



6강 데이터처리와 가공을 위한 오라클 함수

06-1 오라클 함수

함수란?

- 함수(function)는 수학에서 정의한 개념으로, 변수 x , y 가 존재하고 x 값이 변하면 그 변화에 따라 어떤 연산 또는 가공을 거쳐 y 값도 함께 변할 때, 이 y 를 함수라고 한다.
- x 값의 변화에 따라 y 값이 종속적으로 변하기 때문에 '따름수'라고도 한다.
- 변수?
 - 수학; 변하는 수
 - 프로그래밍 언어; 명령에 따라 변할 수 있는 데이터를 저장하는 공간

오라클 함수의 종류

- 내장 함수(built-in function): 오라클에서 기본을 제공하고 있는 함수
- 사용자 정의 함수(user-defined function): 사용자가 필요에 의해 직접 정리하는 함수

내장 함수의 종류

- 입력 방식에 따라 데이터 처리에 사용하는 행이 나뉜다.
- **단일행 함수(single-row function)**: 데이터가 한 행씩 입력되고 입력된 한 행당 결과가 하나씩 나오는 함수
(한 행 -> 한 행)
- **다중행 함수(multiple-row function)**: 여러 행이 입력되어 하나의 행으로 결과가 반환되는 함수
(여러 행 -> 한 행)
- 단일행 함수와 다중행 함수는 다루는 자료형에 따라 조금 더 세분화 된다.

단일행 함수

열 1	열 2	...	열 N		열 1	열 2	...	열 N
행 1				→	행 1			
행 2				→	행 2			
...				→	...			
행 N				→	행 N			

다중행 함수

열 1	열 2	...	열 N		열 1	열 2	...	열 N
행 1				→	행 1			
행 2								
...								
행 N								

06-2 문자 데이터를 가공하는 문자 함수

- 문자함수는 문자 데이터를 가공하거나 문자 데이터로부터 특정 결과를 얻고자 할 때 사용하는 함수이다.
- 실무에서는 문자, 숫자, 날짜 데이터를 자주 사용한다.

대, 소문자를 바꿔주는 UPPER, LOWER, INITCAP 함수

함수	설명
UPPER (문자열)	괄호 안 문자 데이터를 모두 대문자로 변환하여 반환
LOWER (문자열)	괄호 안 문자 데이터를 모두 소문자로 변환하여 반환
INITCAP (문자열)	괄호 안 문자 데이터 중 첫 글자는 대문자로, 나머지 문자를 소문자로 변환 후 반환

- UPPER, LOWER, INITCAP 함수 사용하기

```
SELECT ENAME, UPPER(ENAME), LOWER(ENAME), INITCAP(ENAME)
FROM EMP;
```

- UPPER, LOWER, INITCAP 함수를 사용하려면, 입력 데이터에 **열 이름**이나 **데이터**를 직접 지정해야 한다.
- 예; 게시판의 글 제목이나 글 본문과 같이 가변 길이 문자열 데이터에서 특정 문자열을 포함하는 데이터를 조회하는 경우

```
// LIKE 연산자를 와일드 카드와 함께 사용 가능
SELECT *
FROM 게시판테이블
WHERE 게시판 제목 열 LIKE '%Oracle%'
OR 게시판 제목 열 LIKE '%Oracle%';
```

- 위의 예시의 경우, 검색하는 사람이 대소문자를 섞어서 검색할 경우, 원하는 결과 값을 얻기 어렵다.
- 따라서 이 경우에 조건식 양쪽 항목의 문자열을 **모두 대문자나 소문자로 바꿔서 비교**한다면, 실제 검색어의 대소문자 여부와 상관 없이 검색 단어와 일치한 문자열을 포함한 데이터를 찾을 수 있다.
- UPPER 함수로 문자열 비교하기(사원 이름이 SCOTT인 데이터 찾기)

```
SELECT *
FROM EMP
WHERE UPPER(ENAME) = UPPER('scott');
```

- UPPER 함수로 문자열 비교하기(사원 이름에 SCOTT 단어를 **포함한** 데이터 찾기)

```
SELECT *
FROM EMP
WHERE UPPER(ENAME) LIKE '%scott%';
```

- 찾으려는 문자열 데이터는 scott으로 소문자로 명시했지만, **양쪽 항목을 모두 UPPER 함수를 통해 대문자로 변환한 후 비교**했기 때문에 **대소문자 상관없이** SCOTT 데이터가 출력되는 것을 확인할 수 있다.

