

데이터베이스 프로그래밍

GROUP 함수

학습 목표

- GROUP 함수의 특징을 이해한다.
- GROUP 함수를 응용한 쿼리를 작성할 수 있다.
- GROUP BY 절을 사용해 GROUPING 할 수 있다.
- HAVING 절을 이용해 GROUP 함수 조건 처리를 하여 제한을 할 수 있다.
- ROLLUP 연산자를 사용하여 하위 총계 값을 계산할 수 있다.
- CUBE 연산자를 사용하여 모든 열 조합에 대한 그룹 함수 적용을 수행할 수 있다.
- GROUPING 함수를 사용하여 ROLLUP 또는 CUBE를 통해 만들어진 결과를 식별할 수 있다.
- GROUPING SETS 절을 사용하여 원하는 조합을 설정할 수 있다.

GROUP 함수

- = 집계함수(Aggregate Function)
- 집계
 - 하나 이상의 데이터들을 대상으로 일종의 통계 정보(전체 개수, 평균, 최대값, 최소값)
- 단일 행 함수와 달리 전체 집합 또는 그룹으로 분류된 집합에 작용하여 그룹당 하나의 결과 생성
- ROLLUP, CUBE, GROUPING SETS 결과에 대한 정렬이 필요한 경우 ORDER BY 절에 정렬 컬럼 명시

GROUP 함수의 종류

- 집계 함수
- ROLLUP 함수
 - 소그룹 간의 소계 계산
 - GROUP BY의 확장된 형태
 - 사용하기 쉬우며, 병렬로 수행이 가능하기 때문에 매우 효과적
 - 시간 및 지역처럼 계층적 분류를 포함하고 있는 데이터의 집계에 적합
- CUBE 함수
 - GROUP BY 함수들 간 다차원적인 소계를 계산
 - 결합 가능한 모든 값에 대해 다차원적인 집계 생성
 - 장점 : ROLLUP에 비해 다양한 데이터를 얻을 수 있음
 - 단점 : 시스템 부하를 많이 줌
- GROUPING SETS
 - 특정 항목에 대한 소계 계산
 - 장점 : 원하는 부분의 소계만 손쉽게 추출

GROUP 함수

함수	설 명
COUNT(*)	<ul style="list-style-type: none">· NULL 값을 포함한 행의 수 출력· 숫자, 문자 날짜 타입 가능
COUNT(표현식)	<ul style="list-style-type: none">· 표현식의 행의 수 출력(표현식의 결과값이 NULL인 경우 제외)· 숫자, 문자 날짜 타입 가능
SUM([DISTINCT ALL] 표현식)	<ul style="list-style-type: none">· 표현식의 합계를 출력(표현식의 결과값이 NULL인 경우 제외)· 숫자 타입만 가능
AVG([DISTINCT ALL] 표현식)	<ul style="list-style-type: none">· 표현식의 평균을 출력(표현식의 결과값이 NULL인 경우 제외)· 숫자 타입만 가능
MAX([DISTINCT ALL] 표현식)	<ul style="list-style-type: none">· 표현식의 최대값을 출력(표현식의 결과값이 NULL인 경우 제외)· 숫자, 문자 날짜 타입 가능
MIN([DISTINCT ALL] 표현식)	<ul style="list-style-type: none">· 표현식의 최소값을 출력(표현식의 결과값이 NULL인 경우 제외)· 숫자, 문자 날짜 타입 가능
STDDEV([DISTINCT ALL] 표현식)	<ul style="list-style-type: none">· 표현식의 표준편차를 출력(표현식의 결과값이 NULL인 경우 제외)· 숫자 타입만 가능
VARIANCE([DISTINCT ALL] 표현식)	<ul style="list-style-type: none">· 표현식의 분산을 출력(표현식의 결과값이 NULL인 경우 제외)· 숫자 타입만 가능

GROUP 함수의 동작

```
SELECT salary FROM employees;
```

SALARY

2400
3400

24000
17000
17000
9000
6000
4800
4800
4200

SALARY

12008
9000
8200
7700
7800
6900
11000
3100

SALARY

111 rows selected.

SQL> █

```
SELECT AVG(salary) FROM employees;
```

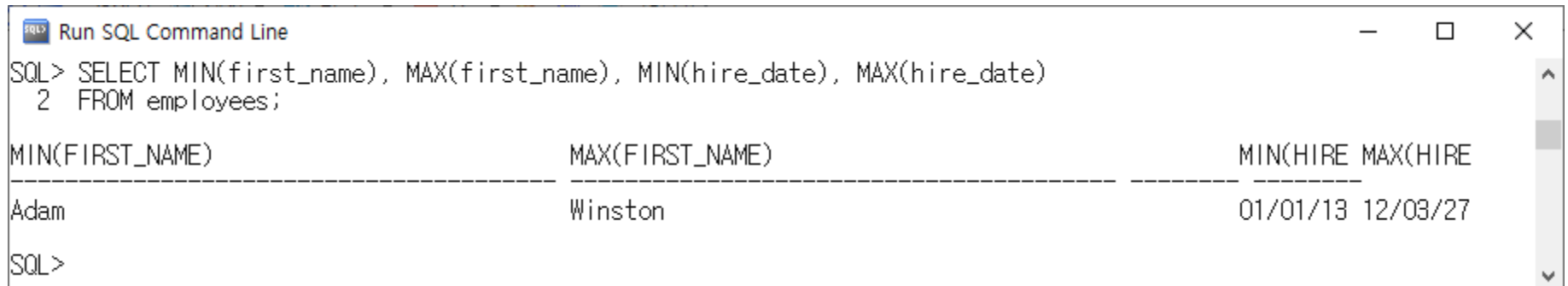
AVG(SALARY)

6396.47706

SQL>

다음 결과 확인

```
SELECT MIN(first_name), MAX(first_name), MIN(hire_date), MAX(hire_date)
FROM employees;
```



The screenshot shows a window titled "Run SQL Command Line" with a standard Windows interface (minimize, maximize, close buttons). The command prompt displays the following SQL query and its results:

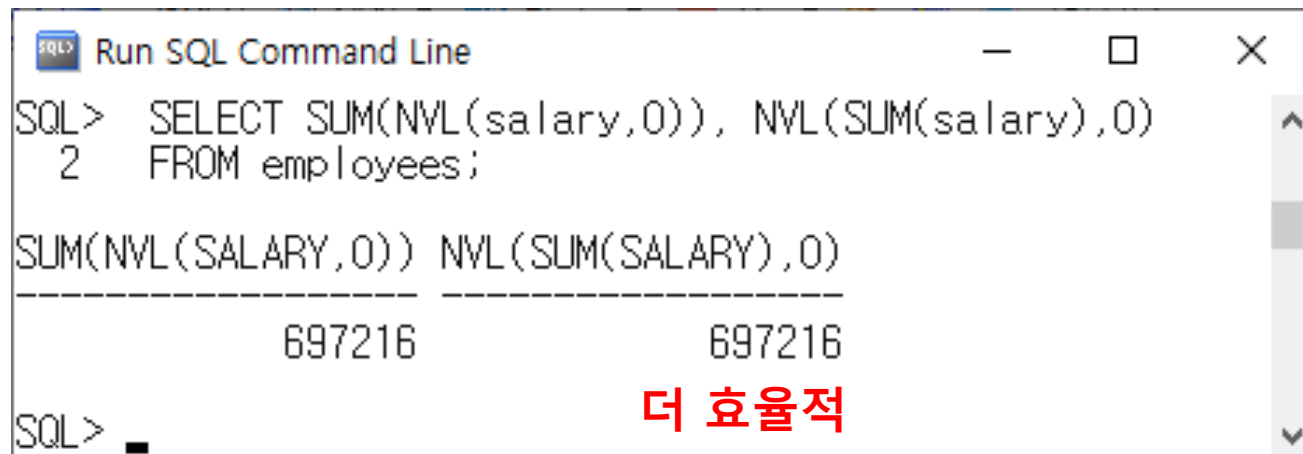
```
SQL> SELECT MIN(first_name), MAX(first_name), MIN(hire_date), MAX(hire_date)
2  FROM employees;
```

MIN(FIRST_NAME)	MAX(FIRST_NAME)	MIN(HIRE DATE)	MAX(HIRE DATE)
Adam	Winston	01/01/13	12/03/27

The prompt "SQL>" is visible at the bottom of the window.

다음 결과 확인

```
SELECT SUM(NVL(salary,0)), NVL(SUM(salary),0)
FROM employees;
```



The screenshot shows a window titled "Run SQL Command Line" with a standard Windows interface (minimize, maximize, close buttons). The command prompt shows the following SQL query being executed:

```
SQL> SELECT SUM(NVL(salary,0)), NVL(SUM(salary),0)
2 FROM employees;
```

The output of the query is displayed in a table format:

SUM(NVL(SALARY,0))	NVL(SUM(SALARY),0)
697216	697216

The prompt "SQL>" is followed by a cursor. The text "더 효율적" (More efficient) is written in red below the output.

