

04. 그룹 함수

SUM, AVG, MIN, MAX, COUNT 그룹 함수를 학습한다.

칼럼의 값 별로 그룹 함수의 결과 값을 구하는 GROUP BY 절을 학습한다.

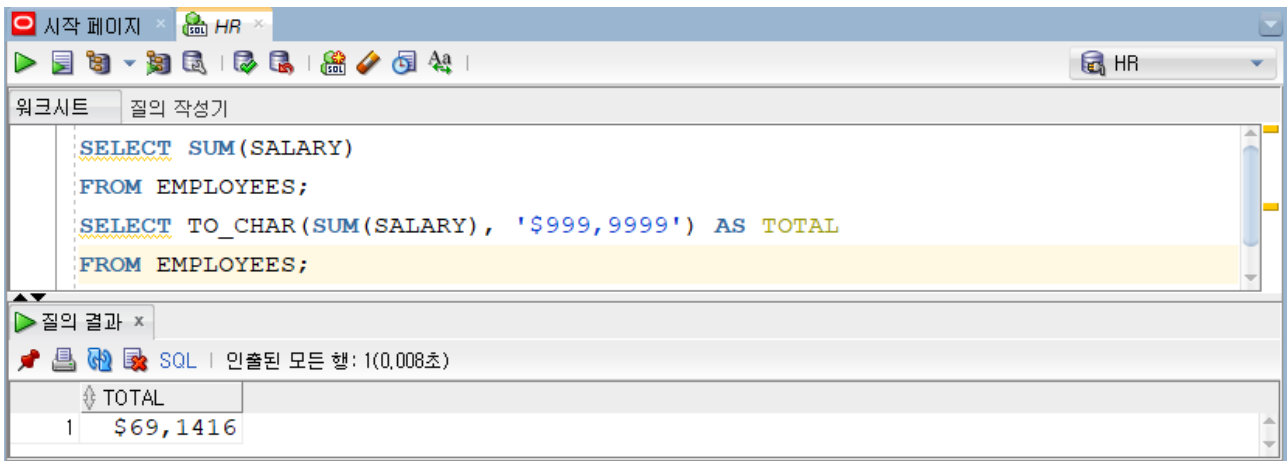
그룹의 결과를 제한할 때는 HAVING 절을 학습한다.

1) 그룹 함수의 종류

구 분	설 명
SUM	그룹의 누적 합계를 반환한다.
AVG	그룹의 평균을 반환한다.
MAX	그룹의 최댓값을 반환한다.
MIN	그룹의 최솟값을 반환한다.
COUNT	그룹의 총 개수를 반환한다.

<예> 직원의 총 급여 구하기(SUM함수)

```
SELECT SUM(SALARY)
FROM EMPLOYEES;
SELECT TO_CHAR(SUM(SALARY), '$999,9999') AS TOTAL
FROM EMPLOYEES;
```

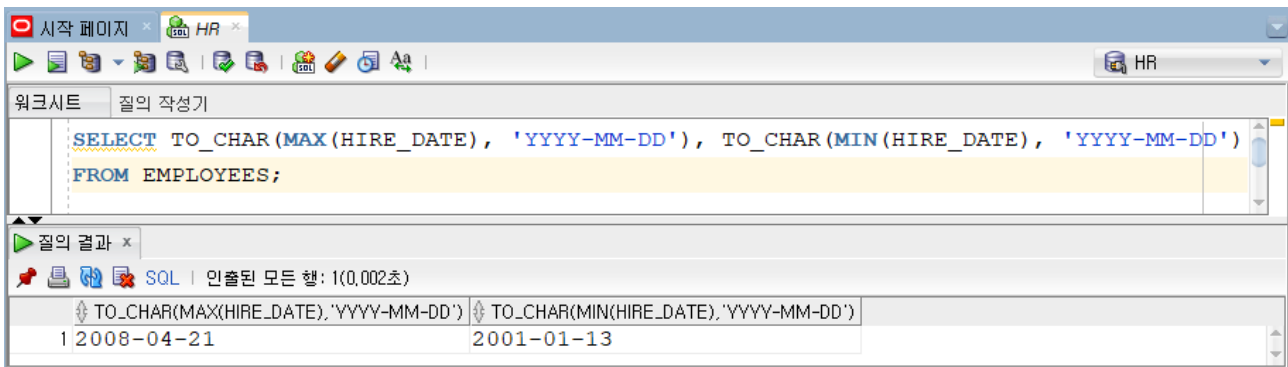


<예> 직원의 평균 급여 구하기 (AVG함수)

SELECT AVG(SALARY) FROM EMPLOYEES;	<pre>AVG(SALARY) 1 6461.831775700934579439252336448598130841</pre>
SELECT ROUND(AVG(SALARY),1) FROM EMPLOYEES;	<pre>ROUND(AVG(SALARY),1) 1 6461.8</pre>

<예> 최근에 입사한 사원과 가장 오래전에 입사한 직원의 입사일 출력하기(MAX/MIN 함수)

