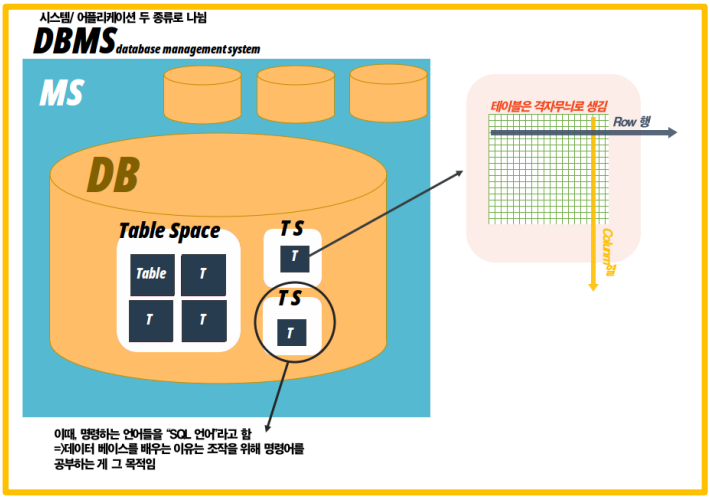
DATABASE 기본개념

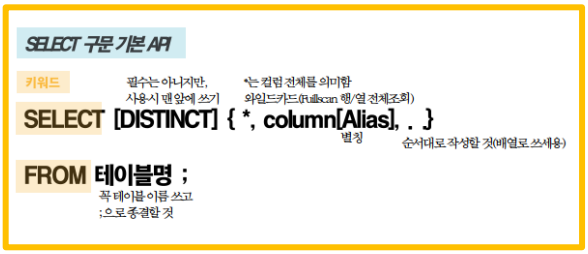
①데이터베이스 기본개념

- **DATABASE 응용 프로그램** : Oracle / mySQL, mssoSQL 등 2천개가 넘지만, 대부분 Oracle을 사용한다
  = 이런 프로그램을 관계형 데이터서비스라고하며, 그 관계에 대해서 집중해 공부함
- **oracle을** 사용하기 위해서는 계정이 필요하며, 대부분 학습용 계정은 scott 비밀번호는 tiger로 일치함



- **RDBMS_관계형 데이터베이스를 기반으로 하는 시스템** : 하나의 덩어리로 처리하는 것이 아니라 나눠서 문제를 처리함 문제 처리속도가 빠름
  구성 - 기본적인 데이터 저장 단위는 테이블로, 로우-칼럼으로 구성됨, 이때 로우는 "레코드" 칼럼은 일종의 목차임
- **SQL : Structured Query Language**
  데이터를 조회SELECT, 저장INSERT, 수정UPDATE, 삭제DELETE 등의 조작 + 테이터들을 저장하기 위해 객체 생성 역할
  =〉 줄여서 "**ISUD 한다"고** 하며 비슷한 의미로 C(CREATE = INSERT)R(READ = SELECT)UD가 있음
- INSERT하면 파일에 바로 업로드 되는게 아니라 메모리에 올라간다 =〉 메모리에 올라간 내용을 지워주는게 **롤백ROLLBACK**

② **SELECT 구문**



= 단문쿼리로 SELECT와 FROM중간에는 컬럼명을 쓴다

SELECT	키워드, 조회 가능(조회:전체데이터를 보는 것 & 검색:여러로우에있는것중에 특정한걸 찾아내는것)
[ ]	사용해도되고, 안해도 된다
DISTINCT	키워드 : 중복제거
{}	순차적으로 쓸 수 있다는 표시 (배열)
*****	컬럼전체를 표시하는 와일드카드 = 전체조회Fullscan으로 컬럼과 로우 전체를 의미함 시간이 오래걸릴수도있으므로 잘사용하지않음
,	컬럼 구분자
columm	컬럼명
Alias	컬럼의 별칭
FROM	키워드 : 뒤에는 테이블 명을 기술
;	문장종결연상자
DESC	키워드 : 테이블에 있는 컬럼명을 전부 보여준다
‘’	오라클에도 데이터타입이 있으며, 문자열의 경우 꼭 싱글포텐션으로 묶어서 기술해야함

- **별칭** : 자바의 변수와 같은 역할로 **테이블명과 컬럼명을 작성할 때 별칭은 반드시 필수적으로 작성**할 것,
  별칭에서 쓰는 AS 는 어떤 컬럼을 ~~로서 사용하겠다는 의미로 **AS는 생략가능하나, 별칭은 생략할 수 없다.**
  별칭에 공백문자나 특수문자를 표현하고 싶거나 대소문자를 구별하고 싶으면 " " 를 사용해서 묶어주며 한글사용 가능하다
- SELECT문의 WHERE구문 : 항상 WHERE이 TURE 값일때만 돌아간다(같지 않다의 연산자 〈〉 != ^= 셋다 알아둘 것)

```
SELECT
    A.EMPNO      EMPNO      -- 사원번호
   ,A.ENAME      ENAME      -- 사원이름
   ,A.JOB        JOB        -- 업무명
   ,A.MGR        MGR        -- 해당사원 상사번호
   ,A.HIREDATE   HIREDATE   --입사일
   ,A.SAL        SAL        --급여
   ,A.COMM       COMM       --커미션
   ,A.DEPTNO     DEPTNO     --부서번호
FROM EMP A
WHERE A.SAL < 2000 ;
```