문자열 길이를 구하는 LENGTH 함수

- 특정 문자열 길이를 구할 때 LENGTH 함수를 사용한다
 - 선택한 열의 문자열 길이 구하기(각 행별 사원 이름이 몇 글자인지 출력하기)

```
SELECT ENAME, LENGTH(ENAME)
FROM EMP;
```

- WHERE절에서 LENGTH함수 사용하기
 - LENGTH함수를 WHERE절에 사용하면 문자열 길이를 비교하여 행을 선별하는 것도 가능하다.

```
SELECT ENAME, LENGTH(ENAME)
FROM EMP
WHERE LENGTH(ENAME) >= 5;
```

- 문자열 데이터 길이가 아닌 바이트 수를 반환하는 LENGTHB함수도 존재한다. (참고로 한 글은 한 문자당 2byte로 처리 된다.)

DUAL 테이블?

- DUAL 테이블은 오라클의 최고 권한 관리자 계정인 SYS 소유의 테이블로 SCOTT 계정도 사용할 수 있는 더미(dummy) 테이블이다.
- 임시 연산이나 함수의 단일 결과 값 확인 용도로 종종 사용된다.

문자열 일부를 추출하는 SUBSTR 함수

- 주민등록번호 중 생년월일 앞자리만 필요하거나 전화번호의 마지막 네 자리 숫자만 추출하는 경우와 같이 **문자열 중 일부를 추출할 때 SUBSTR 함수를 사용한다.**

Aa Title	≡ 함수	≡ 설명
Untitled	SUBSTR(문자열 데이터, 시작 위치, 추출길이)	문자열 데이터의 시작 위치부터 추출 길이 만큼 추출한다. 시작 위치가 음수일 경우 마지막 위치부터 거슬러 올라간 위치에서 다시 오른쪽으로 시작한다.
Untitled	SUBSTR(문자열 데이터, 시작 위치)	문자열 데이터의 시작 위치부터 문자열 데이터 끝까지 추출한다. 시작 위치가 음수일 경우, 마지막 위치부터 거슬러 올라간 위치에서 끝까지 추출한다.

- 인덱스는 **1부터** 시작한다.
- SUBSTR 함수를 사용하는 예

```
SELECT JOB
SUBSTR(JOB, 1, 2), // 1~2
SUBSTR(JOB, 3, 2), //3~4
SUBSTR(JOB, 5) // 5~끝까지
FROM EMP;
```

- SUBSTR 함수와 다른 함수 함께 사용하기
 - 다른 함수의 결과 값을 SUBSTR함수의 입력 값으로 사용할 수도 있다.

```
SELECT JOB
SUBSTR(JOB, -LENGTH(JOB)), 1~끝까지
SUBSTR(JOB, -LENGTH(JOB), 2), //1~2
SUBSTR(JOB, -3) // 뒤에서 3번째 ~ 끝까지
FROM EMP;
```

문자열 데이터 안에서 특정 문자 위치를 찾는 INSTR 함수

- 문자열 데이터 안에 특정 문자나 문자열이 어디에 포함되어 있는지를 알고자 할 때, INSTR 함수를 사용한다.
- INSTR 함수는 총 네 개의 입력 값을 지정할 수 있다.
- 최소 두 개의 입력 값, (원본 문자열 데이터와 원본 문자열 데이터에서 찾으려는 문자) 이렇게 두 가지는 반드시 지정해주어야 한다.

```
INSTR([대상 문자열 데이터(필수)],
[위치를 찾으려는 부분 문자(필수)],
[위치 찾기를 시작할 대상 문자열 데이터 위치(선택, 기본값은 1)],
[시작 위치에서 찾으려는 문자가 몇 번째로 등장하는지 지정(선택, 기본값은 1)])
```

