

Ch 3.

Basic SELECT

3.1 | 개요

3.2 | SELECT 구문

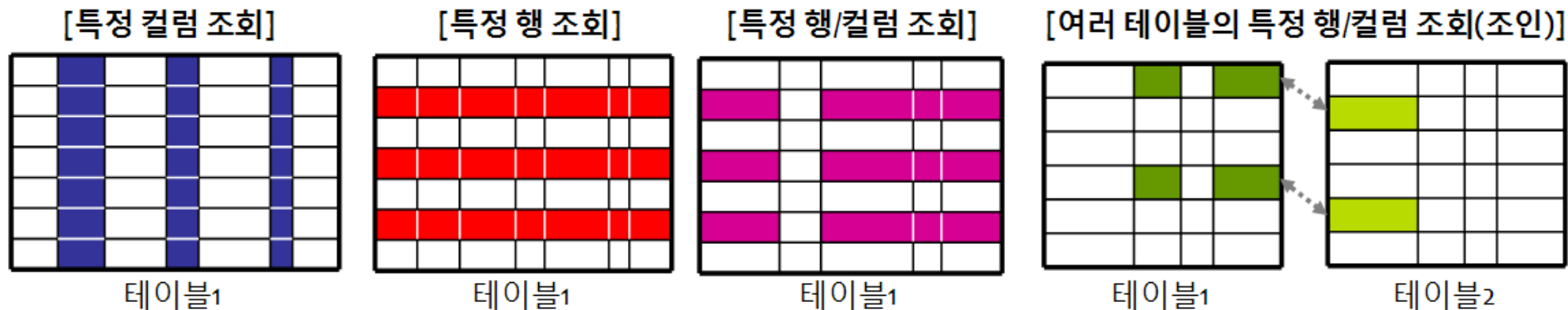
3.3 | 연산자^{operator}

Objective

- ❖ SELECT 구문의 기능을 설명할 수 있다
- ❖ SELECT 구문 실행 결과인 Result Set 개념을 설명할 수 있다
- ❖ SELECT 구문의 구성 요소들을 설명할 수 있다
- ❖ 데이터를 조회하기 위해 기본적인 SELECT 구문을 작성할 수 있다
- ❖ 작성된 SELECT 구문을 이해할 수 있다

3.1.1 데이터 조회 범위 및 결과

■ 데이터 조회 범위



■ 데이터 조회 결과

- 데이터를 조회한 결과를 'Result Set' 이라고 함
- SELECT 구문에 의해 반환된 행들의 집합을 의미
- Result Set에는 0개, 1개, 여러 개 행이 포함될 수 있음
- Result Set은 특정한 기준에 의해 정렬될 수 있음

3.1.2 SELECT 구문 작성 시 고려 사항 1

- 키워드^{Keyword}, 테이블 이름, 컬럼 이름은 대/소문자를 구분하지 않는다

```
SELECT DEPT_ID, DEPT_NAME  
FROM   DEPARTMENT;
```

```
select dept_id, dept_name  
from   department;
```

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

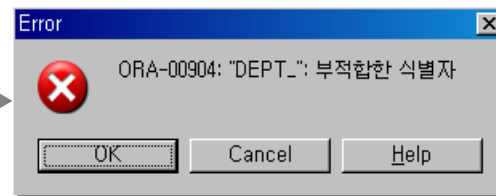
- 나누어 쓰기/들여 쓰기^{Indentation}를 하면 가독성^{Readability}이 좋아지고 편집이 쉬워진다

```
SELECT DEPT_ID, DEPT_NAME FROM DEPARTMENT;  
  
SELECT DEPT_ID, DEPT_NAME  
FROM   DEPARTMENT;  
  
SELECT DEPT_ID,  
       DEPT_NAME  
FROM   DEPARTMENT;
```