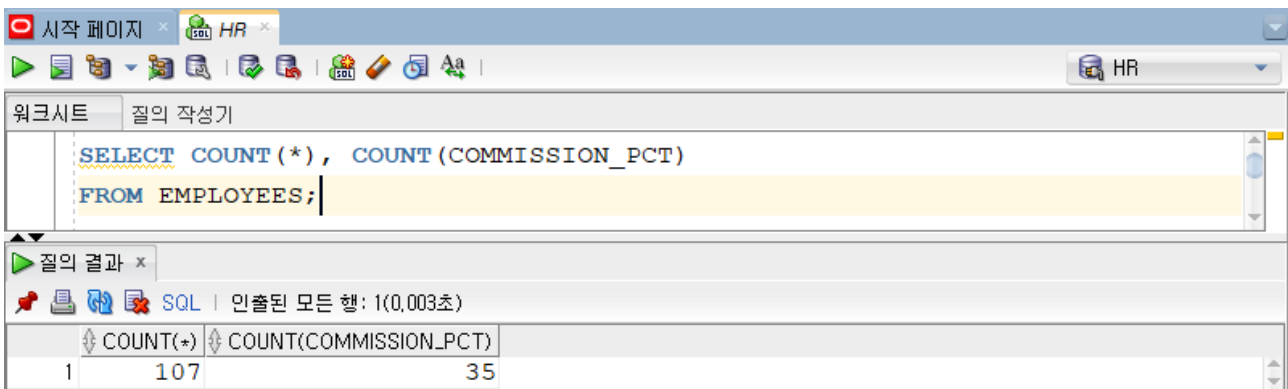
```
SELECT TO_CHAR(MAX(HIRE_DATE), 'YYYY-MM-DD'), TO_CHAR(MIN(HIRE_DATE), 'YYYY-MM-DD')
FROM EMPLOYEES;
```



NULL을 저장한 컬럼과 연산한 결과는 NULL이다. 그러나 그룹함수는 다른 연산자와는 달리, 해당 컬럼 값이 NULL인 것을 제외하고 계산하기 때문에 결과를 NULL로 반환하지 않는다. 그래서 로우(레코드) 개수 구하는 COUNT 함수는 NULL 값에 대해서는 세지 않는다.

<예> 전체 사원의 수와 커미션을 받는 사원의 수

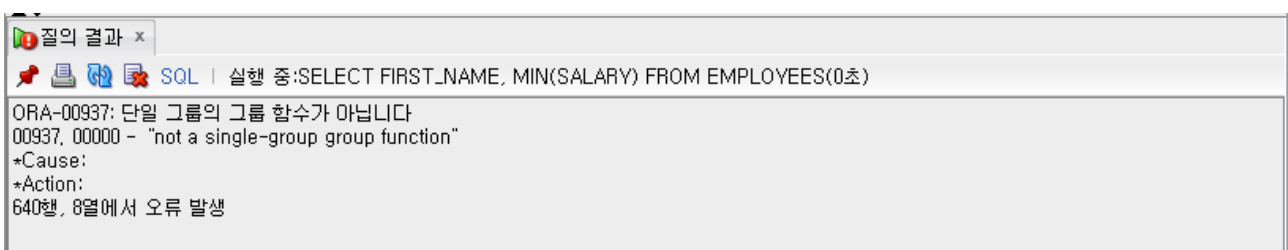
```
SELECT COUNT(*), COUNT(COMMISSION_PCT)
FROM EMPLOYEES;
```



<문제> JOB의 종류가 몇 개인지 즉, 중복되지 않은 직업의 개수를 구해보자.

※ 컬럼과 그룹 함수를 같이 사용할 때 유의할 점

```
SELECT FIRST_NAME, MIN(SALARY) FROM EMPLOYEES;
```



MIN(SALARY) 결과치	FIRST_NAME
800	SMITH
	ALLEN
	WARD
	KING

에러가 발생하는 이유는 위의 그림처럼 그룹함수의 결과값은 하나인데 비해 그룹함수를 적용하지 않은 단순 칼럼의 로우 개수는 107개로 각각 산출되는 로우가 달라 둘을 매치시킬 수가 없기 때문이다. 즉, 800이라는 값을 SMITH에 붙일 수도 없고 ALLEN에 붙일 수도 없기 때문이다. 이렇듯 SELECT 문에 그룹 함수를 사용하는 경우 그룹 함수를 적용하지 않은 단순 칼럼은 올 수 없다.

2) GROUP BY 절을 사용해 특정 조건으로 세부적인 그룹화하기

GROUP BY 절은 일반적으로 쿼리문으로부터 얻은 결과에 대해 GROUP BY 절에 명시한 컬럼의 값이 같을 때 그룹을 만들고, 이 그룹으로부터 SQL 표준 함수인 그룹함수를 통해 다양한 결과는 얻는다.

```
SELECT 컬럼명, 그룹함수(컬럼명)
FROM 테이블명
WHERE 조건문
GROUP BY 컬럼명
```

- 특정 그룹으로 묶어 데이터 집계 시 사용
- WHERE와 ORDER BY절 사이에 위치
- 집계(그룹)함수와 함께 사용
- SELECT 리스트에서 집계(그룹)함수를 제외한 모든 컬럼과 표현식은 GROUP BY 절에 명시해야 함

직원들을 부서번호를 기준으로

```
SELECT DEPARTMENT_ID
FROM EMPLOYEES
GROUP BY DEPARTMENT_ID
ORDER BY DEPARTMENT_ID;
```

질의 결과 x	
SQL 인출된 모든 행: 12(0.013초)	
DEPARTMENT_ID	
1	10
2	20
3	30
4	40
5	50
6	60
7	70
8	80
9	90
10	100
11	110
12	(null)

