Oracle DBMS

CRUD
CREATE, READ(SELECT 활용), UPDATE, DELETE
DB를 검색하고 읽어내는 것이 중요하다.

함수는 대부분 미리 만들어져있어서 잘 이용하면 편리하다.

숫자 함수
함수는 이름을 보고 이름의 내용을 이해하는 것이 중요하다.
ROUND - 반올림
TRUNC - 버림
CEIL - 큰 정수 중 작은 정수 반환
FLOOR - 작은 정수 중 큰 정수 반환
MOD - 몫을 제외한 나머지를 나타냄
이름뒤 () - 대부분 함수를 의미

<ROUND 함수>
```
SELECT ROUND(1234.5678) AS ROUND,
       ROUND(1234.5678, 0) AS ROUND_0,
       ROUND(1234.5678, 1) AS ROUND_1,
       ROUND(1234.5678, 2) AS ROUND_2,
       ROUND(1234.5678, -1) AS ROUND_MINUS1,
       ROUND(1234.5678, -2) AS ROUND_MINUS2
  FROM DUAL;
```

매개변수 : 입력하고 싶은 내용을 의미
0, 1, 2 = 소숫점 자리를 나타내는 것을 의미
-1, -2는 소숫점 앞의 정수 자릿수를 의미

<TRUNC 함수>
```
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;
```
5 이하 값을 버림한다.

<CEIL, FLOOR 함수>

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

올림과 내림을 의미

<MOD 함수>

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

몫을 제외한 나머지 값을 나타낸다.

수학적인 것을 함수로 만들어 두었기 때문에 잘 활용하면 된다.

날짜 함수

<SYSDATE 함수>

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

오늘, 어제, 내일을 나타냄

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

띄워서 써도 같다.

<ADD_MONTHS 함수>

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

현재 날짜로부터 3개월 뒤 날짜를 계산해 준다.

<입사 10주년이 되는 사원 데이터 출력>

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

<입사일 32년 미만 사원 데이터 출력>

```
SELECT EMPNO, ENAME, HIREDATE, SYSDATE
  FROM EMP
 WHERE ADD_MONTHS(HIREDATE, 384) > SYSDATE;
```

<MONTHS_BETWEEN>

```
SELECT EMPNO, ENAME, HIREDATE, SYSDATE,
        MONTHS_BETWEEN(HIREDATE, SYSDATE) AS MONTHS1,
        MONTHS_BETWEEN(SYSDATE, HIREDATE) AS MONTHS2,
        TRUNC(MONTHS_BETWEEN(SYSDATE, HIREDATE)) AS MONTHS3
FROM EMP;
```

<NEXT_DAY, LAST_DAY>

돌아오는 요일, 1달의 마지막 날 구하는 것

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

<날짜 반올림, 버림>

```
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;
```

<<외우지 마세요>>

<형 변환 함수 - 자료형 변환 함수>

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

숫자 + 문자의 형태 = 잘못된 형태

본래 표현을 할 때는 숫자 + 숫자로 해야 함.

컴퓨터는 잘못된 형태를 알아서 고쳐서 나타냄

인터넷 검색시에도 영타 한타 오타때 한영 변환 없이 검색할 때 자동으로 바꿔준다.

```
SELECT 'ABCD' + EMPNO, EMPNO
  FROM EMP
 WHERE ENAME = 'SMITH';
```
에러가 걸린다.

<현재 날짜 표기>
```
SELECT TO_CHAR(SYSDATE, 'YYYY/MM/DD HH24:MI:SS') AS 현재날짜시간
  FROM DUAL;
```
2023/05/02 11:04:26와 같이 나타남

포맷 - 사용하는 형식이라는 의미
TO_CHAR : 문자변환

통합툴 환경설정
파일 아니면 도구에 환경설정이 있음.
<날짜 형식 보기>
환경설정 - 데이터베이스 - NLS

<여러 언어로 날짜 표현>
```
SELECT SYSDATE,
       TO_CHAR(SYSDATE, 'MM') AS MM,
       TO_CHAR(SYSDATE, 'MON', 'NLS_DATE_LANGUAGE = KOREAN' ) AS MON_KOR,
       TO_CHAR(SYSDATE, 'MON', 'NLS_DATE_LANGUAGE = JAPANESE') AS MON_JPN,
       TO_CHAR(SYSDATE, 'MON', 'NLS_DATE_LANGUAGE = ENGLISH' ) AS MON_ENG,
       TO_CHAR(SYSDATE, 'MONTH', 'NLS_DATE_LANGUAGE = KOREAN' ) AS MONTH_KOR,
       TO_CHAR(SYSDATE, 'MONTH', 'NLS_DATE_LANGUAGE = JAPANESE') AS MONTH_JPN,
       TO_CHAR(SYSDATE, 'MONTH', 'NLS_DATE_LANGUAGE = ENGLISH' ) AS MONTH_ENG
  FROM DUAL;
```

