

# sql class day2



## 조건절에서

### ▼ 조건절 중 여러 개의 조건절

```
select *  
from 테이블  
where 조건1  
and 조건2
```

### ▼ 비교연산자

=	같다
!=	다르다 ( 산업표준 : <> )
>	크다
<	작다
> =	크거나 같다
< =	작거나 같다

```
select *  
from employees  
where employee_id = 100;
```

질의 결과 x

SQL | 인출된 모든 행: 1(0.77초)

	EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY
1	100	Steven	King	SKING	515.123.4567	03/06/17	AD_PRES	24000

## ▼ 논리연산자

AND	모든 조건을 동시에 전부 만족할 때만 true
OR	조건 중 한 가지만 만족해도 true
NOT	조건의 반대 결과를 반환한다

### AND 사용

```
select *  
from employees  
where salary > 4000  
      and job_id = 'IT_PROG';
```

### OR 사용

```
select *  
from employees  
where salary > 10000  
      or job_id = 'IT_PROG';
```

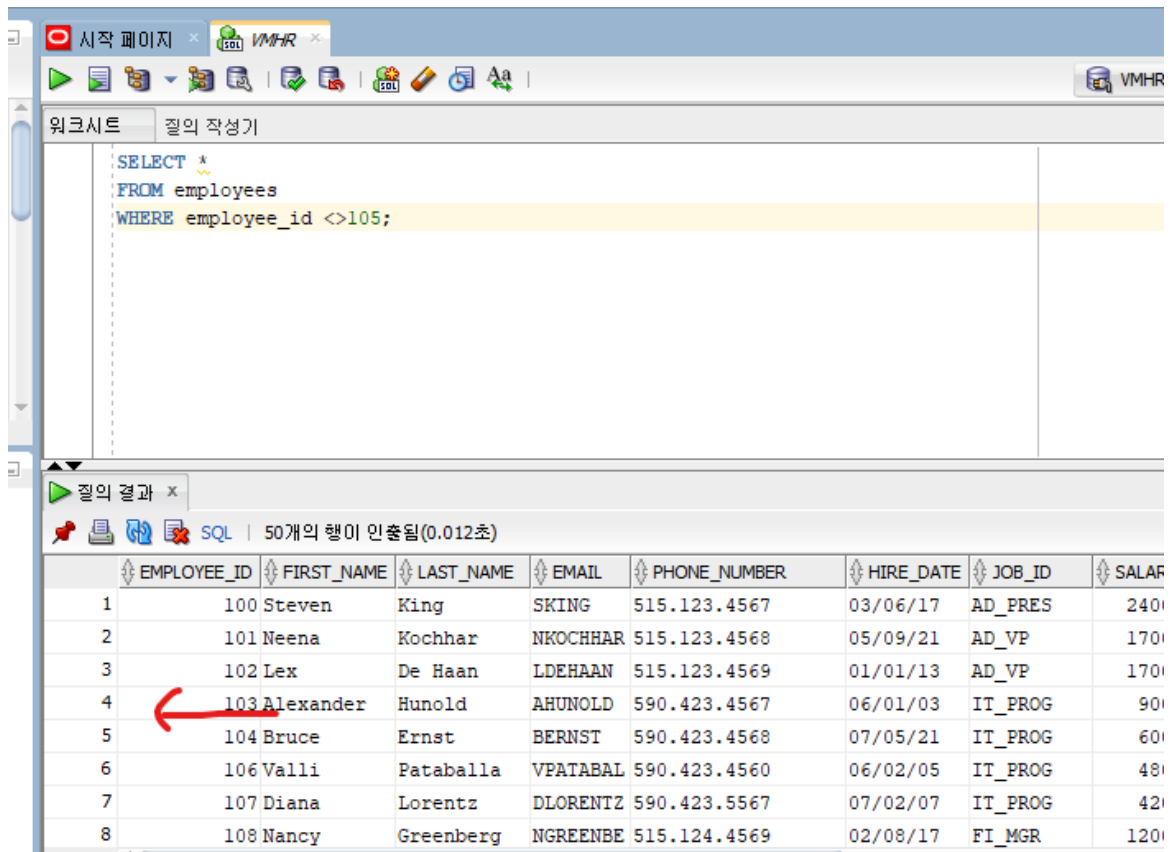
### 복합 사용

```
select *  
from employees  
where salary > 4000  
      AND job_id = 'IT_PROG'  
      or job_id = 'FI_ACCOUNT';
```

### NOT 사용

```
SELECT *  
FROM employees
```

WHERE employee\_id <>105;      \*\* <> 표준적인 not의 의미 \*\*



SQL | 50개의 행이 인출됨(0.012초)

	EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY
1	100	Steven	King	SKING	515.123.4567	03/06/17	AD_PRES	24000
2	101	Neena	Kochhar	NKOCHHAR	515.123.4568	05/09/21	AD_VP	17000
3	102	Lex	De Haan	LDEHAAN	515.123.4569	01/01/13	AD_VP	17000
4	103	Alexander	Hunold	AHUNOLD	590.423.4567	06/01/03	IT_PROG	9000
5	104	Bruce	Ernst	BERNST	590.423.4568	07/05/21	IT_PROG	6000
6	106	Valli	Pataballa	VPATABAL	590.423.4560	06/02/05	IT_PROG	4800
7	107	Diana	Lorentz	DLORENTZ	590.423.5567	07/02/07	IT_PROG	4200
8	108	Nancy	Greenberg	NGREENBE	515.124.4569	02/08/17	FI_MGR	12000

⇒ 105번만 빠진 것을 볼 수 있음

## ▼ NULL / NOT NULL

```
SELECT *  
FROM employees  
WHERE manager_id IS NOT NULL;
```



## 함수 사용

-웹개발에서 자주 쓰이지는 않음-

### ▼ 함수 의미( 단일행 함수 / 그룹 함수 )

#### • 단일행 함수

- 특정 행에 적용
- 데이터 값을 하나씩 계산함
- 대문자 / 소문자 / 첫 글자만 대문자 ( BOY, boy, Boy )  
※ 프로그램마다 차이 존재
- 글자 자르기 ( SUBSTR('원본글자', 시작위치, 자를 갯수) )
- 글자 바꾸기 ( REPLACE('문자열', '특정 문자', '바꿀 문자') )
- 특정 문자로 자리 채우기 ( LPAD , RPAD )
- 문자열 / 열이름에서 특정 문자열삭제 ( LTRIM / RTRIM )

#### • 그룹 함수

- 여러 행에 함수가 적용되어 하나의 결과를 나타냄
- 그룹 전체 적용
- 여러 개의 값을 그룹으로 계산함

##### ▼ count 갯수 ( null값도 셈 )

- count는 null값도 세는 특성상, 어떤 열로도 동일한 값이 나오기 때문에 주로 \* 사용

```
SELECT COUNT (*)  
FROM employees;
```