- 특정 문자 위치 찾기

```
SELECT INSTR('HELLO, ORACLE!', 'L') AS INSTR_1, //3, 처음부터 검색
INSTR('HELLO, ORACLE!', 'L', 5) AS INSTR_2, //12
INSTR('HELLO, ORACLE!', 'L', 2, 2) AS INSTR_3 //4
FROM DUAL;
```

- 만약 찾으려는 문자가 문자열 데이터에 포함되어 있지 않다면, 위치 값이 없으므로 0을 반환한다.

==> INSTR 함수를 LIKE와 비슷한 용도로 사용할 수 있다.
(자주 사용하지 않지만 알아두면 좋다.)

```
// INSTR함수로 사원 이름에 문자 S가 있는 행 구하기
SELECT *
FROM EMP
WHERE INSTR(ENAME, 'S') > 0;
```

```
// LIKE 연산자로 사원 이름에 문자 S가 있는 행 구하기
SELECT *
FROM EMP
WHERE ENAME LIKE '%S%';
```

특정 문자를 **다른 문자로 바꾸는 **REPLACE 함수

- REPLACE 함수는 특정 문자열 데이터에 포함될 문자를 다른 문자로 대체할 경우에 유용한 함수이다.
- REPLACE 함수의 기본 형식

```
REPLACE([문자열 데이터 또는 열 이름(필수)], [찾는 문자(필수)], [대체할 문자(선택)])
```

- 만약 대체할 문자를 **입력하지 않는다면** 찾는 문자로 지정한 문자는 문자열 데이터에서 **삭제** 된다.

```
SELECT '010-1234-5678' AS REPLACE_BEFORE,
// -가 공백으로 교체되어 출력
REPLACE('010-1234-5678', '-', ' ') AS REPLACE_1,
// -가 아예 삭제되어 숫자가 붙어서 나타남
REPLACE('010-1234-5678', '-') AS REPLACE_2
FROM DUAL;
```

- REPLACE 함수는 **카드 번호나 주민 번호, 계좌 번호, 휴대전화 번호** 또는 2017-12-25나 13:59:23과 같이 **날짜나 시간**을 나타내는 데이터처럼 특정 문자가 중간중간 끼어 있는 데이터에서 **해당 문자를 없애 버리거나 다른 문자로 바꾸어 출력**할 때 종종 사용되므로 기억해두면 좋다.

데이터의 빈 공간을 특정 문자로 채우는 LPAD, RPAD 함수

- LPAD: Left Padding(왼쪽 패딩)
- RPAD: Right Padding(오른쪽 패딩)
- 데이터와 자릿수를 지정한 후 **데이터 길이가 지정한 자릿수보다 작을 경우에 나머지 공간을 특정 문자로 채우는 함수**
- LPAD는 남은 빈 공간 왼쪽에 채우고 RPAD는 오른쪽에 채운다.
- 기본 형식

```
LPAD([문자열 데이터 또는 열이름(필수)], [데이터의 자릿수(필수)], [빈 공간에 채울 문자(선택)])
RPAD([문자열 데이터 또는 열이름(필수)], [데이터의 자릿수(필수)], [빈 공간에 채울 문자(선택)])
```

- LPAD, RPAD 함수 이용하여 출력하기

```
SELECT 'Oracle',
       LPAD('Oracle', 10, '#') AS LPAD_1,
       RPAD('Oracle', 10, '*') AS RPAD_1,
       LPAD('Oracle', 10) AS LPAD_2,
       RPAD('Oracle', 10) AS RPAD_2
FROM DUAL;
```

질의 결과 x

SQL | 인출된 모든 행: 1(0.004초)

