SQL초보에서 Schema Object까지

교재 샘플

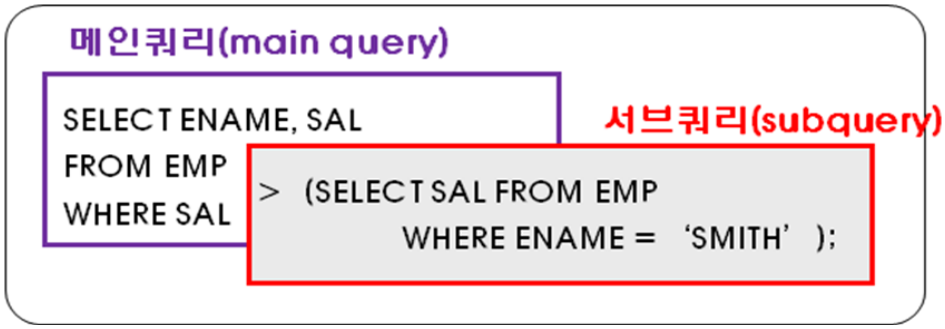
오라클자바커뮤니티(ojcedu.com, ojc.asia)

5. 서브 쿼리(SUB QUERY)

5.1 서브 쿼리(SUB QUERY) 개요

서브 쿼리는 SELECT한 결과를 조건 비교시 사용하거나 UPDATE, INSERT등에 사용되는 내장된 SELECT 문장이며 메인 쿼리 이전에 한번만 실행 된다. 테이블 자체의 데이터에 의존하는 조건으로 테이블의 행을 검색할 필요가 있을 때 서브쿼리는 아주 유용하게 이용될 수 있다.

EMP 테이블에서 SMITH의 급여보다 급여가 많은 사람을 추출하는 경우



괄호로 싸인 부분이 서브 쿼리 인데 Inner Query or Sub Query 라고 하며 Inner Query의 결과를 비교 조건으로 사용하는 외부에 있는 것을 Main Query or Outer Query 라고 한다. 서브쿼리(Sub Query)는 메인 쿼리 실행 전에 한번씩 실행되며 그 결과가 메인 쿼리(Main Query)에 전달된다.

[서브 쿼리 지침]

- 서브 쿼리는 괄호로 싸야 한다.
- 단일 행 및 복수 행 서브 쿼리는 연산자의 우측에 나타나야 한다.
- 서브 쿼리에는 ORDER BY 절을 포함 할 수 없다.

5.2 단일행 서브쿼리(Single-Row Sub Query)

서브 쿼리에서 하나의 결과가 반환되는 구조이며 이와 같은 구조에서 사용되는 연산자는 단일 행 연산자(> , >= , < , <= , = , <>) 이다.

--EMP 테이블에서 "SMITH"와 같은 JOB을 가지는 사원들의 ENAME, SAL, JOB을 추출하려 한다고 하자. 만약 SMITH 사원의 JOB 이 "CLERK" 이라는 것을 알고 있다면 다음과 같이 쉽게 할 수 있을 것이다. 그러나 아래와 같은 경우 "SMITH" 사원의 JOB이 바뀌게 되면 어떻게 할 것인가? "SMITH"의 JOB을 바꿀 때 마다 기억을 한다는 것은 어려운 일다. 그렇다고 질의(Query) 문 을 아래와 같이 매번 두 번 만드는 것도 번거러운 일 이다.

```
SQL> select job from emp
  2   where ename = 'SMITH';

JOB
---------
CLERK

SQL> select ename, sal, job from emp
  2   where job = 'CLERK';

ENAME                SAL JOB
---------- ---------- ---------
SMITH                800 CLERK
ADAMS               1100 CLERK
JAMES                950 CLERK
MILLER              1300 CLERK
```

-- 위의 두 예문을 합친 단일 행 서브 쿼리 예문이다.
-- 서브 쿼리는 아래와 같이 테이블 자체의 데이터에 의존하는 비교 조건으로 데이터를 검색 할 때 유용 하다.

```
SQL> select ename, sal, job from emp
  2   where job = (select job from emp
  3                where ename = 'SMITH');

ENAME                SAL JOB
---------- ---------- ---------
SMITH                800 CLERK
ADAMS               1100 CLERK
JAMES                950 CLERK
MILLER              1300 CLERK
```

--아래의 예문은 EMP 테이블에서 급여가 가장 적은 사원의 이름과 급여를 출력 하는 예문이다.

```
SQL> select ename, sal from emp
  2   where sal = (select min(sal) from emp);

ENAME                SAL
---------- ----------
SMITH                800
```

-- EMP 테이블에서 부서코드가 30번인 부서의 급여 최소값보다 해당 부서 급여의 최소값이 큰 부서만 출력하되 부서 순으로 오름차순으로 정렬 하시오. 이 예문에서 기억 해야 하는 사실은 첫째 서브쿼리는 WHERE절 뿐 아니라 HAVING절에서도 사용 가능하며 둘째 WHERE절이 각 행에 조건을 줘서 선택되는 행을 제어 하듯이 HAVING절은 GROUP BY에 의해 그룹화 되는 그룹에 조건을 줄 때 사용하는 것으로 반드시 HAVING은 GROUP BY 뒤에 와야 하며, 셋째 HAVING이 사용되면 대부분 GROUP BY가 있지만 GROUP BY 없는 HAVING의 사용도 가능 하다. 넷째 ORDER BY절은 SELECT문의 마지막에 오며, 다섯째 SELECT절에 그룹 함수 외의 컬럼이 나타나면 반드시 GROUP BY절에 해당 컬럼이 나타나야 한다.

```
SQL> select deptno, min(sal) from emp
  2  group by deptno
  3  having min(sal) > (select min(sal) from emp
  4                     where deptno = 30)
  5  order by deptno;

 DEPTNO     MIN(SAL)
---------- ----------
10           1300
```

--GROUP BY 없이 사용되는 HAVING 예문(테이블 전체를 하나의 그룹으로 간주)
--EMP 테이블의 SAL의 최대값은 5000 이다. 급여 평균은 2073.21429인 상태

```
SQL> select max(sal) from emp;

  MAX(SAL)
----------
      5000
SQL> select avg(sal) from emp;

  AVG(SAL)
----------
2073.21429
```

--SAL의 최대값을 구하는데 SAL의 평균이 2000보다 크다고 했으므로 현재의 하나 밖에 없는 그룹(테이블 전체)에는 SAL의 평균이 2073이므로 MAX(SAL)은 5000이 된다.

```
SQL> select max(sal) from emp
  2  having avg(sal) > 2000;

  MAX(SAL)
----------
      5000
```

```
--EMP Table에는 여러 종류의 직무(JOB)가 있다. 다음 예문은 각 JOB의 평균 급여가 최대인 JOB과 그 평균 급여를 출력 하는 예문이다.
SQL> select job, avg(sal) from emp
  2  group by job
  3  having avg(sal)   = (select max(avg(sal)) from emp
  4                          group by job);

JOB              AVG(SAL)
--------- ----------
PRESIDENT          5000

--부서코드가 10인 사원들 중 최대/최소 급여를 받는 사원의 이름, 급여, 부서를 출력
SQL> select ename, sal, deptno from emp
  2  where deptno = 10
  3  and sal in ( (select max(sal) from emp where deptno = 10),
  4                  (select min(sal) from emp where deptno = 10)
  5                );

ENAME                SAL       DEPTNO
---------- ---------- ----------
KING               5000           10
MILLER             1300           10

--SMITH와 같은 JOB, 같은 부서를 가지는 사원의 이름, 직무, 부서를 출력하는데 SMITH는 출력하지 마시오
SQL> select ename, job, deptno from emp
  2  where job = (select job from emp where ename = 'SMITH')
  3  and   deptno = (select deptno from emp where ename = 'SMITH')
  4  and ename != 'SMITH';

ENAME      JOB              DEPTNO
---------- --------- ----------
ADAMS      CLERK                20
```

5.2 복수행 서브쿼리(Multi-Row Sub Query)

서브 쿼리에서 여러 건의 결과가 반환되는 구조 이다. 이와 같은 구조에서 사용되는 연산자는 IN, ANY, SOME(ANY와 동일), ALL, EXISTS 등과 같은 복수 행 연산자 이다. 이 연산자들은 이전의 SQL 연산자 부분에 자세히 나와 있으니 참조하길 바라며 아래의 예문을 따라 하면서 이해해 보자.

--EMP 테이블에서 각 부서별로 급여를 가장 적게 받는 사원의 부서, 이름, 급여를 출력 하는 예문이다. IN 연산자 오른쪽 서브 쿼리에서 리턴되는 행(ROW)은 여러 건이다. 아마도 부서별로 가장 적은 급여가 추출되어 Outer Query와 IN 연산자에 의해 비교되는 것이다. 결국 부서별로 급여를 가장 적게 받는 사원을 선택 하게 되는데 IN연산자는 OR로 풀어 쓸 수 있음을 기억 하자.

```
SQL> select deptno, ename, sal from emp
  2   where (deptno, sal) in (select deptno, min(sal) from emp
  3                                group by deptno);

DEPTNO ENAME                SAL
------ ---------- ----------
     30 JAMES                950
     20 SMITH                800
     10 MILLER              1300
```

-- 위와 같은 WHERE 비교를 PAIRWISE방식이라 한다. 즉 칼럼을 쌍으로 묶어서 비교하는 것이다. 만약 아래처럼 쿼리문을 작성한다면 예상치 못한 결과가 나올 수도 있다. 현재 EMP 테이블은 다행히도 10번 부서의 최소급여인 1300을 다른 부서사원들이 가지고 있지 않아서 PAREWISE 방식으로 안 하더라도 결과는 같이 나온다. 만약 20번 부서 사원이 1300을 가지고 있다면 그 사원도 출력될 것이다.

```
select deptno, ename, sal from emp
where sal in (select min(sal) from emp
                  group by deptno)
  and deptno in (select distinct deptno from emp)
```

--간단히 ANY의 개념에 대해 이해하자. 여러 값 중 하나하고만 조건을 만족시키면 되므로 OR로 풀어 쓸 수 있다.

```
SQL> SELECT empno, sal
  2   FROM    emp
  3   WHERE   sal > ANY (2000, 3000, 4000);

     EMPNO          SAL
---------- ----------
      7782         2450
```

```
      7698       2850
      7566       2975
      7788       3000
      7902       3000
      7839       5000

6 개의 행이 선택되었습니다.

SQL> SELECT empno, sal
  2  FROM    emp
  3  WHERE   sal > 2000 OR sal > 3000 OR sal > 4000;

     EMPNO        SAL
---------- ----------
      7782       2450
      7698       2850
      7566       2975
      7788       3000
      7902       3000
      7839       5000

6 개의 행이 선택되었습니다.
```

--아래 예문은 ANY를 이용한 서브 쿼리 예문이다. EMP 테이블에서 부서코드가 10번인 사원 급여의 임의 값보다 큰 급여를 가지는 사원의 이름, 급여를 출력하는 하는 것이다. 서브 쿼리에서 괄호안을 풀면 두번째 줄은 where sal > any (2450, 5000, 1300)의 의미가 되고 2450, 5000, 1300을 대입하여 Outer Query를 수행 했을 때 하나라도 만족하면 되는 것이므로, where sal > 1300의 의미와 같은 것이다. 만약 ANY를 사용하고 등호를 사용 했다면 IN의 의미와 같다는 것도 기억 하자. 즉 sal = any (2450, 5000, 1300)과 sal in (2450, 5000, 1300)과 같은 의미이다.

```
SQL> select sal from emp
  2  where deptno = 10;

       SAL
----------
      2450
      5000
      1300
```

```
SQL> select ename, sal, job from emp
  2   where sal > any (select sal from emp
  3                        where deptno = 10);

ENAME                SAL JOB
---------- ---------- ---------
ALLEN              1600 SALESMAN
JONES              2975 MANAGER
BLAKE              2850 MANAGER
CLARK              2450 MANAGER
SCOTT              3000 ANALYST
KING               5000 PRESIDENT
TURNER             1500 SALESMAN
FORD               3000 ANALYST

8 개의 행이 선택되었습니다.
```