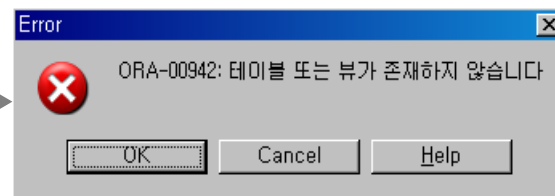
3.1.2 SELECT 구문 작성 시 고려 사항 2

- 키워드, 테이블 이름, 컬럼 이름은 약자로 줄여 쓰거나 분리할 수 없다

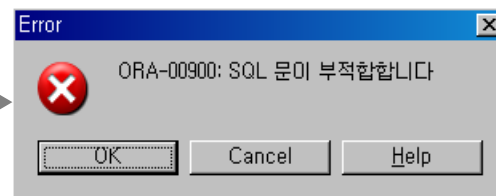
```
SELECT DEPT_  
        ID, DEPT_NAME  
FROM    DEPARTMENT;
```



```
SELECT DEPT_ID, DEPT_NAME  
FROM    DEPAR  
TMENT;
```



```
SEL  
ECT DEPT_ID, DEPT_NAME  
FR  
OM DEPARTMENT;
```



- 구문은 ;semi-colon 이나 /slash 로 종료된다

```
SELECT DEPT_NAME  
FROM    DEPARTMENT;
```

```
SELECT DEPT_NAME  
FROM    DEPARTMENT  
;
```

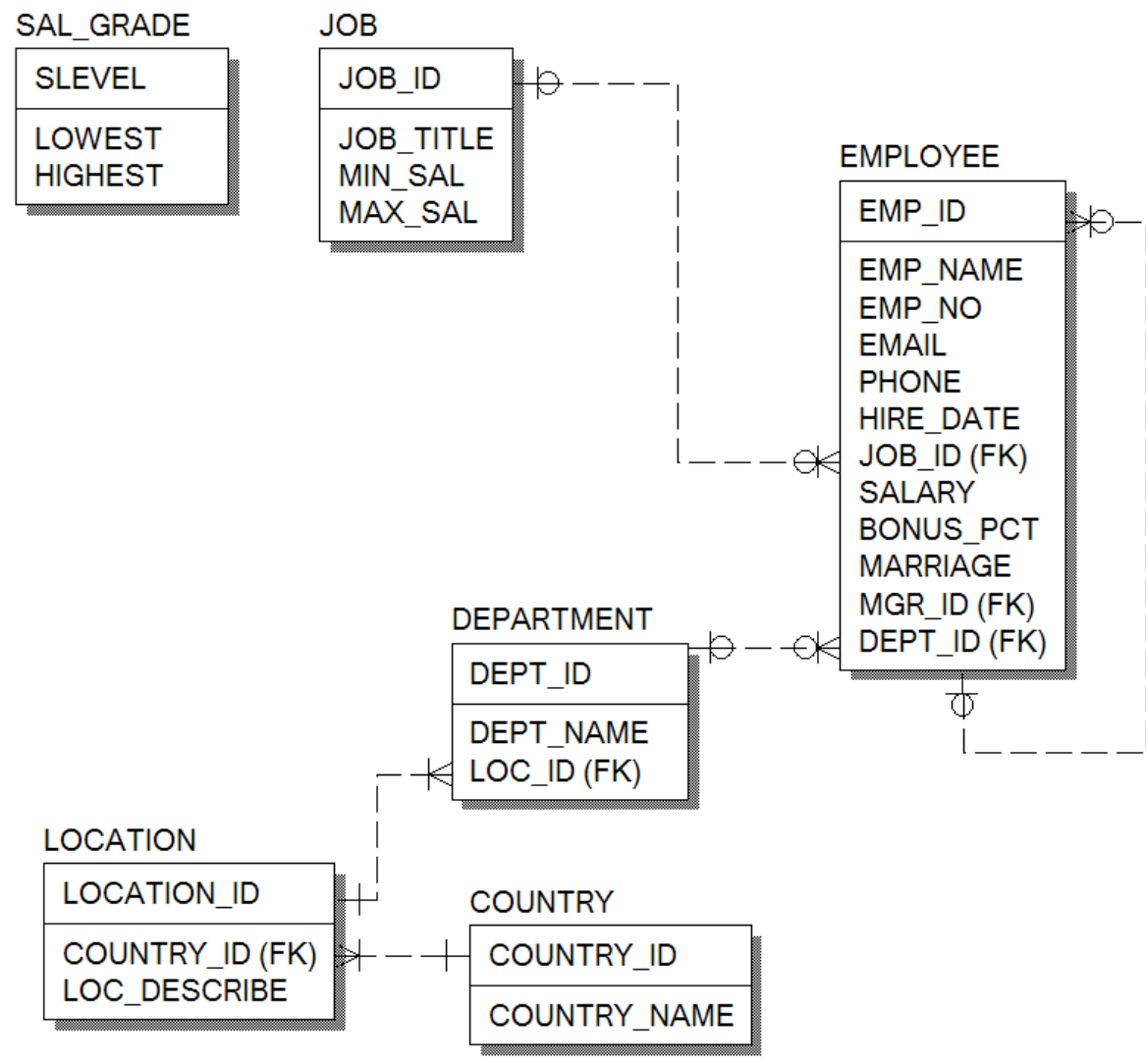
```
SELECT DEPT_NAME  
FROM    DEPARTMENT  
/
```

