데이터 정의어 **DDL(Data Definition Language)**

- 데이터 구조를 정의하기 위한 테이블 생성, 변경, 삭제 명령
- CREATE, DROP, ALTER

1. 테이블 생성 명령

CREATE TABLE 테이블명**(**
컬럼명 데이터 타입[(크기)][NOT NULL] [DEFAULT] 값][,]
:
컬럼명 데이터 타입[(크기)][NOT NULL] [DEFAULT] 값][,]
[CONSTRAINT 기본키설정명 PRIMARY KEY(컬럼명[,컬럼명,...])[,]]
[CONSTRAINT 외래키설정명1 FOREIGN KEY(컬럼명) REFERENCES 테이블명(컬럼명) DELETE ON CASCADE[,]]
:
[CONSTRAINT 외래키설정명n FOREIGN KEY(컬럼명) REFERENCES 테이블명(컬럼명) DELETE ON CASCADE];

- 기본키 설정명 : 부여한 이름으로 중복 사용 불가
- 외래키 설정명 : 부여한 이름으로 중복 사용 불가
- REFERENCES 테이블명 : 부모테이블명
- REFERENCES 테이블명(컬럼명) : 부모테이블에서 사용한 컬럼명
- DELETE ON CASCADE : 부모T에서 특정행(ROW) 삭제 시 자식테이블 자료부터 삭제하고 부모테이블 자료 삭제 허용

EX)

- 근무 테이블

```
CREATE      TABLE       BL_WORK(
            EMP_ID      CHAR(4),
            SITE_ID     NUMBER(3),
            INS_DATE    DATE,
CONSTRAINT pk_tbl_work  PRIMARY KEY(EMP_ID,SITE_ID),
CONSTRAINT fk_tbl_work_emp      FOREIGN KEY(EMP_ID)     REFERENCES      EMP(EMP_ID),
CONSTRAINT fk_tbl_work_site     FOREIGN KEY(SITE_ID)    REFERENCES      SITE(SITE_ID));
```

2. 테이블 삭제 명령

DROP TABLE 테이블명;

- 관계가 설정된 부모테이블은 임의로 삭제 불가 ⇒ 관계가 삭제된 후 또는 자식테이블이 삭제된 후 삭제 가능
- TABLE의 구조체를 없애는 것

EX)

- 사업장 테이블을 삭제하시오.
  - 자식테이블부터 삭제

```
DROP TABLE TBL_MAT;
DROP TABLE TBL_WORK;
DROP TABLE TBL_SITE;
--제약조건 삭제 후 테이블 삭제
```

(ALTER TABLE 테이블명 DROP CONSTRAINT 기본키설정명|외래키설정명;)

```
ALTER TABLE TBL_MAT    DROP CONSTRAINT fk_tbl_mat_site;
ALTER TABLE TBL_WORK   DROP CONSTRAINT fk_tbl_work_site;
DROP TABLE TBL_SITE;
```

3. 변경 명령

- 테이블 이름 변경, 컬럼 이름 변경, 컬럼 타입 변경
- 컬럼 삽입, 제약조건 추가 삽입
- 컬럼 삭제, 제약조건 삭제 등의 기능 수행
- 제약 조건은 변경 X → 삭제하고 삽입해야 함

1) 테이블 이름 변경

ALTER TABLE 원본테이블명 RENAME TO 변경테이블명;

EX)

- 사업장 테이블(TBL_SITE)를 생성하고 SITE로 테이블명을 변경하시오.

```
ALTER TABLE TBL_SITE    RENAME    TO    SITE;
```

2) 컬럼 이름 변경

ALTER TABLE 테이블명 **RENAME COLUMN** 이전컬럼명 **TO** 변경컬럼명**;**

EX)

- SITE테이블에 사업장주소(SITE_ADDR) 컬럼명을 SITE ADDRESS로 변경하시오.
  ALTER TABLE SITE RENAME COLUMN SITE_ADDR TO SITE_ADDRESS;

3) 컬럼 데이터 타입**,** 크기 변경

ALTER TABLE 테이블명 **MODIFY** 컬럼명 타입**[(**크기**)]**

- 원본 컬럼의 크기보다 작은 크기로 변경하는 것은 허용되지 않음(오라클은 절대 불가!!) /80->50xxxx 80->100OOO ->100->80XXXX
- "cannot decrease column length because some value is too big"
- CHAR은 고정길이라 남은 길이가 날라가는 게 아니라 공백이 다 채워져서 저장됨. 공백도 유효자료임

EX)

- SITE테이블의 SITE ADDRESS컬럼(VARCHAR2(255))을 고정길이 문자열 100 BYTE(CHAR(100))로 변경하시오.
  ALTER TABLE SITE MODIFY SITE_ADDRESS CHAR(30);;
  ALTER TABLE SITE MODIFY SITE_ADDRESS CHAR(80);
  ALTER TABLE SITE MODIFY SITE_ADDRESS VARCHAR(255); – VARCHAR은 가변길이라 가능

4) 컬럼**,** 제약조건 추가

ALTER TABLE 테이블명 **ADD(**컬럼명 데이터타입**[(**크기**)])**
ALTER TABLE 테이블명 **ADD(CONSTRAINT** 기본키설정명 **PRIMARY KEY(**컬럼명**[,...]))**
ALTER TABLE 테이블명 **ADD(CONSTRAINT** 외래키설정명 **FOREIGN KEY(**컬럼명**) REFERENCES** 테이블명**(**컬럼명**))**

EX)

