🧡

# 20220721 박지민

| | |
|---|---|
| 활동 이름 | 6강 데이터 처리와 가공을 위한 오라클 함수 |
| 속성 | |
| 속성 1 | |

## 06-1 오라클 함수

### 함수란?

- 함수(function)는 수학에서 정의한 개념으로, 변수 x, y가 존재하고 x값이 변하면 그 변화에 따라 어떤 연산 또는 가공을 거쳐 y 값도 함께 변할 때, 이 y를 함수라고 한다.
- x값의 변화에 따라 y값이 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 functino): 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 해이 입력되어 하나의 행으로 결과가 반환되는 함수
  (여러 행 -> 한 행)
- 단일행 함수와 다중행 함수는 다루는 자료형에 따라 조금 더 세분화 된다.

# 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 대이터를 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, UPER(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 출력되

는 것을 확인할 수 있다.

## 문자열 길이를 구하는 LENGTH 함수

- 특정 문자열 길이를 구할 때 LEGNTH 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 LEGNTHB함수도 존재한다. (참고로 한글은 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 추출**할 때 SUBSTR 함수를 사용한다.

| Aa Title | 함수 | 설명 |
|---|---|---|
| Untitled | **SUBSTR(문자열 데이터, 시작 위치, 추출길이)** | 문자열 데이터의 **시작 위치부터 추출 길이 만큼 추출**한다.<br>시작 위치가 음수일 경우 마지막 위치부터 거슬러 올라간 위치에서 다시 오른쪽으로 시작한다. |

| Aa Title | 함수 | 설명 |
| --- | --- | --- |
| Untitled | **SUBSTR(문자열 데이터, 시작 위치)** | 문자열 데이터의 **시작 위치부터 문자열 데이터 끝까지 추출**한다.<br>시작 위치가 음수일 경우, 마지막 위치부터 거슬러 올라간 위치에서 끝까지 추출한다. |

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

```
SELECT JOB
SUBSRT(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
SUBSRT(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 포함**되어 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 사용하진 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 경우**에 **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 디음과 같이 사용되기도 한다.

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

## 두 문자열 데이터를 합치는 CONCAT 함수

- CONCAT 함수: **두 개의 문자열 데이터**를 **하나의 데이터로 연결**해주는 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 CONACAT함수의 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 생략**할 수 있다.
- 삭제할 문자가 생략될 경우, 기본적으로 공백을 제거한다.
- 삭제 옵션
  - **LEADING**: **왼쪽**에 있는 글자를 지운다.
  - **TRAILING**: **오른쪽**에 있는 글자를 지운다.
  - **BOTH**: **양쪽**의 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, RTIRM 함수는 각각 왼쪽, 오른쪽 지정 문자를 삭제하는데 사용된다.
- TRIM함수와 마찬가지로 삭제할 문자를 **지정하지 않을** 경우, **공백 문자**가 삭제된다.
- TRIM 함수와 다른 점은 **삭제할 문자**를 하나만 지정하는 것이 아니라 **여러 문자 지정이 가능**하다는 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 ×

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 가능**하면 **전부 삭제** 된다.
- 조합이 불가능한 문자가 시작될 때 함수의 삭제 작업이 끝난다.
- 보통 실무에서 TRIM 함수는 검색 기준이 되는 데이터에서 **혹시나 들어 있을지도 모르는 양쪽 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 위치**에서** 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 0**인 것과 **반올림 위치를 지정하지 않은** 반환값은 **같은 결과**를 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 처리**하는 함수이다.
- 반올림 위치를 **지정하지 않으**면 **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

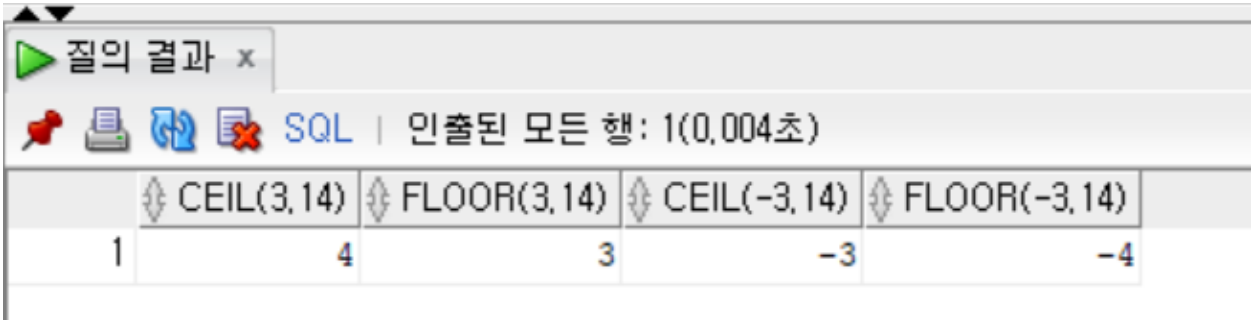
```
CEIL([숫자(필수)]);
```

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

