

# PL/SQL

## ■ PL/SQL 이란?

1. Procedural Language extension to SQL 의 약자  
: SQL 에 없는 절차적 프로그래밍 기능을 제공하여 SQL 단점을 보완해 준다.
2. Oracle 에서 제공하는 Oracle 전용 SQL 데이터베이스 언어  
: 다른 데이터베이스 제품에서는 사용할 수 없다.
3. SQL 에서 프로그래밍 언어의 역할을 함  
: 변수를 만들고 조건문, 반복문 등을 실행할 수 있고, 커서를 사용하여 여러 행을 검색할 수 있다.

## ■ PL/SQL 유형

PL/SQL 유형은 Anonymous Block (익명 블록), Procedure (프로시저), Function(함수) 이렇게 3가지로 나뉜다.

- **PL/SQL 블록(Anonymous Block)** : 이름없는 블록이라 불리며 간단한 block 수행시 사용됨.
- **프로시저(Procedure)** : 지정된 특정 처리를 실행하는 서브 프로그램의 한 유형으로 단독으로 실행되거나 다른 프로시저나 다른 툴(Oracle Developer, sqlplus, sqlgate) 등에 호출되어 실행됨.
- **함수(Function)** : Procedure 와 수행되는 결과가 유사하나 값 반환 여부에 따라 차이가 있음.

	Anonymous Block (익명블록)	Procedure(프로시저)	Function(함수)
객체로 저장	X	O	O
값 반환	X	X	O
이름 지정 여부	X	O	O
파라미터 사용 여부	X	O	O
타 응용프로그래밍 호출여부	X	O	O

## ■ PL/SQL 블록 문법

```
DECLARE
    [ 선언부 ]
BEGIN
    [ 실행부 ]
EXCEPTION
    [ 예외처리부 ]
END;
/
```

-- DECLARE 로 시작 / 선언부 : 변수나 상수들을 정의(선택)  
-- BEGIN 으로 시작 / 실행부 : 실제 실행될 PL/SQL 이 들어감, 제어문, 반복문,  
함수 정의 등 로직을 기술 (필수)  
-- EXCEPTION 으로 시작 / 예외처리부 : 예외 발생 해결 구문을 적는 부분(선택)

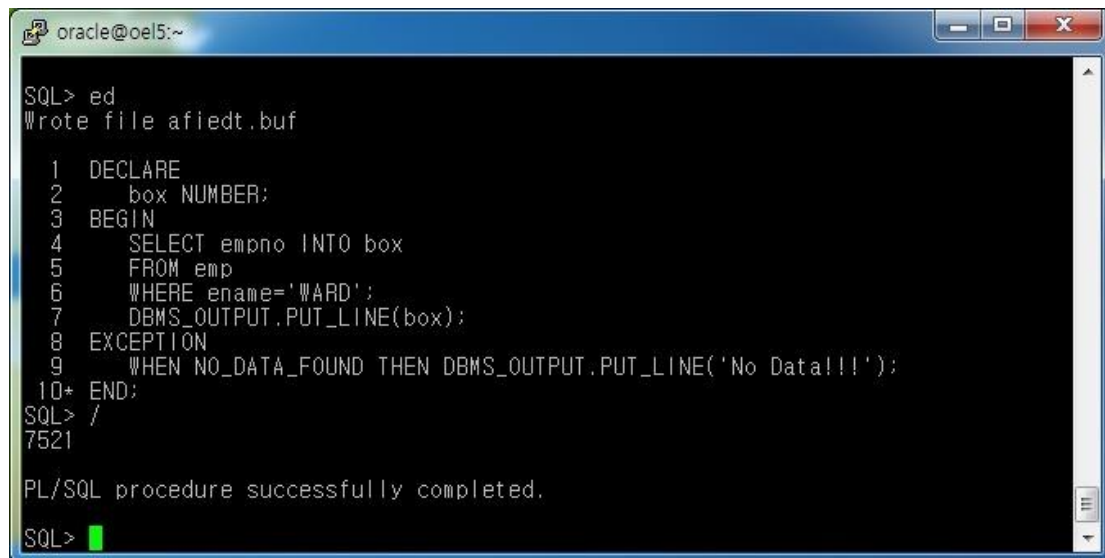
- 위 단락을 PL/SQL Block 라고 부르기도 한다.