<예> 부서별 최대 급여와 최소 급여 구하기

```
SELECT DEPARTMENT_ID, MAX(SALARY) "최대 급여", MIN(SALARY) "최소 급여"
FROM EMPLOYEES
GROUP BY DEPARTMENT_ID;
```

The screenshot shows the SQL Developer interface with a query window and a results window. The query is as follows:

```
SELECT DEPARTMENT_ID, MAX(SALARY) "최대 급여", MIN(SALARY) "최소 급여"
FROM EMPLOYEES
GROUP BY DEPARTMENT_ID
ORDER BY DEPARTMENT_ID;
```

The results window displays the following data:

DEPARTMENT_ID	최대 급여	최소 급여
10	4400	4400
20	13000	6000
30	11000	2500
40	6500	6500
50	8200	2100
60	9000	4200
70	10000	10000
80	14000	6100
90	24000	17000
100	12008	6900
110	12008	8300
(null)	7000	7000

```
SELECT DEPARTMENT_ID, SALARY
FROM EMPLOYEES
ORDER BY DEPARTMENT_ID, SALARY DESC;
```

DEPARTMENT_ID	SALARY
10	4400
20	13000
20	6000
30	11000
30	3100
30	2900
30	2800
30	2600
30	2500
40	6500

최대값

최소값

```
SELECT DEPARTMENT_ID, MAX(SALARY) "
최대 급여", MIN(SALARY) "최소 급여"
FROM EMPLOYEES
GROUP BY DEPARTMENT_ID
ORDER BY DEPARTMENT_ID;
```

DEPARTMENT_ID	최대 급여	최소 급여
10	4400	4400
20	13000	6000
30	11000	2500
40	6500	6500
50	8200	2100
60	9000	4200
70	10000	10000
80	14000	6100
90	24000	17000
100	12008	6900
110	12008	8300
(null)	7000	7000

```
SELECT DEPARTMENT_ID, SUM(SALARY), AVG(SALARY)
FROM EMPLOYEES
GROUP BY DEPARTMENT_ID
ORDER BY DEPARTMENT_ID;
```

<문제> 부서별로 직원의 수와 커미션을 받는 직원의 수를 카운트해 보자.

SELECT 절에 조건을 사용하여 결과를 제한할 때는 WHERE 절을 사용하지만, 그룹의 결과를 제한할 때는 HAVING절을 사용한다.

- ```
SELECT DEPARTMENT_ID, AVG(SALARY)
FROM EMPLOYEES
GROUP BY DEPARTMENT_ID
ORDER BY DEPARTMENT_ID;
```

[illegible]

```
SELECT DEPARTMENT_ID, AVG(SALARY)
FROM EMPLOYEES
GROUP BY DEPARTMENT_ID
HAVING AVG(SALARY) >= 5000
ORDER BY DEPARTMENT_ID;
```

질의 결과 x

SQL | 인출된 모든 행: 9(0,004초)

|   | DEPARTMENT_ID | AVG(SALARY)                               |
|---|---------------|-------------------------------------------|
| 1 | 20            | 9500                                      |
| 2 | 40            | 6500                                      |
| 3 | 60            | 5760                                      |
| 4 | 70            | 10000                                     |
| 5 | 80            | 8955.882352941176470588235294117647058824 |
| 6 | 90            | 19333.3333333333333333333333333333333333  |
| 7 | 100           | 8601.3333333333333333333333333333333333   |
| 8 | 110           | 10154                                     |
| 9 | (null)        | 7000                                      |

```
SELECT DEPARTMENT_ID, MAX(SALARY), MIN(SALARY)
FROM EMPLOYEES
GROUP BY DEPARTMENT_ID
HAVING MAX(SALARY) > 5000
ORDER BY DEPARTMENT_ID;
```

The screenshot shows the SQL Developer interface. The top pane contains the following SQL query:

```

SELECT DEPARTMENT_ID, MAX(SALARY), MIN(SALARY)
FROM EMPLOYEES
GROUP BY DEPARTMENT_ID
HAVING MAX(SALARY) > 5000
ORDER BY DEPARTMENT_ID;

```

The bottom pane shows the query results in a table with 11 rows. The status bar indicates that 11 rows were returned in 0.004 seconds.

|    | DEPARTMENT_ID | MAX(SALARY) | MIN(SALARY) |
|----|---------------|-------------|-------------|
| 1  | 20            | 13000       | 6000        |
| 2  | 30            | 11000       | 2500        |
| 3  | 40            | 6500        | 6500        |
| 4  | 50            | 8200        | 2100        |
| 5  | 60            | 9000        | 4200        |
| 6  | 70            | 10000       | 10000       |
| 7  | 80            | 14000       | 6100        |
| 8  | 90            | 24000       | 17000       |
| 9  | 100           | 12008       | 6900        |
| 10 | 110           | 12008       | 8300        |
| 11 | (null)        | 7000        | 7000        |

#### 4) ROLLUP (expr1, expr2, ...)

- GROUP BY 절에서 사용됨
- expr로 명시한 표현식을 기준으로 집계한 결과, 추가 정보 집계
- expr로 명시한 표현식 수와 순서에 따라 레벨 별로 집계

