5강 더 정확하고 다양하게 결과를 출력하는 WHERE절과 연산자

05-1 필요한 데이터만 쏙 출력하는 WHERE절

- **WHERE절?** SELECT문으로 데이터를 조회할 때 **특정 조건**을 기준으로 **원하는 행**을 출력하는데 사용한다.

```
SELECT * FROM EMP WHERE DEPTINO = 30;
```

- **=** 기호: 기호 양쪽의 대상이 **같은 값**을 가지고 있는지 검사하는 **비교 연산자**
- WHERE절을 사용한 SELECT문의 기본 형식

```
SELECT [조회할 열1 이름], [열2 이름], ..., [열N 이름]
FROM [조회할 테이블 이름]
WHERE [조회할 행을 선별하기 위한 조건식];
```

- WHERE절이 포함된 SELECT문을 실행하면 조회할 테이블의 **각 행에 WHERE절의 조건식을 대입**하여 결과가 '**참**'인 경우에만 출력된다.
  --> 각 행에서 원하는 열 값을 검사 후, 결과 값이 TRUE인 데이터만 출력한다.

05-2 여러 개 조건식을 사용하는 AND, OR 연산자

- WHERE절에서는 조건식을 여러 개 지정할 수 있는데, 이때 사용하는 것이 논리연산자 AND와 OR이다.

```
SELECT * FROM EMP
WHERE DEPTNO = 30
AND JOB = 'SALESMAN';
```

- WHERE절에서 비교하는 데이터가 **문자열**일 경우, 작은 따옴표(**''**)로 묶어 준다.
- 직접 **열**을 비교하는 **문자열 데이터**는 반드시 **대문자**로 작성해야 한다.
  -> SQL문에 사용하는 기본 형식은 대, 소문자를 구별하지 않고 사용할 수 있지만, 테이블

안에 들어 있는 문자 또는 문자열 데이터는 대, 소문자를 구별하기 때문이다.

- AND: 피연산자가 둘 다 TRUE -> 결과 값: TURE
- OR: 피연산자가 둘 다 또는 둘 중 하나 TRUE -> 결과 값: TRUE

WHERE절 조건식의 개수

- 제한 없음

실무에서의 AND, OR 연산자

- 실무에서 사용하는 SELECT문은 OR 연산자보다 AND 연산자를 더 많이 사용하는 경향이 있다.
  --> 다양한 조건을 한 번에 만족시키는 데이터만을 추출해야할 때가 많기 때문이다.

05-3 연산자 종류와 활용 방법 알아보기

산술 연산자

- \+, -, *, /
- SQL문에서 나머지 연산자는 제공되지 않지만, 오라클의 경우 MOD함수를 통해 나머지 연산 같은 기능을 사용할 수 있다.

```
SELECT * FROM EMP
WHERE SAL*12 = 36000;
```

비교 연산자

1. 대소 비교 연산자

- 연산자 앞 뒤에 있는 데이터 값을 **비교**하는데 사용한다.

```
SELECT * FROM EMP
WHERE SAL >= 3000;
```

- 대소비교연산자는 비교 대상인 데이터가 숫자가 아닌 **문자열**일 때도 사용할 수 있다.

```
SELECT * FROM EMP
WHERE ENAME >= 'F';
```

- 문자열을 비교할 때, 영어 사전처럼 알파벳 순서로 문자열의 대소를 비교한다.
- 예; ENAME >= 'F'는 ENAME 열 값의 첫 문자와 대문자 F를 비교했을 때 알파벳 순서상 F와 같거나 F보다 뒤에 있는 문자열을 출력하라는 의미이다.
- 비교 문자열이 문자 여러 개 일때도 가능하다.
- 자주 사용되는 내용은 아니다.

2. 등가 비교 연산자

- 연산자 양쪽 항목이 **같은 값**인지 검사하는 연산자이다.
- 양쪽 항목이 같으면 TRUE가 반환된다. (양쪽 값이 다를 때 TRUE가 반환될 수도 있음)
- 많은 프로그래밍 언어에서 = 기호는 대입의 의미로 사용되지만, SQL문에서는 본래 기호 의미 그대로 '**양쪽 데이터가 같은지 다른지**' **확인**하는데 사용된다.
- 등가 비교 연산자의 종류

Aa 연산자	사용법	의미
=	A=B	A값이 B값과 같을 경우 TRUE, 다를 경우 FALSE 반환
!=	A!=B	A값과 B값이 다를 경우 TRUE, 같을 경우 FALSE를 반환
<>	A<>B	
^=	A^=B	

```
SELECT * FROM EMP
WHERE SAL != 3000;
```

```
SELECT * FROM EMP
WHERE SAL <> 3000;
```

```
SELECT * FROM EMP
WHERE SAL ^= 3000;
```

- 실무세어는 !=와 <>를 많이 사용한다.