작성 시 꼭 지켜야 할 것들 : 구문(블록) 동작은 컨트롤+엔터키를 눌러야 가능

a. 문자 데이터 조회 방법 (문자데이터는 반드시 단일 따옴표안에 표시하며 대소문자를 구별함)

참고, 데이터베이스의 자료형

CHAR(size)	정해진 길이만큼 저장 영역을 차지, 최소 크기는 1Byte로 처음 설정한 사이즈만큼 자리를 차지함
VARCHAR2(size)	**1~2000Byte 가변길이 문자 데이터로, 처음 지정한 size외에 실제 입력된 크기만큼의 저장 영역을 차지 String**
NUMBER	최고 40자리 숫자 저장하며, 부호는 길이에 포함되지 않는다(Int in java)
NUMBER(w)	**w자리까지의 수치로 최대 38자리까지 가능함(Int in java)**
NUMBER(w, d)	**w = 전체길이, d는 소수점 이하 자리수를 말하며, 소수점은 자릿수에 포함되지 않는다(double in java)**
DATE	날짜
LONG	길이가 너무 긴 경우에 사용하나, 제약이 많기 때문에 되도록 지양을 권장하며 하나의 테이블에서 오직 하나의 long타입 컬럼만 지정가능함
LOB	저장하는 데이터 타입으로 이미지 문서, 실행파일을 저장할 수 있다
ROWID	행의 유일한 위치값을 가진다 데이터베이스는 ROW가 중요하며 테이블 내 행의 위치를 지정하는 논리적인 값으로 행의 위치를 기억하는 것을 이르며 데이터베이스 전체에서 중복되지 않는 유일한 값임
ROWNUM * 아직까진 딱히 안중요해	조회된 쿼리에 행번호를 붙이는 것 = 쿼리(조회된 데이터)에 추출된 각 행에 부여되는 일련번호로, 순차적으로 부여된다 추출값은 1부터 시작해서 순차값으로 할당함(자바는 0부터~~ 헷갈리지말자) 추출값 (AAAR3s AAE AAAACX AAA) AAAR3s : 데이터 오브젝트 번호 / AAE : 데이터 파일 번호 / AAAACX : 데이터 블럭 번호 / AAA : 블럭내의 행 번호

- 문자데이터는 반드시 ' '안에 작성해야 하며 숫자는 바로 작성하면 됨
- 함수를 사용할때에는 조건과 값을 모두 같이 적용해야한다(일반적으로 자바에서 쓰는 함수 모두 포함됨)

```
SELECT
   A.EMPNO EMPNO --사원번호
  ,A.ENAME ENAME -- 사원명
  ,A.JOB   JOB      -- 직급
  FROM EMP A
-- WHERE A.ENAME= UPPER('miller'); -- 이런식으로 작성하면 곤란해~~
  WHERE UPPER(A.ENAME) = UPPER('miller'); - 함수를 사용할때에는 조건과 값 모두 같이 적용해야함
```

- "특정문자보다 큰 값을 도출하라"는 조건은 ASCII 코드를 기반으로 문자를 숫자로 변환해서 비교함

```
문제
SELECT
   A.EMPNO EMPNO --사원번호
  ,A.ENAME ENAME -- 사원명
  ,A.JOB   JOB   -- 직급
  FROM EMP A
  WHERE A.ENAME >= 'MILLER';  - 밀러보다 크거나 같은 값을 도출하라는 문제
결과값  이름의 첫글자가 M의 아스키코드보다 크거나 같은 값이 도출됨
7369        SMITH     CLERK
7521        WARD      SALESMAN
7788        SCOTT     ANALYST
7844        TURNER    SALESMAN
7934        MILLER    CLERK
```

b. 날짜 데이터 조회(데이터베이스 검색시 날짜검색이 제일 중요하다)

사전개념

DUAL 테이블 : = 더미테이블, 듀얼테이블 / 테스트 및 결과를 출력하기 위한 테이블로 흔적을 찾는다 FROM에 작성함

DUMP 함수 : 듀얼테이블에는 항상 NULL이 들어가있는지 확인해야하므로 DUMP 함수와 짝을 이룬다,NULL 값은 ASCII 코드로 32임 = NULL값 꼭 확인해볼것

```
식                                  결과값
SELECT ' BEE ' FROM DUAL;           = BEE
SELECT DUMP(' BEE ') FROM DUAL;     = Typ=96 Len=5: 32,66,69,69,32
```

