AS "Int_Q4",
         NVL(SUM(CASE WHEN channel_id = 9 AND quarter = '01' THEN quantity_sold END),0) AS "Tel_Q1",
         NVL(SUM(CASE WHEN channel_id = 9 AND quarter = '02' THEN quantity_sold END),0) AS "Tel_Q2",
         NVL(SUM(CASE WHEN channel_id = 9 AND quarter = '03' THEN quantity_sold END),0) AS "Tel_Q3",
         NVL(SUM(CASE WHEN channel_id = 9 AND quarter = '04' THEN quantity_sold END),0) AS "Tel_Q4"
     FROM (SELECT p.prod_name AS product, s.channel_id, s.quantity_sold,
                  LPAD(TO_CHAR(s.time_id,'Q'),2,'0') AS quarter
           FROM products p, sales s
           WHERE p.prod_id = s.prod_id)
     GROUP BY product ;
SQL> @xplan
```

Id	Operation	Name	Starts	E-Rows	A-Rows	A-Time	Buffers	OMem	1Mem	Used-Mem
0	SELECT STATEMENT		1		71	00:00:02.34	3682			
1	HASH GROUP BY		1	71	71	00:00:02.34	3682	685K	685K	1264K (0)
* 2	HASH JOIN		1	72	72	00:00:02.34	3682	791K	791K	1232K (0)
3	TABLE ACCESS FULL	PRODUCTS	1	72	72	00:00:00.01	3			
4	VIEW	VW_GBC_5	1	72	72	00:00:02.34	3679			
5	HASH GROUP BY		1	72	72	00:00:02.34	3679	686K	686K	1286K (0)
6	INDEX FAST FULL SCAN	SALES_X01	1	918K	918K	00:00:00.26	3679			

```
Predicate Information (identified by operation id):
---------------------------------------------------
   2 - access("P"."PROD_ID"="ITEM_1")
```

일반적으로 CASE 문보다 DECODE의 성능이 좋으나 DECODE 가 중첩되거나 복잡한 표현식이 사용되고 있다면 CASE 문이 더 효율적인 경우도 있다. 현재 상황은 효과가 없다.

> 답안 4. 병렬 처리

```
SQL> SELECT /*+ parallel */ product,
            NVL(SUM(DECODE(channel_id,2,DECODE(quarter,'01',quantity_sold))),0) AS "Part_Q1",
            NVL(SUM(DECODE(channel_id,2,DECODE(quarter,'02',quantity_sold))),0) AS "Part_Q2",
            NVL(SUM(DECODE(channel_id,2,DECODE(quarter,'03',quantity_sold))),0) AS "Part_Q3",
            NVL(SUM(DECODE(channel_id,2,DECODE(quarter,'04',quantity_sold))),0) AS "Part_Q4",
            NVL(SUM(DECODE(channel_id,3,DECODE(quarter,'01',quantity_sold))),0) AS "Dir_Q1",
            NVL(SUM(DECODE(channel_id,3,DECODE(quarter,'02',quantity_sold))),0) AS "Dir_Q2",
            NVL(SUM(DECODE(channel_id,3,DECODE(quarter,'03',quantity_sold))),0) AS "Dir_Q3",
            NVL(SUM(DECODE(channel_id,3,DECODE(quarter,'04',quantity_sold))),0) AS "Dir_Q4",
            NVL(SUM(DECODE(channel_id,4,DECODE(quarter,'01',quantity_sold))),0) AS "Int_Q1",
            NVL(SUM(DECODE(channel_id,4,DECODE(quarter,'02',quantity_sold))),0) AS "Int_Q2",
            NVL(SUM(DECODE(channel_id,4,DECODE(quarter,'03',quantity_sold))),0) AS "Int_Q3",
            NVL(SUM(DECODE(channel_id,4,DECODE(quarter,'04',quantity_sold))),0) AS "Int_Q4",
            NVL(SUM(DECODE(channel_id,9,DECODE(quarter,'01',quantity_sold))),0) AS "Tel_Q1",
            NVL(SUM(DECODE(channel_id,9,DECODE(quarter,'02',quantity_sold))),0) AS "Tel_Q2",
            NVL(SUM(DECODE(channel_id,9,DECODE(quarter,'03',quantity_sold))),0) AS "Tel_Q3",
            NVL(SUM(DECODE(channel_id,9,DECODE(quarter,'04',quantity_sold))),0) AS "Tel_Q4"
     FROM (SELECT p.prod_name AS product, s.channel_id, s.quantity_sold,
                  LPAD(TO_CHAR(s.time_id,'Q'),2,'0') AS quarter
           FROM products p, sales s
           WHERE p.prod_id = s.prod_id)
     GROUP BY product ;
SQL> @iostat
```

Id	Operation	Name	Starts	E-Rows	A-Rows	A-Time	Buffers
0	SELECT STATEMENT		1		71	00:00:01.37	8
1	PX COORDINATOR		1		71	00:00:01.37	8
2	PX SEND QC (RANDOM)	:TQ10004	0	71	0	00:00:00.01	0
3	HASH GROUP BY		0	71	0	00:00:00.01	0
4	PX RECEIVE		0	72	0	00:00:00.01	0
5	PX SEND HASH	:TQ10003	0	72	0	00:00:00.01	0
* 6	HASH JOIN BUFFERED		0	72	0	00:00:00.01	0
7	PX RECEIVE		0	72	0	00:00:00.01	0
8	PX SEND HASH	:TQ10001	0	72	0	00:00:00.01	0
9	VIEW	VW_GBC_5	0	72	0	00:00:00.01	0
10	HASH GROUP BY		0	72	0	00:00:00.01	0
11	PX RECEIVE		0	72	0	00:00:00.01	0
12	PX SEND HASH	:TQ10000	0	72	0	00:00:00.01	0

...

> 답안 5. Inline View 활용하여 함수의 호출 작업 최소화

```
SQL> SELECT product,
            NVL(SUM(DECODE(channel_id,2,DECODE(quarter,'01',qty_sum))),0) AS "Part_Q1",
            NVL(SUM(DECODE(channel_id,2,DECODE(quarter,'02',qty_sum))),0) AS "Part_Q2",
            NVL(SUM(DECODE(channel_id,2,DECODE(quarter,'03',qty_sum))),0) AS "Part_Q3",
            NVL(SUM(DECODE(channel_id,2,DECODE(quarter,'04',qty_sum))),0) AS "Part_Q4",
            NVL(SUM(DECODE(channel_id,3,DECODE(quarter,'01',qty_sum))),0) AS "Dir_Q1",
            NVL(SUM(DECODE(channel_id,3,DECODE(quarter,'02',qty_sum))),0) AS "Dir_Q2",
            NVL(SUM(DECODE(channel_id,3,DECODE(quarter,'03',qty_sum))),0) AS "Dir_Q3",
            NVL(SUM(DECODE(channel_id,3,DECODE(quarter,'04',qty_sum))),0) AS "Dir_Q4",
            NVL(SUM(DECODE(channel_id,4,DECODE(quarter,'01',qty_sum))),0) AS "Int_Q1",
            NVL(SUM(DECODE(channel_id,4,DECODE(quarter,'02',qty_sum))),0) AS "Int_Q2",
            NVL(SUM(DECODE(channel_id,4,DECODE(quarter,'03',qty_sum))),0) AS "Int_Q3",
            NVL(SUM(DECODE(channel_id,4,DECODE(quarter,'04',qty_sum))),0) AS "Int_Q4",
            NVL(SUM(DECODE(channel_id,9,DECODE(quarter,'01',qty_sum))),0) AS "Tel_Q1",
            NVL(SUM(DECODE(channel_id,9,DECODE(quarter,'02',qty_sum))),0) AS "Tel_Q2",
            NVL(SUM(DECODE(channel_id,9,DECODE(quarter,'03',qty_sum))),0) AS "Tel_Q3",
            NVL(SUM(DECODE(channel_id,9,DECODE(quarter,'04',qty_sum))),0) AS "Tel_Q4"
     FROM (SELECT p.prod_name AS product, s.channel_id, s.quarter, s.qty_sum
           FROM products p, (SELECT prod_id, channel_id,
                                    LPAD(TO_CHAR(time_id,'Q'),2,'0') AS quarter,
                                    SUM(quantity_sold) AS qty_sum
                             FROM sales
                             GROUP BY prod_id, channel_id, LPAD(TO_CHAR(time_id,'Q'),2,'0')) s
           WHERE p.prod_id = s.prod_id ) GROUP BY product ;
SQL> @xplan
```

Id	Operation	Name	Starts	E-Rows	A-Rows	A-Time	Buffers	OMem	1Mem	Used-Mem
0	SELECT STATEMENT		1		71	00:00:00.85	3682			
1	HASH GROUP BY		1	71	71	00:00:00.85	3682	685K	685K	1287K (0)
* 2	HASH JOIN		1	72	72	00:00:00.85	3682	791K	791K	1232K (0)
3	TABLE ACCESS FULL	PRODUCTS	1	72	72	00:00:00.01	3			
4	VIEW	VW_GBC_5	1	72	72	00:00:00.85	3679			
5	HASH GROUP BY		1	72	72	00:00:00.85	3679	686K	686K	1236K (0)
6	VIEW		1	895	895	00:00:00.85	3679			
7	HASH GROUP BY		1	895	895	00:00:00.85	3679	776K	776K	1282K (0)
8	INDEX FAST FULL SCAN	SALES_X01	1	918K	918K	00:00:00.27	3679			

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

   2 - access("P"."PROD_ID"="ITEM_1")
```

> 답안 6. PIVOT 문 사용

```
SQL> ALTER SESSION set "_pivot_implementation_method" = pivot2 ;
SQL> SELECT product,
            NVL("Part_Q1_SUM",0)  AS "Part_Q1",  NVL("Part_Q2_SUM",0)  AS "Part_Q2",
            NVL("Part_Q3_SUM",0)  AS "Part_Q3",  NVL("Part_Q4_SUM",0)  AS "Part_Q4",
            NVL("Dir_Q1_SUM",0)   AS "Dir_Q1",   NVL("Dir_Q2_SUM",0)   AS "Dir_Q2",
            NVL("Dir_Q3_SUM",0)   AS "Dir_Q3",   NVL("Dir_Q4_SUM",0)   AS "Dir_Q4",
            NVL("Int_Q1_SUM",0)   AS "Int_Q1",   NVL("Int_Q2_SUM",0)   AS "Int_Q2",
            NVL("Int_Q3_SUM",0)   AS "Int_Q3",   NVL("Int_Q4_SUM",0)   AS "Int_Q4",
            NVL("Tel_Q1_SUM",0)   AS "Tel_Q1",   NVL("Tel_Q2_SUM",0)   AS "Tel_Q2",
            NVL("Tel_Q3_SUM",0)   AS "Tel_Q3",   NVL("Tel_Q4_SUM",0)   AS "Tel_Q4"
    FROM (SELECT p.prod_name AS product, s.channel_id, s.quantity_sold,
                 LPAD(TO_CHAR(s.time_id,'Q'),2,'0') AS quarter
          FROM products p, sales s WHERE p.prod_id = s.prod_id)
    PIVOT (SUM(quantity_sold) AS "SUM"
           FOR (channel_id, quarter) IN ((2,'01') AS "Part_Q1", (2,'02') AS "Part_Q2",
                                         (2,'03') AS "Part_Q3", (2,'04') AS "Part_Q4",
                                         (3,'01') AS "Dir_Q1",  (3,'02') AS "Dir_Q2",
                                         (3,'03') AS "Dir_Q3",  (3,'04') AS "Dir_Q4",
                                         (4,'01') AS "Int_Q1",  (4,'02') AS "Int_Q2",
                                         (4,'03') AS "Int_Q3",  (4,'04') AS "Int_Q4",
                                         (9,'01') AS "Tel_Q1",  (9,'02') AS "Tel_Q2",
                                         (9,'03') AS "Tel_Q3",  (9,'04') AS "Tel_Q4")) ;
SQL> @xplan
```

```
-------------------------------------------------------------------------------------------------------------------------------
| Id  | Operation                | Name      | Starts | E-Rows | A-Rows |   A-Time   | Buffers |  OMem |  1Mem | Used-Mem |
-------------------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT         |           |      1 |        |     71 |00:00:01.89 |    3682 |       |       |          |
|   1 |  VIEW                    |           |      1 |    207K|     71 |00:00:01.89 |    3682 |       |       |          |
|   2 |   TRANSPOSE              |           |      1 |        |     71 |00:00:01.89 |    3682 |       |       |          |
|   3 |    SORT GROUP BY PIVOT   |           |      1 |    207K|    883 |00:00:01.89 |    3682 | 80896 | 80896 |71680  (0)|
|*  4 |     HASH JOIN            |           |      1 |    918K|    918K|00:00:01.09 |    3682 |   794K|   794K| 1179K (0)|
|   5 |      TABLE ACCESS FULL   | PRODUCTS  |      1 |     72 |     72 |00:00:00.01 |       3 |       |       |          |
|   6 |      INDEX FAST FULL SCAN| SALES_X01 |      1 |    918K|    918K|00:00:00.21 |    3679 |       |       |          |
-------------------------------------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   4 - access("P"."PROD_ID"="S"."PROD_ID")
```

복잡한 조건식을 비교할 때 중첩된 DECODE 문을 사용하는 것보다 PIVOT 절을 사용하는 것이 문장의 작성 편의성과 성능을 향상화 시킬 수 있다.

> 답안 7. Inline View를 이용한 PIVOT 문 최적화

```
SQL> SELECT product,
            NVL("Part_Q1_SUM",0)  AS "Part_Q1",  NVL("Part_Q2_SUM",0)  AS "Part_Q2",
            NVL("Part_Q3_SUM",0)  AS "Part_Q3",  NVL("Part_Q4_SUM",0)  AS "Part_Q4",
            NVL("Dir_Q1_SUM",0)   AS "Dir_Q1",   NVL("Dir_Q2_SUM",0)   AS "Dir_Q2",
            NVL("Dir_Q3_SUM",0)   AS "Dir_Q3",   NVL("Dir_Q4_SUM",0)   AS "Dir_Q4",
            NVL("Int_Q1_SUM",0)   AS "Int_Q1",   NVL("Int_Q2_SUM",0)   AS "Int_Q2",
            NVL("Int_Q3_SUM",0)   AS "Int_Q3",   NVL("Int_Q4_SUM",0)   AS "Int_Q4",
            NVL("Tel_Q1_SUM",0)   AS "Tel_Q1",   NVL("Tel_Q2_SUM",0)   AS "Tel_Q2",
            NVL("Tel_Q3_SUM",0)   AS "Tel_Q3",   NVL("Tel_Q4_SUM",0)   AS "Tel_Q4"
    FROM (SELECT p.prod_name AS product, s.channel_id, s.quarter, s.qty_sum
            FROM products p, (SELECT prod_id, channel_id,
                                     LPAD(TO_CHAR(time_id,'Q'),2,'0') AS quarter,
                                     SUM(quantity_sold) AS qty_sum
                              FROM sales
                              GROUP BY prod_id, channel_id, LPAD(TO_CHAR(time_id,'Q'),2,'0')) s
            WHERE p.prod_id = s.prod_id)
    PIVOT (SUM(qty_sum) AS "SUM"
            FOR (channel_id, quarter) IN ((2,'01') AS "Part_Q1", (2,'02') AS "Part_Q2",
                                          (2,'03') AS "Part_Q3", (2,'04') AS "Part_Q4",
                                          (3,'01') AS "Dir_Q1",  (3,'02') AS "Dir_Q2",
                                          (3,'03') AS "Dir_Q3",  (3,'04') AS "Dir_Q4",
                                          (4,'01') AS "Int_Q1",  (4,'02') AS "Int_Q2",
                                          (4,'03') AS "Int_Q3",  (4,'04') AS "Int_Q4",
                                          (9,'01') AS "Tel_Q1",  (9,'02') AS "Tel_Q2",
                                          (9,'03') AS "Tel_Q3",  (9,'04') AS "Tel_Q4")) ;

SQL> @xplan
-------------------------------------------------------------------------------------------------------------------------------
| Id  | Operation                  | Name      | Starts | E-Rows | A-Rows |   A-Time   | Buffers |  OMem |  1Mem | Used-Mem |
-------------------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT           |           |      1 |        |     71 |00:00:00.85 |    3682 |       |       |          |
|   1 |  VIEW                      |           |      1 |    201 |     71 |00:00:00.85 |    3682 |       |       |          |
|   2 |   TRANSPOSE                |           |      1 |        |     71 |00:00:00.85 |    3682 |       |       |          |
|   3 |    SORT GROUP BY PIVOT     |           |      1 |    201 |    883 |00:00:00.85 |    3682 | 80896 | 80896 |71680  (0)|
|*  4 |     HASH JOIN              |           |      1 |    895 |    895 |00:00:00.85 |    3682 |   794K|   794K| 1197K (0)|
|   5 |      TABLE ACCESS FULL     | PRODUCTS  |      1 |     72 |     72 |00:00:00.01 |       3 |       |       |          |
|   6 |      VIEW                  |           |      1 |    895 |    895 |00:00:00.85 |    3679 |       |       |          |
|   7 |       HASH GROUP BY        |           |      1 |    895 |    895 |00:00:00.85 |    3679 |   776K|   776K| 2803K (0)|
|   8 |        INDEX FAST FULL SCAN| SALES_X01 |      1 |    918K|    918K|00:00:00.27 |    3679 |       |       |          |
-------------------------------------------------------------------------------------------------------------------------------

...
SQL> DROP INDEX sales_x01 ;
```

› 문제 4.

소속 부서의 평균 급여보다 더 많은 급여를 받는 사원을 검색하시오.

