

Ch 4.

Additional SELECT

4.1 | 함수 Function

4.2 | **Additional SELECT Option**

- 1) ORDER BY
- 2) GROUP BY
- 3) HAVING
- 4) JOIN
- 5) SET Operator
- 6) Subquery

4.2.1 ORDER BY

SELECT 구문 실행 결과를 특정 컬럼 값 기준으로 정렬할 때 사용

```
SELECT ...  
FROM ...  
WHERE ...  
ORDER BY 기준1 [ ASC | DESC ] [, 기준2 [ASC | DESC], ... ];
```

[구문 설명]

- 항상 SELECT 구문의 맨 마지막에 위치
- 지정한 컬럼 값을 기준으로 정렬
- 정렬 조건
 - ASC : '오름차순'(기본 값)
 - DESC : '내림차순'
- 여러 개의 정렬 기준을 사용하면 기술된 순서대로 적용
- 기준 별로 정렬 조건 구분 적용 가능
- 컬럼 이름, 컬럼 별칭, 컬럼 기술 순서로 표현 가능

4.2.1 ORDER BY - 사용 예

```
SELECT  EMP_NAME, SALARY
FROM    EMPLOYEE
WHERE   DEPT_ID = '50'
OR      DEPT_ID IS NULL
ORDER BY SALARY DESC;
```

EMP_NAME	SALARY
김순이	3400000
박하일	2600000
나승원	2300000
심하균	2300000
김예수	2100000
성해교	1900000
염정하	1500000
정지현	1500000

```
SELECT  EMP_NAME, HIRE_DATE, DEPT_ID
FROM    EMPLOYEE
WHERE   HIRE_DATE > TO_DATE('20030101', 'YYYYMMDD')
ORDER BY DEPT_ID DESC, HIRE_DATE, EMP_NAME;
```

EMP_NAME	HIRE_DATE	DEPT_ID
염정하	03/09/17	
심하균	04/09/30	
강중훈	04/04/30	90
한선기	90/04/01	90
염정하	04/07/21	80
성해교	03/08/16	50
정지현	04/07/15	50
박하일	04/11/10	50
이중기	04/10/01	20
감우섭	05/07/31	20
고승우	03/04/11	10

4.2.1 ORDER BY - 사용 예

```

SELECT  EMP_NAME AS 이름,
        HIRE_DATE AS 입사일,
        DEPT_ID AS 부서코드
FROM    EMPLOYEE
WHERE   HIRE_DATE > TO_DATE('20030101','YYYYMMDD')
ORDER BY 부서코드 DESC, 입사일, 이름;

```

컬럼 별칭을 사용한 경우 별칭 사용 가능

```

SELECT  EMP_NAME AS 이름,
        HIRE_DATE AS 입사일,
        DEPT_ID AS 부서코드
FROM    EMPLOYEE
WHERE   HIRE_DATE > TO_DATE('20030101','YYYYMMDD')
ORDER BY 3 DESC, 2, 1;

```

SELECT 절에 기술된 순서로 표시 가능

이름	입사일	부서코드
염정하	03/09/17	
심하균	04/09/30	
강중훈	04/04/30	90
한선기	90/04/01	90
엄정하	04/07/21	80
성해교	03/08/16	50
정지현	04/07/15	50
박하일	04/11/10	50
이중기	04/10/01	20
감우섭	05/07/31	20
고승우	03/04/11	10

4.2.2 GROUP BY - 하위 데이터 그룹 개념

EMP_NAME	SALARY	DEPT_ID
권상후	3410000	10
임영애	2640000	10
고승우	1500000	10
김술오	2500000	20
이중기	2500000	20
감우섭	2500000	20
김순이	3400000	50
나승원	2300000	50
김예수	2100000	50
정지현	1500000	50
성해교	1900000	50
박하일	2600000	50
정도연	2600000	60
조재형	3800000	60
안석규	3500000	60
전우성	2090000	80
엄정하	2420000	80
한선기	9000000	90
최만식	3600000	90
강중훈	5500000	90
염정하	1500000	
심하균	2300000	

EMP_NAME	SALARY	DEPT_ID
권상후	3410000	10
임영애	2640000	10
고승우	1500000	10
김술오	2500000	20
이중기	2500000	20
감우섭	2500000	20
김순이	3400000	50
나승원	2300000	50
김예수	2100000	50
정지현	1500000	50
성해교	1900000	50
박하일	2600000	50
정도연	2600000	60
조재형	3800000	60
안석규	3500000	60
전우성	2090000	80
엄정하	2420000	80
한선기	9000000	90
최만식	3600000	90
강중훈	5500000	90
염정하	1500000	
심하균	2300000	

하위그룹1

하위그룹2

하위그룹3

하위그룹4

하위그룹5

하위그룹6

하위그룹7

DEPT_ID	SUM(SALARY)
	3800000
50	13800000
20	7500000
10	7550000
90	18100000
80	4510000
60	9900000

- 각 부서가 하나의 데이터 그룹
- 부서 수 만큼 데이터 그룹 생성
- 그룹 별(부서 별)로 그룹함수 적용

4.2.2 GROUP BY

```
SELECT ...  
FROM ...  
WHERE ...  
GROUP BY column_name | expr  
ORDER BY 기준1 [ ASC | DESC ] [, 기준2 [ ASC | DESC ], ... ];
```

[구문 설명]

- GROUP BY 절에 기술한 컬럼이나 표현식을 기준으로 데이터 그룹 생성
- 각 그룹 별로 SELECT 절에 기술한 그룹 함수가 적용
- SELECT 절에 기술한 컬럼 중, 그룹 함수에 사용되지 않은 컬럼은 GROUP BY 절에 반드시 기술되어야 함
- 제약 사항
 - WHERE 절에는 그룹 함수를 사용할 수 없음
 - GROUP BY 절에는 컬럼 이름만 사용 가능(별칭, 순서 사용 불가)

4.2.2 GROUP BY

```
SELECT  DEPT_ID AS 부서,  
        ROUND(AVG(SALARY),-4) AS 평균급여  
FROM    EMPLOYEE  
GROUP BY DEPT_ID  
ORDER BY 1;
```

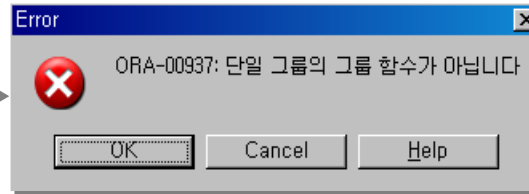
부서	평균급여
10	2520000
20	2500000
50	2300000
60	3300000
80	2260000
90	6030000
	1900000

```
SELECT  DECODE(SUBSTR(EMP_NO,8,1),  
              '1', '남', '3', '남', '여') AS 성별,  
        ROUND(AVG(SALARY),-4) AS 평균급여  
FROM    EMPLOYEE  
GROUP BY DECODE(SUBSTR(EMP_NO,8,1),  
              '1', '남', '3', '남', '여')  
ORDER BY 2;
```

성별	평균급여
여	2260000
남	3360000

4.2.2 GROUP BY 사용 예

```
SELECT DEPT_ID, COUNT(*)  
FROM   EMPLOYEE;
```



'ORA-00937' 에러가 발생하면
GROUP BY 가 누락되었는지
확인

그룹 함수 COUNT는 유일한 반환 값만 생성할 수 있는데, SELECT 절에서는 여러 개의 DEPT_ID 값을 함께 표시하도록 작성하였음 → 그룹 함수가 동작해야 하는 그룹을 찾지 못한 경우

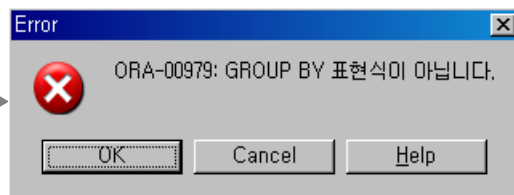
[수정 구문]

```
SELECT  DEPT_ID,  
        COUNT(*)  
FROM    EMPLOYEE  
GROUP BY DEPT_ID  
ORDER BY 1;
```

DEPT_ID	COUNT(*)
10	3
20	3
50	6
60	3
80	2
90	3
	2

4.2.2 GROUP BY 사용 예

```
SELECT EMP_NAME, DEPT_ID, COUNT(*)
FROM   EMPLOYEE
GROUP BY DEPT_ID;
```



'ORA-00979' 에러가 발생하면
GROUP BY 절에 누락된 select
list가 있는지 확인

[수정 구문]

```
SELECT   EMP_NAME, DEPT_ID, COUNT(*)
FROM     EMPLOYEE
GROUP BY EMP_NAME, DEPT_ID;
```



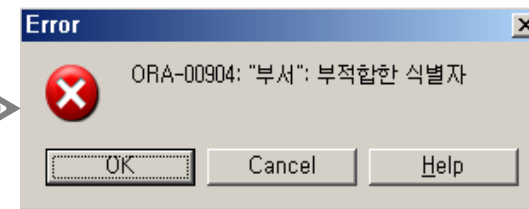
결과(일부)

EMP_NAME	DEPT_ID	COUNT(*)
조재형	60	1
성해교	50	1
심하균		1
김예수	50	1
박하일	50	1
임영애	10	1
염정하		1
이중기	20	1
전우성	80	1
권상후	10	1