```
FLOOR([숫자(필수)]);
```

- CEIL, FLOOR 함수로 숫자 출력하기

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




| | CEIL(3.14) | FLOOR(3.14) | CEIL(-3.14) | FLOOR(-3.14) |
|---|---|---|---|---|
| 1 | 4 | 3 | -3 | -4 |

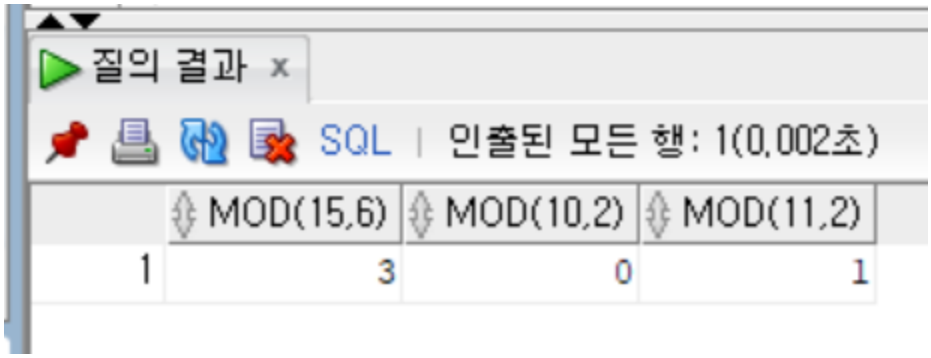
## 숫자를 나눈 나머지 값을 구하는 MOD 함수

- MOD 함수: 나머지를 구하는 함수

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

- MOD 함수를 사용하여 나머지 값 출력하기

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




# 06-4 날짜 데이터를 다루는 날짜 함수

- 오라클에서 **날짜 데이터, DATE**형 데이터는 다음과 같은 간단한 연산이 가능하다.

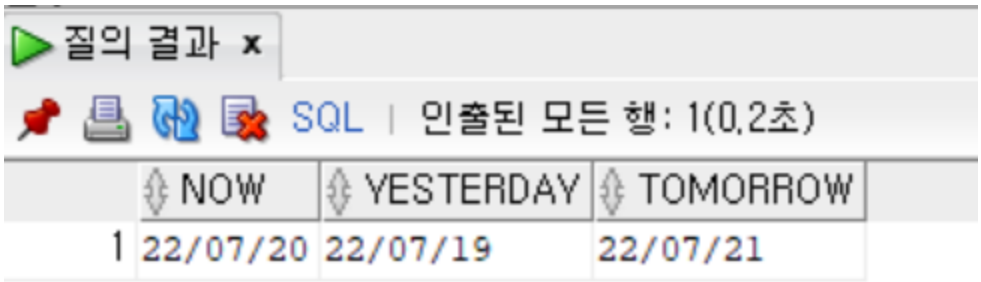
| Aa Title | 연산 | 설명 |
|---|---|---|
| Untitled | 날짜 데이터 + 숫자 | 날짜 데이터보다 숫자만큼 일수 이후의 날짜 |
| Untitled | 날짜 데이터 - 숫자 | 날짜 데이터보다 숫자만큼의 일수 이전의 날짜 |
| Untitled | 날짜 데이터 - 날짜 데이터 | 두 날짜 데티어 간의 일수 차이 |
| Untitled | 날짜 데이터 **+** 날짜 데이터 | **연산 불가**, **지원**하지 **않음** |

## SYSDATE 함수

- 오라클에서 제공하는 **날짜 함수** 중 가장 대표 함수

- 별 다른 입력 데이터 없이, 오라클 데이터베이스 서버가 놓인 OS(Operating System)의 **현재 날짜와 시간**을 보여 준다.
- SYSDATE 함수를 사용하여 날짜 출력하기

```
SELECT SYSDATE AS NOW,
    SYSDATE - 1 AS YESTERDAY,
    SYSDATE +1 AS TOMORROW
FROM DUAL;
```




## 몇 개월 이후 날짜를 구하는 ADD_MONTHS 함수

- ADD_MONTHS 함수: 특정 날짜에 지정한 개월 수 이후 날짜 데이터를 반환하는 함수
- 기본 형식

```
ADD_MONTHS([날짜 데이터(필수)], [더할 개월수(정수)(필수)]
```

- SYSDATE와 ADD_MONTHS 함수로 3개월 후 날짜 구하기

```
SELECT SYSDATE,
    ADD_MONTHS(SYSDATE, 3)
    FROM DUAL;
```

질의 결과

SQL | 인출된 모든 행: 1(0.006초)

| | SYSDATE | ADD_MONTHS(SYSDATE,3) |
|---|---|---|
| 1 | 22/07/20 | 22/10/20 |

- ADD_MONTHS 함수는 은근 자주 사용되는 함수이다.
  -> 윤년 등의 이유로 복잡해질 수 있는 날짜 계산을 간단하게 만들어주기 때문이다.
- 예; EMP 테이블에서 사원이 입사한지 **10주년**이 되는 **날짜**를 구하고 싶다면,
  **ADD_MONTHS** 함수에 **120개월**, 즉 **10년만큼의 개월 수를 입력하여 날짜를 구하는 것**이 가능하다.

```
SELECT EMPNO, ENAME, HIREDATE,
    ADD_MONTHS(HIREDATE, 120) AS WORK10YEAR
FROM EMP;
```

질의 결과

SQL | 인출된 모든 행: 14(0.01초)

| | EMPNO | ENAME | HIREDATE | WORK10YEAR |
|---|---|---|---|---|
| 1 | 7369 | SMITH | 80/12/17 | 90/12/17 |
| 2 | 7499 | ALLEN | 81/02/20 | 91/02/20 |
| 3 | 7521 | WARD | 81/02/22 | 91/02/22 |
| 4 | 7566 | JONES | 81/04/02 | 91/04/02 |
| 5 | 7654 | MARTIN | 81/09/28 | 91/09/28 |
| 6 | 7698 | BLAKE | 81/05/01 | 91/05/01 |
| 7 | 7782 | CLARK | 81/06/09 | 91/06/09 |
| 8 | 7788 | SCOTT | 87/04/19 | 97/04/19 |
| 9 | 7839 | KING | 81/11/17 | 91/11/17 |

- 만약 입사한지 36년이 되지 않은 사원을 출력하고자 한다면, ADD_MONTHS함수와 WHERE절을 동시에 사용하는 것도 가능하다.

```
SELECT EMPNO, ENAME, HIREDATE, SYSDATE
FROM EMP
WHERE ADD_MONTHS(HIREDATE, 36*12) > SYSDATE;
```

질의 결과 x

SQL | 인출된 모든 행: 2(0.003초)

| | EMPNO | ENAME | HIREDATE | SYSDATE |
|---|---|---|---|---|
| 1 | 7788 | SCOTT | 87/04/19 | 22/07/20 |
| 2 | 7876 | ADAMS | 87/05/23 | 22/07/20 |

## 두 날짜 간의 개월 수 차이를 구하는 MONTHS_BETWEEN 함수

- MONTHS_BETWEEN 함수는 두 개의 날짜 데이터를 입력하고 두 날짜 간의 개월 수 차이를 구하는 데 사용한다.
- 기본 형식

```
MONTHS_BETWEEN([날짜 데이터(필수)], [날짜 데이터2(필수)])
```

- HIREDATE와 SYSDATE 사이의 개월 수를 MONTHS_BETWEEN 함수로 출력하기