- SITE테이블에 기본키(SITE_ID)를 설정하시오.
  ALTER TABLE SITE ADD(CONSTRAINT pk_site PRIMARY KEY(SITE_ID));

5) 컬럼 삭제**,** 제약조건 삭제

ALTER TABLE 테이블명 **DROP COLUMN** 컬럼명**|DROP CONSTRAINT** 제약이름**;**

- 삽입, 삭제는 거의 잘 안 일어남 -> CREATE와 마찬가지로 ALTER는 사용빈도가 낮음 / 보통 DB매니저가 하는 것! 사용자는 변경 권한X

4. 기타 명령

COMMIT;

- 하드웨어 공간에 저장하는 것으로 “완료”의 의미
- COMMIT하기 전까지는 메모리 부분에 저장해놓은 것

ROLLBACK;

- COMMIT하기 전으로 돌리는 것
- DROP은 ROLLBACK의 대상이 아님
- DROP은 TABLE의 구조체를 없애는 것

데이터 조작어 **DML(Data Manipulation Language)**

- INSERT(삽입), UPDATE(데이터 변경), DELETE(데이터 삭제), MERGE(머지? 크큭ㅋ크큭 합치기)

1. INSERT

INSERT INTO 테이블명[(컬럼명,...)] VALUES(값,...값);

- 테이블에 새로운 자료를 추가할 때 사용
- (컬럼명,...) : VALUES 절에 정의된 값을 저장할 컬럼명
  - 생략하면 모든 컬럼에 값을 기술해야 함
  - 컬럼명을 기술할 때 NOT NULL인 것은 언급해야 함!
  - 일부 컬럼에만 값을 삽입할 경우로, 컬럼설정 시 NOT NULL 정의된 컬럼은 절대 생략 불가
  - 사용된 컬럼의 개수, 순서와 VALUES 절 값의 개수, 순서는 일치해야 함
- 제일 상위 테이블부터 하위 테이블의 방향으로 입력해야 함

EX)

- 다음 자료를 사원테이블(EMP)에 저장하시오.

사원번호	사원명	부서명
B100	홍길동	영업부
B200	이정훈	
B121	김영훈	영업부

```
INSERT      INTO  EMP   VALUES('B100','홍길동','영업부');
INSERT      INTO  EMP   VALUES('B200','이정훈','');
INSERT      INTO  EMP(DEPT_NAME, EMP_ID, EMP_NAME) VALUES('영업부','B121','김영훈');
```

  - 데이터 넣을 게 없다면 (안 넣으면 안 돌아감) NULL이나 ''화이트 스페이스
  - 오라클은 모두 문자열, 1번부터 셈(자바는 문자 있고, 0부터 셈, 문자열 표현은 "")
  - 컬럼 순서는 내 마음대로 정해서 써도 됨

```
SELECT      *       FROM EMP;
```

  - *=ALL

2. UPDATE

```
UPDATE      테이블명
    SET     컬럼명 =      값[,]
            컬럼명 =      값[,]
                :
            컬럼명 =      값
[WHERE 조건]
```

- 저장된 자료를 수정할 때 사용
- WHERE 생략 시 모든 행을 같은 값으로 적용
  *다른 계정 테이블을 쓰고 싶다면(접근하고 싶다면) “다른계정명.테이블”

  EX)

```
UPDATE HR.EMPLOYEES
        SET EMP_NAME=FIRST_NAME||' '||LAST_NAME;
        -- 107명 다 해야 되니까 WHERE 절 필요 없음
        --"||" 자바에서 "+"와 같이 문자를 두 개 붙임
```

데이터 타입

- 오라클에는 문자열, 숫자, 날짜, 이진수 자료타입이 제공됨

1. 문자 데이터타입

CHAR VARCHAR VARCHAR2 NVACHAR2 LONG CLOB NCLOB 등

- 오라클의 문자자료는 ‘ ’ 안에 기술된 자료
- 예약어와 같더라도 ‘’ 안에 글자들은 ASCII 코드로 저장됨
- ‘’ 안에는 대소문자 구별 → 코드 값이 달라서!! (A랑 a랑 다름)

1) CHAR

CHAR(n[BYTE|CHAR])

- 고정길이 문자열 저장(최대 2000BYTE까지 저장가능)
  *CHAR 빼고는 다 가변길이(빈 공간의 경우 운영체제에 반납함)
  - 큰 걸 작은 곳에 넣을 때 오라클은 잘려나가는 것이 없고, 입력이 그냥 안 됨
  - n[BYTE|CHAR]이 생략되면 BYTE로 취급
  - n글자수까지 저장(영문 기준)
- (완성형) 한글 한 글자는 3BYTE로 저장(n이 2000이라면 666개까지 사용 가능) ←→ (조합형)
- 기본키나 길이가 고정된 자료(주민번호, 우편번호)의 정당성을 확보하기 위해 사용
- 반드시 크기 지정 필요함! *소문자 c 캐릭터 d 데이터
  *문자는 왼쪽 정렬 / 빈공간은 오른쪽 정렬

EX)

```
CREATE    TABLE TEMP01
          COL1  CHAR(10),
          COL2  CHAR(10 BYTE),
          COL3  CHAR(10 CHAR));
INSERT    INTO  TEMP01     VALUES('대한','대한민','대한민국');
SELECT    *     FROM TEMP01;
SELECT    LENGTHB(COL1)    AS    COL1, –LENGTHB 공백 확인
          LENGTHB(COL2)    AS    COL2,
          LENGTHB(COL3)    AS    COL3
FROM      TEMP01;
```

//대한민국(4*3=12) 나머지 6개는 영어 기준으로 저장됨 – CHAR!!!이라

2) VARCHAR2

VARCHAR2(n[BYTE|CHAR])