--아래 예문은 sal = any (SELECT절) 과 sal in (SELECT절)과 같다는 것을 보이는 예문이다.

```
SQL> select ename, sal, job from emp
  2   where sal = any (select sal from emp
  3                        where deptno = 10);

ENAME                SAL JOB
---------- ---------- ---------
MILLER             1300 CLERK
CLARK              2450 MANAGER
KING               5000 PRESIDENT

SQL> select ename, sal, job from emp
  2   where sal in (select sal from emp
  3                      where deptno = 10);

ENAME                SAL JOB
---------- ---------- ---------
MILLER             1300 CLERK
CLARK              2450 MANAGER
KING               5000 PRESIDENT
```

-- 아래 ALL 예문을 보면 ALL의 의미에 대해 이해가 될 것이다. 모든 값을 만족하려면 결국 AND로

모든 것을 비교하는 것이다.

```
SQL> SELECT empno, ename, sal
  2  FROM    emp
  3  WHERE   sal > ALL (2000, 3000, 4000);

     EMPNO ENAME                SAL
---------- ---------- ----------
      7839 KING                5000

SQL>   SELECT empno, ename, sal
  2    FROM    emp
  3    WHERE   sal > 2000 AND sal > 3000 AND sal > 4000;

     EMPNO ENAME                SAL
---------- ---------- ----------
      7839 KING                5000
```

--아래 예문도 ALL을 사용한 것이다. ALL은 서브 쿼리에서 리턴되는 데이터들 모두가 Outer Query에서 조건을 만족시켜야 하는 것이다. 즉 sal > all (SELECT절)의 경우 SELECT절에서 추출되는 자료의 최대값보다 많다는 의미이고, sal < all (SELECT절)의 경우 SELECT절에서 추출되는 자료의 최소값보다 적다는 의미이다.

```
SQL> select ename, sal, job from emp
  2    where sal > all (select sal from emp
  3                          where deptno = 20);

ENAME                SAL JOB
---------- ---------- ---------------
KING               5000 PRESIDENT
```

-- ALL을 ANY를 이용하여 변경했다.

```
SQL> select ename, sal, job from emp
  2  where not (sal <= any ( select sal from emp
  3                                where deptno = 20));

ENAME                SAL JOB
---------- ---------- ---------------
KING               5000 PRESIDENT

SQL> select ename, sal, job from emp
```

```
  2    where sal < all (select sal from emp
  3                     where deptno = 20);
선택된 레코드가 없습니다.

-- EXISTS실습을 위해 테이블을 하나 만들자.(EMP 테이블에서 10번 부서 사원들로 EMP_10 생성)
SQL> create table emp_10 as select * from emp where deptno = 10;
테이블이 생성되었습니다.

-- 아래는 EMP 테이블 사원 중 EMP_10에 존재하는 사원들을 추출하는 예문이다.
SQL> SELECT empno, ename, sal
  2  FROM emp e
  3  WHERE EXISTS (SELECT 1 FROM emp_10 WHERE empno = e.empno);

     EMPNO ENAME                SAL
---------- ---------- ----------
      7782 CLARK                2450
      7839 KING                 5000
      7934 MILLER               1300


-- 아래는 UPDATE문에서 EXISTS를 사용하는 예문이다.
-- EMP 테이블에서 입사년도가 1980년인 사원이 존재하면 수당을 0으로 UPDATE
SQL> UPDATE emp e
  2  SET comm = 0
  3  WHERE EXISTS (SELECT 1
  4                FROM emp
  5                WHERE EMPNO = e.empno
  6                AND    to_char(hiredate,'YYYY') = '1980' );

1 행이 갱신되었습니다.

SQL> rollback;
롤백이 완료되었습니다.
```

5.3 상관 서브쿼리(Correlated Sub Query)

서브 쿼리와 메인쿼리간에 서로 상관 참조 작용하는 쿼리 이다. 즉 Inner Query에서 Outer Query의 어떤 컬럼을 이용하는 경우 이다. 일반적인 Query의 경우 서브 쿼리의 결과를 메인에서는 단순히 이용만 하지만 상관 서브 쿼리에서는 서브 쿼리가 메인 쿼리의 값을 이용하여 값을 구하면 그 값을 다시 메인 쿼리에서 이용하는 구조이다. 예를 들어 서브 쿼리 에서 10번 부서에서 가장 많은 급여를 받는 사람, 20번 부서에서 가장 많은 급여를 받는 사람, 30번 부서에서 같은 결과를 구하고자 한다면 부서만 다르고 같은 내용을 입력 시켜야 하므로 이 경우 상관 쿼리로 만들어 처리하면 편리 하다.

[상관 서브쿼리 실행순서]

(1) Outer query를 실행하여 행을 하나 읽는다.
(2) (1)에서 읽은행의 data를 이용하여 서브쿼리에서 필요한 값을 넣고 Subquery를 수행한다.
(3) (2)의 결과값으로 Outer query의 WHERE절을 평가하여 읽은 행의 선택여부를 결정한다. 참이면 데이터 추출, 아니면 버리고 Outer Query의 다음 레코드를 읽음
(4) Outer query의 테이블에 행이 없을 때까지 (1)-(3)을 반복 수행한다.

--각 부서의 최대 급여를 받는 사원의 부서코드, 이름, 급여를 출력하는데 부서코드 순으로 오름차순 정렬하여 출력하는 예문 첫번째 상관서브쿼리로 구현

```
SQL> select deptno, ename, sal from emp e1
     where sal = (select max(sal) from emp e2
                  where e1.deptno = e2.deptno)
     order by deptno;

     DEPTNO ENAME                SAL
---------- ---------- ----------
         10 KING                5000
         20 SCOTT               3000
         20 FORD                3000
         30 BLAKE               2850
```

-- 두번째 Pairwise 서브쿼리로 구현

```
SQL> select ename, sal, deptno
       from emp
       where (deptno, sal ) in (select deptno, max(sal) from emp
                                group by deptno)
       order by deptno;
```

-- 세번째 조인, 인라인뷰로 구현

```
SQL> select e1.ename, e1.sal, e1.deptno
     from emp e1, ( select deptno, max(sal) as msal from emp group by deptno ) e2
     where e1.deptno = e2.deptno
```

```
      and   e1.sal = e2.msal
      order by e1.deptno;
```

--각 직무(JOB)별로 최대 급여를 받는 사원의 직무, 이름, 급여를 출력하는데 직무 명 순으로 오름차순 정렬하여 출력하는 예문

```
SQL> select job, ename, sal from emp e1
     where sal = (select max(sal) from emp e2
                        where e1.job = e2.job)
     order by deptno;

JOB           ENAME                SAL
--------- ---------- ----------
PRESIDENT    KING                 5000
CLERK          MILLER              1300
MANAGER     JONES                 2975
ANALYST       SCOTT                3000
ANALYST       FORD                 3000
SALESMAN    ALLEN                1600

6 개의 행이 선택되었습니다.
```

--해당 부서의 평균 급여보다 급여를 적게 받는 사원의 부서, 사원명, 급여를 출력하되 부서 순으로 오름차순 정렬하여 출력 하는 예문

```
SQL> select deptno, ename, sal from emp e1
     where sal < (select avg(sal) from emp e2
                        where e1.deptno = e2.deptno)
     order by deptno;

     DEPTNO ENAME                SAL
---------- ---------- ----------
          10 CLARK               2450
          10 MILLER              1300
          20 SMITH                800
          20 ADAMS               1100
          30 WARD                1250
          30 JAMES                950
          30 TURNER              1500
          30 MARTIN              1250
```

```
--EMP 테이블에서 급여가 높은 사원 5명을 출력 하는 예문
SQL> select ename, sal from emp e1
     where 5 > (select count(*) from emp e2
                    where e2.sal > e1.sal)
     order by sal desc;

ENAME                 SAL
---------- ----------
KING                 5000
SCOTT                3000
FORD                 3000
JONES                2975
BLAKE                2850

--각 부서별로 급여가 높은 사람 2명씩 출력 하는 예문
SQL> select deptno, ename, sal from emp e1
     where 2 > (select count(*) from emp e2
                    where e2.sal > e1.sal
                    and e2.deptno = e1.deptno)
     order by deptno, sal desc;

     DEPTNO ENAME                SAL
---------- ---------- ----------
         10 KING                5000
         10 CLARK               2450
         20 SCOTT               3000
         20 FORD                3000
         30 BLAKE               2850
         30 ALLEN               1600

6 개의 행이 선택되었습니다.

--사원이 한명이라도 있는 부서명 출력
SQL> SELECT dname FROM dept d
     WHERE EXISTS ( SELECT 1 FROM emp WHERE deptno = d.deptno);

DNAME
--------------
```

```
ACCOUNTING
RESEARCH
SALES
```

5.4 Scalar SubQuery

하나의 Scalar값을 나타내기 위해 SELECT구문을 SELECT LIST, WHERE절, ORDER BY절, DML등에 사용하는 서브쿼리를 지칭하며 데이터건수가 적을 경우 조인 방식보다 유리하다. 유효한 수식이 사용될 수 있는 곳이라면 어디든 사용가능하며 반드시 하나의 결과만 되돌려야 한다.

```
--사원테이블과 부서테이블을 조인하여 사번, 사원명, 부서코드, 부서명을 추출한다고 하자.
SQL> select empno, ename, emp.deptno, dname from emp, dept
  2   where emp.deptno = dept.deptno;

     EMPNO ENAME        DEPTNO DNAME
---------- ---------- ------ --------------
      7782 CLARK              10 ACCOUNTING
      ......
      7654 MARTIN             30 SALES

14 개의 행이 선택되었습니다.

-- 스칼라 서브쿼리로 바꾸면
SQL> select empno, ename, deptno, (select dname from dept where emp.deptno = dept.deptno)
  2   from emp;

     EMPNO ENAME        DEPTNO (SELECTDNAMEFR
---------- ---------- ------ --------------
      7369 SMITH              20 RESEARCH
      ......
      7934 MILLER             10 ACCOUNTING

14 개의 행이 선택되었습니다.

-- 부서별로 사원수를 출력하는 쿼리를 스칼라 서브쿼리를 이용해 보자.
SQL> select deptno, dname,
```

```
  2           (select nvl(count(*),0) from emp where emp.deptno = dept.deptno) emp_cnt
  3    from dept        ;

DEPTNO DNAME                EMP_CNT
------ -------------- ----------
    10 ACCOUNTING                3
    20 RESEARCH                  5
    30 SALES                     6
    40 OPERATIONS                0
```

-- insert 구문에 스칼라 서브쿼리를 사용해보자.

```
SQL>   create table emp_summary (
  2      sum_sal number,
  3      avg_sal number,
  4      max_sal number,
  5      min_sal number
  6    );

테이블이 생성되었습니다.

SQL>   insert into emp_summary (
  2              sum_sal,
  3              avg_sal,
  4              max_sal,
  5              min_sal )
  6    values (
  7                (select sum(sal) from emp),
  8                (select avg(sal) from emp),
  9                (select max(sal) from emp),
 10                (select min(sal) from emp)
 11             );

1 개의 행이 만들어졌습니다.

SQL> commit;
```

-- 아래처럼 order by절에서도 사용 가능하다. EMP테이블에서 부서이름으로 정렬

```
SQL> select empno, ename, sal
  2    from   emp e
```

```
  3   order by   (   select dname from dept d
  4                    where e.deptno = d.deptno);

     EMPNO ENAME                SAL
---------- ---------- ----------
      7782 CLARK                2450
      7839 KING                 5000
      7934 MILLER               1300
      7566 JONES                2975
      ......
```

5.5 인라인뷰(IN_LINE VIEW)