```
SELECT EMPNO, ENAME, HIREDATE, SYSDATE,
    MONTHS_BETWEEN(HIREDATE, SYSDATE) AS MONTH1,
    MONTHS_BETWEEN(SYSDATE, HIREDATE) AS MONTH2,
    TRUNC(MONTHS_BETWEEN(SYSDATE, HIREDATE)) AS MONTH3
FROM EMP;
```

질의... ×

SQL | 인출된 모든 행: 14(0.005초)

| | EMPNO | ENAME | HIREDATE | SYSDATE | MONTH1 | MONTH2 | MONTH3 |
|---|---|---|---|---|---|---|---|
| 1 | 7369 | SMITH | 80/12/17 | 22/07/20 | -499.10897065412186379928315412186379928 3 | 499.108970654121863799283154121863799283 | 499 |
| 2 | 7499 | ALLEN | 81/02/20 | 22/07/20 | -497 | 497 | 497 |
| 3 | 7521 | WARD | 81/02/22 | 22/07/20 | -496.947680331541218637992831541218637993 | 496.947680331541218637992831541218637993 | 496 |
| 4 | 7566 | JONES | 81/04/02 | 22/07/20 | -495.592841621863799283154121863799283154 | 495.592841621863799283154121863799283154 | 495 |
| 5 | 7654 | MARTIN | 81/09/28 | 22/07/20 | -489.754131944444444444444444444444444444 | 489.754131944444444444444444444444444444 | 489 |
| 6 | 7698 | BLAKE | 81/05/01 | 22/07/20 | -494.625099686379928315412186379928315412 | 494.625099686379928315412186379928315412 | 494 |
| 7 | 7782 | CLARK | 81/06/09 | 22/07/20 | -493.367035170250896057347670250896057348 | 493.367035170250896057347670250896057348 | 493 |
| 8 | 7788 | SCOTT | 87/04/19 | 22/07/20 | -423.044454525089605734767025089605734767 | 423.044454525089605734767025089605734767 | 423 |
| 9 | 7839 | KING | 81/11/17 | 22/07/20 | -488.108970654121863799283154121863799283 | 488.108970654121863799283154121863799283 | 488 |

- 비교 **날짜의 입력 위치**에 따라 **음수 또는 양수**가 나올 수 있다.
- 개월 수 차이는 **소수점 단위**까지 결과가 나오므로 **TRUNC** 함수(버림 함수)를 조합하면 개월 수 차이를 **정수로** 출력할 수 있다.

## 돌아오는 요일, 달의 마지막 날짜를 구하는 NEXT_DAY, LAST_DAY 함수

### - NEXT_DAY

- NEXT_DAY 함수는 날짜 데이터와 요일 문자열을 입력하고,
  입력한 날짜 데이터에서 돌아오는 요일의 날짜를 반환한다.
- NEXT_DAY 함수의 기본 사용법

```
NEXT_DAY([날짜 데이터(필수)], [요알 문자(필수)])
```

### - LAST_DAY

- LAST_DAY 함수는 하나의 날짜 데이터만으로 데이터로 사용하며 **해당 날짜가 속한 달의 마지막 날짜**를 반환해주는 함수이다.
- LAST_DAY 함수의 기본 사용법

```
LAST_DAY([날짜 데이터(필수)])
```

- NEXT_DAY, LAST_DAY 함수를 사용하여 출력하기

```
SELECT SYSDATE,
    NEXT_DAY(SYSDATE, '월요일'),
    LAST_DAY(SYSDATE)
FROM DUAL;
```

질의 결과 x

SQL | 인출된 모든 행: 1(0.007초)

| | SYSDATE | NEXT_DAY(SYSDATE,'월요일') | LAST_DAY(SYSDATE) |
|---|---|---|---|
| 1 | 22/07/20 | 22/07/25 | 22/07/31 |

## 날짜의 반올림, 버림을 하는 ROUND, TRUNC 함수

- ROUND, TRUNC함수는 숫자 데이터뿐만 아니라, 날짜 데이터를 입력데이터로 사용할 수도 있다.
- 소수점 위치 정보를 입력하지 않고 반올림, 버림의 기준이 될 포맷(FORMAT)값을 지정해 준다.

| Aa 입력 데이터 종류 | 사용 방식 |
|---|---|
| 숫자 데이터 | ROUND([숫자(필수)], [반올림 위치]) |
| Untitled | TRUNC([숫자(필수)], [버림 위치] |
| 날짜 데이터 | ROUND([날짜데이터(필수)], [반올림 기준 포맷]) |
| Untitled | TRUNC([날짜데이터(필수)], [버림 기준 포맷]) |

- 날짜 기준 포맷의 종류가 많기도 하고 TRUNC, ROUND 함수의 사용은 날짜를 기준으로 삼아야하는 일부 업무에서 제한적으로 사용 되기 때문에 필요할 때 찾아볼 수 있을 정도로만 기억해도 좋다.
- **ROUND 함수를 이용한 반올림과 TRUNC함수를 이용한 버림이 날짜 데이터에도 적용 가능하다는 것은 꼭 기억하자!**
- ROUND 함수 사용하여 날짜 데이터 출력하기

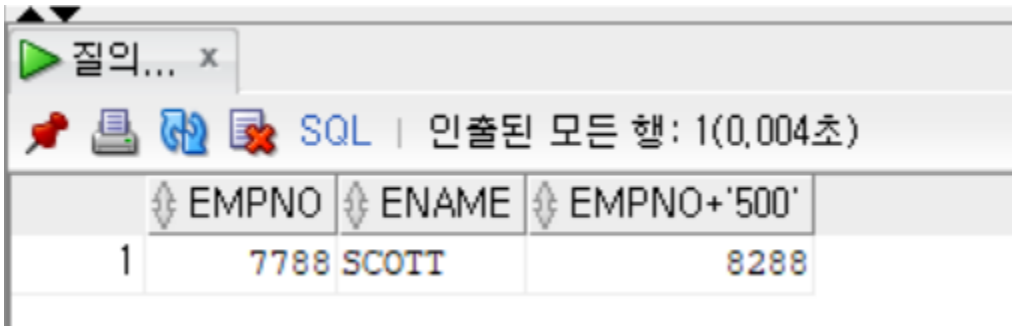
```
SELECT SYSDATE,
    ROUND(SYSDATE, 'CC') AS FORMAT_CC,
    ROUND(SYSDATE, 'YYYY') AS FORMAT_YYYY,
    ROUND(SYSDATE, 'Q') AS FORMAT_Q,
    ROUND(SYSDATE, 'DDD') AS FORMAT_DDD,
    ROUND(SYSDATE, 'HH') AS FORMAT_HH
FROM DUAL;
```