'/' 기호는 반드시
새로운 줄에서
사용해야 함

3.1.3 SELECT 구문 작성 가이드

- 키워드를 포함한 모든 SQL 문장은 대문자로 작성
- 한 칸 공백 사용
 - 단어와 단어 사이
 - ', ' 다음
 - 비교 연산자(<, >, = 등)의 앞뒤
- 줄 나눔 및 세로 열 맞춤
 - SELECT, FROM, WHERE 절은 각각 다른 줄에 왼쪽 정렬하여 작성
 - 불필요한 공백 라인은 사용하지 않는다

3.1.4 교재용 실습 테이블 샘플 ERD



3.2.1 기본 구문 1

```
SELECT * | { [DISTINCT] {{column_name | expr} {[AS] [ alias ]}, ...} }  
FROM   table_name  
WHERE  search_condition [ {AND | OR } , search_condition ...] ;
```

[구문 설명]

- 키워드 SELECT 다음에는 조회하려고 하는 컬럼 이름(또는 표현 식)을 기술
 - 모든 컬럼을 조회하는 경우 *^{asterisk} 기호를 사용할 수 있음 → 컬럼은 생성된 순서대로 표시됨
 - 여러 컬럼을 조회하는 경우 각 컬럼 이름은 쉼표로 구분(마지막 컬럼 이름 다음에는 쉼표를 사용하지 않음)
 - 조회 결과는 기술된 컬럼 이름 순서로 표시
- 키워드 FROM 다음에는 조회 대상 컬럼이 포함된 테이블 이름을 기술
- 키워드 WHERE 다음에는 행을 선택하는 조건을 기술
 - 제한 조건을 여러 개 포함할 수 있으며, 각각의 제한 조건은 논리 연산자로 연결
 - 제한 조건을 만족시키는 행(논리 연산 결과가 TRUE 인 행)들만 Result set에 포함

3.2.2 SELECT 사용 예 1

직원들의 사번과 이름을 조회하는 SELECT 구문

```
SELECT EMP_ID,  
       EMP_NAME  
FROM   EMPLOYEE;
```

결과(일부)

EMP_ID	EMP_NAME	
100	한선기	...
101	강중훈	...
102	최만식	...
103	정도연	...
104	안석규	...
107	조재형	...
124	정지현	...

3.2.2 SELECT 사용 예 2

직원들의 모든 정보를 조회하는 SELECT 구문

```
SELECT EMP_ID, EMP_NAME, EMP_NO, EMAIL, PHONE, HIRE_DATE, JOB_ID, SALARY, BONUS_PCT,
       MARRIAGE, MGR_ID, DEPT_ID
FROM   EMPLOYEE;

또는

SELECT * FROM EMPLOYEE;
```

결과(일부)