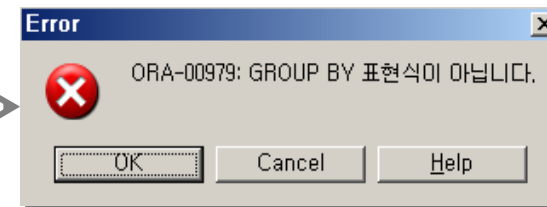
4.2.2 GROUP BY 사용 예

컬럼 별칭이나 컬럼 기술 순서는 사용할 수 없음

```
SELECT  DEPT_ID AS 부서,  
        SUM(SALARY)  
FROM    EMPLOYEE  
GROUP BY 부서;
```



```
SELECT  DEPT_ID AS 부서,  
        SUM(SALARY)  
FROM    EMPLOYEE  
GROUP BY 1;
```



4.2.2 GROUP BY 사용 예

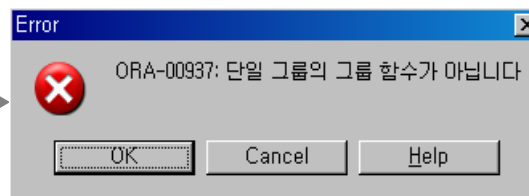
```
SELECT  MAX(SUM(SALARY))  
FROM    EMPLOYEE  
GROUP BY DEPT_ID;
```

MAX(SUM(SALARY))
181000000

그룹 함수는 2번까지
중첩 사용 가능

DEPT_ID	SUM(SALARY)
	38000000
50	138000000
20	75000000
10	75500000
90	181000000
80	45100000
60	99000000

```
SELECT  DEPT_ID,  
        MAX(SUM(SALARY))  
FROM    EMPLOYEE  
GROUP BY DEPT_ID;
```




4.2.3 HAVING

GROUP BY에 의해 그룹화 된 데이터에 대한 그룹 함수 실행 결과를 제한하기 위해 사용
(WHERE는 테이블에 포함된 원본 데이터를 제한하기 위해 사용)

```
SELECT ...  
FROM ...  
WHERE ...  
GROUP BY column_name | expr  
HAVING condition  
ORDER BY 기준1 [ ASC | DESC] [, 기준2 [ASC | DESC], ... ];
```

4.2.3 HAVING

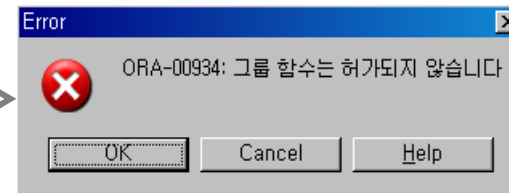
```
SELECT  DEPT_ID, SUM(SALARY)
FROM    EMPLOYEE
GROUP BY DEPT_ID
HAVING  SUM(SALARY) > 9000000;
```



DEPT_ID	SUM(SALARY)
50	13800000
90	18100000
60	9900000

부서 별 급여 총합을 계산한 결과 중 9000000 이상인 경우만 선택

```
SELECT  DEPT_ID, SUM(SALARY)
FROM    EMPLOYEE
WHERE    SUM(SALARY) > 9000000
GROUP BY DEPT_ID;
```



WHERE 절에는 그룹 함수를 사용할 수 없음

→ WHERE 절이 수행되어야 그룹 함수가 실행 될 대상 그룹이 결정

4.2.4 JOIN - 필요성

'EMP_NAME' 컬럼은 EMPLOYEE 테이블에 있고, 'DEPT_NAME' 컬럼은 DEPARTMENT 테이블에 있을 때 두 컬럼을 한번에 볼 수 있는 방법이 필요함

	EMP_NAME	DEPT_ID
EMPLOYEE	한선기	90
	강중훈	90
	최만식	90
	정도연	60
	안석규	60
	조재형	60
	정지현	50
	김예수	50
	나승원	50
	김순이	50
	성해교	50
	전우성	80
	엄정하	80
	심하균	
	고승우	10
	박하일	50
	권상후	10
	임영애	10
	염정하	
	김술오	20
	이중기	20
	감우섭	20

DEPT_ID	DEPT_NAME
20	회계팀
10	본사 인사팀
50	해외영업1팀
60	기술지원팀
80	해외영업2팀
90	해외영업3팀
30	마케팅팀

DEPARTMENT

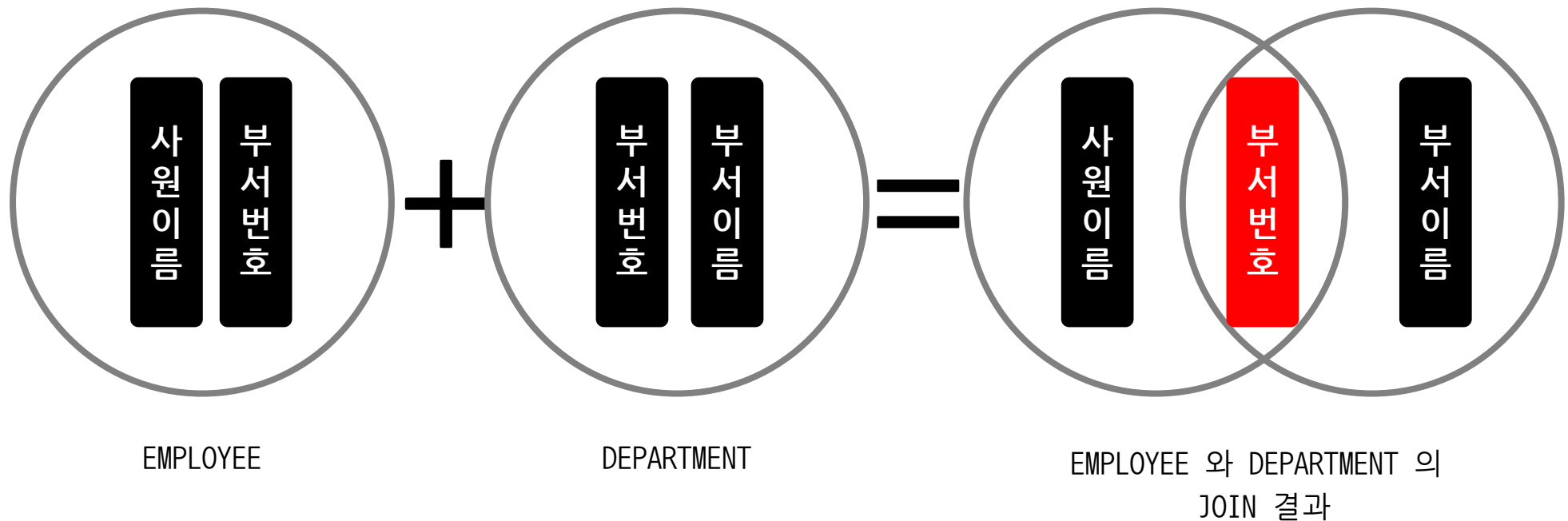
EMP_NAME	DEPT_NAME
감우섭	회계팀
이중기	회계팀
김술오	회계팀
임영애	본사 인사팀
권상후	본사 인사팀
고승우	본사 인사팀
박하일	해외영업1팀
성해교	해외영업1팀
김순이	해외영업1팀
나승원	해외영업1팀
김예수	해외영업1팀
정지현	해외영업1팀
조재형	기술지원팀
안석규	기술지원팀
정도연	기술지원팀
엄정하	해외영업2팀
전우성	해외영업2팀
최만식	해외영업3팀
강중훈	해외영업3팀
한선기	해외영업3팀
염정하	
심하균	

????

4.2.4 JOIN - 개념

서로 연관되고 다른 테이블에 존재하는 컬럼들을 한번에 조회하기 위해 사용하는 대표적인 기법

[JOIN 개념 도식 1]



4.2.4 JOIN - 오라클 전용 구문

```

SELECT EMP_NAME, DEPT_NAME
FROM   EMPLOYEE E,
       DEPARTMENT D
WHERE  E.DEPT_ID = D.DEPT_ID;

```

- FROM 절에 조회 대상 테이블을 심표로 구분하여 기술
- WHERE 절에 테이블 사이의 관계를 표시하는 조건 기술
- 동일한 이름의 컬럼이 여러 테이블에 존재하는 경우
→ SELECT 절/WHERE 절에 컬럼 이름을 기술할 때 어떤
테이블에 포함된 컬럼인지 구분해서 표시
- 테이블 이름도 별칭을 사용할 수 있다

EMPLOYEE(일부)

EMP_NAME	DEPT_ID
한선기	90
강중훈	90
최만식	90
정도연	60
안석규	60
조재형	60
정지현	50
김예수	50
나승원	50
김순이	50
성해교	50

DEPARTMENT

DEPT_ID	DEPT_NAME
20	회계팀
10	본사 인사팀
50	해외영업1팀
60	기술지원팀
80	해외영업2팀
90	해외영업3팀
30	마케팅팀

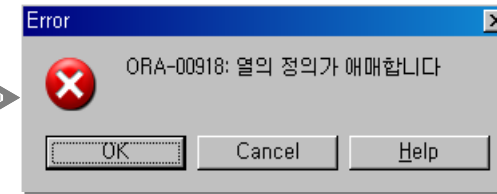
????(일부)

EMP_NAME	DEPT_NAME
감우섭	회계팀
이중기	회계팀
김술오	회계팀
임영애	본사 인사팀
권상후	본사 인사팀
고승우	본사 인사팀
박하일	해외영업1팀
성해교	해외영업1팀
김순이	해외영업1팀