논리 부정 연산자 NOT

- NOT 연산자

```
SELECT * FROM EMP
WHERE NOT SAL =3000;
```

- 보통 NOT연산자를 **IN, BETWEEN, IS NULL 연산자와 함께** 복합적으로 사용하는 경우가 많다. (위의 예시와 같이 대소, 등가 비교 연산자에 직접 사요하는 경우는 별로 없다.)
- 복잡한 여러 개 조건식이 AND, OR로 묶여 있는 상태에서 정반대의 결과를 얻고자 할 때, 유용하게 사용할 수 있다.
- 복잡한 조건식에서 정반대의 최종 결과를 얻고자 할 때, NOT연산자로 한 번에 뒤집어서 SQL문 작성 시간을 줄일 수 있다.

IN 연산자

- 조건이 늘어날 수록 조건식을 많이 작성해야 하기 때문에 번거로운데, 이때 IN연산자를 사용하면 특정 열에 해당하는 조건을 여러 개 지정할 수 있다.
- **IN 연산자의 기본 형식**

```
SELECT [조회할 열1 이름], [열2 이름], ..., [열N 이름]
FROM [조회할 테이블 이름]
WHERE 열 이름 IN (데이터1, 데이터2, ..., 데이터N);
```

- **IN** 연산자가 **OR**로 데이터를 연결하는 것을 간단하게 바꿔준다고 생각하면 좋다.
- 다음의 예시와 같이 OR로 길게 늘어진 조건문을 IN 연산자가 간단히 바꿔준다.

```
//OR 사용
SELECT * FROM EMP
WHERE JOB = 'MANAGER'
```

```
OR JOB = 'SALESMAN'
OR JOB = 'CLERK';
```

```
//IN 사용
SELECT * FROM EMP
WHERE JOB IN ('MANAGER', 'SALESMAN', 'CLERK');
```

- IN 연산자 앞에 논리 부정 연산자 NOT을 사용하면 좀 더 간단하게 반대 경우를 조회할 수 있다. (물론 AND와 != 등의 연산을 이용해서도 가능)

```
//IN 사용
SELECT * FROM EMP
WHERE JOB NOT IN ('MANAGER', 'SALESMAN', 'CLERK');
```

BETWEEN A AND B 연산자

- 특정 열 값의 **최소, 최대 범위**를 지정하여 해당 범위 내의 데이터만 조회할 경우 사용된다.
- **BETWEEN A AND B 연산자**의 **기본 형식**

```
SELECT [조회할 열1 이름], [열2 이름], ..., [열N 이름]
FROM [조회할 테이블 이름]
WHERE 열 이름 BETWEEN 최솟값 AND 최댓값;
```

```
//대소 비교 연산자와 AND 사용
SELECT * FROM EMP
WHERE SAL >= 2000 AND SAL <= 3000;
```

```
//BETWEEN 사용
SELECT * FROM EMP
WHERE SAL BETWEEN 2000 AND 3000;
```

- NOT 연산자를 앞에 붙이면 범위 밖의 데이터 값을 출력할 수 있다.

```
//BETWEEN 사용
SELECT * FROM EMP
WHERE SAL NOT BETWEEN 2000 AND 3000;
```

LIKE 연산자와 와일드 카드

- LIKE 연산자는 이메일이나 게시판 제목 또는 **내용 검색 기능**처럼 **일부 문자열이 포함된 데이터를 조회**할 때 사용한다.
- LIKE 연산자 사용하여 출력하기

```
SELECT * FROM EMP
WHERE ENAME LIKE 'S%;
```

- ENAME LIKE 'S%' 조건식은 ENAME 열값이 ENAME 열 값이 대문자 S로 시작하는 데이터를 조회하라는 뜻이다.
- **와일드 카드?** 특정 문자 또는 문자열을 대체하거나 문자열 데이터의 패턴을 표기하는 특수 문자
- LIKE 연산자와 함께 사용할 수 있는 와일드 카드는 _와 %이다.
- **%**
  - 맨 앞이나 맨 뒤에 적어도 하나는 필요
  - 길이와 상관없이(문자 없는 경우도 포함) **모든 문자 데이터**를 의미
- _
  - 어떤 값이든 상관없이 **한 개**의 문자 데이터를 의미
- 두 번째 글자가 L인 사원 데이터 조회 -> '_L%' 이용
- 어떤 단어가 포함된 제목 또는 본문 검색과 같은 기능 구현을 위해서는 원하는 문자열 앞뒤 모두 와일드 카드(%)를 붙여 줄 수 있다. -> 특정 단어가 포함된 데이터

```
SELECT * FROM EMP
WHERE ENAME LIKE '%AM%;
```