EMP_ID	EMP_NAME	EMP_NO	EMAIL	PHONE	HIRE_DATE	JOB_ID	SALARY	BONUS_PCT	MARRIAGE	MGR_ID	DEPT_ID
100	한선기	621133-1483658	sg_ahn@vcc.com	0199949999	90/04/01	J1	9000000	0.2	Y		90
101	강중훈	621136-1006405	jh_park@vcc.com	0193334433	04/04/30	J2	5500000		Y	100	90
102	최만식	861011-1940062	ms_choi@vcc.com	0198879908	95/12/30	J2	3600000		Y	101	90
103	정도연	631127-2519077	sy_kang@vcc.com	0196654436	97/06/03	J4	2600000		Y	104	60
104	안석규	651031-1962810	sg_han@vcc.com	0192347654	98/07/01	J3	3500000	0.25	Y	100	60
107	조재형	721128-1732822	jh_jo@vcc.com	0193325548	98/11/23	J3	3800000		Y	104	60
124	정지현	641231-2269080	jih_jeon@vcc.com	01922976129	04/07/15	J7	1500000		N	141	50
141	김예수	651122-2592930	hs_kim@vcc.com	0194087600	01/03/20	J5	2100000	0.1	Y	100	50
143	나승원	871024-1945881	sw_cha@vcc.com	0197243979	01/03/20	J5	2300000		Y	141	50
144	김순이	741122-2515789	sm_kim@vcc.com	0192213306	99/10/20	J3	3400000	0.1	Y	141	50
149	성해교	640524-2148639	hg_song@vcc.com	01992882295	03/08/16	J7	1900000		N	141	50

3.2.3 SELECT 사용 예 3 - 컬럼 값에 대한 산술 연산

컬럼 값에 대해 산술 연산한 결과를 조회할 수 있음

```
SELECT EMP_NAME,  
       SALARY*12,  
       ( SALARY+( SALARY*BONUS_PCT ) ) *12  
FROM   EMPLOYEE;
```

결과(일부)



EMP_NAME	SALARY*12	(SALARY+(SALARY*BONUS_PCT))*12
한선기	108000000	129600000
강중훈	66000000	
최만식	43200000	
정도연	31200000	
안석규	42000000	52500000
조재형	45600000	
정지현	18000000	

- 컬럼 헤더는 산술 연산 식으로 표시됨
- 실제 컬럼 값이 변경되는 것은 아님

3.2.4 SELECT 사용 - 컬럼 별칭^{Alias}

컬럼 별칭을 사용하면 SELECT 절에 기술된 내용과 동일하게 표시되는 실행 결과 헤더 부분을 변경할 수 있음

컬럼 별칭을 사용하지 않은 결과(일부)

EMP_NAME	SALARY*12	(SALARY+(SALARY*BONUS_PCT))*12
한선기	108000000	129600000
강중훈	66000000	
최만식	43200000	
정도연	31200000	
안석규	42000000	52500000
조재형	45600000	
정지현	18000000	

컬럼 별칭을 사용한 결과(일부)

이름	1년 급여	총소득
한선기	108000000	129600000
강중훈	66000000	
최만식	43200000	
정도연	31200000	
안석규	42000000	52500000
조재형	45600000	
정지현	18000000	

3.2.4 SELECT 사용 - 컬럼 별칭 구문

```
SELECT * | { [DISTINCT] {{column_name | expr } {[AS] [ alias ]}, ...} }  
FROM   table_name  
WHERE  search_condition [ {AND | OR } , search_condition ...] ;
```

[구문 설명]

- 별칭을 사용하려는 대상 뒤에 'AS + 원하는 별칭'을 기술(별칭은 공백으로 구분)
- AS는 생략 가능
- 따옴표^{Quotation Mark}를 반드시 사용해야 하는 경우가 있음
 - 영문 대/소문자를 구분해서 별칭을 표시해야 하는 경우(기본적으로 대문자로 표시)
 - 별칭에 ^주특수 문자(공백, &, 괄호 등)가 포함된 경우

^주 별칭이 숫자로 시작하거나 숫자 자체인 경우 숫자도 특수 문자로 취급됨

3.2.4 SELECT 사용 예 4 - 컬럼 별칭 구문

```
SELECT EMP_NAME AS 이름,
       SALARY*12 AS "1년 급여",
       ( SALARY+( SALARY*BONUS_PCT ) ) *12 AS 총소득
FROM   EMPLOYEE;
```

결과(일부)