```
SELECT SYSDATE,
    TRUNC(SYSDATE, 'CC') AS FORMAT_CC,
    TRUNC(SYSDATE, 'YYYY') AS FORMAT_YYYY,
    TRUNC(SYSDATE, 'Q') AS FORMAT_Q,
    TRUNC(SYSDATE, 'DDD') AS FORMAT_DDD,
    TRUNC(SYSDATE, 'HH') AS FORMAT_HH
FROM DUAL;
```

## 06-5 자료형을 변환하는 형 변환 함수

- 오라클에서는 저장할 데이터 종류, 즉 자료형을 다양하게 제공한다.
- **형 변환 함수?** 각 데이터에 지정된 **자료형을 바꿔 주는** 함수
- 숫자와 문자열(숫자)을 더하여 출력하기

```
SELECT EMPNO, ENAME, EMPNO + '500'
FROM EMP
WHERE ENAME = 'SCOTT';
```




| | EMPNO | ENAME | EMPNO+'500' |
|---|---|---|---|
| 1 | 7788 | SCOTT | 8288 |

- 위의 예시에서는 문자열 500이 자동으로 사원 번호에 더해졌는데, 이는 숫자로 인식 가능한 문자데이터가 자동으로 숫자로 바뀐 후 연산이 수행된 것이다.
- 자동형 변환 = 암시적 형 변환(implicit type conversion)
- 하지만 밑의 예시는 오류가 나게 된다.
- 문자열(문자)과 숫자를 더하여 출력하기

```
SELECT 'ABC' + EMPNO, EMPNO
FROM EMP
WHERE ENAME = 'SCOTT';
```

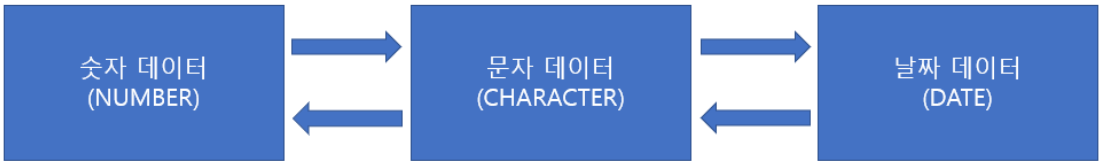
```
질의 결과 x
SQL | 실행 중:SELECT 'ABC' + EMPNO, EMPNOFROM EMPWHERE ENAME = 'SCOTT'(0초)
ORA-01722: 수치가 부적합합니다
01722, 00000 - "invalid number"
*Cause:    The specified number was invalid,
*Action:   Specify a valid number,
```

## - 명시적 형 변환? explicit type conversion, 자동으로 변환되는 방식이 아닌 사용자, 즉 우리가 자료형을 직접 지정 해주는 방식

- 형 변환 함수를 사용하여 자료형을 변환해주는 방식

| Aa 종류 | 설명 |
|---|---|
| TO_CHAR | 숫자 또는 날짜 데이터 -> 문자 데이터 |
| TO_NUMBER | 문자 데이터 -> 숫자 데이터 |
| TO_DATE | 문자 데이터 -> 날짜 데이터 |

- 문자를 중심으로 숫자 또는 날짜 데이터의 변환이 가능하다.




## 날짜, 숫자 데이터를 문자 데이터로 변환하는 TO_CHAR 함수

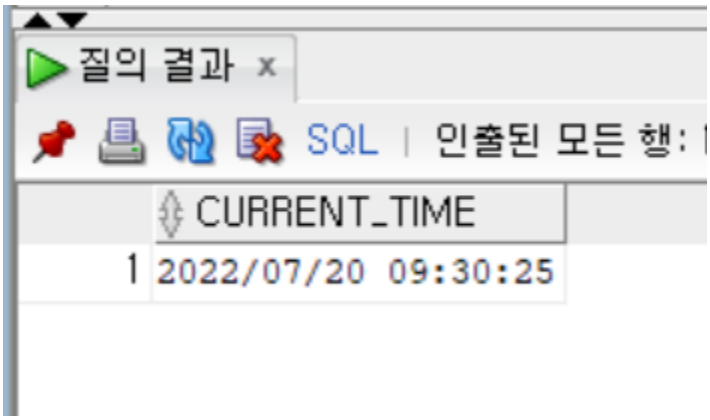
- TO_CHAR 함수는 날짜, 숫자 데이터를 문자 데이터로 변환해주는 함수이다.
- 날짜 데이터에서 문자 데이터로 변환하는데 많이 사용 된다.
- 기본 형식

```
TO_CHAR([날짜데이터(필수)], '[출력되기 원하는 문자 형태(필수)]')
```

## - 원하는 출력형태로 날짜 출력하기

- 현재 날짜와 시간을 '연/월/일 시:분:초' 형태로 출력하기

```
SELECT TO_CHAR(SYSDATE, 'YYYY/MM/DD HH24:MI:SS')
AS CURRENT_TIME
FROM DUAL;
```




- 'YYYY/MM/DD HH24:MI:SS'는 날짜 데이터를 '연/월/일 시:분:초'로 표현하기 위해 사용하는 형식,(format)이다.
- 월과 요일의 표기는 사용 언어에 따라 출력을 달리 할 수 있다.

```
SELECT SYSDATE,
    TO_CHAR(SYSDATE, 'MM') AS MM,
    TO_CHAR(SYSDATE, 'MON') AS MON,
    TO_CHAR(SYSDATE, 'MONTH') AS MONTH,
    TO_CHAR(SYSDATE, 'DD') AS DD,
    TO_CHAR(SYSDATE, 'DY') AS DY,
    TO_CHAR(SYSDATE, 'DAY') AS DAY
FROM DUAL;
```

질의 결과

SQL | 인출된 모든 행: 1(0.016초)

| | SYSDATE | MM | MON | MONTH | DD | DY | DAY |
|---|---|---|---|---|---|---|---|
| 1 | 22/07/20 | 07 | 7월 | 7월 | 20 | 수 | 수요일 |

## - 특정 언어에 맞춰서 출력하기

- 특정 언어에 맞는 월, 요일 이름으로 출력하고 싶을 때, 다음과 같이 기존 TO_CHAR 함수에 날짜 출력 언어를 추가로 지정해 줄 수 있다.