서브 쿼리는 FROM절에서도 가능 한데 이와 같은 것을 뷰((View)는 뷰(View)인데 Create 명령어로 만들지 않고 해당 라인에 직접 기술한다고 해서 하여 인라인 뷰(InLine View)라고 일컫는다. 인라인 뷰는 Create 명령어를 이용하여 만들지 않으므로 필요한 시점에만 사용된다는 특징이 있다. 가령 어떤 테이블의 데이터 건수, 칼럼 개수가 많다고 했을 때 FROM절 다음에 전체 테이블을 기술 하면 쿼리실행시 수행속도에 악 영향을 미칠 수가 있으므로 FROM절 뒤에 테이블 데이터 중 필요한 행과 열만 선택 한다면 좀더 효율적인 쿼리 할 수 있을 것이다. 또한 인라인뷰는 조인 연산을 줄이기 위해 또는 분리된 쿼리를 하나의 쿼리에 모아서 사용하기 위해 사용한다.

```
--EMP 테이블에서 직무가 "SALESMAN"인 사람들의 이름, 부서명, 직무를 출력하는 예문
SQL> select ename, dname, job from (select ename, job, deptno from emp
  2                                    where job = 'SALESMAN') e, dept d
  3   where e.deptno = d.deptno;

ENAME      DNAME          JOB
---------- -------------- ---------
ALLEN      SALES          SALESMAN
WARD       SALES          SALESMAN
MARTIN     SALES          SALESMAN
TURNER     SALES          SALESMAN

--아래의 경우 인라인뷰를 이용하지 않은 예문이다.
SQL> select ename, dname, job from emp e, dept d
  2   where e.job = 'SALESMAN'
  3   and e.deptno = d.deptno;
```

```
ENAME      DNAME          JOB
---------- -------------- ---------
ALLEN      SALES          SALESMAN
WARD       SALES          SALESMAN
MARTIN     SALES          SALESMAN
TURNER     SALES          SALESMAN

--이번에는 WITH문을 이용해 보자.
SQL> with emp2 as (
  2      select ename, job, deptno from emp
  3      where job = 'SALESMAN'
  4  )
  5  select ename, dname, job from emp2 e, dept d
  6    where e.deptno = d.deptno;

ENAME      DNAME          JOB
---------- -------------- ---------------
ALLEN      SALES          SALESMAN
TURNER     SALES          SALESMAN
MARTIN     SALES          SALESMAN
WARD       SALES          SALESMAN
```

5.6 WITH 구문

ORACLE9i R2이후 사용 가능한 WITH구문은 이름이 부여된 서브쿼리 블록으로 global temporary tables, virtual table or an inline view 처럼 작동된다. 복잡한 SQL에서 동일 쿼리블록이 반복적으로 사용되는 경우 그 블록에 이름을 부여하여 한곳에서 정의하고 이를 재사용 할 수 있게 함으로서 쿼리문 코딩량도 줄이고 성능도 향상 시킬 수 있는데, WITH절을 이용하여 미리 이름을 부여해서 Query Block을 만들 수 있다. 자주 실행되는 경우 한번만 Parsing되고 실행 계획이 수립되므로 성능 향상에 도움이 된다.

WITH문은 주로 SELECT 문과 함께 사용 되며 Query Block을 미리 정할 수 있는데 자주 사용되는 QUERY의 블록이 고 비용인 경우 이용하면 성능상 장점이 있다.

--아래의 예문을 보면 WITH에 대해 이해가 될 것이다. 먼저 WITH를 사용하지 않은 예문이다.

```
-- 64초 소요
select e.empno, e.deptno, e.sal, d.dname from (
                    select empno, sal, deptno from myemp1 where sal > 5000000
                ) e, mydept1 d
where e.deptno = d.deptno
and     e.deptno = '1'
union
select e.empno, e.deptno, e.sal, d.dname from (
                    select empno, sal, deptno from myemp1 where sal > 5000000
                ) e, mydept1 d
where e.deptno = d.deptno
and     e.deptno = '2'
union
select e.empno, e.deptno, e.sal, d.dname from (
                    select empno, sal, deptno from myemp1 where sal > 5000000
                ) e, mydept1 d
where e.deptno = d.deptno
and     e.deptno = '3'

-- WITH구를 사용한 예문
-- 33초 소요
with e as (
    select empno, sal, deptno from myemp1 where sal > 5000000
)
select e.empno, e.deptno, e.sal, d.dname from e, mydept1 d
where e.deptno = d.deptno
and     e.deptno = '1'
union
select e.empno, e.deptno, e.sal, d.dname from e, mydept1 d
where e.deptno = d.deptno
and     e.deptno = '2'
union
select e.empno, e.deptno, e.sal, d.dname from e, mydept1 d
where e.deptno = d.deptno
and     e.deptno = '3'

-- INSERT문에서 WITH구가 사용되는 예제
SQL> create table emp20 as select empno, ename from emp where 1 > 100;
테이블이 생성되었습니다.
```

```
SQL> INSERT INTO emp20
  2  SELECT * FROM (
  3     WITH dept20 AS (
  4        SELECT empno, ename FROM emp WHERE deptno = 20)
  5     SELECT * FROM dept20);

5 개의 행이 만들어졌습니다.

SQL> commit;
커밋이 완료되었습니다.
```

6. 조인(JOIN)

6.1 테이블 별명(Table Alias)

테이블 명칭이 너무 긴 경우 원래 테이블 명칭 대신 별도의 이름을 부여해서 사용하는데 이를 테이블 별명(Table Alias)이라고 한다. 보통 테이블의 이름이 긴 경우 직관적이고 짧은 이름의 별명(Alias)을 이용하여 Query를 간결히 하는데 많이 이용되고 이 별명을 이용하여 같은 테이블이 조인되는 경우 칼럼 이름의 모호함을 방지 할 수 있다(EMP테이블의 MGR 칼럼이 참조하는 EMPNO 칼럼과의 조인, 사원별 관리자 이름 출력하는 경우). 이전에 배웠던 칼럼별명(Column Alias)과 비슷하게 생각하면 된다.

```
--주로 많이 사용하는 방식이다.
SQL> select e.ename, e.sal, e.deptno, d.dname
     from emp e, dept d
     where e.deptno = d.deptno
     and e.deptno = 10;

ENAME                SAL       DEPTNO DNAME
---------- ---------- ---------- --------------
CLARK              2450         10 ACCOUNTING
KING               5000         10 ACCOUNTING
MILLER             1300         10 ACCOUNTING
```

```
SQL> select "사원".ename, "사원".sal, "사원".deptno, "부서".dname
     from emp "사원", dept "부서"
     where "사원".deptno = "부서".deptno
     and "사원".deptno = 10;

ENAME                SAL       DEPTNO DNAME
---------- ---------- ---------- --------------
CLARK              2450         10 ACCOUNTING
KING               5000         10 ACCOUNTING
MILLER             1300         10 ACCOUNTING
```

--아래의 경우 1라인의 deptno의 경우 EMP와 DEPT 테이블 두 테이블에 공통으로 있는 칼럼이다. ename, sal, dname등은 컬럼 이름 앞에 소속을 밝히지 않아도 어느 테이블에 있는지를 알 수 있지만 deptno와 같은 경우는 어느 테이블의 것인지를 알기가 모호하다.

```
SQL> select ename, sal, deptno, dname
     from emp e, dept d
     where e.deptno = d.deptno
     and e.deptno = 10;
select ename, sal, deptno, dname
                           *
1행에 오류:
ORA-00918: 열의 정의가 애매합니다
```

6.2 카티션 프로덕트(Cartesian Product)

조인을 하는 경우 WHERE절을 기술 하지 않으면 FROM절 뒤에 기술된 각 테이블의 레코드 건수의 곱 만큼 결과가 나오게 되는데 이를 카티션 프로덕트(Cartesian Product) 라고 한다. 즉 테이블간의 조인에 있어 조인되는 조건을 기술하지 않거나 생략하는 경우에 발생한다.

```
SQL> select count(*) from emp;

  COUNT(*)
----------
        14
```

```
SQL> select count(*) from dept;

  COUNT(*)
----------
         4

--WHERE절을 기술 하지 않았으므로 결과는 (EMP의 데이터 건수) * (DEPT의 데이터 건수)가 된다.
SQL> select count(*) from emp, dept;

  COUNT(*)
----------
        56

--아래의 경우 결과로 나타나는 칼럼은 EMP, DEPT의 모든 칼럼이 나타나게 된다.
SQL> select * from emp, dept;

     EMPNO ENAME      JOB              MGR HIREDATE         SAL       COMM
---------- ---------- --------- ---------- -------- ---------- ----------
    DEPTNO     DEPTNO DNAME          LOC
---------- ---------- -------------- -------------
      7369 SMITH      CLERK            7902 80/12/17        800         96
        20         10 ACCOUNTING       NEW YORK

      7499 ALLEN      SALESMAN          7698 81/02/20        1600        300
        30         10 ACCOUNTING       NEW YORK
..... 중략.....

56 개의 행이 선택되었습니다.
```

6.3 EquiJoin, Non EquiJoin

조인(Join)이란? 두개의 테이블에 대해 연관된 행들을 조인 칼럼을 기준으로 비교하여 새로운 행 집합을 만드는 것으로 각 테이블간 의미있는 행들을 연결하는 개념이다. 일반적인 조인은 내부조인(Inner Join)으로 두 테이블의 공통칼럼(조인칼럼)을 기준으로 데이터를 추출하는 것 이며 EQUI JOIN, NON EQUI JOIN이 있다.

조인과 서브쿼리는 유사하지만 중요한 차이는 두테이블간의 관계에 있다. 조인은 동일한 등급, 레벨에서 이루어 지는 것이지만 서브쿼리는 주종관계, 즉 하나는 메인쿼리(MAIN QUERY)이고 하나는 부속쿼리(SUB QUERY) 이다.

EquiJoin : 흔히 사용하는 조인의 형태이며 조건 절에 조인되는 두 테이블의 컬럼을 Equal 연산자(=)로 연결하는 경우 이다. 주로 Primary Key와 Foreign Key 컬럼이 서로 조인될 때 이용되는 형태 이다.

Non EquiJoin : 조인 조건이 Equal 연산자 이외의 >, >=, <, <=, <>, BETWEEN ... AND 연산자들을 이용하여 조인을 하는 경우 이다. 한 테이블의 어떠한 칼럼도 조인 할 테이블의 칼럼에 직접적으로 일치하지 않는 경우에 사용 한다.