이름	1년 급여	총소득
한선기	108000000	129600000
강중훈	66000000	
최만식	43200000	
정도연	31200000	

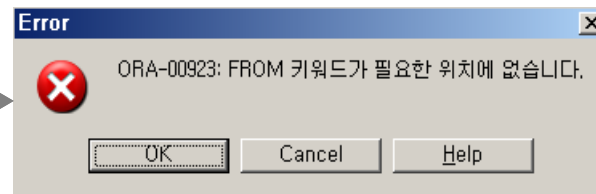
```
SELECT EMP_ID AS "empid",
       EMP_NAME AS "이름",
       SALARY AS "급여(원)"
FROM   EMPLOYEE;
```

결과(일부)

empid	이름	급여(원)
100	한선기	9000000
101	강중훈	5500000
102	최만식	3600000
103	정도연	2600000

```
SELECT EMP_NAME,
       SALARY AS 급여(원)
FROM   EMPLOYEE;
```

```
SELECT EMP_NAME,
       SALARY AS 1달급여
FROM   EMPLOYEE;
```



3.2.5 SELECT 사용 - 리터럴^{Literal}

- 임의로 지정한 문자열
- SELECT 절에 사용하면 테이블에 존재하는 데이터처럼 사용할 수 있다

```
SELECT EMP_ID,  
       EMP_NAME,  
       '재직' AS 근무여부  
FROM   EMPLOYEE ;
```

문자(또는 날짜) 리터럴은 ' ' 기호를 사용해서 표현

결과(일부)

EMP_ID	EMP_NAME	근무여부
100	한선기	재직
101	강중훈	재직
102	최만식	재직
103	정도연	재직
104	안석규	재직

리터럴은 Result Set의 모든 행에 반복적으로 표시됨

3.2.6 SELECT 사용 - DISTINCT

컬럼에 포함된 중복 값을 한번씩만 표시하고자 할 때 사용

```
SELECT * | { [DISTINCT] {{column_name | expr } [[AS] [ alias ]}, ...} }  
FROM   table_name  
WHERE  search_condition [ {AND | OR } , search_condition ...] ;
```

[구문 설명]

- SELECT 절에 1회만 기술
- 컬럼 별로 적용 불가능
- 여러 개 컬럼을 조회하는 경우에는 조회 대상 컬럼들의 조합 결과를 기준으로 중복 여부 판단

3.2.6 SELECT 사용 예 5 - DISTINCT

[EMPLOYEE 테이블(일부)]

DEPT_ID	JOB_ID	DEPT_ID
90	11	90
90	12	90
90	12	90
60	14	60
60	13	60
60	13	60
50	17	50
50	15	50
50	15	50
50	13	50
50	17	50
80	16	80
80	16	80
10	17	10
50	15	50
10	16	10
10	16	10
	17	
20	14	20
20	14	20
20	14	20

```
SELECT DISTINCT DEPT_ID
FROM EMPLOYEE;
```

DEPT_ID
50
20
10
90
80
60

```
SELECT DISTINCT JOB_ID, DEPT_ID
FROM EMPLOYEE;
```

JOB_ID	DEPT_ID
17	50
16	80
16	10
17	
14	60
12	90
14	20
11	90
15	50
13	50
17	10
13	60

3.2.7 SELECT 사용 예 6 - WHERE

```
SELECT EMP_NAME AS 이름,  
       DEPT_ID AS 부서  
FROM   EMPLOYEE  
WHERE  DEPT_ID = '90';
```

이름	부서
한선기	90
강중훈	90
최만식	90

DEPT_ID 값이 '90'인 행만
Result Set에 포함되었음

[급여가 4000000 보다 많은 직원 이름과 급여 조회]

```
SELECT EMP_NAME AS 이름,  
       SALARY AS 급여  
FROM   EMPLOYEE  
WHERE  SALARY > 4000000;
```

이름	급여
한선기	9000000
강중훈	5500000

SALARY 값이 4000000
보다 큰 행만 Result
Set에 포함되었음

3.2.7 SELECT 사용 예 6 - WHERE

[부서 코드가 '90'이고 급여를 2000000보다 많이 받는 부서원 이름과 부서 코드, 급여 조회]

```
SELECT EMP_NAME AS 이름,  
       DEPT_ID AS 부서,  
       SALARY AS 급여  
FROM   EMPLOYEE  
WHERE  DEPT_ID = '90'  
AND    SALARY > 2000000;
```