## ■ PL/SQL 프로그램 작성 요령

-- PL/SQL 블록 내에서는 한 문장을 종료할 때마다 세미콜론(;) 사용함  
-- END 뒤에 세미콜론(;)을 사용하여 하나의 블록이 끝났다는 것을 명시함  
-- PL/SQL 블록의 작성은 메모장과 같은 편집기를 통해 sql 파일로 저장할 수도  
있고, SQL> 프롬프트에서 바로 작성할 수도 있다.  
-- SQL\*PLUS 환경에서는 DECLARE 나 BEGIN 이라는 키워드로 PL/SQL 블록이  
시작하는 것을 알 수 있음.  
-- 단일행 주석은 -- 이고, 여러 행 주석은 /\*\*/이다.  
-- 쿼리문을 수행하기 위해서 /가 반드시 입력되어야 하며, PL/SQL 블록은 행에  
/가 있으면 종결된 것으로 간주함

## ■ PL/SQL 샘플

```
DECLARE
    box  NUMBER;
BEGIN
    SELECT empno
    INTO box
    FROM emp
    WHERE ename='WARD';
    DBMS_OUTPUT.PUT_LINE(box);
EXCEPTION
    WHEN NO_DATA_FOUND THEN DBMS_OUTPUT.PUT_LINE('No Data!!!');
END;
/
```



```
SQL> ed
Wrote file afiedt.buf

 1  DECLARE
 2    box NUMBER;
 3  BEGIN
 4    SELECT empno INTO box
 5    FROM emp
 6    WHERE ename='WARD';
 7    DBMS_OUTPUT.PUT_LINE(box);
 8  EXCEPTION
 9    WHEN NO_DATA_FOUND THEN DBMS_OUTPUT.PUT_LINE('No Data!!!');
10* END;
SQL> /
7521

PL/SQL procedure successfully completed.
SQL> █
```

-- Oracle 이 샘플로 제공하는 scott 계정으로 테스트 해본다.

위의 PL/SQL 을 간단히 설명하면 직원 이름에 WARD 가 있으면 그의 사번 (empno) 를 조회해 와서 box 변수에 넣은 후 box 변수에 있는 값을 DBMS\_OUTPUT.PUT\_LINE() 함수로 출력하라는 구문이다.

단, 해당 값이 없으면 'No Data!!!' 라고 예외 구문을 발생 시키도록 되어있다.

결과를 보려면 다음 설정이 필요하다.

## PL/SQL 사용을 위한 기본 설정 및 확인 방법

### ■ SERVEROUTPUT 설정

```
SQL> SET SERVEROUTPUT ON;  
SQL> /
```

기본적으로 PL/SQL 은 결과물을 보여주지 않는다.