##### ▼ sum 합계 ( null값 제외 계산 )

```
SELECT SUM(salary) 급여합계
FROM employees;
```

#### ▼ avg 평균 ( null값 제외 계산 )

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

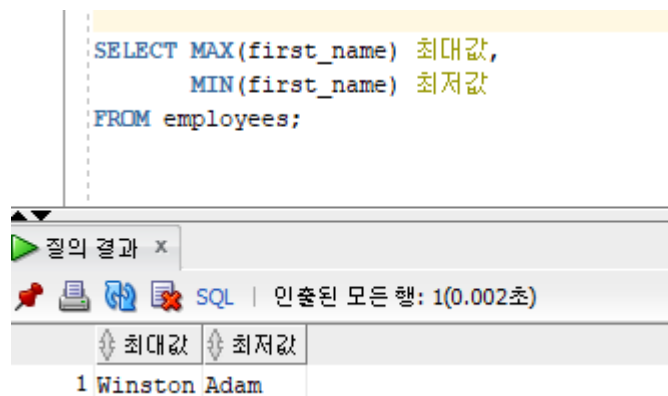
#### ▼ max 최대값 ( null값 제외 계산 )

```
SELECT MAX(salary) 최대급여
FROM employees;
```

#### ▼ min 최소값 ( null값 제외 계산 )

```
SELECT MIN(salary) 최저급여
FROM employees;
```

#### ▼ 최대값/최소값은 문자도 가능하다 ?!



## ▼ 단일행 함수 살펴보기

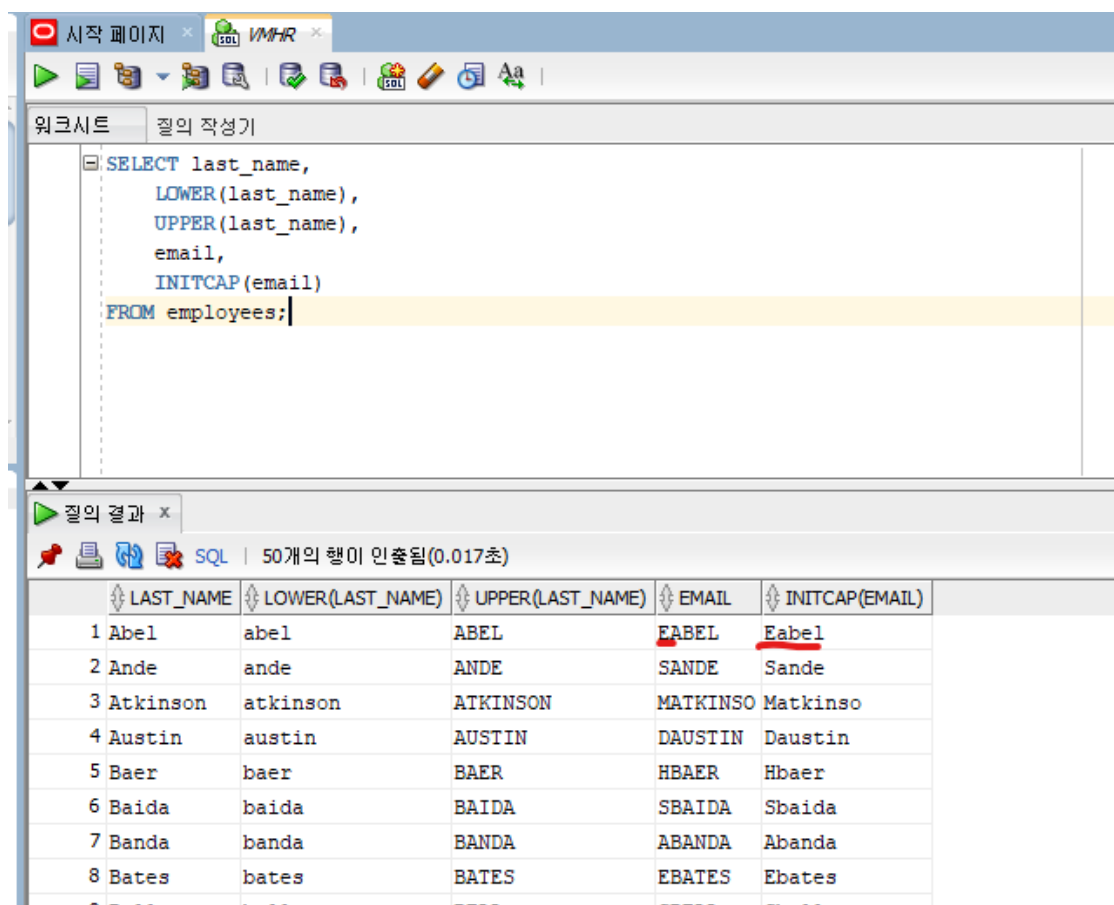
### ▼ 문자 관련 함수

LOWER ⇒ 소문자로

UPPER ⇒ 대문자로

INITCAP ⇒ 첫 글자만 대문자로

```
SELECT last_name,  
       LOWER(last_name),  
       UPPER(last_name),  
       email,  
       INITCAP(email)  
FROM employees;
```



The screenshot shows a SQL IDE window with a query editor and a results pane. The query editor contains the following SQL statement:

```
SELECT last_name,  
       LOWER(last_name),  
       UPPER(last_name),  
       email,  
       INITCAP(email)  
FROM employees;
```

The results pane displays the output of the query, showing 50 rows of data. The columns are LAST\_NAME, LOWER(LAST\_NAME), UPPER(LAST\_NAME), EMAIL, and INITCAP(EMAIL). The first few rows are as follows:

	LAST_NAME	LOWER(LAST_NAME)	UPPER(LAST_NAME)	EMAIL	INITCAP(EMAIL)
1	Abel	abel	ABEL	EABEL	Eabel
2	Ande	ande	ANDE	SANDE	Sande
3	Atkinson	atkinson	ATKINSON	MATKINSO	Matkinso
4	Austin	austin	AUSTIN	DAUSTIN	Daustin
5	Baer	baer	BAER	HBAER	Hbaer
6	Baida	baida	BAIDA	SBAIDA	Sbaida
7	Banda	banda	BANDA	ABANDA	Abanda
8	Bates	bates	BATES	EBATES	Ebates

## SUBSTR ⇒ 글자 자르기

SUBSTR( '원본글자', 시작위치, 자를 개수 ) ⇒ SQL은 인덱스 1로 시작

```
SELECT job_id, SUBSTR(job_id, 1, 2)
FROM employees;
```

The screenshot shows a SQL IDE window with a query editor and a results pane. The query editor contains the following SQL statement:

```
SELECT job_id, SUBSTR(job_id, 1, 2)
FROM employees;
```