4.2.4 JOIN - 오라클 전용 구문

```
SELECT EMP_NAME, DEPT_NAME  
FROM   EMPLOYEE, DEPARTMENT  
WHERE  DEPT_ID = DEPT_ID;
```

```
SELECT EMP_NAME, DEPT_ID,  
       DEPT_NAME  
FROM   EMPLOYEE E,  
       DEPARTMENT D  
WHERE  E.DEPT_ID = D.DEPT_ID;
```



양쪽 테이블에 동일한 이름의 컬럼이 모두 존재하는 경우

- WHERE 절: 테이블 구분 필요(테이블 별칭 사용 가능)
- SELECT 절: 양쪽 테이블의 컬럼 값은 동일하지만, 문법 상 어떤 테이블의 컬럼 값을 표시할 것인지 구분 필요

4.2.4 JOIN - ANSI 표준 구문

- JOIN 유형을 세분화
- WHERE 절에서 JOIN 조건을 별도로 분리하고 'JOIN' 키워드를 명시적으로 사용


```
SELECT ...  
FROM   table1  
{[INNER] JOIN table2 ON (condition1 [AND condition2 ...]) |  
  [INNER] JOIN table2 USING (column1 [, ...]) |  
  NATURAL [INNER] JOIN table2 |  
  LEFT|RIGHT|FULL [OUTER] JOIN table2 ON (condition1 [AND condition2 ...]) |  
  LEFT|RIGHT|FULL [OUTER] JOIN table2 USING (column1 [, ...]) |  
  CROSS JOIN table2 }  
WHERE ...  
GROUP BY column_name | expr  
HAVING condition  
ORDER BY 기준1 [ ASC | DESC ] [, 기준2 [ASC | DESC], ... ];
```

4.2.4 JOIN - ANSI 표준 구문 ¹⁾JOIN USING

조인 조건으로 사용하는 컬럼 이름이 동일한 경우 사용

```
SELECT EMP_NAME, DEPT_NAME  
FROM   EMPLOYEE  
JOIN   DEPARTMENT USING (DEPT_ID)  
WHERE  JOB_ID = 'J6';
```

- 조인 조건에 사용되는 컬럼은 SELECT 절이나 JOIN 절에서 테이블 구분이 필요 없음
- 조인 조건으로 컬럼 여러 개를 사용하려면 쉼표로 구분
- 테이블 별칭은 사용할 수 없음




EMP_NAME	DEPT_NAME
전우성	해외영업2팀
엄정하	해외영업2팀
권상후	본사 인사팀
임영애	본사 인사팀

4.2.4 JOIN - ANSI 표준 구문 ¹⁾JOIN USING

```
SELECT EMP_NAME, LOC_ID  
FROM   EMPLOYEE2  
JOIN   DEPARTMENT USING (DEPT_ID, LOC_ID);
```

- EMPLOYEE2 테이블은 EMPLOYEE 테이블에 LOC_ID 컬럼이 추가된 구조
- 모두 'A1' 이 입력되어 있음




EMP_NAME	LOC_ID
고승우	A1
권상후	A1
임영애	A1
김술오	A1
이중기	A1
감우섭	A1

4.2.4 JOIN - ANSI 표준 구문 ²⁾JOIN ON

조인 조건으로 사용하는 컬럼 이름이 서로 다른 경우 사용

```
SELECT DEPT_NAME,  
       LOC_DESCRIBE  
FROM   DEPARTMENT  
JOIN   LOCATION ON (LOC_ID = LOCATION_ID);
```



DEPT_NAME	LOC_DESCRIBE
회계팀	아시아지역1
본사 인사팀	아시아지역1
해외영업1팀	미주지역
기술지원팀	기타지역
해외영업2팀	아시아지역2
해외영업3팀	아시아지역3
마케팅팀	아시아지역1

4.2.4 JOIN - OUTER JOIN

조건을 만족시키지 못하는 행까지 Result Set에 포함시키는 조인 유형

EMP_NAME	DEPT_ID
한선기	90
강중훈	90
최만식	90
정도연	60
안석규	60
조재형	60
정지현	50
김예수	50
나승원	50
김순이	50
성해교	50
전우성	80
엄정하	80
심하균	
고승우	10
박하일	50
권상후	10
임영애	10
염정하	
김술오	20
이중기	20
감우섭	20

DEPT_ID	DEPT_NAME
20	회계팀
10	본사 인사팀
50	해외영업1팀
60	기술지원팀
80	해외영업2팀
90	해외영업3팀
30	마케팅팀

EMP_NAME	DEPT_NAME
한선기	해외영업3팀
강중훈	해외영업3팀
최만식	해외영업3팀
정도연	기술지원팀
안석규	기술지원팀
조재형	기술지원팀
정지현	해외영업1팀
김예수	해외영업1팀
나승원	해외영업1팀
김순이	해외영업1팀
성해교	해외영업1팀
전우성	해외영업2팀
엄정하	해외영업2팀
고승우	본사 인사팀
박하일	해외영업1팀
권상후	본사 인사팀
임영애	본사 인사팀
김술오	회계팀
이중기	회계팀
감우섭	회계팀

EMP_NAME	DEPT_NAME
감우섭	회계팀
이중기	회계팀
김술오	회계팀
임영애	본사 인사팀
권상후	본사 인사팀
고승우	본사 인사팀
박하일	해외영업1팀
성해교	해외영업1팀
김순이	해외영업1팀
나승원	해외영업1팀
김예수	해외영업1팀
정지현	해외영업1팀
조재형	기술지원팀
안석규	기술지원팀
정도연	기술지원팀
엄정하	해외영업2팀
전우성	해외영업2팀
최만식	해외영업3팀
강중훈	해외영업3팀
한선기	해외영업3팀
염정하	
심하균	

4.2.4 JOIN - OUTER JOIN ¹⁾오라클 전용 구문

- 연산자 '+' 사용
- 조인 조건을 만족시키는 행이 없는 테이블 기준

```
SELECT EMP_NAME, DEPT_NAME
FROM   EMPLOYEE E, DEPARTMENT D
WHERE  E.DEPT_ID = D.DEPT_ID(+);
```

소속 부서가 없는 직원까지 포함하는 의미

EMP_NAME	DEPT_NAME
감우섭	회계팀
이중기	회계팀
김술오	회계팀
임영애	본사 인사팀
권상후	본사 인사팀
고승우	본사 인사팀
박하일	해외영업1팀
성해교	해외영업1팀
김순이	해외영업1팀
나승원	해외영업1팀
김예수	해외영업1팀
정지현	해외영업1팀
조재형	기술지원팀
안석규	기술지원팀
정도연	기술지원팀
엄정하	해외영업2팀
전우성	해외영업2팀
최만식	해외영업3팀
강중훈	해외영업3팀
한선기	해외영업3팀
염정하	
심하균	

EMP_NAME	DEPT_ID
한선기	90
강중훈	90
최만식	90
정도연	60
안석규	60
조재형	60
정지현	50
김예수	50
나승원	50
김순이	50
성해교	50
전우성	80
엄정하	80
심하균	
고승우	10
박하일	50
권상후	10
임영애	10
염정하	
김술오	20
이중기	20
감우섭	20

DEPT_ID	DEPT_NAME
20	회계팀
10	본사 인사팀
50	해외영업1팀
60	기술지원팀
80	해외영업2팀
90	해외영업3팀
30	마케팅팀

--	--

4.2.4 JOIN - OUTER JOIN ¹⁾오라클 전용 구문

```
SELECT EMP_NAME, DEPT_NAME
FROM   EMPLOYEE E, DEPARTMENT D
WHERE  D.DEPT_ID = E.DEPT_ID(+);
```

소속 직원이 없는 부서까지 포함하는 의미

EMP_NAME	DEPT_NAME
한선기	해외영업3팀
강중훈	해외영업3팀
최만식	해외영업3팀
정도연	기술지원팀
안석규	기술지원팀
조재형	기술지원팀
정지현	해외영업1팀
김예수	해외영업1팀
나승원	해외영업1팀
김순이	해외영업1팀
성해교	해외영업1팀
전우성	해외영업2팀
엄정하	해외영업2팀
고승우	본사 인사팀
박하일	해외영업1팀
권상후	본사 인사팀
임영애	본사 인사팀
김술오	회계팀
이중기	회계팀
감우섭	회계팀
	마케팅팀

EMP_NAME	DEPT_ID
한선기	90
강중훈	90
최만식	90
정도연	60
안석규	60
조재형	60
정지현	50
김예수	50
나승원	50
김순이	50
성해교	50
전우성	80
엄정하	80
심하균	
고승우	10
박하일	50
권상후	10
임영애	10
염정하	
김술오	20
이중기	20
감우섭	20
	30

DEPT_ID	DEPT_NAME
20	회계팀
10	본사 인사팀
50	해외영업1팀
60	기술지원팀
80	해외영업2팀
90	해외영업3팀
30	마케팅팀

4.2.4 JOIN - OUTER JOIN ²⁾ANSI 표준 구문

- 전체 행을 모두 포함시켜야 하는 테이블 기준
- LEFT, RIGHT 키워드 사용
 - LEFT: 기준 테이블이 JOIN 키워드보다 먼저 기술된 경우
 - RIGHT: 기준 테이블이 JOIN 키워드보다 나중에 기술된 경우

```
SELECT EMP_NAME, DEPT_NAME
FROM   EMPLOYEE
LEFT   JOIN DEPARTMENT USING (DEPT_ID)
ORDER BY 1;
```

```
SELECT EMP_NAME, DEPT_NAME
FROM   DEPARTMENT
RIGHT  JOIN EMPLOYEE USING (DEPT_ID)
ORDER BY 1;
```