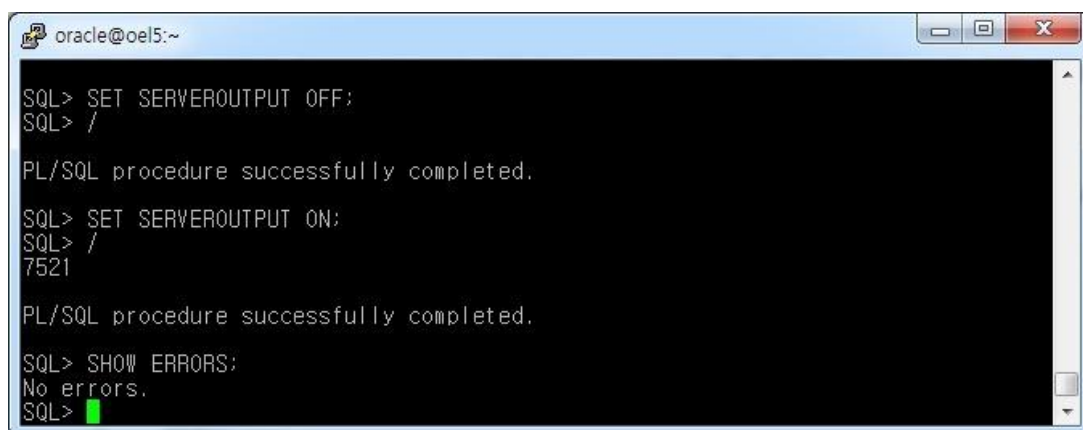
결과물을 보고 싶다면 SERVEROUTPUT 설정을 ON 으로 설정해 주어야 한다.

### ■ Error

```
SQL> SHOW ERRORS;  
SQL> /
```

PL/SQL 은 오류를 안 보여준다.

그렇기 때문에 위 명령어로 확인한다.

A screenshot of a terminal window titled 'oracle@oel5:~'. The terminal shows a sequence of SQL\*Plus commands and their outputs. The commands are: 'SET SERVEROUTPUT OFF;', followed by a slash to execute; 'SET SERVEROUTPUT ON;', followed by a slash to execute; and 'SHOW ERRORS;', followed by a slash to execute. The outputs are: 'PL/SQL procedure successfully completed.' for the first two commands, and 'No errors.' for the third command. A green cursor is visible at the end of the last command line.

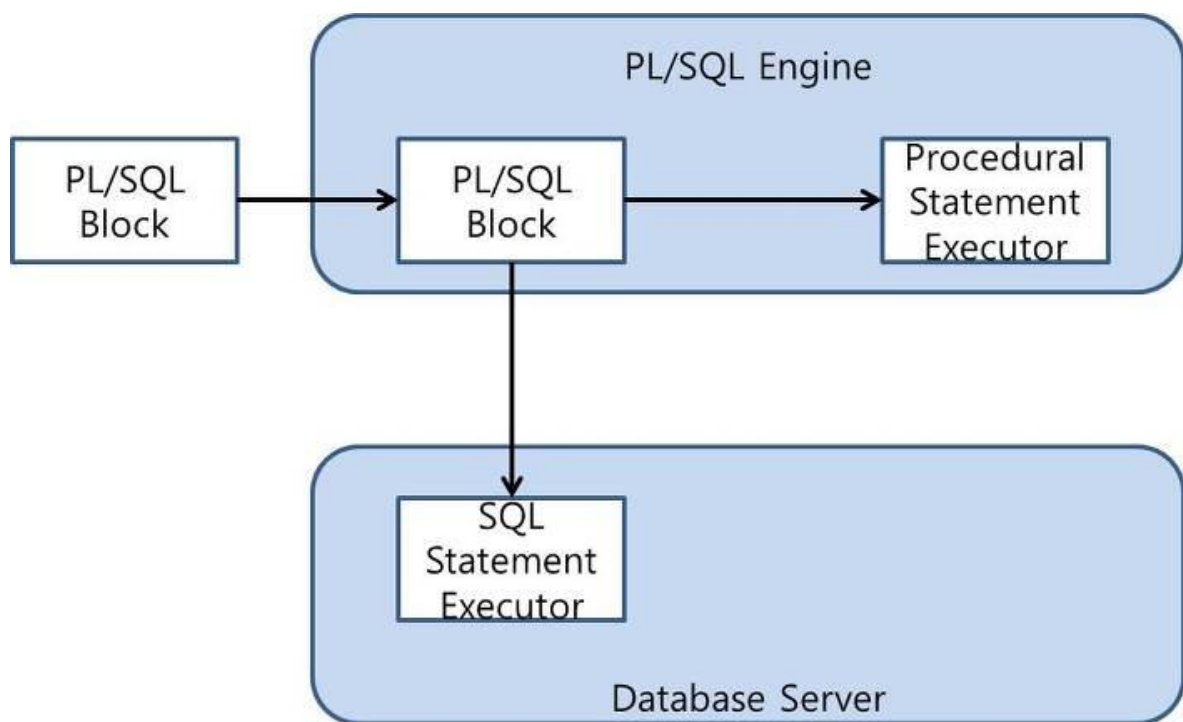
```
oracle@oel5:~  
SQL> SET SERVEROUTPUT OFF;  
SQL> /  
PL/SQL procedure successfully completed.  
SQL> SET SERVEROUTPUT ON;  
SQL> /  
7521  
PL/SQL procedure successfully completed.  
SQL> SHOW ERRORS;  
No errors.  
SQL> █
```

