Chapter 6. 그룹 함수

그룹 함수는 단일행 함수와는 달리 복수 개의 행을 입력으로 받아 한건의 결과를 리턴하는 함수이다. 이 장에서는 그룹 함수의 사용방법을 알아보고 GROUP BY 구문을 이용하여 테이블을 여러개로 그룹핑하는 방법, HAVING 구문을 이용하여 각 그룹을 제한하는 방법을 설명한다.

그룹 함수

그룹 함수란 다음 그림과 같이 복수 개의 행을 입력으로 받아 한건의 결과를 리턴하는 함수이다.

EMPNO	ENAME	...	SAL	DEPTNO
7369	SMITH	...	800	20
7499	ALLEN	...	1600	30
7521	WARD	...	1250	30
7566	JONES	...	2975	20
7654	MARTIN	...	1250	30
...	...	...	...	...
7902	FORD	...	3000	20
7934	MILLER	...	1300	10

SAL의 합

SUM(SAL)
29025

그림 6-1. 그룹 함수의 개념

그룹 함수는 다음과 같은 함수들이 있다.

표 6-1. 그룹 함수

함수	설명
AVG([DISTINCT|ALL]*n*)	*n*의 평균값, Null 무시
COUNT({*|[DISTINCT|ALL]*expr*})	행의 수, 여기서 *expr*은 Null이 아님(Null을 포함하여 전체 행의 수를 계산할 때는 *을 사용)
MAX([DISTINCT|ALL]*expr*)	*expr*의 최대값, Null 무시
MIN([DISTINCT|ALL]*expr*)	*expr*의 최소값, Null 무시
STDDEV([DISTINCT|ALL]*n*)	*n*의 표준편자, Null 무시
SUM([DISTINCT|ALL]*n*)	*n*의 합, Null 무시
VARIANCE([DISTINCT|ALL]*n*)	*n*의 분산, Null 무시

그룹 함수를 사용하는 경우 다음과 같은 작성 지침을 따르도록 한다.

- DISTINCT는 중복되지 않는 값들에 대해서 연산을 수행하는 반면 ALL은 모든 값에 대하여 연산을 수행한다. ALL이 디폴트이므로 매번 ALL을 기술할 필요는 없다.
- *expr*은 CHAR, VARCHAR2, NUMBER, DATE 타입이 될 수 있다.
- 모든 그룹 함수는 Null 값을 연산에 포함시키지 않으므로, NULL을 연산에 포함시키기 위해서는 NVL, NVL2, COALESCE와 같은 NULL 관련 단일행 함수를 사용해야 한다.

그룹 함수를 사용하는 방법은 다음과 같다.

```
SELECT [column,] group_function(column), ...
FROM table;
```

■ AVG, SUM

AVG 함수는 평균값을, SUM 함수는 합계를 리턴하는 함수이다.

```
SQL> SELECT SUM(SAL), AVG(SAL)
  2  FROM EMP;

  SUM(SAL)   AVG(SAL)
---------- ----------
     29025 2073.21429
```

■ MIN, MAX

MIN 함수는 최소값을, MAX 함수는 최대값을 리턴하는 함수이다.

```
SQL> SELECT MIN(SAL), MAX(SAL)
  2  FROM EMP;

  MIN(SAL)   MAX(SAL)
---------- ----------
       800       5000
```

■ COUNT

COUNT(*)는 테이블의 전체 행의 개수를, COUNT(*expr*)은 *expr*이 NULL이 아닌 값들의 개수를 리턴한다.

```
SQL> SELECT COUNT(*), COUNT(COMM)
  2  FROM EMP;

  COUNT(*) COUNT(COMM)
---------- -----------
        14           4
```

COUNT(DISTINCT *expr*)은 *expr*이 NULL이 아닌 값들에 대해서 중복된 값을 제거하고 값들의 개수를 리턴한다.

```
SQL> SELECT COUNT(DISTINCT JOB)
  2  FROM EMP;

COUNT(DISTINCTJOB)
------------------
                 5
```

지금까지 설명한 그룹 함수들은 입력 값으로 NULL이 들어오면 연산에 포함시키지 않는다. 그러면, 테이블내에 NULL이 포함된 컬럼에 대하여 그룹 함수를 사용했을 때 실수하기 쉬운 예를 몇가지 살펴보도록 하자.