-- EMP, DEPT 테이블의 DEPTNO라는 공통 컬럼을 Equal(=) 연산자로서 Equi Join 하는 예이다.
-- EMP 테이블에는 부서명(DNAME) 칼럼을 가지고 있지 않으므로 조인을 해야만 부서명을 얻을 수 있다. EMP 테이블의 DEPTNO는 Foreign Key이고 DEPT 테이블에서 DEPTNO는 Primary Key 이다. 아래 예문에서 두 테이블의 공통 칼럼인 DEPTNO가 서로 Primary Key, Foreign Key 관계이므로 USING구를 이용한 형태도 가능하다는 것을 참고로 알아 두자. 아래 4가지 표현은 모두 같은 쿼리문이다.

```
SQL> select ename "사원명", dname "부서명"
  2  from emp e, dept d
  3  where e.job = 'SALESMAN'
  4  and e.deptno = d.deptno;

사원명      부서명
---------- --------------
ALLEN      SALES
WARD       SALES
MARTIN     SALES
TURNER     SALES


SQL> select ename "사원명", dname "부서명"
  2  from emp e join dept d
  3  using (deptno)
  4  where job = 'SALESMAN';

사원명      부서명
---------- --------------
ALLEN      SALES
```

```
WARD           SALES
MARTIN         SALES
TURNER         SALES

SQL> select ename "사원명", dname "부서명"
  2   from emp e join dept d
  3   on e.deptno = d.deptno
  4   where e.job = 'SALESMAN';

사원명         부서명
---------- --------------
ALLEN          SALES
WARD           SALES
MARTIN         SALES
TURNER         SALES
```

-- 오라클9i 이후 EQUI JOIN을 자연조인(Natural Join)이라하고 공통 조인칼럼을 자동으로 알아서 내부적인 조인문을 만들어 조인하므로 ON절이나 USING문을 사용하지 않아도 된다. 주의할 점은 SELECT절에 JOIN컬럼이 출현시 테이블 별칭을 사용하면 안 된다.

```
SQL> select ename "사원명", dname "부서명"
  2    from emp e natural join dept d
  3    where e.job = 'SALESMAN';

사원명         부서명
---------- --------------
ALLEN          SALES
TURNER         SALES
MARTIN         SALES
WARD           SALES


SQL> select ename "사원명", dname "부서명", e.deptno
  2    from emp e natural join dept d
  3    where e.job = 'SALESMAN';

select ename "사원명", dname "부서명", e.deptno
                                       *
1행에 오류:
ORA-25155: NATURAL 조인에 사용된 열은 식별자를 가질 수 없음
```

--EMP 테이블에서 10번 부서 사원들의 이름, 급여 , 급여등급(GRADE)을 SALGRADE를 참고하여 출력하는 예문이다. SALGRADE 테이블에는 등급별로 하한 값과 상한 값을 가지고 있기 때문에 각 사원들의 등급을 알기 위해서는 SALGRADE 테이블과 Non-Equi Join을 해야 한다. 즉 조인을 하는 각 테이블의 칼럼이 직접적으로 일치 하는 않는 경우이다.

```
SQL> select e.ename "이름", e.sal "급여", s.grade "등급"
  2  from emp e, salgrade s
  3  where e.deptno = 10
  4  and e.sal between s.losal and s.hisal;

이름                 급여         등급
---------- ---------- ----------
MILLER               1300          2
CLARK                2450          4
KING                 5000          5

SQL>  select e.ename "이름", e.sal "급여", s.grade "등급"
  2  from emp e join salgrade s
  3  on e.deptno = 10
  4  and e.sal between s.losal and s.hisal;

이름                 급여         등급
---------- ---------- ----------
MILLER               1300          2
CLARK                2450          4
KING                 5000          5
```

6.4 Self Join

한 개의 테이블을 두 개의 별도의 테이블처럼 이용하여 서로 조인 하는 형태인데 반드시 테이블 Alias를 사용하게 되는 조인 이다. 보통 한 행(ROW)에 두개의 레코드 정보를 보여 주고자 할 때 유용하다. (EMP테이블에서 사원이름과, 관리자이름을 같이 출력하는 경우)

```
--EMP 테이블에서 사번, 이름, 관리자이름을 출력하라.(관리자도 사원테이블에 존재하여 EMPNO를 가진다)
```

```
SQL> select "사원".empno, "사원".ename, "관리자".ename
  2    from emp "사원", emp "관리자"
  3    where "사원".mgr = "관리자".empno;

     EMPNO ENAME      ENAME
---------- ---------- ----------
      7902 FORD       JONES
      7788 SCOTT      JONES
      7844 TURNER     BLAKE
      7499 ALLEN      BLAKE
      7521 WARD       BLAKE
      7900 JAMES      BLAKE
      7654 MARTIN     BLAKE
      7934 MILLER     CLARK
      7876 ADAMS      SCOTT
      7698 BLAKE      KING
      7566 JONES      KING
      7782 CLARK      KING
      7369 SMITH      FORD

13 개의 행이 선택되었습니다.
```

--아래의 CREATE TABLE 구문에서 SAWON_ID 칼럼은 주키(Primary Key)로 선언 하였고 MANAGER_ID 칼럼은 NULL을 허용하나 만약 값이 들어 온다면 SAWON_ID에 있는 값이 들어 와야 한다는 의미의 외래키(Foreign Key) 조건을 주어 테이블을 만들었다.
아래의 실습용 테이블인 SAWON 이 이미 있다는 오류가 뜨면 DROP TABLE SAWON; 구문으로 먼저 삭제 한 후 CREATE TABLE 예문을 실행하기 바란다.

```
SQL> create table sawon (
  2   sawon_id number(5,0) not null primary key,
  3   name varchar2(10) not null,
  4   buseo varchar2(20) ,
  5   manager_id number(5,0) constraint fk_sawon references sawon(sawon_id)
  6   );

테이블이 생성되었습니다.

SQL> insert into sawon (sawon_id, name, buseo) values (1, '가길동', '관리부');
```

```
1 개의 행이 만들어졌습니다.
SQL> insert into sawon (sawon_id, name, buseo, manager_id)
  2  values (2, '나길동', '영업부', 1);
1 개의 행이 만들어졌습니다.
SQL> insert into sawon (sawon_id, name, buseo, manager_id)
  2  values (3, '다길동', '관리부', 1);
1 개의 행이 만들어졌습니다.
SQL> commit;
커밋이 완료되었습니다.
```

--sawon 테이블에서 사원리스트를 출력하는데 사원ID, 이름, 부서, 관리자명 형태로 출력 하고자 한다. 이 예문의 핵심은 관리자의 이름을 어떻게 가지고 올 것인가 이다. 관리자도 결국 사원이므로 sawon 테이블에 존재 한다는 원리에 착안하여 같은 테이블을 테이블 별칭(Table Alias)를 사용하여 두 개의 테이블로 구분하는데 하나는 사원테이블 , 또 하나를 관리자테이블로 분리하여 사원 테이블의 관리자 아이디(MANAGER_ID)와 관리자 테이블의 사원아이디(sawon_id)를 조인하면 사원 별 관리자의 이름을 가지고 올 수 있을 것이다. 아래 예문에서 1번 사원은 관리자가 없는 관계로 화면에 출력되지 않았다. 물론 1번 사원도 나타나게 하려면 Outer Join을 이용하면 되는데 이 부분은 Outer Join 부분에서 설명하겠다.

```
SQL> select * from sawon;

  SAWON_ID NAME       BUSEO                MANAGER_ID
---------- ---------- -------------------- ----------
         1 가길동     관리부
         2 나길동     영업부                        1
         3 다길동     관리부                        1

SQL> select "사원".sawon_id "사원ID",
  2         "사원".name "사원명",
  3         "사원".buseo "부서명",
  4         "관리자".name "관리자명"
  5  from sawon "사원", sawon "관리자"
  6  where "사원".manager_id = "관리자".sawon_id;

    사원ID 사원명     부서명               관리자명
---------- ---------- -------------------- ----------
         2 나길동     영업부               가길동
         3 다길동     관리부               가길동
```

6.5 Outer Join

내부 조인(EquiJoin, Non EquiJoin, Natural Join, Self Join)을 하다보면 한쪽 테이블의 값이 일치하지 않아 조인조건을 만족하지 않는 경우에도 결과로 출력해야 되는 경우가 있는데 이럴때 외부조인(Outer Join)을 사용하면 된다. 외부조인은 LEFT OUTER JOIN, RIGTH OUTER JOIN, FULL OUTER JOIN이 있다.

내부 조인은 조인 조건을 만족하는 행들만 나타나게 된다. 그러므로 앞의 Self Join 예문처럼 조인 조건을 만족하지 않는다면(NULL 값을 가지는 경우) 해당 레코드는 출력되지 않을 것이다. Outer Join이란 조인에서 한쪽 테이블의 행에 대하여 다른 쪽 테이블에 일치하는 행이 없더라도 다른 쪽 행을 NULL로 하여 행을 반환하게 하는 것이다.

오라클에서 Outer Join의 연산자는 "(+)"이며 MS의 SQL Server의 경우는 "*" 이다. 그럼 "(+)"를 조인되는 테이블 중 어디에다 둘 것 인가 이다. 오라클의 경우 모두 출력 되어야 하는 테이블의 반대쪽에 "(+)"를 표시하며 SQL Server의 경우 모두 출력 되어야 하는 곳 쪽에 "*"를 표시 한다. 모두 출력되어야 한다는 것의 의미는 조인 조건을 만족하지 않다고 해도 해당 레코드를 표시한다는 것을 의미하며, 조인되는 테이블의 해당 칼럼은 NULL로 표시된다.

앞의 Self Join 예문에서 사원 별 관리자 이름을 출력하는데 1번 사원의 레코드가 출력되지 않았던 것을 기억 할 것이다. 이 경우 어디에 "(+)"를 하여야 할까? 사원 쪽이 다 출력 되어야 한다. 즉 관리자가 있든지, 없든지 사원 리스트의 모든 사원은 나와야 하는 것이다. 그렇다면 관리자 쪽에 (+) 표시를 하면 될 것이다. 다음과 같이 SQL문을 수정하면 된다. 마지막에 (+)만 추가 되었다.

```
SQL> select "사원".sawon_id "사원ID",
  2            "사원".name "사원명",
  3            "사원".buseo "부서명",
  4            "관리자".name "관리자명"
  5   from sawon "사원", sawon "관리자"
  6   where "사원".manager_id = "관리자".sawon_id(+);

    사원ID 사원명      부서명               관리자명
---------- ---------- -------------------- ----------
         1 가길동      관리부
         2 나길동      영업부               가길동
         3 다길동      관리부               가길동
```

또는 아래와 같이 표현해도 된다. 아래의 표기는 ANSI 표준에 부합 하므로 MS SQL Server, MySQL

등에서 똑같이 사용해도 무방 하다. 아래에서 유심히 볼 부분은 from 절 다음 부분이다. left outer join 이라고 한 것은 왼쪽 부분인 "사원" 테이블의 내용은 조인에 맞지 않더라도 관리자이름은 NULL을 채우면서 다 출력 되어야 한다는 것이다. left의 반대인 right Outer Join도 있다. 그러나 의미는 같은 것이니 예문을 통해 이해 하기 바란다.

```
SQL> select "사원".sawon_id "사원ID",
  2            "사원".name "사원명",
  3            "사원".buseo "부서명",
  4            "관리자".name "관리자명"
  5   from sawon "사원" left outer join sawon "관리자"
6          on "사원".manager_id = "관리자".sawon_id;

사원ID 사원명       부서명                관리자명
---------- ---------- -------------------- ----------
         1 가길동       관리부
         2 나길동       영업부                가길동
         3 다길동       관리부                가길동
```

만약 여러분이 나중에 MS의 SQL Server등에서 질의를 하는 경우라면 다음과 같이 하면 된다. 물론 오라클에서 아래와 같이 사용하면 오류가 난다.

```
 SQL> select "사원".sawon_id "사원ID",
  2            "사원".name "사원명",
  3            "사원".buseo "부서명",
  4            "관리자".name "관리자명"
  5   from sawon "사원", sawon "관리자"
6          where "사원".manager_id *= "관리자".sawon_id;
```

--이해가 되었다면 예문을 통해 Outer Join을 확실히 이해 하도록 하자. Outer Join을 이해 하지 못한 상태에서 실제 업무에서 조인을 사용한다면 대량의 데이터를 처리 할 때 조인 처리 미숙으로 인해 누락되는 데이터가 나와 원인을 찾으려면 어려운 경우를 만날 수 있을 것이다. 반드시 이해하고 넘어 가도록 하자.