맨 위의 예제를 실행하고 SERVEROUTPUT 옵션의 확인과 ERRORS 확인을 해본 것이다.

OFF 옵션일 때는 결과가 안 나오지만 ON 옵션일 때 7521 이라는 결과 값이 나왔다.

그리고 Error 확인 시 Error 는 없는 것으로 나오고 있다.

## PL/SQL 의 실행 구조



1. PL/SQL Block 실행
2. PL/SQL 엔진에서 PL/SQL 부분과 일반 SQL 부분이 분리
3. SQL 부분을 Database Server 가 처리
4. Database Server 에서 처리된 SQL 결과를 PL/SQL Engine 로 전달
5. PL/SQL Engine 에서 처리된 SQL 결과를 바탕으로 나머지 작업 수행

## PL/SQL 의 변수 선언과 대입

### ■ 변수 선언

변수명 [CONSTANT] 자료형(바이트크기) [NOT NULL] [:= 초기값];

-- CONSTANT : 값 변경 못하게 함(상수)  
-- NOT NULL : 반드시 값 지정, NULL 사용 못 함  
-- 초기값에 계산식 사용 가능함

```
예>  
DECLARE  
EMPNO NUMBER(4);  
ENAME VARCHAR2(10);
```

### ■ 변수에 값 대입

변수명 := 값;

```
예>  
BEGIN  
EMPNO := 1001;  
ENAME := '김사랑';  
DBMS_OUTPUT.PUT_LINE('      사번      이름');  
DBMS_OUTPUT.PUT_LINE('-----');  
DBMS_OUTPUT.PUT_LINE('      ' || EMPNO || '      ' || ENAME);  
END;  
/
```

### ■ 변수의 자료형

- 기본 자료형  
문자 -> varchar2, blob, clob  
숫자 -> number  
날짜 -> date

부울(Boolean) -> boolean := true, false, null

- **Composite datatype**

레코드(Record)

컬렉션(Collection)

## ■ 변수의 종류

PL/SQL 변수는 일반(스칼라)변수, 상수, %TYPE, %ROWTYPE, 레코드(Record), 컬렉션 (Collection)등이 있다.

▶ 스칼라 변수(일반 변수) : 기존 SQL 의 자료형과 유사함

사용방법 : 변수명 데이터타입(바이트) [:= 초기값];

사용 예>

```
ENO NUMBER(4);  
ENME VARCHAR2(10);
```

▶ 상수 : 일반변수와 유사하나 CONSTANT 라는 키워드가 자료형 앞에 붙고 선언시에 값을 할당해 주어야 함.