EMP 테이블에는 전체 14개의 행이 입력되어 있지만 COMM 컬럼에 값이 입력되어 있는 행은 총 4개 밖에 되지 않는다. 이런 경우, EMP 테이블에서 COMM 컬럼의 평균을 계산해 보자.

```
SQL> SELECT SUM(COMM)/COUNT(*), AVG(COMM)
  2  FROM EMP;

SUM(COMM)/COUNT(*)  AVG(COMM)
------------------ ----------
        157.142857        550
```

SELECT절 뒤의 SUM(COMM)/COUNT(*)은 COMM에서 NULL을 제외한 합산 결과에 전체 행의 개수 즉, 사원 수로 나누어 평균을 계산한 것이며, AVG(COMM)은 COMM에서 NULL을 제외한 합산 결과에 Null이 아닌 값의 개수로 나누어 평균을 계산한 것이다. 즉, 첫 번째 컬럼은 (300+500+1400+0)/14, 두 번째 컬럼은 (300+500+1400+0)/4을 계산한 것이다. 그렇다면 어떤 결과가 과연 옳은 것인가를 생각해보자. 사용자의 관점에 따라 두 경우 모두 맞는다고 볼 수 있다. 그러므로, 사용자가 어떤 결과 값이 필요한지를 신중히 생각해보고 적절히 구현해야 한다. 첫 번째 경우는 NVL 함수를 사용하면 좀 더 간단하게 작성할 수 있다.

```
SQL> SELECT AVG(NVL(COMM, 0)), AVG(COMM)
  2  FROM EMP;

AVG(NVL(COMM,0))  AVG(COMM)
---------------- ----------
      157.142857        550
```

그렇다면 그룹 함수중에서 어떤 함수에 대해 위와 같은 경우를 고려해야 하는지 살펴보자. SUM, MAX, MIN은 해당 컬럼에 NULL 값이 있어도 그 결과에는 영향을 미치지 못함을 유추할 수 있다. 그러나, 위에서 예를 들었던 AVG, STDDEV, VARIANCE와 같이 연산과정에서 값의 개수를 이용하여 나눗셈 연산이 되는 경우는 모두 주의해야 할 필요가 있다.

```
SQL> SELECT STDDEV(NVL(COMM, 0)), STDDEV(COMM)
  2  FROM EMP;

STDDEV(NVL(COMM,0)) STDDEV(COMM)
------------------- ------------
         387.723704   602.771377

SQL> SELECT VARIANCE(NVL(COMM, 0)), VARIANCE(COMM)
  2  FROM EMP;

VARIANCE(NVL(COMM,0)) VARIANCE(COMM)
--------------------- --------------
            150329.67     363333.333
```

GROUP BY

앞 절에서는 주어진 테이블 전체에 대하여 그룹 함수를 이용하는 방법을 설명하였다. 그러

나, 실제 통계작업에 있어서는 전체 집계보다는 부분 집계를 압도적으로 많이 사용하게 된다. 예를 들어, 우리 회사 사원들의 부서별 평균 급여를 계산하거나 직급별 총 급여, 월별 매출액, 분기별 매출액 등등을 계산해야 할 일이 더 많다는 얘기이다. 부분 집계에 대한 개념은 다음 그림과 같다. 아래 그림은 부서별 총급여액을 계산한 것이다.

EMPNO	ENAME	...	SAL	DEPTNO
7782	CLARK	...	2450	10
7839	KING	...	5000	10
7934	MILLER	...	1300	10
7369	SMITH	...	800	20
7876	ADAMS	...	1100	20
7902	FORD	...	3000	20
7788	SCOTT	...	3000	20
7566	JONES	...	2975	20
7499	ALLEN	...	1600	30
7698	BLAKE	...	2850	30
7654	MARTIN	...	1250	30
7900	JAMES	...	950	30
7844	TURNER	...	1500	30
7521	WARD	...	1250	30

DEPTNO별 SAL의 합

DEPTNO	SUM(SAL)
10	8750
20	10875
30	9400

그림 6-2. GROUP BY

■ GROUP BY 작성 지침

이러한 부분집계를 수행하는데 유용한 것이 GROUP BY절이다. 먼저, GROUP BY를 사용하는 방법은 다음과 같다.

```
SELECT [column,] group_function(column), ...
FROM table
[WHERE condition]
[GROUP BY column]
[ORDER BY column];
```

GROUP BY를 사용할 때는 다음의 작성 지침을 따르도록 한다.