```
TO_CHAR([날짜 데이터(필수)],
'[출력되길 원하는 문자 형태(필수)]',
'NLS_DATE_LANGUAGE = language'(선택))
```

```
SELECT SYSDATE,
    TO_CHAR(SYSDATE, 'MON',
    'NLS_DATE_LANGUAGE = KOREAN') AS MON_KOR,
    TO_CHAR(SYSDATE, 'MON',
    'NLS_DATE_LANGUAGE = JAPANESE') AS JAPAN,
    TO_CHAR(SYSDATE, 'MON',
    'NLS_DATE_LANGUAGE = ENGLISH') AS ENGLISH,
    TO_CHAR(SYSDATE, 'MONTH',
    'NLS_DATE_LANGUAGE = KOREAN') AS MON_KOR,
    TO_CHAR(SYSDATE, 'MONTH',
    'NLS_DATE_LANGUAGE = JAPANESE') AS JAPAN,
    TO_CHAR(SYSDATE, 'MONTH',
    'NLS_DATE_LANGUAGE = ENGLISH') AS ENGLISH
FROM DUAL;
```

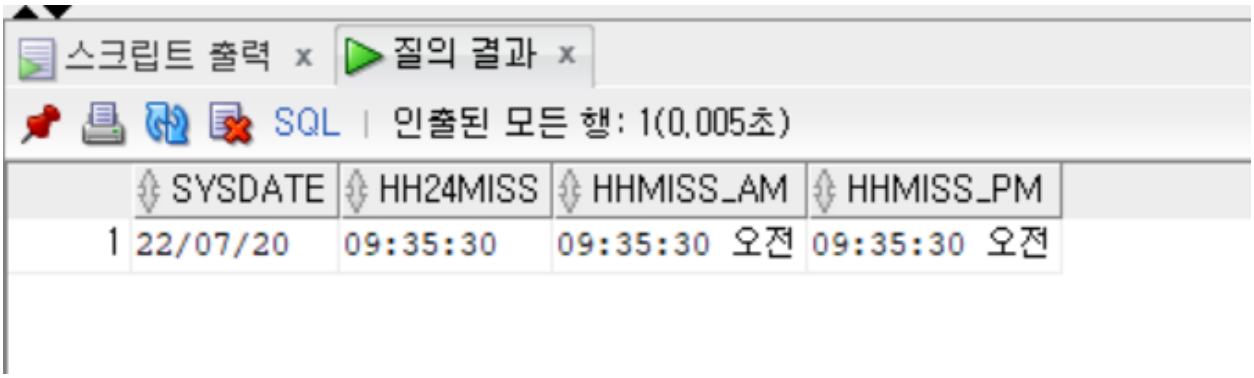
스크립트 출력 | 질의 결과

SQL | 인출된 모든 행: 1(0.009초)

| | SYSDATE | MON_KOR | JAPAN | ENGLISH | MON_KOR_1 | JAPAN_1 | ENGLISH_1 |
|---|---|---|---|---|---|---|---|
| 1 | 22/07/20 | 7월 | 7月 | JUL | 7월 | 7月 | JULY |

## - 시간 형식 지정하여 출력하기

```
SELECT SYSDATE,
    TO_CHAR(SYSDATE, 'HH24:MI:SS') AS HH24MISS,
    TO_CHAR(SYSDATE, 'HH12:MI:SS AM') AS HHMISS_AM,
    TO_CHAR(SYSDATE, 'HH:MI:SS P.M.') AS HHMISS_PM
    FROM DUAL;
```




| | SYSDATE | HH24MISS | HHMISS_AM | HHMISS_PM |
|---|---|---|---|---|
| 1 | 22/07/20 | 09:35:30 | 09:35:30 오전 | 09:35:30 오전 |

## - TO_CHAR 함수로 숫자데이터를 문자 데이터로 변환하는 방식은 자주 사용 X

```
SELECT SAL,
    TO_CHAR(SAL, '$999,999') AS SAL_$, //9는 숫자 한자리
    TO_CHAR(SAL, 'L999,999') AS SAL_L,
    TO_CHAR(SAL, '999,999.00') AS SAL_1,
    TO_CHAR(SAL, '000,999,999.00') AS SAL_2,
    TO_CHAR(SAL, '000999999.99') AS SAL_3,
    TO_CHAR(SAL, '999,999,00') AS SAL_4
FROM EMP;
```

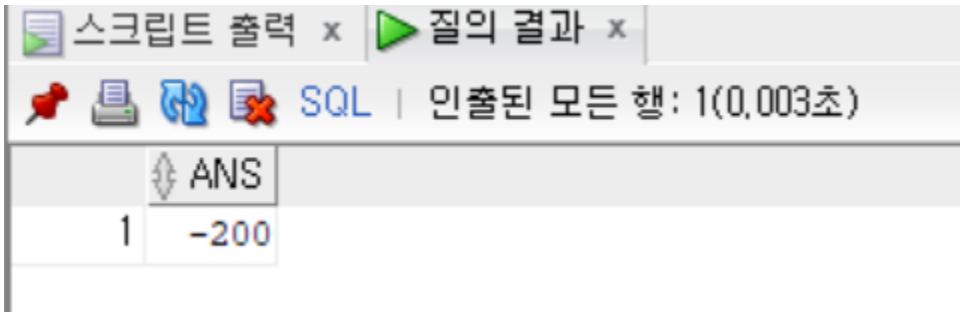
## 문자 데이터를 숫자 데이터로 변환하는 TO_NUMBER 함수

- 문자열을, 지정한 형태의 숫자로 인식하여 숫자 데이터로 변환한다.
- 기본형식

```
TO_NUMBER('[문자열 데이터(필수)]', [인식될 숫자 형태(필수)]')
```

- TO_NUMBER 함수로 연산하여 출력하기

```
SELECT TO_NUMBER('1,300', '999,999') -
TO_NUMBER('1,500', '999,999') AS ANS
FROM DUAL;
```



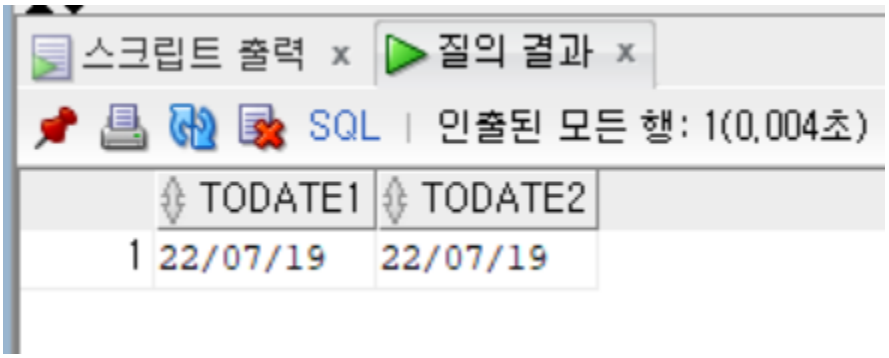

## 문자 데이터를 날짜 데이터로 변환하는 TO_DATE 함수

- TO_DATE 함수를 사용하면 문자열 데이터를 날짜 데이터로 바꿔줄 수 있다.
- 기본 형식

```
TO_DATE9('[문자열 데이터(필수)]', '[인식돌 날짜형태(필수)]')
```

- 2018-07-14, 20180714와 같은 문자 데이터를 날짜 데이터로 바꿔 주려면 다음과 같이 TO_DATE 함수를 사용하면 된다.