EMP_NAME	DEPT_NAME
감우섭	회계팀
이중기	회계팀
김술오	회계팀
임영애	본사 인사팀
권상후	본사 인사팀
고승우	본사 인사팀
박하일	해외영업1팀
성해교	해외영업1팀
김순이	해외영업1팀
나승원	해외영업1팀
김예수	해외영업1팀
정지현	해외영업1팀
조재형	기술지원팀
안석규	기술지원팀
정도연	기술지원팀
엄정하	해외영업2팀
전우성	해외영업2팀
최만식	해외영업3팀
강중훈	해외영업3팀
한선기	해외영업3팀
염정하	
심하균	

4.2.4 JOIN - OUTER JOIN ²⁾ANSI 표준 구문

```
SELECT EMP_NAME, DEPT_NAME
FROM   DEPARTMENT
LEFT   JOIN EMPLOYEE USING (DEPT_ID);
```

```
SELECT EMP_NAME, DEPT_NAME
FROM   EMPLOYEE
RIGHT  JOIN DEPARTMENT USING (DEPT_ID);
```

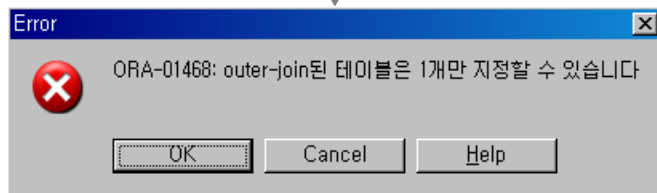
EMP_NAME	DEPT_NAME
한선기	해외영업3팀
강중훈	해외영업3팀
최만식	해외영업3팀
정도연	기술지원팀
안석규	기술지원팀
조재형	기술지원팀
정지현	해외영업1팀
김예수	해외영업1팀
나승원	해외영업1팀
김순이	해외영업1팀
성해교	해외영업1팀
전우성	해외영업2팀
엄정하	해외영업2팀
고승우	본사 인사팀
박하일	해외영업1팀
권상후	본사 인사팀
임영애	본사 인사팀
김술오	회계팀
이중기	회계팀
감우섭	회계팀
	마케팅팀

4.2.4 JOIN - FULL OUTER JOIN

- 양쪽 테이블을 동시에 OUTER JOIN하는 ANSI 표준 구문
- 오라클 전용 구문은 지원되지 않음

```
SELECT EMP_NAME, DEPT_NAME
FROM   EMPLOYEE
FULL   JOIN DEPARTMENT USING (DEPT_ID);
```

```
SELECT EMP_NAME, DEPT_NAME
FROM   EMPLOYEE E, DEPARTMENT D
WHERE  E.DEPT_ID(+) = D.DEPT_ID(+);
```



EMP_NAME	DEPT_NAME
감우섭	회계팀
이중기	회계팀
김솔오	회계팀
임영애	본사 인사팀
권상후	본사 인사팀
고승우	본사 인사팀
박하일	해외영업1팀
성해교	해외영업1팀
김순이	해외영업1팀
나승원	해외영업1팀
김예수	해외영업1팀
정지현	해외영업1팀
조재형	기술지원팀
안석규	기술지원팀
정도연	기술지원팀
엄정하	해외영업2팀
전우성	해외영업2팀
최만식	해외영업3팀
강중훈	해외영업3팀
한선기	해외영업3팀
엄정하	
심하균	
	마케팅팀

4.2.4 JOIN - CROSS JOIN

- 대상 테이블의 모든 행에 대해 가능한 모든 조합을 생성하는 ANSI 표준 구문
- 오라클 구문에서는 조인 조건이 누락된 경우를 의미

결과(일부)

```
SELECT *
FROM   COUNTRY
CROSS JOIN LOCATION;
```

```
SELECT *
FROM   COUNTRY, LOCATION;
```

COUNTRY_ID	COUNTRY_NAME	LOCATION_ID	COUNTRY_ID	LOC_DESCRIBE
KO	한국	A1	KO	아시아지역1
JP	일본	A1	KO	아시아지역1
CH	중국	A1	KO	아시아지역1
US	미국	A1	KO	아시아지역1
ID	인도	A1	KO	아시아지역1
KO	한국	A2	JP	아시아지역2
JP	일본	A2	JP	아시아지역2
CH	중국	A2	JP	아시아지역2

[COUNTRY]

COUNTRY_ID	COUNTRY_NAME
KO	한국
JP	일본
CH	중국
US	미국
ID	인도

[LOCATION]

LOCATION_ID	COUNTRY_ID	LOC_DESCRIBE
A1	KO	아시아지역1
A2	JP	아시아지역2
A3	CH	아시아지역3
U1	US	미주지역
OT	ID	기타지역

COUNTRY 테이블의 5건과 LOCATION 테이블의 5건에 대한 모든 조합 25건이 생성

4.2.4 JOIN - Non Equijoin

```

SELECT EMP_NAME, SALARY, SLEVEL
FROM   EMPLOYEE
JOIN   SAL_GRADE ON (SALARY BETWEEN LOWEST AND HIGHEST)
ORDER BY 3;

```

[SAL_GRADE]

SLEVEL	LOWEST	HIGHEST
A	3000000	9000000
B	2500000	2999999
C	2000000	2499999
D	1500000	1999999
E	1000000	1499999

컬럼 값이 같은 경우가 아닌
범위에 속하는지의 여부를
확인하는 의미


결과(일부)

EMP_NAME	SALARY	SLEVEL
강중훈	5500000	A
조재형	3800000	A
최만식	3600000	A
안석규	3500000	A
권상후	3410000	A
김순이	3400000	A
한선기	9000000	A
감우섭	2500000	B
임영애	2640000	B
박하일	2600000	B

4.2.4 JOIN - SELF JOIN

- 한 테이블을 두 번 조인하는 유형
- 테이블 별칭을 사용해야 함

```
SELECT E.EMP_NAME AS 직원,
       M.EMP_NAME AS 관리자
FROM   EMPLOYEE E
JOIN   EMPLOYEE M ON (E.MGR_ID = M.EMP_ID)
ORDER BY 1;
```



직원	관리자
강중훈	한선기
고승우	한선기
권상후	고승우
김순이	김예수
김예수	한선기
나승원	김예수
성해교	김예수
안석규	한선기
엄정하	전우성
임영애	고승우
전우성	한선기
정도연	안석규
정지현	김예수
조재형	안석규
최만식	강중훈

4.2.4 JOIN - N개 테이블 JOIN

N개의 테이블을 조인하려면 최소 N-1개의 조인 조건(또는 N-1개의 JOIN 키워드)이 필요

[EMPLOYEE]			[JOB]		[EMPLOYEE]			[DEPARTMENT]	
EMP_NAME	JOB_TITLE	DEPT_NAME	JOB_ID	JOB_TITLE	EMP_NAME	JOB_ID	DEPT_ID	DEPT_ID	DEPT_NAME
한선기	대표이사	해외영업3팀	J1	대표이사	한선기	J1	90	20	회계팀
최만식	부사장	해외영업3팀	J2	부사장	강중훈	J2	90	10	본사 인사팀
강중훈	부사장	해외영업3팀	J3	부장	최만식	J2	90	50	해외영업1팀
김순이	부장	해외영업1팀	J4	차장	정도연	J4	60	60	기술지원팀
조재형	부장	기술지원팀	J5	과장	안석규	J3	60	80	해외영업2팀
안석규	부장	기술지원팀	J6	대리	조재형	J3	60	90	해외영업3팀
감우섭	차장	회계팀	J7	사원	정지현	J7	50	30	마케팅팀
이중기	차장	회계팀			김예수	J5	50		
김술오	차장	회계팀			나승원	J5	50		
정도연	차장	기술지원팀			김순이	J3	50		
박하일	과장	해외영업1팀			성해교	J7	50		
나승원	과장	해외영업1팀			전우성	J6	80		
김예수	과장	해외영업1팀			엄정하	J6	80		
임영애	대리	본사 인사팀			심하균				
권상후	대리	본사 인사팀			고승우	J7	10		
엄정하	대리	해외영업2팀			박하일	J5	50		
전우성	대리	해외영업2팀			권상후	J6	10		
고승우	사원	본사 인사팀			임영애	J6	10		
성해교	사원	해외영업1팀			염정하	J7			
정지현	사원	해외영업1팀			김술오	J4	20		
					이중기	J4	20		
					감우섭	J4	20		

4.2.4 JOIN - N개 테이블 JOIN

```

SELECT EMP_NAME,
       JOB_TITLE,
       DEPT_NAME
FROM   EMPLOYEE
JOIN   JOB USING (JOB_ID)
JOIN   DEPARTMENT USING (DEPT_ID);

```

- EMPLOYEE와 JOB이 먼저 조인되어 새로운 테이블처럼 취급
- 그 결과와 DEPARTMENT가 조인되는 개념

EMP_NAME	JOB_TITLE	DEPT_ID	DEPT_ID	DEPT_NAME
한선기	대표이사	90	20	회계팀
강중훈	부사장	90	10	본사 인사팀
최만식	부사장	90	50	해외영업1팀
정도연	차장	60	60	기술지원팀
안석규	부장	60	80	해외영업2팀
조재형	부장	60	90	해외영업3팀
정지현	사원	50	30	마케팅팀
김예수	과장	50		
나승원	과장	50		
김순이	부장	50		
성해교	사원	50		
전우성	대리	80		
엄정하	대리	80		
고승우	사원	10		
박하일	과장	50		
권상후	대리	10		
임영애	대리	10		
염정하	사원			
김술오	차장	20		
이중기	차장	20		
감우섭	차장	20		