· GROUP BY가 사용될 때, SELECT 절 뒤에 사용할 수 있는 컬럼은 GROUP BY 뒤에 기술된 컬럼 또는 그룹 함수가 적용된 컬럼이어야만 한다.
· WHERE 절을 사용하면 테이블에서 주어진 조건을 만족하는 결과 행들에 대해서만 GROUP BY에 적용 된다.
· GROUP BY는 WHERE 조건을 만족하는 결과 행들에 대해서 GROUP BY 뒤에 기술한 컬럼의 값이 같은 것들끼리 그룹으로 묶는다.
· GROUP BY 절 뒤에는 컬럼 별칭을 사용할 수 없다.
· GROUP BY를 사용하면 내부적으로 GROUP BY에 기술된 컬럼으로 오름차순 정렬된다.

■ GROUP BY의 이해

GROUP BY는 Oracle 입문자들이 SQL을 학습하는 과정에서 처음으로 만나게 되는 어

려운 부분 중의 하나이다. 물론, GROUP BY 문법을 완벽히 외어서 사용하는 개발자도 있겠지만 SQL 문법을 단순히 암기해서는 고급 SQL 문장을 구사할 수 없다. 그러므로 사고의 전환을 통해, 하나의 프로그램을 개발한다는 생각으로 SQL 문장을 작성해야 할 것이다.

먼저, GROUP BY의 이해를 돕기 위해 다음과 같은 시나리오를 생각해보자. 모 초등학교에서 학생들의 성적처리를 위해 데이터베이스를 구축하고 다음과 같은 간략한 테이블을 만들어 학생들의 시험 성적 결과를 모두 입력했다고 가정하자. (물론, 성적처리를 위한 실제 데이터베이스는 훨씬 더 복잡하다.) 이 초등학교는 전체 6개 학년, 10개반, 반별 정원은 30명이며 전교생이 월말고사를 치른 후, 그 시험 결과를 모두 저장하였다고 가정한다. 그렇다면 SUNGJEOK 테이블에는 1800개의 행이 입력되어 있을 것이다.

표 6-2. SUNGJEOK 테이블

HAK	BAN	BUN	JUMSOO
1	1	1	92
1	1	2	85
...	...	...	...
3	1	1	75
3	1	2	95
3	1	3	82
...	...	...	...
6	10	29	84
6	10	30	95

위와 같은 상황에서 다음과 같은 문제를 해결해보자.
· 전교생의 평균시험점수는?
· 전교생의 학년별 평균시험점수는? (예, 1학년 88.2점, 6학년 92.5점 등)
· 전교생의 반별 평균시험점수는? (예, 1학년 1반 92.3점, 6학년 10반 89.1점 등)

첫 번째 문제는 단순히 그룹함수만을 사용하면 해결할 수 있으며, 그 보다 나은 SQL문장은 없다.

```
SELECT AVG(JUMSOO) FROM SUNGJUK;
```

두 번째 문제는 학년별 평균시험점수를 계산하는 것이다. 이 경우는 WHERE 절만을 사용하는 방법과 이 장에서 설명하고 있는 GROUP BY를 사용하는 방법의 두가지 방법이 있다. 먼저, WHERE 절만을 사용해서 학년별 평균시험점수를 모두 계산해보자.

```
SELECT AVG(JUMSOO) FROM SUNGJUK WHERE HAK = 1;
SELECT AVG(JUMSOO) FROM SUNGJUK WHERE HAK = 2;
SELECT AVG(JUMSOO) FROM SUNGJUK WHERE HAK = 3;
SELECT AVG(JUMSOO) FROM SUNGJUK WHERE HAK = 4;
SELECT AVG(JUMSOO) FROM SUNGJUK WHERE HAK = 5;
SELECT AVG(JUMSOO) FROM SUNGJUK WHERE HAK = 6;
```