다음 결과 확인

```
SELECT AVG(salary), SUM(salary)/COUNT(*)  
FROM employees;
```

```
SQL> SELECT AVG(salary), SUM(salary)/COUNT(*)  
2 FROM employees;
```

AVG(SALARY)	SUM(SALARY)/COUNT(*)
6396.47706	6281.22523

```
SQL>
```

```
SQL> SELECT AVG(NVL(salary, 0)), SUM(salary)/COUNT(*)  
2 FROM employees;
```

AVG(NVL(SALARY,0))	SUM(SALARY)/COUNT(*)
6281.22523	6281.22523

```
SQL> _
```

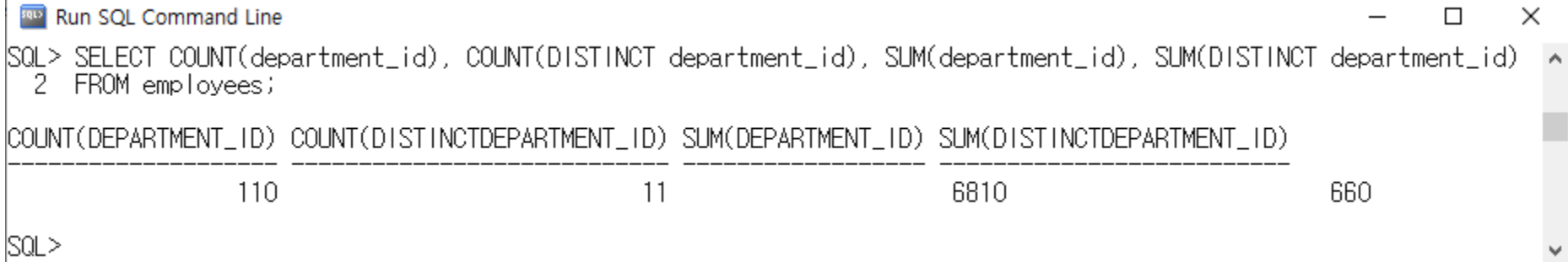
```
SQL> SELECT NVL(AVG(salary),0), SUM(salary)/COUNT(*)  
2 FROM employees;
```

NVL(AVG(SALARY),0)	SUM(SALARY)/COUNT(*)
6396.47706	6281.22523

```
SQL> _
```

다음 결과 확인

```
SELECT COUNT(department_id), COUNT(DISTINCT department_id),  
SUM(department_id), SUM(DISTINCT department_id)  
FROM employees;
```



The screenshot shows a SQL Command Line window titled "Run SQL Command Line". The command entered is: `SQL> SELECT COUNT(department_id), COUNT(DISTINCT department_id), SUM(department_id), SUM(DISTINCT department_id) FROM employees;`. The results are displayed in a table with four columns: `COUNT(DEPARTMENT_ID)`, `COUNT(DISTINCTDEPARTMENT_ID)`, `SUM(DEPARTMENT_ID)`, and `SUM(DISTINCTDEPARTMENT_ID)`. The values are 110, 11, 6810, and 660 respectively.

COUNT(DEPARTMENT_ID)	COUNT(DISTINCTDEPARTMENT_ID)	SUM(DEPARTMENT_ID)	SUM(DISTINCTDEPARTMENT_ID)
110	11	6810	660

SQL>

GROUP BY 절

- SELECT 그룹함수([DISTINCT]/ALL), { * | column명 | 표현식 [별칭] }
FROM 테이블명
[WHERE 검색조건]
[GROUP BY column | 표현식]
[HAVING 검색조건]
[ORDER BY {속성} [ASC | DESC]] ;
- GROUP BY 절을 통해 소그룹별 기준을 정한 후, SELECT 절에 집계 함수를 사용
- 집계 함수의 통계 정보는 NULL을 가진 행을 제외하고 수행
- GROUP BY 절을 통해 그룹을 묶은 후 SELECT 절에는 GROUP BY에서 사용한 <속성>과 집계 함수만 사용 가능
- GROUP BY 절에서는 SELECT 절과는 달리 ALIAS 사용 불가능
- 하나 이상의 GROUP BY 열을 나열하여 그룹에 대한 요약 결과 조회 가능
- 집계 함수는 WHERE 절에는 사용 불가능
 - ∴ 집계 함수를 사용할 수 있는 GROUP BY 절보다 WHERE 절 먼저 수행
 - ∴ HAVING 절에서 조건 처리
- WHERE 절은 전체 데이터를 GROUP으로 나누기 전 행들을 미리 제거

GROUP BY 절의 동작

```
SELECT department_id, salary  
FROM employees  
ORDER BY department_id;
```

20	3400
20	6000
30	2500
30	2600
30	2900
30	3100
30	11000
30	2400
DEPARTMENT_ID	SALARY
30	2800
40	6500
50	8000
50	8200
50	7900
50	6500
DEPARTMENT_ID	SALARY
90	17000
90	24000
90	17000
100	7800
100	7700
100	8200
100	9000
100	6900
100	12008
110	8300
110	12008
DEPARTMENT_ID	SALARY
	7000

111 rows selected.

```
SELECT department_id, AVG(salary)  
FROM employees  
GROUP BY department_id  
ORDER BY department_id;
```

DEPARTMENT_ID	AVG(SALARY)
10	4400
20	7466.66667
30	3900
40	6500
50	3475.55556
60	5760
70	10000
80	8955.88235
90	19333.3333
100	8601.33333
110	10154
DEPARTMENT_ID	AVG(SALARY)
	7000

12 rows selected.

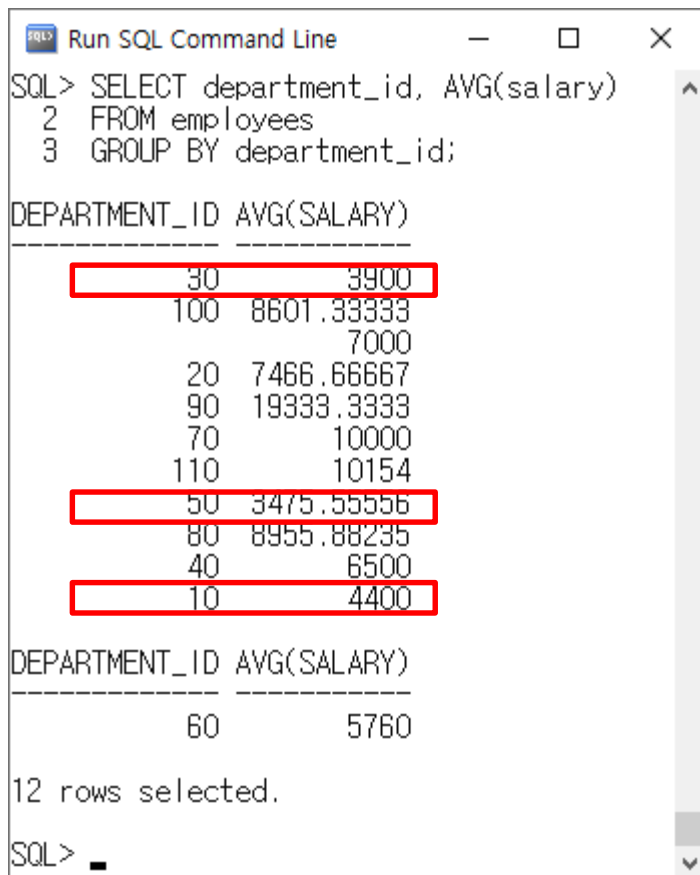
SQL>

HAVING 절

- GROUP BY 절의 기준 항목이나 소그룹의 집계 함수를 이용한 조건 표시 가능
- GROUP BY 절에 의한 소그룹별로 만들어진 집계 데이터 중, HAVING 절에서 제한 조건을 두어 조건을 만족하는 내용만 출력
- HAVING 절을 GROUP BY 앞에 사용해도 되지만 GROUP BY 절을 앞에 사용할 것을 권장
- HAVING 절 사용시 수행 단계
 - 행 분류(그룹 형성)
 - 그룹 함수 그룹에 적용
 - HAVING 절의 조건에 일치하는 그룹 표시

HAVING 절의 동작

```
SELECT department_id, AVG(salary)
FROM employees
GROUP BY department_id;
```



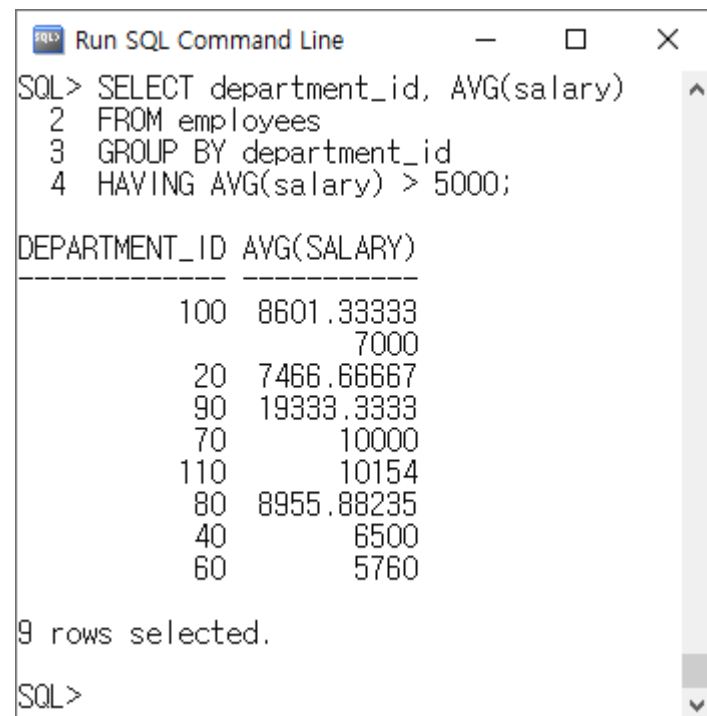
```
SQL> SELECT department_id, AVG(salary)
2 FROM employees
3 GROUP BY department_id;
```

DEPARTMENT_ID	AVG(SALARY)
30	3900
100	8601.33333
	7000
20	7466.66667
90	19333.3333
70	10000
110	10154
50	3475.55556
80	8955.88235
40	6500
10	4400
DEPARTMENT_ID	AVG(SALARY)
60	5760

12 rows selected.