<시간 형식 지정 출력>
```
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;
```

<숫자 형식 사용하여 급여 출력하기>
```
SELECT SAL,
       TO_CHAR(SAL, '$999,999') AS SAL_$,
       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;
```

L=LOCALE(현지의)
빈 숫자를 0으로 채워서 표기 하는 경우도 있음

<문자 데이터를 숫자로 변환하여 출력>
TO_NUM 활용
숫자는 ' '을 표시하여 사용하지 않음.

```
SELECT TO_DATE('49/12/10', 'YY/MM/DD') AS YY_YEAR_49,
       TO_DATE('49/12/10', 'RR/MM/DD') AS RR_YEAR_49,
       TO_DATE('50/12/10', 'YY/MM/DD') AS YY_YEAR_50,
       TO_DATE('50/12/10', 'RR/MM/DD') AS RR_YEAR_50,
       TO_DATE('51/12/10', 'YY/MM/DD') AS YY_YEAR_51,
       TO_DATE('51/12/10', 'RR/MM/DD') AS RR_YEAR_51
  FROM DUAL;
```

날짜에 ROUND를 적용한다.

<NULL 값>
NVL, NVL2로 NULL값을 추출한다.
0으로 값을 바

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

SELECT EMPNO, ENAME, COMM,
       NVL2(COMM, '0', 'X'),
       NVL2(COMM, SAL*12+COMM, SAL*12) AS ANNSAL
  FROM EMP;
```

0와 X NULL 값이 맞다 아니다를 표시
]

<DECODE와 CASE>
본의미로 ENCODE 암호화, DECODE 암호화된 것을 푼다는 의미임
복잡한 코드를 단순하게 만든다는 의미

DECODE문

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

DECODE 각각하는 역할을 나누어서 한다.(다른 프로그래밍에서는 IF와 SWITCH문과 같음)

CASE문

```
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;
```

CASE는 END로 표현된다.
SQL에서는 WHEN THEN으로 표현
IF THEN으로 프로그래밍 언어에서는 표현

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

|| : 문자 결합

다중행 함수
<SUM 함수>

급여 합계 출력

```
SELECT SUM(SAL) FROM EMP;
```

한 번에 계산해서 합계를 낸다.

다중행 함수는 행과 동시에 나타낼 수 없다.
★ SELECT ENAME, SUM(SAL) FROM EMP;와 같이 동시에 나타낼 수 없다.

봉급합계 출력
SELECT SUM(COMM) FROM EMP;
SUM=summarize의 의미

급여 합계 구하기(중복되는 것과 전체 모두 나타내기)
```
SELECT SUM(DISTINCT SAL),
       SUM(ALL SAL),
       SUM(SAL)
  FROM EMP;
```

<COUNT 함수>
데이터의 개수를 구해준다.
테이블의 데이터 개수 구하기

EMP 테이블에서의 데이터 개수 구하기
SELECT COUNT(*) FROM EMP;
부서 번호가 30번인 직원 수 구하기
SELECT COUNT(*) FROM EMP WHERE DEPTNO = 30;

COUNT에서의 DISTINCT, ALL 사용하기
SELECT COUNT(DISTINCT SAL), COUNT(ALL SAL), COUNT(SAL) FROM EMP;

추가 수당 열 개수 출력하기
```
SELECT COUNT(COMM) FROM EMP
WHERE COMM IS NOT NULL;
```

SELECT COUNT(COMM) FROM EMP;도 결과가 같다.

<최댓값과 최솟값 MAX, MIN 함수>

부서 번호가 10번인 사원의 최대급여 최소급여 출력
SELECT COUNT(COMM) FROM EMP WHERE COMM IS NOT NULL;
SELECT MIN(SAL) FROM EMP WHERE DEPTNO = 10;

최근 입사일 구하기
SELECT MAX(HIREDATE) FROM EMP WHERE DEPTNO = 20;

<평균값 구하기 AVG 함수>

SELECT AVG(SAL) FROM EMP WHERE DEPTNO = 30;
AVG = AVereGe

중복 제거 평균값 구하기
SELECT AVG(DISTINCT SAL) FROM EMP WHERE DEPTNO = 30;

<결과 값을 묶어서 출력하기 - GROUP BY>
ORDER BY와 같이 쓰이는 경우가 많음.

```
SELECT AVG(SAL), '10' AS DEPTNO FROM EMP WHERE DEPTNO = 10
UNION ALL
SELECT AVG(SAL), '20' AS DEPTNO FROM EMP WHERE DEPTNO = 20
UNION ALL
SELECT AVG(SAL), '30' AS DEPTNO FROM EMP WHERE DEPTNO = 30;
```
같이 하려면 일일이 수정을 해야 함.

```
SELECT AVG(SAL), DEPTNO FROM EMP
GROUP BY DEPTNO;
```
GROUP BY를 사용하여서 간결하게 표현한다.

<GROUP BY와 ORDER BY>
★ 같이 활용하는 빈도가 높음
```
SELECT DEPTNO, JOB, AVG(SAL) FROM EMP
GROUP BY DEPTNO, JOB
ORDER BY DEPTNO, JOB;
```

GROUP BY가 우선 순위로 고려하여 ORDER BY에 의해서 정렬이 된다.