4.2.4 JOIN - N개 테이블 JOIN

```

SELECT EMP_NAME,
       JOB_TITLE,
       DEPT_NAME
FROM   EMPLOYEE
LEFT JOIN JOB USING (JOB_ID)
LEFT JOIN DEPARTMENT USING (DEPT_ID);

```

EMPLOYEE 테이블의 JOB_ID 컬럼과 DEPT_ID 컬럼이 조인 조건으로 사용되고 있기 때문에 해당 정보가 없는 직원을 결과에 포함하려면 OUTER JOIN을 사용해야 함

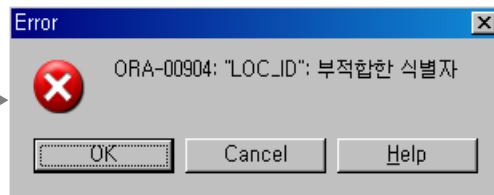
EMP_NAME	JOB_TITLE	DEPT_NAME
김술오	차장	회계팀
이중기	차장	회계팀
감우섭	차장	회계팀
고승우	사원	본사 인사팀
권상후	대리	본사 인사팀
임영애	대리	본사 인사팀
정지현	사원	해외영업1팀
성해교	사원	해외영업1팀
김예수	과장	해외영업1팀
나승원	과장	해외영업1팀
박하일	과장	해외영업1팀
김순이	부장	해외영업1팀
정도연	차장	기술지원팀
안석규	부장	기술지원팀
조재형	부장	기술지원팀
전우성	대리	해외영업2팀
엄정하	대리	해외영업2팀
강중훈	부사장	해외영업3팀
최만식	부사장	해외영업3팀
한선기	대표이사	해외영업3팀
심하균		
엄정하	사원	

4.2.4 JOIN - N개 테이블 JOIN

```

SELECT EMP_NAME,
       LOC_DESCRIBE,
       DEPT_NAME
FROM   EMPLOYEE
JOIN   LOCATION ON (LOCATION_ID = LOC_ID)
JOIN   DEPARTMENT USING (DEPT_ID);

```



- 구문 순서대로 EMPLOYEE와 LOCATION이 먼저 조인 되어야 함
- 두 테이블은 직접적인 연관 관계가 없으므로 오류 발생

```

SELECT EMP_NAME,
       LOC_DESCRIBE,
       DEPT_NAME
FROM   EMPLOYEE
JOIN   DEPARTMENT USING (DEPT_ID)
JOIN   LOCATION ON (LOCATION_ID = LOC_ID);

```

결과(일부)

EMP_NAME	LOC_DESCRIBE	DEPT_NAME
한선기	아시아지역3	해외영업3팀
강중훈	아시아지역3	해외영업3팀
최만식	아시아지역3	해외영업3팀
정도연	기타지역	기술지원팀
안석규	기타지역	기술지원팀
조재형	기타지역	기술지원팀

DEPARTMENT 테이블이 EMPLOYEE와 LOCATION을 연결시킬 수 있도록 구문을 수정해야 함

4.2.4 JOIN - N개 테이블 JOIN

```
SELECT EMP_NAME,  
       DEPT_NAME  
FROM   EMPLOYEE  
JOIN   JOB USING (JOB_ID)  
JOIN   DEPARTMENT USING (DEPT_ID)  
JOIN   LOCATION ON (LOC_ID = LOCATION_ID)  
WHERE  JOB_TITLE='대리'  
AND    LOC_DESCRIBE LIKE '아시아%';
```

EMP_NAME	DEPT_NAME
전우성	해외영업2팀
엄정하	해외영업2팀
권상후	본사 인사팀
임영애	본사 인사팀

- 컬럼 조회 목적이 아닌 WHERE 조건을 위해 조인에 포함되어야 하는 테이블이 필요
- JOB 테이블은 직급 조건을 위해, LOCATION 테이블은 지역 조건을 위해 조인되었음

4.2.5 SET Operator - 개념

- 두 개 이상의 쿼리 결과를 하나로 결합시키는 연산자
- SELECT 절에 기술하는 컬럼 개수와 데이터 타입은 모든 쿼리에서 동일해야 함

유형	설명	도식
UNION	양쪽 쿼리 결과를 모두 포함 (중복 결과는 1번만 표현)	
UNION ALL	양쪽 쿼리 결과를 모두 포함(중복 결과도 모두 표현)	
INTERSECT	양쪽 쿼리 결과에 모두 포함되는 행만 표현	
MINUS	쿼리1 결과에만 포함되고 쿼리2 결과에는 포함되지 않는 행만 표현	

4.2.5 SET Operator - UNION

```
SELECT EMP_ID,  
       ROLE_NAME  
FROM   EMPLOYEE_ROLE  
  
UNION  
  
SELECT EMP_ID,  
       ROLE_NAME  
FROM   ROLE_HISTORY;
```

쿼리1

쿼리2

모든 직원의 현재 ROLE과 이전
ROLE을 함께 표시

[쿼리1 결과]

EMP_ID	ROLE_NAME
100	SALES
101	SALES
102	SALES
103	SE
104	SE
107	SE
124	MKT
141	MKT
143	MKT
144	MKT
149	MKT
174	SALES-MKT
176	SALES-MKT
178	
200	HR
201	MKT
202	HR
205	HR
206	
207	FIN
208	FIN
210	FIN

22건

[쿼리2 결과]

EMP_ID	ROLE_NAME
104	SE-PKG
176	MKT
104	SE-ANLY
104	SE

4건

[UNION 결과]

EMP_ID	ROLE_NAME
100	SALES
101	SALES
102	SALES
103	SE
104	SE
104	SE-ANLY
104	SE-PKG
107	SE
124	MKT
141	MKT
143	MKT
144	MKT
149	MKT
174	SALES-MKT
176	MKT
176	SALES-MKT
178	
200	HR
201	MKT
202	HR
205	HR
206	
207	FIN
208	FIN
210	FIN

25건
중복 1건 제외

4.2.5 SET Operator - UNION ALL

```
SELECT EMP_ID,  
       ROLE_NAME,  
       DEPT_ID  
FROM   EMPLOYEE_ROLE  
  
UNION ALL  
  
SELECT EMP_ID,  
       ROLE_NAME,  
       DEPT_ID  
FROM   ROLE_HISTORY;
```

쿼리1

쿼리2

모든 직원의 현재 ROLE과 이전
ROLE을 중복 값 포함 표시

[쿼리1 결과]

EMP_ID	ROLE_NAME	DEPT_ID
100	SALES	90
101	SALES	90
102	SALES	90
103	SE	60
104	SE	60
107	SE	60
124	MKT	50
141	MKT	50
143	MKT	50
144	MKT	50
149	MKT	50
174	SALES-MKT	80
176	SALES-MKT	80
178		
200	HR	10
201	MKT	50
202	HR	10
205	HR	10
206		
207	FIN	20
208	FIN	20
210	FIN	20

[쿼리2 결과]

EMP_ID	ROLE_NAME	DEPT_ID
104	SE-PKG	90
176	MKT	50
104	SE-ANLY	90
104	SE	60

4건

[UNION ALL 결과]

EMP_ID	ROLE_NAME	DEPT_ID
100	SALES	90
101	SALES	90
102	SALES	90
103	SE	60
104	SE	60
107	SE	60
124	MKT	50
141	MKT	50
143	MKT	50
144	MKT	50
149	MKT	50
174	SALES-MKT	80
176	SALES-MKT	80
178		
200	HR	10
201	MKT	50
202	HR	10
205	HR	10
206		
207	FIN	20
208	FIN	20
210	FIN	20
104	SE-PKG	90
176	MKT	50
104	SE-ANLY	90
104	SE	60

26건
중복 포함

22건

4.2.5 SET Operator - INTERSECT

```
SELECT EMP_ID,  
       ROLE_NAME  
FROM   EMPLOYEE_ROLE  
INTERSECT  
SELECT EMP_ID,  
       ROLE_NAME  
FROM   ROLE_HISTORY;
```

쿼리1

쿼리2

입사 후 현재 ROLE과 동일한
ROLE을 가졌던 적이 있는 직원
조회

[쿼리1 결과]

EMP_ID	ROLE_NAME
100	SALES
101	SALES
102	SALES
103	SE
104	SE
107	SE
124	MKT
141	MKT
143	MKT
144	MKT
149	MKT
174	SALES-MKT
176	SALES-MKT
178	
200	HR
201	MKT
202	HR
205	HR
206	
207	FIN
208	FIN
210	FIN

[쿼리2 결과]

EMP_ID	ROLE_NAME
104	SE-PKG
176	MKT
104	SE-ANLY
104	SE

[INTERSECT 결과]

EMP_ID	ROLE_NAME
104	SE

4.2.5 SET Operator - MINUS

```
SELECT EMP_ID,  
       ROLE_NAME  
FROM   EMPLOYEE_ROLE  
MINUS  
SELECT EMP_ID,  
       ROLE_NAME  
FROM   ROLE_HISTORY;
```

쿼리1

쿼리2

입사 후 현재 ROLE과 동일한
ROLE을 부여 받은 적이 없는
직원 조회

[쿼리1 결과]