--실습을 위해 SALES(매출 테이블)와 CUSTOMER(고객 테이블)을 먼저 만들자. 혹시 이미 만들어져 있다면 DROP TABLE 구문으로 삭제 후 다시 만들자.

```
SQL> create table customer (
  2   cust_id varchar2(4) not null primary key,
  3   name varchar2(10),
```

```
  4   tel varchar2(20)
  5   );

테이블이 생성되었습니다.

SQL> create table sales (
  2   cust_id varchar2(4) not null constraints fk_cust_id references customer(cust_id),
  3   ilja varchar2(8) not null,
  4   amt number(7,0) not null
5         );

SQL> insert into customer values ('1004','가길동','111-1111');
1 개의 행이 만들어졌습니다.
SQL> insert into customer values ('1005','나길동','222-2222');
1 개의 행이 만들어졌습니다.
SQL> insert into customer values ('1006','다길동','333-3333');
1 개의 행이 만들어졌습니다.
SQL> insert into sales values ('1004','20140301',10000);
1 개의 행이 만들어졌습니다.
SQL> insert into sales values ('1006','20140310',20000);
1 개의 행이 만들어졌습니다.
SQL> commit;
커밋이 완료되었습니다.

SQL> select * from sales;

CUST ILJA                 AMT
---- -------- ----------
1004 20140301         10000
1006 20140310         20000

SQL> select * from customer;

CUST NAME          TEL
---- ---------- --------------------
1004 가길동        111-1111
1005 나길동        222-2222
1006 다길동        333-3333
```

--고객성명, 매출일자, 매출액의 형태로 고객 테이블에 등록된 모든 고객에 대해 출력해야 한다고 하자. 즉 매출이 없더라도 매출액이 0원으로 매출리스트에 나타나야 한다는 의미이다. 매출내역 테이블에는 고객의 이름이 없으니 고객 테이블과 조인을 하여 고객의 이름을 가지고 와야 한다. 그런데? 매출을 일으키지 않은 1005번 고객이 누락된 것이 보이는가? EquiJoin 등을 구사하는 경우 종니 조건을 만족하지 않는 행은 출력 되지 않는 것이 기본이다. 그럼 어떻게 할 것 이다. 물론 이런 경우를 위해 Outer Join이 있는 것이다. 아래의 3가지 형태는 같은 기능을 하는 예문 이다. ON을 사용 한것과 USING을 사용 한 두 번째, 세 번째 예문은 ANSI 표준에 부합되므로 다른 데이터베이스등에서 동일하게 사용 할 수 있다. 주로 첫번째 형태를 많이 이용하며 두번째, 세번째 예문의 경우 from절 다음을 다음과 같이 from sales s inner join customer c 와 같이 써도 무방하다.

```
SQL> select c.name, s.ilja, s.amt
  2  from sales s, customer c
  3  where s.cust_id = c.cust_id;

NAME       ILJA            AMT
---------- -------- ----------
가길동     20140301        10000
다길동     20140310        20000

SQL> select c.name, s.ilja, s.amt
  2  from sales s join customer c
  3  on s.cust_id = c.cust_id;

NAME       ILJA            AMT
---------- -------- ----------
가길동     20140301        10000
다길동     20140310        20000

SQL>  select c.name, s.ilja, s.amt
  2  from sales s join customer c
  3  using (cust_id);

NAME       ILJA            AMT
---------- -------- ----------
가길동     20140301        10000
다길동     20140310        20000
```

--아래는 Left Outer Join의 예문이다. 왼쪽에 있는 것은 조인 조건이 맞지 않더라도 다 나타나야 한다는 것이다. 아래의 두 예문은 같은 결과를 나타내는 예문이다. 두번째에서 (+)의 위치를 잘 보라,

CUSTOMER쪽에 있으니 SALES쪽이 조건에 맞지 않는 컬럼이 있더라도 모두 나타나야 한다는 것이다.

```
SQL> select c.name, s.ilja, s.amt
  2  from sales s left outer join customer c
  3  on s.cust_id = c.cust_id;

NAME       ILJA            AMT
---------- -------- ----------
가길동     20140301        10000
다길동     20140310        20000

SQL> select c.name, s.ilja, s.amt
  2  from sales s, customer c
  3  where s.cust_id = c.cust_id(+);

NAME       ILJA            AMT
---------- -------- ----------
가길동     20140301        10000
다길동     20140310        20000
```

--아래 예문은 Right Outer Join의 예문이다. 바로 아래 예문을 보면 오른쪽에 있는 CUSTOMER 테이블의 내용은 조인에 맞지 않더라도 다 나와야 한다는 의미이다. 즉 매출이 없더라도 고객테이블에 등록되어 있는 고객 이라면 출력이 된다는 의미이다. 그 다음 예문을 보라 (+)의 위치가 어디인가? SALES쪽이므로 CUSTOMER가 조인 조건에 맞지 않더라도 모두 출력 되어야 한다는 의미이다.

```
SQL> select c.name, s.ilja, nvl(s.amt, 0)
  2  from sales s right outer join customer c
  3  on s.cust_id = c.cust_id;

NAME       ILJA       NVL(S.AMT,0)
---------- -------- ------------
가길동     20140301          10000
다길동     20140310          20000
나길동                           0

SQL>  select c.name, s.ilja, nvl(s.amt, 0)
  2  from sales s, customer c
  3  where s.cust_id(+) = c.cust_id;

NAME       ILJA       NVL(S.AMT,0)
```

```
---------- -------- ------------
가길동       20140301          10000
나길동                            0
다길동       20140310          20000
```

--다음은 Full Outer Join에 관련된 예문이다. 조인시 양쪽 테이블의 데이터를 전부 출력 하기 위해 사용하는 형식이다. 즉 서로 조인 조건에 맞지 않는 것이 있더라도 모두 표시 하라는 의미인데 두번째 예문에서 주의 할점은 (+) 표시는 한곳에만 할 수 있는데, (+)형태로 Full Outer Join을 표시 하지 못한다는 것을 기억하자.

--실습을 위해 sales 테이블에 고객테이블에 없는 고객으로 한건입력하자.

```
SQL>> insert into sales values ('1007','20140301',20000);


SQL> select c.name, s.ilja, nvl(s.amt, 0)
  2  from sales s full outer join customer c
  3  on s.cust_id = c.cust_id;

NAME          ILJA       NVL(S.AMT,0)
---------- -------- ------------
가길동       20040301          10000
다길동       20040310          20000
나길동                            0
             20040301          20000

-- 오라클에서 양쪽 모두에 (+)사용은 오류.
SQL> select c.name, s.ilja, nvl(s.amt, 0)
  2  from sales s, customer c
  3  where s.cust_id(+) = c.cust_id(+);
where s.cust_id(+) = c.cust_id(+)
                     *
3행에 오류:
ORA-01468: outer-join된 테이블은 1개만 지정할 수 있습니다
```

6.6 계층형 쿼리(Hierarchical Query)

■ EMP 테이블의 MGR 칼럼은 외래키로 자기자신 테이블 EMP의 EMPNO 값을 참조하는 관리자를 뜻하는 칼럼이다. 이렇게 계층관계가 있는 칼럼이 있는 경우 계층구조를 이용하여 데이터를 추출할 수 있는 방법이 있는데 계층적 질의문을 이용하면 된다.

[형식]

```
SELECT 칼럼
FROM 테이블
WHERE 조건
START WITH 조건
CONNECT BY [PRIOR] [NOCYCLE]
[ORDER SIBLINGS BY 칼럼, 칼럼]
```

START WITH : 시작 데이터를 지정한다.
CONNECT BY : 계층구조에서 다음에 연결될 데이터를 지정한다.
PRIOR : CONNECT BY 절에 이용되며 현재 읽은 칼럼을 지정한다.
PRIOR 자식 = 부모 (TOP DOWN 형태 출력)
PRIOR 부모 = 자식 (BOTTOM UP 형태 출력)
NOCYCLE : 데이터를 펼치면서 이미 나타났던 데이터가 다시 나타나는 경우 CYCLE이 형성되었다고 한다. 이때 오류가 발생하는데 NOCYCLE을 추가하면 사이클이 발생한 이후의 데이터를 출력하지 않는다.
ORDER SIBLINGS BY : 동일한 LEVEL인 경우 노드를 형제 노드라고 하고 형제노드들의 정렬순서를 지정한다.

■ 계층형 질의에서 사용되는 가상칼럼
LEVEL : 최상위 루트 데이터가 1, 하위로 갈수록 1씩 증가
CONNECT_BY_ISLEAF : 최하단 리프데이터이면 1, 아니면 0
CONNECT_BY_ISCYCLE : 해당 데이터가 조상으로써 존재하면 1, 아니면 0, 조상이란 자신으로부터 루트까지의 경로에 존재하는 데이터를 의미한다.

■ 계층형 쿼리에서 사용가능한 함수
SYS_CONNECT_BY_PATH (칼럼, 경로분리자): 루트데이터에서부터 전개할 데이터 까지의 경로를 표시한다.
CONNECT_BY_ROOT 칼럼 : 현재 전개할 데이터의 루트 데이터를 표시한다.

```
SELECT   EMPNO, ENAME,
         CONNECT_BY_ROOT ENAME,
```

```
          SYS_CONNECT_BY_PATH(ENAME, '/') 경로
FROM EMP
START WITH MGR IS NULL   --시작데이터를 지정
CONNECT BY PRIOR EMPNO = MGR; --상위(부모)의 EMPNO를 MGR값으로 가지는
                                데이터가 다음 연결될 데이터이다. 이 연결고리를 지정

7839        KING    KING    /KING
7566        JONES   KING    /KING/JONES
7788        SCOTT   KING    /KING/JONES/SCOTT
......
7844        TURNER KING     /KING/BLAKE/TURNER
7900        JAMES   KING    /KING/BLAKE/JAMES
7782        CLARK   KING    /KING/CLARK
7934        MILLER  KING    /KING/CLARK/MILLER

-- EMP 테이블에서 KING부터 시작하여 아래와 같은 결과를 출력하는 쿼리를 작성하세요.
-- 트리구조에서 위에 있는 직원이 관리자임을 뜻한
-- (empno, mgr 칼럼을 적절히 이용하세요)

KING            5000
  JONES         2975    20
    SCOTT       3000    20
      ADAMS             1100    20
    FORD        3000    20
      SMITH     800     20
  BLAKE         2850    30
    ALLEN       1600    30
    WARD        1250    30
    MARTIN      1250    30
    TURNER      1500    30
    JAMES       950     30
  CLARK         2450    10
    MILLER      1300    10

select lpad(' ',(level-1)*2,' ') || ename, sal, deptno
  from emp
start with ename = 'KING'
connect by prior empno = mgr
```

```
create table Emp2 (
   ename varchar2(20),
   mname varchar2(20),
   job     varchar2(20)
);

insert into Emp2 values ('김길동','가길동','SALESMAN');
insert into Emp2 values ('남길동','하길동','ANALYST');
insert into Emp2 values ('박길동','가길동','ANALYST');
insert into Emp2 values ('가길동','남길동','CLERK');
insert into Emp2 values ('하길동',NULL,'SALESMAN');
commit;

select * from emp2;

ENAME                MNAME                JOB
-------------------- -------------------- --------------------
김길동               가길동               SALESMAN
남길동               하길동               ANALYST
박길동               가길동               ANALYST
가길동               남길동               CLERK
하길동                                    SALESMAN

-- 최고 관리자 하길동부터 TOP DOWN형태로 출력
select LPAD(' ',(level-1)*2,' ')||ename, job
   from emp2
start with ename = '하길동'
connect by prior ename = mname   -- 상위(관리자) ename을 mname으로 가지는레코드

하길동               SALESMAN
   남길동            ANALYST
      가길동         CLERK
         김길동      SALESMAN
         박길동      ANALYST

-- 최하위 박길동부터 최상위 하길동까지... 김길동은 안나온다.
select LPAD(' ',(level-1)*2,' ')||ename, job
   from emp2
```

```
start with ename = '박길동'
connect by prior mname = ename
```

```
박길동          ANALYST
  가길동        CLERK
    남길동      ANALYST
      하길동    SALESMAN
```

-- 계층쿼리에서 가지제거(노드제거)

where절로 제한하면 그 노드만 빠지지만 connect by 절에서 제거하면 그 이하 모든 노드가 제거

*** where절에서 가길동만 제거**

```
select LPAD(' ',(level-1)*2,' ')||ename, job
  from emp2
where ename != '가길동'
start with ename = '하길동'
connect by prior ename = mname
```

```
하길동          SALESMAN
  남길동        ANALYST
      김길동 SALESMAN
      박길동 ANALYST
```

*** connect by 절에서 제거, 가길동 이하 모든 노드가 제거된다.**

```
select LPAD(' ',(level-1)*2,' ')||ename, job
  from emp2
start with ename = '하길동'
connect by prior ename = mname
and ename != '가길동'
```

```
하길동          SALESMAN
  남길동        ANALYST
```

-- EMP 테이블에서 'SMITH' 사원에 대해 입사이후 현재까지의working day수(토, 일제외)를 출력하세요.