날짜*

- SYSDATE : 현재 날짜 월 시간을 나타내는 함수로 형식, 포맷 변경이 가능하며 산술연산도 가능하다

반드시 *단일 따옴표* 안에 표시하며, 년/ 월/ 일 형식으로 기술함 EX) SELECT SYSDATE+1 FROM DUAL;

-TO_CHAR : 날짜 타입을 문자로 바꿔주는 함수 EX) SELECT TO_CHAR(SYSDATE, 'YYYY-MM-DD') FROM DUAL;

```
실제로 사용하는 코드 값 * 각 포맷에 맞춰 비교하고 더해줌
SELECT *
FROM EMP
WHERE TO_CHAR(HIREDATE, 'YYYY-MM-DD')
   >= TO_CHAR(TO_DATE('1995-01-01'),'YYYY-MM-DD');
```

시간

```
SELECT TO_CHAR(SYSDATE, 'HH24:MI:SS') FROM DUAL; -- 24시간버전
SELECT TO_CHAR(SYSDATE, 'HH:MI:SS') FROM DUAL;
```

= 외부에서 들어오는 데이터타입에 맞게 조회할 수 있도록 하는게 제일 중요함(비교식을 써서 잘 변경할 수 있는게 중요하다)

c. 논리 연산자

(a). AND 연산자 : 여러조건을 모두 만족해야 하는 경우 사용하는 연산자로, OR보단 AND를 더 많이 사용함, 맨 마지막에 사용한다

```
SELECT
      ENAME
     ,EMPNO
     ,DEPTNO
FROM   EMP
WHERE  A.SAL >= 1000
AND    A.SAL <= 3000   BETWEEN 1000 AND 3000;
```

이처럼 ~부터 ~까지 등 범위를 검색하는 것을 *구간 검색, FROM-TO검색* 이라고 하며 (두 번째로 중요한 내용)

BEWEEN이 같은 역할을 하나, 일자 - 숫자 검색에서 최대한 BEWEEN 지양해야함(조그만 오류에도 속도가 매우 느려진다)

(b).OR연산자 : 두 조건 중 한가지만 만족하더라도 검색할 수 있도록 하기 위해서 사용하는 연산자

```
SELECT
      ENAME
     ,EMPNO
     ,DEPTNO
FROM   EMP
WHERE  EMPNO = 7844
OR     EMPNO = 7654
OR     EMPNO = 7521;          WHERE  EMPNO IN (7844, 7654, 7521);
```

OR 사용시 IN 키워드를 사용하면 한번에 묶어서 사용할 수 있다.

(c). NOT 연산자 : 반대되는 논리값만 배출함

```
SELECT
      ENAME
     ,EMPNO
     ,DEPTNO
FROM   EMP
WHERE NOT  EMPNO = 10;  결과값은 EMPNO = 10;를 제외한 모든 값이 도출됨
```

d. LIKE 연산자 : 다루기 까다로운 연산자로, 검색하고자 하는 값을 정확히 모를 경우 와일드카드와 함께 사용됨

_ : 지정한 칸 수를 나타내며 사용빈도 낮다

% : 그 뒤로 무엇이든 올 수 있는 기호로 길이/ 내용 모두 상관없음(검색하고자하는 값을 정확히 모를 경우에 사용함)

e. NULL을 위한 연산자 : 오라클 데이터베이스는 칼럼에 NULL 값이 저장되는 것을 허용함

- NULL : 빈공간도 아닌, 어떤 값이 존재하기는 하나, 어떤 값인지 알아낼 수 없는 것을 의미하며 연산/ 할당/ 비교가 전부 불가능함
- NULL을 비교할 때는 '=, <>, != ' 을 사용할 수 없고, IS를 이용해서 비교함

```
SELECT * FROM EMP
WHERE COMM IS NULL; -- NULL값인 경우를 출력

SELECT * FROM EMP
WHERE COMM IS NOT NULL; -- NULL이 아닌 경우를 출력
```

f. Order By 연산자(=Sort =정렬하기)

- 정렬기준은 order by에서 선언한 자료를 기준으로 기본값은 오름차순(), 내림차순의 경우 명시적으로 'DESC'를 선언해야함

```
SELECT *
FROM EMP
ORDER BY EMPNO DESC;  --FROM 뒤에 작성하며 작성된 컬럼을 기준으로 정렬됨
```

- 조회되는 칼럼은 칼럼아이디가 생김, SELECT에서 선언한 순차적으로 1번부터 시작해 숫자값을 가진다

```
SELECT ENAME, JOB, EMPNO ENAME이 1번, JOB 2번 EMPNO 3번
FROM EMP
ORDER BY 1 DESC, 2 ASC ;  -- 1번(ENAME)기준으로 내림차순, 2번째는 2번(JOB)기준으로 오름차순으로 정리함
```

- 오라클에서 NULL은 오름차순에선 마지막에, 내림차순에선 제일 먼저나옴(데이터베이스 프로그램에 따라서 다르다.)