EMP_ID	ROLE_NAME
100	SALES
101	SALES
102	SALES
103	SE
104	SE
107	SE
124	MKT
141	MKT
143	MKT
144	MKT
149	MKT
174	SALES-MKT
176	SALES-MKT
178	
200	HR
201	MKT
202	HR
205	HR
206	
207	FIN
208	FIN
210	FIN

[쿼리2 결과]

EMP_ID	ROLE_NAME
104	SE-PKG
176	MKT
104	SE-ANLY
104	SE

[MINUS 결과]

EMP_ID	ROLE_NAME
100	SALES
101	SALES
102	SALES
103	SE
107	SE
124	MKT
141	MKT
143	MKT
144	MKT
149	MKT
174	SALES-MKT
176	SALES-MKT
178	
200	HR
201	MKT
202	HR
205	HR
206	
207	FIN
208	FIN
210	FIN

4.2.5 SET Operator - 활용

쿼리1과 쿼리2의 SELECT 목록은 반드시 동일(컬럼 개수, 데이터 타입)해야 하므로 이를 위해 Dummy Column을 사용할 수 있다

```

SELECT EMP_NAME,
       JOB_ID,
       HIRE_DATE
FROM   EMPLOYEE
WHERE  DEPT_ID = '20'

UNION

SELECT DEPT_NAME,
       DEPT_ID,
       NULL
FROM   DEPARTMENT
WHERE  DEPT_ID = '20';
  
```

쿼리1

EMP_NAME	JOB_ID	HIRE_DATE
김술오	J4	96/10/01
이중기	J4	04/10/01
감우섭	J4	05/07/31

쿼리2

DEPT_NAME	DEPT_ID
회계팀	20

DEPT_NAME	DEPT_ID	NULL
회계팀	20	

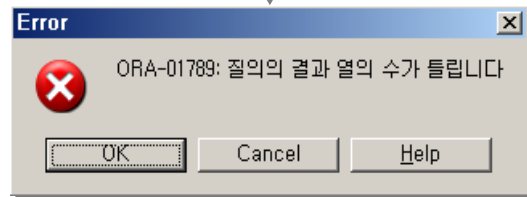
[결과]

EMP_NAME	JOB_ID	HIRE_DATE
감우섭	J4	05/07/31
김술오	J4	96/10/01
이중기	J4	04/10/01
회계팀	20	

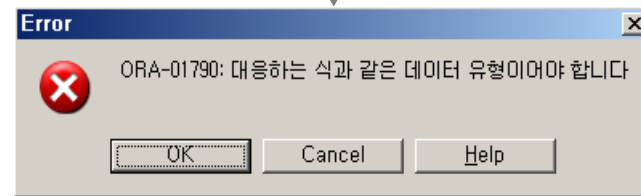
- 쿼리1과 쿼리2는 결과 표시 구조가 동일해야 함
- 구조를 맞추기 위해 해당 위치에 dummy column 사용

4.2.5 SET Operator - 활용

```
SELECT EMP_NAME,  
       JOB_ID,  
       HIRE_DATE  
FROM   EMPLOYEE  
WHERE  DEPT_ID = '20'  
UNION  
SELECT DEPT_NAME,  
       DEPT_ID  
FROM   DEPARTMENT  
WHERE  DEPT_ID = '20';
```



```
SELECT EMP_NAME,  
       SALARY  
FROM   EMPLOYEE  
WHERE  DEPT_ID = '20'  
UNION  
SELECT DEPT_NAME,  
       DEPT_ID  
FROM   DEPARTMENT  
WHERE  DEPT_ID = '20';
```



4.2.5 SET Operator - 활용

실습 : UNION 구문을 이용하여 '50' 부서원을 관리자와 직원을 구분하여 표시하시오

```

SELECT EMP_ID,
       EMP_NAME,
       '관리자' AS 구분
FROM   EMPLOYEE
WHERE  EMP_ID = '141'
AND    DEPT_ID = '50'

UNION

SELECT EMP_ID,
       EMP_NAME,
       '직원' AS 구분
FROM   EMPLOYEE
WHERE  DEPT_ID = '50'
AND    EMP_ID != '141';

```

쿼리1

쿼리2

관리자 정보

EMP_ID	EMP_NAME	구분
141	김예수	관리자

직원 정보

EMP_ID	EMP_NAME	구분
124	정지현	직원
143	나승원	직원
144	김순이	직원
149	성해교	직원
201	박하일	직원

EMP_ID	EMP_NAME	구분
124	정지현	직원
141	김예수	관리자
143	나승원	직원
144	김순이	직원
149	성해교	직원
201	박하일	직원

관리자와 직원 구분을 위해
임의 컬럼을 추가하고
리터럴을 함께 사용

컬럼 별칭은 첫 번째 쿼리에서 지정한 경우에만 적용

4.2.5 SET Operator - 활용

- 기본적으로 첫째 컬럼의 오름차순으로 정렬(UNION ALL은 제외)
- ORDER BY 키워드는 마지막 쿼리 부분에서 한 번만 기술
- 각 쿼리 SELECT 절의 컬럼/별칭 이름이 다르면 ORDER BY 절에 컬럼 기술 순서만 사용

```
SELECT EMP_ID,  
       EMP_NAME,  
       '관리자' AS 구분  
FROM   EMPLOYEE  
WHERE  EMP_ID = '141'  
AND    DEPT_ID = '50'  
UNION  
SELECT EMP_ID,  
       EMP_NAME,  
       '직원' AS 구분  
FROM   EMPLOYEE  
WHERE  DEPT_ID = '50'  
AND    EMP_ID != '141'  
ORDER BY 3, 1;
```



EMP_ID	EMP_NAME	구분
141	김예수	관리자
124	정지현	직원
143	나승원	직원
144	김순이	직원
149	성해교	직원
201	박하일	직원

4.2.5 SET Operator - 활용

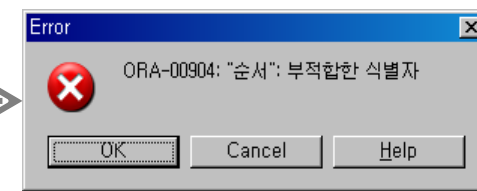
```
SELECT 'SQL을 공부하고 있습니다' 문장, 3 순서 FROM DUAL
UNION
SELECT '우리는 지금', 1 FROM DUAL
UNION
SELECT '아주 재미있게', 2 FROM DUAL;
```

문장	순서
SQL을 공부하고 있습니다	3
아주 재미있게	2
우리는 지금	1

```
SELECT 'SQL을 공부하고 있습니다' 문장, 3 순서 FROM DUAL
UNION
SELECT '우리는 지금', 1 FROM DUAL
UNION
SELECT '아주 재미있게', 2 FROM DUAL
ORDER BY 2;
```

문장	순서
우리는 지금	1
아주 재미있게	2
SQL을 공부하고 있습니다	3

```
SELECT 'SQL을 공부하고 있습니다' 문장, 3 순서 FROM DUAL
UNION
SELECT '우리는 지금', 1 FROM DUAL
UNION
SELECT '아주 재미있게', 2 FROM DUAL
ORDER BY 순서;
```




4.2.5 SET Operator - 활용

SET Operator와 JOIN 관계

```
SELECT EMP_ID,  
       ROLE_NAME  
FROM   EMPLOYEE_ROLE  
JOIN   ROLE_HISTORY USING (EMP_ID, ROLE_NAME);
```

```
SELECT EMP_ID,  
       ROLE_NAME  
FROM   EMPLOYEE_ROLE  
INTERSECT  
SELECT EMP_ID,  
       ROLE_NAME  
FROM   ROLE_HISTORY;
```




EMP_ID	ROLE_NAME
104	SE

4.2.5 SET Operator - 활용

SET Operator와 IN Operator 관계

```
SELECT EMP_NAME,  
       JOB_TITLE 직급  
FROM   EMPLOYEE  
JOIN   JOB USING (JOB_ID)  
WHERE  JOB_TITLE IN ('대리', '사원')  
ORDER BY 2,1;
```

```
SELECT EMP_NAME, '사원' 직급  
FROM   EMPLOYEE  
JOIN   JOB USING (JOB_ID)  
WHERE  JOB_TITLE = '사원'  
UNION  
SELECT EMP_NAME, '대리'  
FROM   EMPLOYEE  
JOIN   JOB USING (JOB_ID)  
WHERE  JOB_TITLE = '대리'  
ORDER BY 2,1;
```



EMP_NAME	직급
권상후	대리
엄정하	대리
임영애	대리
전우성	대리
고승우	사원
성해교	사원
엄정하	사원
정지현	사원

4.2.6 Subquery - 개념

- 하나의 쿼리가 다른 쿼리에 포함되는 구조
- 다른 쿼리에 포함된 내부 쿼리(서브 쿼리)는 외부 쿼리(메인 쿼리)에 사용될 값을 반환하는 역할

요구사항 : '나승원' 직원과 같은 부서원들을 조회하라

요구사항을 해결하기 위해서는

- ❶ '나승원' 직원이 속한 부서가 어떤 부서인지(부서 번호 또는 부서 이름) 찾고
- ❷ 부서 정보(부서 번호 또는 부서 이름)를 이용하여 해당 부서에 속한 직원들을 찾아야 함

메인쿼리

```
SELECT EMP_NAME  
FROM   EMPLOYEE  
WHERE  DEPT_ID = '나승원의 소속부서 ID' ;
```