SQL> _

```
SELECT department_id, AVG(salary)
FROM employees
GROUP BY department_id
HAVING AVG(salary) > 5000;
```



```
SQL> SELECT department_id, AVG(salary)
2 FROM employees
3 GROUP BY department_id
4 HAVING AVG(salary) > 5000;
```

DEPARTMENT_ID	AVG(SALARY)
100	8601.33333
	7000
20	7466.66667
90	19333.3333
70	10000
110	10154
80	8955.88235
40	6500
60	5760

9 rows selected.

SQL>

다음 실행 결과 확인

```
SELECT job_id, AVG(salary), SUM(salary)
FROM employees
GROUP BY job_id
HAVING AVG(salary) >= 3000;
```

```
SELECT job_id, AVG(salary), SUM(salary)
FROM employees
HAVING AVG(salary) >= 3000
GROUP BY job_id;
```

동일한 결과이나
HAVING절의 위치는
GROUP BY
다음에 쓸 것을 권장

Run SQL Command Line

```
SQL> SELECT job_id, AVG(salary), SUM(salary)
2 FROM employees
3 GROUP BY job_id
4 HAVING AVG(salary) >= 3000;
```

JOB_ID	AVG(SALARY)	SUM(SALARY)
IT_PROG	5760	28800
AC_MGR	12008	12008
AC_ACCOUNT	8300	8300
ST_MAN	7280	36400
PU_MAN	11000	11000
AD_ASST	4400	4400
AD_VP	17000	34000
SH_CLERK	3215	64300
FI_ACCOUNT	7000	42000
FI_MGR	7704	15408
PR_REP	10000	10000

JOB_ID	AVG(SALARY)	SUM(SALARY)
SA_MAN	12200	61000
MK_MAN	13000	13000
AD_PRES	24000	24000
SA_REP	8350	250500
MK_REP	6000	6000
HR_REP	6500	6500

17 rows selected.

SQL>

Run SQL Command Line

```
SQL> SELECT job_id, AVG(salary), SUM(salary)
2 FROM employees
3 HAVING AVG(salary) >= 3000
4 GROUP BY job_id;
```

JOB_ID	AVG(SALARY)	SUM(SALARY)
IT_PROG	5760	28800
AC_MGR	12008	12008
AC_ACCOUNT	8300	8300
ST_MAN	7280	36400
PU_MAN	11000	11000
AD_ASST	4400	4400
AD_VP	17000	34000
SH_CLERK	3215	64300
FI_ACCOUNT	7000	42000
FI_MGR	7704	15408
PR_REP	10000	10000

JOB_ID	AVG(SALARY)	SUM(SALARY)
SA_MAN	12200	61000
MK_MAN	13000	13000
AD_PRES	24000	24000
SA_REP	8350	250500
MK_REP	6000	6000
HR_REP	6500	6500

17 rows selected.

SQL>

다음 실행 결과 확인

```
SELECT department_id, SUM(salary)
FROM employees
GROUP BY department_id
HAVING SUM(salary) > 10000 AND
```

WHERE 절에 쓰는 게 성능 상 더 좋음

```
department_id IN(20, 30);
```

```
SQL> Run SQL Command Line
SQL> SELECT department_id, SUM(salary)
2  FROM employees
3  GROUP BY department_id
4  HAVING SUM(salary) > 10000 AND department_id IN(20, 30);

DEPARTMENT_ID  SUM(SALARY)
-----
30              27300
20              22400

SQL>
```

```
SELECT department_id, SUM(salary)
FROM employees
WHERE department_id IN(20, 30)
GROUP BY department_id
HAVING SUM(salary) > 10000;
```

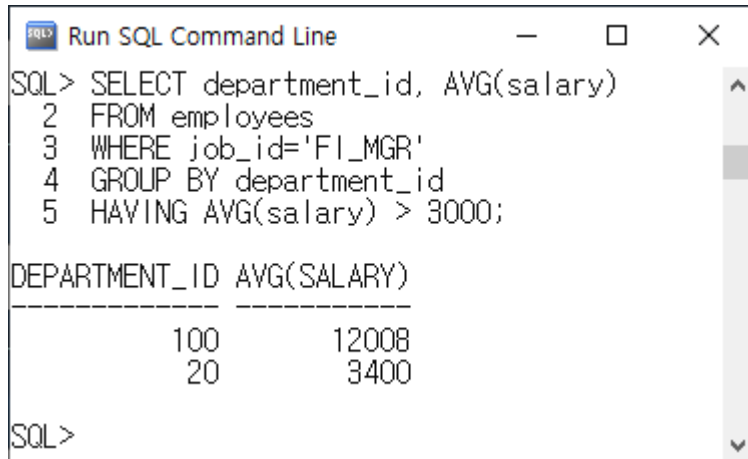
```
SQL> Run SQL Command Line
SQL> SELECT department_id, SUM(salary)
2  FROM employees
3  WHERE department_id IN(20, 30)
4  GROUP BY department_id
5  HAVING SUM(salary) > 10000;

DEPARTMENT_ID  SUM(SALARY)
-----
20              22400
30              27300

SQL>
```


다음 실행 결과 확인

```
SELECT department_id, AVG(salary)
FROM employees
WHERE job_id='FI_MGR'
GROUP BY department_id
HAVING AVG(salary) > 3000;
```

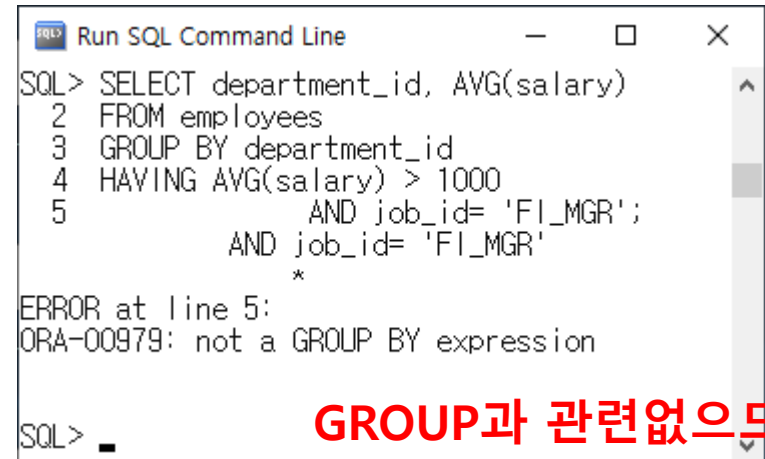


```
SQL> SELECT department_id, AVG(salary)
2 FROM employees
3 WHERE job_id='FI_MGR'
4 GROUP BY department_id
5 HAVING AVG(salary) > 3000;

DEPARTMENT_ID  AVG(SALARY)
-----
100             12008
20              3400

SQL>
```

```
SELECT department_id, AVG(salary)
FROM employees
GROUP BY department_id
HAVING AVG(salary) > 1000
AND job_id= 'FI_MGR';
```



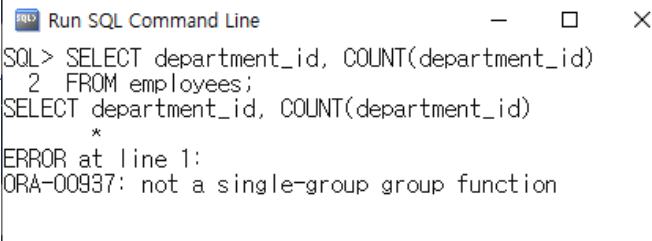
```
SQL> SELECT department_id, AVG(salary)
2 FROM employees
3 GROUP BY department_id
4 HAVING AVG(salary) > 1000
5     AND job_id= 'FI_MGR';
      AND job_id= 'FI_MGR'
      *
ERROR at line 5:
ORA-00979: not a GROUP BY expression

SQL> _
```