** 컬럼명에서 별칭을 쓰는 이유 : 조회 시 컬럼명이 길게 작성되기 때문에 별칭으로 간단하게 써준다
또한 테이블의 경우 어떤 테이블의 상속을 받았는지 더 간편하게 보기 위해서 별칭을 사용함
=별칭은 동사 말고 명사형, 연산기호 쓰지말고 되도록 영어를 사용할 것

③ **SQL함수의 종류** _함수는 크게 **그룹 함수 / 단일 행 함수**로 나누어 볼 수 있다.

- **그룹 함수** : 조회되는 행의 수와 상관없이 return되는 행은 단일행, 결과적으로 다수의 row가 최종적으로 단일행값임 ex)SUM, COUNT MAX
: 조회후 여러건이 나오면 가능을 부여, 조작해 그룹화한 뒤 하나의 값으로 리턴함
- 종류

SUM	그룹의 누적 합계를 반환
AVG	그룹의 평균을 반환
MAX	**그룹의 최댓값을 반환**
MIN	**그룹의 최솟값을 반환**
COUNT	**그룹의 총 개수를 반환**

a. **GROUP BY** : 같은 값끼리 모아서 리턴한다는 의미로 특정 칼럼 값을 기준으로 테이블을 그룹별로 나누기 위해 사용함
= GROUP BY의 경우 조건절로 WHERE이 아니라 **HAVING을** 쓰며 다른데서는 **HAVING**은 오직 GROUP BY에서만 사용된다

```
SELECT DEPTNO, SUM(SAL)
FROM EMP
GROUP BY DEPTNO -- GRIOUP BY뒤에오는 칼럼을 기준으로 그룹을 정리한다
HAVING DEPTNO = 30;
```

. *일반레코드 함수와 그룹함수를 함께쓸때는 그룹함수가 이긴다(즉, 일반레코드를 쓸 때 처럼 행이 두 개 이상 나올 수 없다)*

```
SELECT DEPTNO, SUM(SAL)  -- SUM은 그룹함수(일반레코드 함수가 아님) = return 되지 않음
FROM EMP
WHERE DEPTNO = 30;  -- 사원번호가 30일때를 기준으로 사원번호와 총 수익합계를 내고싶었으나, 상기의 이유로 불가
↓ 수정
SELECT DEPTNO, SUM(SAL)
FROM EMP
GROUP BY DEPTNO -- 그룹바이를 통해서 묶어줌으로서 행이 한 개만 나오게 됨 >> 값이 return 된다
HAVING DEPTNO = 30;
```

b. **ROLLUP** : GROUP BY함수와 함께쓰이는 함수로 SELECT된 데이터와 그 데이터 총계를 구함

```
-부서별 총액을 구하시오
SELECT
    DEPTNO
   ,SUM(SAL)
   ,SUM(COMM)
FROM EMP --특정 칼럼을 기준으로 그룹을 나누는 GROUP BY 함수로, 각 행의 중복값을 제거하고 목차만 둔다
GROUP BY DEPTNO;

SELECT
    DEPTNO
   ,SUM(SAL) -- 월급의 총합
   ,SUM(COMM) --커미션의 총 합
FROM EMP
GROUP BY ROLLUP(DEPTNO);
-- 롤업 된 DEPTNO를 기준으로 월급의 총합과 커미션의 총합과 사원번호가 열거됨
```

c. **RANK OVER** : 값을 비교해서 정확한 순위를 매기는 함수로 **ROWNUM**과 구별해 쓸 줄 알아야함(**ROWNUM**은 순차적으로 순서를 매기는 함수)

```
SELECT
   EMPNO
  ,ENAME
  ,SAL,
   RANK( ) OVER(ORDER BY SAL DESC) AS RK -- OVER뒤에 괄호 기준으로 정렬함
FROM EMP;
```

- **단일 행 함수** : 조회되는 행과 return되는 행의 수가 동일한 함수ex) SELECT 함수

a. 날짜함수 : DATA형 = 날짜형에 사용하는 함수, 결과값을 날짜 또는 기간을 얻는다:

** 요일의 숫자표현 1 : 일 2 : 월, 3 : 화, 4 : 수, 5 : 목, 6 : 금, 7 :토*

- SYSDATE : 날짜와 시간이 모두 나오는 함수 (더하기나 빼기 같은 연산이 가능하다)
- SYSTIMESTAMP : 날짜, 시간이 초단위로 나오는 함수로 금융프로그램에선 반드시 얘를 사용할 것

```
SELECT SYSDATE FROM DUAL; --따로 받아오는 정보가 없으니 바로 SELECT로 함수를 부름
SELECT SYSTIMESTAMP FROM DUAL;
```