- 특정 단어가 포함된 데이터를 제외한 결과를 얻고자 할 경우에는 LIKE 연산자 앞에 NOT을 붙여주면 된다.

```
SELECT * FROM EMP
WHERE ENAME NOT LIKE '%AM%;
```

와일드 카드 문자가 데이터 일부일 경우

- 데이터에 와일드 카드 기호로 사용되는 _나 % 문자가 데이터로 포함된 경우가 간혹 있는데, 이때는 ESCAPE절을 사용해준다.

```
SELECT * FROM SOME_TABLE
WHERE SOME_COLUMN LIKE 'A\\_A%' ESCAPE '\\';
```

A_A%에서 \ 문자 바로 뒤에 있는 _는 와일드 카드 기호가 아니라 데이터에 포함된 문자로 인식하라는 의미이므로, ESCAPE절에서 ESCAPE문자를 설정(여기서는 \)하여 데이터의 포함된 문자로 인식되게 할 수 있다.
(실무에서 잘 사용 안함)

IS NULL 연산자

- **NULL?** 데이터 값이 완전히 비어 있는 상태
- 이때, 숫자 0 값은 **값 0이 존재**한다는 뜻이므로 NULL과 혼동하지 않도록 주의해야한다.

Aa 의미	예
값이 존재하지 않음	통장을 개설한 적 없는 은행 조객의 계좌 번호
해당 사항 없음	미혼인 고객의 결혼 기념일
노출할 수 없는 값	고객 비밀번호 찾기 같은 열람을 제한해야하는 특정 개인 정보
확정되지 않은 값	미성년자의 출신 대학

- NULL ==> 현재 **무슨 값인지 확정되지 않은 상태**이거나 **값 자체가 존재하지 않은 상태**를 나타내는 데이터에 사용한다.
- NULL은 **산술 연산자와 비교 연산자로 비교**해도 **결과 값이 NULL**이 된다.
  ==> 수학에서 사용하는 무한대 또는 의문 부호인 물음표 같은 의미로 이해해도 무방하다.
- **특정 열 또는 연산의 결과 값**이 **NULL인지 여부를 확인**하려면 **IS NULL 연산자**를 사용해야 한다.

```
SELECT * FROM EMP WHERE COMM = NULL;
// 결과값 존재 X
```

```
SELECT * FROM EMP WHERE COMM IS NULL;
// 결과값 존재 O
```

- 반대의 경우, 즉 열 값이 **NULL이 아닌 데이터만 조회**하려면 **IS NOT NULLL**을 사용하면 된다.

```
SELECT * FROM EMP WHERE COMM IS NOT NULL;
```

- 데이터가 NULL인지 아닌지를 확인하는 용도로 사용하는 IS NULL과 IS NOT NULL 연산자는 **매우 자주 사용**되므로 사용법을 꼭 기억해야한다.

집합 연산자

- 관계형 데이터베이스 개념은 집합론에서 시작되었다.
- SQL문에서는 **SELECT문**을 통해 데이터를 조회한 결과를 **하나의 집합**과 같이 다룰 수 있는 **집합 연산자**를 사용할 수 있다.
- **두 개 이상의 SELECT문**의** 결과 값**을** 연결**할 때 사용한다.

```
SELECT EMPNO, ENAME, SAL, DEPTINO
FROM EMP
WHERE DEPTNO = 20;
```

- 주의해야 할 점: 집합 연산자로 두 개의 SELECT문의 결과 값을 연결할 때, **각 SELECT문**이 출력하려는 **열 개수**와 **열 자료형**이 **순서별로 일치**해야한다.
- 연결하려는 두 SELECT문의 열 개수와 자료형이 같다면, 서로 다른 테이블에서 조회하거나 조회하려는 열 이름이 다른 것은 문제가 되지는 않는다. (하지만 이상한 값이 나올 수는 있다.)
- **최종 출력**되는 **열 이름**은 **먼저 작성한 SELECT문의 열 이름**으로 표기된다.
- 오라클 데이터베이스에서 사용하는 집합 연산자는 다음과 같이 4가지 종류가 있다.

1. **UNION**
    a. 연결된 SELECT문의 결과 값을 **합집합**으로 묶어 준다.
    b. 결과 값의 **중복은 제거**된다.
2. **UNION ALL**
    a. 연결된 SELECT문의 결과 값을 **합집합**으로 묶어 준다.
    b. **중복된 결과** 값도 **제거 없이 모두 출력**된다.
3. **MINUS**
    a. 먼저 작성한 SELECT문의 결과 값에서 다음 SELECT문의 결과 값을 **차집합** 처리한다.
    b. 먼저 작성한 SELECT문의 결과 값 중, 다음 SELECT문에 **존재하지 않는 데이터**만 출력된다.
4. **INTERSECT**
    a. 먼저 작성한 SELECT문의 **결과 값이 같은 데이터만** 출력된다.
    b. **교집합**과 같은 의미이다.