사용방법 : 변수명 CONSTANT 데이터타입 := 초기값;

사용 예>

```
user_name CONSTANT VARCHAR2(20) := '홍길동';
```

▶ 참조(REFERENCE) 변수 : 이전에 선언된 다른 변수 또는 테이블의 컬럼 자료형에 맞추어 변수를 선언하기 위한 변수

- **%TYPE 속성을 사용한 참조 변수**

사용방법 : 변수명 테이블.컬럼명%TYPE;

사용 예>

%TYPE 으로 특정 테이블의 컬럼의 자료형을 참조하는 레퍼런스 변수 선언하기

```
VEMPNO EMP.EMPNO%TYPE;  
VENAME EMP.ENAME%TYPE;
```

-- %TYPE 속성을 사용하여 선언한 VEMPNO 변수는 해당 테이블의 해당 컬럼의 자료형과 크기를 그대로 참조해서 만들어짐  
-- 컬럼의 자료형이 변경되면 참조하는 레퍼런스 변수의 자료형과 크기도 자동으로 반영되므로 별도로 수정할 필요가 없다.

-- %TYPE 이 컬럼 단위 참조라면, 행(ROW) 단위로 참조하는 %ROWTYPE 속성도 있음. 테이블을 참조하는 변수가 됨. 테이블을 대신하는 변수로 생각해도 됨.

```
VEMP EMP%ROWTYPE;
```

-- 특정 테이블의 컬럼 개수나 데이터 형식을 몰라도 지정할 수 있음.  
-- SELECT 문장으로 행을 검색할 때 유리함.

### ● %ROWTYPE 속성을 사용한 참조 변수

%TYPE 과 유사하게 참조할 테이블의 컬럼 데이터 타입을 자동적으로 가져오나 1 개의 컬럼이 아니라 여러 개의 컬럼값을 자동으로 가져오는 역할을 함

사용방법 : 변수명 테이블%ROWTYPE;

사용 예>

```
userInfo EMP%ROWTYPE;
```

▶ 레코드(RECORD) 변수 : %ROWTYPE 이 참조할 테이블의 컬럼 데이터타입을 자동으로 가져오는 반면 RECORD 는 직접적으로 컬럼 타입을 지정해 줄 수 있다

사용방법 : TYPE 레코드타입명 IS RECORD (변수명 변수타입, ... );  
레코드 객체 레코드타입명;

▶ 컬렉션(COLLECTION) 변수 : 컬렉션은 배열과 같은 형태를 가지고 있으며, 컬렉션을 정의해서 생성한 뒤 이를 변수로 선언해서 사용이 가능함.



컬렉션의 종류는 varray, 중첩테이블, Associative array 3 가지로 나뉜다.

	Varray	중첩 테이블	Associative array
형태	Array 와 유사	Vector 와 유사	HashMap 과 유사
배열 크기 설정 유무	O	X	X
참조순서	순서대로	상관없음	상관없음

사용방법 :

**varray :** TYPE 배열변수명 IS VARRAY OF 자료형(정수) ;

**중첩테이블 :** TYPE 배열변수명 IS TABLE OF 자료형(정수) ;

**Associative array :**

TYPE 배열변수명 IS TABLE OF 자료형(갯수) INDEX BY 키타입(바이트);

Associative array 에 들어가는 키 데이터타입 유형은 2 가지임.

- 숫자일 경우 :

BINARY INTEGER(이진 정수로 처리)

또는 PLS\_INTEGER(원시 계산 방식으로 처리)

- 문자일 경우 : VARCHAR2 또는 하위 유형중 하나

## ■ PL/SQL 에서 SELECT 문

SQL 에서 사용하는 명령어를 그대로 사용할 수 있으며, SELECT 쿼리 결과로 나온 값을 변수에 할당하기 위해 사용함.

**형식]**

**SELECT** 컬럼명 나열

**INTO** 값 받을 변수명

**FROM** 테이블명

**WHERE** 조건식;

예>

```
SELECT EMPNO, ENAME
INTO ENO, ENM
FROM EMP
WHERE ENAME = 'ALLEN';
```

-- SELECT 절에 컬럼과 값 보관할 변수는 1 대 1 로 대응하므로 개수와 자료형이 일치해야 함.