- 가변길이 문자열 저장(최대 4000BYTE까지 저장 가능)
- VARCHAR, NVARCHAR2와 저장형식 동일(N = NATIONAL(다국어 형식) → UTF-16, 8)
- 오라클에서만 쓰고 VARCHAR와 기능적으로 동일함

EX)

```
CREATE    TABLE TEMP02(
          COL1  CHAR(20),
          COL2  VARCHAR2(2000 BYTE),
```

```
                COL3  VARCHAR2(4000 CHAR));
INSERT          INTO  TEMP02       VALUES('ILPOSTINO', 'BOYHOOD',
                                   '무궁화 꽃이 피었습니다-김진명');
SELECT          *     FROM TEMP02;

SELECT          LENGTHB(COL1)      AS     COL1,
                LENGTHB(COL2)      AS     COL2,
                LENGTHB(COL3)      AS     COL3,
                LENGTH(COL1)       AS     COL1,
                LENGTH(COL2)       AS     COL2,
                LENGTH(COL3)       AS     COL3
FROM            TEMP02;
```

3) LONG

컬럼명 **LONG**

- 가변길이 문자열 저장(최대 2GB까지 저장 가능)
- 현재 기능 개선 서비스 종료(오라클8i) → CLOB(CHARACTER LARGE OBJECT)로 UPGRADE
- 한 테이블에 한 컬럼만 LONG타입 사용 가능 → 요즘 새롭게 쓰지는 않음
    - LONG 타입 자료를 참조하기 위해 최소 31bit가 필요함 ⇒ 일부 기능(LENGTHB 등의 함수)이 제한 → CLOB는 사용 가능
    - SELECT문의 SELECT절, UPDATE의 SET절, INSERT문의 VALUES절에서 사용 가능

EX)

```
CREATE          TABLE TEMP03(
                COL1  VARCHAR2(2000),
                COL2  LONG);

INSERT          INTO  TEMP03       VALUES ('대전시 중구 계룡로 846',' 대전시 중구 게룡로 846');

SELECT          SUBSTR(COL1,8,3)
                      --SUBSTR(COL2,8,3)
                      --LENGTHB(COL2)
FROM            TEMP03;
```

4) CLOB

컬럼명 **CLOB**

- 가변길이 문자열 저장(최대 4GB까지 저장 가능)
- 한 테이블에 복수 개의 CLOB 타입 정의 가능
- 일부 기능은 DBMS_LOB API(Application Programming Interface)에서 제공하는 함수 사용

EX)

```
CREATE      TABLE TEMP04(
            COL1  VARCHAR2(255),
            COL2  CLOB,
            COL3  CLOB);

INSERT      INTO  TEMP04      VALUES('APPLE BANANA PERSIMMON','APPLE BANANA PERSIMMON','APPLE BANANA PERSIMMON');

SELECT      *       FROM TEMP04;
SELECT      SUBSTR(COL1,7,6)  AS     COL1,
            SUBSTR(COL3,7,6)  AS     COL3, --> 7번재에서 6글자
                  -- LENGTHB(COL2) AS COL4, --> 지원되지 않음
            DBMS_LOB.GETLENGTH(COL2)   AS     COL4, --> 글자수 반환(LONG타입은 안 됨)
            DBMS_LOB.SUBSTR(COL2,7,6)  AS     COL2 --> 6번째에서 7글자
FROM        TEMP04;
```

//TRIM 사용하면 불필요한 공백 잘라냄! LONG은 SUBSTR 안 됐는데 CLOB은 처리할 수 있는 길이면 SUBSTR이 됨

2. 숫자 자료형

- 정수와 실수 저장

1) NUMBER

NUMBER[(정밀도|*[,스케일])]

- 값의 표현 범위 : 10e-130 ~ 9.999.9E125
- 정밀도 : 전체 자릿수(1~38)
- 스케일 : 소수점 이하의 자릿수

- '*'는 38자리 이내에서 사용자가 입력한 데이터를 저장할 수 있는 최적의 기억공간을 시스템이 설정하는 것
- 스케일이 양수인 경우 : 저장은 소수점 이하 '스케일' + 1번째 자리에서 반올림하여 '스케일'절까지 저장
- 스케일이 음수인 경우 : 정수부분 '스케일' 자리에서 반올림하여 저장
    - 생략되면 0을 쓴 것으로 간주함
    - 1 2 3 4 5 6 7 8 9 11(?) 12(?)
    - -5 -4 -3 -2 -1 1 2 3 4 5 6
- 오라클 같은 곳에서는 데이터 절삭을 안 하는 것이 현명하다 / NUMBER가 알아서 해줌

EX)

선언	입력값	저장형태
NUMBER	12345.6789	12345.6789
NUMBER(*,2)	12345.6789	12345.68
NUMBER(6,2)	12345.6789	ERROR -- 소수점 자리는 반올림 - 정수자리가 모자르면 오류남 - 기억장소가 부족
NUMBER(7,2)	12345.6789	12345.68
NUMBER(8,0)	12345.6789	12346 --공백도 필요해 - 3개
NUMBER(6)	12345.6789	12346
NUMBER(6,-2)	12345.6789	12300 -- 정수 자리밖에 없다. 소수점 자리는 없다!!

```
CREATE    TABLE TEMP05(
          COL1  NUMBER,
          COL2  NUMBER(*,2),
          COL3  NUMBER(6,2),
          COL4  NUMBER(7,2),
          COL5  NUMBER(8,0),
          COL6  NUMBER(6),
          COL7  NUMBER(6,-2));

INSERT INTO TEMP05 VALUES(12345.6789,12345.6789,2345.6789,12345.6789,
                  12345.6789,12345.6789,12345.6789);
        SELECT * FROM TEMP05;
```