**GROUP과 관련없으므로
SYNTAX ERROR**

다음 실행 결과 확인

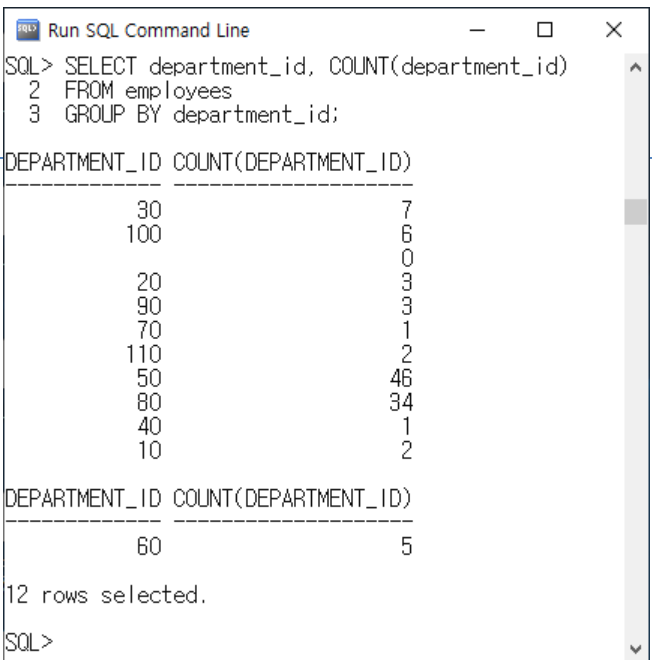
```
SELECT department_id, COUNT(department_id)
FROM employees;
```



Run SQL Command Line

```
SQL> SELECT department_id, COUNT(department_id)
2 FROM employees;
SELECT department_id, COUNT(department_id)
*
ERROR at line 1:
ORA-00937: not a single-group group function
```

```
SELECT department_id, COUNT(department_id)
FROM employees
GROUP BY department_id;
```



Run SQL Command Line

```
SQL> SELECT department_id, COUNT(department_id)
3 FROM employees
3 GROUP BY department_id;
```

DEPARTMENT_ID	COUNT(DEPARTMENT_ID)
30	7
100	6
0	0
20	3
90	3
70	1
110	2
50	46
80	34
40	1
10	2
DEPARTMENT_ID	COUNT(DEPARTMENT_ID)
60	5

12 rows selected.

```
SQL>
```

ORA-00937 : not a single-group group function

```
SELECT first_name, hire_date, MIN(hire_date)
FROM employees
WHERE department_id = 20;
```

```
Run SQL Command Line
SQL> SELECT first_name, hire_date, MIN(hire_date)
  2  FROM employees
  3  WHERE department_id = 20;
SELECT first_name, hire_date, MIN(hire_date)
*
ERROR at line 1:
ORA-00937: not a single-group group function

SQL> _
```

```
SELECT first_name, hire_date, MIN(hire_date)
FROM employees
WHERE department_id=20
GROUP BY first_name, hire_date;
```

```
SQL> SELECT first_name, hire_date, MIN(hire_date)
  2  FROM employees
  3  WHERE department_id=20
  4  GROUP BY first_name, hire_date;
```

FIRST_NAME	HIRE_DAT	MIN(HIRE
Michael	04/02/17	04/02/17
Pat	05/08/17	05/08/17
M%ary	11/11/14	11/11/14

SQL>

```
SELECT MIN(hire_date) FROM employees
WHERE department_id=20;
```

```
SQL> SELECT MIN(hire_date) FROM employees
  2  WHERE department_id=20;

MIN(HIRE
-----
04/02/17

SQL>
```

```
SELECT MIN(hire_date) FROM employees;
```

```
SQL> SELECT MIN(hire_date) FROM employees;

MIN(HIRE
-----
01/01/13

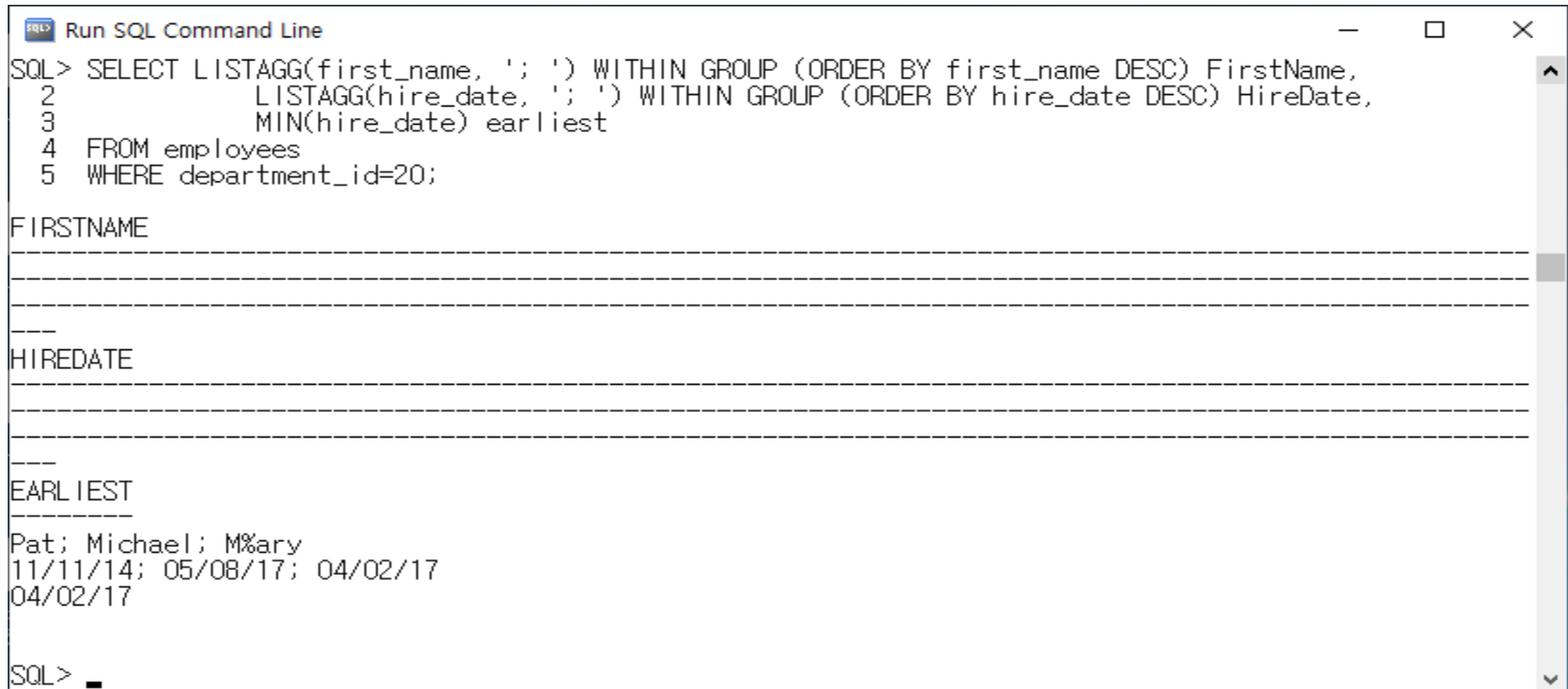
SQL>
```

LISTAGG 함수

- ORDER BY 절에 지정된 각 그룹 내의 데이터를 열의 값으로 병합
- 형식
LISTAGG(컬럼명, 구분자) WITHIN GROUP(ORDER BY 가로로 나열하고 싶은 규칙)
- 기능
 - 단일 세트 집계 함수 : 모든 행에 대해 작업하고 단일 출력 행 반환
 - 그룹 집합 집계 함수 : GROUP BY 절에 정의된 각 그룹에 대해 작업하고 결과 행 반환
 - 분석함수 : query_partition_clause를 기반으로 쿼리 결과 세트를 그룹으로 분할

다음 실행 결과 확인 (단일 세트 집계 함수)

```
SELECT LISTAGG(first_name, ';' ) WITHIN GROUP (ORDER BY first_name DESC) FirstName,  
       LISTAGG(hire_date, ';' ) WITHIN GROUP (ORDER BY hire_date DESC) HireDate,  
       MIN(hire_date) earliest  
FROM employees  
WHERE department_id=20;
```



The screenshot shows a SQL Command Line window titled "Run SQL Command Line". The query entered is:

```
SQL> SELECT LISTAGG(first_name, ';' ) WITHIN GROUP (ORDER BY first_name DESC) FirstName,  
2      LISTAGG(hire_date, ';' ) WITHIN GROUP (ORDER BY hire_date DESC) HireDate,  
3      MIN(hire_date) earliest  
4 FROM employees  
5 WHERE department_id=20;
```

The results are displayed in three sections, each with a header and a separator line:

FIRSTNAME
Pat; Michael; Mary

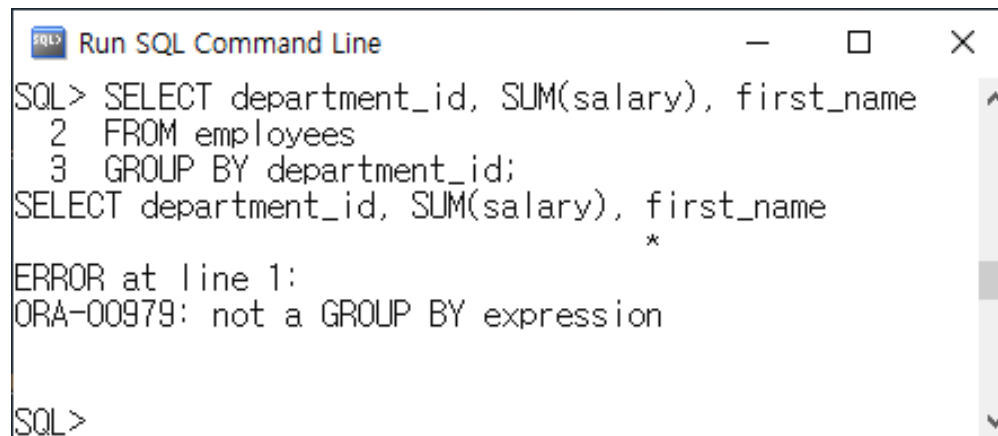
HIREDATE
11/11/14; 05/08/17; 04/02/17

EARLIEST
04/02/17

The SQL prompt "SQL>" is visible at the bottom left of the window.