서브쿼리

```
SELECT DEPT_ID  
FROM   EMPLOYEE  
WHERE  EMP_NAME = '나승원' ;
```


4.2.6 Subquery - 구문

```
SELECT ...  
FROM ...  
WHERE expr operator ( SELECT ...  
                        FROM ...  
                        WHERE ... ) ;
```

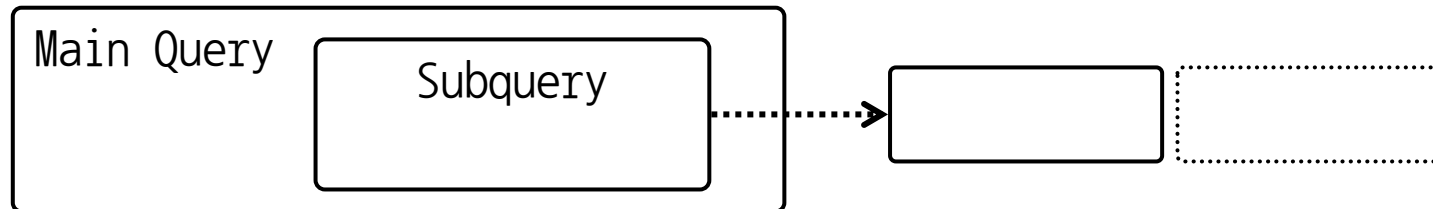
[구문 설명]

- 서브쿼리는 일반적인 SQL 구문과 동일(별도 형식이 존재하는 것이 아님)
- SELECT, FROM, WHERE, HAVING 절 등에서 사용 가능
- 서브쿼리는 ()로 묶어서 표현
- 서브쿼리에는 ; 를 사용하지 않음
- 유형에 따라 연산자를 구분해서 사용

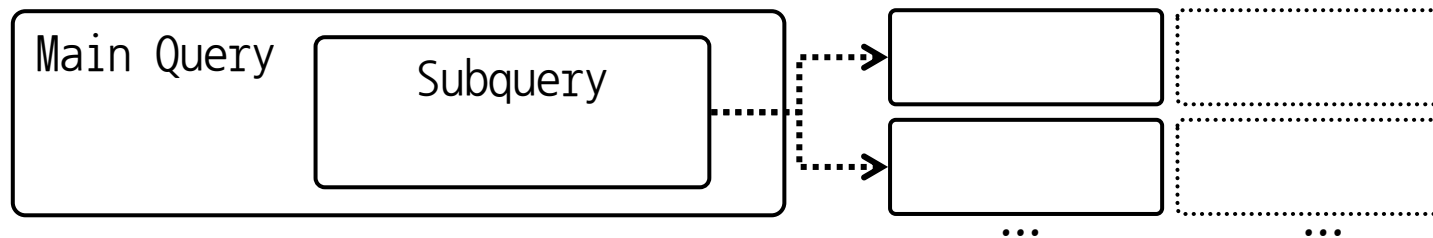
4.2.6 Subquery - 유형

- 단일 행 서브쿼리
 - 단일 행 반환
 - 단일 행 비교 연산자(=, >, >=, <, <=, <> 등) 사용
- 다중 행 서브쿼리
 - 여러 행 반환
 - 다중 행 비교 연산자(IN, ANY, ALL 등) 사용

단일 행 서브쿼리 도식

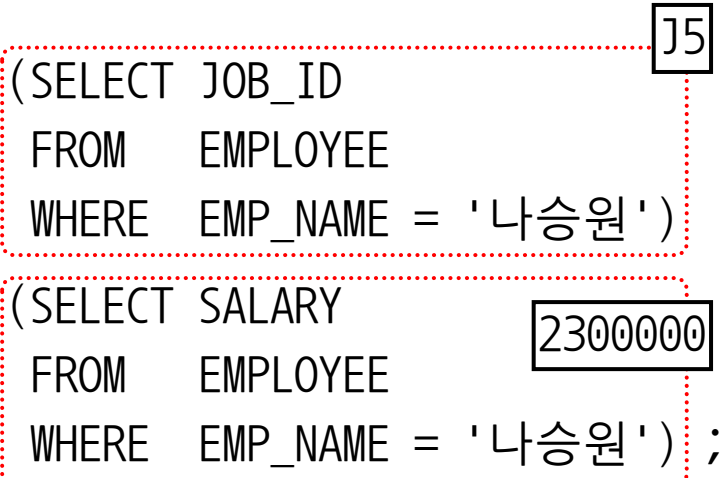


다중 행 서브쿼리 도식




4.2.6 Subquery - 단일 행 서브쿼리

```
SELECT EMP_NAME,  
       JOB_ID,  
       SALARY  
FROM   EMPLOYEE  
WHERE  JOB_ID = (SELECT JOB_ID  
                 FROM   EMPLOYEE  
                 WHERE  EMP_NAME = '나승원')  
  
AND    SALARY > (SELECT SALARY  
                 FROM   EMPLOYEE  
                 WHERE  EMP_NAME = '나승원');
```



```
SELECT EMP_NAME,  
       JOB_ID,  
       SALARY  
FROM   EMPLOYEE  
WHERE  JOB_ID = 'J5'  
AND    SALARY > 23000000;
```



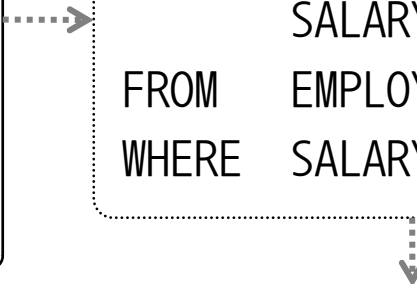
EMP_NAME	JOB_ID	SALARY
박하일	J5	2600000

4.2.6 Subquery - 단일 행 서브쿼리

```
SELECT EMP_NAME,  
       JOB_ID,  
       SALARY  
FROM   EMPLOYEE  
WHERE  SALARY = (SELECT MIN(SALARY)  
                FROM   EMPLOYEE);
```

1500000

```
SELECT EMP_NAME,  
       JOB_ID,  
       SALARY  
FROM   EMPLOYEE  
WHERE  SALARY = 1500000;
```



EMP_NAME	JOB_ID	SALARY
정지현	J7	1500000
고승우	J7	1500000
염정하	J7	1500000

4.2.6 Subquery - 단일 행 서브쿼리

```
SELECT  DEPT_NAME,  
        SUM(SALARY)  
FROM    EMPLOYEE  
LEFT JOIN DEPARTMENT USING (DEPT_ID)  
GROUP BY DEPT_ID, DEPT_NAME  
HAVING  SUM(SALARY) = (SELECT  MAX(SUM(SALARY))  
                       FROM    EMPLOYEE  
                       GROUP BY DEPT_ID);
```

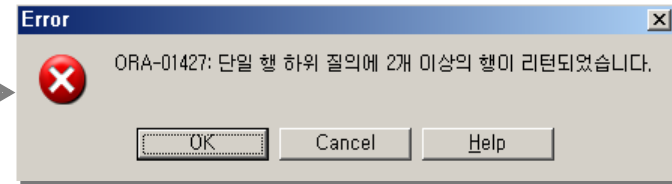
181000000

```
SELECT  DEPT_NAME,  
        SUM(SALARY)  
FROM    EMPLOYEE  
LEFT JOIN DEPARTMENT USING (DEPT_ID)  
GROUP BY DEPT_ID, DEPT_NAME  
HAVING  SUM(SALARY) = 181000000;
```

DEPT_NAME	SUM(SALARY)
해외영업3팀	181000000

4.2.6 Subquery - 단일 행 서브쿼리

```
SELECT  EMP_ID,  
        EMP_NAME  
FROM    EMPLOYEE  
WHERE   SALARY = (SELECT  MIN(SALARY)  
                  FROM    EMPLOYEE  
                  GROUP BY DEPT_ID);
```



MIN(SALARY)
1500000
1500000
2500000
1500000
3600000
2090000
2600000

Subquery 실행 결과 여러 행이 생성되었음
→ 비교 대상 값이 여러 개이므로 오류 발생

4.2.6 Subquery - 다중 행 서브쿼리 ¹⁾IN, NOT IN 연산자

```

SELECT EMP_ID,
       EMP_NAME,
       '관리자' AS 구분
FROM   EMPLOYEE
WHERE  EMP_ID IN (SELECT MGR_ID FROM EMPLOYEE)
UNION
SELECT EMP_ID,
       EMP_NAME,
       '직원'
FROM   EMPLOYEE
WHERE  EMP_ID NOT IN (SELECT MGR_ID FROM EMPLOYEE
                      WHERE MGR_ID IS NOT NULL)

ORDER BY 3, 1;

```

EMP_ID	EMP_NAME	구분
100	한선기	관리자
101	강중훈	관리자
104	안석규	관리자
141	김예수	관리자
174	전우성	관리자
200	고승우	관리자
102	최만식	직원
103	정도연	직원
107	조재형	직원
124	정지현	직원
143	나승원	직원
144	김순이	직원
149	성해교	직원
176	엄정하	직원
178	심하균	직원
201	박하일	직원
202	권상후	직원
205	임영애	직원
206	염정하	직원
207	김술오	직원
208	이중기	직원
210	감우섭	직원