– MONTHS_BETWEEN: 두 날짜 사이의 간격을 계산하는 함수로, 개월수를 구함

: MONTHS_BETWEEN(날짜1, 날짜2) 형태로 사용하며 소수점 처리가 어렵기 때문에 TRUNC와 ROUND중 골라서 사용한다

– NEXT_DAY : 오늘 날짜를 기준으로 다음에 돌아오는(가장 빨리 돌아오는) 해당 요일의 일자를 구하는 함수

```
SELECT
   SYSDATE
  ,NEXT_DAY(SYSDATE, '수요일') -오늘을 기준으로 다음에 돌아오는 수요일은 언제야?
FROM DAUL;
```

- LAST_DAY : 해당 달의 마지막 날짜를 반환하는 함수 LAST_DAY(날짜)

```
SELECT
   SYSDATE,
  ,LAST_DAY(SYSDATE)
FROM DAUL;
```

- 날짜에 대해서도 반올림 가능하다 = 포맷만 지정해주면 날짜도 ROUND 사용 가능 ROUND(요일, '포맷')

CC SCC	4자리 연도의 끝 두 글자를 기준으로 반올림	Q	한 분기의 두 번째 달의 16일 기준 반올림
SYYY YYYY YEAR SYEAR	년 - 반올림 기준 7월 1일	MONTH RM MON MM	월 16일 기준 반올림
DDD D J	일 기준	DAY DY D	한 주가 시작되는 날짜
HH HH12 HH24	시 기준	MI	분 기준

b. 변환함수

숫자형을 문자형으로 변환하는 표

0	자릿수를 나타내며, 자릿수가 맞지 않을 경우 0으로 채움
9	자릿수를 나타내며, 채우지 않는다
L	각 지역별 통화 기호를 앞에 표시
.	소수점
,	천 단위 자리 구분(사용시 ,이 그대로 출력됨)

- TO_CHAR를 이용해서 숫자형을 문자형으로 반환하기

```
SELECT
   A.ENAME ,
  .SAL
  ,TO_CHAR(A.SAL, 'L999,999') – TO_CHAR( 반환할 숫자, 반환할 형태) 통화기호를 갖고 소수점 3자리까지 표기함
FROM EMP A;
```

- TO_DATE : 문자형을 날짜형으로 변환하는 함수
- TO_NUMBER : 문자형을 숫자형으로 반환하는 함수 TO_NUMBER('반환할 숫자', '반환할 형태')

```
SELECT
TO_NUMBER('12,345' , '999,999') + TO_NUMBER( '12,345' , '999,999')
FROM DUAL;
```

조회된 컬럼은 0이 아닌 1부터 시작하는 것을 중요할 것

**NVL 함수 : NVL(첫 번째 인자, 두 번째 인자) 〉〉 첫 번째 인자로 받은 값이 NULL이면 두 번째 인자 값으로 변경함

c. 문자함수

* 날짜의 경우 좌변과 우변 및 형식을 맞춰서 작성한다(형식에 상관없이 나타내는 값이 동일하기 때문에)
  하지만 문자 함수의 경우 대소문자에따라서 값이 달라지기 때문에 구하고자하는 값 외에는 함수를 사용해서는 안된다.
  = 문자함수는 대소문자에 따라서 상이한 값이 도출됨을 유의할 것

** 문자함수 뒤에 B가 붙은건 바이트를 기준으로 한 함수로, 한글은 한글자가 2바이트임을 명심할 것
  인덱스의 경우 한글과 영어가 모두 동일하게 한 글자가 한 인덱스를 차지함

- **LOWER** : 지정된 문자를 소문자로 리턴하는 함수

```
SELECT LOWER(' DATABASE') FROM DUAL;  -- 리턴값 database
```

- **UPPER** : 지정된 문자를 대문자로 리턴하는 함수

```
SELECT UPPER('database') FROM DUAL --리턴값 DATABASE

SELECT
    A.EMPNO
   ,A.ENAME
   ,A.JOB
FROM EMP A
WHERE LOWER(A.JOB) = UPPER('manager'); -- 값이 도출되지 않음

SELECT
    EMPNO
   ,ENAME
   ,JOB
FROM EMP
WHERE UPPER(JOB) = UPPER('manager'); --값이 도출되나 이런식으로 작성하면 안됨 ; 문자의 대소문자는 큰 차이가 있음
```