다음 실행 결과 확인

```
SELECT department_id, SUM(salary), first_name  
FROM employees  
GROUP BY department_id;
```

A screenshot of a 'Run SQL Command Line' window. The window title bar includes a small SQL icon, the text 'Run SQL Command Line', and standard window controls (minimize, maximize, close). The command prompt shows the following text:
SQL> SELECT department_id, SUM(salary), first_name
2 FROM employees
3 GROUP BY department_id;
SELECT department_id, SUM(salary), first_name
*
ERROR at line 1:
ORA-00979: not a GROUP BY expression
SQL>
A vertical scrollbar is visible on the right side of the text area.

```
SQL> SELECT department_id, SUM(salary), first_name  
2 FROM employees  
3 GROUP BY department_id;  
SELECT department_id, SUM(salary), first_name  
*  
ERROR at line 1:  
ORA-00979: not a GROUP BY expression  
SQL>
```

다음 실행 결과 확인

```
SELECT department_id, SUM(salary),  
       LISTAGG(first_name, '/') WITHIN GROUP (ORDER BY salary) first_name  
FROM employees  
GROUP BY department_id;
```



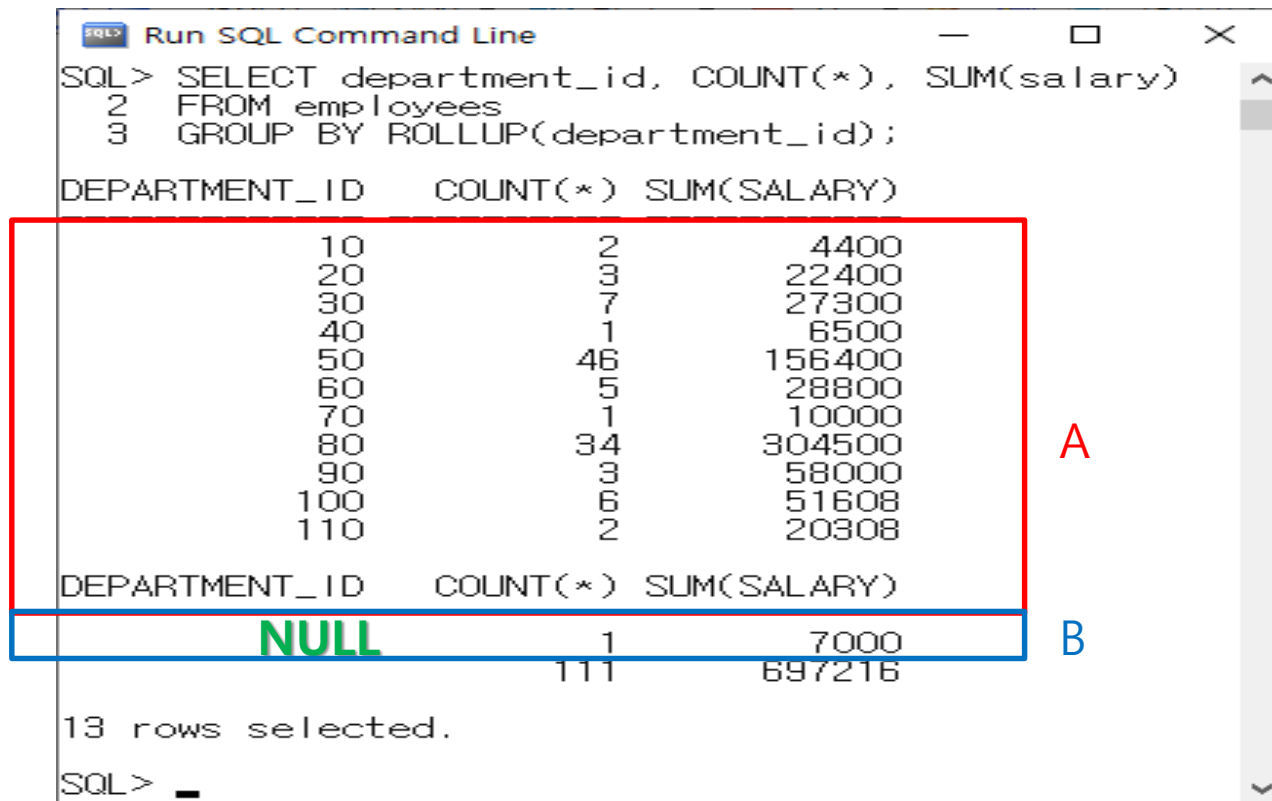
```
Run SQL Command Line  
SQL> SELECT department_id, SUM(salary), LISTAGG(first_name, '/') WITHIN GROUP (ORDER BY salary) first_name  
2 FROM employees  
3 GROUP BY department_id;  
DEPARTMENT_ID SUM(SALARY)  
FIRST_NAME  
-----  
Jennifer/Mickey 10 4400  
Mary/Pat/Michael 20 22400  
Andy/Karen/Guy/Sigal/Shelli/Alexander/Den 30 27300  
DEPARTMENT_ID SUM(SALARY)  
FIRST_NAME  
-----  
Susan 40 6500  
TJ/Hazel/Steven/James/Ki/James/Joshua/Martha/Peter/Randall/Donald/Douglas/Randall/Irene/John/Girard/Mozhe/Vance/Michael/Timothy/Anthony/Kevin/AJana/Curti 50 156400  
S/Jean/Julia/Samuel/Stephen/Winston/Jason/Laura/Julia/Trenna/Jennifer/Renske/Kelly/Britney/Sarah/Alexis/Nandita/Kevin/Shanta/Payan/Matthew/Adam/JOH  
N  
Diana/David/Valli/Bruce/Alexander 60 29900  
DEPARTMENT_ID SUM(SALARY)  
FIRST_NAME  
-----  
Hermann 70 10000  
Sundita/Amit/Charles/Sundar/David/Oliver/Sarath/Mattea/Elizabeth/William/Louise/Nanette/Christopher/Lindsey/Jack/Jonathon/Alyssa/Allan/Peter/Danielle/Dav 80 304500  
id/Patrick/Taylor/Harrison/Janette/Peter/Clara/Eleni/Ellen/Gerald/Lisa/Alberto/Karen/John  
Lex/Neena/Steven 90 58000  
DEPARTMENT_ID SUM(SALARY)  
FIRST_NAME  
-----  
Luis/Ismael/Jose Manuel/John/Daniel/Nancy 100 51608  
William/Shelley 110 20308  
Kimberely 120 7000  
DEPARTMENT_ID SUM(SALARY)  
FIRST_NAME  
-----  
12 rows selected.  
SQL>
```

ROLLUP 연산자

- GROUP BY의 확장 기능
- 보고서 작성시 집합에서 통계 및 요약정보 추출에 사용
- GROUP BY 절에 지정된 열 목록에 따라 오른쪽에서 왼쪽방향으로 하나씩 그룹 생성한 후 그룹함수를 생성한 그룹에 적용
- ROLLUP에 지정된 Grouping Columns의 List는 Subtotal을 생성하기 위해 사용
- Grouping Columns의 수를 N이라 할 때 N+1 level의 Subtotal 생성
- ROLLUP 연산자 없이 N차원의 하위 총계 산출시 N+1개의 SELECT 문을 UNION ALL로 연결하여야 함
 - ➔ 모든 SELECT 문이 각각 테이블에 액세스하므로 질의가 비효율적으로 실행됨
 - ➔ ROLLUP 연산자는 단 한번 테이블에 액세스하여 해당 결과 취합
- ROLLUP의 인수는 계층 구조이므로 인수 순서가 바뀌면 수행 결과도 바뀌게 되므로 인수 순서에 주의
- 하위 총계를 산출하는데 필요한 열이 많은 경우 유용

employees 테이블에서 부서별 인원수, 급여의 합 조회시 ROLLUP을 사용하여 총 집계 조회

```
SELECT department_id, COUNT(*), SUM(salary)
FROM employees
GROUP BY ROLLUP(department_id);
```



Run SQL Command Line

```
SQL> SELECT department_id, COUNT(*), SUM(salary)
2 FROM employees
3 GROUP BY ROLLUP(department_id);
```

DEPARTMENT_ID	COUNT(*)	SUM(SALARY)
10	2	4400
20	3	22400
30	7	27300
40	1	6500
50	46	156400
60	5	28800
70	1	10000
80	34	304500
90	3	58000
100	6	51608
110	2	20308
NULL	111	697216

13 rows selected.