	'ORACLE'	LPAD_1	RPAD_1	LPAD_2	RPAD_2
1	Oracle	####Oracle	Oracle****	Oracle	Oracle

- **패딩처리?** 문자열 데이터의 특정 문자로의 채움
- 이 패딩처리는 **데이터의 일부만 노출해야 하는 개인정보**를 출력할 때 다음과 같이 사용되기도 한다.

```
SELECT
  RPAD('971225-', 14, '*') AS RPAD_JMNO,
  RPAD('010-1234-', 13, '*') AS RPAD_PHONE
FROM DUAL;
```

질의 결과 x		
SQL 인출된 모든 행: 1(0.003초)		
	RPAD_JMNO	RPAD_PHONE
1	971225-*****	010-1234-****

두 문자열 데이터를 합치는 CONCAT 함수

- CONCAT 함수: 두 개의 문자열 데이터를 하나의 데이터로 연결해주는 역할을 한다.
- 두 열 사이에 클론(:)을 넣고 연결하기

```
SELECT CONCAT(EMPNO, ENAME),
       CONCAT(EMPNO, CONCAT(':', ENAME))
  FROM EMP
 WHERE ENAME = 'SCOTT';
```

- 위의 예시와 같이 CONCAT을 사용한 결과 값을 다시 다른 CONCAT함수의 입력 값으로 사용하는 것도 가능하다.

문자열 데이터를 연결하는 || 연산자

- || 연산자는 CONCAT 함수와 유사하게 열이나 문자열을 연결한다. => **CONCAT 대신 || 사용 가능**

```
SELECT EMPNO || ENAME,
       EMPNO || ':' || ENAME
  FROM ...
```

특정 문자를 지우는 TRIM, LTRIM, RTRIM 함수

- TRIM, LTRIM, RTRIM 함수는 문자열 데이터 내에서 특정 문자를 지우기 위해 사용한다.
- 원본 문자열 데이터를 제외한 나머지 데이터는 모두 생략할 수 있다.
- 삭제할 문자가 생략될 경우, 기본적으로 공백을 제거한다.

- 삭제 옵션
 - **LEADING**: 왼쪽에 있는 글자를 지운다.
 - **TRAILING**: 오른쪽에 있는 글자를 지운다.
 - **BOTH**: 양쪽의 글자를 모두 지운다.
- TRIM 함수의 기본 사용법

```
TRIM([삭제 옵션(선택)] [삭제할 문자(선택)])
FROM [원본 문자열 데이터(필수)]
```

- TRIM 함수 사용하기(삭제할 문자가 없을 때 -> 공백 제거)

```
SELECT '[' || TRIM('_' FROM '_ _Oracle_ _') || ']' AS TRIM,
 '[' || TRIM(LEADING '_' FROM '_ _Oracle_ _') || ']' AS TRIM_LEADING,
 '[' || TRIM(TRAILING '_' FROM '_ _Oracle_ _') || ']' AS TRIM_TRAILING,
 '[' || TRIM(BOTH '_' FROM '_ _Oracle_ _') || ']' AS TRIM_BOTH
FROM DUAL;
```

질의 결과 x

SQL | 인출된 모든 행: 1(0.004초)

	TRIM	TRIM_LEADING	TRIM_TRAILING	TRIM_BOTH
1	[_Oracle_]	[_Oracle_ _]	[_ _Oracle_]	[_Oracle_]

- TRIM 함수 사용하기(삭제할 문자 '_'가 있을 때)
각각 다른 위치의 _문자가 삭제된다.

```
SELECT '[' || TRIM('_' FROM '_ _Oracle_ _') || ']' AS TRIM,
 '[' || TRIM(LEADING '_' FROM '_ _Oracle_ _') || ']' AS TRIM_LEADING,
 '[' || TRIM(TRAILING '_' FROM '_ _Oracle_ _') || ']' AS TRIM_TRAILING,
 '[' || TRIM(BOTH '_' FROM '_ _Oracle_ _') || ']' AS TRIM_BOTH
FROM DUAL;
```