```
SELECT ENAME, DEPTNO, AVG(SAL)
  FROM EMP
GROUP BY DEPTNO;
```
AVG(SAL)과 다중행이 아니여서 에러가 걸림.

<GROUP BY와 HAVING>
SELECT일때는 WHERE을 사용하여 조건을 나타낸다.
★ GROUP BY일때는 HAVING을 사용하여 조건을 나타낸다.

```
SELECT DEPTNO, JOB, AVG(SAL) FROM EMP
GROUP BY DEPTNO, JOB
HAVING AVG(SAL) >= 2000
ORDER BY DEPTNO, JOB;
```

HAVING 대신 WHERE를 사용했을 때 그룹 함수가 작동을 하지 않음.

```
SELECT DEPTNO, JOB, AVG(SAL)
  FROM EMP
 WHERE AVG(SAL) >= 2000
GROUP BY DEPTNO, JOB
ORDER BY DEPTNO, JOB;
```

단순하게 출력을 제한하는 것이 WHERE GROUP화 된 기준에서는 WHERE를 쓰면 의도한대로 나오지 않는다. 그렇기에 HAVING을 써야한다.

<WHERE와 HAVING을 모두 사용하는 경우>

```
SELECT DEPTNO, JOB, AVG(SAL)
  FROM EMP
  WHERE SAL <= 3000
GROUP BY DEPTNO, JOB
HAVING AVG(SAL) >= 2000
ORDER BY DEPTNO, JOB;
```

★ 그룹에 관계되는 조건에서는 GROUP BY ~ HAVING으로 조건을 달고 그 외에는 WHERE로 적용하면 된다.

<다양하게 그룹을 나타내는 방법 - ROLLUP, CUBE, GROUP SETS>
실전에서는 많이 쓰이는 편이 아님
그룹의 디테일한 표현을 하기 위해서 사용하는 것이다.
난해한 그룹화

<ROLLUP>

```
SELECT DEPTNO, JOB, COUNT(*), MAX(SAL), SUM(SAL), AVG(SAL)
  FROM EMP
GROUP BY ROLLUP(DEPTNO, JOB);
```

각 그룹별로 결과를 출력하고 마지막에 총 데이터의 결과를 출력한다.

<CUBE>

```
SELECT DEPTNO, JOB, COUNT(*), MAX(SAL), SUM(SAL), AVG(SAL)
  FROM EMP
GROUP BY CUBE(DEPTNO, JOB)
ORDER BY DEPTNO, JOB;
```

모든 열에 가능한 조합의 결과를 나타냄

<그룹화 후 ROLLUP 이용하여 결과값 내기>

```
SELECT DEPTNO, JOB, COUNT(*)
  FROM EMP
GROUP BY DEPTNO, ROLLUP(JOB);

<GROUPING SETS>
SELECT DEPTNO, JOB, COUNT(*)
  FROM EMP
GROUP BY GROUPING SETS(DEPTNO, JOB)
ORDER BY DEPTNO, JOB;
```

그룹화 함수 - 어려운 부분 이여서 패스해도 될 듯.
<GROUPING 함수>
SELECT절 아래에 GROUPING을 넣어서 그룹화시키는 함수.

```
SELECT DEPTNO, JOB, COUNT(*), MAX(SAL), SUM(SAL), AVG(SAL),
        GROUPING(DEPTNO),
        GROUPING(JOB)
  FROM EMP
GROUP BY CUBE(DEPTNO, JOB)
ORDER BY DEPTNO, JOB;
```

GROUPING을 이용해서 데이터를 취합했을 때는 이용한 데이터는 0, 그렇지 않은 데이터(사용하지 않은 데이터)는 1로 표시

```
<DECODE문으로 GROUPING 함수표현>
SELECT DECODE(GROUPING(DEPTNO), 1, 'ALL_DEPT', DEPTNO) AS DEPTNO,
        DECODE(GROUPING(JOB), 1, 'ALL_JOB', JOB) AS JOB,
        COUNT(*), MAX(SAL), SUM(SAL), AVG(SAL)
  FROM EMP
GROUP BY CUBE(DEPTNO, JOB)
ORDER BY DEPTNO, JOB;
```

NULL값을 의미있는 값으로 표현하는 방법

```
<GROUPING_ID>
SELECT DEPTNO, JOB, COUNT(*), SUM(SAL),
        GROUPING(DEPTNO),
        GROUPING(JOB),
        GROUPING_ID(DEPTNO, JOB)
  FROM EMP
GROUP BY CUBE(DEPTNO, JOB)
ORDER BY DEPTNO, JOB;
```

GROUPING_ID로 데이터 이용한 것(이진수) 0과 1을 합하여 십진수로 나타냄
※ 윈도우 계산기 - 프로그래머 계산기를 사용하면 정밀하게 사용가능 함
HEX(16진수), DEC(10진수), OCT(8진수), BIN(2진수)로 보여줌.