-- 토/일 이외의 휴일은 4일로 offday라는 테이블에 다음과 같이 존재한다고 하자.

```
create table offday (
      ilja    varchar2(8),
      cmt varchar2(50)
)

insert into offday values ('20100301','삼일절');
insert into offday values ('20120301','삼일절');
insert into offday values ('20121225','성탄절');
insert into offday values ('20130505','어린이날');
commit;

SELECT COUNT(*) AS WORKINGDAY
 FROM(
      SELECT LEVEL,
                TO_CHAR((HIREDATE + LEVEL), 'YYYYMMDD') PLUSDATE,
                TO_CHAR((HIREDATE + LEVEL), 'DAY') PLUSDAY
      FROM (
                SELECT HIREDATE, SYSDATE AS SYSD
                FROM    emp
                WHERE    ename = 'SMITH'
                AND HIREDATE IS NOT NULL
                )
      CONNECT BY LEVEL < SYSD – HIREDATE + 1
      ORDER BY LEVEL DESC
      )
WHERE PLUSDATE NOT IN(SELECT ILJA FROM OFFDAY)
AND PLUSDAY NOT IN ('토요일', '일요일')
```

7. 오라클 분석함수(Analytic Function) 및 SQL 활용

7.1 분석함수란?

테이블의 로우(행, 레코드)를 그룹핑 하여 집계를 하는 기능을 하는 함수를 이야기 하는데 일반 집계함수(MAX, MIN, COUNT등)와 다른 점은 Multiple Rows를 리턴 한다.(일반 집계함수가 그룹당 하나의 결과를 리턴하지만 분석함수는 행당 하나의 결과를 리턴)

분석을 위한 행(로우)들의 그룹을 윈도우(window)라 부르며 analytic_clause에서 정의한다. window는 현재 함수가 적용되는 행에 대해 분석을 위해 집계를 할 행들의 범위를 결정하는 역할을 하며 다양한 분석함수의 제공으로 서브쿼리등 불필요한 쿼리 사용을 줄일 수 있으며 성능에도 이점이 있다.

집계함수(Aggregate Function)뒤에 Analytic구를 두어 행그룹의 정의를 범위(Window)를 지정하고 각 그룹당 결과값을 반복하여 출력하는 형태로 구성되며, 윈도우를 통해 집계를 하기위한 타겟 레코드의 범위를 결정하게 된다. 분석함수는 조인, WHERE, GROUP BY, HAVING등과 함께 쓰이는 경우 가장 마지막에 집계연산을 수행하며 SELECT절과 ORDER BY절에서 사용 가능하다.

```
-- 분석함수를 사용할 때와 안할 때 비교

SQL> set pagesize 30
SQL> select empno, ename, sal, deptno,
  2            max(sal) over(partition by deptno) max_sal,
  3            min(sal) over(partition by deptno) min_sal
  4   from emp;

     EMPNO ENAME                SAL       DEPTNO      MAX_SAL      MIN_SAL
---------- ---------- ---------- ---------- ---------- ----------
       7782 CLARK              2450          10         5000         1300
       7839 KING               5000          10         5000         1300
       ......
       7900 JAMES               950          30         2850          950
       7698 BLAKE              2850          30         2850          950
       7654 MARTIN             1250          30         2850          950

14 개의 행이 선택되었습니다.

SQL> select e1.empno, e1.ename, e1.sal, e1.deptno, e2.max_sal, e2.min_sal
  2   from emp e1, ( select deptno, max(sal) max_sal, min(sal) min_sal
  3                  from   emp
  4                  group by deptno) e2
  5   where e1.deptno = e2.deptno;

     EMPNO ENAME                SAL       DEPTNO      MAX_SAL      MIN_SAL
---------- ---------- ---------- ---------- ---------- ----------
       7900 JAMES               950          30         2850          950
       7844 TURNER             1500          30         2850          950
       ......
```

```
      7934 MILLER        1300        10        5000        1300
      7839 KING          5000        10        5000        1300
      7782 CLARK         2450        10        5000        1300

14 개의 행이 선택되었습니다.
```

7.2 분석함수 기본형식(PARTITION BY, ORDER BY, WINDOW구, MAX, MIN, SUM, AVG, DENSE RANK FIRST/LAST, KEEP, OVER)

[분석함수 기본형식]

```
analytic_function([ arguments ])  OVER (analytic_clause)
```

Analytic clause

```
[ query_partition_clause ]
[ order_by_clause [ windowing_clause ] ]
```

Query partiotion clause

PARTITION BY
{ value_expr[, value_expr]...
| (value_expr[, value_expr]...)
}

- PARTITON BY구는 하나 또는 여러행을 그룹핑 하기 위하여 사용한다. GROUP BY절과 동일한 기능을 수행한다. 결국 Group By절을 사용하지 않고 행들을 그룹핑 하는 것이다.

Order by clause

ORDER [**SIBLINGS**] **BY**
{ expr | position | c_alias }
[**ASC** | **DESC**]
[**NULLS FIRST** | **NULLS LAST**]
[, { expr | position | c_alias }
[**ASC** | **DESC**]
[**NULLS FIRST** | **NULLS LAST**]
]...

- Partition By로 정의된 윈도우내에서 행들의 정렬 순서를 지정한다.
- ORDER BY절의 ASC|DESC는 결과에 대해 오름차순 또내림차순으로 정렬할 때 사용한다.
- ORDER BY절의 NULLS FIRST|NULL LAST구는 NULL값을 먼저 보일것인지 나중에 보일 것인지를 결정한다._

Windowing_clause

```
{ ROWS | RANGE }
{ BETWEEN
  { UNBOUNDED PRECEDING
  | CURRENT ROW
  | value_expr { PRECEDING | FOLLOWING }
  }
  AND
  { UNBOUNDED FOLLOWING
  | CURRENT ROW
  | value_expr { PRECEDING | FOLLOWING }
  }
| { UNBOUNDED PRECEDING
  | CURRENT ROW
  | value_expr PRECEDING
  }
}
```

- ROWS 와 RANGE : ROWS는 현재 행을 기준으로 몇개의 행을 포함하는지, RANGE는 현재 행을 기준으로 어떤 범위를 포함하는지를 명시한다.
- BETWEEN ... AND절은 분석을 위한 행(로우)들의 그룹의 시작점과 끝점을 이야기 한다.
- UNBOUNDED PRECEDING 구문은 window가 해당 파티션의 처음부터 시작된다는 것을 의미한다.
- UNBOUNDED FOLLOWING 구문은 window의 끝이 해당 파티션의 마지막 행임을 의미한다.
- UNBOUNDED는 한계를 두지 않고 해당 파티션의 끝까지를 의미한다.
- CURRENT ROW 구문은 해당 window의 시작이 현재행임을 의미 한다.
- PRECEDING과 FOLLOWING은 현재 ROW에서 앞쪽인지 뒤쪽인지 방향성을 나타낸다. 만약 5 PRECEDING이라고 기술할 경우 현재행부터 앞5행을 범위로 해서 함수가 적용됨을 의미한다.

FIRST : 주어진 ORDER BY절에 의한 값중 첫번째를 가리킨다.
LAST : 주어진 ORDER BY절에 의한 값중 마지막을 가리킨다.

aggregate_function

KEEP

```
(DENSE_RANK FIRST ORDER BY
 expr [ DESC | ASC ]
      [ NULLS { FIRST | LAST } ]
 [, expr [ DESC | ASC ]
         [ NULLS { FIRST | LAST } ]
 ]...
)
[ OVER query_partition_clause ]
```

- KEEP구는 FIRST, LAST분석함수가 오직 집계함수의 첫번째값 또는 마지막 값만을 리턴한다는 것을 의미한다.
- DENSE_RANK FIRST or DENSE_RANK LAST : 집계함수가 최소값(FIRST), 최대값(LAST)을 반환하는데 같은 값인 경우 동등한 순위를 반환한다. (동일한 값 다음 값은 동일한 값의 순위는 동일한 값을 가진행이 몇 개이든 관계없이 다음 순위가 된다. 만약 RANK라면 다음값의 순위는 동일한 값을 가진 값의 수에 따라 달라진다.)

```
-- EMP 테이블에서 수당(COMM) 내림차순으로 순위를 매기는데 1등이 몇 명이라도 다음 순위는 2등이 되도록 하라. (NULL값은 정렬시 가장 큰 값이 된다)
SQL> SELECT EMPNO, ENAME, SAL, COMM,
            DENSE_RANK() OVER (ORDER BY COMM DESC) D_RANK
     FROM EMP;

EMPNO ENAME   SAL   COMM   D_RANK
-------------------------------------------
7369     SMITH   800              1
7782     CLARK   2450             1
.....
7839     KING    5000             1
7654     MARTIN 1250    1400     2
7521     WARD   1250    500      3
7499     ALLEN   1600    300      4
7844     TURNER 1500    0        5

-- 위예문에서 ORDER BY 기본 설정에 따라 NULL값이 맨 앞으로 왔는데 COMM 내림차순으로 하고 맨 뒤로 보낼려면 NULLS LAST 구문을 사용하면 된다.

SQL> SELECT EMPNO, ENAME, SAL, COMM,
            DENSE_RANK() OVER (ORDER BY COMM DESC NULLS LAST) D_RANK
      FROM EMP;
```

```
EMPNO ENAME    SAL    COMM    D_RANK
-------------------------------------------
7654      MARTIN 1250      1400      1
7521      WARD    1250      500       2
7499      ALLEN    1600      300       3
7844      TURNER 1500      0         4
7788      SCOTT    3000               5
……
7369      SMITH    800                5
7782      CLARK    2450               5
```

-- EMP 테이블에서 부서별로 급여의 최대값, 최대급여사원번호, 최소값, 최소급여사원번호를 출력

```
SELECT deptno,
        MAX(SAL) max_sal,
        MAX(empno) KEEP(DENSE_RANK FIRST ORDER BY sal DESC) max_empno,
        MIN(sal) min_sal,
                MIN(empno) KEEP(DENSE_RANK FIRST   ORDER BY sal Asc) as min_empno
FROM emp
GROUP BY DEPTNO
ORDER BY DEPTNO
```

```
    DEPTNO      MAX_SAL   MAX_EMPNO      MIN_SAL   MIN_EMPNO
---------- ---------- ---------- ---------- ----------
         10         5000         7839         1300         7934
         20         3000         7902          800         7369
         30         2850         7698          950         7900
```

-- 10번 부서의 사원들을 급여일자 순으로 오름차순 정렬하여 출력

```
select empno,ename, sal, deptno, hiredate
from emp
where deptno = 10
order by hiredate
```

```
      EMPNO ENAME                SAL       DEPTNO HIREDATE
---------- ---------- ---------- ---------- --------
       7782 CLARK               2450           10 81/06/09
       7839 KING                5000           10 81/11/17
```

```
      7934 MILLER             1300          10 82/01/23
```

-- 입사일자 순으로 오름차순 정렬했을 때의 급여의 맨처음값, 맨마지막값 출력

```
SELECT deptno,
MIN(sal) KEEP (DENSE_RANK FIRST ORDER BY hiredate) "Worst",
MAX(sal) KEEP (DENSE_RANK LAST ORDER BY hiredate) "Best"
FROM emp
GROUP BY deptno
ORDER BY deptno, "Worst", "Best";
```

```
    DEPTNO      Worst       Best
---------- ---------- ----------
        10       2450       1300
        20        800       3000
        30       1600        950
```