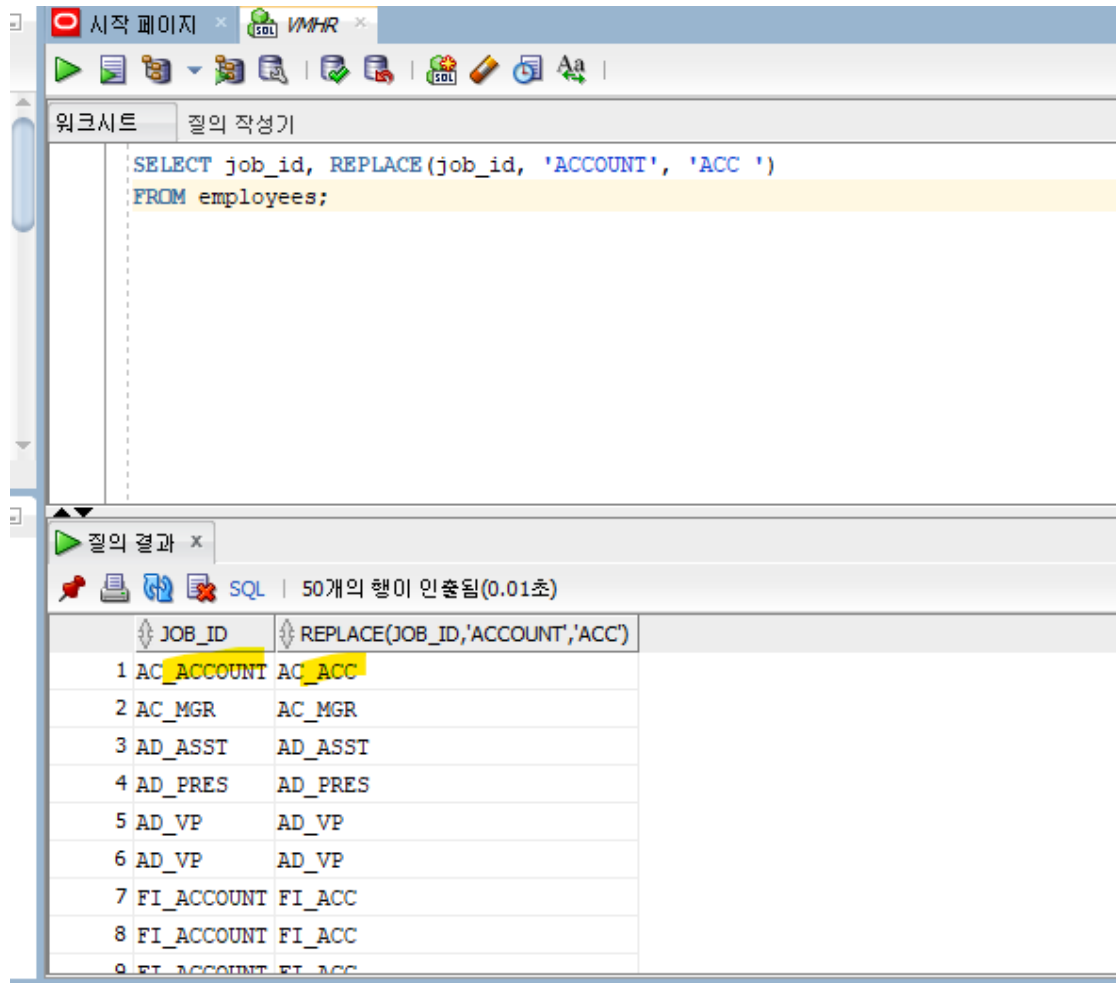
The results pane displays the output of the query, showing 50 rows of data. The columns are JOB\_ID and SUBSTR(JOB\_ID,1,2). The first 9 rows are visible in the screenshot:

	JOB_ID	SUBSTR(JOB_ID,1,2)
1	AC_ACCOUNT	AC
2	AC_MGR	AC
3	AD_ASST	AD
4	AD_PRES	AD
5	AD_VP	AD
6	AD_VP	AD
7	FI_ACCOUNT	FI
8	FI_ACCOUNT	FI
9	FI_ACCOUNT	FI

## REPLACE ⇒ 특정 문자 찾아서 바꾸기

## REPLACE ( '문자열', '특정문자', '바꿀문자' )

```
SELECT job_id, REPLACE(job_id, 'ACCOUNT', 'ACC ')
FROM employees;
```



SQL | 50개의 행이 인출됨(0.01초)

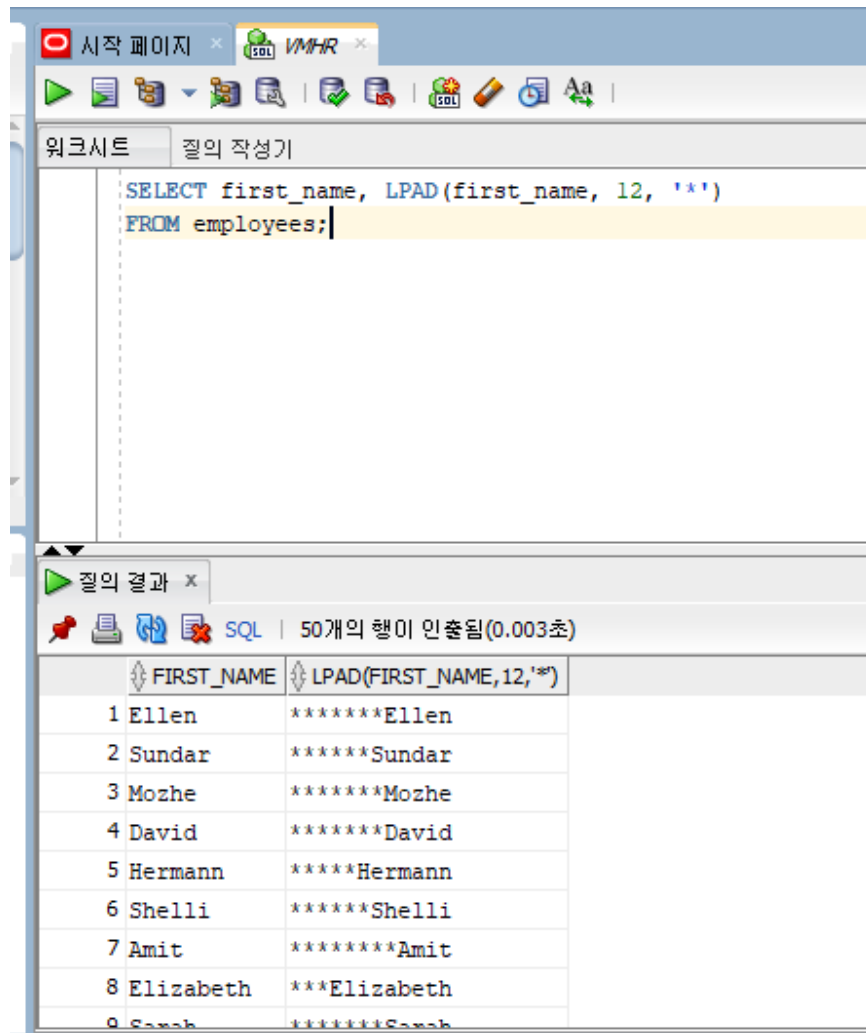
	JOB_ID	REPLACE(JOB_ID,'ACCOUNT','ACC')
1	AC_ACCOUNT	AC_ACC
2	AC_MGR	AC_MGR
3	AD_ASST	AD_ASST
4	AD_PRES	AD_PRES
5	AD_VP	AD_VP
6	AD_VP	AD_VP
7	FI_ACCOUNT	FI_ACC
8	FI_ACCOUNT	FI_ACC
9	FI_ACCOUNT	FI_ACC