› 출력 결과

```
     EMPNO ENAME             SAL     DEPTNO
---------- ---------- ---------- ----------
      7839 KING             5000         10
      7566 JONES            2975         20
      7788 SCOTT            3000         20
      7902 FORD             3000         20
      7499 ALLEN            1600         30
      7698 BLAKE            2850         30

6 rows selected.
```

> 답안 1. Correlated Subquery, Inline View 사용

```
SQL> SELECT empno, ename, sal, deptno
     FROM emp e
     WHERE sal > ( SELECT 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	6			
1	MERGE JOIN		1	1	6	00:00:00.01	6			
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	2			
* 5	SORT JOIN		14	3	14	00:00:00.01	2	2048	2048	2048 (0)
6	VIEW	VW_SQ_1	1	3	3	00:00:00.01	2			
7	SORT GROUP BY		1	3	3	00:00:00.01	2	2048	2048	2048 (0)
8	TABLE ACCESS FULL	EMP	1	14	14	00:00:00.01	2			

```
Predicate Information (identified by operation id):
---------------------------------------------------
   4 - filter("SAL">"AVG(SAL)")
   5 - access("ITEM_1"="E"."DEPTNO")
       filter("ITEM_1"="E"."DEPTNO")
```

```
SQL> SELECT e.empno, e.ename, e.sal, e.deptno
     FROM emp e, (SELECT deptno, AVG(sal) AS avg
                  FROM emp
                  GROUP BY deptno) a
     WHERE e.deptno = a.deptno
       AND e.sal > a.avg ;
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	6			
1	MERGE JOIN		1	1	6	00:00:00.01	6			
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	2			
* 5	SORT JOIN		14	3	14	00:00:00.01	2	2048	2048	2048 (0)
6	VIEW		1	3	3	00:00:00.01	2			
7	SORT GROUP BY		1	3	3	00:00:00.01	2	2048	2048	2048 (0)
8	TABLE ACCESS FULL	EMP	1	14	14	00:00:00.01	2			

```
Predicate Information (identified by operation id):
---------------------------------------------------
   4 - filter("E"."SAL">"A"."AVG")
   5 - access("E"."DEPTNO"="A"."DEPTNO")
       filter("E"."DEPTNO"="A"."DEPTNO")
```

두 문장 모두 동일한 결과를 검색한다. 또한 실행 계획까지 동일할 수 있다. Correlated Subquery가 Unnesting 되면서 조인의 문장으로 처리되었다. 단, Inline View의 형식은 필요에 따라 평균 급여도 함께 출력이 가능하다. 하지만 Correlated Subquery 문장은 평균 급여를 화면에 출력할 수는 없다. 때문에 경우에 따라 두 문장 중 출력해야 할 결과가 무엇인지를 먼저 파악하고 적절한 선택을 해야 한다. 당연히 필요에 따라서 힌트를 이용하면 조인의 종류나 순서 등을 변경할 수 있다.

추가적으로 확인할 부분은 Correlated Subquery를 이용한 First Rows 환경에서의 최적화 작업이다.

```
SQL> SELECT empno, ename, sal, deptno
     FROM emp e
     WHERE 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 |       8 |
|*  1 |  FILTER                       |               |      1 |        |      6 |00:00:00.01 |       8 |
|   2 |   TABLE ACCESS FULL           | EMP           |      1 |     14 |     14 |00:00:00.01 |       3 |
|   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("SAL">)
   5 - access("DEPTNO"=:B1)
```

위의 서브 쿼리는 결과(평균 급여)를 함께 출력할 수 없다는 특징이 있다. 즉, 내부적으로 필터링을 하기 위해서만 사용된다. 때문에 Inline View 형식을 더 선호할 수도 있다. 하지만 위와 같이 Unnesting을 수행하지 못하게 한다면 조건에 만족하는 첫 번째 행을 가장 빠르게 검색할 수 있는 문장이기도 하다. Correlated Subquery의 사용은 First Rows 환경에서 최적의 성능을 보장할 수 있다.

만약 Inline View 형식의 문장을 힌트를 통해 다음과 같이 변경한다면?

```
SQL> SELECT /*+ leading(e) use_nl(a) */ e.empno, e.ename, e.sal, e.deptno
     FROM emp e, (SELECT /*+ no_merge */ deptno, AVG(sal) AS avg
                  FROM emp
                  GROUP BY deptno) a
     WHERE e.deptno = a.deptno
       AND e.sal > a.avg ;
SQL> @xplan
```

```
-----------------------------------------------------------------------------------------------------------------
| Id  | Operation                      | Name          | Starts | E-Rows | A-Rows |   A-Time   | Buffers |
-----------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT               |               |      1 |        |      6 |00:00:00.01 |       9 |
|   1 |  NESTED LOOPS                  |               |      1 |      1 |      6 |00:00:00.01 |       9 |
|   2 |   TABLE ACCESS FULL            | EMP           |      1 |     14 |     14 |00:00:00.01 |       3 |
|*  3 |   VIEW PUSHED PREDICATE        |               |     14 |      1 |      6 |00:00:00.01 |       6 |
|*  4 |    FILTER                      |               |     14 |        |     14 |00:00:00.01 |       6 |
|   5 |     SORT AGGREGATE             |               |     14 |      1 |     14 |00:00:00.01 |       6 |
|   6 |      TABLE ACCESS BY INDEX ROWID| EMP           |     14 |      5 |     70 |00:00:00.01 |       6 |
|*  7 |       INDEX RANGE SCAN         | EMP_DEPTNO_IX |     14 |      5 |     70 |00:00:00.01 |       4 |
-----------------------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   3 - filter("E"."SAL">"A"."AVG")
   4 - filter(COUNT(*)>0)
   7 - access("DEPTNO"="E"."DEPTNO")
```

생성된 실행 계획은 Correlated Subquery와 유사하기는 하지만 성능상의 이점은 앞선 실행 계획에 더 있다. 위의 문장은 GROUP BY를 사용한 서브 쿼리가 존재하기 때문에 Nested Loops Join을 수행하려고 해도 제약이 뒤따른다. 한가지 다행인 부분은 Grouping을 수행하는 컬럼이 조인 조건에 존재하기 때문에 조건식이 서브 쿼리에 Push Down 되었다. 때문에 특정 부서의 평균 급여만 검색이 가능했지만, 후보 행이 14개이므로 14번의 반복 접근이 수행되었다. 앞서 확인한 Correlated Subquery의 실행 계획은 중복되는 부서 번호의 평균 급여는 문장이 종료될 때까지 캐싱 되기 때문에 평균 급여의 탐색은 3번만 수행되었다.

동일한 결과를 검색하더라도 현재 검색하고자 하는 결과가 무엇인지 파악하고, 상황에 만족하는 문장의 형식과 실행 계획의 선택 유/무를 확인해야 한다.

> 답안 2. Analytic Function 사용

```
SQL> SELECT empno, ename, sal, deptno
     FROM ( SELECT empno, ename, sal, deptno,
                   AVG(sal) OVER(PARTITION BY deptno) AS avg
            FROM emp )
     WHERE sal > avg ;
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 |       2 |       |       |          |
|*  1 |  VIEW               |      |      1 |     14 |      6 |00:00:00.01 |       2 |       |       |          |
|   2 |   WINDOW SORT       |      |      1 |     14 |     14 |00:00:00.01 |       2 |  2048 |  2048 | 2048  (0)|
|   3 |    TABLE ACCESS FULL| EMP  |      1 |     14 |     14 |00:00:00.01 |       2 |       |       |          |
-------------------------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   1 - filter("SAL">"AVG")
```

분석 함수 또는 Window 함수의 사용은 동일 집합의 반복 접근을 제거하는데 매우 큰 역할을 한다. 생성된 실행 계획 역시 EMP 테이블을 한 번만 액세스하고 처리된 결과를 검색해 준다. 물론 추가적인 메모리를 사용할 수 있고, All Rows 상황에 최적화된 실행 계획일 수 있다. 하지만 대량의 데이터를 갖는 테이블의 또는 인덱스의 반복 액세스를 제거한다면 그 역시 성능상 매우 큰 이점이 존재할 것이다.

분석 함수의 사용이 항상 정답은 아니다. 경우에 따라 일반적인 Subquery를 이용하는 것이 필요할 수도 있다. 단, 다양한 문장을 통해 동일한 결과를 검색하도록 문장을 작성할 수 있다면 경우에 따라 필요한 문장의 선택을 보다 손쉽게 할 수 있다.

앞서 확인한 모든 문장을 위해서 (DEPTNO, SAL) 컬럼을 함께 저장하고 있는 결합 인덱스의 생성은 각 문장의 성능을 더 최적화 시키는데 도움을 준다.

› 문제 5.

EMP 테이블에서 사원들의 정보를 EMPNO 컬럼으로 정렬하고, 각 사원의 급여를 행 별로 누적하여 TOTAL 컬럼을 검색하시오.

› 출력 결과

```
     EMPNO ENAME             SAL      TOTAL
---------- ---------- ---------- ----------
      7369 SMITH             800        800
      7499 ALLEN            1600       2400
      7521 WARD             1250       3650
      7566 JONES            2975       6625
      7654 MARTIN           1250       7875
      7698 BLAKE            2850      10725
      7782 CLARK            2450      13175
      7788 SCOTT            3000      16175
      7839 KING             5000      21175
      7844 TURNER           1500      22675
      7876 ADAMS            1100      23775
      7900 JAMES             950      24725
      7902 FORD             3000      27725
      7934 MILLER           1300      29025

14 rows selected.
```

> 답안 1. Self Join, Correlated Subquery 이용

```
SQL> SELECT a.empno, a.ename, a.sal , SUM(b.sal) AS TOTAL
     FROM emp a , emp b
     WHERE a.empno >= b.empno
     GROUP BY a.empno, a.ename, a.sal
     ORDER BY a.empno ;
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 |       4 |       |       |          |
|   1 |  SORT GROUP BY                |        |      1 |     14 |     14 |00:00:00.01 |       4 |  2048 |  2048 | 2048  (0)|
|   2 |   MERGE JOIN                  |        |      1 |    105 |    105 |00:00:00.01 |       4 |       |       |          |
|   3 |    SORT JOIN                  |        |      1 |     14 |     14 |00:00:00.01 |       2 |  2048 |  2048 | 2048  (0)|
|   4 |     TABLE ACCESS BY INDEX ROWID| EMP    |      1 |     14 |     14 |00:00:00.01 |       2 |       |       |          |
|   5 |      INDEX FULL SCAN          | PK_EMP |      1 |     14 |     14 |00:00:00.01 |       1 |       |       |          |
|*  6 |    SORT JOIN                  |        |     14 |     14 |    105 |00:00:00.01 |       2 |  2048 |  2048 | 2048  (0)|
|   7 |     TABLE ACCESS FULL         | EMP    |      1 |     14 |     14 |00:00:00.01 |       2 |       |       |          |
-------------------------------------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   6 - access(INTERNAL_FUNCTION("A"."EMPNO")>=INTERNAL_FUNCTION("B"."EMPNO"))
       filter(INTERNAL_FUNCTION("A"."EMPNO")>=INTERNAL_FUNCTION("B"."EMPNO"))
```

```
SQL> SELECT a.empno, a.ename, a.sal, (SELECT SUM(sal)
                                      FROM emp
                                      WHERE empno <= a.empno) AS TOTAL
     FROM emp a
     ORDER BY a.empno ;
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               |        |     14 |      1 |     14 |00:00:00.01 |       5 |
|   2 |   TABLE ACCESS BY INDEX ROWID| EMP    |     14 |      1 |    105 |00:00:00.01 |       5 |
|*  3 |    INDEX RANGE SCAN           | PK_EMP |     14 |      1 |    105 |00:00:00.01 |       3 |
|   4 |  TABLE ACCESS BY INDEX ROWID | EMP    |      1 |     14 |     14 |00:00:00.01 |       4 |
|   5 |   INDEX FULL SCAN             | PK_EMP |      1 |     14 |     14 |00:00:00.01 |       2 |
-----------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   3 - access("EMPNO"<=:B1)
```

> 답안 2. 분석 함수 이용

```
SQL> SELECT empno, ename, sal, SUM(sal) OVER (ORDER BY empno) AS TOTAL
       FROM emp ;
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	2			
1	WINDOW BUFFER		1	14	14	00:00:00.01	2	2048	2048	2048 (0)
2	TABLE ACCESS BY INDEX ROWID	EMP	1	14	14	00:00:00.01	2			
3	INDEX FULL SCAN	PK_EMP	1	14	14	00:00:00.01	1			

JOIN, Subquery 등을 이용하여도 결과는 생성 가능하지만 동일 집합의 반복 접근은 I/O를 증가시키며 성능을 악화 시킬 수 있다. 필요에 따라 Join, Subquery를 이용해야 할 수도 있지만 분석 함수의 사용을 고려한다.

> 문제 6.

EMP 테이블에서 HIREDATE, EMPNO 컬럼으로 정렬하여 다음과 같이 사원 정보를 검색한다.
이때 해당 사원보다 위, 아래(앞, 뒤)의 입사 일자를 함께 출력하시오.

> 출력 결과

EMPNO	ENAME	HIREDATE	PREV_HIRE	NEXT_HIRE
7369	SMITH	1980/12/17		1981/02/20
7499	ALLEN	**1981/02/20**	1980/12/17	1981/02/22
7521	**WARD**	**1981/02/22**	**1981/02/20**	**1981/04/02**
7566	JONES	**1981/04/02**	1981/02/22	1981/05/01
7698	BLAKE	1981/05/01	1981/04/02	1981/06/09
7782	CLARK	1981/06/09	1981/05/01	1981/09/08
7844	TURNER	1981/09/08	1981/06/09	1981/09/28
7654	MARTIN	1981/09/28	1981/09/08	1981/11/17
7839	KING	1981/11/17	1981/09/28	1981/12/03
7900	JAMES	1981/12/03	1981/11/17	1981/12/03
7902	FORD	1981/12/03	1981/12/03	1982/01/23
7934	MILLER	1982/01/23	1981/12/03	1987/04/19
7788	SCOTT	1987/04/19	1982/01/23	1987/05/23
7876	ADAMS	1987/05/23	1987/04/19	

14 rows selected.

> 답안 1. ROWNUM을 이용한 Outer Join 사용

```
SQL> SELECT a.empno, a.ename, a.hiredate,b.hiredate AS PREV_HIRE,c.hiredate AS NEXT_HIRE
     FROM (SELECT ROWNUM no1, empno, ename, hiredate
           FROM (SELECT empno, ename, hiredate
                 FROM emp
                 ORDER BY hiredate,empno)) a,
          (SELECT ROWNUM+1 no2, empno, ename, hiredate
           FROM (SELECT empno, ename, hiredate
                 FROM emp
                 ORDER BY hiredate,empno)) b,
          (SELECT ROWNUM-1 no3, empno, ename, hiredate
           FROM (SELECT empno, ename, hiredate
                 FROM emp
                 ORDER BY hiredate,empno)) c
     WHERE a.no1 = b.no2 (+)
       AND a.no1 = c.no3 (+)
     ORDER BY a.hiredate, a.empno ;
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	6			
1	SORT ORDER BY		1	14	14	00:00:00.01	6	2048	2048	2048 (0)
* 2	HASH JOIN OUTER		1	14	14	00:00:00.01	6	788K	788K	1081K (0)
* 3	HASH JOIN OUTER		1	14	14	00:00:00.01	4	821K	821K	1079K (0)
4	VIEW		1	14	14	00:00:00.01	2			
5	COUNT		1		14	00:00:00.01	2			
6	VIEW		1	14	14	00:00:00.01	2			
7	SORT ORDER BY		1	14	14	00:00:00.01	2	2048	2048	2048 (0)
8	TABLE ACCESS FULL	EMP	1	14	14	00:00:00.01	2			
9	VIEW		1	14	14	00:00:00.01	2			
10	COUNT		1		14	00:00:00.01	2			
11	VIEW		1	14	14	00:00:00.01	2			
12	SORT ORDER BY		1	14	14	00:00:00.01	2	2048	2048	2048 (0)
13	TABLE ACCESS FULL	EMP	1	14	14	00:00:00.01	2			
14	VIEW		1	14	14	00:00:00.01	2			
15	COUNT		1		14	00:00:00.01	2			
16	VIEW		1	14	14	00:00:00.01	2			
17	SORT ORDER BY		1	14	14	00:00:00.01	2	2048	2048	2048 (0)
18	TABLE ACCESS FULL	EMP	1	14	14	00:00:00.01	2			

```
Predicate Information (identified by operation id):
---------------------------------------------------
   2 - access("A"."NO1"="C"."NO3")
   3 - access("A"."NO1"="B"."NO2")
```

> 답안 2. 분석 함수 이용

```
SQL> SELECT empno, ename, hiredate,
            LAG(hiredate)  OVER (ORDER BY hiredate,empno) AS PREV_HIRE,
            LEAD(hiredate) OVER (ORDER BY hiredate,empno) AS NEXT_HIRE
     FROM emp ;
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	2			
1	WINDOW SORT		1	14	14	00:00:00.01	2	2048	2048	2048 (0)
2	TABLE ACCESS FULL	EMP	1	14	14	00:00:00.01	2			

› 문제 7.

EMP 테이블에서 부서별 가장 많은 급여를 받는 사원을 한 명씩 검색하시오. 단, 동일한 급여를 받는 사원이 존재할 경우 임의의 한 명을 검색한다.

› 출력 결과

```
     EMPNO ENAME             SAL     DEPTNO
---------- ---------- ---------- ----------
      7839 KING             5000         10
      7788 SCOTT            3000         20
      7698 BLAKE            2850         30
```

> 답안 1.

```
SQL> SELECT empno, ename, sal, deptno
       FROM ( SELECT * FROM emp WHERE deptno = 10 ORDER BY sal DESC )
      WHERE rownum = 1
     UNION ALL
     SELECT empno, ename, sal, deptno
       FROM ( SELECT * FROM emp WHERE deptno = 20 ORDER BY sal DESC )
      WHERE rownum = 1
     UNION ALL
     SELECT empno, ename, sal, deptno
       FROM ( SELECT * FROM emp WHERE deptno = 30 ORDER BY sal DESC )
      WHERE rownum = 1 ;
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	6			
1	UNION-ALL		1		3	00:00:00.01	6			
* 2	COUNT STOPKEY		1		1	00:00:00.01	2			
3	VIEW		1	5	1	00:00:00.01	2			
* 4	SORT ORDER BY STOPKEY		1	5	1	00:00:00.01	2	2048	2048	2048 (0)
5	TABLE ACCESS BY INDEX ROWID	EMP	1	5	3	00:00:00.01	2			
* 6	INDEX RANGE SCAN	EMP_DEPTNO_IX	1	5	3	00:00:00.01	1			
* 7	COUNT STOPKEY		1		1	00:00:00.01	2			
8	VIEW		1	5	1	00:00:00.01	2			
* 9	SORT ORDER BY STOPKEY		1	5	1	00:00:00.01	2	2048	2048	2048 (0)
10	TABLE ACCESS BY INDEX ROWID	EMP	1	5	5	00:00:00.01	2			
* 11	INDEX RANGE SCAN	EMP_DEPTNO_IX	1	5	5	00:00:00.01	1			
* 12	COUNT STOPKEY		1		1	00:00:00.01	2			
13	VIEW		1	5	1	00:00:00.01	2			
* 14	SORT ORDER BY STOPKEY		1	5	1	00:00:00.01	2	2048	2048	2048 (0)
15	TABLE ACCESS BY INDEX ROWID	EMP	1	5	6	00:00:00.01	2			
* 16	INDEX RANGE SCAN	EMP_DEPTNO_IX	1	5	6	00:00:00.01	1			

```
Predicate Information (identified by operation id):
---------------------------------------------------
   2 - filter(ROWNUM=1)
   4 - filter(ROWNUM=1)
   6 - access("DEPTNO"=10)
   7 - filter(ROWNUM=1)
   9 - filter(ROWNUM=1)
  11 - access("DEPTNO"=20)
  12 - filter(ROWNUM=1)
  14 - filter(ROWNUM=1)
  16 - access("DEPTNO"=30)
```

ROWNUM은 특정 컬럼의 값을 기준으로 행 번호를 새로 시작할 수 없다. 때문에 각각의 부서별로 TOP-n 질의 문장을 이용하여 UNION ALL으로 묶어주면 원하는 결과를 검색할 수 있으나, 부서 번호의 개수가 많을 경우 문장이 복잡해질 것이고 성능 역시 저하된다.