예제 1> PL/SQL 의 SELECT 문으로 EMP 테이블에서 사번과 이름 조회하기

```
DECLARE
    VEMPNO EMP.EMPNO%TYPE;
    VENAME EMP.ENAME%TYPE;
BEGIN
    SELECT EMPNO, ENAME
    INTO VEMPNO, VENAME
    FROM EMP
    WHERE ENAME = 'ALLEN';
    DBMS_OUTPUT.PUT_LINE('사번    이름');
    DBMS_OUTPUT.PUT_LINE('-----');
    DBMS_OUTPUT.PUT_LINE(VEMPNO || '    ' || VENAME);
END;
/
```

예제 2> 사원번호를 입력 받아서 사원의 사원번호와 이름, 입사일을 출력하시오.

```
SET SERVEROUTPUT ON ;

DECLARE
    VEMPNO EMP.EMPNO%TYPE;
    VENAME EMP.ENAME%TYPE;
    VSAL EMP.SAL%TYPE;
    VHIREDATE EMP.HIREDATE%TYPE;
BEGIN
    SELECT EMPNO, ENAME, SAL, HIREDATE
```

```

        INTO VEMPNO, VENAME, VSAL, VHIREDATE
    FROM EMP
    WHERE EMPNO = '&EMPNO';          --유동적으로 값을 넣어주는 기능!!!
    SYS.DBMS_OUTPUT.PUT_LINE(VENAME || ' ' || VSAL || ' ' || VHIREDATE);
END;
/

```

## ■ PL/SQL 에서 NULL 문

PL/SQL 에서 제공하는 NULL 은 컬럼이나 변수에서 사용되는 것과 비슷한 개념으로 사용됨.

```

CASE grade
    WHEN 'A' THEN
        DBMS_OUTPUT.PUT_LINE('Excellent');
    WHEN 'B' THEN
        DBMS_OUTPUT.PUT_LINE ('Good');
    ELSE NULL;
END CASE;

```

## PL/SQL 의 선택문

모든 문장들은 기술한 순서대로 순차적으로 수행됨  
문장을 선택적으로 수행하려면 IF 문을 사용하면 됨

### ■ IF ~ THEN ~ END IF 문

조건에 따라 어떤 명령을 선택적으로 처리하기 위한 가장 기본 구문임.

```
IF 조건 THEN
    조건을 만족할 경우 처리구문;
END IF;
```

#### 예제> 부서번호로 부서명 알아내기

```
DECLARE
    VEMPNO EMP.EMPNO%TYPE;
    VENAME EMP.ENAME%TYPE;
    VDEPTNO EMP.DEPTNO%TYPE;
    VDNOME VARCHAR2(20) := NULL;
BEGIN
    SELECT EMPNO, ENAME, DEPTNO
    INTO VEMPNO, VENAME, VDEPTNO
    FROM EMP
    WHERE EMPNO = &EMPNO;

    IF (VDEPTNO = 10) THEN
        VDNOME := 'ACCOUNTING';
    END IF;
    IF (VDEPTNO = 20) THEN
        VDNOME := 'RESEARCH';
    END IF;
    IF (VDEPTNO = 30) THEN
        VDNOME := 'SALES';
    END IF;
```

```

        IF (VDEPTNO = 40) THEN
            VDNNAME := 'OPERATIONS';
        END IF;

        DBMS_OUTPUT.PUT_LINE('사번    이름    부서명');
        DBMS_OUTPUT.PUT_LINE('-----');
        DBMS_OUTPUT.PUT_LINE(VEMPNO || '    ' || VENNAME || '    ' || VDNNAME);
    END;
/

```

## ■ IF ~ THEN ~ ELSE ~ END IF 문

```

IF 조건 THEN
    조건을 만족할 경우 처리구문;
ELSE
    조건을 만족하지 않을 경우 처리구문;
END IF;

```

### 예제> 연봉을 구하는 예제

```

DECLARE
    VEMP EMP%ROWTYPE;
    ANNSAL NUMBER(7, 2);
BEGIN
    SELECT * INTO VEMP
    FROM EMP
    WHERE ENAME = '&ENAME';

    IF (VEMP.COMM IS NULL) THEN
        ANNSAL := VEMP.SAL * 12;
    ELSE
        ANNSAL := VEMP.SAL * 12 + VEMP.COMM;
    END IF;

    DBMS_OUTPUT.PUT_LINE('사번    이름    연봉');