-- 모든 사원들을 대상으로 그 사원이 속한 부서의 급여 최소, 최대, 급여합, 급여순위를 출력

```
SELECT ename, deptno, sal,
       MIN(sal) OVER (PARTITION BY deptno) "Worst",
       MAX(sal) OVER (PARTITION BY deptno) "Best",
       SUM(sal) OVER (PARTITION BY deptno) "SUM(SAL)",
       AVG(sal) OVER (PARTITION BY deptno) "AVG(SAL)",
       RANK()   OVER (PARTITION BY deptno ORDER BY SAL DESC) "RANK"
FROM emp
```

```
ENAME      DEPTNO      SAL       Worst       Best    SUM(SAL)    AVG(SAL)       RANK
---------- ---------- ---------- ---------- ---------- ---------- ---------- ----------
KING               10       5000       1300       5000       8750 2916.66667          1
CLARK              10       2450       1300       5000       8750 2916.66667          2
……
MARTIN             30       1250        950       2850       9400 1566.66667          4
WARD               30       1250        950       2850       9400 1566.66667          4
JAMES              30        950        950       2850       9400 1566.66667
```

-- MYEMP1에서 직무(JOB)가 'CLERK' 인 사원을 출력하면서 사번, 이름, 급여, 그 사원이 속한 부서의 급여 평균을 같이 출력하라.(급여 평균은 소수이하 첫째자리에서 절삭하세오)

```
SQL> SELECT EMPNO, ENAME, SAL,
            TRUNC(AVG(SAL)) OVER (PARTITION BY DEPTNO) AVG_SAL
     FROM MYEMP1
     WHERE JOB = 'CLERK';
```

```
Execution Plan
----------------------------------------------------------------------------------------
| Id  | Operation           | Name    | Rows  | Bytes |TempSpc| Cost (%CPU)| Time     |
----------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT    |         |   100K|  3906K|       | 26294   (2)| 00:05:16 |
|   1 |  WINDOW SORT        |         |   100K|  3906K|  5528K| 26294   (2)| 00:05:16 |
|*  2 |   TABLE ACCESS FULL| MYEMP1  |   100K|  3906K|       | 25264   (2)| 00:05:04 |
----------------------------------------------------------------------------------------
```

-- MYEMP1, MYDEPT1에서 '1'번 부서의 사원을 출력하면서 사번, 이름, 급여, 부서명, 사원이 속한 부서의 급여 평균을 같이 출력하라.(급여 평균은 소수이하 첫째자리에서 절삭하세오)

```
SQL> SELECT E.EMPNO, E.ENAME, E.SAL, D.DNAME,
            TRUNC(AVG(E.SAL) OVER (PARTITION BY E.DEPTNO)) AVG_SAL
       FROM MYEMP1 E, MYDEPT1 D
       WHERE E.DEPTNO = D.DEPTNO
      AND D.DEPTNO = '1';

EMPNO   ENAME              SAL   DNAME              AVG_SAL
-------------------------------------------------------------------------------------
  541   홍길동541           541 개발2팀                 547
  545   가길동545           545 개발2팀                 547
  549   나길동549           549 개발2팀                 547
  553   홍길동553           553 개발2팀                 547
  ......
```

```
Execution Plan
----------------------------------------------------------------------------------------
| Id  | Operation           | Name     | Rows  | Bytes | Cost (%CPU)| Time     |
----------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT    |          |  2510K|   100M| 52440   (1)| 00:10:30 |
|   1 |  WINDOW BUFFER      |          |  2510K|   100M| 52440   (1)| 00:10:30 |
|   2 |   NESTED LOOPS      |          |  2510K|   100M| 25232   (1)| 00:05:03 |
|*  3 |    TABLE ACCESS FULL| MYDEPT1  |     1 |    14 |     3   (0)| 00:00:01 |
|*  4 |    TABLE ACCESS FULL| MYEMP1   |  2510K|    67M| 25229   (1)| 00:05:03 |
----------------------------------------------------------------------------------------
```

-- MYEMP1 테이블에서 입사일자(내림차순), 급여(내림차순) 기준으로 가장 최근 입사일자 이면서 급여가장 많이 받는 사원을 출력한다고 했을 때 ROW_NUMBER() 함수를 이용하여 순위(행번호)를 부여하고 이를 이용하여 가장 최근 입사자(입사일이 같다면 급여 가장많이 받는 사원)의 이름, 급여, 입사일을 출력하시오.

```
SQL> SELECT EMPNO, ENAME, SAL, HIREDATE
       FROM (
              SELECT EMPNO, ENAME, SAL, HIREDATE,
```

```
                        ROW_NUMBER() OVER (ORDER BY HIREDATE DESC, SAL DESC) RNUM
                    FROM MYEMP1
                )
        WHERE RNUM = 1;

경     과: 00:00:15.67

      EMPNO ENAME                          SAL HIREDATE
---------- -------------------- ---------- --------
   9999000 박길동9999000              3999000 15/05/01

Execution Plan
----------------------------------------------------------------------------------------------
| Id  | Operation                  | Name    | Rows  | Bytes |TempSpc| Cost (%CPU)| Time     |
----------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT           |         |    10M|   953M|       |   118K  (1)| 00:23:44 |
|*  1 |  VIEW                      |         |    10M|   953M|       |   118K  (1)| 00:23:44 |
|*  2 |   WINDOW SORT PUSHED RANK  |         |    10M|   333M|   459M|   118K  (1)| 00:23:44 |
|   3 |    TABLE ACCESS FULL       | MYEMP1  |    10M|   333M|       | 25271   (2)| 00:05:04 |
----------------------------------------------------------------------------------------------
```

7.3 분석함수(LISTAGG)

LISTAGG() : String Aggregation 함수로 하나의 칼럼의 값을 그룹핑하고 결합시키는 함수(오라클11g R2 이후 가능). 칼럼의 데이터를 특정칼럼을 기준으로 그룹핑하여 WITHIN GROUP절에서 정의한 순서대로 하나의 로우로 생성한다.

[기본 형식]
LISTAGG([,]) WITHIN GROUP (ORDER BY) [OVER (PARTITION BY)]

여러 행의 값을 그룹핑 하고 결합시키는 것이다. 예를 들면 아래와 같은 데이터를

```
DEPTNO ENAME
--------- ----------
        10 CLARK
        10 KING
        10 MILLER
        20 ADAMS
```

```
        20 FORD
        20 JONES
```

Deptno에 따라 그룹핑 하고 결합시킨다면 아래와 같이 될 것이다.

```
DEPTNO AGGREGATED_ENAMES
--------- ------------------------
        10 CLARK,KING,MILLER
        20 ADAMS,FORD,JONES
```

```
--ENAME을 나열하는데 ENAME 순서대로 DEPTNO별로 그룹핑 하여 출력
SQL>   column employees format a50
SQL> SELECT deptno,
            LISTAGG(ename, ',') WITHIN GROUP (ORDER BY ename) AS employees
       FROM     emp
       GROUP
          BY   deptno;

     DEPTNO EMPLOYEES
---------- --------------------------------------------------
          10 CLARK,KING,MILLER
          20 ADAMS,FORD,JONES,SCOTT,SMITH
          30 ALLEN,BLAKE,JAMES,MARTIN,TURNER,WARD

--EMP 테이블의 데이터를 출력하면서 같은 부서 사원들을 EMPLOYEES 칼럼에 보여주는 예
SQL> set pagesize 50
SQL> column employees format a40
SQL> SELECT deptno,
              ename,
              hiredate,
               LISTAGG(ename, ',')
                    WITHIN GROUP (ORDER BY hiredate)
                    OVER (PARTITION BY deptno) AS employees
       FROM     emp;

     DEPTNO ENAME      HIREDATE EMPLOYEES
---------- ---------- -------- ----------------------------------------
          10 CLARK      81/06/09 CLARK,KING,MILLER
```

```
        10 KING          81/11/17 CLARK,KING,MILLER
        ……
        30 MARTIN        81/09/28 ALLEN,WARD,BLAKE,TURNER,MARTIN,JAMES
        30 JAMES         81/12/03 ALLEN,WARD,BLAKE,TURNER,MARTIN,JAMES

14 개의 행이 선택되었습니다.
```

7.4 분석함수(FIRST_VALUE, LAST_VALUE, RANK, DENSE_RANK, ROW_NUMBER)

FIRST_VALUE : 정렬된 칼럼 값중 처음 값을 리턴한다. NULL이라면 IGNORE NULLS을기술하지 안았다면 NULL을 리턴한다.
LAST_VALUE : 정렬된 칼럼 값중 마지막 값을 리턴한다. NULL이라면 IGNORE NULLS을기술하지 안았다면 NULL을 리턴한다.

[기본 형식]
FIRST_VALUE(*column*) [REPECT|IGNORE NULLS] over(PARTITION BY *column* order by *column* [ASC|DESC]) [ROWS|RANGE UNBOUNDED PRECEDING])

LAST_VALUE(column) [REPECT|IGNORE NULLS] over(PARTITION BY column order by column [ASC|DESC]) [ROWS|RANGE UNBOUNDED PRECEDING])

REPECT|IGNORE NULLS 구문은 NULL값을 연산에 포함할지를 결정하는데 IGNORE인 경우 NULL 값은 제외된다.

```
--EMP 테이블에서 10번 부서 사원중에서 급여가 가장 적은 사원이름을 출력
SQL> select empno, ename, sal,deptno,
            first_value(ename) over(order by sal) as lowest_sal
     from (select * from emp where deptno=10 )
     order by empno;

     EMPNO ENAME                SAL       DEPTNO LOWEST_SAL
---------- ---------- ---------- ---------- ----------
      7782 CLARK              2450         10 MILLER
      7839 KING               5000         10 MILLER
      7934 MILLER             1300         10 MILLER
```

--10번 부서 사원들중 급여가 가장 높은 직원의 입사일을 출력.

```
SQL>select empno, ename, sal,deptno,hiredate,
        last_value(hiredate) over(order by sal ROWS BETWEEN UNBOUNDED PRECEDING AND
UNBOUNDED FOLLOWING) as highsal_hiredate
from (select * from emp where deptno=10 order by hiredate)
order by empno;

EMPNO ENAME                SAL       DEPTNO HIREDATE HIGHSAL_
-------- ---------- ---------- ---------- -------- --------
     7782 CLARK               2450         10 81/06/09 81/11/17
     7839 KING                5000         10 81/11/17 81/11/17
     7934 MILLER              1300         10 82/01/23 81/11/17
```

--전체 사원을 출력하는데 부서별로 가장 오래된 입사일과 최근 입사일을 같이 출력.

```
SQL> select empno, ename, sal,deptno,hiredate,
        first_value(hiredate) over(PARTITION BY deptno  order by hiredate ROWS UNBOUNDED
PRECEDING ) as hiredate1,
        last_value(hiredate) over(PARTITION BY deptno  order by hiredate ROWS BETWEEN
UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING) as hiredate2
from emp
order by deptno, empno;

EMPNO ENAME                SAL       DEPTNO HIREDATE HIREDATE HIREDATE
----- ---------- ---------- ---------- -------- -------- --------
 7782 CLARK               2450         10 81/06/09 81/06/09 82/01/23
 7839 KING                5000         10 81/11/17 81/06/09 82/01/23
 ......
 7698 BLAKE               2850         30 81/05/01 81/02/20 81/12/03
 7844 TURNER              1500         30 81/09/08 81/02/20 81/12/03
 7900 JAMES                950         30 81/12/03 81/02/20 81/12/03
```

-- EMP 테이블의 모든 레코드를 급여 오름차순으로 출력하는데 EMPNO, ENAME, SAL, 이전레코드 SAL값, 다음레코드SAL값을 같이 출력하세요

```
SQL> SELECT empno, ename, deptno, sal,
        FIRST_VALUE (sal) over
          (order by sal asc ROWS BETWEEN 1 PRECEDING AND CURRENT ROW) as "PREVIOUS_SAL",
        LAST_VALUE(sal) over
          (order by sal asc ROWS BETWEEN CURRENT ROW AND 1 FOLLOWING) as "NEXT_SAL"
FROM emp
```

```
EMPNO   ENAME       DEPTNO      SAL      PREVIOUS_SAL   NEXT_SAL
-------------------------------------------------------------------------------
 7369   SMITH          20       800          800          950
 7900   JAMES          30       950          800         1100
 7876   ADAMS          20      1100          950         1250
 7521   WARD           30      1250         1100         1250
 7654   MARTIN         30      1250         1250         1300
 ……
 7839   KING           10      5000         3000         5000
```

RANK() OVER (query_partition_clause ORDER_BY clause) : 전체 행을 대상으로 각각의 행에 대해 순위를 계산하는 역할을 한다.