현재 요구 조건에는 만족하지 않지만, 부서별 최대 급여를 받는 사원을 검색하는 경우라면 다음과 같은 문장을 이용할 수도 있다.

```
SQL> SELECT empno, ename, sal, deptno
       FROM emp a
      WHERE sal = ( SELECT MAX(sal)
                      FROM emp
                     WHERE deptno = a.deptno ) ;
     EMPNO ENAME             SAL     DEPTNO
---------- ---------- ---------- ----------
      7839 KING             5000         10
      7902 FORD             3000         20
      7698 BLAKE            2850         30
      7788 SCOTT            3000         20
```

> 답안 2.

```
SQL> SELECT empno, ename, sal, deptno
     FROM ( SELECT empno, ename, sal, deptno,
                   ROW_NUMBER() OVER(PARTITION BY deptno ORDER BY sal DESC) AS rank
            FROM emp )
     WHERE rank = 1 ;
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 |       2 |       |       |          |
|*  1 |  VIEW                    |      |      1 |     14 |      3 |00:00:00.01 |       2 |       |       |          |
|*  2 |   WINDOW SORT PUSHED RANK|      |      1 |     14 |      7 |00:00:00.01 |       2 |  2048 |  2048 | 2048  (0)|
|   3 |    TABLE ACCESS FULL     | EMP  |      1 |     14 |     14 |00:00:00.01 |       2 |       |       |          |
-------------------------------------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   1 - filter("RANK"=1)
   2 - filter(ROW_NUMBER() OVER ( PARTITION BY "DEPTNO" ORDER BY INTERNAL_FUNCTION("SAL") DESC )<=1)
```

분석 함수는 PARTITION BY 절을 통해 순위를 초기화할 수 있다. 또한 RANK, DENSE_RANK, ROW_NUMBER 의 함수를 이용하여 원하는 결과를 다양하게 생성할 수 있다. 각각의 함수가 어떠한 결과를 검색하는지 정확히 구분하고 필요한 함수를 사용한다.

```
SQL> SELECT empno, ename, sal,
            ROW_NUMBER() OVER(ORDER BY sal DESC) AS row_num,
            RANK()       OVER(ORDER BY sal DESC) AS rank,
            DENSE_RANK() OVER(ORDER BY sal DESC) AS drank
     FROM emp ;

     EMPNO ENAME             SAL    ROW_NUM       RANK      DRANK
---------- ---------- ---------- ---------- ---------- ----------
      7839 KING             5000          1          1          1
      7902 FORD             3000          2          2          2
      7788 SCOTT            3000          3          2          2
      7566 JONES            2975          4          4          3
      7698 BLAKE            2850          5          5          4
      7782 CLARK            2450          6          6          5
      7499 ALLEN            1600          7          7          6
      7844 TURNER           1500          8          8          7
      7934 MILLER           1300          9          9          8
      7521 WARD             1250         10         10          9
      7654 MARTIN           1250         11         10          9
      7876 ADAMS            1100         12         12         10
      7900 JAMES             950         13         13         11
      7369 SMITH             800         14         14         12

14 rows selected.
```

ROW_NUMBER 함수는 ROWNUM과 비슷하다. 정렬 컬럼의 값이 동일하더라도 행 번호는 증가되며, RANK, DENSE_RANK 함수는 정렬 컬럼의 값이 동일할 경우 동일한 순위를 할당한다. 단, 동일 순위가 만들어졌을 때 다음 순위를 정의하는 기준이 서로 다르다. 또한 3개의 함수 모두 PARTITION BY 절을 이용하여 원하는 컬럼을 기준으로 순위의 초기화가 가능하다.

DML Tuning

〉문제 1.

EMP 테이블에서 1982년 이전에 입사한 사원 정보를 SAL_HIST 테이블에 입력하고, 1982년 이후에 입사한 사원 정보는 MGR_HIST 테이블에 입력한다.

```
SQL〉 SELECT * FROM sal_hist ;
     EMPNO ENAME      HIREDATE        SAL
---------- ---------- -------- ----------
      7369 SMITH      80/12/17        800
      7499 ALLEN      81/02/20       1600
      7521 WARD       81/02/22       1250
      7566 JONES      81/04/02       2975
      7934 MILLER     82/01/23       1300
...

12 rows selected.

SQL〉 SELECT * FROM mgr_hist ;
     EMPNO ENAME      HIREDATE        MGR
---------- ---------- -------- ----------
      7788 SCOTT      87/04/19       7566
      7876 ADAMS      87/05/23       7788
      7934 MILLER     82/01/23       7782
```

〉답안 1.

```
SQL〉 INSERT INTO sal_hist
     SELECT empno, ename, hiredate, sal FROM emp
     WHERE hiredate <= TO_DATE('82/12/31','RR/MM/DD') ;
12 rows created.

SQL〉 @xplan
-------------------------------------------------------------------------------------------------------------
| Id  | Operation                     | Name        | Starts | E-Rows | A-Rows |   A-Time   | Buffers |
-------------------------------------------------------------------------------------------------------------
|   0 | INSERT STATEMENT              |             |      1 |        |      0 |00:00:00.01 |       8 |
|   1 |  LOAD TABLE CONVENTIONAL      |             |      1 |        |      0 |00:00:00.01 |       8 |
|   2 |   TABLE ACCESS BY INDEX ROWID| EMP         |      1 |      6 |     12 |00:00:00.01 |       2 |
|*  3 |    INDEX RANGE SCAN           | EMP_HIRE_IX |      1 |      6 |     12 |00:00:00.01 |       1 |
-------------------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   3 - access("HIREDATE"<=TO_DATE('82/12/31','RR/MM/DD'))

SQL〉 INSERT INTO mgr_hist
     SELECT empno, ename, hiredate, mgr FROM emp
     WHERE hiredate >= TO_DATE('82/01/01','RR/MM/DD') ;
3 rows created.

SQL〉 @xplan
-------------------------------------------------------------------------------------------------------------
| Id  | Operation                     | Name        | Starts | E-Rows | A-Rows |   A-Time   | Buffers |
-------------------------------------------------------------------------------------------------------------
|   0 | INSERT STATEMENT              |             |      1 |        |      0 |00:00:00.01 |       6 |
|   1 |  LOAD TABLE CONVENTIONAL      |             |      1 |        |      0 |00:00:00.01 |       6 |
|   2 |   TABLE ACCESS BY INDEX ROWID| EMP         |      1 |     13 |      3 |00:00:00.01 |       2 |
|*  3 |    INDEX RANGE SCAN           | EMP_HIRE_IX |      1 |     13 |      3 |00:00:00.01 |       1 |
-------------------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   3 - access("HIREDATE">=TO_DATE('82/01/01','RR/MM/DD'))
```

필요한 조건식을 정의하여 두 개의 INSERT 문장으로 처리 가능하다. 단, EMP 테이블에는 동일한 행의 정보를 반복적으로 액세스한다. 이러한 반복 접근은 대량의 처리를 수행할 경우 성능을 저하시킬 수 있다.

```
SQL〉 ROLLBACK ;
```

〉답안 2. 다중 테이블의 INSERT

```
SQL〉 INSERT ALL
     WHEN hiredate <= TO_DATE('82/12/31','RR/MM/DD') THEN
       INTO sal_hist    VALUES (empno,ename,hiredate,sal)
     WHEN hiredate >= TO_DATE('82/01/01','RR/MM/DD') THEN
       INTO mgr_hist    VALUES (empno,ename,hiredate,mgr)
     SELECT empno, ename, hiredate, sal, mgr
     FROM emp ;
15 rows created.

SQL〉 @xplan
-------------------------------------------------------------------------------------------
| Id  | Operation           | Name     | Starts | E-Rows | A-Rows |   A-Time   | Buffers |
-------------------------------------------------------------------------------------------
|   0 | INSERT STATEMENT    |          |      1 |        |      0 |00:00:00.01 |      11 |
|   1 |  MULTI-TABLE INSERT |          |      1 |        |      0 |00:00:00.01 |      11 |
|   2 |   TABLE ACCESS FULL | EMP      |      1 |     14 |     14 |00:00:00.01 |       2 |
|   3 |   INTO              | SAL_HIST |      0 |        |      0 |00:00:00.01 |       0 |
|   4 |   INTO              | MGR_HIST |      0 |        |      0 |00:00:00.01 |       0 |
-------------------------------------------------------------------------------------------

SQL〉 ROLLBACK ;
```

Oracle 9i Database부터 지원하며 Subquery의 결과를 둘 이상의 테이블에 동시에 입력할 때 사용한다. Subquery의 결과는 한 번만 액세스하므로 동일 집합의 반복 접근을 제거하면서 성능을 향상화 시킨다.

• **Unconditional INSERT ALL**

별도의 조건 없이 Subquery의 결과를 둘 이상의 테이블에 입력 (입력된 행의 개수가 동일함)

```
SQL> INSERT ALL
     INTO sal_hist  VALUES (empno,ename,hiredate,sal)
     INTO mgr_hist  VALUES (empno,ename,hiredate,mgr)
     SELECT empno, ename, hiredate, sal, mgr FROM emp ;
28 rows created.
SQL> ROLLBACK ;
```

• **Conditional INSERT ALL**

WHEN 절을 이용하여 조건에 만족하는 경우에만 지정 테이블에 입력 (두 조건을 동시에 만족하는 행은 둘 이상의 테이블에 입력 가능)

```
SQL> INSERT ALL
     WHEN hiredate <= TO_DATE('82/12/31','RR/MM/DD') THEN
       INTO sal_hist    VALUES (empno,ename,hiredate,sal)
     WHEN hiredate >= TO_DATE('82/01/01','RR/MM/DD') THEN
       INTO mgr_hist    VALUES (empno,ename,hiredate,mgr)
     SELECT empno, ename, hiredate, sal, mgr
     FROM emp ;
15 rows created.
SQL> ROLLBACK ;
```

• **Conditional INSERT FIRST**

WHEN 절의 순서에 따라 특정 테이블에 입력된 행은 다음 WHEN 절의 조건식을 비교하지 않음

```
SQL> INSERT FIRST
     WHEN hiredate <= TO_DATE('82/12/31','RR/MM/DD') THEN
       INTO sal_hist    VALUES (empno,ename,hiredate,sal)
     WHEN hiredate >= TO_DATE('82/01/01','RR/MM/DD') THEN
       INTO mgr_hist    VALUES (empno,ename,hiredate,mgr)
     SELECT empno, ename, hiredate, sal, mgr
     FROM emp ;
14 rows created.

SQL> ROLLBACK ;
```

> 문제 2.

EMP 테이블에 DNAME 컬럼을 추가한 후, DEPT 테이블의 DNAME 컬럼을 이용하여 EMP.DNAME 컬럼에 부서 이름을 UPDATE 하시오.

```
SQL> ALTER TABLE emp ADD (dname   VARCHAR2(14)) ;
```

> 출력 결과

```
SQL> SELECT empno, ename, deptno, dname FROM emp ;
     EMPNO ENAME          DEPTNO DNAME
---------- ---------- ---------- --------------
      7369 SMITH              20 RESEARCH
      7499 ALLEN              30 SALES
      7521 WARD               30 SALES
      7566 JONES              20 RESEARCH
      7654 MARTIN             30 SALES
      7698 BLAKE              30 SALES
      7782 CLARK              10 ACCOUNTING
...

14 rows selected.
```

> 답안 1. Correlated Subquery를 이용한 UPDATE

```
SQL> UPDATE emp e
     SET dname = (SELECT dname
                  FROM dept
                  WHERE deptno = e.deptno) ;
14 rows updated.

SQL> @xplan
-------------------------------------------------------------------------------------------------------
| Id  | Operation                    | Name    | Starts | E-Rows | A-Rows |   A-Time   | Buffers |
-------------------------------------------------------------------------------------------------------
|   0 | UPDATE STATEMENT             |         |      1 |        |      0 |00:00:00.01 |       9 |
|   1 |  UPDATE                      | EMP     |      1 |        |      0 |00:00:00.01 |       9 |
|   2 |   TABLE ACCESS FULL          | EMP     |      1 |     14 |     14 |00:00:00.01 |       2 |
|   3 |   TABLE ACCESS BY INDEX ROWID| DEPT    |      3 |      1 |      3 |00:00:00.01 |       6 |
|*  4 |    INDEX UNIQUE SCAN         | PK_DEPT |      3 |      1 |      3 |00:00:00.01 |       3 |
-------------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   4 - access("DEPTNO"=:B1)
```

EMP의 후보 값에 따라 DEPT 테이블로부터 DNAME을 검색하고, 그 값으로 UPDATE를 수행한다. 한가지 다행인 부분은 DEPT 테이블을 EMP의 후보 행 개수만큼 반복 접근을 하지 않은 점이다. EMP 테이블의 DEPTNO는 중복 값을 가지고 있어서 동일 부서 번호에 대해서는 한 번만 DEPT에 접근된 것을 확인할 수 있다. 단, WHERE 절에 사용된 Correlated Subquery가 아니기 때문에 Unnesting이 불가능하고, FILTER 방식처럼 수행되었다. 때문에 후보 값의 종류가 많을 경우 DEPT 테이블의 반복 액세스로 인해 성능이 저하될 수 있다.

〉답안 2. Updatable Join View 사용

```
SQL〉 UPDATE  (SELECT e.deptno, e.dname emp_dname, d.deptno, d.dname dept_dname
                FROM emp e JOIN dept d
                  ON e.deptno = d.deptno )
      SET emp_dname = dept_dname ;
14 rows updated.
SQL〉 @xplan
```

```
-------------------------------------------------------------------------------------------------------------------------------------
¦ Id  ¦ Operation                    ¦ Name          ¦ Starts ¦ E-Rows ¦ A-Rows ¦   A-Time   ¦ Buffers ¦  OMem ¦  1Mem ¦ Used-Mem ¦
-------------------------------------------------------------------------------------------------------------------------------------
¦   0 ¦ UPDATE STATEMENT             ¦                 ¦      1 ¦        ¦      0 ¦00:00:00.01 ¦      20 ¦       ¦       ¦          ¦
¦   1 ¦  UPDATE                      ¦ EMP             ¦      1 ¦        ¦      0 ¦00:00:00.01 ¦      20 ¦       ¦       ¦          ¦
¦   2 ¦   MERGE JOIN                ¦                 ¦      1 ¦     14 ¦     14 ¦00:00:00.01 ¦       4 ¦       ¦       ¦          ¦
¦   3 ¦    TABLE ACCESS BY INDEX ROWID¦ EMP           ¦      1 ¦     14 ¦     14 ¦00:00:00.01 ¦       2 ¦       ¦       ¦          ¦
¦   4 ¦     INDEX FULL SCAN          ¦ EMP_DEPTNO_IX ¦      1 ¦     14 ¦     14 ¦00:00:00.01 ¦       1 ¦       ¦       ¦          ¦
¦*  5 ¦    SORT JOIN                 ¦                 ¦     14 ¦      4 ¦     14 ¦00:00:00.01 ¦       2 ¦ 73728 ¦ 73728 ¦          ¦
¦   6 ¦     TABLE ACCESS FULL        ¦ DEPT            ¦      1 ¦      4 ¦      4 ¦00:00:00.01 ¦       2 ¦       ¦       ¦          ¦
-------------------------------------------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   5 - access("E"."DEPTNO"="D"."DEPTNO")
       filter("E"."DEPTNO"="D"."DEPTNO")
```

INSERT, UPDATE, DELETE 명령문에 Join 결과를 검색하는 Subquery를 이용할 수 있다. 또한 조인의 종류를 선택하는 힌트 등을 이용한다면, 상황에 맞는 조인 종류 및 순서를 조정할 수도 있고 대량의 수정 작업을 보다 빠르게 처리할 수 있다.
단, 이러한 Updatable Join View를 사용할 경우에는 제약 조건이 필요하다. 만약 제약 조건을 제거한다면?

```
SQL〉 ALTER TABLE dept DROP PRIMARY KEY CASCADE ;
SQL〉 UPDATE  (SELECT e.deptno, e.dname emp_dname, d.deptno, d.dname dept_dname
                FROM emp e JOIN dept d ON e.deptno = d.deptno )
      SET emp_dname = dept_dname ;
ERROR at line 4:
ORA-01779: cannot modify a column which maps to a non key-preserved table

SQL〉 UPDATE /*+ bypass_ujvc */
             (SELECT e.deptno, e.dname emp_dname, d.deptno, d.dname dept_dname
                FROM emp e JOIN dept d ON e.deptno = d.deptno )
      SET emp_dname = dept_dname ;
ERROR at line 5:
ORA-01779: cannot modify a column which maps to a non key-preserved table
```

데이터의 유효성을 보장할 수 없기 때문에 (Key-preserved 관계를 확인할 수 없음) UPDATE는 불가능하다.
BYPASS_UJVC 히든 힌트를 이용하면 실행 가능했지만 Oracle 11g부터는 이를 지원하지 않는다.