SQL> _

다음 결과 확인

```
SELECT department_id, job_id, manager_id, SUM(salary),
grouping(department_id), grouping(job_id), grouping(manager_id)
FROM employees
GROUP BY ROLLUP(department_id , job_id, manager_id);
```

null이 실제 data인지 여부를 check

0 return : 실제 data인 경우

1 return : 실제 data가 아닌 경우

Run SQL Command Line

```
SQL> SELECT department_id, job_id, manager_id, SUM(salary), grouping(department_id), grouping(job_id), grouping(manager_id)
2 FROM employees
3 GROUP BY ROLLUP(department_id , job_id, manager_id);
```

DEPARTMENT_ID	JOB_ID	MANAGER_ID	SUM(SALARY)	GROUPING(DEPARTMENT_ID)	GROUPING(JOB_ID)	GROUPING(MANAGER_ID)
	SA_REP	149	7000	0	0	0
	SA_REP		7000	0	0	1
	SA_REP		7000	0	1	1
	10 PR_REP	124		0	0	0
	10 PR_REP			0	0	1
	10 AD_ASST	101	4400	0	0	0
	10 AD_ASST		4400	0	0	1
	10 AD_ASST		4400	0	1	1
	20 FI_MGR	114	3400	0	0	0
	20 FI_MGR		3400	0	0	1
	90 AD_VP		34000	0	0	1
	90 AD_PRES		24000	0	0	0
	90 AD_PRES		24000	0	0	1
	90 AD_PRES		58000	0	1	1
	100 FI_MGR	101	12008	0	0	0
	100 FI_MGR		12008	0	0	1
	100 FI_ACCOUNT	108	39600	0	0	0
	110 AC_ACCOUNT		8300	0	0	1
	110 AC_ACCOUNT		20308	0	1	1
			697216	1	1	1

72 rows selected.

SQL>

CUBE 연산자

- 결합 가능한 모든 값에 대하여 다차원 집계 생성
- Grouping Columns이 가질 수 있는 모든 경우에 대하여 Subtotal을 생성해야 하는 경우에 사용하는 것이 바람직 → ROLLUP에 비해 시스템에 많은 부담을 주므로 주의
- 표시된 인수들에 대한 계층별 집계 가능
 - 표시된 인수들 간에는 계층 구조인 ROLLUP과는 달리 평등한 관계이므로 인수의 순서가 바뀌는 경우 정렬 순서는 바뀔 수 있어도 데이터 결과 동일
- 결과에 대한 정렬이 필요한 경우 ORDER BY 절에 명시적으로 정렬 컬럼 표시
- CUBE는 GROUP BY 절에 지정된 모든 그룹의 조합에 대해 하위 총계와 최상의 총계 산출 → ROLLUP은 가능한 하위 총계 조합 중 일부만 산출
- AVG, SUM, MAX, MIN, COUNT를 포함한 모든 그룹 함수에 적용 가능
- CROSS TABULATION 결과 집합 산출에 사용
- GROUP BY 절에 N개의 열이 있을 경우 가능한 상위 집계 조합 수 : 2의 N승

n=3인 경우

- ① GROUP BY a, b, c
- ② GROUP BY a, b
- ③ GROUP BY a, c
- ④ GROUP BY b, c
- ⑤ GROUP BY a
- ⑥ GROUP BY b
- ⑦ GROUP BY c
- ⑧ 총계

다음 실행결과 확인

```
SELECT department_id, job_id, SUM(salary)
FROM employees
WHERE department_id IS NOT NULL
GROUP BY CUBE(department_id , job_id);
```

N이 2인 경우 2의 N승
4종류
A : a, b 데이터
B : a 데이터
C : b 데이터
D : 전체

Run SQL Command Line

```
SQL> SELECT department_id, job_id, SUM(salary)
2 FROM employees
3 WHERE department_id IS NOT NULL
4 GROUP BY CUBE(department_id , job_id);
```

DEPARTMENT_ID	JOB_ID	SUM(SALARY)
D		690216
C	AD_VP	34000
	AC_MGR	12008
	FI_MGR	15408
	HR_REP	6500
	MK_MAN	13000
	MK_REP	6000
	PR_REP	10000
	PU_MAN	11000
	SA_MAN	61000
	SA_REP	243500
B		4400
A	ST_MAN	36400
	AD_ASST	4400
	AD_PRES	24000
	IT_PROG	28800
	PU_CLERK	13900
	SH_CLERK	64300
	ST_CLERK	55700
	AC_ACCOUNT	8300
	FI_ACCOUNT	42000
	PR_REP	10000
B		22400
A	AD_ASST	4400
	FI_MGR	34000
	MK_MAN	13000
	MK_REP	6000
	PU_MAN	11000
	PU_CLERK	13900
	FI_ACCOUNT	24000
	HR_REP	6500
	ST_CLERK	55700
	ST_MAN	36400
B		156400
A	IT_PROG	28800
	PR_REP	10000
	SA_MAN	61000
	SA_REP	243500
	AC_ACCOUNT	8300
	AC_MGR	12008
	AD_PRES	24000
	AD_VP	34000
	FI_ACCOUNT	39600
	FI_MGR	12008
B		20308
A	AC_MGR	12008
	AC_ACCOUNT	8300

53 rows selected.

SQL>

다음 실행결과 확인

```
SELECT department_id, job_id, SUM(salary)
FROM employees
WHERE department_id IS NOT NULL
GROUP BY CUBE(job_id , department_id);
```

N이 2인 경우 2의 N승

4종류

A : a, b 데이터

B : a 데이터

C : b 데이터

D : 전체

DEPARTMENT_ID JOB_ID SUM(SALARY)		
D		690216
	10	4400
	20	22400
	30	27300
	40	6500
	50	156400
	60	28800
	70	10000
	80	304500
	90	58000
	100	51608
DEPARTMENT_ID JOB_ID SUM(SALARY)		
C	110	20308
	AD_VP	34000
	90 AD_VP	34000
	AC_MGR	12008
	100 AC_MGR	12008
	FI_MGR	13408
	20 FI_MGR	3400
	100 FI_MGR	12008
	HR_REP	6500
	40 HR_REP	6500
B	MK_MAN	13000
	DEPARTMENT_ID JOB_ID SUM(SALARY)	
	20 MK_MAN	13000
	MK_REP	6000
	20 MK_REP	6000
	PR_REP	10000
	10 PR_REP	10000
	70 PR_REP	10000
	PU_MAN	11000
	30 PU_MAN	11000
A	SA_MAN	61000
	80 SA_MAN	61000
	SA_REP	243500
	DEPARTMENT_ID JOB_ID SUM(SALARY)	
	80 SA_REP	243500
	ST_MAN	36400
	50 ST_MAN	36400
	AD_ASST	4400
	10 AD_ASST	4400
	AD_PRES	24000
B	90 AD_PRES	24000
	IT_PROG	28800
	60 IT_PROG	28800
	PU_CLERK	13900
	30 PU_CLERK	13900
	DEPARTMENT_ID JOB_ID SUM(SALARY)	
	SH_CLERK	64300
	50 SH_CLERK	64300
	ST_CLERK	55700
	50 ST_CLERK	55700
A	AC_ACCOUNT	8300
	110 AC_ACCOUNT	8300
	FI_ACCOUNT	42000
	30 FI_ACCOUNT	2400
	100 FI_ACCOUNT	39600

53 rows selected.

SQL>

employees 테이블에서 부서번호, 업무, 매니저번호, 급여합 조회시 CUBE를 사용하여 부서내 업무담당 매니저별 집계, 총 집계 조회

```
SELECT department_id, job_id, manager_id, SUM(salary)
FROM employees
GROUP BY CUBE(department_id, job_id, manager_id);
```

100	131800
101	44916
102	9000
103	19800
108	39600
114	17300
120	22100
121	25400

DEPARTMENT_ID	JOB_ID	MANAGER_ID	SUM(SALARY)
		122	23600
		123	25900
		124	23000
		145	51000
		146	51000
		147	46600
		148	51900
		149	7000
		149	50000
		201	6000
		205	8300

DEPARTMENT_ID	JOB_ID	MANAGER_ID	SUM(SALARY)
	AD_VP		34000
	AD_VP	100	34000
	AC_MGR		12008
	AC_MGR	101	12008
	FI_MGR		15408
	FI_MGR	101	12008
	FI_MGR	114	3400
	HR_REP		6500
	HR_REP	101	6500
	MK_MAN		13000
	MK_MAN	100	13000

DEPARTMENT_ID	JOB_ID	MANAGER_ID	SUM(SALARY)
	MK_REP		6000
	MK_REP	201	6000
	PR_REP		10000
	PR_REP	101	10000

N이 3이므로 2의 N승 8종류
group by grouping sets

()
manager_id
job_id
department_id
(manager_id, job_id)
(manager_id, department_id)
(job_id, department_id)
(department_id, job_id, manager_id)

DEPARTMENT_ID	JOB_ID	MANAGER_ID	SUM(SALARY)
		147	46600
		148	51900
		149	43000
	SA_MAN		61000
	SA_MAN	100	61000
	SA_REP		243500
	SA_REP	145	51000
	SA_REP	146	51000
	SA_REP	147	46600
	SA_REP	148	51900
	SA_REP	149	43000

DEPARTMENT_ID	JOB_ID	MANAGER_ID	SUM(SALARY)
			24000
			58000
		100	34000
	AD_VP		34000
	AD_VP	100	34000
	AD_PRES		24000
	AD_PRES		24000
		101	51608
		101	12008
		108	39600
	FI_MGR		12008