** 정밀도<스케일인 경우

- 정밀도 : 소수점 이하에서 0이 아닌 유효숫자의 개수
- 스케일 : 소수점 이하의 자릿수
- [스케일 - 정밀도] : 소수점 이하에서 존재해야할 0의 개수

입력값	선언	저장된 값
1234.5678	NUMBER(2,4)	ERROR -- 정수 부분이 들어가려면 일단 저장 안 됨
0.12	NUMBER(3,5)	ERROR -- 복합적인 문제들로 인해...그런데 그 문제가 뭔지는 모르겠고 ㅋㅋ
0.003456	NUMBER(2,4)	0.0035
0.0345678	NUMBER(2,3)	0.035

3. 날짜 자료형

- 날짜 시각 정보를 저장(년, 월, 일, 시, 분, 초)
- 날짜 자료는 덧셈과 뺄셈이 가능함

1) DATE

컬럼명 **DATE**

- 기본 날짜 및 시각정보 저장 (년/월/일 –구분– 시/분/초) → DATE는 크기가 없음
  - SELECT로 불러올 때 시/분/초는 안 나옴(하지만 저장된 것)
- 덧셈은 더해진 정수만큼 다가올 날짜(미래)
- 뺄셈은 차감한 정수만큼 지나온 날짜(과거)
  - 날짜 자료 사이의 뺄셈은 날 수(DAYS) 반환(큰 날짜 - 작은 날짜)
  - 곱셈, 나눗셈은 안 됨

** 시스템이 제공하는 날짜정보는 SYSDATE함수를 통하여 참조할 수 있음

EX)

```
CREATE      TABLE TEMP06(
            COL1  DATE,
            COL2  DATE,
            COL3  DATE);

INSERT      INTO  TEMP06      VALUES(SYSDATE, SYSDATE-10, SYSDATE+10);
SELECT      *     FROM TEMP06;
SELECT      TO_CHAR(COL1, 'YYYY-MM-DD'), -- 형식 지정 문자열
            TO_CHAR(COL2, 'YYYY-MM-DD HH24:MI:SS'),
            TO_CHAR(COL3, 'YYYY-MM-DD HH12:MI:SS')
FROM        TEMP06;
```

```
SELECT      CASE  MOD(TRUNC (SYSDATE)    -       TRUNC(TO_DATE('00010101'))-1,7) -- TRUNC 절삭(자료 버리기) MOD : 나머지
            WHEN 1      THEN  '월요일' -- 분기문 / IF문 SWITCH문 따로 없음 그래도 이렇게 비슷하게 씀
            WHEN 1      THEN  '화요일'
            WHEN 1      THEN  '수요일'
            WHEN 1      THEN  '목요일'
            WHEN 1      THEN  '금요일'
            WHEN 1      THEN  '토요일'
            ELSE  '일요일'
            END   AS      요일
FROM DUAL; -- SELECT 쓰려면 FROM까지 꼭 써줘야 함, 테이블은 필요는 없지만 FROM 규격을 위해 필요함

SELECT      SYSDATE-TO_date('20200807')      FROM DUAL;
```

2) TIMESTAMP 타입

컬럼명 **TIMESTAMP**

- 시간대 정보 없이 정교한 시각정보 저장

컬럼명 **TIMESTAMP WITH LOCAL TIME ZONE**

- 데이터베이스가 운영 중인 서버의 시간대를 기준으로 서버에 접속하는 클라이언트와 시차가 계산된 시간 입력
- 시간은 클라이언트 지역의 시간으로 자동 변환 출력되기 때문에 시간대 정보는 저장되지 않음

컬럼명 **TIMESTAMP WITH TIME ZONE**

- 서버의 시간대 정보 저장 → 대륙명/도시(아시아/서울)

EX)

```
CREATE      TABLE TEMP07
            (COL1 DATE,
            COL2  TIMESTAMP,
            COL3  TIMESTAMP  WITH  LOCAL       TIME  ZONE,
            COL4  TIMESTAMP  WITH  TIME  ZONE);

INSERT      INTO  TEMP07      VALUES(SYSDATE, SYSDATE, SYSDATE, SYSDATE);
```

```
SELECT	*	FROM TEMP07;
```

4. 기타 자료형

- 이진자료를 저장하기 위한 데이터 타입
- RAW, LONG RAW, BLOB, BFILE 등이 제공됨
- 이진자료는 오라클이 해석하거나 변환하지 않음(HTML, SCRIPT, SPRING, JSP 등의 영역)

1) RAW

컬럼명 **RAW(크기)**

- 작은 이진자료 저장(최대 2000BYTE까지 저장 가능) → 인덱스 처리가 가능(?)
- 16진수와 2진수 형태로 저장 *2진수 BINARY, 10진수 DECIMAL, 16진수 HEXADECIMAL, 8진수 OCTAL

EX)

```
CREATE	TABLE TEMP08(
		COL1	RAW(2000));

INSERT	INTO	TEMP08	VALUES('2A7F'); -- 2BYTE
INSERT	INTO	TEMP08	VALUES(HEXTORAW('2A7F'));
INSERT	INTO	TEMP08	VALUES('0010101001111111'); -- (2)0010 (A)1010 (7)0111 (F)1111

SELECT	*	FROM TEMP08
```