위에서 보는 바와 같이 WHERE 절의 조건만 조금씩 수정하여 총 6번의 SQL 문장을 작

성하면 원하는 결과를 얻을 수 있다. 그렇다면, 세 번째 문제도 WHERE 절만을 사용해서 반별 평균시험점수를 모두 계산할 수 있을 것이다.

```
SELECT AVG(JUMSOO) FROM SUNGJUK WHERE HAK = 1 AND BAN = 1;
SELECT AVG(JUMSOO) FROM SUNGJUK WHERE HAK = 1 AND BAN = 2;
...
SELECT AVG(JUMSOO) FROM SUNGJUK WHERE HAK = 3 AND BAN = 1;
SELECT AVG(JUMSOO) FROM SUNGJUK WHERE HAK = 3 AND BAN = 2;
...
SELECT AVG(JUMSOO) FROM SUNGJUK WHERE HAK = 6 AND BAN = 1;
SELECT AVG(JUMSOO) FROM SUNGJUK WHERE HAK = 6 AND BAN = 2;
```

특히, 위와 같이 WHERE 절만을 사용하는 경우 전체 60번의 SQL 문장을 작성하여야만 원하는 결과를 얻을 수 있게 되므로 현실적으로 사용하기에는 무리가 따른다. 그렇다면, 두 번째 문제와 세 번째 문제를 하나의 SQL 문장으로 해결할 수는 없는지 살펴보자. 다음 SQL 문장을 보면서 GROUP BY를 암기하는 것이 아니라 이해해보도록 하자.

```
SELECT HAK, AVG(JUMSOO) - ③
FROM SUNGJUK - ①
GROUP BY HAK; - ②
```

SQL 문장을 사용하지 않고 선생님 입장에서 초등학생들에게 직접 명령을 내려 학년별 평균점수를 계산한다고 가정해보자.

· 1단계 : 초등학생들을 모두 운동장에 집합시킨다. (①)
· 2단계 : 학년별로 줄을 세운다. 가장 좌측에 1학년부터 맨 우측에는 6학년. (②)
· 3단계 : 학년별 학생 대표에게 각 학년별 평균점수를 계산하도록 하고, 그 결과를 "학년, 평균점수" 형식으로 제출하도록 지시한다. (③)

위와 같은 시나리오와 SQL 문장이 정확히 일치되어 이해가 되었다면 반별 평균점수도 계산해보자.

· 1단계 : 초등학생들을 모두 운동장에 집합시킨다. (①)
· 2단계 : 반별로 줄을 세운다. 가장 좌측에 1학년 1반, 맨 우측에는 6학년 10반. (②)
· 3단계 : 각 반의 반장에게 각 반 평균점수를 계산하도록 하고, 그 결과를 "학년, 반, 평균점수" 형식으로 제출하도록 지시한다. (③)

```
SELECT HAK, BAN, AVG(JUMSOO) - ③
FROM SUNGJUK - ①
GROUP BY HAK, BAN; - ②
```

추가적으로 전교생 중에서 번호가 홀수인 학생들의 반별 평균점수를 계산해보자.

· 1단계 : 초등학생들을 모두 운동장에 집합시킨다. (①)
· 2단계 : 번호가 홀수인 학생들만 운동장에 남고, 나머지 학생은 교실로 들여보낸다.(②)
· 3단계 : 반별로 줄을 세운다. 가장 좌측에 1학년 1반, 우측에는 6학년 10반. (③)
· 4단계 : 각 반의 반장에게 각 반 평균점수를 계산하도록 하고, 그 결과를 "학년, 반, 평균점수" 형식으로 제출하도록 지시한다. (④)

```
SELECT HAK, BAN, AVG(JUMSOO) - ④
FROM SUNGJUK - ①
WHERE MOD(BUN, 2) = 1 - ②
GROUP BY HAK, BAN; - ③
```

마지막으로 위 결과에서 검색 결과를 평균점수가 높은 반부터 출력되도록 하려면 다음과 같이 ORDER BY를 추가해주면 된다.

```
SELECT HAK, BAN, AVG(JUMSOO)
FROM SUNGJUK
WHERE MOD(BUN, 2) = 1
GROUP BY HAK, BAN
ORDER BY AVG(JUMSOO) DESC;
```