부서별, 직무별, 급여의 합

(부서코드가 바뀔때마다 부서별 집계 결과가 출력되고 모든 부서가 출력되면 전체 집계 정보가 출력된다.)

```

SELECT DEPARTMENT_ID, JOB_ID, COUNT(*), SUM(SALARY)
FROM EMPLOYEES
GROUP BY ROLLUP(DEPARTMENT_ID, JOB_ID)
ORDER BY DEPARTMENT_ID;

```

시작 페이지 \* PDB\_HR.sql \*  
 SQL 워크시트(W) 내역  
 워크시트 | 질의 작성기

```

SELECT DEPARTMENT_ID, JOB_ID, COUNT(*), SUM(SALARY)
FROM EMPLOYEES
GROUP BY ROLLUP(DEPARTMENT_ID, JOB_ID)
ORDER BY DEPARTMENT_ID;

```

질의 결과 \*  
 SQL | 인출된 모든 행: 33(0,014초)

|   | DEPARTMENT_ID | JOB_ID   | COUNT(*) | SUM(SALARY) |
|---|---------------|----------|----------|-------------|
| 1 | 10            | AD_ASST  | 1        | 4400        |
| 2 | 10            | (null)   | 1        | 4400        |
| 3 | 20            | MK_MAN   | 1        | 13000       |
| 4 | 20            | MK_REP   | 1        | 6000        |
| 5 | 20            | (null)   | 2        | 19000       |
| 6 | 30            | PU_CLERK | 5        | 13900       |
| 7 | 30            | PU_MAN   | 1        | 11000       |
| 8 | 30            | (null)   | 6        | 24900       |

```

SELECT JOB_ID, DEPARTMENT_ID, COUNT(*), SUM(SALARY)
FROM EMPLOYEES
GROUP BY ROLLUP(JOB_ID, DEPARTMENT_ID) ORDER BY JOB_ID;

```

시작 페이지 \* PDB\_HR.sql \*  
 SQL 워크시트(W) 내역  
 워크시트 | 질의 작성기

```

SELECT JOB_ID, DEPARTMENT_ID, COUNT(*), SUM(SALARY)
FROM EMPLOYEES
GROUP BY ROLLUP(JOB_ID, DEPARTMENT_ID)
ORDER BY JOB_ID;

```

질의 결과 \*  
 SQL | 인출된 모든 행: 40(0,011초)

|   | JOB_ID     | DEPARTMENT_ID | COUNT(*) | SUM(SALARY) |
|---|------------|---------------|----------|-------------|
| 1 | AC_ACCOUNT | 110           | 1        | 8300        |
| 2 | AC_ACCOUNT | (null)        | 1        | 8300        |
| 3 | AC_MGR     | 110           | 1        | 12008       |
| 4 | AC_MGR     | (null)        | 1        | 12008       |
| 5 | AD_ASST    | 10            | 1        | 4400        |
| 6 | AD_ASST    | (null)        | 1        | 4400        |
| 7 | AD_PRES    | 90            | 1        | 24000       |
| 8 | AD_PRES    | (null)        | 1        | 24000       |



5) CUBE(exp1, exp2, ...)

CUBE는 명시한 표현식 개수에 따라 가능한 모든 조합별로 집계한 결과를 반환한다.

```
SELECT DEPARTMENT_ID, JOB_ID, COUNT(*), SUM(SALARY)
FROM EMPLOYEES
GROUP BY CUBE(DEPARTMENT_ID, JOB_ID);
```

질의 결과 x

SQL | 인출된 모든 행: 52(0.008초)

|    | DEPARTMENT_ID | JOB_ID     | COUNT(+) | SUM(SALARY) |
|----|---------------|------------|----------|-------------|
| 1  | 10            | AD_ASST    | 1        | 4400        |
| 2  | 10            | (null)     | 1        | 4400        |
| 3  | 20            | MK_MAN     | 1        | 13000       |
| 4  | 20            | MK_REP     | 1        | 6000        |
| 5  | 20            | (null)     | 2        | 19000       |
| 6  | 30            | PU_CLERK   | 5        | 13900       |
| 7  | 30            | PU_MAN     | 1        | 11000       |
| 8  | 30            | (null)     | 6        | 24900       |
| 9  | 40            | HR_REP     | 1        | 6500        |
| 10 | 40            | (null)     | 1        | 6500        |
| 11 | 50            | SH_CLERK   | 20       | 64300       |
| 12 | 50            | ST_CLERK   | 20       | 55700       |
| 13 | 50            | ST_MAN     | 5        | 36400       |
| 14 | 50            | (null)     | 45       | 156400      |
| 15 | 60            | IT_PROG    | 5        | 28800       |
| 16 | 60            | (null)     | 5        | 28800       |
| 17 | 70            | PR_REP     | 1        | 10000       |
| 18 | 70            | (null)     | 1        | 10000       |
| 19 | 80            | SA_MAN     | 5        | 61000       |
| 20 | 80            | SA_REP     | 29       | 243500      |
| 21 | 80            | (null)     | 34       | 304500      |
| 22 | 90            | AD_PRES    | 1        | 24000       |
| 23 | 90            | AD_VP      | 2        | 34000       |
| 24 | 90            | (null)     | 3        | 58000       |
| 25 | 100           | FI_ACCOUNT | 5        | 39600       |
| 26 | 100           | FI_MGR     | 1        | 12008       |
| 27 | 100           | (null)     | 6        | 51608       |
| 28 | 110           | AC_ACCOUNT | 1        | 8300        |
| 29 | 110           | AC_MGR     | 1        | 12008       |
| 30 | 110           | (null)     | 2        | 20308       |
| 31 | (null)        | AC_ACCOUNT | 1        | 8300        |
| 32 | (null)        | AC MGR     | 1        | 12008       |
| 33 | (null)        | AD_ASST    | 1        | 4400        |
| 34 | (null)        | AD_PRES    | 1        | 24000       |
| 35 | (null)        | AD_VP      | 2        | 34000       |
| 36 | (null)        | FI_ACCOUNT | 5        | 39600       |