2) BFILE

컬럼명 **BFILE**

- 이진자료 저장(최대 4GB까지 저장 가능)
- 대상이 되는 이진자료를 데이터베이스 외부에 저장하고 데이터베이스에는 경로 정보만 저장
  - 해당 원본이 테이블 안에 저장되면 BLOB / 밖에 저장되면 BFILE

EX)

자료 저장 순서

(1) 자료 준비

D:\A_TeachingMaterial\02_Oracle\SAMPLE.JPG

(2) 테이블 생성

```
CREATE TABLE TEMP09(
        COL1 BFILE);
```

(3) 디렉토리 객체 생성 - 경로정보 및 파일명

```
CREATE    OR    REPLACE    DIRECTORY  별칭    AS    '경로명';
CREATE    OR    REPLACE    DIRECTORY  TEST_DIR    AS    'D:\A_TeachingMaterial\02_Oracle';
```

(4) 저장

```
INSERT    INTO  TEMP09    VALUES(BFILENAME('TEST_DIR','SAMPLE.jpg'))
SELECT * FROM TEMP09; -- 오라클은 절대로 해석하지 않는다! 그냥 보여줄 뿐이다. TEMP09에 사진을 넣은 것이 아님
-- 그냥 경로랑 이름이랑 합쳐서 넣어짐 / DB에서 내용만(사진을 바꾸고 싶다면) 바꾸지 이름이랑 경로를 바꾼 것은 아님
```

3) BLOB

컬럼명 **BLOB**

- 원본 이진자료를 테이블 내부에 저장(최대 4GB까지 저장 가능)

EX)

```
CREATE    TABLE       TEMP10(
          COL1  BLOB);
데이터 삽입
DECLARE
L_DIR       VARCHAR2(20):='TEST_DIR';
L_FILE      VARCHAR2(30):='SAMPLE.jpg';
L_BFILE     BFILE;
L_BLOB      BLOB;
BEGIN
INSERT      INTO  TEMP10      VALUES(EMPTY_BLOB())
RETURN      COL1  INTO  L_BLOB;

L_BFILE:=BFILENAME(L_DIR,L_FILE);

DBMS_LOB.FILEOPEN(L_BFILE,DBMS_LOB.FILE_READONLY);
DBMS_LOB.LOADFROMFILE(L_BLOB,L_BFILE, DBMS_LOB.GETLENGTH(L_BFILE));
DBMS_LOB.FILECLOSE(L_BFILE);
```

자료 검색 명령

1. SELECT

1) SELECT

SELECT *|[DISTINCT] 컬럼명 **[AS** 별칭**][,]**
컬럼명 **[AS** 별칭**][,]**
:
컬럼명 **[AS** 별칭**]**
FROM 테이블명**;**
[WHERE 조건**]**
[ORDER BY 컬럼명|컬럼인덱스 **[ASC|DESC] [,** 컬럼명|컬럼인덱스 **[ASC|DESC],...]];**

- '*' = ALL / 컬럼럼명 쓰는 것 = 일부 컬럼만 쓰겠다
- DISTINCT : 컬럼에서 중복되어 있는 것 중 대표되어지는 것만 출력(중복 배제)
- 별칭 : 해당 컬럼을 참조할 때, 출력 시 컬럼의 구별자로 사용(컬럼 제목 지정)
  FROM / WHERE / SELECT 순으로 실행되고 이 세 개가 기본임
- WHERE : 출력할 행 / 생략되면 전부 다 출력하겠다(행과 관련) ⇒ 조건이 거짓이면 출력 안 함
- ORDER BY : 순서화(오라클은 1번부터)
- 컬럼인덱스 SELECT 절에서 해당 컬럼의 사용 순번(1번부터)
- ASC : 오름차순, DESC : 내림차순, 생략하면 ASC 기본

연산자

1. 관계(비교) 연산자

- 자료의 대소 관계를 비교하는 연산자로 결과는 참(TRUE)과 거짓(FALSE)로 반환 → 숫자로 나타나는 것 아님
- >,<,>=,<=,=,!=(<> 같지 않다 → 다른 언어에서는 ><도 씀)
- 표현식 (CASE WHEN ~ THEN, DECODE)이나 WHERE 조건절에 사용

EX)

```
WHERE       DEPARTMENT_ID=50;
WHERE       MEM_JOB = '주부';
```

2. 산술 연산자

- '+', '-', '*', '/' => 4칙 연산자 *나머지 연산자도 없음(함수로 씀 → remainder) // infix/prefix/postfix

EX)

사원테이블(HR.EMPLOYEES)에서 보너스를 계산하고 지급액을 결정하여 출력하시오.(모든 값은 정수 부분만 출력)

보너스=본봉*영업실적의 30% / 지급액=본봉+보너스

Alias는 사원번호, 사원명, 본봉, 영업실적, 보너스, 지급액

```
        --DEFAULT INITIAL VALUE
        --표준 SQL은 변수 설정 불가

SELECT      EMPLOYEE_ID         AS      사원번호,
            EMP_NAME            AS      사원명,
                -- FIRST_NAME||' '||LAST_NAME AS 사원명,
            SALARY AS 본봉,
            COMMISSION_PCT  AS      영업실적,
            NVL(ROUND(SALARY * COMMISSION_PCT*0.3),0)       AS      보너스,
                -- ROUND() 소수점 첫째 자리 반올림 /TRUNC --> 쓰는 것 권하지 않음
                -- NVL을 쓰면, 영업실적 없을 때 그냥 본봉만 나옴
            SALARY + NVL(ROUND(SALARY * COMMISSION_PCT*0.3),0)  AS      지급액
                -- 변수를 못 써서 이것을 다시 써줘야 함
FROM        HR.EMPLOYEES;
            -- NULL 값이 연산에 사용되면 볼 것도 없이 NULL 값으로 반환됨
```