■ GROUP BY 예제

실습 테이블을 이용하여 GROUP BY 문장을 연습해보자. 사원 테이블에서 부서별 평균 급여를 계산하면 다음과 같다.

```
SQL> SELECT DEPTNO, AVG(SAL)
  2  FROM EMP
  3  GROUP BY DEPTNO;

    DEPTNO   AVG(SAL)
---------- ----------
        10 2916.66667
        20       2175
        30 1566.66667
```

특히, 주의 할 점은 SELECT 뒤의 컬럼들은 GROUP BY에 기술된 컬럼이어야만 한다. 입문자들이 GROUP BY 사용에 있어서 가장 많은 오류를 경험하는 경우이다.

```
SQL> SELECT ENAME, AVG(SAL)
  2  FROM EMP
  3  GROUP BY DEPTNO;
SELECT ENAME, AVG(SAL)
       *
1행에 오류:
ORA-00979: GROUP BY 의 식이 없습니다
```

그러나, GROUP BY 뒤의 컬럼이 반드시 SELECT 뒤에 나와야 할 필요는 없다.

```
SQL> SELECT AVG(SAL)
  2  FROM EMP
  3  GROUP BY DEPTNO;

  AVG(SAL)
----------
2916.66667
      2175
1566.66667
```

또한, GROUP BY 절이 없이 SELECT 뒤에 일반 컬럼과 그룹함수가 사용된 컬럼이 동

시에 올 수 없다.

```
SQL> SELECT DEPTNO, AVG(SAL)
  2  FROM EMP;
SELECT DEPTNO, AVG(SAL)
       *
1행에 오류:
ORA-00937: 단일 그룹의 그룹 함수가 아닙니다
```

마지막으로 GROUP BY와 WHERE가 같이 쓰이는 SQL 문장에서 WHERE 절에는 그룹 함수가 사용된 컬럼이 올 수 없다. 이러한 경우, 그룹을 제한하기 위해서는 뒤에 설명할 HAVING 절을 사용해야한다.

```
SQL> SELECT DEPTNO, AVG(SAL)
  2  FROM EMP
  3  WHERE AVG(SAL) > 2000
  4  GROUP BY DEPTNO;
WHERE AVG(SAL) > 2000
      *
3행에 오류:
ORA-00934: 그룹 함수는 허가되지 않습니다
```

사원 테이블에서 부서, 업무별 급여 합계를 계산하면 다음과 같다.

```
SQL> SELECT DEPTNO, JOB, SUM(SAL)
  2  FROM EMP
  3  GROUP BY DEPTNO, JOB;

    DEPTNO JOB         SUM(SAL)
---------- --------- ----------
        10 CLERK           1300
        10 MANAGER         2450
        10 PRESIDENT       5000
        20 CLERK           1900
        20 ANALYST         6000
        20 MANAGER         2975
        30 CLERK            950
        30 MANAGER         2850
        30 SALESMAN        5600

9 개의 행이 선택되었습니다.
```

HAVING

HAVING은 GROUP BY에 의해 분류된 그룹들을 제한하기 위한 구문이다. 다음 그림은 부서별 급여의 합계가 9000이 넘는 경우만을 검색한 것이다.

EMPNO	ENAME	...	SAL	DEPTNO
7782	CLARK	...	2450	10
7839	KING	...	5000	10
7934	MILLER	...	1300	10
7369	SMITH	...	800	20
7876	ADAMS	...	1100	20
7902	FORD	...	3000	20
7788	SCOTT	...	3000	20
7566	JONES	...	2975	20
7499	ALLEN	...	1600	30
7698	BLAKE	...	2850	30
7654	MARTIN	...	1250	30
7900	JAMES	...	950	30
7844	TURNER	...	1500	30
7521	WARD	...	1250	30

DEPTNO별 SAL의
합이 9000보다 큰 경우

DEPTNO	SUM(SAL)
20	10875
30	9400

그림 6-3. HAVING

■ HAVING 작성 지침

HAVING을 사용하는 방법은 다음과 같다.

```
SELECT [column,] group_function(column), ...
FROM table
[WHERE condition]
[GROUP BY column]
[HAVING group_condition]
[ORDER BY column];
```

HAVING이 SELECT 문장에 포함되어 있으면 다음과 같은 절차에 의해 결과가 출력된다.