```
SELECT TO_DATE('2022-07-19', 'YYYY-MM-DD') AS TODATE1,
    TO_DATE('20220719', 'YYYY-MM-DD') AS TODATE2
FROM DUAL;
```



- 날짜 데이터끼리는 간단한 연산이 가능한데, 날짜 데이터는 상대적으로 **이전 날짜인 데이터가 이후 날짜 데이터보다 크기가 작은 데이터**로 여겨지기 때문이다.
- TO_DATE 함수와 비교 연산자를 사용하여 EMP 테이블에서 1981.6.1 이후 입사한 사원을 찾는 것도 가능하다.

```
SELECT * FROM EMP
WHERE TO_DATE('1981/06/01', 'YYYY-MM-DD') < HIREDATE;
```

스크립트 출력 x | 질의 결과 x

SQL | 인출된 모든 행: 9(0.006초)

| | EMPNO | ENAME | JOB | MGR | HIREDATE | SAL | COMM | DEPTNO |
|---|---|---|---|---|---|---|---|---|
| 1 | 7654 | MARTIN | SALESMAN | 7698 | 81/09/28 | 1250 | 1400 | 30 |
| 2 | 7782 | CLARK | MANAGER | 7839 | 81/06/09 | 2450 | (null) | 10 |
| 3 | 7788 | SCOTT | ANALYST | 7566 | 87/04/19 | 3000 | (null) | 20 |
| 4 | 7839 | KING | PRESIDENT | (null) | 81/11/17 | 5000 | (null) | 10 |
| 5 | 7844 | TURNER | SALESMAN | 7698 | 81/09/08 | 1500 | 0 | 30 |
| 6 | 7876 | ADAMS | CLERK | 7788 | 87/05/23 | 1100 | (null) | 20 |
| 7 | 7900 | JAMES | CLERK | 7698 | 81/12/03 | 950 | (null) | 30 |
| 8 | 7902 | FORD | ANALYST | 7566 | 81/12/03 | 3000 | (null) | 20 |
| 9 | 7934 | MILLER | CLERK | 7782 | 82/01/23 | 1300 | (null) | 10 |

- 날짜 데이터 형식을 지정할 때, YYYY, RRRR, YY, RR를 사용할 수 있는데, 네 자리로 표현하는 연도는 문제가 없지만, 두 자리로 연도를 표현하는 YY, RR은 사용상 주의를 기울여야 한다.

# 06-6 NULL 처리 함수

- 데이터가 NULL이면 산술 연산자나 비교 연산자가 우리가 예상한대로 동작하지 않는다.
- 특정 열의 데이터가 NULL일 경우에 연산 수행을 위해 데이터를 NULL이 아닌 다른 값으로 대체해 주어야 할 때 NVL과 NVL2함수를 사용한다.

## NVL 함수의 기본 사용법

- NVL 함수는 첫 번째 입력 데이터가 NULL이 아니라면 데이터를 그대로 반환하고, NULL이라면 두 번째 입력 데이터에 지정한 값을 반환한다.
- 기본 형식

```
NVL([NULL인지 여부를 검사할 데이터 또는 열(필수)],
  [앞의 데이터가 NULL일 경우 반환할 데이터](필수))
```

- NVL 함수를 사용하여 출력하기

```
SELECT EMPNO, ENAME, SAL, COMM, SAL+COMM,
    NVL(COMM, 0), SAL+NVL(COMM, 0)
    FROM EMP;
```

스크립트 출력 × 질의 결과 ×

SQL | 인출된 모든 행: 14(0.004초)

| | EMPNO | ENAME | SAL | COMM | SAL+COMM | NVL(COMM,0) | SAL+NVL(COMM,0) |
|---|---|---|---|---|---|---|---|
| 1 | 7369 | SMITH | 800 | (null) | (null) | 0 | 800 |
| 2 | 7499 | ALLEN | 1600 | 300 | 1900 | 300 | 1900 |
| 3 | 7521 | WARD | 1250 | 500 | 1750 | 500 | 1750 |
| 4 | 7566 | JONES | 2975 | (null) | (null) | 0 | 2975 |
| 5 | 7654 | MARTIN | 1250 | 1400 | 2650 | 1400 | 2650 |
| 6 | 7698 | BLAKE | 2850 | (null) | (null) | 0 | 2850 |
| 7 | 7782 | CLARK | 2450 | (null) | (null) | 0 | 2450 |
| 8 | 7788 | SCOTT | 3000 | (null) | (null) | 0 | 3000 |
| 9 | 7839 | KING | 5000 | (null) | (null) | 0 | 5000 |

- EMP 테이블의 급여 외 추가 수당을 의미하는 COMM 열 값이 NULL인 데이터를 0으로 대체하여 **연산이 가능**하다는 것을 확인할 수 있다.
- NVL 함수는 NULL 처리를 위해 자주 사용한다.

## NVL2 함수의 기본 사용법

- NVL2 함수는 열 도는 데이터를 입력하여 해당 데이터가 NULL이 아닐 때와 NULL일 때 출력 데이터를 각각 지정한다.
- 기본 형식

```
NVL2([NULL인지 여부를 검사할 데이터 또는 열(필수)],
   [앞 데이터가 NULL이 아닐 경우 반환할 데이터 또는 계산식(필수)],
   [앞 데이터가 NULL일 경우 반환할 데이터 또는 계산식(필수)])
```

- NVL2 함수를 사용하여 출력하기