DEPARTMENT_ID	JOB_ID	MANAGER_ID	SUM(SALARY)
	FI_MGR	101	12008
	FI_ACCOUNT		39600
	FI_ACCOUNT	108	39600
			20308
		101	12008
		205	8300
	AC_MGR		12008
	AC_MGR	101	12008
	AC_ACCOUNT		8300
	AC_ACCOUNT	205	8300

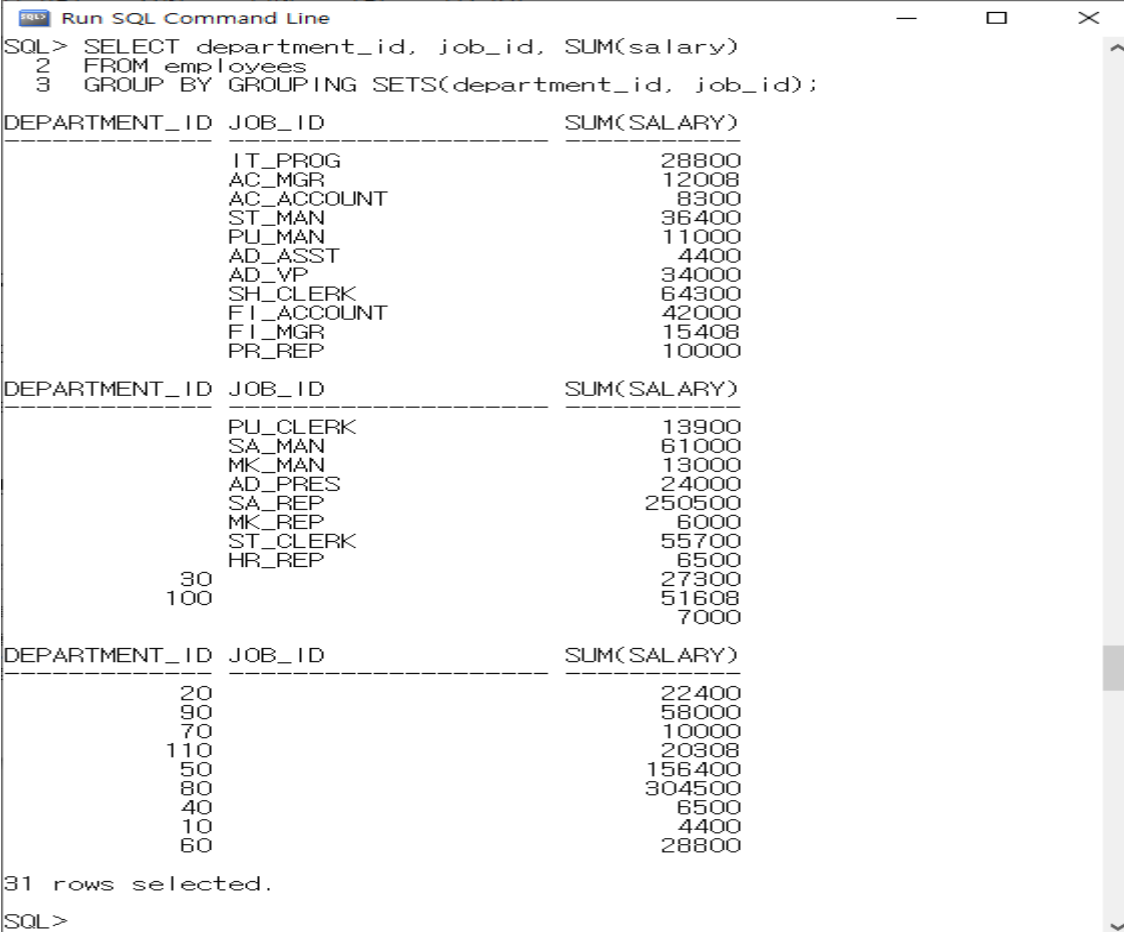
175 rows selected.

GROUPING SETS

- 더욱 다양한 소계 집합 생성 가능
- GROUP BY의 확장 기능
- GROUP BY SQL 문장을 여러 번 반복하지 않아도 원하는 결과를 쉽게 얻을 수 있음
- 인수들에 대한 개별 집계를 구할 수 있음
- 표시된 인수들 간에는 계층 구조인 ROLLUP과는 달리 평등한 관계이므로 인수의 순서가 바뀌어도 결과 동일
- UNION ALL 연산자를 이용하여 여러 개의 SELECT 문을 결합하는 대신 GROUPING SETS를 사용하여 하나의 SELECCT 문으로 작성하여 다양한 그룹화 지정 가능
- 결과에 대한 정렬이 필요한 경우 ORDER BY 이용

다음 실행결과 확인

```
SELECT department_id, job_id, SUM(salary)
FROM employees
GROUP BY GROUPING SETS(department_id, job_id);
```



Run SQL Command Line

```
SQL> SELECT department_id, job_id, SUM(salary)
2 FROM employees
3 GROUP BY GROUPING SETS(department_id, job_id);
```

DEPARTMENT_ID	JOB_ID	SUM(SALARY)
	IT_PROG	28800
	AC_MGR	12008
	AC_ACCOUNT	8300
	ST_MAN	36400
	PL_MAN	11000
	AD_ASST	4400
	AD_VP	34000
	SH_CLERK	64300
	FI_ACCOUNT	42000
	FI_MGR	15408
	PR_REP	10000
	PL_CLERK	13900
	SA_MAN	61000
	MK_MAN	13000
	AD_PRES	24000
	SA_REP	250500
	MK_REP	6000
	ST_CLERK	55700
	HR_REP	6500
30		27300
100		51608
		7000
20		22400
90		58000
70		10000
110		20308
50		156400
80		304500
40		6500
10		4400
60		28800

31 rows selected.

SQL>

다음 실행결과 확인

SELECT department_id, NULL job_id, SUM(salary) FROM employees
GROUP BY department_id

UNION ALL

SELECT NULL department_id, job_id, SUM(salary) FROM employees
GROUP BY job_id;

A UNION ALL B

B UNION ALL A로 변환

DEPARTMENT_ID	JOB_ID	SUM(SALARY)
30		27300
100		51608
		7000
20		22400
90		58000
70		10000
110		20308
50		156400
80		304500
40		6500
10		4400

DEPARTMENT_ID	JOB_ID	SUM(SALARY)
60		28800
	IT_PROG	28800
	AC_MGR	12008
	AC_ACCOUNT	8300
	ST_MAN	36400
	PU_MAN	11000
	AD_ASST	4400
	AD_VP	34000
	SH_CLERK	64300
	FI_ACCOUNT	42000
	FI_MGR	15408

DEPARTMENT_ID	JOB_ID	SUM(SALARY)
	PR_REP	10000
	PU_CLERK	13900
	SA_MAN	61000
	MK_MAN	13000
	AD_PRES	24000
	SA_REP	250500
	MK_REP	6000
	ST_CLERK	55700
	HR_REP	6500

31 rows selected.

DEPARTMENT_ID	JOB_ID	SUM(SALARY)
	IT_PROG	28800
	AC_MGR	12008
	AC_ACCOUNT	8300
	ST_MAN	36400
	PU_MAN	11000
	AD_ASST	4400
	AD_VP	34000
	SH_CLERK	64300
	FI_ACCOUNT	42000
	FI_MGR	15408
	PR_REP	10000

DEPARTMENT_ID	JOB_ID	SUM(SALARY)
	PU_CLERK	13900
	SA_MAN	61000
	MK_MAN	13000
	AD_PRES	24000
	SA_REP	250500
	MK_REP	6000
	ST_CLERK	55700
	HR_REP	6500
30		27300
100		51608
		7000

DEPARTMENT_ID	JOB_ID	SUM(SALARY)
20		22400
90		58000
70		10000
110		20308
50		156400
80		304500
40		6500
10		4400
60		28800

31 rows selected.

```
SELECT department_id, job_id, manager_id, AVG(salary)
FROM employees
GROUP BY GROUPING SETS((department_id, job_id, manager_id), department_id, job_id);
```

```

Run SQL Command Line
DEPARTMENT_ID JOB_ID MANAGER_ID AVG(SALARY)
-----
90 AD_VP 100 17000
110 AC_MGR 101 12008
20 FI_MGR 114 24000
100 FI_MGR 101 12008
40 HR_REP 101 6500
20 MK_MAN 100 13000
20 MK_REP 201 6000
10 PR_ASST 124 4400
70 PR_REP 101 10000
30 PU_MAN 100 11000
80 SA_MAN 100 12200

DEPARTMENT_ID JOB_ID MANAGER_ID AVG(SALARY)
-----
80 SA_REP 149 7000
80 SA_REP 145 8500
80 SA_REP 146 8500
80 SA_REP 147 7766.66667
80 SA_REP 148 8600
80 SA_REP 149 8600
50 ST_MAN 100 7200
10 AD_ASST 101 4400
90 AD_PRES 102 24000
60 IT_PROG 103 9000
60 IT_PROG 103 4950

DEPARTMENT_ID JOB_ID MANAGER_ID AVG(SALARY)
-----
30 PU_CLERK 114 2780
20 SH_CLERK 120 2900
50 SH_CLERK 120 3675
50 SH_CLERK 120 3600
50 SH_CLERK 123 3475
50 SH_CLERK 124 3625
50 ST_CLERK 101 6500
50 ST_CLERK 101 3675
50 ST_CLERK 103 3700
50 ST_CLERK 103 3000
50 ST_CLERK 124 2925

DEPARTMENT_ID JOB_ID MANAGER_ID AVG(SALARY)
-----
110 AC_ACCOUNT 205 8300
30 FI_ACCOUNT 100 2400
10 FI_ACCOUNT 108 1920
AD_VP 100 17000
AC_MGR 101 12008
FI_MGR 114 24000
HR_REP 101 6500
MK_MAN 100 13000
MK_REP 201 6000
PR_REP 101 10000
PU_MAN 100 11000

DEPARTMENT_ID JOB_ID MANAGER_ID AVG(SALARY)
-----
SA_MAN 12200
SA_REP 8350
ST_MAN 7200
AD_ASST 4400
AD_PRES 24000
IT_PROG 9000
PU_CLERK 2780
SH_CLERK 3675
ST_CLERK 2785
AC_ACCOUNT 8300
FI_ACCOUNT 7000

DEPARTMENT_ID JOB_ID MANAGER_ID AVG(SALARY)
-----
100 8601.33333
30 3900
90 7000
20 19333.3333
70 7466.66667
110 10000
40 10154
80 6500
80 8955.88235
50 3475.55556
100 4400

DEPARTMENT_ID JOB_ID MANAGER_ID AVG(SALARY)
-----
60 5760