3. 논리 연산자

- 두 개 이상의 관계식을 연결(AND, OR)하거나 반전(NOT) 결과 반환

입력		출력	
A	B	OR	AND
0	0	0	0
0	1	1	0
1	0	1	0
1	1	1	1

1) AND

관계식 **AND** 관계식

- 두 개 이상의 관계식을 연결

EX)

1. 상품테이블(PROD)에서 판매가격이 30만원 이상이고 적정재고가 5개 이상인 제품의 제품번호, 제품명, 매입가, 판매가를 조회하시오.

```
SELECT      PROD_ID             AS 제품번호,
            PROD_NAME           AS 제품명,
            PROD_COST           AS 매입가,
            PROD_PRICE          AS 판매가
FROM        PROD
WHERE       PROD_PRICE >= 300000
AND         PROD_PROPERSTOCK >= 5
ORDER BY    4;
```

2. 매입테이블(BUYPROD)에서 매입일이 2020년 1월이고 매입수량이 10개 이상인 매입정보를 조회하시오. Alias는 매입일, 매입상품, 매입수량, 매입금액

```
SELECT      BUY_DATE    AS      매입일,
            BUY_PROD    AS      매입상품,
            BUY_QTY     AS      매입수량,
            BUY_QTY*BUY_COST        AS      매입금액
FROM        BUYPROD
WHERE       BUY_DATE    >=      TO_DATE('20200101')
AND         BUY_DATE    <=      TO_DATE('20200131')
AND         BUY_QTY     >=      10
ORDER BY    1;
```

--문자열은 HIERARCHY가 낮음 날짜-문자열 -> 날짜가 승
--TO_DATE('') 날짜로 변환하세요 () 안에 꼭 '' 문자로 넣어야 함? 숫자는? 안 됨 => 문자를 날짜로 바꾸는 조건임!!
--TO_DATE('20200101')<=BUY_DATE<=TO_DATE('20200131') => 이건 안 됨!!!!!!!

3. 회원테이블에서 연령대가 20대이거나 여성 회원을 조회하시오.
   Alias는 회원번호, 회원명, 주민번호, 마일리지

```
SELECT      MEM_ID      AS      회원번호,
            MEM_NAME    AS      회원명,
            MEM_REGNO1||'-'||MEM_REGNO2 AS      주민번호,
            TRUNC(EXTRACT(YEAR FROM SYSDATE) - EXTRACT(YEAR FROM MEM_BIR),-1)   AS      연령대,
            MEM_MILEAGE     AS      마일리지
FROM        MEMBER
WHERE       TRUNC(EXTRACT(YEAR FROM SYSDATE) - EXTRACT(YEAR FROM MEM_BIR),-1)   =       20
OR          SUBSTR(MEM_REGNO2,1,1)      IN      ('2','4');
```

--SUBSTR() 문자 추출 IN() ~안에 포함됨 /또는/
--SUBSTR(MEM_REGNO02,1,1)='2'
--SUBSTR(MEM_REGNO02,1,1)='4'
--SUBSTR(MEM_REGNO02,1,1)='2' OR SUBSTR(MEM_REGNO02,1,1)='4'
--TRUNC(,-1) == 일의 자리를 버려라 => 20대 추출

4. 회원테이블에서 연령대가 20대이거나 여성 회원이면서 마일리지가 2000이상인 회원을 조회하시오.
   Alias는 회원번호, 회원명, 주민번호, 마일리지

```
SELECT      MEM_ID      AS      회원번호,
            MEM_NAME    AS      회원명,
            MEM_REGNO1||'-'||MEM_REGNO2 AS      주민번호,
```

```
        TRUNC(EXTRACT(YEAR FROM SYSDATE) - EXTRACT(YEAR FROM MEM_BIR),-1)   AS      연령대,
        MEM_MILEAGE     AS      마일리지
FROM    MEMBER
WHERE   TRUNC(EXTRACT(YEAR FROM SYSDATE) - EXTRACT(YEAR FROM MEM_BIR),-1)   =       20
OR      SUBSTR(MEM_REGNO2,1,1)      IN      ('2','4')
AND   MEM_MILEAGE     >=      2000;
```

5. 키보드로 년도를 입력받아 윤년과 평년을 판단하시오.
   윤년 : 4의 배수이면서 100의 배수가 아니거나, 또는 400의 배수가 되는 년도

```
ACCEPT      P_YEAR      PROMPT      '년도입력: '
DECLARE
V_YEAR      NUMBER:=TO_NUMBER('&P_YEAR');
V_RES       VARCHAR2(100);
BEGIN
IF      (MOD(V_YEAR,4)=0 AND   MOD(V_YEAR,100)!=0)      OR      (MOD(V_YEAR,400)=0)      THEN-- MOD : 나머지 계산
V_RES:=TO_CHAR(V_YEAR)||'년도는 윤년입니다.';
ELSE
V_RES:=TO_CHAR(V_YEAR)||'년도는 평년입니다.';
END   IF;
DBMS_OUTPUT.PUT_LINE(V_RES);
END;
```

4. 기타 연산자

- 오라클에서 제공하는 기타 연산자는 IN, ANY, SOME, ALL, EXISTS, BETWEEN, LIKE가 있음

1) IN

expr IN(값1, 값2,..., 값n);

```
=>      expr    =       값1
OR      expr    =       값2
OR      :
OR      expr    =       값n
```

- IN 연산자에는 '='(Equal to) 기능이 내포

- IN 다음 '( )' 안에 기술된 값 중 어느 하나와 일치하면 전체 결과가 참(TRUE)을 반환
- IN 연산자는 '=ANY', '=SOME'으로 치환 가능('ANY', 'SOME'이 더 다양한 범위로 쓸 수 있음)
    - ANY와 SOME에는 같다라는 기호가 붙어야 IN 연산자로 치환할 수 있음
- IN 연산자는 OR 연산자로 치환 가능
- IN 연산자는 불연속적인 값이나 불규칙한 값을 비교할 때 주로 사용
    - 연속적인 값은 보통 BETWEEN 사용

EX)

사원테이블에서 부서번호가 20, 50, 60, 100번에 속한 사원들을 조회하시오.