질의 결과 x				
SQL 인출된 모든 행: 1(0.003초)				
	TRIM	TRIM_LEADING	TRIM_TRAILING	TRIM_BOTH
1	[_Oracle_]	[_Oracle_ _]	[_ _Oracle_]	[_Oracle_]

- LTRIM, RTRIM 함수의 기본 사용법

- LTRIM, RTRIM 함수는 각각 왼쪽, 오른쪽 지정 문자를 삭제하는데 사용된다.
- TRIM 함수와 마찬가지로 삭제할 문자를 **지정하지 않을 경우, 공백 문자**가 삭제된다.
- TRIM 함수와 다른 점은 **삭제할 문자**를 하나만 지정하는 것이 아니라 **여러 문자 지정**이 가능하다는 것이다.
- 기본 형식

```
LTRIM([원본 문자열 데이터(필수)], [삭제할 문자 집합(선택)])
RTRIM([원본 문자열 데이터(필수)], [삭제할 문자 집합(선택)])
```

- LTRIM: 원본 문자열의 왼쪽에서 삭제할 문자열 지정(삭제할 문자열을 지정하지 않으면 공백이 삭제됨)
- RTRIM: 원본 문자열의 오른쪽에서 삭제할 문자열을 지정(삭제할 문자열을 지정하지 않으면 공백이 삭제됨)
- TRIM, LTRIM, RTRIM 사용하여 문자열 출력하기

```
SELECT '[' || TRIM(' _Oracle_ ') || ']' AS TRIM,
       '[' || LTRIM(' _Oracle_ ') || ']' AS LTRIM,
       '[' || LTRIM('<_Oracle_>', '<_') || ']' AS LTRIM2,
       '[' || LTRIM('<<_Oracle_>', '<_') || ']' AS LTRIM3,
       '[' || RTRIM(' _Oracle_ ') || ']' AS RTRIM,
       '[' || RTRIM('<_Oracle_>', '>_') || ']' AS RTRIM_2
FROM DUAL
```

질의 결과 x						
SQL 인출된 모든 행: 1(0,004초)						
	TRIM	LTRIM	LTRIM2	LTRIM3	RTRIM	RTRIM_2
1	[_Oracle_]	[_Oracle_]	[Oracle_>]	[Oracle_>]	[_Oracle_]	[<_Oracle]

- 삭제할 문자로 지정한 집합에서 **순서에 상관없이 이 집합 원소로 조합이 가능하면 전부 삭제** 된다.
- 조합이 불가능한 문자가 시작될 때 함수의 삭제 작업이 끝난다.
- 보통 실무에서 TRIM 함수는 검색 기준이 되는 데이터에서 **혹시나 들어 있을지도 모르는 양 쪽 끝의 공백을 제거**할 때 많이 사용된다.
 - 예; 유저가 로그인을 하려고 아이디를 입력했을 때 사용자가 실수로 스페이스 바를 눌러서 공백이 함께 입력되는 경우

06-3 숫자 데이터를 연산하고 수치를 조정하는 숫자 함수

- 숫자 데이터를 다루는 함수

Aa Title	함수	설명
Untitled	ROUND	지정된 숫자의 특정 위치에서 반올림 한 값을 반환
Untitled	TRUNC	지정된 숫자의 특정 위치에서 버림 한 값을 반환
Untitled	CEIL	지정된 숫자보다 큰 정수 중 가장 작은 정수 를 반환
Untitled	FLOOR	지정된 숫자보다 작은 정수 중 가장 큰 정수 를 반환
Untitled	MOD	지정된 숫자를 나눈 나머지 값 을 반환

- 특정 위치에서 반올림하는 ROUND 함수
 - 반올림 위치 지정 안하면 소수점 첫째 자리에서 반올림한 결과가 반환된다.
 - 기본 형식

```
ROUND([숫자(필수)], [반올림 위치(선택)])
```