- 연산자는 WHERE절 조건식에서 가장 많이 활용하지만, WHERE절 외에 SELECT절, HAVING절 및 여러 함수에서도 사용할 수 있다.

연산자 우선 순위

- 지금까지 WHERE절 조건식에서 사용한 여러 연산자는 **우선순위**(Priority)를 가지고 있다.
- 다음의 표에서 위에 있을 수록 우선순위가 높고 밑에 있을 수록 우선순위가 낮다.

연산자
• , /
=, !=, ^=, <>, >, >=, <=, <
IS (NOT) NULL, (NOT)LIKE, (NOT) IN
BETWEEN A AND B
NOT

AND

OR

- 수학식에서와 마찬가지로 소괄호()로 묶어주면 연산자의 기본 우선 순위와는 별개로 괄호 안의 연산식을 먼저 수행한다.

5강 연습문제

1. 사원 이름이 S로 끝나는 사원 데이터 모두 출력

```
SELECT *
FROM EMP
WHERE ENAME LIKE '%S';
```

2. 30번 부서에서 근무하고 있는 사원 중에 직책이 SALESMAN인 사원의 사원 번호, 이름, 직책, 급여, 부서 번호 출력

```
SELECT EMPNO, ENAME, JOB, SAL, EDPTNO
FROM EMP
WHERE JOB = 'SALESMAN';
```

3. 20번, 30번 부서에 근무하고 있는 사원 중 급여가 2000 초과인 사원을 집합 연산자를 사용하는 방식과 사용하지 않는 방식으로 출력

```
SELECT EMPNO, ENAME, JOB, SAL, EDPTNO
FROM EMP
WHERE DEPTNO IN (20, 30)
AND SAL >2000;
```

```
SELECT EMPNO, ENAME, JOB, SAL, EDPTNO
FROM EMP
WHERE DEPTNO IN (20, 30)
INTERSECT
SELECT EMPNO, ENAME, JOB, SAL, EDPTNO
FROM EMP
WHERE SAL > 2000;
```

4. NOT BETWEEN A AND B 연산자를 쓰지 않고, 급여 열 값이 2000 이상 3000 이하 범위의 값을 가진 데이터만 출력하도록 SQL 작성

```
SELECT *
FROM EMP
WHERE
NOT(SAL >= 2000 AND SAL <=3000);
```

5. 사원 이름에 E가 포함되어 있는 30번 부서의 사원 중 급여가 1000~2000 사이가 아닌 값 출력

```
SELECT EMPNO, ENAME, JOB, SAL, EDPTNO
FROM EMP
WHERE ENAME LIKE '%E%'
AND DEPTNO = 30
AND SAL NOT BETWEEN 1000 AND 2000;
```

6. 추가 수당이 존재하지 않고 상급자가 있고 직책이 MANAGER, CLERK인 사원 중에서 사원 이름의 두 번째가 L이 아닌 사원의 정보 출력

```
SELECT *
FROM EMP
WHERE MGR IS NOT NULL
AND JOB IN ('MANAGER', 'CLERK')
AND ENAME NOT LIKE "_L%';
```

프로그래머스 SELECT

1. 모든 레코드 조회하기

```
SELECT * FROM ANIMAL_INS ORDER BY ANIMAL_ID;
```

2. 역순 정렬하기

```
SELECT NAME, DATETIME FROM ANIMAL_INS ORDER BY ANIMAL_ID DESC;
```

3. 아픈 동물 찾기

```
SELECT ANIMAL_ID, NAME
FROM ANIMAL_INS
WHERE INTAKE_CONDITION =  'Sick' ORDER BY ANIMAL_ID;
```

4. 어린 동물 찾기

```
SELECT ANIMAL_ID, NAME
FROM ANIMAL_INS
WHERE INTAKE_CONDITION!='Aged' ORDER BY ANIMAL_ID;
```

5. 동물의 아이디와 이름

```
SELECT ANIMAL_ID, NAME FROM ANIMAL_INS ORDER BY ANIMAL_ID;
```

6. 여러 기준으로 정렬하기

```
SELECT ANIMAL_ID, NAME, DATETIME
FROM ANIMAL_INS
ORDER BY NAME, DATETIME DESC;
```

7. 상위 n개 레코드

```
SELECT NAME
FROM (SELECT * FROM ANIMAL_INS ORDER BY DATETIME)
WHERE ROWNUM = 1;
```