- **INITCAP** : 첫글자만 대문자로, 나머지는 소문자로 변환함 SELECT INITCAP ('문장') 형태로 사용함
- **CONTCAT** : 두 문자를 연결하는 함수
- **SUBSTR** / **SUBSTRB** : 문자열 일부만 추출하는 함수 SUBSTR는 인덱스를 기준으로 SUBSTRB는 바이트를 기준으로 숫자를 센다
  : 자바에서의 INDEX OF 함수와 동일한 함수임

```
SELECT
 SUBSTR('DATABASE', 1, 3)  --SUBSTR('대상', 위치, 추출할 문자 개수)
FROM DUAL; -- "DAT"글자가 추출됨

/* 시작 위치 인자값이 음수인 경우 뒤쪽부터 센다
= 가장 마지막 인덱스에 -1를 부여함 */
SELECT
    SUBSTR('DataBase',-4,3)
FROM DUAL; --' Bas'가 추출됨
-1 e -2 s -3 a -4 B부터 순차적으로 3개인 Bas가 추출되는 것임

/* 바이트를 기준으로 추출할 때
한글은 1글자가 2바이트임을 명시할 것 */
인덱스를 기준으로 추출 : 3번째 인덱스인 터!부터 4개 추출
SELECT SUBSTR('데이터베이스',3,4)
FROM DUAL; --추출값은 '터베이스'

바이트를 기준으로 추출 : 2번째 글자인 이!부터 2개 추출
SELECT SUBSTRB('데이터베이스',3,4)
FROM DUAL; --추출값은 '이터'

** 외에 1104파일에 있는 138번라인의 심화문제 다시보기
```

- **LENGTH** / **LENGTHB**: 문자 상수나 칼럼에 저장된 데이터의 값이 몇 개의 문자로 구성된건지 길이를 알려주는 함수

```
SELECT LENGTH('어쩌고저쩌고'),LENGTH('FHJKHDKSHF')
FROM DUAL; -- 결과값 한글은 6, 영어는 10글자 (인덱스라서 모두 1칸씩 동일함)

/*바이트를 기준으로 추출할 때
메모리에 차지하는 바이트 수를 구하기 때문에 값이 다르다*/
SELECT LENGTHB('DataBase'),LENGTHB('데이터베이스')
FROM DUAL; -- 추출 리턴값 영어는 8글자, 한글은 12글자가 추출됨
```

*** 쿼리짜기 전에 유의해야할 점은
- 데이터의 개수를 파악하고, NULL값이 있는지 없는지 NVL함수로 확인하고, 데이터 포맷을 확인해야 함

- **INSTR / INSTRB** : 특정 문자의 위치를 구하는 함수 INSERT(대상, 찾을글자, 시작위치, 몇 번째 발견)

```
SELECT
   INSTR('DataBase', 'a', 3, 2)
FROM DUAL; -- 추출값 6
/*3번째에서부터 시작해서 a를 찾는데
2번째로 보이는 a를 찾는다
B가 붙으면 한글은 2바이트로 시작함*/
```

- **LPAD / RPAD** : PADS는 DB에선 채운다는 의미로 들여쓰기할 때 주로 사용하는 함수
  명시된 자릿수만큼만 글자를 나타내고, 남은 자리는 왼쪽부터 특정 기호(작성된 인수)로 채우며 숫자는 내가 쓴 문자열보다 길게작성
  = 자릿수는 빈칸을 의미하는 게 아니라 문자열 전체를 의미하며 LPAD는 왼쪽을 채우고, RPAD는 오른쪽부터 채움

```
SELECT LPAD('DataBase',3,'$') FROM DUAL;
/* 자릿수는 첫번째 인수로 쓰인 문장보다 더 길이를 짧게쓰면 그만큼 문장이 잘려서 나옴 */
-- 값 "DAT"만 출력됨
SELECT LPAD('DataBase',10,'$') FROM DUAL; -- "$$DataBase" 출력
SELECT RPAD('DataBase',10,'★') FROM DUAL; -- "DataBase★" 출력
```

- **TRIM** : 특정 문자열을 잘라내는 함수로 문장 가운데 글자는 지워지지 않고 양 옆에 빈칸(혹은 문자)만 지워진다.

```
TRIM([LEADING/TRALING/BOTH][,tring_character]FROM trim_source)
/* LEADING : 왼쪽글자가 지워짐
   TRALING : 오른쪽 글자가 지워짐
   BOTH : 양 옆 글자가 지워짐 */

SELECT TRIM(LEADING FROM '  ABCD  ') LT, --ABCD00
     LENGTH(TRIM(LEADING FROM '  ABCD  ')) LT_LEN, -- 6 / 왼쪽의 데이터만 지워지기 때문에, 오른쪽 공백이 남아있다
     TRIM(TRAILING FROM '  ABCD  ') RT, --00ABCD
     LENGTH(TRIM(TRAILING FROM '  ABCD  '))RT_LEN, -- 6 / 오른쪽의 공백 데이터만 지워지기 때문에 왼쪽 공백이 남아있다
     TRIM(BOTH FROM '  ABCD  ') BOTH1, --ABCD
     LENGTH(TRIM(BOTH FROM '  ABCD  ')) BOTH1_LEN, --4/ FROM 사용하기
     TRIM('  ABCD  ') BOTH2, --ABCD / FROM 없이 공백 지우기
     LENGTH(TRIM('  ABCD  ')) BOTH2_LEN -- 4
FROM DUAL;
```