- 반올림 위치에서 반올림 하는 것임

```
SELECT ROUND(1234.5678) AS ROUND,
ROUND(1234.5678, 0) AS ROUND_0,
ROUND(1234.5678, 1) AS ROUND_1, // 소수점 둘째자리에서 반올림
ROUND(1234.5678, 2) AS ROUND_2,
ROUND(1234.5678, -1) AS ROUND_MINUS1, // 실수 첫째자리에서 반올림
ROUND(1234.5678, -2) AS ROUND_MINUS2
FROM DUAL;
```

질의 결과 x

SQL | 인출된 모든 행: 1(0.005초)

	ROUND	ROUND_0	ROUND_1	ROUND_2	ROUND_MINUS1	ROUND_MINUS2
1	1234	1235	1234.6	1234.57	1230	1200

- 결국 반올림 위치가 0인 것과 반올림 위치를 지정하지 않은 반환값은 같은 결과를 출력한다.
- 1234.56789라는 부동소수를 가지고 인덱스를 정리해보자!

Aa 수	# 1	# 2	# 3	# 4	≡ .	# 5	# 6	# 7	# 8	# 9
인덱스	-4	-3	-2	-1	.	0	1	2	3	4

특정 위치에서 버리는 TRUNC 함수

- TRUNC 함수는 지정된 자리에서 숫자를 버림 처리하는 함수이다.
- 반올림 위치를 지정하지 않으면 소수점 첫째 자리에서 버림 처리 된다.
- 기본 형식

```
TRUNC([숫자(필수)], [버림 위치(선택)])
```

- TRUNC함수를 사용하여 숫자 출력하기

```
SELECT TRUNC(1234.5678) AS TRUNC,
TRUNC(1234.5678, 0) AS TRUNC_0,
```

```
TRUNC(1234.5678, 1) AS TRUNC_1,
TRUNC(1234.5678, 2) AS TRUNC_2,
TRUNC(1234.5678, -1) AS TRUNC_MINUS1,
TRUNC(1234.5678, -2) AS TRUNC_MINUS2
FROM DUAL;
```

지정한 숫자와 가까운 정수를 찾는 CEIL, FLOOR 함수

- CEIL 함수: 입력된 숫자와 가까운 가장 큰 정수 반환
 - 기본형식

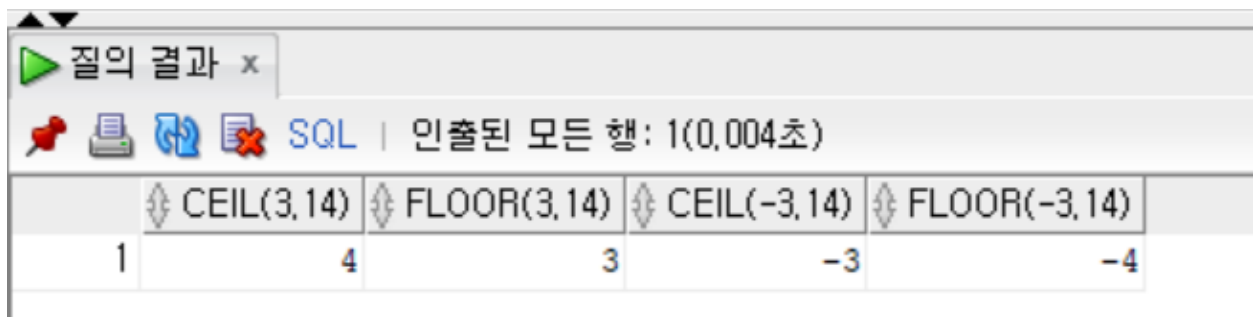
```
CEIL([숫자(필수)]);
```

- FLOOR 함수: 입력된 숫자와 가까운 가장 작은 정수 반환
 - 기본형식

```
FLOOR(필수)];
```

- CEIL, FLOOR 함수로 숫자 출력하기

```
SELECT CEIL(3.14),
FLOOR(3.14),
CEIL(-3.14),
FLOOR(-3.14)
FROM DUAL;
```



The screenshot shows a SQL query result window with the title '질의 결과 x'. The query executed is 'SQL | 인출된 모든 행: 1(0,004초)'. The result is displayed in a table with 4 columns: CEIL(3,14), FLOOR(3,14), CEIL(-3,14), and FLOOR(-3,14). The first row shows the results: 4, 3, -3, and -4.