> 답안 3. MERGE 문 사용

```
SQL> MERGE INTO emp e
     USING dept d
        ON (d.deptno = e.deptno)
     WHEN MATCHED THEN
     UPDATE
     SET e.dname = d.dname ;
14 rows merged.

SQL> @xplan
```

Id	Operation	Name	Starts	E-Rows	A-Rows	A-Time	Buffers	OMem	1Mem	Used-Mem
0	MERGE STATEMENT		1		0	00:00:00.01	20			
1	MERGE	EMP	1		0	00:00:00.01	20			
2	VIEW		1		14	00:00:00.01	4			
3	MERGE JOIN		1	14	14	00:00:00.01	4			
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	4	14	00:00:00.01	2	73728	73728	
7	TABLE ACCESS FULL	DEPT	1	4	4	00:00:00.01	2			

```
Predicate Information (identified by operation id):
---------------------------------------------------
   6 - access("D"."DEPTNO"="E"."DEPTNO")
       filter("D"."DEPTNO"="E"."DEPTNO")
```

제약 조건에 상관없이 MERGE 문을 이용하면 해결 가능하다. MERGE 문은 Oracle 9i부터 지원하며 10g 버전부터는 UPDATE만을 수행하는 MERGE도 가능하다. 또한 힌트를 사용하여 사용자 정의의 조인 방식도 정의할 수 있다. 하나의 테이블에 INSERT, UPDATE, DELETE의 작업을 동시에 수행하려면 반복적인 테이블의 접근을 제거하면서 최적화를 시도할 수 있다.

```
SQL> ALTER TABLE dept ADD CONSTRAINT PK_DEPT PRIMARY KEY (deptno) ;
SQL> ALTER TABLE emp DROP COLUMN dname ;
```

> 추가 실습

```
SQL> DROP TABLE cp_emp PURGE ;
SQL> CREATE TABLE cp_emp AS SELECT * FROM emp WHERE deptno = 20 ;
SQL> UPDATE emp
     SET sal = sal * 1.5
     WHERE JOB = 'CLERK' ;

SQL> SELECT * FROM emp ;
     EMPNO ENAME      JOB              MGR HIREDATE         SAL       COMM     DEPTNO
---------- ---------- --------- ---------- -------- ---------- ---------- ----------
      7369 SMITH      CLERK          7902 80/12/17       1200                    20
      7499 ALLEN      SALESMAN       7698 81/02/20       1600        300         30
      7521 WARD       SALESMAN       7698 81/02/22       1250        500         30
...
14 rows selected.

SQL> SELECT * FROM cp_emp ;
     EMPNO ENAME      JOB              MGR HIREDATE         SAL       COMM     DEPTNO
---------- ---------- --------- ---------- -------- ---------- ---------- ----------
      7369 SMITH      CLERK          7902 80/12/17        800                    20
      7566 JONES      MANAGER        7839 81/04/02       2975                    20
      7788 SCOTT      ANALYST        7566 87/04/19       3000                    20
...
5 rows selected.

SQL> MERGE INTO cp_emp c
     USING emp e
        ON (c.empno = e.empno)
      WHEN MATCHED THEN
      UPDATE SET c.sal    = e.sal ,
                 c.comm   = e.comm
      WHEN NOT MATCHED THEN
      INSERT VALUES (e.empno,e.ename,e.job, e.mgr, e.hiredate, e.sal, e.comm, e.deptno) ;
14 rows merged.

SQL> SELECT * FROM cp_emp ;
     EMPNO ENAME      JOB              MGR HIREDATE         SAL       COMM     DEPTNO
---------- ---------- --------- ---------- -------- ---------- ---------- ----------
      7369 SMITH      CLERK          7902 80/12/17       1200                    20
      7566 JONES      MANAGER        7839 81/04/02       2975                    20
      7788 SCOTT      ANALYST        7566 87/04/19       3000                    20
...
14 rows selected.
```

```
SQL> MERGE INTO cp_emp c
      USING emp e ON (c.empno = e.empno)
      WHEN MATCHED THEN
      UPDATE
      SET c.sal    = e.sal ,
          c.comm   = e.comm
      DELETE WHERE (c.comm IS NOT NULL)
      WHEN NOT MATCHED THEN
      INSERT
      VALUES (e.empno,e.ename,e.job, e.mgr, e.hiredate, e.sal, e.comm, e.deptno) ;
14 rows merged.

SQL> SELECT * FROM cp_emp ;
     EMPNO ENAME      JOB              MGR HIREDATE         SAL        COMM     DEPTNO
---------- ---------- --------- ---------- -------- ---------- ---------- ----------
      7369 SMITH      CLERK           7902 80/12/17       1200                    20
      7566 JONES      MANAGER         7839 81/04/02       2975                    20
      7788 SCOTT      ANALYST         7566 87/04/19       3000                    20
      7876 ADAMS      CLERK           7788 87/05/23       1650                    20
      7902 FORD       ANALYST         7566 81/12/03       3000                    20
      7698 BLAKE      MANAGER         7839 81/05/01       2850                    30
      7782 CLARK      MANAGER         7839 81/06/09       2450                    10
      7839 KING       PRESIDENT            81/11/17       5000                    10
      7900 JAMES      CLERK           7698 81/12/03       1425                    30
      7934 MILLER     CLERK           7782 82/01/23       1950                    10

10 rows selected.

SQL> ROLLBACK ;
SQL> DROP TABLE cp_emp PURGE ;
```

〉문제 3.

EMP 테이블에서 부서별 평균 급여보다 적은 급여를 받는 사원을 검색하여 JOB의 종류에 따라 급여를 수정하시오. 각각의 인상률은 다음의 식을 참고한다. (실제 테이블의 데이터를 DML 명령을 수행하여 수정한다.)

'PRESIDENT' : 30% 인상
'MANAGER' : 25% 인상
'ANALYST' : 20% 인상
그 외의 JOB : 10% 인상

〉출력 결과

```
SQL〉 SELECT * FROM emp ;
     EMPNO ENAME      JOB              MGR HIREDATE        SAL       COMM     DEPTNO
---------- ---------- --------- ---------- -------- ---------- ---------- ----------
      7369 SMITH      CLERK            7902 80/12/17        880                    20
      7499 ALLEN      SALESMAN         7698 81/02/20       1600        300         30
      7521 WARD       SALESMAN         7698 81/02/22       1375        500         30
      7566 JONES      MANAGER          7839 81/04/02       2975                    20
      7654 MARTIN     SALESMAN         7698 81/09/28       1375       1400         30
      7698 BLAKE      MANAGER          7839 81/05/01       2850                    30
      7782 CLARK      MANAGER          7839 81/06/09     3062.5                    10
      7788 SCOTT      ANALYST          7566 87/04/19       3000                    20
      7839 KING       PRESIDENT             81/11/17       5000                    10
      7844 TURNER     SALESMAN         7698 81/09/08       1650          0         30
      7876 ADAMS      CLERK            7788 87/05/23       1210                    20
      7900 JAMES      CLERK            7698 81/12/03       1045                    30
      7902 FORD       ANALYST          7566 81/12/03       3000                    20
      7934 MILLER     CLERK            7782 82/01/23       1430                    10

14 rows selected.
```

> 답안 1. PL/SQL 사용

복잡한 조건식 비교 및 둘 이상의 DML 명령을 함께 수행한다면 PL/SQL과 같은 Language를 이용할 수 있다.

```
SQL> DECLARE
       CURSOR emp_cur IS SELECT * FROM emp ;
       v_avg_10  NUMBER ;
       v_avg_20  NUMBER ;
       v_avg_30  NUMBER ;
     BEGIN
       SELECT AVG(sal) INTO v_avg_10 FROM emp WHERE deptno = 10 ;
       SELECT AVG(sal) INTO v_avg_20 FROM emp WHERE deptno = 20 ;
       SELECT AVG(sal) INTO v_avg_30 FROM emp WHERE deptno = 30 ;

       FOR emp_rec IN emp_cur LOOP
         IF    (emp_rec.sal < v_avg_10 AND emp_rec.deptno = 10)
            OR (emp_rec.sal < v_avg_20 AND emp_rec.deptno = 20)
            OR (emp_rec.sal < v_avg_30 AND emp_rec.deptno = 30)
         THEN
           UPDATE emp
           SET sal = DECODE(job,'PRESIDENT',sal*1.3
                               ,'MANAGER',sal*1.25
                               ,'ANALYST',sal*1.2
                               ,sal*1.1)
           WHERE empno = emp_rec.empno ;
         END IF ;
       END LOOP ;
  END ;
/
PL/SQL procedure successfully completed.

SQL> ROLLBACK ;
```

가장 절차적인 사고방식으로 해결된 문장이다. (부서별 평균 급여를 검색하는 부분도 커서를 이용할 수 있으나 PL/SQL 과정은 아니기 때문에 생략한다. PL/SQL의 개선 방법은 Advanced PL/SQL 과정을 참고한다.)

> 답안 2. MERGE 문 사용

```
SQL> MERGE INTO emp a
     USING (SELECT * FROM ( SELECT e.empno, e.job, e.sal, e.deptno,
                                   DECODE(job,'PRESIDENT', sal*1.3, 'MANAGER', sal*1.25
                                             ,'ANALYST',   sal*1.2, sal*1.1) AS inc_sal,
                                   rowid AS rid,
                                   AVG(sal) OVER(PARTITION BY deptno) AS avg
                            FROM emp e )
            WHERE sal < avg) b
     ON (a.rowid = b.rid)
     WHEN MATCHED THEN
     UPDATE SET a.sal = b.inc_sal ;
8 rows merged.

SQL> @xplan
-------------------------------------------------------------------------------------------------------------------------
| Id  | Operation              | Name | Starts | E-Rows | A-Rows |   A-Time   | Buffers |  OMem |  1Mem | Used-Mem |
-------------------------------------------------------------------------------------------------------------------------
|   0 | MERGE STATEMENT        |      |      1 |        |      0 |00:00:00.01 |      46 |       |       |          |
|   1 |  MERGE                 | EMP  |      1 |        |      0 |00:00:00.01 |      46 |       |       |          |
|   2 |   VIEW                 |      |      1 |        |      8 |00:00:00.01 |       4 |       |       |          |
|*  3 |    HASH JOIN           |      |      1 |     14 |      8 |00:00:00.01 |       4 |   746K|   746K| 1064K (0)|
|   4 |     TABLE ACCESS FULL  | EMP  |      1 |     14 |     14 |00:00:00.01 |       2 |       |       |          |
|*  5 |     VIEW               |      |      1 |     14 |      8 |00:00:00.01 |       2 |       |       |          |
|   6 |      WINDOW SORT       |      |      1 |     14 |     14 |00:00:00.01 |       2 | 73728 | 73728 |          |
|   7 |       TABLE ACCESS FULL| EMP  |      1 |     14 |     14 |00:00:00.01 |       2 |       |       |          |
-------------------------------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   3 - access("A".ROWID="from$_subquery$_007"."RID")
   5 - filter("SAL"<"AVG")

SQL> ROLLBACK ;
```

분석 함수를 이용하여 한 번의 테이블 액세스로 업데이트 대상 집합을 선정하고 ROWID를 이용하여 MERGE 문을 수행하였다. MERGE 문은 힌트를 이용하여 조인의 종류 및 순서 등을 조절할 수도 있으므로 필요한 실행 계획 선택이 가능하다.

> 답안 3. Correlated Subquery 사용

```
SQL> UPDATE emp e
     SET sal = DECODE(job,'PRESIDENT', sal*1.3, 'MANAGER', sal*1.25
                         ,'ANALYST',   sal*1.2, sal*1.1)
      WHERE EXISTS ( SELECT 1 FROM emp
                     WHERE deptno    = e.deptno
                     HAVING AVG(sal) > e.sal ) ;
8 rows updated.

SQL> @xplan
-------------------------------------------------------------------------------------------------------------------
| Id  | Operation                     | Name          | Starts | E-Rows | A-Rows |   A-Time   | Buffers |
-------------------------------------------------------------------------------------------------------------------
|   0 | UPDATE STATEMENT              |               |      1 |        |      0 |00:00:00.01 |      70 |
|   1 |  UPDATE                       | EMP           |      1 |        |      0 |00:00:00.01 |      70 |
|*  2 |   FILTER                      |               |      1 |        |      8 |00:00:00.01 |      26 |
|   3 |    TABLE ACCESS FULL          | EMP           |      1 |     14 |     14 |00:00:00.01 |       2 |
|*  4 |    FILTER                     |               |     12 |        |      7 |00:00:00.01 |      24 |
|   5 |     SORT AGGREGATE            |               |     12 |      1 |     12 |00:00:00.01 |      24 |
|   6 |      TABLE ACCESS BY INDEX ROWID| EMP           |     12 |      5 |     59 |00:00:00.01 |      24 |
|*  7 |       INDEX RANGE SCAN        | EMP_DEPTNO_IX |     12 |      5 |     59 |00:00:00.01 |      12 |
-------------------------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   2 - filter( IS NOT NULL)
   4 - filter(AVG("SAL")>:B1)
   7 - access("DEPTNO"=:B1)

SQL> ROLLBACK ;
```

위와 같은 Correlated Subquery를 이용한 UPDATE 문도 동일한 결과를 볼 수 있다. 하지만 서브 쿼리의 수행 횟수가 12번이다. 이는 서브 쿼리에 E.SAL 컬럼도 함께 공급되므로 DEPTNO, SAL 컬럼을 묶어서 입력된 결과를 캐싱 하기 때문에 발생한 현상이다. ("DISTINCT deptno, sal"의 결과가 12개이다.) FILTER 방식이 아닌 Semi Join 형식으로 처리하면 이러한 반복 처리를 제거하고, 조인의 종류를 선택할 수 있지 않을까?

```
SQL> UPDATE emp e
     SET sal = DECODE(job,'PRESIDENT', sal*1.3, 'MANAGER', sal*1.25
                         ,'ANALYST',   sal*1.2, sal*1.1)
      WHERE EXISTS ( SELECT /*+ unnest */ 1 FROM emp
                     WHERE deptno    = e.deptno
                     HAVING AVG(sal) > e.sal ) ;
8 rows updated.
```

```
SQL> @xplan
```

```
-------------------------------------------------------------------------------------------------------------------
| Id  | Operation                            | Name          | Starts | E-Rows | A-Rows |   A-Time   | Buffers |
-------------------------------------------------------------------------------------------------------------------
|   0 | UPDATE STATEMENT                     |               |      1 |        |      0 |00:00:00.01 |      69 |
|   1 |  UPDATE                              | EMP           |      1 |        |      0 |00:00:00.01 |      69 |
|*  2 |   FILTER                             |               |      1 |        |      8 |00:00:00.01 |      27 |
|   3 |    TABLE ACCESS FULL                 | EMP           |      1 |     14 |     14 |00:00:00.01 |       3 |
|*  4 |    FILTER                            |               |     12 |        |      7 |00:00:00.01 |      24 |
|   5 |     SORT AGGREGATE                   |               |     12 |      1 |     12 |00:00:00.01 |      24 |
|   6 |      TABLE ACCESS BY INDEX ROWID BATCHED| EMP        |     12 |      5 |     59 |00:00:00.01 |      24 |
|*  7 |       INDEX RANGE SCAN               | EMP_DEPTNO_IX |     12 |      5 |     59 |00:00:00.01 |      12 |
-------------------------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   2 - filter( IS NOT NULL)
   4 - filter(AVG("SAL")>:B1)
   7 - access("DEPTNO"=:B1)
```

```
SQL> ROLLBACK ;
```

UNNEST 힌트가 적용되지 못하였다. 서브 쿼리에서의 조건식이 WHERE 절만 사용된 것이 아니고, HAVING 절을 함께 사용했기 때문이다. 결과는 틀리지 않았지만 실행 계획은 최적화가 불가능하다.

```
SQL> DROP TABLE emp2 PURGE ;
SQL> CREATE TABLE emp2 AS
     SELECT *
     FROM emp , (SELECT level AS ID
                 FROM dual
                CONNECT BY level <= 10000) ;

SQL> UPDATE emp2 e
     SET sal = DECODE(job,'PRESIDENT', sal*1.3, 'MANAGER', sal*1.25
                         ,'ANALYST',   sal*1.2, sal*1.1)
      WHERE EXISTS ( SELECT 1
                     FROM emp2
                     WHERE deptno    = e.deptno
                     HAVING AVG(sal) > e.sal ) ;
80000 rows updated.

Elapsed: 00:00:11.51
```

```
SQL> @xplan
```

```
-------------------------------------------------------------------------------------------
| Id  | Operation             | Name | Starts | E-Rows | A-Rows |   A-Time   | Buffers |
-------------------------------------------------------------------------------------------
|   0 | UPDATE STATEMENT      |      |      3 |        |      0 |00:00:08.79 |     158K|
|   1 |  UPDATE               | EMP2 |      3 |        |      0 |00:00:08.79 |     158K|
|*  2 |   FILTER              |      |      3 |        |    163K|00:00:10.31 |   34552 |
|   3 |    TABLE ACCESS FULL  | EMP2 |      3 |    140K|    286K|00:00:01.03 |    1864 |
|*  4 |    FILTER             |      |     36 |        |     21 |00:00:07.88 |   32688 |
|   5 |     SORT AGGREGATE    |      |     36 |      1 |     36 |00:00:07.88 |   32688 |
|*  6 |      TABLE ACCESS FULL| EMP2 |     36 |  46667 |   1770K|00:00:04.07 |   32688 |
-------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   2 - filter( IS NOT NULL)
   4 - filter(AVG("SAL")>:B1)
   6 - filter("DEPTNO"=:B1)
```

```
SQL> ROLLBACK ;
```

임의적으로 생성한 테이블이지만 FILTER 방식으로 처리되는 위의 실행 계획은 반복 접근의 증가로 I/O가 현저히 증가된 것을 확인할 수 있다. 또한 EMP 테이블과 동일하게 12개의 구분 값을 가지고 있는 상황이지만 서브 쿼리의 수행 횟수는 36번 수행되었다. FETCH 단위를 벗어나는 행이 존재하므로 계산된 부서별 평균 급여를 계속 사용하지 못했음을 의미한다.

```
SQL> MERGE INTO emp2 a
     USING (SELECT * FROM ( SELECT e.empno, e.job, e.sal, e.deptno,
                                   DECODE(job,'PRESIDENT', sal*1.3, 'MANAGER', sal*1.25
                                             ,'ANALYST',   sal*1.2, sal*1.1) AS inc_sal,
                                   rowid AS rid,
                                   AVG(sal) OVER(PARTITION BY deptno) AS avg
                              FROM emp2 e )
            WHERE sal < avg) b
     ON (a.rowid = b.rid)
     WHEN MATCHED THEN
     UPDATE SET a.sal = b.inc_sal ;
80000 rows merged.

Elapsed: 00:00:04.76
```

```
SQL> @xplan
```

```
-------------------------------------------------------------------------------------------------------------------------------
| Id  | Operation              | Name | Starts | E-Rows | A-Rows |   A-Time   | Buffers |  OMem |  1Mem | Used-Mem |
-------------------------------------------------------------------------------------------------------------------------------
|   0 | MERGE STATEMENT        |      |      1 |        |      0 |00:00:04.76 |   84635 |       |       |          |
|   1 |  MERGE                 | EMP2 |      1 |        |      0 |00:00:04.76 |   84635 |       |       |          |
|   2 |   VIEW                 |      |      1 |        |  80000 |00:00:03.73 |    1816 |       |       |          |
|*  3 |    HASH JOIN           |      |      1 |    140K|  80000 |00:00:03.39 |    1816 |    14M|  2547K|   14M (0)|
|   4 |     TABLE ACCESS FULL  | EMP2 |      1 |    140K|    140K|00:00:00.29 |     908 |       |       |          |
|*  5 |     VIEW               |      |      1 |    140K|  80000 |00:00:01.50 |     908 |       |       |          |
|   6 |      WINDOW SORT       |      |      1 |    140K|    140K|00:00:01.01 |     908 |  7990K|  1115K| 7102K (0)|
|   7 |       TABLE ACCESS FULL| EMP2 |      1 |    140K|    140K|00:00:00.29 |     908 |       |       |          |
-------------------------------------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   3 - access("A".ROWID="from$_subquery$_007"."RID")
   5 - filter("SAL"<"AVG")
```

```
SQL> ROLLBACK ;
```

MERGE 문을 이용하여 각 집합의 접근을 최소화하고 Hash Join을 이용하여 MERGE 되었다. 테이블의 접근된 I/O 수가 감소한 것을 확인할 수 있다. 때문에 수행 시간도 앞선 문장보다 개선되었다.