```

앞의 예제를 UNION ALL을 이용하여 수정

```
SELECT department_id, job_id, manager_id, AVG(salary)
FROM employees
GROUP BY (department_id, job_id, manager_id)
UNION ALL
SELECT department_id, NULL job_id, NULL manager_id, AVG(salary)
FROM employees
GROUP BY department_id
UNION ALL
SELECT NULL department_id, job_id, NULL manager_id, AVG(salary)
FROM employees
GROUP BY job_id;
```

Run SQL Command Line

```
SQL> SELECT department_id, job_id, manager_id, AVG(salary)
2 FROM employees
3 GROUP BY (department_id, job_id, manager_id)
4 UNION ALL
5 SELECT department_id, NULL job_id, NULL manager_id, AVG(salary)
6 FROM employees
7 GROUP BY department_id
8 UNION ALL
9 SELECT NULL department_id, job_id, NULL manager_id, AVG(salary)
10 FROM employees
11 GROUP BY job_id;
```

DEPARTMENT_ID	JOB_ID	MANAGER_ID	AVG(SALARY)
80	SA_REP	145	8650
50	SH_CLERK	120	2900
10	PR_REP	124	2400
90	AD_VP	100	17000
30	FI_ACCOUNT	100	24000
90	AD_PRES	102	9000
60	IT_PROG	100	7280
50	ST_MAN	120	2825
50	ST_CLERK	121	2675
50	ST_CLERK	122	2700
50	ST_CLERK	123	3000
50	ST_CLERK	124	2925
20	MK_REP	201	6000
60	IT_PROG	103	4950
50	SH_CLERK	124	2825
100	FI_ACCOUNT	108	7920
30	PU_MGR	100	11000
80	SA_REP	147	7766.66667
50	SH_CLERK	121	3675
40	HR_REP	101	8500
80	SA_REP	146	8500
80	SA_REP	149	7000
20	MK_MGR	100	13000
70	PR_REP	101	10000
110	AC_MGR	101	12008
30	PU_CLERK	114	2780
80	SA_MGR	100	12200
80	SA_REP	145	8500
80	SA_REP	149	8600
50	SH_CLERK	122	3200
50	SH_CLERK	123	3475
10	AD_ASST	101	4400
20	FI_MGR	114	8400
100	FI_MGR	101	12008
110	AC_ACCOUNT	205	8300
30			3900
100			8801.93333
			7000
20			7466.66667
90			19333.3333
70			10000
110			10184
50			9475.55556
80			8965.88335
40			8500
10			4400
60			5760
	IT_PROG		5760
	AC_MGR		12008
	AC_ACCOUNT		8300
	ST_MGR		7280
	PU_MGR		11000
	AD_ASST		4400
	AD_VP		17000
50	SH_CLERK		3215
	FI_ACCOUNT		7000
	FI_MGR		7704
	PR_REP		10000
	PU_CLERK		2780
	SA_MGR		12200
	MK_MGR		13000
	AD_PRES		24000
	SA_REP		8350
	MK_REP		6000
	ST_CLERK		2785
40	HR_REP		8500

87 rows selected.

GROUPING SETS

- ROLLUP과 CUBE를 GROUPING SETS을 사용한 결과로 표시

함수	설 명
ROLLUP(a, b, c)	GROUPING SETS ((a, b, c), (a, b), (a), ())
CUBE(a, b, c)	GROUPING SETS ((a, b, c), (a, b), (a, c), (b, c), (a), (b), (c), ())

- GROUPING SETS 문장은 다음 문장의 UNION ALL 결과

GROUPING SETS 문	GROUP BY 문
GROUP BY GROUPING SETS(a, b, c)	GROUP BY a UNION ALL GROUP BY b UNION ALL GROUP BY c
GROUP BY GROUPING SETS(a, b, (b, c))	GROUP BY a UNION ALL GROUP BY b UNION ALL GROUP BY b, c
GROUP BY GROUPING SETS((a, b, c))	GROUP BY a, b, c
GROUP BY GROUPING SETS(a, (b), ())	GROUP BY a UNION ALL GROUP BY b UNION ALL GROUP BY ()



조합 열을 사용한 결과 조회

```
SELECT department_id, job_id, manager_id, SUM(salary)
FROM employees
GROUP BY ROLLUP(department_id, (job_id, manager_id));
```

Run SQL Command Line

```
SQL> SELECT department_id, job_id, manager_id, SUM(salary)
2 FROM employees
3 GROUP BY ROLLUP(department_id, (job_id, manager_id));
```

DEPARTMENT_ID	JOB_ID	MANAGER_ID	SUM(SALARY)
	SA_REP	149	7000
			7000
10	PR_REP	124	4400
10	AD_ASST	101	4400
200	FI_MGR	114	3400
200	MK_MAN	100	13000
200	MK_REP	201	6000
30	PU_MAN	100	22400
30	PU_CLERK	114	11000
			13900
40	FI_ACCOUNT	100	2400
40			27300
50	HR_REP	101	6500
50	ST_MAN	100	36400
50	SH_CLERK	120	11600
50	SH_CLERK	121	14700
50	SH_CLERK	122	12900
50	SH_CLERK	123	13900
50	SH_CLERK	124	11300
50	ST_CLERK	120	10500
60	ST_CLERK	121	10700
60	ST_CLERK	122	10800
60	ST_CLERK	123	12000
60	ST_CLERK	124	11700
			156400
60	IT_PROG	102	9000
60	IT_PROG	103	19800
70	PR_REP	101	28800
70			10000
80	SA_MAN	100	61000
80	SA_REP	145	51000
80	SA_REP	146	51000
80	SA_REP	147	46600
80	SA_REP	148	51900
80	SA_REP	149	43000
			304500
90	AD_VP	100	34000
90	AD_PRES		24000
100	FI_MGR	101	58000
100	FI_ACCOUNT	108	12008
			39608
110	AC_MGR	101	51608
110	AC_ACCOUNT	205	12008
			8300
			20308
			697216

49 rows selected.
SQL> _

1. GROUP BY department_id, job_id, manager_id
2. GROUP BY department_id
3. GROUP BY null



다음 결과 조회

```
SELECT department_id, job_id, manager_id, SUM(salary)
FROM employees
GROUP BY GROUPING SETS(department_id, (job_id, manager_id));
```

Run SQL Command Line

```
SQL> SELECT department_id, job_id, manager_id, SUM(salary)
2 FROM employees
3 GROUP BY GROUPING SETS(department_id, (job_id, manager_id));
```

DEPARTMENT_ID	JOB_ID	MANAGER_ID	SUM(SALARY)
	FI_MGR	114	3400
	SH_CLERK	122	12800
	AC_MGR	101	12008
	ST_MAN	100	36400
	ST_CLERK	121	10700
	SA_REP	148	51900
	SH_CLERK	120	11600
	SH_CLERK	124	11300
	MK_MAN	100	13000
	AD_PRES		24000
	FI_MGR	101	12008
	SA_REP	146	51000
	SH_CLERK	123	13900
	AD_ASST	101	4400
	IT_PROG	102	9000
	IT_PROG	103	19800
	FI_ACCOUNT	108	39600
	PJ_MAN	100	11000
	ST_CLERK	122	10800
	SA_REP	145	51000
	AC_ACCOUNT	205	8300
	FI_ACCOUNT	100	2400
	PR_REP	124	
	AD_VP	100	34000
	ST_CLERK	120	10500
	ST_CLERK	124	11700
	SA_REP	147	46600
	SA_REP	149	50000
	HR_REP	101	6500
	PR_REP	101	10000
	ST_CLERK	123	12000
	SH_CLERK	121	14700
	PJ_CLERK	114	13900
	SA_MAN	100	61000
	MK_REP	201	6000
30			27300
100			51608
			7000
20			22400
30			58000
70			10000
110			20308
50			156400
80			304500
40			6500
10			4400
60			28800

47 rows selected.

SQL> _

1. GROUP BY job_id, manager_id
2. GROUP BY department_id

조합이 아닌,
하나의 group by 로 인식