d.숫자함수

- **ABS** : 절대값을 반환
- **FLOOR** : 소수점 아래를 잘라냄(버림) -> 정수로 만듦
- **ROUND** : 특정 자릿수에서 반올림하는 함수
- **TRUNC** : 특정 자릿수에서 잘라 버림 - TRUNC(소수, 잘라 버리는 자릿수 : '0 이하 소숫점자릿수' 는 정수, '양수 자릿수는 음수로 나타냄)

```
SELECT TRUNC(12.345, 2), TRUNC(34.456, 0), TRUNC(56.789), TRUNC(78.901,-1)
FROM DUAL; --12.34(소숫점 두번째 자리(양수))  3456          70(-는 정수자리)
```

- MOD : 나머지를 반올림해서 내보냄 - MOD(정수, 나누는 값)

```
SELECT MOD(34,2), MOD(34, 5), MOD(34,7) FROM DUAL;
          --0         4          6
```

e.일반함수

- **NVL :** 첫 번째 인자로 받은 값이 NULL과 같으면 두 번째 인자 값으로 변경하는 함수

```
SELECT COUNT(NVL(TO_CHAR(HIREDATE),0))
-- COUNT함수를 사용할때 NULL값이 나오면 오류가 나오기 때문에 NVL로 값을 줌 (0값으로 줌)
FROM EMP -- NULL이면 오류 발생 NVL을 통해서 값을 준다
```

- **DECODE** : 첫 번째 인자로 받은 값을 조건에 맞춰 변경하며, 자바의 if와 유사함

```
DECODE( 표현식, 조건, 결과,
        조건2, 결과2,
        기본결과)

SELECT DEPTNO FROM EMP GROUP BY DEPTNO;
SELECT DEPTNO, DECODE(DEPTNO,  10, 'ACCOUNTING' -- 조건, 결과 값이 계속 반복됨
                             ,20, 'RESEARCE'
                             ,30, 'SALES'
```

```
                                        ,40, 'OFRATIONS')AS DNAME -- 표에 표현되는건 DNAME
FROM EMP;

SELECT SUM(DECODE(TO_CHAR(A.HIREDATE, 'MM'),'01',1)) "1월" -- 1
    ,SUM(DECODE(TO_CHAR(A.HIREDATE, 'MM'),'02',1)) "2월" -- 2
    -- (TO_CHAR(HIREDATE, 'MM') = 입사일의 월을 문자형 숫자로 변환함
    -- DECODE(TO_CHAR(HIREDATE, 'MM'),'02') 변환한 숫자가 02일 경우 1의 값을 주고
    -- SUM을 통해 부여된 값을 다 더해줌, 한 경우당 1의 값이 나옴
    ,SUM(DECODE(TO_CHAR(A.HIREDATE, 'MM'),'03',1)) "3월" -- null
    ,SUM(DECODE(TO_CHAR(A.HIREDATE, 'MM'),'04',1)) "4월"
    ,SUM(DECODE(TO_CHAR(A.HIREDATE, 'MM'),'05',1)) "5월"
    ,SUM(DECODE(TO_CHAR(A.HIREDATE, 'MM'),'06',1)) "6월"
    ,SUM(DECODE(TO_CHAR(A.HIREDATE, 'MM'),'07',1)) "7월"
    ,SUM(DECODE(TO_CHAR(A.HIREDATE, 'MM'),'08',1)) "8월"
    ,SUM(DECODE(TO_CHAR(A.HIREDATE, 'MM'),'09',1)) "9월"
    ,SUM(DECODE(TO_CHAR(A.HIREDATE, 'MM'),'10',1)) "10월"
    ,SUM(DECODE(TO_CHAR(A.HIREDATE, 'MM'),'11',1)) "11월"
    ,SUM(DECODE(TO_CHAR(A.HIREDATE, 'MM'),'12',1)) "12월"
FROM EMP A;
```

-CASE : DECODE 보다 용량이 적은 조건 함수로 조건에 맞는 문장을 수행한다.

(CASE가 만들어진 이후로 DECODE문은 EDIT+에서 에러처럼 뜨나, 실제로 에러아님, 용량문제)

CASE WHEN 조건 THEN 결과

ELSE 결과 식으로 문장이 나옴

```
SELECT ENAME
   ,CASE
      WHEN A.SAL < 1000 THEN A.SAL + (A.SAL * 0.8)
      WHEN A.SAL BETWEEN 1000 AND 2000 THEN A.SAL + (A.SAL * 0.5)
      WHEN A.SAL BETWEEN 2001 AND 3000 THEN A.SAL + (A.SAL * 0.3)
      ELSE A.SAL
   END SALUP
FROM EMP A;
```