1. GROUP BY에 의해 테이블내의 행들이 주어진 컬럼별로 그룹화된다.
2. HAVING 뒤의 그룹함수가 각 그룹에 적용된다.
3. HAVING 뒤의 조건이 만족하는 그룹들만 출력된다.

■ HAVING 예제

사원 테이블에서 부서별 급여의 합계가 9000보다 큰 경우는 다음과 같다.

```
SQL> SELECT DEPTNO, SUM(SAL)
  2  FROM EMP
  3  GROUP BY DEPTNO
  4  HAVING SUM(SAL) > 9000;

    DEPTNO   SUM(SAL)
---------- ----------
        20      10875
        30       9400
```

사원 테이블에서 급여가 1000보다 큰 사원들에 대하여 부서별 SAL의 합계가 9000보다 큰 경우는 다음과 같다.

```
SQL> SELECT DEPTNO, SUM(SAL)
  2  FROM EMP
  3  WHERE SAL > 1000
  4  GROUP BY DEPTNO
  5  HAVING SUM(SAL) > 9000;

    DEPTNO   SUM(SAL)
---------- ----------
        20      10075
```

그룹함수도 다음과 같이 중첩될 수 있다.

```
SQL> SELECT MAX(SUM(SAL))
  2  FROM EMP
  3  GROUP BY DEPTNO;

MAX(SUM(SAL))
-------------
        10875
```

피벗 테이블

지금까지 GROUP BY를 이용한 부분 집계에 대하여 알아보았다. 예를 들어, 사원 테이블에서 부서, 업무별 급여의 합을 구하기 위해서는 아래와 같은 SQL 문장을 사용했다.

```
SQL> SELECT DEPTNO, JOB, SUM(SAL)
  2  FROM EMP
  3  GROUP BY DEPTNO, JOB;

    DEPTNO JOB         SUM(SAL)
---------- --------- ----------
        10 CLERK           1300
        10 MANAGER         2450
        10 PRESIDENT       5000
        20 CLERK           1900
        20 ANALYST         6000
        20 MANAGER         2975
        30 CLERK            950
        30 MANAGER         2850
        30 SALESMAN        5600

9 개의 행이 선택되었습니다.
```

위 결과를 보면, 원하는 데이터를 얻을 수는 있으나 우리가 실제적으로 사용하는 출력 양식은 위와 같은 형태가 아니라 다음과 같은 피벗 테이블의 형태이다.

표 6-3. 피벗 테이블 형식

	CLERK	MANAGER	PRESIDENT	ANALYST	SALESMAN
10	1300	2450	5000		
20	1900	2975		6000	
30	950	2850			5600

이와 같은 결과를 얻기 위해서는 DECODE 함수와 GROUP BY를 사용해야만 한다. 그렇다면, 다음 절차에 따라 SQL 문장을 적절히 구성하여 위와 같은 피벗 테이블 형태의 결과를 만들어보자.

· 1단계 : 원본 테이블에서 피벗 테이블의 가로줄 컬럼을 DECODE를 이용하여 분류한다.

```
SQL> SELECT DEPTNO,
  2  DECODE(JOB, 'CLERK', SAL) CLERK,
  3  DECODE(JOB, 'MANAGER', SAL) MANAGER,
  4  DECODE(JOB, 'PRESIDENT', SAL) PRESIDENT,
  5  DECODE(JOB, 'ANALYST', SAL) ANALYST,
  6  DECODE(JOB, 'SALESMAN', SAL) SALESMAN
  7  FROM EMP;

    DEPTNO      CLERK    MANAGER  PRESIDENT    ANALYST   SALESMAN
---------- ---------- ---------- ---------- ---------- ----------
        20        800
        30                                              1600
        30                                              1250
        20                  2975
        30                                              1250
        30                  2850
        10                  2450
        20                                        3000
        10                            5000
        30                                              1500
        20       1100
        30        950
        20                                        3000
        10       1300

14 개의 행이 선택되었습니다.
```