만약 Correlated Subquery를 Unnesting이 가능하도록 수정하면 어떨까?

```
SQL> UPDATE emp2 e
     SET sal = DECODE(job,'PRESIDENT', sal*1.3, 'MANAGER', sal*1.25, 'ANALYST',   sal*1.2, sal*1.1)
     WHERE SAL < ( SELECT /*+ unnest */ AVG(sal) FROM emp2
                   WHERE deptno    = e.deptno ) ;
80000 rows updated.

Elapsed: 00:00:03.27
```

```
SQL> @xplan
```

```
-------------------------------------------------------------------------------------------------------------------------------
| Id  | Operation              | Name    | Starts | E-Rows | A-Rows |   A-Time   | Buffers |  OMem |  1Mem | Used-Mem |
-------------------------------------------------------------------------------------------------------------------------------
|   0 | UPDATE STATEMENT       |         |      1 |        |      0 |00:00:03.27 |   84618 |       |       |          |
|   1 |  UPDATE                | EMP2    |      1 |        |      0 |00:00:03.27 |   84618 |       |       |          |
|*  2 |   HASH JOIN            |         |      1 |   7000 |  80000 |00:00:02.74 |    1816 |  1368K|  1368K|  641K (0)|
|   3 |    VIEW                | VW_SQ_1 |      1 |      3 |      3 |00:00:00.61 |     908 |       |       |          |
|   4 |     SORT GROUP BY      |         |      1 |      3 |      3 |00:00:00.61 |     908 | 73728 | 73728 |          |
|   5 |      TABLE ACCESS FULL | EMP2    |      1 |    140K|    140K|00:00:00.30 |     908 |       |       |          |
|   6 |    TABLE ACCESS FULL   | EMP2    |      1 |    140K|    140K|00:00:00.51 |     908 |       |       |          |
-------------------------------------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   2 - access("ITEM_1"="E"."DEPTNO")
       filter("SAL"<"AVG(SAL)")
```

가장 최적의 성능을 확인할 수 있다. MERGE 문에서 사용한 서브 쿼리는 UPDATE 대상을 정의하기 위해 분석 함수의 사용이 필요했다. 때문에 추가적인 연산 및 메모리 공간을 필요로 하고 이는 성능상 취약점을 가져간다. 하지만 위의 Correlated Subquery는 Unnesting이 가능하면서도 GROUP BY 절을 통해 집합의 크기를 현저히 줄여 줄 수 있으므로 메모리 사용량에서도 차이를 보여 준다. 그렇다면 MERGE 문에서도 GROUP BY 절을 통해 처리하면 안 될까?

```
SQL> MERGE INTO emp2 a
     USING (SELECT deptno, AVG(sal) AS avg
            FROM emp2
            GROUP BY deptno) b
     ON (a.deptno = b.deptno AND a.sal < b.avg)
     WHEN MATCHED THEN
       UPDATE SET a.sal = DECODE(a.job,'PRESIDENT', a.sal*1.3,
                                       'MANAGER'  , a.sal*1.25,
                                       'ANALYST'  , a.sal*1.2, a.sal*1.1) ;

ERROR at line 5:
ORA-38104: Columns referenced in the ON Clause cannot be updated: "A"."SAL"

SQL> DROP TABLE emp2 PURGE ;
```

안타깝게도 조건에 비교해야 하는 SAL 컬럼의 값은 MERGE 문에서 UPDATE는 불가능하다.

DML 명령문은 조건을 통해 소량의 데이터가 수정되는 경우에는 성능상 특별한 이슈가 없을 것이다. 만약 성능 저하가 있다면 Lock에 의한 대기 현상이 가장 큰 Waiting의 원인일 수 있다. 하지만 대량의 데이터를 수정해야 하는 경우라면 상황에 따른 처리 방법을 잘 선택해야 한다. 경우에 따라서 업무 로직이 복잡하여 PL/SQL을 이용해야 하는 경우도 존재할 수 있고, Correlated Subquery를 이용한 DML 작업 시 실행 계획을 확인하여 최적화가 불가능한 상황이라면 MERGE 문의 사용도 함께 고려한다.

Appendixes

1. Join, Subquery 문장 작성
2. WITH 절
3. 계층 질의
4. Entity Relationship Diagram (ERD)

조인, 서브쿼리 활용

〉 문제 1.

EMP, FAMILY, SAL_HISTORY 테이블을 이용하여 한 명 이상의 부양가족을 갖는 사원이 2013 년도에 수령한 급여의 합계를 검색하시오.

EMP : 사원 테이블
FAMILY : 부양 가족 테이블
SAL_HISTORY : 월별 급여 테이블

〉 출력 결과

```
     EMPNO ENAME      SUM(S.SALARY)
---------- ---------- -------------
      7521 WARD               12000
      7844 TURNER             14400
      7876 ADAMS              10560
      7499 ALLEN              15360
      7369 SMITH               7680
      7900 JAMES               9120
      7654 MARTIN             12000
      7934 MILLER             12480
8 rows selected.
```

〉 잘못된 답안

```
SQL〉 SELECT e.empno, e.ename, SUM(s.salary)
     FROM emp e, sal_history s, family f
     WHERE e.empno = s.empno
       AND e.empno = f.empno
       AND s.pay_day BETWEEN TO_DATE('2013/01/31', 'YYYY/MM/DD')
                         AND TO_DATE('2013/12/31', 'YYYY/MM/DD')
     GROUP BY e.empno, e.ename ;
```

```
     EMPNO ENAME      SUM(S.SALARY)
---------- ---------- -------------
      7521 WARD               36000
      7844 TURNER             14400
      7876 ADAMS              21120
      7499 ALLEN              46080
      7369 SMITH              23040
      7900 JAMES              18240
      7654 MARTIN             36000
      7934 MILLER             24960

8 rows selected.
```

EMP, FAMILY 테이블은 1:M 관계이다. 때문에 두 테이블의 조인 결과는 M 의 집합이 생성되고, SAL_HISTORY 테이블과 조인이 시 중복되는 EMPNO 를 처리하지 않으면 M:M 의 결과가 생성된다. 때문에 부양가족이 한 명인 사원은 급여의 합계가 틀리지 않지만 경우에 따라 결과가 유효하지 않을 수 있다.

> 답안 1.

```
SQL> SELECT e.empno, e.ename, SUM(s.salary)
     FROM emp e, sal_history s
     WHERE e.empno = s.empno
       AND s.pay_day BETWEEN TO_DATE('2013/01/31', 'YYYY/MM/DD')
                         AND TO_DATE('2013/12/31', 'YYYY/MM/DD')
       AND EXISTS ( SELECT 1 FROM family
                    WHERE empno = e.empno )
     GROUP BY e.empno, e.ename ;
```

FAMILY 테이블은 부양가족의 유무를 확인하기 위한 부분이기에 함께 조인될 필요는 없다. EXISTS 연산자를 이용하여 존재 유무만을 확인한다.

> 답안 2.

```
SQL> SELECT e.empno, e.ename, SUM(s.salary)
     FROM sal_history s,
          ( SELECT DISTINCT a.empno, a.ename
            FROM emp a, family b
            WHERE a.empno = b.empno ) e
     WHERE e.empno = s.empno
       AND s.pay_day BETWEEN TO_DATE('2013/01/31', 'YYYY/MM/DD')
                         AND TO_DATE('2013/12/31', 'YYYY/MM/DD')
     GROUP BY e.empno, e.ename ;
```

중복되는 EMPNO 를 제거하기 위해 DISTINCT 이용하여 1 의 집합을 생성하고 SAL_HISTORY 테이블과 조인을 수행한다.

› 문제 2.

SALES_PLAN, SALES_RESULT 테이블은 판매 계획 및 실제 판매 정보를 저장하는 테이블이다. 두 테이블을 이용하여 제품별 판매 계획 및 실제 판매 정보를 다음과 같이 검색하시오.

› 출력 결과

```
ITEM                 SALE_DATE             PLAN_QTY RESULT_QTY
-------------------- -------------------- ---------- ----------
Accessories          20120101                    300        200
Accessories          20120102                    300        100
Accessories          20120120                    100        100
Accessories          20120124                    100          0
CD-ROM               20120101                      0        300
Camcorders           20120122                    600        600
Cameras              20120104                    800        800
Monitors             20120101                      0        100
Piano                20120121                      0        100
Piano                20120124                      0        100
piano                20120121                    100          0
piano                20120124                    100          0
```

› 잘못된 답안

```
SQL> SELECT NVL(a.item, b.item)              AS item ,
            NVL(a.plan_date, b.sale_date)    AS sale_date ,
            NVL(a.qty, 0)                   AS plan_qty ,
            NVL(b.qty, 0)                   AS result_qty
     FROM sales_plan a, sales_result b
     WHERE a.item      = b.item
       AND a.plan_date = b.sale_date ;

ITEM                 SALE_DATE             PLAN_QTY RESULT_QTY
-------------------- -------------------- ---------- ----------
Accessories          20120101                    100        200
Accessories          20120101                    100        200
Accessories          20120101                    100        200
Accessories          20120102                    300        100
Accessories          20120120                    100        100
...

13 rows selected.
```

두 테이블은 M:M 관계이므로 M:M 관계를 해소하지 못하면 잘못된 결과가 검색될 수 있다. 제대로 된 결과를 검색하기 위해서는 1:M, 1:1 관계를 생성할 수 있도록 상황에 맞는 집합의 가공이 필요하며, 현재와 같은 상황에서는 일자별 존재하지 않는 데이터도 있을 수 있으므로 Full Outer Join 또는 UNION ALL 문장을 사용한다.

> 답안 1. 중복 제거 후 Full Outer Join 사용

```
SQL> SELECT NVL(a.item, b.item)             AS item ,
            NVL(a.plan_date, b.sale_date)   AS sale_date ,
            NVL(plan_qty, 0)               AS plan_qty ,
            NVL(result_qty, 0)             AS result_qty
     FROM (SELECT item, plan_date, SUM(qty) AS plan_qty
           FROM sales_plan GROUP BY item, plan_date) a
     FULL OUTER JOIN
         (SELECT item, sale_date, SUM(qty) AS result_qty
          FROM sales_result GROUP BY item, sale_date) b
     ON a.item       = b.item
     AND a.plan_date = b.sale_date
     ORDER BY 1, 2 ;
```

> 답안 2. UNION ALL 사용

```
SQL> SELECT item,
            sale_date,
            NVL(SUM(plan_qty),0)   AS plan_qty,
            NVL(SUM(result_qty),0) AS result_qty
     FROM (SELECT item, plan_date AS sale_date, qty AS plan_qty, NULL AS result_qty
           FROM sales_plan
           UNION ALL
           SELECT item, sale_date, null, qty
           FROM sales_result)
     GROUP BY item, sale_date
     ORDER BY 1, 2 ;
```

문제 3.

그림 3 의 ERD 를 참고하여 "K9" 차량의 렌트 현황을 검색하시오.

› 출력 결과

```
CONTRACT_ID CAR_NAME        START_DATE END_DATE   CUST_TYPE          CUST_ID CUST_NAME
----------- --------------- ---------- ---------- --------------- ---------- ---------------
         34 K9              15/04/10   15/04/15   Corporation         103589 Anderson
         49 K9              15/01/06   15/01/15   Corporation         104433 Lau
         13 K9              15/01/19   15/01/26   Person                 379 Perez
          9 K9              15/01/18   15/01/24   Person                 309 Kotch
         25 K9              15/01/10   15/01/13   Person                 862 Stone
```

› 확인 사항

```
SQL> SELECT r.contract_id, cr.car_name, r.start_date, r.end_date,
            r.cust_typ, c.cust_id, c.cust_name
     FROM car cr, rental r, corporation c
     WHERE cr.car_id = r.car_id  AND r.cust_id = c.cust_id
       AND r.cust_typ = 'C'      AND cr.car_name = 'K9' ;

CONTRACT_ID CAR_NAME        START_DATE END_DATE   C CUST_NAME
----------- --------------- ---------- ---------- - ---------------
         34 K9              15/04/10   15/04/15   C Anderson
         49 K9              15/01/06   15/01/15   C Lau

SQL> SELECT r.contract_id, cr.car_name, r.start_date, r.end_date, r.cust_typ,
            p.cust_id, p.cust_name
     FROM car cr, rental r, person p
     WHERE cr.car_id = r.car_id  AND r.cust_id = p.cust_id
       AND r.cust_typ = 'P'      AND cr.car_name = 'K9' ;

CONTRACT_ID CAR_NAME        START_DATE END_DATE   C CUST_NAME
----------- --------------- ---------- ---------- - ---------------
         25 K9              15/01/10   15/01/13   P Stone
          9 K9              15/01/18   15/01/24   P Kotch
         13 K9              15/01/19   15/01/26   P Perez
```

RENTAL, PERSON, CORPORATION 테이블은 배타적 관계를 가지고 있다. RENTAL 테이블의 CUST_TYP 컬럼을 통해 개인, 법인 고객을 구분을 하며(Sub Type) 필요에 따라 각각 다른 테이블을 조인하여 원하는 결과를 검색할 수 있다. 위의 두 문장을 UNION ALL 을 이용하여 합집합의 결과를 출력하면 원하는 결과를 확인할 수 있으나 동일 테이블의 반복 접근이 많아지므로 성능은 저하될 것이다.

> 답안. OUTER JOIN 사용

```
SQL> SELECT r.contract_id, cr.car_name, r.start_date, r.end_date,
            DECODE(r.cust_typ,'P','Person',
                              'C','Corporation') AS cust_type,
            NVL(p.cust_name, c.cust_name) AS cust_name
     FROM car cr, rental r, person p, corporation c
     WHERE cr.car_id = r.car_id
       AND r.cust_id = p.cust_id
       AND r.cust_id = c.cust_id
       AND cr.car_name = 'K9' ;
no rows selected
```

PERSON, CORPORATION 테이블의 CUST_ID 는 서로 같은 값을 가지고 있지 않으므로 위와 같은 조인 문장은 제대로 된 결과를 검색할 수 없다. (배타적인 관계에서는 두 테이블 중 어느 한쪽에서만 RENTAL 테이블과 관계를 갖는다.)

```
SQL> SELECT r.contract_id, cr.car_name, r.start_date, r.end_date,
            DECODE(r.cust_typ,'P','Person',
                              'C','Corporation') AS cust_type,
            NVL(p.cust_id, c.cust_id) AS cust_id,
            NVL(p.cust_name, c.cust_name) AS cust_name
     FROM car cr, rental r, person p, corporation c
     WHERE cr.car_id = r.car_id
       AND r.cust_id = p.cust_id (+)
       AND r.cust_id = c.cust_id (+)
       AND cr.car_name = 'K9' ;
CONTRACT_ID CAR_NAME        START_DATE END_DATE   CUST_TYPE          CUST_ID CUST_NAME
----------- --------------- ---------- ---------- --------------- ---------- ---------------
         34 K9              15/04/10   15/04/15   Corporation         103589 Anderson
         49 K9              15/01/06   15/01/15   Corporation         104433 Lau
         13 K9              15/01/19   15/01/26   Person                 379 Perez
          9 K9              15/01/18   15/01/24   Person                 309 Kotch
         25 K9              15/01/10   15/01/13   Person                 862 Stone
```

배타적 관계의 테이블의 조인에서는 Outer Join 을 이용하면 위와 같이 실행 결과를 검색할 수 있다. 단, 위의 문장은 신뢰할 수 없는 결과이다. 그 이유는?

```
SQL> SAVE q1 REPLACE
SQL> UPDATE corporation SET cust_id = 379 WHERE cust_id = 104433 ;
SQL> UPDATE rental      SET cust_id = 379 WHERE contract_id = 49 ;
```

법인 고객의 CUST_ID 를 개인 고객 테이블에 존재하는 CUST_ID 의 값과 동일한 값으로 수정했다. 그리고 위의 문장을 다시 실행하면?

```
SQL> @q1
```

CONTRACT_ID	CAR_NAME	START_DATE	END_DATE	CUST_TYPE	CUST_ID	CUST_NAME
34	K9	15/04/10	15/04/15	Corporation	103589	Anderson
13	K9	15/01/19	15/01/26	Person	379	Perez
49	**K9**	**15/01/06**	**15/01/15**	**Corporation**	**379**	**Perez**
9	K9	15/01/18	15/01/24	Person	309	Kotch
25	K9	15/01/10	15/01/13	Person	862	Stone