## 6) 집합 연산자

데이터 집합이 대상이므로 집합 연산자(UNION, UNION ALL, INTERSECT, MINUS)를 사용할 때 데이터 집합의 수는 한 개 이상을 사용할 수 있다. 즉 여러 개의 SELECT문을 연결해 또 다른 하나의 쿼리를 만드는 역할을 하는 것이 집합 연산자이다.

```
CREATE TABLE exp_goods_asia (-- 한국과 일본의 10대 수출품 테이블
 country VARCHAR2(10), -- 나라명
 seq NUMBER, -- 번호
 goods VARCHAR2(80) -- 상품명
);

INSERT INTO exp_goods_asia VALUES ('한국', 1, '원유제외 석유류');
INSERT INTO exp_goods_asia VALUES ('한국', 2, '자동차');
INSERT INTO exp_goods_asia VALUES ('한국', 3, '전자집적회로');
INSERT INTO exp_goods_asia VALUES ('한국', 4, '선박');
INSERT INTO exp_goods_asia VALUES ('한국', 5, 'LCD');
INSERT INTO exp_goods_asia VALUES ('한국', 6, '자동차부품');
INSERT INTO exp_goods_asia VALUES ('한국', 7, '휴대전화');
INSERT INTO exp_goods_asia VALUES ('한국', 8, '환식탄화수소');
INSERT INTO exp_goods_asia VALUES ('한국', 9, '무선송신기 디스플레이 부속품');
INSERT INTO exp_goods_asia VALUES ('한국', 10, '철 또는 비합금강');

INSERT INTO exp_goods_asia VALUES ('일본', 1, '자동차');
INSERT INTO exp_goods_asia VALUES ('일본', 2, '자동차부품');
INSERT INTO exp_goods_asia VALUES ('일본', 3, '전자집적회로');
INSERT INTO exp_goods_asia VALUES ('일본', 4, '선박');
INSERT INTO exp_goods_asia VALUES ('일본', 5, '반도체웨이퍼');
INSERT INTO exp_goods_asia VALUES ('일본', 6, '화물차');
INSERT INTO exp_goods_asia VALUES ('일본', 7, '원유제외 석유류');
INSERT INTO exp_goods_asia VALUES ('일본', 8, '건설기계');
INSERT INTO exp_goods_asia VALUES ('일본', 9, '다이오드, 트랜지스터');
INSERT INTO exp_goods_asia VALUES ('일본', 10, '기계류');
```

### ① UNION

UNION은 합집합을 의미한다. 예를 들어, 두 개의 데이터 집합이 있으면 각 집합 원소(SELECT 결과)를 모두 포함한 결과가 반환된다.

```
SELECT *
FROM EXP_GOODS_ASIA
WHERE COUNTRY='한국';
```

|    | COUNTRY | SEQ | GOODS           |
|----|---------|-----|-----------------|
| 1  | 한국      | 1   | 원유제외 석유류        |
| 2  | 한국      | 2   | 자동차             |
| 3  | 한국      | 3   | 전자집적회로          |
| 4  | 한국      | 4   | 선박              |
| 5  | 한국      | 5   | LCD             |
| 6  | 한국      | 6   | 자동차부품           |
| 7  | 한국      | 7   | 휴대전화            |
| 8  | 한국      | 8   | 환식탄화수소          |
| 9  | 한국      | 9   | 무선송신기 디스플레이 부속품 |
| 10 | 한국      | 10  | 철 또는 비합금강       |

```
SELECT *
FROM EXP_GOODS_ASIA
WHERE COUNTRY='일본';
```

|    | COUNTRY | SEQ | GOODS       |
|----|---------|-----|-------------|
| 1  | 일본      | 1   | 자동차         |
| 2  | 일본      | 2   | 자동차부품       |
| 3  | 일본      | 3   | 전자집적회로      |
| 4  | 일본      | 4   | 선박          |
| 5  | 일본      | 5   | 반도체웨이퍼      |
| 6  | 일본      | 6   | 화물차         |
| 7  | 일본      | 7   | 원유제외 석유류    |
| 8  | 일본      | 8   | 건설기계        |
| 9  | 일본      | 9   | 다이오드, 트랜지스터 |
| 10 | 일본      | 10  | 기계류         |