**LPAD / RPAD ⇒ 특정문자로 자리 채우기**

**LPAD( '문자열', 만들어질 자릿수, '채울 문자' )**



```
SELECT first_name, LPAD(first_name, 12, '*')
FROM employees;
```



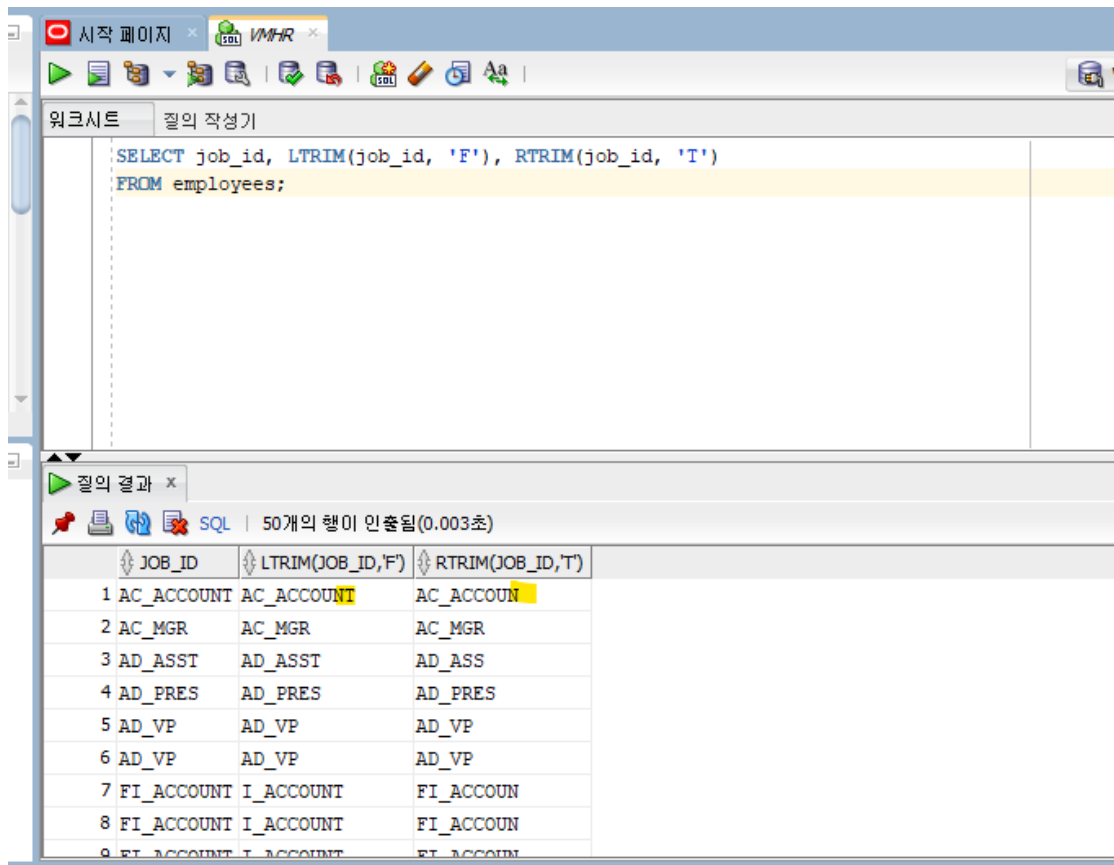
SQL | 50개의 행이 인출됨(0.003초)

	FIRST_NAME	LPAD(FIRST_NAME,12,'*')
1	Ellen	*****Ellen
2	Sundar	*****Sundar
3	Mozhe	*****Mozhe
4	David	*****David
5	Hermann	*****Hermann
6	Shelli	*****Shelli
7	Amit	*****Amit
8	Elizabeth	***Elizabeth
9	Sarah	*****Sarah

**LTRIM / RTRIM ⇒ 삭제 ( 불필요한 여백 삭제할 때 사용 多 )**

**LTRIM ( '문자열'or'열이름' , '삭제할문자' )**

```
SELECT job_id, LTRIM(job_id, 'F'), RTRIM(job_id, 'T')
FROM employees;
```



워크시트: 질의 작성기

```
SELECT job_id, LTRIM(job_id, 'F'), RTRIM(job_id, 'T')
FROM employees;
```