NOT IN 연산자와 다중 행 서브쿼리를 함께 사용하는 경우, 서브쿼리 결과에 NULL이 포함되면 전체 결과가 NULL이 됨

→ 서브쿼리 결과에서 NULL인 경우를 제외시켜야 함

4.2.6 Subquery - 다중 행 서브쿼리 ¹⁾IN, NOT IN 연산자

전체 직원에 대해 관리자와 직원을 구분하여 표시하시오(UNION 구문을 사용하지 말 것)

```
SELECT EMP_ID , EMP_NAME,
       CASE
         WHEN EMP_ID IN (SELECT MGR_ID FROM EMPLOYEE) THEN '관리자'
         ELSE '직원' END AS 구분
FROM   EMPLOYEE
ORDER BY 3, 1;
```

EMP_ID	EMP_NAME	구분
100	한선기	관리자
101	강중훈	관리자
104	안석규	관리자
141	김예수	관리자
174	전우성	관리자
200	고승우	관리자
102	최만식	직원
103	정도연	직원
107	조재형	직원
124	정지현	직원
143	나승원	직원
144	김순이	직원
149	성해교	직원
176	엄정하	직원
178	심하균	직원
201	박하일	직원
202	권상후	직원
205	임영애	직원
206	염정하	직원
207	김술오	직원
208	이중기	직원
210	감우섭	직원

4.2.6 Subquery - 다중 행 서브쿼리 2) ANY 연산자

- < ANY : 비교 대상 중 최대 값 보다 작음
- > ANY : 비교 대상 중 최소 값 보다 큼
- = ANY : IN 연산자와 동일

```
SELECT EMP_NAME,
       SALARY
FROM   EMPLOYEE
JOIN   JOB USING (JOB_ID)
WHERE  JOB_TITLE = '대리'
AND    SALARY > ANY
      (SELECT SALARY
       FROM   EMPLOYEE
       JOIN   JOB USING (JOB_ID)
       WHERE  JOB_TITLE = '과장');
```

EMP_NAME	SALARY
엄정하	2420000
권상후	3410000
임영애	2640000

과장 급여 현황

EMP_NAME	SALARY
김예수	2100000
나승원	2300000
박하일	2600000

대리 급여 현황

EMP_NAME	SALARY
전우성	2090000
엄정하	2420000
권상후	3410000
임영애	2640000

과장 직급보다 급여가 많은 대리 직급 직원 조회

4.2.6 Subquery - 다중 행 서브쿼리 ³⁾ALL 연산자

- < ALL : 비교 대상 중 최소 값 보다 작음
- > ALL : 비교 대상 중 최대 값 보다 큼

```

SELECT EMP_NAME,
       SALARY
FROM   EMPLOYEE
JOIN   JOB USING (JOB_ID)
WHERE  JOB_TITLE = '대리'
AND    SALARY > ALL
        (SELECT SALARY
         FROM   EMPLOYEE
         JOIN   JOB USING (JOB_ID)
         WHERE  JOB_TITLE = '과장');

```

EMP_NAME	SALARY
권상후	3410000
임영애	2640000

과장 급여 현황

EMP_NAME	SALARY
김예수	2100000
나승원	2300000
박하일	2600000

대리 급여 현황

EMP_NAME	SALARY
전우성	2090000
엄정하	2420000
권상후	3410000
임영애	2640000

모든 과장들의 직급보다 급여가 많은 대리 직급 직원 조회

4.2.6 Subquery - 다중 행 서브쿼리

자기 직급의 평균 급여를 받는 직원을 조회하시오

```
SELECT EMP_NAME,
       JOB_TITLE,
       SALARY
FROM   EMPLOYEE
LEFT JOIN JOB USING (JOB_ID)
WHERE  SALARY IN
       (SELECT TRUNC(AVG(SALARY), -5)
        FROM   EMPLOYEE
        GROUP BY JOB_ID)
ORDER BY JOB_ID;
```

계산 편의를 위해 TRUNC를 사용하였음

EMP_NAME	JOB_TITLE	SALARY
한선기	대표이사	9000000
안석규	부장	3500000
정도연	차장	2600000
이중기	차장	2500000
감우섭	차장	2500000
김술오	차장	2500000
박하일	과장	2600000
나승원	과장	2300000
심하균		2300000

직급 별 평균 급여 현황

JOB_TITLE	AVG
과장	2300000
사원	1600000
차장	2500000
대리	2600000
부장	3500000
부사장	4500000
대표이사	9000000

단순히 급여만 비교했기 때문에 다른 직급의 평균
급여와 동일한 경우까지 결과에 포함되었음

→ 직급과 급여를 동시에 비교해야 함

4.2.6 Subquery - 다중 행/다중 열 서브쿼리

자기 직급의 평균 급여를 받는 직원을 조회하시오

```
SELECT EMP_NAME,
       JOB_TITLE,
       SALARY
FROM   EMPLOYEE
LEFT   JOIN JOB USING (JOB_ID)
WHERE  (NVL(JOB_ID, ' '), SALARY) IN
       (SELECT NVL(JOB_ID, ' '),
              TRUNC(AVG(SALARY), -5)
        FROM   EMPLOYEE
        GROUP BY JOB_ID)
ORDER BY JOB_ID;
```

EMP_NAME	JOB_TITLE	SALARY
한선기	대표이사	9000000
안석규	부장	3500000
김술오	차장	2500000
감우섭	차장	2500000
이중기	차장	2500000
나승원	과장	2300000
심하균		2300000

직급 별 평균 급여 현황

JOB_TITLE	AVG
과장	2300000
	2300000
사원	1600000
차장	2500000
대리	2600000
부장	3500000
부사장	4500000
대표이사	9000000

4.2.6 Subquery - FROM 절 사용

```

SELECT EMP_NAME,
       JOB_TITLE,
       SALARY
FROM   (SELECT JOB_ID,
               TRUNC(AVG(SALARY), -5) AS JOBAVG
        FROM   EMPLOYEE
        GROUP BY JOB_ID) V
JOIN   EMPLOYEE E ON
       (JOBAVG = SALARY AND
        NVL(E.JOB_ID, ' ') = NVL(V.JOB_ID, ' '))
LEFT JOIN JOB J ON (E.JOB_ID = J.JOB_ID)
ORDER BY JOB_ID;

```

EMP_NAME	JOB_TITLE	SALARY
한선기	대표이사	9000000
안석규	부장	3500000
김술오	차장	2500000
감우섭	차장	2500000
이중기	차장	2500000
나승원	과장	2300000
심하균		2300000

JOB_ID	JOBAVG
J2	4500000
J7	1600000
	2300000
J3	3500000
J6	2600000
J5	2300000
J1	9000000
J4	2500000

서브쿼리 결과를
테이블처럼 간주

4.2.6 Subquery - Correlated Subquery

메인 쿼리에서 고려된 각 후보 행들에 대해 서브쿼리가 다른 결과를 반환해야 하는 경우(메인 쿼리에서 처리되는 각 행의 값에 따라 응답이 달라져야 하는 경우)에 유용

```
SELECT EMP_NAME,
       JOB_TITLE,
       SALARY
FROM   EMPLOYEE E
LEFT JOIN JOB J  ON (E.JOB_ID = J.JOB_ID)
WHERE  SALARY = (SELECT TRUNC(AVG(SALARY), -5)
                 FROM    EMPLOYEE
                 WHERE   NVL(JOB_ID, ' ') =
                        NVL(E.JOB_ID, ' '))
ORDER BY E.JOB_ID;
```

EMP_NAME	JOB_TITLE	SALARY
한선기	대표이사	9000000
안석규	부장	3500000
김술오	차장	2500000
이중기	차장	2500000
감우섭	차장	2500000
나승원	과장	2300000
심하균		2300000

- 서브쿼리의 WHERE 절 수행을 위해서는 메인쿼리가 먼저 수행되는 구조
- 메인쿼리 행 수 만큼 반복적으로 수행됨

4.2.6 Subquery - EXISTS, NOT EXISTS 연산자

- 결과에 포함되는지의 여부를 확인해야 하는 경우에 유용한 연산자
- 존재 여부에 따라 TRUE 값을 반환

메인쿼리 한 행을
읽고 해당 값을
서브쿼리에서
참조하여 서브쿼리
결과에 존재하면
TRUE 반환

메인쿼리 한 행을
읽고 해당 값을
서브쿼리에서
참조하여 서브쿼리
결과에 존재하지
않으면 TRUE 반환

```
SELECT EMP_ID, EMP_NAME, '관리자' AS 구분
FROM   EMPLOYEE E
WHERE  EXISTS (SELECT NULL
                FROM   EMPLOYEE
                WHERE  E.EMP_ID = MGR_ID)

UNION

SELECT EMP_ID, EMP_NAME, '직원'
FROM   EMPLOYEE E2
WHERE  NOT EXISTS (SELECT NULL
                   FROM   EMPLOYEE
                   WHERE  E2.EMP_ID = MGR_ID)

ORDER BY 3, 1;
```

EMP_ID	EMP_NAME	구분
100	한선기	관리자
101	강중훈	관리자
104	안석규	관리자
141	김예수	관리자
174	전우성	관리자
200	고승우	관리자
102	최만식	직원
103	정도연	직원
107	조재형	직원
124	정지현	직원
143	나승원	직원
144	김순이	직원
149	성해교	직원
176	엄정하	직원
178	심하균	직원
201	박하일	직원
202	권상후	직원
205	임영애	직원
206	엄정하	직원
207	김술오	직원
208	이중기	직원
210	감우섭	직원

참고 존재 여부를 확인하는 목적이므로 서브쿼리에서 특정 값을 조회할 필요가 없음