Alias는 사원번호, 사원명, 부서번호, 입사일

(OR 연산자 사용)

```
SELECT      EMPLOYEE_ID      AS    사원번호,
            EMP_NAME         AS    사원명,
            DEPARTMENT_ID    AS    부서번호,
            HIRE_DATE        AS    입사일
FROM        HR.EMPLOYEES
WHERE       DEPARTMENT_ID    =     20
OR          DEPARTMENT_ID    =     50
OR          DEPARTMENT_ID    =     60
OR          DEPARTMENT_ID    =     100
ORDER BY    3;
```

(IN 연산자 사용)

```
SELECT      EMPLOYEE_ID      AS    사원번호,
            EMP_NAME         AS    사원명,
            DEPARTMENT_ID    AS    부서번호,
            HIRE_DATE        AS    입사일
FROM        HR.EMPLOYEES
WHERE       DEPARTMENT_ID    IN    (20, 50, 60, 100)
ORDER BY    3;
```

(ANY 연산자 사용)

```
SELECT      EMPLOYEE_ID      AS    사원번호,
            EMP_NAME         AS    사원명,
            DEPARTMENT_ID    AS    부서번호,
            HIRE_DATE        AS    입사일
```

```
FROM        HR.EMPLOYEES
WHERE       DEPARTMENT_ID    =ANY(20, 50, 60, 100)
-- WHERE    DEPARTMENT_ID    =SOME(20, 50, 60, 100)
ORDER BY    3;
```

2) ANY(SOME) 연산자

expr 관계 연산자ANY|SOME(값1,...값n);

- 관계연산자 필수! **<>=** 기술해줘야 함
- expr의 값이 () 안의 값 중 어느 하나와 제시된 관계연산자를 만족하면 전체가 참(TRUE)을 반환함
- IN 연산자와 비슷한 기능 제공
- ANY와 SOME은 완벽하게 내부적으로 동일한 기능을 가짐
- 가장 작은 값을 기준으로 함

EX)
사원테이블에서 부서번호 60번 부서에 속한 사원들의 급여 중 가장 적은 급여보다 더 많은 급여를 받는 사원들을 조회하시오.
Alias는 사원번호, 사원명, 급여, 부서번호이며 급여가 적은 사람부터 출력하시오.

```
SELECT      EMPLOYEE_ID      AS    사원번호,
            EMP_NAME         AS    사원명,
            SALARY           AS    급여,
            DEPARTMENT_ID    AS    부서번호
FROM        HR.EMPLOYEES
WHERE       SALARY >ANY (SELECT SALARY FROM HR.EMPLOYEES WHERE DEPARTMENT_ID=60)
        -- 행의 수가 같아야 비교 가능함 ORA-01428
        -- 다중행 연산자
AND   DEPARTMENT_ID!=60
ORDER BY    3;

SELECT      SALARY
FROM        HR.EMPLOYEES
WHERE       DEPARTMENT_ID=60;
        -- 4200보다 많은 사람 알아보기!
        -- 알려지지 않은 값을 갖고 뭔가를 비교하는 방법 SUBQUERY
```

-- 매입 : 내가 물건을 팔려고 다른 곳에서 사오는 행위 BUYPROD - 거래처 정보가 있어야 하는데 없음 -> 알려면 JOIN 써랏
-- CART : 여기서 누가 사갔나, 날짜/순번(장바구니 번호), 뭘 사갔나, 얼마나 사갔나!

2020년 4월 판매된 상품 중 매입되지 않은 상품을 조회하시오.
Alias는 상품코드이다.

```
SELECT      DISTINCT    CART_PROD AS    상품코드
FROM        CART
WHERE       CART_NO LIKE '202004%' --WHERE절이 거짓이면 SELECT 수행 안 함
AND   NOT   CART_PROD =ANY(SELECT DISTINCT BUY_PROD FROM BUYPROD WHERE BUY_DATE >= '20200401' AND BUY_DATE <=
'20200430')
```

3) ALL 연산자

expr ALL(값1,...값n);

- expr의 값이 주어진 '값1' ~ '값n'의 모든 값과 관계연산을 수행한 결과가 참이면 WHERE 절의 결과 TRUE로 반환
- ANY(SOME)은 가장 작은 값을 기준으로 하고, ALL은 가장 큰 값을 기준으로 함
- ALL과 =은 절대 같이 안 쓰임
    - =을 쓴다고 오류가 나는 것은 아니지만 논리적으로 맞지 않음