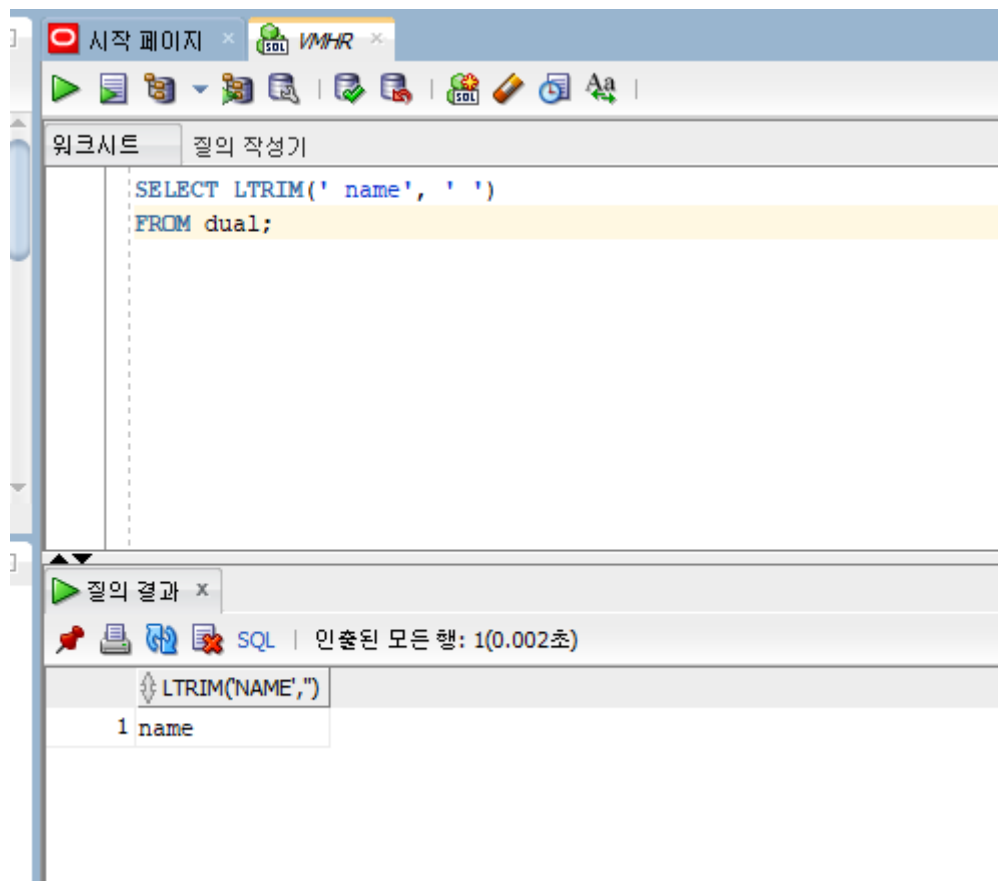
질의 결과 x

SQL | 50개의 행이 인출됨(0.003초)

	JOB_ID	LTRIM(JOB_ID,'F')	RTRIM(JOB_ID,'T')
1	AC_ACCOUNT	AC_ACCOUNT	AC_ACCOUNT
2	AC_MGR	AC_MGR	AC_MGR
3	AD_ASST	AD_ASST	AD_ASS
4	AD PRES	AD PRES	AD PRES
5	AD VP	AD VP	AD VP
6	AD VP	AD VP	AD VP
7	FI_ACCOUNT	I_ACCOUNT	FI_ACCOUNT
8	FI_ACCOUNT	I_ACCOUNT	FI_ACCOUNT
9	FI_ACCOUNT	I_ACCOUNT	FI_ACCOUNT

## DUAL 테이블과 함께 활용한 쿼리문

```
SELECT LTRIM(' name', ' ')
FROM dual;
```



## ▼ 숫자 관련 함수

**ROUND ⇒ 반올림 함수**







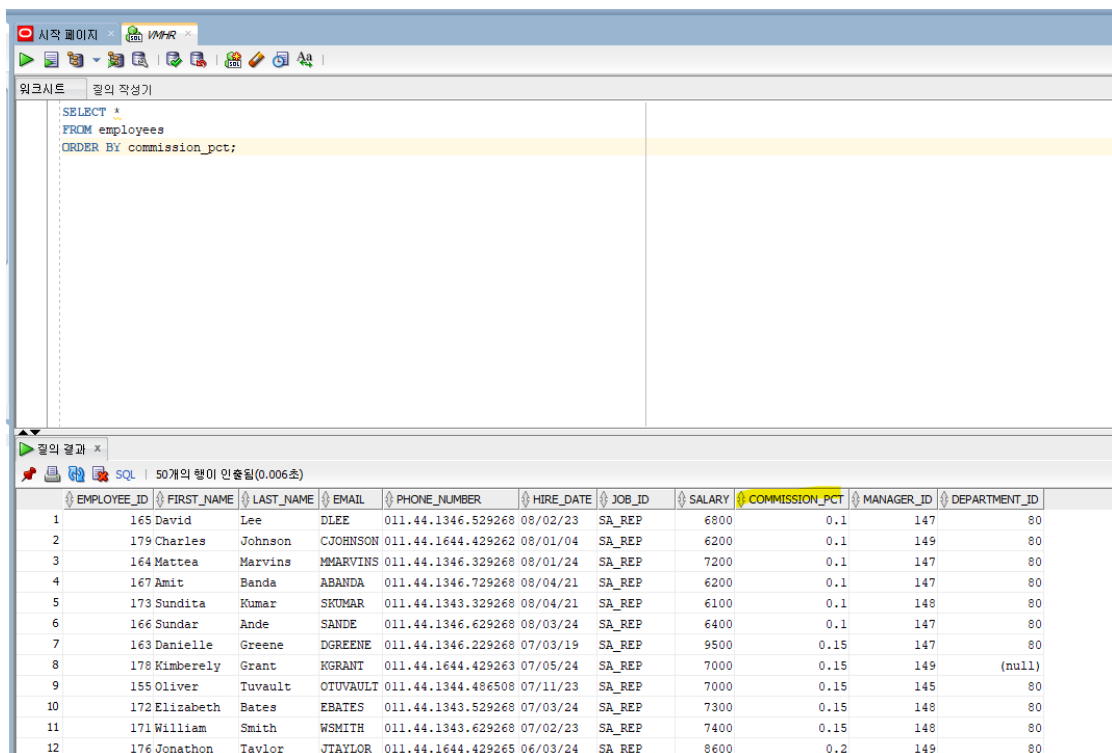
## ▼ 기타등등 함수

NVL 함수 ⇒ null값을 특정 값으로 치환

NVL ( 특정 열, 치환값 )

```
SELECT salary * NVL(commission_pct, 1)
FROM employees
ORDER BY commission_pct;
```

⇒ **commission\_pct** 값으로 null이 있는 경우, 1로 치환하겠다



EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	COMMISSION_PCT	MANAGER_ID	DEPARTMENT_ID
1	David	Lee	DLEE	011.44.1346.529268	08/02/23	SA_REP	6800	0.1	147	80
2	Charles	Johnson	CJOHNSON	011.44.1644.429262	08/01/04	SA_REP	6200	0.1	149	80
3	Mattea	Marvins	MMARVINS	011.44.1346.329268	08/01/24	SA_REP	7200	0.1	147	80
4	Amit	Banda	ABANDA	011.44.1346.729268	08/04/21	SA_REP	6200	0.1	147	80
5	Sundita	Kumar	SKUMAR	011.44.1343.329268	08/04/21	SA_REP	6100	0.1	148	80
6	Sundar	Ande	SANDE	011.44.1346.629268	08/03/24	SA_REP	6400	0.1	147	80
7	Danielle	Greene	DGREENE	011.44.1346.229268	07/03/19	SA_REP	9500	0.15	147	80
8	Kimberely	Grant	KGRANT	011.44.1644.429263	07/05/24	SA_REP	7000	0.15	149	(null)
9	Oliver	Tuvault	OTUVAULT	011.44.1344.486508	07/11/23	SA_REP	7000	0.15	145	80
10	Elizabeth	Bates	EBATES	011.44.1343.529268	07/03/24	SA_REP	7300	0.15	148	80
11	William	Smith	WSMITH	011.44.1343.629268	07/02/23	SA_REP	7400	0.15	148	80
12	Jonathon	Taylor	JTAYLOR	011.44.1644.429265	06/03/24	SA_REP	8600	0.2	149	80

⇒ salary와 commission\_pct 값을 계산하려고 하면 문제 발생  
왜냐하면, commission 지급받지 않은 직원 (null) 이 있기 때문.