국가에 상관없이 모든 수출품을 조회하는데, 단 자동차나 선박과 같이 두 국가가 겹치는 수출품목은 한 번만 조회되도록 하려면 UNION(합집합 개념)을 사용한다.

```
SELECT GOODS
FROM EXP_GOODS_ASIA
WHERE COUNTRY='한국'
UNION
SELECT GOODS
FROM EXP_GOODS_ASIA
WHERE COUNTRY='일본';
```

|    | GOODS           |
|----|-----------------|
| 1  | LCD             |
| 2  | 건설기계            |
| 3  | 기계류             |
| 4  | 다이오드, 트랜지스터     |
| 5  | 무선송신기 디스플레이 부속품 |
| 6  | 반도체웨이퍼          |
| 7  | 선박              |
| 8  | 원유제외 석유류        |
| 9  | 자동차             |
| 10 | 자동차부품           |
| 11 | 전자집적회로          |
| 12 | 철 또는 비합금강       |
| 13 | 화물차             |
| 14 | 환식탄화수소          |
| 15 | 휴대전화            |

[예제] hr 스키마에 있는 JOB\_HISTORY 테이블은 직원들의 업무 변경 이력을 나타내는 테이블이다. 이 정보를 이용하여 모든 직원의 현재 및 이전의 업무 이력 정보를 출력하고자 한다. 중복된 직원정보가 있을 경우 한 번만 표시하여 출력하시오.

```
SELECT employee_id, job_id
FROM employees
UNION
SELECT employee_id, job_id
FROM job_history;
```

|    | EMPLOYEE_ID | JOB_ID     |
|----|-------------|------------|
| 1  | 100         | AD_PRES    |
| 2  | 101         | AC_ACCOUNT |
| 3  | 101         | AC_MGR     |
| 4  | 101         | AD_VP      |
| 5  | 102         | AD_VP      |
| 6  | 102         | IT_PROG    |
| 7  | 103         | IT_PROG    |
| 8  | 104         | IT_PROG    |
| 9  | 105         | IT_PROG    |
| 10 | 106         | IT_PROG    |
| 11 | 107         | IT_PROG    |
| 12 | 108         | FI_MGR     |
| 13 | 109         | FI_ACCOUNT |
| 14 | 110         | FI_ACCOUNT |
| 15 | 111         | FI_ACCOUNT |
| 16 | 112         | FI_ACCOUNT |
| 17 | 113         | FI_ACCOUNT |
| 18 | 114         | PU_MAN     |
| 19 | 114         | ST_CLERK   |
| 20 | 115         | PU_CLERK   |

[예제] 위 결과를 이용하여 출력된 176번 직원의 업무 이력의 변경 날짜 이력을 조회하시오.

```
SELECT employee_id, job_id, NULL AS "Start Date", NULL AS "End Date"
FROM employees
WHERE employee_id = 176
UNION
SELECT employee_id, job_id, start_date, end_date
FROM job_history
WHERE employee_id = 176;
```

|   | EMPLOYEE_ID | JOB_ID | Start Date | End Date |
|---|-------------|--------|------------|----------|
| 1 | 176         | SA_MAN | 07/01/01   | 07/12/31 |
| 2 | 176         | SA_REP | 06/03/24   | 06/12/31 |
| 3 | 176         | SA_REP | (null)     | (null)   |

## ② UNION ALL

UNION ALL은 중복된 항목도 모두 조회된다는 점이다.

```
SELECT GOODS
FROM EXP_GOODS_ASIA
WHERE COUNTRY='한국'
UNION ALL
SELECT GOODS
FROM EXP_GOODS_ASIA
WHERE COUNTRY='일본';
```

| GOODS             |
|-------------------|
| 1 원유제외 석유류        |
| 2 자동차             |
| 3 전자집적회로          |
| 4 선박              |
| 5 LCD             |
| 6 자동차부품           |
| 7 휴대전화            |
| 8 환식탄화수소          |
| 9 무선송신기 디스플레이 부속품 |
| 10 철 또는 비합금강      |
| 11 자동차            |
| 12 자동차부품          |
| 13 전자집적회로         |
| 14 선박             |
| 15 반도체웨이퍼         |
| 16 화물차            |
| 17 원유제외 석유류       |
| 18 건설기계           |
| 19 다이오드, 트랜지스터    |
| 20 기계류            |

<문제> 위 예제에서 각 사원의 업무 이력 정보를 확인하였다. 하지만 모든 사원의 업무 이력 전체를 보지는 못했다. 여기에서는 모든 사원의 업무 이력 변경 정보 및 업무 변경에 따른 부서정보를 사번이 빠른 순서대로 출력하시오.

## ③ INTERSECT

INTERSECT는 합집합이 아닌 교집합을 의미한다. 즉 데이터 집합에서 공통된 항목만 추출해 낸다.

```
SELECT GOODS
FROM EXP_GOODS_ASIA
WHERE COUNTRY='한국'
INTERSECT
SELECT GOODS
FROM EXP_GOODS_ASIA
WHERE COUNTRY='일본';
```

| GOODS      |
|------------|
| 1 선박       |
| 2 원유제외 석유류 |
| 3 자동차      |
| 4 자동차부품    |
| 5 전자집적회로   |

<문제>사원정보(Employees) 테이블에 JOB\_ID는 사원의 현재 업무를 뜻하고, JOB\_HISTORY에 JOB\_ID는 사원의 이전 업무를 뜻한다. 이 두 테이블을 교차해보면 업무가 변경된 사원의 정보도 볼 수 있지만 이전에 한 번 했던 같은 업무를 그대로 하고 있는 사원의 정보도 볼 수 있다. 이전에 한 번 했던 같은 업무를 보고 있는 사원의 사번과 업무를 출력하시오.