PERSON, CORPORATION 테이블에 동일한 CUST_ID 가 존재할 경우에는 RENTAL 테이블의 CUST_TYP 컬럼을 이용하여, 필요한 테이블에 조인을 수행해야 한다. 만약 CUST_TYP 컬럼에 대한 조건식이 비교되지 않는다면, 동일한 CUST_ID 를 가지고 있는 PERSON, CORPORATION 테이블의 행은 불필요한 연결 결과를 생성할 수 있고, 경우에 따라 잘못된 결과를 검색하게 된다.
다음과 같이 DECODE 를 사용하여 필요에 따라 P.CUST_NAME, C.CUST_NAME 을 서로 다르게 표시할 수도 있지만 성능상 유리한 문장은 아니다.

```
SQL> SELECT r.contract_id, cr.car_name, r.start_date, r.end_date,
            DECODE(r.cust_typ,'P','Person', 'C', 'Corporation') AS cust_type,
            DECODE(r.cust_typ,'P', p.cust_id,   c.cust_id)      AS cust_id,
            DECODE(r.cust_typ,'P', p.cust_name, c.cust_name)    AS cust_name
     FROM car cr, rental r, person p, corporation c
     WHERE cr.car_id = r.car_id
       AND r.cust_id = p.cust_id (+)
       AND r.cust_id = c.cust_id (+)
       AND cr.car_name = 'K9' ;
```

CONTRACT_ID	CAR_NAME	START_DATE	END_DATE	CUST_TYPE	CUST_ID	CUST_NAME
34	K9	15/04/10	15/04/15	Corporation	103589	Anderson
13	K9	15/01/19	15/01/26	Person	379	Perez
49	**K9**	**15/01/06**	**15/01/15**	**Corporation**	**379**	**Lau**
9	K9	15/01/18	15/01/24	Person	309	Kotch
25	K9	15/01/10	15/01/13	Person	862	Stone

결과는 제대로 검색되었으나 49 번의 계약 정보는 PERSON 테이블의 조인 결과도 생성했다. 다만, 화면에만 출력되지 않았다.

```
SQL> SELECT r.contract_id, cr.car_name, r.start_date, r.end_date,
            DECODE(r.cust_typ,'P','Person',
                              'C','Corporation') AS cust_type,
            NVL(p.cust_id, c.cust_id) AS cust_id,
            NVL(p.cust_name, c.cust_name) AS cust_name
     FROM car cr, rental r, person p, corporation c
     WHERE cr.car_id = r.car_id
       AND DECODE(r.cust_typ, 'P', r.cust_id) = p.cust_id (+)
       AND DECODE(r.cust_typ, 'C', r.cust_id) = c.cust_id (+)
       AND cr.car_name = 'K9' ;
```

CONTRACT_ID	CAR_NAME	START_DATE	END_DATE	CUST_TYPE	CUST_ID	CUST_NAME
34	K9	15/04/10	15/04/15	Corporation	103589	Anderson
49	**K9**	**15/01/06**	**15/01/15**	**Corporation**	**379**	**Lau**
13	K9	15/01/19	15/01/26	Person	379	Perez
9	K9	15/01/18	15/01/24	Person	309	Kotch
25	K9	15/01/10	15/01/13	Person	862	Stone

```
SQL> ROLLBACK ;
```

배타적 관계의 테이블의 전체 조인을 수행해야 할 경우에는 DECODE 를 활용하여 CUST_TYP 에 따라 필요한 조인 작업만을 수행하도록 문장을 작성한다. 위의 문장은 PERSON, CORPORATION 테이블에 같은 CUST_ID 가 존재하더라도 필요한 조인만 수행하기 때문에 불필요한 접근은 발생하지 않는다. 또는 배타적인 관계를 제거하여 관리할 수도 있다. (그림 4 참고)

```
SQL> conn user02/oracle
SQL> SELECT r.contract_id, cr.car_name, r.start_date, r.end_date,
            DECODE(c.cust_typ,'P', 'Person', 'C', 'Corporation') AS cust_type,
            c.cust_id, c.cust_name
     FROM car cr, rental r, rent_cust c
     WHERE r.car_id = cr.car_id AND r.cust_id = c.cust_id AND cr.car_name = 'K9' ;
```

CONTRACT_ID	CAR_NAME	START_DATE	END_DATE	CUST_TYPE	CUST_ID	CUST_NAME
25	K9	15/01/10	15/01/13	Person	862	Stone
9	K9	15/01/18	15/01/24	Person	309	Kotch
34	K9	15/04/10	15/04/15	Corporation	103589	Anderson
49	K9	15/01/06	15/01/15	Corporation	104433	Lau
13	K9	15/01/19	15/01/26	Person	379	Perez

```
SQL> conn user01/oracle
```

› 문제 4.

EMP 테이블에서 1981 년도에 입사한 사원들을 입사 월별로 인원수를 검색하시오.
단, 사원이 없는 월도 함께 출력

› 출력 결과

```
HIRE                 CNT
-------------- ----------
1981/01               0
1981/02               2
1981/03               0
1981/04               1
1981/05               1
1981/06               1
1981/07               0
1981/08               0
1981/09               2
1981/10               0
1981/11               1
1981/12               2

12 rows selected.
```

› 답안

```
SQL› SELECT b.hire, NVL(a.cnt,0) CNT
     FROM (SELECT TO_CHAR(hiredate,'YYYY/MM') hire, count(*) cnt
           FROM emp
           WHERE hiredate BETWEEN TO_DATE('81/01/01','RR/MM/DD')
                              AND TO_DATE('81/12/31','RR/MM/DD')
           GROUP BY TO_CHAR(hiredate,'YYYY/MM')) a,
          (SELECT '1981/'||LPAD(LEVEL,2,0) hire
           FROM dual
           CONNECT BY LEVEL <= 12) b
     WHERE a.hire (+) = b.hire
     ORDER BY 1 ;
```

'1981/01' ~ '1981/12'까지의 행 집합을 생성하고 Outer Join 을 사용하여 원하는 결과 집합을 생성할 수 있다.

WITH 절

Oracle Database 10g 부터 지원하는 WITH 절은 하나의 쿼리 내에서 반복되는 쿼리 집합을 미리 정의하여 임시 테이블에 저장된 데이터로 사용한다. 즉, 하나의 쿼리 내에서만 접근 가능한 임시 테이블이 WITH 절이다.

```
SQL> SELECT deptno, SUM(sal) AS SUM
     FROM emp
     GROUP BY deptno
     HAVING SUM(sal) > ( SELECT AVG(SUM(sal))
                         FROM emp
                         GROUP BY deptno ) ;
    DEPTNO        SUM
---------- ----------
        20      10875
```

• 발견 사항

부서별 급여의 합계 계산 작업이 두 번 실행된다.

```
SQL> WITH sum_sal AS ( SELECT deptno, SUM(sal) AS SUM
                       FROM emp
                       GROUP BY deptno )
     SELECT *
     FROM sum_sal
     WHERE sum > ( SELECT AVG(sum) FROM sum_sal ) ;
    DEPTNO        SUM
---------- ----------
        20      10875
```

• 발견 사항

중복될 수 있는 Subquery 를 WITH 절에서 한 번만 실행할 수 있다. WITH 절에 정의된 집합은 해당 문장 내에서만 사용 가능한 임시 테이블의 이름을 가지므로 문장 어디에서든 해당 집합에 대한 접근이 가능하다. 때문에 하나의 쿼리 안에서 반복되는 동일 쿼리 집합이 있다면 이를 WITH 절에 먼저 정의해 놓음으로써 반복적인 쿼리의 재 실행을 줄일 수 있다.

그렇다면 성능은?

```
SQL> WITH sum_sal AS ( SELECT deptno, SUM(sal) AS SUM
                       FROM emp
                       GROUP BY deptno )
     SELECT *
     FROM sum_sal
     WHERE sum > ( SELECT AVG(sum) FROM sum_sal ) ;
SQL> @iostat
```

```
-------------------------------------------------------------------------------------------------------------------------------------
| Id  | Operation                  | Name                     | Starts | E-Rows | A-Rows |   A-Time   | Buffers | Reads  | Writes |
-------------------------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT           |                          |      1 |        |      1 |00:00:00.01 |      17 |      1 |      1 |
|   1 |  TEMP TABLE TRANSFORMATION |                          |      1 |        |      1 |00:00:00.01 |      17 |      1 |      1 |
|   2 |   LOAD AS SELECT           |                          |      1 |        |      0 |00:00:00.01 |       6 |      0 |      1 |
|   3 |    HASH GROUP BY           |                          |      1 |      3 |      3 |00:00:00.01 |       2 |      0 |      0 |
|   4 |     TABLE ACCESS FULL      | EMP                      |      1 |     14 |     14 |00:00:00.01 |       2 |      0 |      0 |
|*  5 |   VIEW                     |                          |      1 |      3 |      1 |00:00:00.01 |       8 |      1 |      0 |
|   6 |    TABLE ACCESS FULL       | SYS_TEMP_0FD9D663C_E5DF4 |      1 |      3 |      3 |00:00:00.01 |       6 |      1 |      0 |
|   7 |    SORT AGGREGATE          |                          |      1 |      1 |      1 |00:00:00.01 |       2 |      0 |      0 |
|   8 |     VIEW                   |                          |      1 |      3 |      3 |00:00:00.01 |       2 |      0 |      0 |
|   9 |      TABLE ACCESS FULL     | SYS_TEMP_0FD9D663C_E5DF4 |      1 |      3 |      3 |00:00:00.01 |       2 |      0 |      0 |
-------------------------------------------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   5 - filter("SUM">)
```

- **발견 사항**

WITH 절의 집합은 문장 실행 시 가장 먼저 실행되며 해당 결과를 임시 데이터로 저장한다. 문제는 이러한 임시 데이터의 반복 접근이 일어날 때마다 물리적인 I/O 가 증가되어 오히려 성능이 저하될 가능성도 있다.

※ WITH 절의 Query Transformation

```
SQL> WITH sum_sal AS ( SELECT deptno, SUM(sal) AS SUM
                       FROM emp
                       GROUP BY deptno )
     SELECT *
     FROM dept d, sum_sal s
     WHERE d.deptno = s.deptno ;
SQL> @iostat
```

```
-------------------------------------------------------------------------------------------------------
| Id  | Operation                     | Name    | Starts | E-Rows | A-Rows |   A-Time   | Buffers |
-------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT              |         |      1 |        |      3 |00:00:00.01 |       6 |
|   1 |  MERGE JOIN                   |         |      1 |      3 |      3 |00:00:00.01 |       6 |
|   2 |   TABLE ACCESS BY INDEX ROWID| DEPT    |      1 |      4 |      4 |00:00:00.01 |       4 |
|   3 |    INDEX FULL SCAN            | PK_DEPT |      1 |      4 |      4 |00:00:00.01 |       2 |
|*  4 |   SORT JOIN                   |         |      4 |      3 |      3 |00:00:00.01 |       2 |
|   5 |    VIEW                       |         |      1 |      3 |      3 |00:00:00.01 |       2 |
|   6 |     HASH GROUP BY             |         |      1 |      3 |      3 |00:00:00.01 |       2 |
|   7 |      TABLE ACCESS FULL        | EMP     |      1 |     14 |     14 |00:00:00.01 |       2 |
-------------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   4 - access("D"."DEPTNO"="S"."DEPTNO")
       filter("D"."DEPTNO"="S"."DEPTNO")
```

• 발견 사항

WITH 절에 정의된 문장이 한 번만 사용되면 Inline View 형식으로 변형되어 사용된다. 즉, 임시 데이터의 집합이 생성되지 않는다.

• 결론

1. WITH 절은 반복되는 쿼리의 결과를 문장 시작 전 액세스하여 임시 테이블로 저장, 재 사용을 목적으로 한다.
2. 문장 안에 WITH 절의 Subquery 가 두 번 이상 사용되면 임시 테이블이 생성되며, 한 번 사용되면 Inline View 형식으로 실행된다.
3. WITH 절의 임시 테이블 형식의 사용은 Physical I/O 를 동반할 수 있으므로 그 크기가 크지 않은 경우에 유리하다.
4. MATERIALIZE, INLINE 힌트를 이용하여 형식 지정 가능 (11gNF)
5. "_with_subquery" 파라미터로 제어 가능 (11gNF)

• 힌트 사용 방법

```
SQL> WITH sum_sal AS ( SELECT /*+ materialize */ deptno, SUM(sal) AS SUM
                       FROM emp GROUP BY deptno )
     SELECT *
     FROM dept d, sum_sal s
     WHERE d.deptno = s.deptno ;
SQL> @iostat
```

Id	Operation	Name	Starts	E-Rows	A-Rows	A-Time	Buffers	Reads	Writes
0	SELECT STATEMENT		1		3	00:00:00.01	18	1	1
1	TEMP TABLE TRANSFORMATION		1		3	00:00:00.01	18	1	1
2	LOAD AS SELECT		1		0	00:00:00.01	6	0	1
3	HASH GROUP BY		1	3	3	00:00:00.01	2	0	0
4	TABLE ACCESS FULL	EMP	1	14	14	00:00:00.01	2	0	0
5	MERGE JOIN		1	3	3	00:00:00.01	9	1	0
6	TABLE ACCESS BY INDEX ROWID	DEPT	1	4	4	00:00:00.01	4	0	0
7	INDEX FULL SCAN	PK_DEPT	1	4	4	00:00:00.01	2	0	0
* 8	SORT JOIN		4	3	3	00:00:00.01	5	1	0
9	VIEW		1	3	3	00:00:00.01	5	1	0
10	TABLE ACCESS FULL	SYS_TEMP_0FD9D663D_E5DF4	1	3	3	00:00:00.01	5	1	0

```
Predicate Information (identified by operation id):
---------------------------------------------------
   8 - access("D"."DEPTNO"="S"."DEPTNO")
       filter("D"."DEPTNO"="S"."DEPTNO")
```

```
SQL> WITH sum_sal AS ( SELECT /*+ inline */ deptno, SUM(sal) AS SUM
                       FROM emp GROUP BY deptno )
     SELECT *
     FROM dept d, sum_sal s
     WHERE d.deptno = s.deptno ;
SQL> @iostat
```

Id	Operation	Name	Starts	E-Rows	A-Rows	A-Time	Buffers
0	SELECT STATEMENT		1		3	00:00:00.01	6
1	MERGE JOIN		1	3	3	00:00:00.01	6
2	TABLE ACCESS BY INDEX ROWID	DEPT	1	4	4	00:00:00.01	4
3	INDEX FULL SCAN	PK_DEPT	1	4	4	00:00:00.01	2
* 4	SORT JOIN		4	3	3	00:00:00.01	2
5	VIEW		1	3	3	00:00:00.01	2
6	HASH GROUP BY		1	3	3	00:00:00.01	2
7	TABLE ACCESS FULL	EMP	1	14	14	00:00:00.01	2

```
Predicate Information (identified by operation id):
---------------------------------------------------
   4 - access("D"."DEPTNO"="S"."DEPTNO")
       filter("D"."DEPTNO"="S"."DEPTNO")
```

```
SQL> WITH sum_sal AS ( SELECT /*+ inline */ deptno, SUM(sal) AS SUM
                       FROM emp
                       GROUP BY deptno )
     SELECT *
     FROM sum_sal
     WHERE sum > ( SELECT AVG(sum) FROM sum_sal ) ;
SQL> @iostat
```

```
-------------------------------------------------------------------------------------------
| Id  | Operation             | Name | Starts | E-Rows | A-Rows |   A-Time   | Buffers |
-------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT      |      |      1 |        |      1 |00:00:00.01 |       4 |
|*  1 |  FILTER               |      |      1 |        |      1 |00:00:00.01 |       4 |
|   2 |   HASH GROUP BY       |      |      1 |      1 |      3 |00:00:00.01 |       2 |
|   3 |    TABLE ACCESS FULL  | EMP  |      1 |     14 |     14 |00:00:00.01 |       2 |
|   4 |   SORT AGGREGATE      |      |      1 |      1 |      1 |00:00:00.01 |       2 |
|   5 |    VIEW               |      |      1 |      3 |      3 |00:00:00.01 |       2 |
|   6 |     SORT GROUP BY     |      |      1 |      3 |      3 |00:00:00.01 |       2 |
|   7 |      TABLE ACCESS FULL| EMP  |      1 |     14 |     14 |00:00:00.01 |       2 |
-------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   1 - filter(SUM("SAL")>)
```

• SQL Tuning 결과

```
SQL> SELECT deptno, sum_sal
     FROM ( SELECT deptno, sum_sal, AVG(sum_sal) OVER() AS avg
            FROM ( SELECT deptno, SUM(sal) AS sum_sal
                     FROM emp
                   GROUP BY deptno ) )
     WHERE sum_sal > avg  ;
SQL> @xplan
```

```
-------------------------------------------------------------------------------------------------------------------------
| Id  | Operation             | Name | Starts | E-Rows | A-Rows |   A-Time   | Buffers |  OMem |  1Mem | Used-Mem |
-------------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT      |      |      1 |        |      1 |00:00:00.01 |       2 |       |       |          |
|*  1 |  VIEW                 |      |      1 |      3 |      1 |00:00:00.01 |       2 |       |       |          |
|   2 |   WINDOW BUFFER       |      |      1 |      3 |      3 |00:00:00.01 |       2 |  2048 |  2048 | 2048  (0)|
|   3 |    HASH GROUP BY      |      |      1 |      3 |      3 |00:00:00.01 |       2 |   801K|   801K|  660K (0)|
|   4 |     TABLE ACCESS FULL | EMP  |      1 |     14 |     14 |00:00:00.01 |       2 |       |       |          |
-------------------------------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   1 - filter("SUM_SAL">"AVG")
```

계층 질의 활용

› 문제 1.

EMP 테이블에서 EMPNO 와 MGR 컬럼을 이용하여, 상사와 부하 직원의 정보를 계층적으로 검색하시오. (KING 을 기준으로 검색)

› 출력 결과

```
NAME                 LEVEL      EMPNO        MGR
--------------- ---------- ---------- ----------
KING                     1       7839
  JONES                  2       7566       7839
    SCOTT                3       7788       7566
      ADAMS              4       7876       7788
    FORD                 3       7902       7566
      SMITH              4       7369       7902
  BLAKE                  2       7698       7839
    ALLEN                3       7499       7698
    WARD                 3       7521       7698
    MARTIN               3       7654       7698
    TURNER               3       7844       7698
    JAMES                3       7900       7698
  CLARK                  2       7782       7839
    MILLER               3       7934       7782

14 rows selected.
```

› 답안. 계층 질의 사용

```
SQL> column name format a15
SQL> SELECT LPAD(' ',LEVEL*2-2)||ename AS NAME, LEVEL, empno, mgr
     FROM emp
     START WITH mgr is null
     CONNECT BY prior empno = mgr ;
```

Oracle Database 에서 사용 가능한 구문이며, START WITH 절을 이용하여 계층 질의의 시작 행을 식별하고 CONNECT BY 절을 이용하여 트리의 조건식을 정의한다. 이러한 계층 질의 구문은 ANSI-SQL 의 포맷은 아니다.