**before ( null처리 안함 )**



시작 페이지 x VMHR x

워크시트 | 질의 작성기

```
SELECT salary * commission_pct
FROM employees
ORDER BY commission_pct;
```

질의 결과 x

SQL | 50개의 행이 인출됨(0.007초)

	SALARY*COMMISSION_PCT
31	2250
32	3500
33	3325
34	3150
35	5600
36	(null)
37	(null)
38	(null)
39	(null)
40	(null)
41	(null)
42	(null)

after ( null값을 1로 치환 )

The screenshot shows a SQL IDE window with a query editor and a results pane. The query editor contains the following SQL statement:

```
SELECT salary * NVL(commission_pct, 1)
FROM employees
ORDER BY commission_pct;
```

The results pane shows the output of the query, displaying the calculated salary for each employee, ordered by their commission percentage. The results are as follows:

	SALARY*NVL(COMMISSION_PCT,1)
52	2900
53	2800
54	2600
55	2500
56	8000
57	8200
58	7900
59	6500
60	5800
61	3200
62	2700
63	2400
64	2200
65	3300

## << null 처리 방법 - 다양 >>

- null 자리에 최빈값 넣기
- null 자리에 평균값 넣기

- null 자리에 0 넣기
- null 자리에 1 넣기

## DECODE ⇒ 조건 처리하기

### DECODE ( 열이름, 조건값, 치환값, 기본값 )

치환값	————>	조건을 만족할 경우
기본값	————>	조건을 만족하지 않을 경우

```
SELECT department_id,
       employee_id,
       first_name,
       salary as "원래급여",
       DECODE(department_id, 60, salary * 1.1, salary)
FROM employees;
```

⇒ 부서번호가 60이면 salary \* 1.1 하고, 아니라면 salary 그대로

시작 페이지 x VMHR x

워크시트 | 질의 작성기

```

SELECT department_id,
       employee_id,
       first_name,
       salary as "원래급여",
       DECODE(department_id, 60, salary * 1.1, salary)
FROM employees;

```

질의 결과 x

SQL | 50개의 행이 인출됨(0.003초)

	DEPARTMENT_ID	EMPLOYEE_ID	FIRST_NAME	원래급여	DECODE(DEPARTMENT_ID,60,SALARY*1.1,SALARY)
1	90	100	Steven	24000	24000
2	90	101	Neena	17000	17000
3	90	102	Lex	17000	17000
4	60	103	Alexander	9000	9900
5	60	104	Bruce	6000	6600
6	60	105	David	4800	5280
7	60	106	Valli	4800	5280
8	60	107	Diana	4200	4620
9	100	108	Nancy	12008	12008
10	100	109	Daniel	9000	9000
11	100	110	John	8200	8200
12	100	111	Ismael	7700	7700
13	100	112	Jose Manuel	7800	7800

```

SELECT department_id,
       employee_id,
       first_name,
       salary as "원래급여",
       DECODE(department_id, 60, salary * 1.1, salary),
       DECODE(department_id, 60, '급여인상', ' ')
FROM employees;

```

시작 페이지 | WMFR

워크시트 | 질의 작성기

```

SELECT department_id,
       employee_id,
       first_name,
       salary as "원래급여",
       DECODE(department_id, 60, salary * 1.1, salary),
       DECODE(department_id, 60, '급여인상', ' ')
FROM employees;

```

질의 결과 | 50개의 행이 인출됨(0.003초)

	DEPARTMENT_ID	EMPLOYEE_ID	FIRST_NAME	원래급여	DECODE(DEPARTMENT_ID,60,SALARY*1.1,SALARY)	DECODE(DEPARTMENT_ID,60,'급여인상','')
1	90	100	Steven	24000	24000	
2	90	101	Neena	17000	17000	
3	90	102	Lex	17000	17000	
4	60	103	Alexander	9000	9900	급여인상
5	60	104	Bruce	6000	6600	급여인상
6	60	105	David	4800	5280	급여인상
7	60	106	Valli	4800	5280	급여인상
8	60	107	Diana	4200	4620	급여인상
9	100	108	Nancy	12008	12008	
10	100	109	Daniel	9000	9000	
11	100	110	John	8200	8200	
12	100	111	Ismael	7700	7700	
...	...	...	...	...	...	...

**CASE ⇒ 경우의 수가 여러 개일 경우**

**CASE WHEN THEN**

**WHEN THEN**

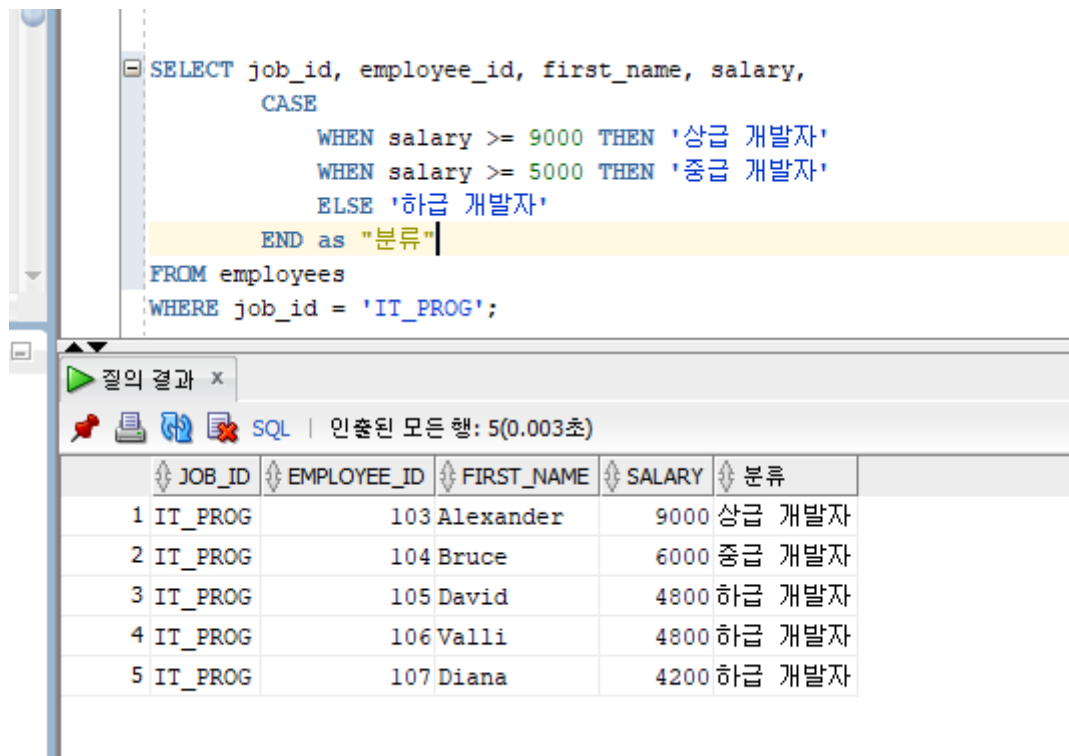
**ELSE**

**END**

```

SELECT job_id, employee_id, first_name, salary,
       CASE
         WHEN salary >= 9000 THEN '상급 개발자'
         WHEN salary >= 5000 THEN '중급 개발자'
         ELSE '하급 개발자'
       END as "분류"
FROM employees
WHERE job_id = 'IT_PROG';