이름	부서	급여
한선기	90	9000000
강중훈	90	5500000
최만식	90	3600000

DEPT_ID 값이 '90'이고 SALARY 값이 2000000 보다 많은 행만 Result Set에 포함되었음

['90' 부서나 '20' 부서에 소속된 부서원 이름, 부서 코드, 급여 조회]

```
SELECT EMP_NAME AS 이름,  
       DEPT_ID AS 부서,  
       SALARY AS 급여  
FROM   EMPLOYEE  
WHERE  DEPT_ID = '90'  
OR     DEPT_ID = '20';
```

이름	부서	급여
한선기	90	9000000
강중훈	90	5500000
최만식	90	3600000
김술오	20	2500000
이중기	20	2500000
감우섭	20	2500000

DEPT_ID 값이 '90'이거나 '20'인 행만 Result Set에 포함되었음

3.3.1 연결 연산자 Concatenation Operator

연결 연산자 '||'를 사용하여 여러 컬럼을 하나의 컬럼인 것처럼 연결하거나, 컬럼과 리터럴을 연결할 수 있다

[컬럼과 컬럼을 연결한 경우]

```
SELECT EMP_ID||EMP_NAME||SALARY
FROM   EMPLOYEE;
```

결과(일부)

EMP_ID	EMP_NAME	SALARY
100	한선기	9000000
101	강중훈	5500000
102	최만식	3600000
103	정도연	2600000
104	안석규	3500000
107	조재형	3800000

[컬럼과 리터럴을 연결한 경우]

```
SELECT EMP_NAME||'의 월급은'||SALARY||'원 입니다.'
FROM   EMPLOYEE;
```

결과(일부)

EMP_NAME	의 월급은	SALARY
한선기	의 월급은	9000000원 입니다.
강중훈	의 월급은	5500000원 입니다.
최만식	의 월급은	3600000원 입니다.
정도연	의 월급은	2600000원 입니다.
안석규	의 월급은	3500000원 입니다.
조재형	의 월급은	3800000원 입니다.

3.3.2 논리 연산자Logical Operator

여러 개의 제한 조건 결과를 하나의 논리 결과(TRUE/FALSE/NULL)로 만들어준다

Operator	의미
AND	여러 조건이 동시에 TRUE 일 경우에만 TRUE 값 반환
OR	여러 조건들 중에 어느 하나의 조건만 TRUE이면 TRUE 값 반환
NOT	조건에 대한 반대 값으로 반환 (NULL 예외)

[AND 연산 결과]

	TRUE	FALSE	NULL
TRUE	T	F	N
FALSE	F	F	F
NULL	N	F	N

[OR 연산 결과]

	TRUE	FALSE	NULL
TRUE	T	T	T
FALSE	T	F	N
NULL	T	N	N

3.3.3 비교 연산자^{Comparison Operator}

- 표현식 사이의 관계를 비교하기 위해 사용
- 비교 결과는 논리 결과 중의 하나(TRUE/FALSE/NULL)가 된다
- 비교하는 두 컬럼 값/표현식은 서로 동일한 데이터 타입이어야 한다

[주요 비교 연산자]

Operator	의미
=	같다
> , <	크다/작다
>= , <=	크거나 같다/작거나 같다
<> , != , ^=	같지 않다
BETWEEN AND	특정 범위에 포함되는지 비교
LIKE/NOT LIKE	문자 패턴을 비교
IS NULL / IS NOT NULL	NULL 여부 비교
IN	비교 값 목록에 포함되는지 여부 비교

3.3.3 비교 연산자 - BETWEEN AND


비교하려는 값이 지정한 범위(상한 값과 하한 값의 경계 포함)에 포함되면 TRUE를 반환하는 연산자

[급여를 3,500,000원 보다 많이 받고 5,500,000원 보다 적게 받는 직원 이름과 급여 조회]

```
SELECT EMP_NAME,  
       SALARY  
FROM   EMPLOYEE  
WHERE  SALARY BETWEEN 3500000 AND 5500000;
```