-- KOR 테이블을 만들고 이름, 성적, 반 3개의 칼럼을 갖도록 구성하자. 아래 CREATE TABLE문의 칼 default 0이라는 의미는 성적(marks)에 값이 들어 오지 않은 경우에는 기본적으로 0이 입력된다는 의미이다.

```
SQL> create table kor (
     name varchar2(10) not null,
     marks number(3,0) default 0,
     ban number(1,0) not null)
     ;

테이블이 생성되었습니다.
SQL> insert into kor values ('가길동', 88, 1);
SQL> insert into kor values ('나길동', 64, 2);
SQL> insert into kor values ('다길동', 78, 1);
SQL> insert into kor values ('라길동', 99, 1);
SQL> insert into kor values ('마길동', 78, 1);
SQL> insert into kor values ('바길동', 89, 2);
SQL> commit;
커밋이 완료되었습니다.
```

-- 반에 관계없이 전체석차를 구하는 것이다. RANK와 DENSE_RANK의 차이에 대해 이해 바란다. 1등이 한명 2등이 2명인 경우 다음순위가 RANK인 경우에는 4등이 되지만 DENSE_RANK인 경우에는 3등이 된다.

```
SQL> select name "이름",
            marks "성적",
```

```
           ban "반",
           rank() over (order by marks desc) "석차1",
           dense_rank() over (order by marks desc) "석차2"
    from kor;

이름                 성적          반       석차1       석차2
---------- ---------- ---------- ---------- ----------
라길동               99          1          1          1
바길동               89          2          2          2
가길동               88          1          3          3
다길동               78          1          4          4
마길동               78          1          4          4
나길동               64          2          6          5

6 개의 행이 선택되었습니다.

-- 반내에서의 석차를 구하는 예문이다. 주요한 키워드는 PARTITION BY이다
SQL> select name "이름",
                marks "성적",
                ban "반",
                rank() over (partition by ban
                             order by marks desc) "반 석차"
    from kor
    /

이름                 성적          반     반 석차
---------- ---------- ---------- ----------
라길동               99          1          1
가길동               88          1          2
다길동               78          1          3
마길동               78          1          3
바길동               89          2          1
나길동               64          2          2

-- EMP 테이블에서 사원의 사번, 이름, 급여, 부서별로 급여 순위를 출력하시오(RANK, DENSE_RANK, ROW_NUMBER의 차이에 대해 이해 하세요)
SQL> SELECT EMPNO, ENAME, SAL, DEPTNO
          RANK() OVER (PARTITION BY DEPTNO ORDER BY SAL DESC) RANK,
          DENSE_RANK() OVER (PARTITION BY DEPTNO ORDER BY SAL DESC) D_RANK,
```

```
           ROW_NUMBER() OVER (PARTITION BY DEPTNO ORDER BY SAL DESC) RNUM
      FROM EMP;

EMPNO          ENAME    SAL    DEPTNO  RANK    D_RANK    RNUM
-------------------------------------------------------------------------
7839           KING     5000   10      1       1         1
7782           CLARK    2450   10      2       2         2
7934           MILLER   1300   10      3       3         3
7788           SCOTT    3000   20      1       1         1
7902           FORD     3000   20      1       1         2
7566           JONES    2975   20      3       2         3
7876           ADAMS    1100   20      4       3         4
7369           SMITH    800    20      5       4         5
7698           BLAKE    2850   30      1       1         1
7499           ALLEN    1600   30      2       2         2
7844           TURNER   1500   30      3       3         3
7654           MARTIN   1250   30      4       4         4
7521           WARD     1250   30      4       4         5
7900           JAMES    950    30      6       5         6
```

7.5 SQL활용

순위 매기기

EMP 테이블에서 급여 상위 1위~5위까지 데이터를 추출해보자. 아래와 같이 여러 방법으로 데이터를 추출할 수 있다. 급여순위로 내림차순 정렬이 필요할 것이고 앞에서 배운 rownum을 적절히 이용하면 된다.

```
SQL> select e.*
      from   (select substr(ename,1, 20) ename,
                    sal,
                    deptno,
                    rank() over (order by sal desc) "순위"
  6              from emp) e
  7   where rownum <= 5 ;

ENAME                          SAL       DEPTNO          순위
```

```
-------------------- ---------- ---------- ----------
KING                           5000         10          1
FORD                           3000         20          2
SCOTT                          3000         20          2
JONES                          2975         20          4
BLAKE                          2850         30          5

SQL> select rownum, e.*
        from (
          select ename, sal from emp a
              where 5 > (select count(*)
                           from emp b
                          where b.sal > a.sal)
           order by sal desc
         ) e     ;

     ROWNUM ENAME                SAL
---------- ---------- ----------
         1 KING                5000
         2 SCOTT               3000
         3 FORD                3000
         4 JONES               2975
         5 BLAKE               2850

SQL> SELECT rownum, ename           ,sal
     FROM (
               SELECT ename   ,sal   ,
                      row_number () over (ORDER BY sal DESC) rn
               FROM emp )
     WHERE rn <= 5;

     ROWNUM ENAME                SAL
---------- ---------- ----------
         1 KING                5000
         2 SCOTT               3000
         3 FORD                3000
         4 JONES               2975
         5 BLAKE               2850
```

```
SQL> SELECT rownum, ename           ,sal
     FROM (
               SELECT ename   ,sal
               FROM emp
               ORDER BY sal DESC) e
     WHERE rownum <= 5   ;

     ROWNUM ENAME                SAL
---------- ---------- ----------
          1 KING                5000
          2 SCOTT               3000
          3 FORD                3000
          4 JONES               2975
          5 BLAKE               2850
```

-- 이번에는 인덱스와 오라클 힌트구문을 이용해 보자. index_desc힌트는 인덱스 영역에서 인덱스 역순 스캔하라는 의미의 힌트다.

```
SQL> create index idx_emp_sal on emp(sal);
SQL> SELECT rownum, ename           ,sal
     FROM (
               SELECT /*+ index_desc(emp idx_emp_sal) */
                         ename   ,sal
               FROM emp
               WHERE sal > 0)
     WHERE rownum <= 5;

     ROWNUM ENAME                SAL
---------- ---------- ----------
          1 KING                5000
          2 FORD                3000
          3 SCOTT               3000
          4 JONES               2975
          5 BLAKE               2850
```

위 방법 중 가장 현명한 방법은 무엇인지 고민해 보고 실습 테이블을 MYEMP1으로 바꾸어서 실습해 보라.

잠시만요

INLINE VIEW : VIEW중에서 SQL명령어 라인 안에서 기술되는 VIEW로써 WHERE절의 서브쿼리 또는

FROM절 다음에 테이블처럼 사용하기 위해 사용된다. 간단한 예로 SELECT절의 FROM다음에 테이블이 오는데, 이곳에 또 다른 SELECT문을 기술하여 이를 인라인뷰(IN_LINE VIEW)하고 한다. 주로 조인 연산을 줄이기 위해 또는 분리된 쿼리를 하나의 쿼리에 모아서 사용하기 위해 사용한다.

COUNT, SUM, DECODE, GROUP BY 활용

이번에는 칼럼의 값이 유일한지 확인하는 쿼리문을 생각해 보자.

```
-- deptno별로 값의 수를 count 후, 최대인것을 골라 0이면 Unique, 아니면 Non Unique를 출력한다.
SQL> select decode(max(count(deptno)),1,'Unique','Non Unique') "Unique ?"
       from emp
     group by deptno ;

Unique ?
----------
Non Unique
```

EMP테이블에서 JOB별, DEPTNO별로 급여의 급여를 합계를 아래처럼 구하는 Matrix Report를 생각해 보자.

```
JOB           10번부서    20번부서    30번부서    40번부서
---------     ---------- ---------- ---------- ----------
ANALYST                      6000
CLERK             1300       1900        950
MANAGER           2450       2975       2850
PRESIDENT         5000
SALESMAN                                 5600
```

```
-- 첫번째 칼럼이 job이 므로 먼저 job별로 group by를 하고, 나머지열은 하나씩 sum, decode를 이용해서 만들어 나가면 된다.
SQL> select *
       from (
              select job,
                     sum(decode(deptno, 10, sal)) "10번부서",
                     sum(decode(deptno, 20, sal)) "20번부서",
                     sum(decode(deptno, 30, sal)) "30번부서",
                     sum(decode(deptno, 40, sal)) "40번부서"
```

```
                from   emp
              group by job
            )
   order by job      ;

JOB            10번부서     20번부서     30번부서     40번부서
--------- ---------- ---------- ---------- ----------
ANALYST                       6000
CLERK             1300         1900          950
MANAGER           2450         2975         2850
PRESIDENT         5000
SALESMAN                                    5600
```

이번에는 EMP 테이블에서 부서별로 JOB의 개수를 세는 쿼리를 생각해 보자. 이 역시 부서별이니 deptno로 group by를 해야 하며 sum, decode를 이용하여 나머지 job들의 칼럼을 만들어 내야 한다.

```
SQL> select
                  deptno "부서코드"
                , sum(decode(job, 'CLERK',      1, 0)) Clerk
                , sum(decode(job, 'SALESMAN',   1, 0)) Salesman
                , sum(decode(job, 'MANAGER',    1, 0)) Manager
                , sum(decode(job, 'ANALYST',    1, 0)) Analyst
                , sum(decode(job, 'PRESIDENT', 1, 0)) President
             FROM      emp e
            GROUP BY deptno;

  부서코드        CLERK     SALESMAN      MANAGER      ANALYST    PRESIDENT
---------- ---------- ---------- ---------- ---------- ----------
        30          1          4          1          0          0
        20          2          0          1          2          0
        10          1          0          1          0          1
```

이번에는 EMP 테이블에서 부서별로 해당 급여 대 인원수를 추출하는 SQL문을 생각해 보자.

부서 $4001-$9999 $3001-$4000 $2001-$3000 $1001-$2000 < $1000

```
---------- ---------- ---------- ---------- ---------- ----------
        30          0          0          1          4          1
        20          0          0          3          1          1
        10          1          0          1          1          0
```

```
SQL> select
            deptno "부서",
            sum(decode(greatest(SAL,4001), least(SAL,9999), 1, 0))   "$4001-$9999",
            sum(decode(greatest(SAL,3001), least(SAL,4000), 1, 0))   "$3001-$4000",
            sum(decode(greatest(SAL,2001), least(SAL,3000), 1, 0))   "$2001-$3000",
            sum(decode(greatest(SAL,1001), least(SAL, 2000), 1, 0)) "$1001-$2000",
            sum(decode(greatest(SAL, 0), least(SAL, 1000), 1, 0))   "<   $1000"
          from   emp
          group by   deptno;

      부서 $4001-$9999 $3001-$4000 $2001-$3000 $1001-$2000    <   $1000
---------- ---------- ---------- ---------- ---------- ----------
        30          0          0          1          4          1
        20          0          0          3          1          1
        10          1          0          1          1          0
```

CASE문을 이용하면 다음과 같다.

```
SELECT
       deptno "부서",
       NVL(SUM((case    when sal between 4001 and 9999 then 1    end)),0) "$4001-$9999" ,
       NVL(SUM((case    when sal between 3001 and 4000 then 1    end)),0) "$3001-4000" ,
       NVL(SUM((case    when sal between 2001 and 3000 then 1    end)),0) "$2001-$3000" ,
       NVL(SUM((case    when sal between 1001 and 2000 then 1   end)),0) "$1001-$2000" ,
       NVL( SUM((case    when sal between 1 and 1000 then 1   end)),0) "$1-$1000"
FROM emp
GROUP BY deptno
```