```



The screenshot shows a SQL IDE window with a query editor and a results pane. The query in the editor is the same as the one in the first block. The results pane, titled "질의 결과 x", shows the execution of the query. It indicates that 5 rows were returned in 0.003 seconds. Below this, a table displays the results of the query.

	JOB_ID	EMPLOYEE_ID	FIRST_NAME	SALARY	분류
1	IT_PROG	103	Alexander	9000	상급 개발자
2	IT_PROG	104	Bruce	6000	중급 개발자
3	IT_PROG	105	David	4800	하급 개발자
4	IT_PROG	106	Valli	4800	하급 개발자
5	IT_PROG	107	Diana	4200	하급 개발자

## ▼ 순위 매기기 ( RANK, DENSE\_RANK, ROW\_NUMBER )

**RANK** ⇒ 공통 순위 만큼 건너뛰고 순위 매기기 ( 1 , 2 , 2 , 4 )

**DENSE\_RANK** ⇒ 공통 순위를 건너뛰지 않고 순위 ( 1 , 2 , 2 , 3 )

dense : 뽕뽕한, 밀집된

**ROW\_NUMBER** ⇒ 공통 순위 없이 출력 ( 1 , 2 , 3 , 4 )

```
SELECT employee_id, first_name, salary,  
       RANK()OVER (ORDER BY salary DESC) rank순위,  
       DENSE_RANK() OVER(ORDER BY salary DESC) dense순위,  
       ROW_NUMBER() OVER(ORDER BY salary DESC) row순위  
FROM employees;
```

```

SELECT employee_id, first_name, salary,
       RANK() OVER (ORDER BY salary DESC) rank순위,
       DENSE_RANK() OVER (ORDER BY salary DESC) dense순위,
       ROW_NUMBER() OVER (ORDER BY salary DESC) row순위
FROM employees;

```

질의 결과 x

SQL | 50개의 행이 인출됨(0.003초)

	EMPLOYEE_ID	FIRST_NAME	SALARY	RANK순위	DENSE순위	ROW순위
1	100	Steven	24000	1	1	1
2	101	Neena	17000	2	2	2
3	102	Lex	17000	2	2	3
4	145	John	14000	4	3	4
5	146	Karen	13500	5	4	5
6	201	Michael	13000	6	5	6
7	108	Nancy	12008	7	6	7
8	205	Shelley	12008	7	6	8
9	147	Alberto	12000	9	7	9
10	168	Lisa	11500	10	8	10
11	114	Den	11000	11	9	11
12	148	Gerald	11000	11	9	12
13	174	Ellen	11000	11	9	13
14	149	Eleni	10500	14	10	14
15	162	Clara	10500	14	10	15

## ▼ 듀얼 테이블



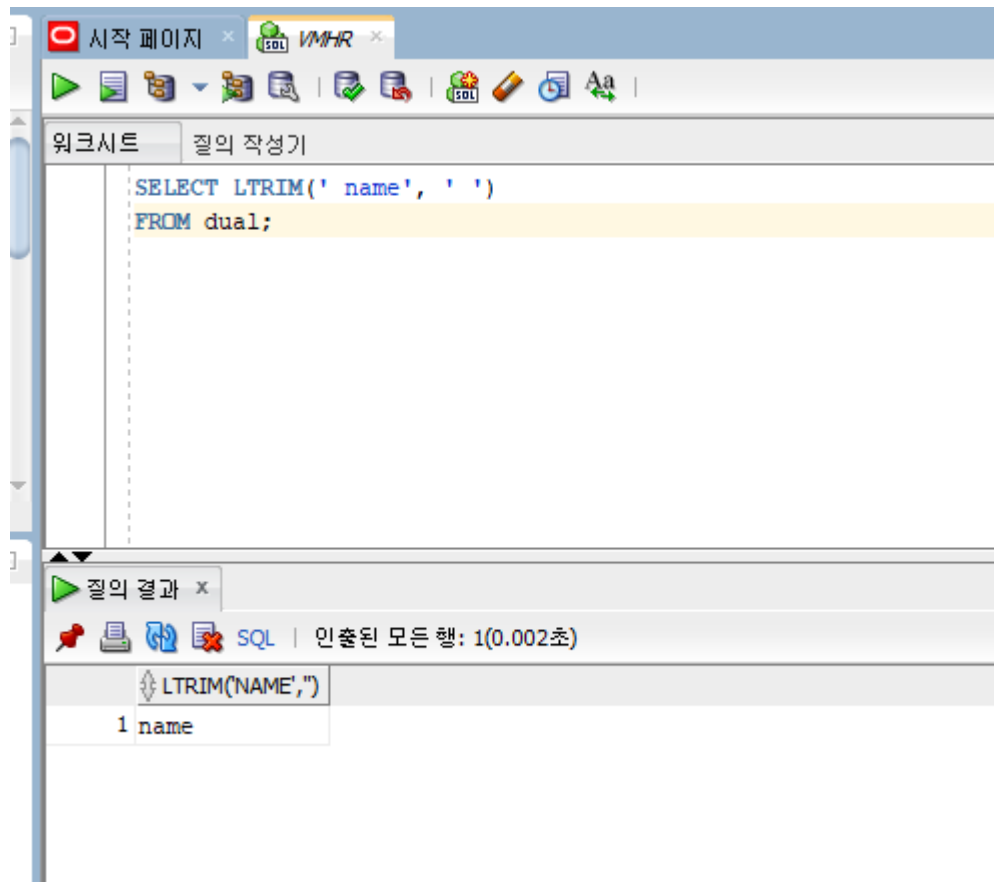
워크시트 | 질의 작성기

```
SELECT 1 + '2'  
FROM dual;
```


질의 결과 x

SQL | 인출된 모든 행: 1(0.008초)