EX)
사원테이블에서 부서번호 60번 부서에 속한 사원들의 급여 중
가장 적은 급여보다 더 많은 급여를 받는 사원들을 조회하시오.
Alias는 사원번호, 사원명, 급여, 부서번호이며 급여가 적은 사람부터 출력하시오.

```
SELECT      EMPLOYEE_ID     AS  사원번호,
            EMP_NAME        AS  사원명,
            SALARY          AS  급여,
            DEPARTMENT_ID   AS  부서번호
FROM        HR.EMPLOYEES
WHERE       SALARY >ALL(9000, 6000, 4800, 4200) --> 9000보다 큼 / 60번 부서는 없음! 이미 탈라쿠
ORDER BY    3;
```

4) LIKE 연산자

expr LIKE 패턴문장열;

- 패턴을 정의하여 패턴 비교를 수행 → 문자열 비교
- 패턴구성에는 '%'와 '_'의 와일드카드(패턴문자) --> 문자열만 됨! 숫자랑 날짜 하지 마!!!!
- '%' : '%'가 사용된 위치 이후의 모든 문자열과 대응됨(공백도 됨)

EX) '김%' : '김'으로 시작하는 모든 단어와 참(TRUE)을 반환
'%김' : '김'으로 끝나는 모든 문자열은 참(TRUE)을 반환
'%김%' : 문자열 내부에 '김'이 존재하면 참(TRUE)을 반환
'_' : '_'이 사용된 위치에서 하나의 문자와 대응됨
EX) '김_' : 두 글자로 구성되고 첫 글자가 '김'이면 참(TRUE)을 반환
'_김' : 두 글자로 구성되고 '김'으로 끝나면 참(TRUE)을 반환
'_김_%' : 세 글자로 구성되고 중간 글자가 '김'이면 참(TRUE)을 반환

- 많은 결과를 반환하기 때문에(참이 되는 경우가 많이 발생)
- 방대한 자료를 저장하고 있는 경우 사용 빈도수가 많으면 검색효율이 떨어짐

EX)
회원테이블에서 거주지가 '대전'인 회원들을 조회하시오.
Alias는 회원번호, 회원명, 주소

```
SELECT      MEM_ID                      AS      회원번호,
            MEM_NAME                    AS      회원명,
            MEM_ADD1||' '||MEM_ADD2 AS      주소
FROM        MEMBER
WHERE       MEM_ADD1   LIKE   '대전%';
```

장바구니테이블에서 2020년 6월에 판매된 상품을 조회하시오.
Alias는 상품번호

```
SELECT      DISTINCT    CART_PROD AS      상품번호
FROM        CART
WHERE       CART_NO     LIKE   '202006%' -- CART_NO는 문자열 그래서 LIKE 사용 가능
ORDER BY    1;
```

매입테이블에서 2020년 6월에 매입된 상품을 조회하시오. -- DATE 타입이라 LIKE는 사용 안 됨!!!!!
Alias는 상품번호

```
SELECT      BUY_PROD   AS      상품번호
FROM        BUYPROD
WHERE       BUY_DATE>=TO_DATE('20200601')
AND         BUY_DATE<=TO_DATE('20200630');
-- WHERE BUY_DATE BETWEEN TO_DATE('20200601') AND TO_DATE('20200630');
-- 날짜일 때는 BETWEEN을 좀 써라!!!!어휴
-- 문자열로 바꾸지 말고! 왜냐하면 날짜포맷이 정해져있어서 일일이 확인해야 함
-- 라이크 절대 문자해ㅜㅜㅜ
```

5) BETWEEN 연산자

expr BETWEEN 값1 AND 값2;

- 범위를 정한 자료를 비교
- expr의 값이 '값1'에서 '값2' 사이에 존재하는 값이면 참을 반환
- BETWEEN 연산자는 AND 연산자로 바꾸어 사용할 수 있음 -- AND로 쓰면 길어짐
- 모든 데이터 타입에 사용 가능 -- 문자열은 LIKE 등등을 사용 / 그 외 날짜 숫자는 BETWEEN을 많이 사용함

EX)
상품테이블에서 판매가격이 10만원~20만원 사이의 상품을 조회하시오.
Alias 상품번호, 상품명, 판매가격

```
SELECT      PROD_ID             AS    상품번호,
            PROD_NAME           AS    상품명,
            PROD_PRICE          AS    판매가격
FROM        PROD
WHERE       PROD_PRICE>=100000      AND    PROD_PRICE<=200000
ORDER BY    3;
```

(BETWEEN으로 사용하기)

```
SELECT      PROD_ID             AS    상품번호,
            PROD_NAME           AS    상품명,
            PROD_PRICE          AS    판매가격
FROM        PROD
WHERE       PROD_PRICE          BETWEEN     100000          AND    200000
ORDER BY    3;
```

사원테이블에서 2005년~2007년 사이에 입사한 사원들을 조회하시오.
Alias 사원번호, 사원명, 부서코드, 직무코드, 입사일

```
SELECT      EMPLOYEE_ID         AS    사원번호,
            EMP_NAME            AS    사원명,
            DEPARTMENT_ID       AS    부서코드,
            JOB_ID              AS    직무코드,
            HIRE_DATE           AS    입사일
FROM        HR.EMPLOYEES
WHERE       HIRE_DATE    BETWEEN     '20050101'      AND    '20071231'
```

```
ORDER BY    5;
```

상품테이블에서 상품의 분류코드가 'P100'번대와 'P300'번대의 상품들을 조회하시오.
Alias 상품번호, 상품명, 분류코드

```
SELECT      PROD_ID             AS      상품번호,
            PROD_NAME           AS      상품명,
            PROD_LGU            AS      분류코드
FROM        PROD
WHERE       PROD_LGU    BETWEEN     'P100'  AND    'P102'
OR          PROD_LGU    BETWEEN     'P301'  AND    'P302'
ORDER BY    3;
```