#### ④ MINUS

MINUS는 차집합을 의미한다. 즉 한 데이터 집합을 기준으로 다른 데이터 집합과 공통된 항목을 제외한 결과만 추출해 낸다.

```
SELECT GOODS
FROM EXP_GOODS_ASIA
WHERE COUNTRY='한국'
MINUS
SELECT GOODS
FROM EXP_GOODS_ASIA
WHERE COUNTRY='일본';
```

| GOODS             |
|-------------------|
| 1 LCD             |
| 2 무선송신기 디스플레이 부속품 |
| 3 철 또는 비합금강       |
| 4 환식탄화수소          |
| 5 휴대전화            |

<문제> 우리 회사는 1년에 한 번 업무를 변경하여 전체적인 회사 업무를 사원들이 익히도록 하는 Role change 정책을 시행하고 있다. 이번 인사이동 때 아직 업무가 변경된 적이 없는 사원들을 적합한 업무로 이동시키려고 한다. HR 부서의 사원정보 테이블과 업무이력정보 테이블을 이용하여 한 번도 업무가 변경되지 않은 사원의 사번을 출력하시오.

#### ⑤ 집합 연산자의 제한 사항

- 집합 연산자로 연결되는 각 SELECT문의 SELECT 리스트의 개수와 데이터 타입은 일치해야 한다.

```
-- ORA-01789: 질의 블록은 부정확한 수의 결과 열을 가지고 있습니다.
SELECT GOODS
FROM EXP_GOODS_ASIA
WHERE COUNTRY='한국'
UNION
SELECT SEQ, GOODS
FROM EXP_GOODS_ASIA
WHERE COUNTRY='일본';
```

```
-- ORA-01790: 대응하는 식과 같은 데이터 유형이어야 합니다
SELECT SEQ
FROM EXP_GOODS_ASIA
WHERE COUNTRY='한국'
```

```

UNION
SELECT GOODS
FROM EXP_GOODS_ASIA
WHERE COUNTRY='일본';

```

- 집합 연산자로 SELECT문을 연결할 때 ORDER BY 절은 맨 마지막 문장에서만 사용할 수 있다.

-- **ORA-00933: SQL 명령어가 올바르게 종료되지 않았습니다**

```

SELECT GOODS
FROM EXP_GOODS_ASIA
WHERE COUNTRY='한국'
ORDER BY GOODS
UNION
SELECT GOODS
FROM EXP_GOODS_ASIA
WHERE COUNTRY='일본';

```

-- 위의 쿼리문을 수정하면

```

SELECT GOODS
FROM EXP_GOODS_ASIA
WHERE COUNTRY='한국'
UNION
SELECT GOODS
FROM EXP_GOODS_ASIA
WHERE COUNTRY='일본'
ORDER BY GOODS;

```

| GOODS             |
|-------------------|
| 1 LCD             |
| 2 건설기계            |
| 3 기계류             |
| 4 다이오드, 트랜지스터     |
| 5 무선송신기 디스플레이 부속품 |
| 6 반도체웨이퍼          |
| 7 선박              |
| 8 원유제외 석유류        |
| 9 자동차             |
| 10 자동차부품          |
| 11 전자집적회로         |
| 12 철 또는 비합금강      |
| 13 화물차            |
| 14 환식탄화수소         |
| 15 휴대전화           |