	CEIL(3,14)	FLOOR(3,14)	CEIL(-3,14)	FLOOR(-3,14)
1	4	3	-3	-4

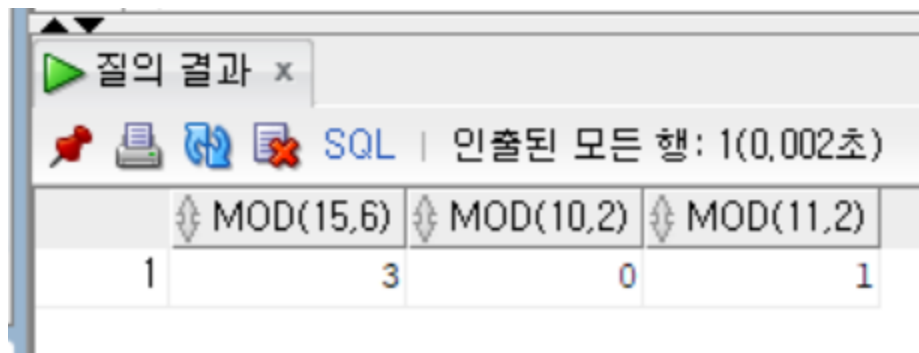
숫자를 나눈 나머지 값을 구하는 MOD 함수

- MOD 함수: 나머지를 구하는 함수

```
MOD([나눠셈 될 숫자(필수)], [나눌 숫자(필수)]);
```

- MOD 함수를 사용하여 나머지 값 출력하기

```
SELECT MOD(15, 6),  
MOD(10, 2),  
MOD(11, 2)  
FROM DUAL;
```



The screenshot shows a window titled '질의 결과' (Query Results) with a toolbar containing icons for execution, save, refresh, and print. Below the toolbar, it says 'SQL | 인출된 모든 행: 1(0.002초)'. The results are displayed in a table with three columns: 'MOD(15,6)', 'MOD(10,2)', and 'MOD(11,2)'. The first row shows the values 3, 0, and 1 respectively.

	MOD(15,6)	MOD(10,2)	MOD(11,2)
1	3	0	1

프로그래머스 SQL 문제 풀이

1. 동물 보호소에 들어온 동물 중, 이름이 없는 채로 들어온 동물의 ID를 조회하는 SQL 문을 작성해주세요. 단, ID는 오름차순 정렬되어야 합니다.

```
SELECT ANIMAL_ID FROM ANIMAL_INS  
WHERE NAME IS NULL;
```

1. 동물 보호소에 들어온 동물 중, 이름이 있는 동물의 ID를 조회하는 SQL 문을 작성해주세요. 단, ID는 오름차순 정렬되어야 합니다.

```
SELECT ANIMAL_ID  
FROM ANIMAL_INS  
WHERE NAME IS NOT NULL  
ORDER BY ANIMAL_ID;
```

1. 입양 게시판에 동물 정보를 게시하려 합니다. 동물의 생물 종, 이름, 성별 및 중성화 여부를 아이디 순으로 조회하는 SQL문을 작성해주세요. 이때 프로그래밍을 모르는 사람들은 NULL이라는 기호를 모르기 때문에, 이름이 없는 동물의 이름은 "No name"으로 표시해 주세요.

(**NVL** 함수 이용)

```
SELECT ANIMAL_TYPE, NVL(NAME, 'No name'), SEX_UPON_INTAKE  
FROM ANIMAL_INS  
ORDER BY ANIMAL_ID;
```