또는

```
SELECT EMP_NAME,  
       SALARY  
FROM   EMPLOYEE  
WHERE  SALARY >= 3500000  
AND    SALARY <= 5500000;
```



EMP_NAME	SALARY
강중훈	5500000
최만식	3600000
안석규	3500000
조재형	3800000

3.3.3 비교 연산자 - LIKE

- 비교하려는 값이 지정한 특정 패턴^{Pattern}을 만족시키면 TRUE를 반환하는 연산자
- 패턴 지정에 위해 와일드 카드^{Wild Card} 사용
 - `%`Percentage : % 부분에는 임의 문자열(0개 이상의 임의의 문자)이 있다는 의미
 - `_`Underscore : _부분에는 문자 1개만 있다는 의미

['김'씨 성을 가진 직원 이름과 급여 조회]

```
SELECT EMP_NAME, SALARY
FROM   EMPLOYEE
WHERE  EMP_NAME LIKE '김%';
```

EMP_NAME	SALARY
김예수	2100000
김순이	3400000
김술오	2500000

[9000번 대 4자리 국번의 전화번호를 사용하는 직원 전화번호 조회]

```
SELECT EMP_NAME, PHONE
FROM   EMPLOYEE
WHERE  PHONE LIKE 주 ' _ _ _9 _ _ _ _ _ ' ;
```

EMP_NAME	PHONE
성해교	01992882295
염정하	01997545623

주 ' _ ' 사이에는 공백이 없음

3.3.3 비교 연산자 - LIKE

EMAIL ID 중 '_' 앞 자리가 3자리인 직원 조회

```
SELECT EMP_NAME,
       EMAIL
FROM   EMPLOYEE
WHERE  EMAIL LIKE '_ _ _ _%';
```

주 '_' 사이에는 공백 없음

EMP_NAME	EMAIL
한선기	sg_ahn@vcc.com
강중훈	jh_park@vcc.com
최만식	ms_choi@vcc.com
정도연	sy_kang@vcc.com
안석규	sg_han@vcc.com
조재형	jh_jo@vcc.com
정지현	jih_jeon@vcc.com
김예수	hs_kim@vcc.com
나승원	sw_cha@vcc.com
김순이	sm_kim@vcc.com
성해교	hg_song@vcc.com
전우성	ws_jeong@vcc.com
엄정하	jh_um@vcc.com
심하균	hk_shin@vcc.com
고승우	sw_jo@vcc.com
박하일	hi_park@vcc.com
권상후	sw_kwon@vcc.com
임영애	jangum_lee@vcc.com
염정하	jh_yeum@vcc.com
김술오	so_kim@vcc.com
이중기	jk_lee@vcc.com
감우섭	manofking@vcc.com

전체 데이터가
모두 조회되었음
→ 와 일드 카드
자체를 처리하는
방법이 필요함

3.3.3 비교 연산자 - LIKE

Escape Option : 와일드 카드('%' , '_') 자체를 데이터로 처리해야 하는 경우에 사용

[email id 중 '_' 앞 자리가 3자리인 직원 조회]

```
SELECT EMP_NAME,
       EMAIL
FROM   EMPLOYEE
WHERE  EMAIL LIKE '___\_%' ESCAPE '\';
```

주) '_' 사이에는 공백 없음

EMP_NAME	EMAIL
정지현	jih_jeon@vcc.com

ESCAPE OPTION에 사용하는 문자는 임의 지정 가능

```
SELECT EMP_NAME,
       EMAIL
FROM   EMPLOYEE
WHERE  EMAIL LIKE '___#_%' ESCAPE '#';
```

주) '_' 사이에는 공백 없음

EMP_NAME	EMAIL
정지현	jih_jeon@vcc.com

3.3.3 비교 연산자 - NOT LIKE

['김'씨 성이 아닌 직원 이름과 급여 조회]

```
SELECT EMP_NAME,
       SALARY
FROM   EMPLOYEE
WHERE  EMP_NAME NOT LIKE '김%';
```