› 문제 2.

1번 문제의 결과에서 SCOTT 사원을 포함한 하위 직원을 제외한 계층 질의 결과를 검색하시오.

› 출력 결과

```
NAME                LEVEL      EMPNO        MGR
--------------- ---------- ---------- ----------
KING                    1       7839
  JONES                 2       7566       7839
    FORD                3       7902       7566
      SMITH             4       7369       7902
  BLAKE                 2       7698       7839
    ALLEN               3       7499       7698
    WARD                3       7521       7698
    MARTIN              3       7654       7698
    TURNER              3       7844       7698
    JAMES               3       7900       7698
  CLARK                 2       7782       7839
    MILLER              3       7934       7782

12 rows selected.
```

› 답안. 부정형 조건식 사용

```
SQL> SELECT LPAD(' ',LEVEL*2-2)||ename AS NAME, LEVEL, empno, mgr
     FROM emp
     WHERE ename != 'SCOTT'
     START WITH mgr is null
     CONNECT BY prior empno = mgr ;
NAME                LEVEL      EMPNO        MGR
--------------- ---------- ---------- ----------
KING                    1       7839
  JONES                 2       7566       7839
     ADAMS              4       7876       7788
    FORD                3       7902       7566
      SMITH             4       7369       7902
...

13 rows selected.
```

```
SQL> SELECT LPAD(' ',LEVEL*2-2)||ename AS NAME, LEVEL, empno, mgr
     FROM emp
     START WITH mgr is null
     CONNECT BY prior empno = mgr AND ename != 'SCOTT' ;
```

WHERE 절은 트리 구조를 생성한 이후에 조건 비교를 진행한다. 때문에 WHERE 절의 부정형 비교는 전체 가지(Branch) 제거를 수행할 수 없다. 하위 계층에 대한 제거도 함께 해야 한다면 CONNECT BY 절에서의 조건 비교를 사용한다.

문제 3.

EMP 테이블에서 KING부터 시작되는 계층 질의 결과를 다음과 같이 검색한다.

Worker는 현재 노드의 ENAME 컬럼이며, Manager는 상위 노드의 ENAME 컬럼을 출력한다.

› 출력 결과

```
Tree                 LEVEL Worker     Manager
--------------- ---------- ---------- ----------
KING                     1 KING
  JONES                  2 JONES      KING
    SCOTT                3 SCOTT      JONES
      ADAMS              4 ADAMS      SCOTT
    FORD                 3 FORD       JONES
      SMITH              4 SMITH      FORD
  BLAKE                  2 BLAKE      KING
    ALLEN                3 ALLEN      BLAKE
    WARD                 3 WARD       BLAKE
    MARTIN               3 MARTIN     BLAKE
    TURNER               3 TURNER     BLAKE
    JAMES                3 JAMES      BLAKE
  CLARK                  2 CLARK      KING
    MILLER               3 MILLER     CLARK

14 rows selected.
```

› 답안. PRIOR 키워드 사용

```
SQL> SELECT LPAD(' ',LEVEL*2-2)||ename AS "Tree",
            level,
            ename AS "Worker", prior ename AS "Manager"
     FROM emp
     START WITH mgr is null
     CONNECT BY prior empno = mgr ;
```

상위 노드의 컬럼이 필요할 때 PRIOR 키워드를 이용하여 참조가 가능하다.

문제 4.

1번 문제의 결과를 NAME 컬럼을 기준으로 정렬된 결과를 검색하시오.

단, 계층 구조를 훼손하지 않도록 한다.

› 출력 결과

```
NAME                LEVEL      EMPNO        MGR
--------------- ---------- ---------- ----------
KING                    1       7839
  BLAKE                 2       7698       7839
    ALLEN               3       7499       7698
    JAMES               3       7900       7698
    MARTIN              3       7654       7698
    TURNER              3       7844       7698
    WARD                3       7521       7698
  CLARK                 2       7782       7839
    MILLER              3       7934       7782
  JONES                 2       7566       7839
    FORD                3       7902       7566
      SMITH             4       7369       7902
    SCOTT               3       7788       7566
      ADAMS             4       7876       7788

14 rows selected.
```

› 답안. SIBLINGS 키워드 사용

```
SQL> SELECT LPAD(' ',LEVEL*2-2)||ename AS NAME, LEVEL, empno, mgr
     FROM emp
     START WITH mgr is null
     CONNECT BY prior empno = mgr
     ORDER BY ename ;
NAME                LEVEL      EMPNO        MGR
--------------- ---------- ---------- ----------
      ADAMS             4       7876       7788
    ALLEN               3       7499       7698
  BLAKE                 2       7698       7839
  CLARK                 2       7782       7839
    FORD                3       7902       7566
    JAMES               3       7900       7698
...

14 rows selected.

SQL> SELECT LPAD(' ',LEVEL*2-2)||ename AS NAME, LEVEL, empno, mgr
     FROM emp
     START WITH mgr is null
     CONNECT BY prior empno = mgr
     ORDER SIBLINGS BY ename ;
```

추가 실습

```
SQL> SELECT LPAD(' ',LEVEL*2-2)||ename AS NAME, LEVEL, empno, mgr, deptno
     FROM emp
     START WITH mgr is null
     CONNECT BY prior empno = mgr
     ORDER BY deptno ;
```

NAME	LEVEL	EMPNO	MGR	DEPTNO
CLARK	2	7782	7839	10
MILLER	3	7934	7782	10
KING	1	7839		10
FORD	3	7902	7566	20
ADAMS	4	7876	7788	20
SCOTT	3	7788	7566	20
JONES	2	7566	7839	20
SMITH	4	7369	7902	20
JAMES	3	7900	7698	30
TURNER	3	7844	7698	30
MARTIN	3	7654	7698	30
WARD	3	7521	7698	30
ALLEN	3	7499	7698	30
BLAKE	2	7698	7839	30

```
14 rows selected.
```

```
SQL> SELECT LPAD(' ',LEVEL*2-2)||ename AS NAME, LEVEL, empno, mgr, deptno
     FROM emp
     START WITH mgr is null
     CONNECT BY prior empno = mgr
     ORDER SIBLINGS BY deptno ;
```

NAME	LEVEL	EMPNO	MGR	DEPTNO
KING	1	7839		10
CLARK	2	7782	7839	10
MILLER	3	7934	7782	10
JONES	2	7566	7839	20
SCOTT	3	7788	7566	20
ADAMS	4	7876	7788	20
FORD	3	7902	7566	20
SMITH	4	7369	7902	20
BLAKE	2	7698	7839	30
ALLEN	3	7499	7698	30
WARD	3	7521	7698	30
MARTIN	3	7654	7698	30
TURNER	3	7844	7698	30
JAMES	3	7900	7698	30

```
14 rows selected.
```

문제 5.

계층 질의 문장을 활용하여 현재 LEVEL까지의 상위 이름을 함께 출력하며 구분자로 "/" 사용하는 문장을 작성하시오.

› 출력 결과

```
NAME            PATH
--------------- ------------------------------
KING            KING
  JONES         KING/JONES
    SCOTT       KING/JONES/SCOTT
      ADAMS     KING/JONES/SCOTT/ADAMS
    FORD        KING/JONES/FORD
      SMITH     KING/JONES/FORD/SMITH
  BLAKE         KING/BLAKE
    ALLEN       KING/BLAKE/ALLEN
    WARD        KING/BLAKE/WARD
    MARTIN      KING/BLAKE/MARTIN
    TURNER      KING/BLAKE/TURNER
    JAMES       KING/BLAKE/JAMES
  CLARK         KING/CLARK
    MILLER      KING/CLARK/MILLER

14 rows selected.
```

› 답안. SYS_CONNECT_BY_PATH 함수 사용

```
SQL> SELECT LPAD(' ',LEVEL*2-2)||ename AS NAME,
            SYS_CONNECT_BY_PATH(ENAME,'/') AS PATH
     FROM emp
     START WITH mgr is null
     CONNECT BY prior empno = mgr ;
NAME            PATH
--------------- ------------------------------
KING            /KING
  JONES         /KING/JONES
    SCOTT       /KING/JONES/SCOTT
      ADAMS     /KING/JONES/SCOTT/ADAMS
    FORD        /KING/JONES/FORD
      SMITH     /KING/JONES/FORD/SMITH
...

14 rows selected.

SQL> SELECT LPAD(' ',LEVEL*2-2)||ename AS NAME,
            LTRIM(SYS_CONNECT_BY_PATH(ENAME,'/'),'/') AS PATH
     FROM emp
     START WITH mgr is null
     CONNECT BY prior empno = mgr ;
```

문제 6.

EMP 테이블에서 계층 질의 결과를 검색하면서 최상의 루트 사원 급여와의 차이를 계산하시오.

› 출력 결과

```
NAME                LEVEL      EMPNO        MGR        SAL       DIFF
--------------- ---------- ---------- ---------- ---------- ----------
KING                    1       7839                  5000          0
  JONES                 2       7566       7839       2975       2025
    SCOTT               3       7788       7566       3000       2000
      ADAMS             4       7876       7788       1100       3900
    FORD                3       7902       7566       3000       2000
      SMITH             4       7369       7902        800       4200
  BLAKE                 2       7698       7839       2850       2150
    ALLEN               3       7499       7698       1600       3400
    WARD                3       7521       7698       1250       3750
    MARTIN              3       7654       7698       1250       3750
    TURNER              3       7844       7698       1500       3500
    JAMES               3       7900       7698        950       4050
  CLARK                 2       7782       7839       2450       2550
    MILLER              3       7934       7782       1300       3700

14 rows selected.
```

› 답안. CONNECT_BY_ROOT 사용

```
SQL> SELECT LPAD(' ',LEVEL*2-2)||ename AS NAME,
            level, empno, mgr, sal,
            CONNECT_BY_ROOT sal - sal AS diff
     FROM emp
     START WITH mgr is null
     CONNECT BY prior empno = mgr ;
```

CONNECT_BY_ROOT 를 이용하여 최상위 루트의 컬럼을 검색할 수 있다. START WITH 의 조건식을 통해 하나의 루트만이 정의되고 있으므로 KING 의 급여를 이용하여 계산이 가능하다.

```
SQL> SELECT LPAD(' ',LEVEL*2-2)||ename AS NAME,
            level, empno, mgr, sal, CONNECT_BY_ROOT sal AS root_sal
     FROM emp
     START WITH mgr is null CONNECT BY prior empno = mgr ;
NAME                LEVEL      EMPNO        MGR        SAL   ROOT_SAL
--------------- ---------- ---------- ---------- ---------- ----------
KING                    1       7839                  5000       5000
  JONES                 2       7566       7839       2975       5000
    SCOTT               3       7788       7566       3000       5000
...

14 rows selected.
```

문제 7.

EMP 테이블에서 부하 직원을 가지고 있지 않은 사원을 다음과 같이 검색하시오.

계층 질의를 활용하여 최상위 루트까지의 각 사원의 이름을 "/"로 구분하고, 최하위 노드의 사원 이름과 LEVEL 값을 출력한다.

› 출력 결과

```
PATH                          ENAME           LEVEL
----------------------------- ---------- ----------
/KING/BLAKE/WARD              WARD                3
/KING/BLAKE/JAMES             JAMES               3
/KING/BLAKE/ALLEN             ALLEN               3
/KING/CLARK/MILLER            MILLER              3
/KING/BLAKE/MARTIN            MARTIN              3
/KING/BLAKE/TURNER            TURNER              3
/KING/JONES/FORD/SMITH        SMITH               4
/KING/JONES/SCOTT/ADAMS       ADAMS               4

8 rows selected.
```

› 답안. CONNECT_BY_ISLEAF 사용

```
SQL> SELECT SYS_CONNECT_BY_PATH(ename, '/') AS path,
            ename, level
     FROM emp
     WHERE CONNECT_BY_ISLEAF = 1
     START WITH mgr is null
     CONNECT BY prior empno = mgr
     ORDER BY level ;


SQL> SELECT SYS_CONNECT_BY_PATH(ename, '/') AS path, ename, level, CONNECT_BY_ISLEAF
     FROM emp
     START WITH mgr is null
     CONNECT BY prior empno = mgr ;
PATH                          ENAME           LEVEL CONNECT_BY_ISLEAF
----------------------------- ---------- ---------- -----------------
/KING                         KING                1                 0
/KING/JONES                   JONES               2                 0
/KING/JONES/SCOTT             SCOTT               3                 0
/KING/JONES/SCOTT/ADAMS       ADAMS               4                 1
/KING/JONES/FORD              FORD                3                 0
/KING/JONES/FORD/SMITH        SMITH               4                 1
/KING/BLAKE                   BLAKE               2                 0
/KING/BLAKE/ALLEN             ALLEN               3                 1
...

14 rows selected.
```

문제 8.

다음의 명령문을 실행하고, 생성된 CP_EMP 테이블에서 계층 질의 결과를 검색하시오.

```
SQL> DROP TABLE cp_emp PURGE ;
SQL> CREATE TABLE cp_emp AS SELECT * FROM emp ;
SQL> UPDATE cp_emp
     SET empno = 7788
     WHERE empno = 7876 ;
SQL> COMMIT ;
```

> 출력 결과

```
NAME                 LEVEL      EMPNO        MGR
--------------- ---------- ---------- ----------
KING                     1       7839
  JONES                  2       7566       7839
    SCOTT                3       7788       7566
    FORD                 3       7902       7566
      SMITH              4       7369       7902
  BLAKE                  2       7698       7839
    ALLEN                3       7499       7698
    WARD                 3       7521       7698
    MARTIN               3       7654       7698
    TURNER               3       7844       7698
    JAMES                3       7900       7698
  CLARK                  2       7782       7839
    MILLER               3       7934       7782

13 rows selected.
```

> 답안. NOCYCLE, CONNECT_BY_ISCYCLE 사용

```
SQL> SELECT LPAD(' ',LEVEL*2-2)||ename AS NAME,
            LEVEL, empno, mgr
     FROM cp_emp
     START WITH mgr is null
     CONNECT BY prior empno = mgr ;
```

```
ERROR:
ORA-01436: CONNECT BY loop in user data
no rows selected
```

```
SQL> SELECT ename, empno, mgr FROM cp_emp WHERE empno = 7788 ;
```

```
ENAME           EMPNO        MGR
---------- ---------- ----------
SCOTT            7788       7566
ADAMS            7788       7788
```

CP_EMP 테이블은 UPDATE 를 통해 ADAMS 의 EMPNO 가 7788 로 수정되었다. 때문에 CONNECT BY 의 조건식이 ADAMS 의 행까지 오면 무한 루프를 수행하는 상황이 발생하기 때문에 문장은 실행되지 못한다. 이러한 무한 루프의 수행을 막기 위해 NOCYCLE 키워드를 사용하면 문제가 되는 노드는 제외한 계층 질의를 수행한다. CONNECT_BY_ISCYCLE 함수는 무한 루프 유무를 확인하는데 사용된다.

```
SQL> SELECT LPAD(' ',LEVEL*2-2)||ename AS NAME,
            LEVEL, empno, mgr, CONNECT_BY_ISCYCLE
     FROM cp_emp
     START WITH mgr is null
     CONNECT BY NOCYCLE prior empno = mgr ;
```

NAME	LEVEL	EMPNO	MGR	CONNECT_BY_ISCYCLE
KING	1	7839		0
JONES	2	7566	7839	0
SCOTT	**3**	**7788**	**7566**	**1**
FORD	3	7902	7566	0
SMITH	4	7369	7902	0
BLAKE	2	7698	7839	0
ALLEN	3	7499	7698	0
WARD	3	7521	7698	0
MARTIN	3	7654	7698	0
TURNER	3	7844	7698	0
JAMES	3	7900	7698	0
CLARK	2	7782	7839	0
MILLER	3	7934	7782	0

```
13 rows selected.

SQL> DROP TABLE cp_emp PURGE ;
```

문제 9.

FLIGHTS 테이블을 검색하면 출발지와 도착지 사이의 비행시간을 확인할 수 있다. 각 출발지에서 경유해서 최종 목적지까지 가는 전체 비행시간을 다음과 같이 검색한다.

```
SQL> SELECT * FROM flights;

DEPARTURE            DESTINATION          FLIGHT_TIME
-------------------- -------------------- -----------
San Jose             Los Angeles                  1.3
New York             Boston                       1.1
Los Angeles          New York                     5.8
```

> 출력 결과

```
DEPARTURE            DESTINATION          ROUTE                                    FLIGHTTIME
-------------------- -------------------- ---------------------------------------- ----------
Los Angeles          New York             Los Angeles/New York                            5.8
Los Angeles          Boston               Los Angeles/New York/Boston                     6.9
New York             Boston               New York/Boston                                 1.1
San Jose             Los Angeles          San Jose/Los Angeles                            1.3
San Jose             New York             San Jose/Los Angeles/New York                   7.1
San Jose             Boston               San Jose/Los Angeles/New York/Boston            8.2

6 rows selected.
```

> 답안 1. 계층 질의와 분석 함수 활용

```
SQL> SELECT departure, destination, level, flight_time,
            CONNECT_BY_ROOT departure AS root_depart,
            SYS_CONNECT_BY_PATH(destination,'/') AS path
     FROM flights
     START WITH departure = 'San Jose'
     CONNECT BY departure = PRIOR destination ;

DEPARTURE    DESTINATION       LEVEL FLIGHT_TIME ROOT_DEPART  PATH
------------ ------------ ---------- ----------- ------------ ------------------------------
San Jose     Los Angeles           1         1.3 San Jose     /Los Angeles
Los Angeles  New York              2         5.8 San Jose     /Los Angeles/New York
New York     Boston                3         1.1 San Jose     /Los Angeles/New York/Boston
```

START WITH 절을 이용하여 하나의 출발지를 대상으로 계층 질의를 수행하면 위와 같이 검색할 수 있다. 이러한 START WITH는 반드시 하나의 행만을 검색하도록 조건식을 정의해야 하는 것은 아니다. 때문에 문장은 다음과 같이 변경할 수 있고, 동일한 루트 노드와 출발지가 같은 그룹끼리 비행시간의 합을 계산한다.
CONNECT_BY_ROOT와 SYS_CONNECT_BY_PATH 등의 함수는 다양한 계산 결과를 생성할 때 매우 유용하게 사용될 수 있으므로 사용법을 숙지한다.

```
SQL> SELECT CASE level WHEN 1 THEN departure
                           ELSE CONNECT_BY_ROOT departure END AS departure,
           destination,
           CONNECT_BY_ROOT departure || SYS_CONNECT_BY_PATH(destination,'/') AS route,
           SUM(flight_time) OVER(PARTITION BY CONNECT_BY_ROOT departure
                                 ORDER BY level) flighttime
     FROM flights
     CONNECT BY departure = PRIOR destination
     ORDER BY departure, flighttime ;
```

> 답안 2. Recursive WITH 절 사용 (11gR2 NF)

```
SQL> WITH flt ( departure, destination, route, flighttime )
     AS ( SELECT departure, destination, departure|| '/' ||destination, flight_time
          FROM flights
          UNION ALL
          SELECT s.departure,
                 d.destination,
                 s.route || '/' || d.destination , s.flighttime + d.flight_time
          FROM flt s, flights d
          WHERE s.destination = d.departure )
     SELECT departure, destination, route, flighttime
     FROM flt
     ORDER BY departure, flighttime ;
```

DEPARTURE	DESTINATION	ROUTE	FLIGHTTIME
Los Angeles	New York	Los Angeles/New York	5.8
Los Angeles	Boston	Los Angeles/New York/Boston	6.9
New York	Boston	New York/Boston	1.1
San Jose	Los Angeles	San Jose/Los Angeles	1.3
San Jose	New York	San Jose/Los Angeles/New York	7.1
San Jose	Boston	San Jose/Los Angeles/New York/Boston	8.2

6 rows selected.