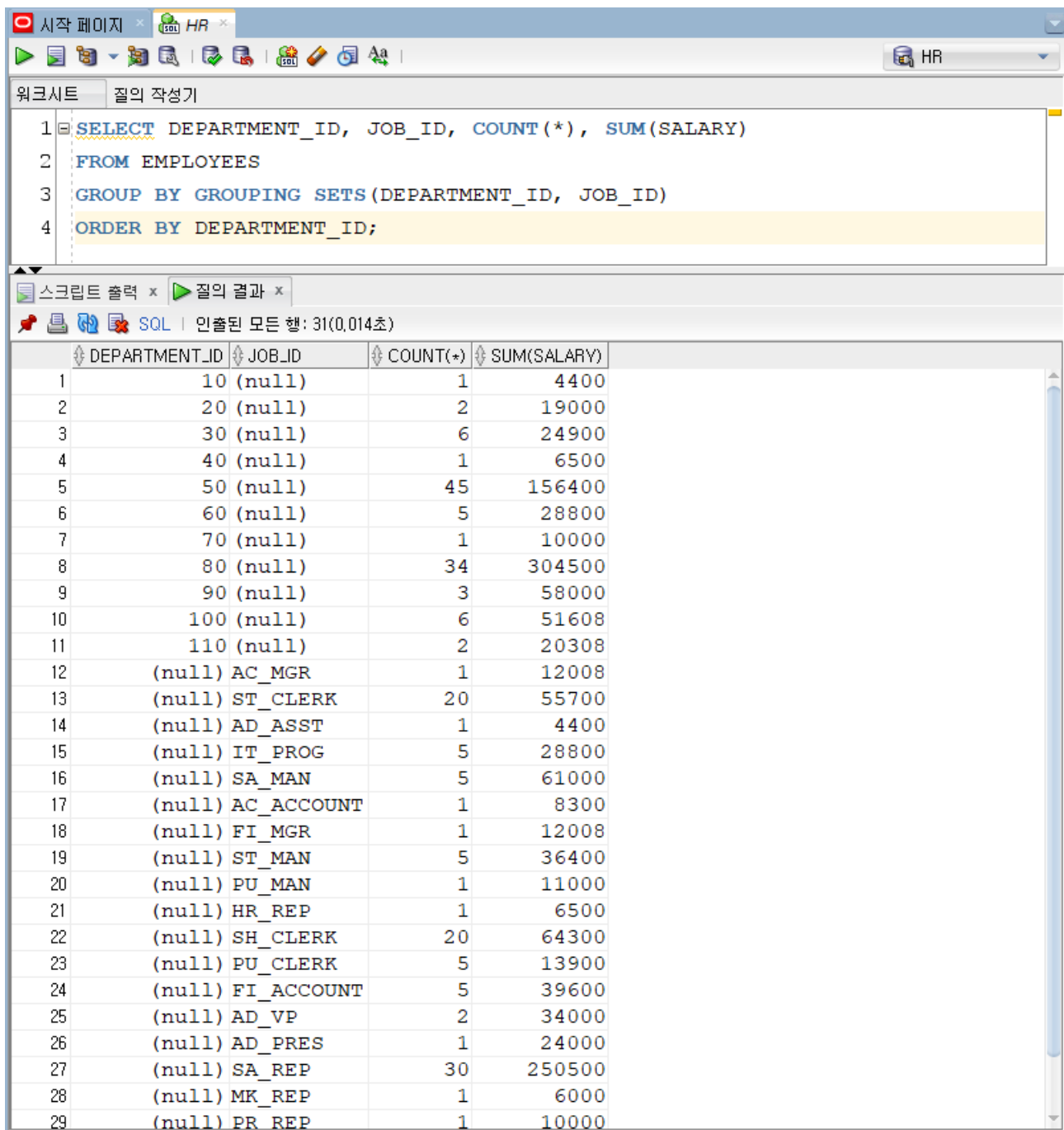
- UNION, INTERSECT, MINUS 연산자는 LONG형 컬럼에는 사용할 수 없다.

## ⑥ GROUPING SETS절

GROUPING SETS은 ROLLUP이나 CUBE처럼 GROUP BY 절에 명시해서 그룹 쿼리에 사용되는 절이다. GROUPING SETS은 그룹 쿼리이긴 하나 UNION ALL 개념이 섞여 있기 때문이다. GROUPING SETS(exp1, exp2, exp3)를 GROUP BY절에 명시했을 때, 괄호 안에 있는 세 표현식별로 각각 집계가 이루어진다.

즉 쿼리 결과는 ((GROUP BY exp1) UNION ALL (GROUP BY exp2) UNION ALL (GROUP BY exp3)) 형태가 된다.

```
SELECT DEPARTMENT_ID, JOB_ID, COUNT(*), SUM(SALARY)
FROM EMPLOYEES
GROUP BY GROUPING SETS(DEPARTMENT_ID, JOB_ID)
ORDER BY DEPARTMENT_ID;
```



스크립트 출력 x | 질의 결과 x

SQL | 인출된 모든 행: 31(0,014초)

|    | DEPARTMENT_ID | JOB_ID     | COUNT(+) | SUM(SALARY) |
|----|---------------|------------|----------|-------------|
| 1  |               | 10 (null)  | 1        | 4400        |
| 2  |               | 20 (null)  | 2        | 19000       |
| 3  |               | 30 (null)  | 6        | 24900       |
| 4  |               | 40 (null)  | 1        | 6500        |
| 5  |               | 50 (null)  | 45       | 156400      |
| 6  |               | 60 (null)  | 5        | 28800       |
| 7  |               | 70 (null)  | 1        | 10000       |
| 8  |               | 80 (null)  | 34       | 304500      |
| 9  |               | 90 (null)  | 3        | 58000       |
| 10 |               | 100 (null) | 6        | 51608       |
| 11 |               | 110 (null) | 2        | 20308       |
| 12 | (null)        | AC_MGR     | 1        | 12008       |
| 13 | (null)        | ST_CLERK   | 20       | 55700       |
| 14 | (null)        | AD_ASST    | 1        | 4400        |
| 15 | (null)        | IT_PROG    | 5        | 28800       |
| 16 | (null)        | SA_MAN     | 5        | 61000       |
| 17 | (null)        | AC_ACCOUNT | 1        | 8300        |
| 18 | (null)        | FI_MGR     | 1        | 12008       |
| 19 | (null)        | ST_MAN     | 5        | 36400       |
| 20 | (null)        | PU_MAN     | 1        | 11000       |
| 21 | (null)        | HR_REP     | 1        | 6500        |
| 22 | (null)        | SH_CLERK   | 20       | 64300       |
| 23 | (null)        | PU_CLERK   | 5        | 13900       |
| 24 | (null)        | FI_ACCOUNT | 5        | 39600       |
| 25 | (null)        | AD_VP      | 2        | 34000       |
| 26 | (null)        | AD_PRES    | 1        | 24000       |
| 27 | (null)        | SA_REP     | 30       | 250500      |
| 28 | (null)        | MK_REP     | 1        | 6000        |
| 29 | (null)        | PR_REP     | 1        | 10000       |