또는

```
SELECT EMP_NAME,
       SALARY
FROM   EMPLOYEE
WHERE  NOT EMP_NAME LIKE '김%';
```

EMP_NAME	SALARY
한선기	9000000
강중훈	5500000
최만식	3600000
정도연	2600000
안석규	3500000
조재형	3800000
정지현	1500000
나승원	2300000
성해교	1900000
전우성	2090000
엄정하	2420000
심하균	2300000
고승우	1500000
박하일	2600000
권상후	3410000
임영애	2640000
염정하	1500000
이중기	2500000
감우섭	2500000

3.3.3 비교 연산자 - IS NULL / IS NOT NULL

NULL 여부를 비교하는 연산자

[관리자도 없고 부서 배치도 받지 않은 직원 이름 조회]

```
SELECT EMP_NAME, MGR_ID, DEPT_ID
FROM   EMPLOYEE
WHERE  MGR_ID IS NULL
AND    DEPT_ID IS NULL;
```

EMP_NAME	MGR_ID	DEPT_ID
심하균		
염정하		

[부서 배치를 받지 않았음에도 보너스를 지급받는 직원 이름 조회]

```
SELECT EMP_NAME, DEPT_ID, BONUS_PCT
FROM   EMPLOYEE
WHERE  DEPT_ID IS NULL
AND    BONUS_PCT IS NOT NULL;
```

EMP_NAME	DEPT_ID	BONUS_PCT
심하균		0.3

3.3.3 비교 연산자 - IN


비교하려는 값 목록에 일치하는 값이 있으면 TRUE를 반환하는 연산자

[60번 부서나 90번 부서원들의 이름, 부서 코드, 급여 조회]

```
SELECT EMP_NAME, DEPT_ID, SALARY
FROM   EMPLOYEE
WHERE  DEPT_ID IN ( '60', '90' );
```

또는

```
SELECT EMP_NAME, DEPT_ID, SALARY
FROM   EMPLOYEE
WHERE  DEPT_ID = '60'
OR     DEPT_ID = '90';
```



EMP_NAME	DEPT_ID	SALARY
한선기	90	9000000
강중훈	90	5500000
최만식	90	3600000
정도연	60	2600000
안석규	60	3500000
조재형	60	3800000

3.3.4 연산자 우선 순위


- 여러 연산자를 함께 사용할 때 우선 순위를 고려해야 함
- ()^{Parenthesis} 를 사용하면 연산자 우선 순위를 조절할 수 있다

1	산술 연산자
2	연결 연산자
3	비교 연산자
4	IS (NOT) NULL, LIKE, (NOT) IN
5	(NOT) BETWEEN-AND
6	논리 연산자 - NOT
7	논리 연산자 - AND
8	논리 연산자 - OR

3.3.4 연산자 우선 순위 예

20번 또는 90번 부서원 중 급여를 3000000원 보다 많이 받는 직원 이름, 급여, 부서코드 조회

```
SELECT EMP_NAME, SALARY, DEPT_ID
FROM   EMPLOYEE
WHERE  DEPT_ID = '20'
OR     DEPT_ID = '90'
AND    SALARY > 3000000;
```



EMP_NAME	SALARY	DEPT_ID
한선기	9000000	90
강중훈	5500000	90
최만식	3600000	90
김술오	2500000	20
이중기	2500000	20
감우섭	2500000	20


논리연산자 AND가 먼저 처리되어 "급여를 3000000원 보다 많이 받는 90번 부서원 또는 20번 부서원"을 조회하는 의미의 구문이 됨

3.3.4 연산자 우선 순위 예

20번 또는 90번 부서원 중 급여를 3000000원 보다 많이 받는 직원 이름, 급여, 부서코드 조회

논리연산자 OR가 먼저
처리되도록 ()를
이용하여 처리 순서를
변경

```
SELECT EMP_NAME, SALARY, DEPT_ID
FROM   EMPLOYEE
WHERE  ( DEPT_ID = '20'
OR      DEPT_ID = '90' )
AND    SALARY > 3000000;
```



EMP_NAME	SALARY	DEPT_ID
한선기	9000000	90
강중훈	5500000	90
최만식	3600000	90