Oracle 11gR2 부터는 ANSI-SQL 에서 지원하는 Recursive WITH 절을 사용할 수 있다.
WITH 절에 UNION ALL 로 분기된 두 개의 Query Block 이 존재한다. 이때 위에 정의된 Query Block 을 Anchor 멤버, 아래 정의된 Query Block 을 Recursive 멤버라 한다. Anchor 멤버는 START WITH 절의 역할을 수행하고, Recursive 멤버는 CONNECT BY 절의 역할을 수행한다. 아래와 같이 Anchor 멤버에 조건식을 정의하면 좀 더 쉽게 이해할 수 있다. Recursive WITH 절에서는 SYS_CONNECT_BY_PATH 와 같은 함수가 없으므로 필요한 식을 직접 만들어서 사용할 수 있다.

```
SQL> WITH flt ( departure, destination, route, flighttime )
     AS ( SELECT departure, destination, departure|| '/' ||destination, flight_time
          FROM flights WHERE departure = 'San Jose'
          UNION ALL
          SELECT s.departure, d.destination,
                 s.route || '/' || d.destination , s.flighttime + d.flight_time
          FROM flt s, flights d
          WHERE s.destination = d.departure )
     SELECT departure, destination, route, flighttime
     FROM flt
     ORDER BY flighttime ;
```

DEPARTURE	DESTINATION	ROUTE	FLIGHTTIME
San Jose	Los Angeles	San Jose/Los Angeles	1.3
San Jose	New York	San Jose/Los Angeles/New York	7.1
San Jose	Boston	San Jose/Los Angeles/New York/Boston	8.2

문제 10.

1 번 문제의 결과를 Recursive WITH 절을 이용하여 생성하시오. (결과와 동일한 정렬 상태 보장)

› 출력 결과

```
NAME                 LEVEL      EMPNO        MGR
--------------- ---------- ---------- ----------
KING                     1       7839
  JONES                  2       7566       7839
    SCOTT                3       7788       7566
      ADAMS              4       7876       7788
    FORD                 3       7902       7566
      SMITH              4       7369       7902
  BLAKE                  2       7698       7839
    ALLEN                3       7499       7698
    WARD                 3       7521       7698
    MARTIN               3       7654       7698
    TURNER               3       7844       7698
    JAMES                3       7900       7698
  CLARK                  2       7782       7839
    MILLER               3       7934       7782

14 rows selected.
```

› 답안

```
SQL> WITH htree(hlevel, ename, empno, mgr)
     AS ( SELECT 1, ename, empno, mgr
          FROM emp WHERE mgr IS NULL
          UNION ALL
          SELECT hlevel + 1, e.ename, e.empno, e.mgr
          FROM emp e, htree h
          WHERE e.mgr = h.empno )
    SELECT LPAD(' ',hlevel * 2 - 2)||ename AS name, hlevel, empno, mgr FROM htree ;

NAME                HLEVEL      EMPNO        MGR
--------------- ---------- ---------- ----------
KING                     1       7839
  JONES                  2       7566       7839
  BLAKE                  2       7698       7839
  CLARK                  2       7782       7839
    SCOTT                3       7788       7566
    FORD                 3       7902       7566
    ALLEN                3       7499       7698
    WARD                 3       7521       7698
    MARTIN               3       7654       7698
    TURNER               3       7844       7698
    JAMES                3       7900       7698
    MILLER               3       7934       7782
      ADAMS              4       7876       7788
      SMITH              4       7369       7902

14 rows selected.
```

Recursive WITH 절에서는 계층 질의에서 사용 가능한 LEVEL 이 없다. 때문에 임의의 계산식으로 LEVEL 값을 생성해야 하며, 검색된 결과의 정렬 상태는 너비를 기준으로 (레벨이 증가되는 값 순서. 즉, 상위 노드의 결과들을 먼저 출력) 진행된다. 때문에 생성된 HLEVEL 값을 이용하여 들여쓰기 형태로 포맷을 바꾸면 계층 질의의 형태와는 유사하지만 틀린 결과가 검색될 수 있다. 또한 ORDER BY 절을 이용한 정렬도 소용이 없다.

```
SQL> WITH htree(hlevel, ename, empno, mgr)
     AS ( SELECT 1, ename, empno, mgr
          FROM emp WHERE mgr IS NULL
          UNION ALL
          SELECT hlevel + 1, e.ename, e.empno, e.mgr
          FROM emp e, htree h
          WHERE e.mgr = h.empno )
    SELECT LPAD(' ',hlevel * 2 - 2)||ename AS name, hlevel, empno, mgr FROM htree
    ORDER BY name;
NAME               HLEVEL      EMPNO        MGR
--------------- ---------- ---------- ----------
      ADAMS             4       7876       7788
      SMITH             4       7369       7902
    ALLEN               3       7499       7698
    FORD                3       7902       7566
    JAMES               3       7900       7698
    MARTIN              3       7654       7698
    MILLER              3       7934       7782
...

14 rows selected.
```

Recursive WITH 절에 SEARCH 절을 이용하면 동일한 레벨에서 특정 컬럼을 기준으로 정렬 상태를 조정할 수 있다. (계층 질의에서 SIBLING 사용과 유사함)

```
SQL> WITH htree(hlevel, ename, empno, mgr)
     AS ( SELECT 1, ename, empno, mgr
          FROM emp WHERE mgr IS NULL
          UNION ALL
          SELECT hlevel + 1, e.ename, e.empno, e.mgr
          FROM emp e, htree h
          WHERE e.mgr = h.empno )
    SEARCH DEPTH FIRST BY empno ASC SET idx
    SELECT LPAD(' ',hlevel * 2 - 2)||ename AS name, hlevel, empno, mgr FROM htree ;
SQL> WITH htree(hlevel, ename, empno, mgr)
```

```
     AS ( SELECT 1, ename, empno, mgr
          FROM emp WHERE mgr IS NULL
          UNION ALL
          SELECT hlevel + 1, e.ename, e.empno, e.mgr
          FROM emp e, htree h
          WHERE e.mgr = h.empno )
    SEARCH DEPTH FIRST BY empno ASC SET idx
    SELECT LPAD(' ',hlevel * 2 - 2)||ename AS name, hlevel, empno, mgr, idx FROM htree ;
```

NAME	HLEVEL	EMPNO	MGR	IDX
KING	1	7839		1
JONES	2	7566	7839	2
SCOTT	3	7788	7566	3
ADAMS	4	7876	7788	4
FORD	3	7902	7566	5
SMITH	4	7369	7902	6

```
...
```

14 rows selected.

```
SQL> WITH htree(hlevel, ename, empno, mgr)
     AS ( SELECT 1, ename, empno, mgr
          FROM emp WHERE mgr IS NULL
          UNION ALL
          SELECT hlevel + 1, e.ename, e.empno, e.mgr
          FROM emp e, htree h
          WHERE e.mgr = h.empno )
    SEARCH DEPTH FIRST BY empno DESC SET idx
    SELECT LPAD(' ',hlevel * 2 - 2)||ename AS name, hlevel, empno, mgr, idx FROM htree ;
```

NAME	HLEVEL	EMPNO	MGR	IDX
KING	1	7839		1
CLARK	2	7782	7839	2
MILLER	3	7934	7782	3
BLAKE	2	7698	7839	4
JAMES	3	7900	7698	5
TURNER	3	7844	7698	6
MARTIN	3	7654	7698	7

```
...
```

14 rows selected.

Entity Relationship Diagram (ERD)

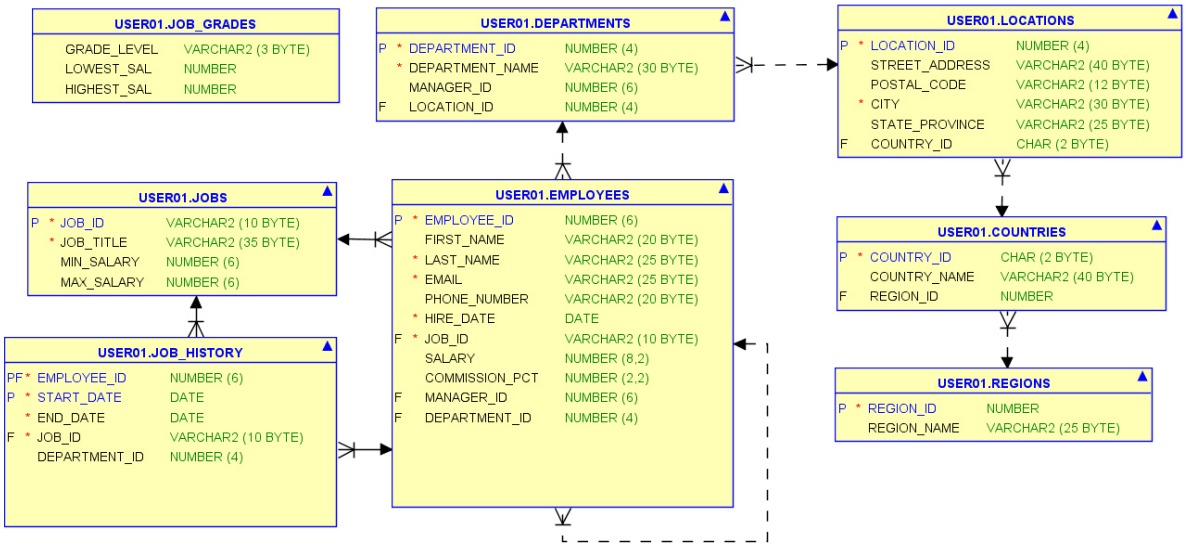

그림 1. 테이블 관계 (1:M, Self-Reference)

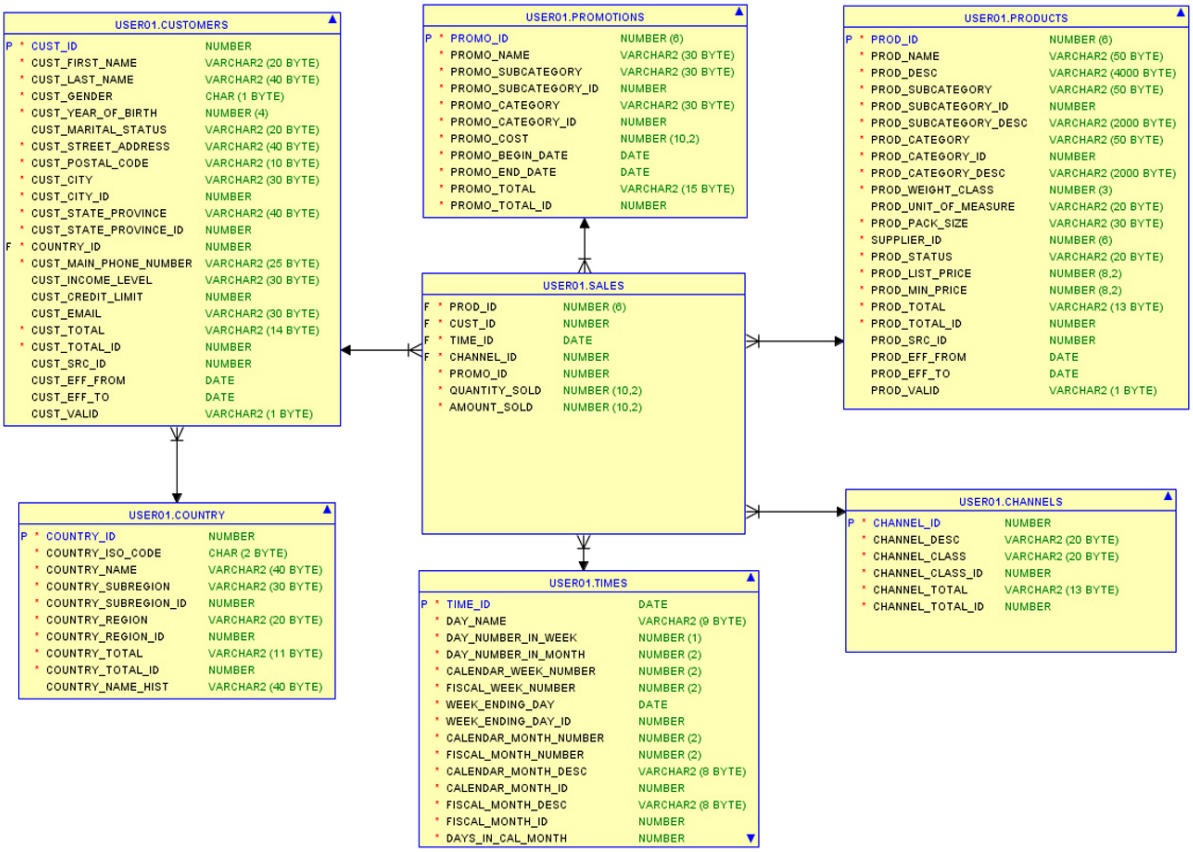

그림 2. 스타 스키마 (Star Schema, Snowflake Schema)

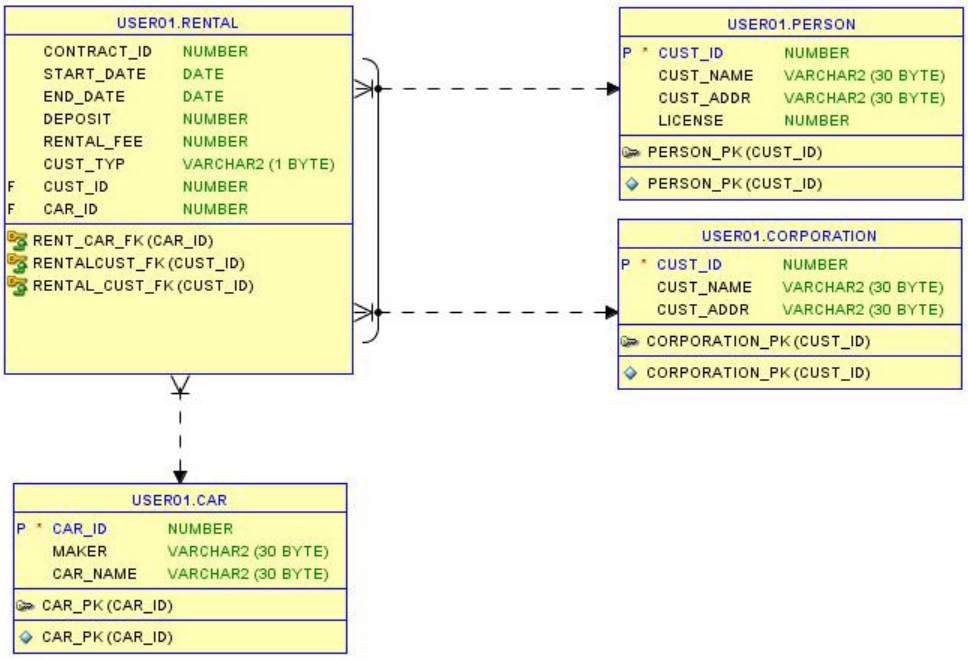

그림 3. 배타적 모델 (Arch Relationship)

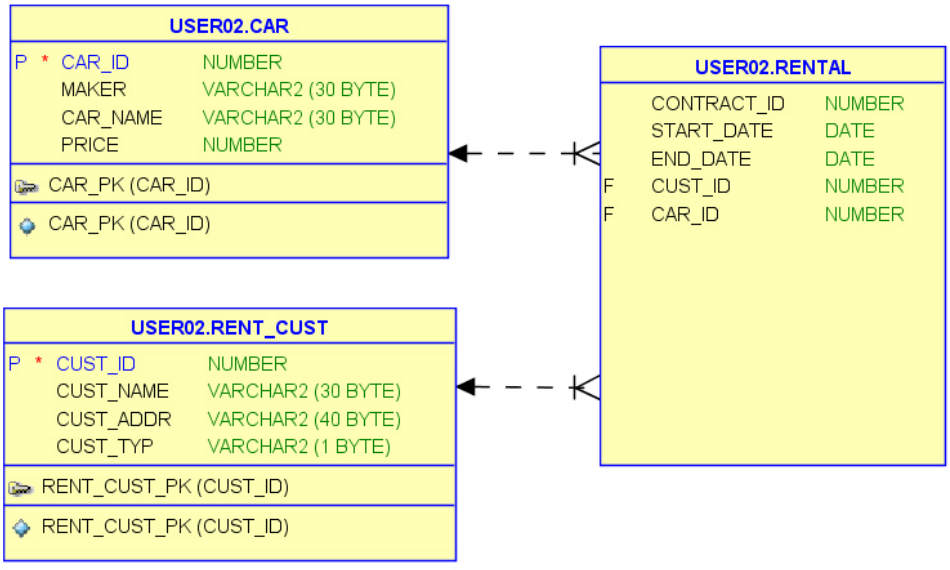

그림 4. M:1 관계 (M:M 관계 해소)