	1+2'
1	3



- 듀얼 테이블
  - ⇒ dummy테이블, 특정 테이블을 활용하지 않고  
문법적으로 오류를 회피하고자 할때 사용하는 일종의 가상의 테이블
- 테이블을 쓰지 않고 특정 문자열을 활용하려고 할 때,  
어떠한 쿼리문을 활용할 것일 때 가상의 테이블 DUAL 활용

 **Group by 그룹으로 묶기**  
 ▼ **group by A : A별로 select하겠다**

```
SELECT job_id, AVG(salary)
FROM employees
group by job_id;
```

```
SELECT job_id, AVG(salary)
FROM employees
group by job_id;
```

	JOB_ID	AVG(SALARY)
1	IT_PROG	5760
2	AC_MGR	12008
3	AC_ACCOUNT	8300
4	ST_MAN	7280
5	PU_MAN	11000
6	AD_ASST	4400
7	AD_VP	17000
8	SH_CLERK	3215
9	FI_ACCOUNT	7920
10	FI_MGR	12008

## ▼ group by + order by : 그룹으로 묶은 것을 정렬

```
SELECT job_id, AVG(salary), SUM(salary), COUNT(*)
FROM employees
group by job_id
ORDER BY AVG(salary) DESC;
```

<pre> SELECT job_id, AVG(salary), SUM(salary), COUNT(*) FROM employees group by job_id ORDER BY AVG(salary) DESC; </pre>				
<div> <div> <div>실행</div> <div>결과</div> <div>SQL</div> </div> <div>인출된 모든 행: 19(0.005초)</div> </div>				
	JOB_ID	AVG(SALARY)	SUM(SALARY)	COUNT(*)
1	AD_PRES	24000	24000	1
2	AD_VP	17000	34000	2
3	MK_MAN	13000	13000	1
4	SA_MAN	12200	61000	5
5	AC_MGR	12008	12008	1
6	FI_MGR	12008	12008	1
7	PU_MAN	11000	11000	1
8	PR_REP	10000	10000	1
9	SA_REP	8350	250500	30
10	AC_ACCOUNT	8300	8300	1
11	FI_ACCOUNT	7920	39600	5
12	ST_MAN	7280	36400	5
13	HR_REP	6500	6500	1
14	MK_REP	6000	6000	1

1. 그룹으로 묶은 후
2. 정렬해야 하므로 ⇒ 순서 바뀌면 안됨

```

SELECT job_id, AVG(salary) 급여평균, SUM(salary) 급여합계, COUNT(*) 부서인원수
FROM employees
group by job_id
ORDER BY 급여평균 DESC, 부서인원수 DESC;

```

질의 결과 x

SQL | 인출된 모든 행: 19(0.005초)

	JOB_ID	급여평균	급여합계	부서인원수
1	AD_PRES	24000	24000	1
2	AD_VP	17000	34000	2
3	MK_MAN	13000	13000	1
4	SA_MAN	12200	61000	5
5	AC_MGR	12008	12008	1
6	FI_MGR	12008	12008	1
7	PU_MAN	11000	11000	1
8	PR_REP	10000	10000	1
9	SA_REP	8350	250500	30
10	AC_ACCOUNT	8300	8300	1
11	FI_ACCOUNT	7920	39600	5
12	ST_MAN	7280	36400	5



## sql문 종류

## ▼ DML ( 데이터 조작 언어, Data Manipulation Language)

sql에서의 용어	기능	웹개발에서의 용어
SELECT	조회	Read
INSERT	삽입	Create
UPDATE	수정	Update
DELETE	삭제	Delete

## ▼ DDL ( 데이터 정의 언어, Data Definition Language )

⇒ DB 생성, Table 생성, 삭제 drop

⇒ CREATE, DROP

## ▼ DCL ( 데이터 제어 언어, Data Control Language )

- 권한 관리



삭제 관련

sql문 종류	삭제	역할
DML	delete	데이터만 삭제
DDL	truncate	구조를 남기고 데이터만 전체 삭제
DDL	drop	구조+데이터 완전 삭제



## 테이블 생성

### ▼ 기본구조

```
CREATE TABLE 테이블이름
  {열이름1    속성,
   열이름2    속성
   ...
   ...};
```

```
CREATE TABLE board
{
  bno number,
  btitle varchar2(50),
  bwriter varchar2(10),
  bcontent varchar2(500)
};
```

## ▼ 이름 정의 방법

- 동일한 이름의 테이블이 존재할 수 없다.
- 예약어(이미 사용중인 명령어)는 이름으로 사용할 수 없다.
- 반드시 문자로 시작해야 한다. ( 숫자로 시작 불가능 )
- 가능하면 의미있는 단어를 사용할 것

## ▼ 테이블 수정 中

### ▼ 항목 추가

```
ALTER TABLE member
ADD (mgender varchar2(10));
```

### ▼ 항목 타입 변경

```
-- 기존엔 10개짜리였는데 20개짜리로 변경--
ALTER TABLE member MODIFY (MTEL varchar2(20));
```

### ▼ 항목 종류 변경

```
ALTER TABLE member RENAME COLUMN MTEL TO MPHONE;
```

### ▼ 항목 삭제

```
ALTER TABLE member DROP COLUMN MADDR;
```

## ▼ 테이블의 모든 항목 일괄삭제 (테이블은 그대로)

```
TRUNCATE TABLE member;
```

## ▼ 테이블 자체를 삭제

```
DROP TABLE board;
```



## 뷰 View

### ▼ View가 무엇?

⇒ View는 데이터베이스에서 가상의 테이블이다.

실제로 테이블에 저장되어 있는 데이터를 그대로 사용하는 것이 아니라,  
필요한 데이터만 추출하여 새로운 가상의 테이블을 만들어서 사용한다.

뷰는, 테이블과 동일하게 사용자에 의해 생성되고,

SQL문으로 조작이 가능하지만,

데이터는 뷰가 참조한 원본 테이블에 저장되어 있다.

뷰는 데이터를 중복해서 저장하지 않아도 되므로 데이터 정규화에 도움을 준다.

또한, 특정 사용자가 필요한 데이터만 조회하도록 제한하거나,

여러 개의 테이블에서 데이터를 조합하여 표시할 수 있다.

### ▼ 뷰의 장점 > 보안성을 높임

#### ▼ 예시 설명

예를 들어, 신입사원한테 vip회원한테 생일카드를 보내라고 했어.

그러면 신입사원이 그 회원의 주소를 알아야겠지?

그런데 그 주소 하나 때문에 모든 정보를 다 보게하면 너무 위험하잖아?



이런 경우에 원하는 내용만 담은 뷰를 만들어서 신입사원에게 보게 하는 거임

## ▼ 뷰 생성 코드

```
CREATE VIEW test_view AS  
SELECT a.employee_id, a.hire_date, b.department_name, b.job_title  
FROM employees A, emp_details_view B;
```