④오라클 테이블 키워드

1. **테이블 생성 관련** 생성 CREATE 변경 ALTER 제거 DROP

a. 생성 CREATE : TABLE 테이블이름(컬럼이름, 데이터타입(), 표현식);

```
CREATE TABLE EMCONINFOR(
    ENAME  VARCHAR2(10)
   ,EAGE   VARCHAR2(3)
   ,EHP    VARCHAR2(11)
   ,EADDR  VARCHAR2(200)
);
```

테이블 만들 때 사이즈나 문자형등의 제한에 관련해서 인서트때는 원래 컬럼형을 쓰지 않아도 되나
초반이기 때문에 꼭 컬럼형에 맞추서 쓰도록 노력할 것
작성시에는 식별하기 편하도록 들어오는 목차와

+ CREATE AS : 기존 코드를 새로운 테이블로 복사하는 구문

b. 제거 DROP : DROP TABLE 테이블이름

2. 테이블 내용 추가 : 추가 INSERT 수정 UPDATE 삭제 DELETE`

a. 추가 INSERT - 특정한 칼럼에만 DATA를 입력하는 경우 (반드시 칼럼명 작성한 후 넣고싶은 데이터를 넣음)

```
INSERT INTO EMCONINFOR (ENAME, EAGE, EHP, EADDR) --여기에만 데이터를 넣음
VALUES('A','B','C','D');
SELECT * FROM EMCONINFOR;
```

- 전체 컬럼에 DATA를 입력하는 경우 : 굳이 컬럼명을 작성할 필요 없지만 최대한 작성할 것

```
INSERT INTO EMCONINFOR --컬럼명 굳이 쓰지 않아도 순서대로 값이 들어감
VALUES('E','F','G','H');
```

+ *COMMIT*: 데이터베이스에 INSERT INTO를 이용해 데이터를 저장하면 "메모리에 올라감" , " 물리적 파일에 올라감" 〉 두가지 단계를 거치게 됨
SQL에서는 1단계에서도 데이터 조회가 가능하나, 외부 응용프로그램에서는 메모리를 조회할 수 없기 때문에 COMMIT 명령어를 선언해
물리적 파일에 저장하는 2단계를 시행해야 한다

++ *ROLLBACK*: COMMIT하기 전 메모리에만 올라간 데이터들을 지워준다(메모리에 올라간 데이터를 지워줌)

3. 무결성 제약 조건 : 잘못된 데이터가 입력되지 않도록 제약조건을 지정함

a. NOT NULL : NULL을 허용하지 않는다 : 연결고리 역할을 하는 데이터에는 반드시 지정해야함
b. UNIQUE : 중복된 값을 허용하지 않는다는 의미
c. PRIMARY : NOT NULL + UNIQUE 를 모두 결합한 형태

```
CREATE TABLE MEM_TEST(
    M_NUM       NUMBER(4) PRIMARY KEY
   ,M_ID        VARCHAR2(8) NOT NULL
   ,M_PW        VARCHAR2(8) NOT NULL
   ,M_NAME      VARCHAR2(20)
   ,M_EMAIL     VARCHAR2(100)
   ,M_TEL       VARCHAR2(16)
   ,M_JUSO      VARCHAR2(200)
   ,M_INSERT    DATE
   ,M_UPDATE    DATE

DESC MEM_TEST; -- Descrive : 해당 테이블에 대한 정보를 보여주는 키워드
```

*** CREATE AS 구문으로 테이블 카피할때는 '이전 테이블의 컬럼명(where문이 false인 경우)' or '데이터' or 'null 설정값만 넘어감
컬럼+데이터(NOT NULL)만 넘어가고 오브젝트들은 카피대상이 아니라 일일이 작업이 필요함
특히나 PRIMARY KEY라는 설정값은 복사되지 않아 반드시 후처리 필요.

/*두고두고 볼 에러코드 설명*/
- ORA-00001 : 무결성 제약 조건에 위배되는 경우_PRINMARY KEY 조건 중 UNIQUE 조건을 위배한 경우
- ORA-00947 : 컬럼의 갯수가 맞지 않는 경우를 말함
- ORA-01400 : NOT NULL 조건인데 NULL이 들어가는 경우 발생하는 에러
- ORA-01722 : 수치가 부족합니다_넘버타입에다가 다른 데이터타입을 넣는 경우
- ORA-00984 : 열을 사용할 수 없습니다_ insert시 유효한 값을 내놓으라는 의미 VARCHAR2에다 다른 데이터타입을 집어 넣는 경우
- ORA-00913 : 값의 수가 너무 많습니다_컬럼의 수보다 많은 데이터를 넣는 경우
- ORA-12899 : 열에 대한 값이 너무 많은경우, 설정한 size보다 많은 값을 넣어서 나는 오류(글자)
- ORA-01438 : 지정된 자릿수보다 더 큰 값을 넣은 경우(숫자)
오류가 한가지가 아닌 경우 = 1. 갯수 2. 적힌 순서 대로 오류 코드가 떠오른다.