· 2단계 : GROUP BY를 이용하여 부서별로 그룹화한다.

```
SQL> SELECT DEPTNO,
  2  SUM(DECODE(JOB, 'CLERK', SAL)) CLERK,
  3  SUM(DECODE(JOB, 'MANAGER', SAL)) MANAGER,
  4  SUM(DECODE(JOB, 'PRESIDENT', SAL)) PRESIDENT,
  5  SUM(DECODE(JOB, 'ANALYST', SAL)) ANALYST,
  6  SUM(DECODE(JOB, 'SALESMAN', SAL)) SALESMAN
  7  FROM EMP
  8  GROUP BY DEPTNO;

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

참고로 위의 결과에서 맨 우측에 세로줄을 추가하여 부서별 합계를 구하면 다음과 같다.

```
SQL> SELECT DEPTNO,
  2  SUM(DECODE(JOB, 'CLERK', SAL)) CLERK,
  3  SUM(DECODE(JOB, 'MANAGER', SAL)) MANAGER,
  4  SUM(DECODE(JOB, 'PRESIDENT', SAL)) PRESIDENT,
  5  SUM(DECODE(JOB, 'ANALYST', SAL)) ANALYST,
  6  SUM(DECODE(JOB, 'SALESMAN', SAL)) SALESMAN,
  7  SUM(SAL) "계"
  8  FROM EMP
  9  GROUP BY DEPTNO;

    DEPTNO      CLERK    MANAGER  PRESIDENT    ANALYST   SALESMAN         계
---------- ---------- ---------- ---------- ---------- ---------- ----------
        10       1300       2450       5000                          8750
        20       1900       2975                  6000                10875
        30        950       2850                             5600       9400
```

마지막으로 맨 마지막줄에 JOB별 합계와 전체 급여 총합을 추가하면 다음과 같다.

```
SQL> SELECT DEPTNO,
  2  SUM(DECODE(JOB, 'CLERK', SAL)) CLERK,
  3  SUM(DECODE(JOB, 'MANAGER', SAL)) MANAGER,
  4  SUM(DECODE(JOB, 'PRESIDENT', SAL)) PRESIDENT,
  5  SUM(DECODE(JOB, 'ANALYST', SAL)) ANALYST,
  6  SUM(DECODE(JOB, 'SALESMAN', SAL)) SALESMAN,
  7  SUM(SAL) "계"
  8  FROM EMP
  9  GROUP BY ROLLUP(DEPTNO);

    DEPTNO      CLERK    MANAGER  PRESIDENT    ANALYST   SALESMAN         계
---------- ---------- ---------- ---------- ---------- ---------- ----------
        10       1300       2450       5000                          8750
        20       1900       2975                  6000                10875
        30        950       2850                             5600       9400
                 4150       8275       5000       6000       5600      29025
```

위 두가지 경우에 대한 분석은 독자들의 몫으로 남겨두도록 하겠다.

복습

1. 사원 테이블에서 최대 급여, 최소 급여, 전체 급여 합, 평균 급여를 검색하시오.

2. 사원 테이블에서 부서별 인원수를 검색하시오.

3. 사원 테이블에서 부서별 인원수가 6명 이상인 부서코드를 검색하시오.

4. 사원 테이블에서 급여가 높은 순서대로 등수를 부여하고자 한다. 다음과 같은 결과를 출력할 수 있는 SQL 문장을 작성하시오.
   (Hint : Self Join, Non-Equi Join, GROUP BY, COUNT 사용)

ENAME	등수
KING	1
FORD	2
SCOTT	2
JONES	4
BLAKE	5
CLARK	6
ALLEN	7
TURNER	8
MILLER	9
MARTIN	10
WARD	10
ADAMS	12
JAMES	13
SMITH	14

5. 사원 테이블로부터 부서명, 업무별 급여 합계를 계산하고자 한다. 다음과 같은 결과를 출력할 수 있는 SQL 문장을 작성하시오.

DNAME	CLERK	MANAGER	PRESIDENT	ANALYST	SALESMAN
ACCOUNTING	1300	2450	5000		
RESEARCH	1900	2975		6000	
SALES	950	2850			5600