```
SELECT EMPNO, ENAME, COMM,
    NVL2(COMM, 'O', 'X') AS OX,
    NVL2(COMM, SAL*12+COMM, SAL*12) AS ANNUAL
FROM EMP;
```

스크립트 출력 x | 질의 결과 x

SQL | 인출된 모든 행: 14(0.004초)

| | EMPNO | ENAME | COMM | OX | ANNUAL |
|---|---|---|---|---|---|
| 1 | 7369 | SMITH | (null) | X | 9600 |
| 2 | 7499 | ALLEN | 300 | O | 19500 |
| 3 | 7521 | WARD | 500 | O | 15500 |
| 4 | 7566 | JONES | (null) | X | 35700 |
| 5 | 7654 | MARTIN | 1400 | O | 16400 |
| 6 | 7698 | BLAKE | (null) | X | 34200 |
| 7 | 7782 | CLARK | (null) | X | 29400 |
| 8 | 7788 | SCOTT | (null) | X | 36000 |
| 9 | 7839 | KING | (null) | X | 60000 |

# 06-7 상황에 따라 다른 데이터를 반환하는 DECODE함수와 CASE문

- 특정 열 값이나 데이터 값에 따라 어떤 데이터를 반환할지 정할 때는 DECODE함수 또는 CASE문을 사용한다.

## DECODE 함수

- DECODE 함수는 **기준이 되는 데이터**를 **먼저 지정**한 후 **해당 데이터 값에 따라 다른 결과 값**을 내보내는 함수
- 기본 형식

```
D=DECODE([검사 대상이 될 열 또는 데이터 연산이나, 함수의 결과],
  [조건1], [데이터가 조건1과 일치할 때 반환할 결과],
    [조건2], [데이터가 조건2와 일치할 때 반환할 결과],
```

```
    ...
    [조건n], [데이터가 조건n와 일치할 때 반환할 결과],
    [위 조건1~조건n과 일치한 경우가 없을 때 반환할 결과])
```

- EMP 테이블에서 직책이 MANAGER인 사람은 급여의 10%를 인상한 급여, SALESMAN인 사람은 급여의 5%, ANALYST인 사람은 그대로, 나머지는 3%만큼 이상된 급여를 보고 싶을 때 다음과 같이 DECODE함수를 사용하여 작성할 수 있다.

```
SELECT EMPNO, ENAME, JOB, SAL,
    DECODE(JOB,
            'MANAGER', SAL*1.1,
            'SALESMAN', SAL*1.05,
            'ANALYST', SAL,
            SAL*1.03) AS UPSAL
        FROM EMP;
```

스크립트 출력 x | 질의 결과 x

SQL | 인출된 모든 행: 14(0.005초)

| | EMPNO | ENAME | JOB | SAL | UPSAL |
|---|---|---|---|---|---|
| 1 | 7369 | SMITH | CLERK | 800 | 824 |
| 2 | 7499 | ALLEN | SALESMAN | 1600 | 1680 |
| 3 | 7521 | WARD | SALESMAN | 1250 | 1312.5 |
| 4 | 7566 | JONES | MANAGER | 2975 | 3272.5 |
| 5 | 7654 | MARTIN | SALESMAN | 1250 | 1312.5 |
| 6 | 7698 | BLAKE | MANAGER | 2850 | 3135 |
| 7 | 7782 | CLARK | MANAGER | 2450 | 2695 |
| 8 | 7788 | SCOTT | ANALYST | 3000 | 3000 |
| 9 | 7839 | KING | PRESIDENT | 5000 | 5150 |

- DECODE 함수 역시 지금까지 다뤄 온 함수와 마찬가지로 **한 행에 데이터를 입력 받아 한 행으로 결과가 나오는 단일행 함수**이다.
- DECODE의 결과 또한 내용이 길어지더라도 **별칭 지정이 가능**하다.
- 만약 DECODE함수의 맨 마지막 데이터, 즉 조건에 해당하는 값이 없을 때, **반환값을 지정하지 않으면 NULL이 반환**된다.

## CASE문

- CASE문 또한 DECODE함수와 마찬가지로 특정 조건에 따라 반환할 데이터를 설정할 때 사용한다.
- DECODE함수는 기준 데이터를 반드시 명시하고 그 값에 따라 반환 데이터를 정한다.
- CASE문은 각 조건에 사용하는 데이터가 서로 상관 없어도 되고, 기준 데이터 값이 같은(=) 데이터 외에 다양한 조건을 사용할 수 있다.
- DECODE -> CASE 항상 가능, O
- CASE -> DECODE 안될 수도 있음 => CASE문의 범용성이 더 큰 것.
- CASE문의 기본 형식

```
CASE [검사 대상이 될 열 또는 데이터, 연산이나 함수의 결과(선택)]
  WHEN [조건1] THEN [조건1의 결과 값이 TRUE일 때, 반환할 결과]
    WHEN [조건2] THEN [조건2의 결과 값이 TRUE일 때, 반환할 결과]
    ...
    WHEN [조건n] THEN [조건n의 결과 값이 TRUE일 때, 반환할 결과]
    ELSE [위 조건1~n과 일치하는 경구가 없을 때 반환할 결과]
  END
```

- CASE문은 WHEN이나 THEN, ELSE를 사용하여 DECODE함수보다 더 프로그래밍 언어적인 표현 방식을 사용한다.

## - DECODE함수와 같은 방식으로 CASE문 사용하기

- EMP 테이블에서 직책이 MANAGER인 사람은 급여의 10%를 인상한 급여, SALESMAN인 사람은 급여의 5%, ANALYST인 사람은 그대로, 나머지는 3%만큼 이상된 급여를 CASE문을 사용하여 출력하기

```
SELECT EMPNO, ENAME, JOB, SAL,
    CASE JOB
        WHEN 'MANAGER' THEN SAL*1.1
        WHEN 'SALESMAN' THEN SAL*1.05
        WHEN 'ANALYST' THEN SAL
        ELSE SAL*1.03
    END AS UPSAL
    FROM EMP;
```

스크립트 출력 x | 질의 결과 x

SQL | 인출된 모든 행: 14(0.004초)

| | EMPNO | ENAME | COMM | COMM_TEXT |
|---|---|---|---|---|
| 1 | 7369 | SMITH | (null) | 해당사항 없음 |
| 2 | 7499 | ALLEN | 300 | 수당 : 300 |
| 3 | 7521 | WARD | 500 | 수당 : 500 |
| 4 | 7566 | JONES | (null) | 해당사항 없음 |
| 5 | 7654 | MARTIN | 1400 | 수당 : 1400 |
| 6 | 7698 | BLAKE | (null) | 해당사항 없음 |
| 7 | 7782 | CLARK | (null) | 해당사항 없음 |
| 8 | 7788 | SCOTT | (null) | 해당사항 없음 |
| 9 | 7839 | KING | (null) | 해당사항 없음 |

## - 기준 데이터 없이 조건식만으로 CASE문 사용하기

- CASE문은 DECODE함수와 달리 비교할 기준 데이터를 지정하지 않고 값이 같은 조건 이외의 조건도 사용할 수 있다.
- 밑에의 예시는 COMM 열 값이 NULL, COMM 값이 0, COMM 열 값이 0을 초과할 때 각각 다른 반환 데이터를 지정한다.

```
SELECT EMPNO, ENAME, COMM,
    CASE
    WHEN COMM IS NULL THEN '해당사항 없음'
    WHEN COMM = 0 THEN '수당 없음'
    WHEN COMM > 0  THEN '수당 : ' || COMM
END AS COMM_TEXT
FROM EMP;
```

스크립트 출력 x | 질의 결과 x

SQL | 인출된 모든 행: 14(0.004초)

| | EMPNO | MASKING_EMPNO | ENAME | MAKING_ENAME |
|---|---|---|---|---|
| 1 | 7369 | 73** | SMITH | S**** |
| 2 | 7499 | 74** | ALLEN | A**** |
| 3 | 7521 | 75** | WARD | W**** |
| 4 | 7566 | 75** | JONES | J**** |
| 5 | 7654 | 76** | MARTIN | M**** |
| 6 | 7698 | 76** | BLAKE | B**** |
| 7 | 7782 | 77** | CLARK | C**** |
| 8 | 7788 | 77** | SCOTT | S**** |
| 9 | 7839 | 78** | KING | K**** |

- **CASE문**은 각 조건식의 ture, false 여부만 검사하므로 **기준 데이터가 없어도 사용이 가능**하다.
- **DECODE함수와 CASE문은 모두 조건별로 동일한 자료형의 데이터를 반환해야한다.**

---

# 6강 실습문제

1.

```
SELECT EMPNO, RPAD(SUBSTR(EMPNO, 1, 2), 4, '*')
        AS MASKING_EMPNO,
        ENAME, RPAD(SUBSTR(ENAME, 1, 1), 5, '*')
            AS MAKING_ENAME
      FROM EMP;
```

스크립트 출력 x | 질의 결과 x

SQL | 인출된 모든 행: 14(0.004초)

| | EMPNO | ENAME | SAL | DAY_PAY | TIME_PAY |
|---|---|---|---|---|---|
| 1 | 7369 | SMITH | 800 | 37.2 | 4.7 |
| 2 | 7499 | ALLEN | 1600 | 74.41 | 9.3 |
| 3 | 7521 | WARD | 1250 | 58.13 | 7.3 |
| 4 | 7566 | JONES | 2975 | 138.37 | 17.3 |
| 5 | 7654 | MARTIN | 1250 | 58.13 | 7.3 |
| 6 | 7698 | BLAKE | 2850 | 132.55 | 16.6 |
| 7 | 7782 | CLARK | 2450 | 113.95 | 14.2 |
| 8 | 7788 | SCOTT | 3000 | 139.53 | 17.4 |
| 9 | 7839 | KING | 5000 | 232.55 | 29.1 |

1.

```
SELECT EMPNO, ENAME, SAL,
        TRUNC(SAL/21.5, 2) AS DAY_PAY,
        ROUND(SAL/21.5/8, 1) AS TIME_PAY
        FROM EMP;
```

스크립트 출력 x | 질의 결과 x

SQL | 인출된 모든 행: 14(0.004초)

| | EMPNO | ENAME | HIREDATE | R_JOB | COMM |
|---|---|---|---|---|---|
| 1 | 7369 | SMITH | 80/12/17 | 1981-03-23 | N/A |
| 2 | 7499 | ALLEN | 81/02/20 | 1981-05-25 | 300 |
| 3 | 7521 | WARD | 81/02/22 | 1981-05-25 | 500 |
| 4 | 7566 | JONES | 81/04/02 | 1981-07-06 | N/A |
| 5 | 7654 | MARTIN | 81/09/28 | 1982-01-04 | 1400 |
| 6 | 7698 | BLAKE | 81/05/01 | 1981-08-03 | N/A |
| 7 | 7782 | CLARK | 81/06/09 | 1981-09-14 | N/A |
| 8 | 7788 | SCOTT | 87/04/19 | 1987-07-20 | N/A |
| 9 | 7839 | KING | 81/11/17 | 1982-02-22 | N/A |

1.

```
SELECT EMPNO, ENAME, HIREDATE,
    TO_CHAR(NEXT_DAY(ADD_MONTHS(HIREDATE, 3), '월요일'),
    'YYYY-MM-DD') AS R_JOB,
    NVL(TO_CHAR(COMM), 'N/A') AS COMM
    FROM EMP;
    --NVL은 두 인자 값이 서로 같은 데이터 타입이어야 하는듯
```

스크립트 출력 x | 질의 결과 x

SQL | 인출된 모든 행: 14(0.007초)

| | EMPNO | ENAME | MGR | CHG_MGR |
|---|---|---|---|---|
| 1 | 7369 | SMITH | 7902 | 7902 |
| 2 | 7499 | ALLEN | 7698 | 6666 |
| 3 | 7521 | WARD | 7698 | 6666 |
| 4 | 7566 | JONES | 7839 | 8888 |
| 5 | 7654 | MARTIN | 7698 | 6666 |
| 6 | 7698 | BLAKE | 7839 | 8888 |
| 7 | 7782 | CLARK | 7839 | 8888 |
| 8 | 7788 | SCOTT | 7566 | 5555 |
| 9 | 7839 | KING | (null) | 0000 |

1.

```
SELECT EMPNO, ENAME, MGR,
    CASE
        WHEN TO_CHAR(MGR) IS NULL THEN '0000'
        WHEN SUBSTR(TO_CHAR(MGR), 1, 2) = '75' THEN '5555'
        WHEN SUBSTR(TO_CHAR(MGR), 1, 2) = '76' THEN '6666'
        WHEN SUBSTR(TO_CHAR(MGR), 1, 2) = '77' THEN '7777'
        WHEN SUBSTR(TO_CHAR(MGR), 1, 2) = '78' THEN '8888'
        ELSE TO_CHAR(MGR)
    END AS CHG_MGR
FROM EMP;
```

---

# 프로그래머스 SQL 문제 풀이 NULL

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;
```