```

```

        DBMS_OUTPUT.PUT_LINE('-----');
        DBMS_OUTPUT.PUT_LINE(VEMP.EMPNO || ' ' || VEMP.ENAME || ' ' || ANNSAL);
END;
/

```

**실습>** 특정 사원이 커미션을 받는지 안 받는지를 구분해서 출력하시오.

출력 예: 사번 7788 은 SCOTT 사원이고 커미션을 받지 않습니다.

출력 예: 사번 7654 은 MARTIN 사원이고 커미션을 1400 받습니다.

```

DECLARE
    VEMPNO EMP.EMPNO%TYPE;
    VCOMM EMP.COMM%TYPE;
    VENAME EMP.ENAME%TYPE;
BEGIN
    SELECT EMPNO, ENAME, COMM
    INTO VEMPNO, VENAME, VCOMM
    FROM EMP
    WHERE EMPNO = &EMPNO;

    IF (VCOMM IS NULL OR VCOMM = 0) THEN
        DBMS_OUTPUT.PUT_LINE('사번' || VEMPNO || '은 ' || VENAME || '사원이고
        커미션을 받지 않습니다. ');
    ELSE
        DBMS_OUTPUT.PUT_LINE('사번' || VEMPNO || '은 ' || VENAME || '사원이고 ' ||
        VCOMM || '을 받습니다. ');
    END IF;
END;
/

```

## ■ IF ~ THEN ~ ELSEIF ~ ELSE ~ END IF 문

```

IF 조건 1 THEN
    조건 1 을 만족할 경우 처리구문 1;
ELSIF 조건 2 THEN
    조건 2 를 만족할 경우 처리구문 2;
ELSIF 조건 3 THEN

```

```

        조건 3 를 만족할 경우 처리구문 3;
ELSE
        모든 조건을 만족하기 않을 경우 처리구문;
END IF;

```

### 예제> 부서번호로 부서명 알아내기

```

DECLARE
    VEMP EMP%ROWTYPE;
    VDNAM VARCHAR2(14);
BEGIN
    SELECT * INTO VEMP
    FROM EMP
    WHERE ENAME = '&ENAME';

    IF (VEMP.DEPTNO = 10) THEN
        VDNAM := 'ACCOUNTING';
    ELSIF (VEMP.DEPTNO = 20) THEN
        VDNAM := 'RESEARCH';
    ELSIF (VEMP.DEPTNO = 30) THEN
        VDNAM := 'SALES';
    ELSIF (VEMP.DEPTNO = 40) THEN
        VDNAM := 'OPERATIONS';
    END IF;

    DBMS_OUTPUT.PUT_LINE('사번      이름      부서명');
    DBMS_OUTPUT.PUT_LINE('-----');
    DBMS_OUTPUT.PUT_LINE(VEMP.EMPNO || ' ' || VEMP.ENAME || ' ' || VDNAM);
END;
/

```

**실습>** score 변수에 85 를 대입하고, 점수에 대한 학점을 출력하시오.

출력 예 : 당신의 SCORE 는 85 점이고,Grade 는 B 입니다.

```

DECLARE
    score int;
    grade varchar2(2);
BEGIN
    score := &score;

```

```

IF score >= 90 THEN
    Grade := 'A';
ELSIF score >= 80 THEN
    Grade := 'B';
ELSIF score >= 70 THEN
    grade := 'C';
ELSIF score >= 60 THEN
    grade := 'D';
ELSE grade := 'F';
END IF;

DBMS_OUTPUT.PUT_LINE('당신의 SCOR 는 ' || score || '점이고,' || 'Grade 는 ' || grade || '
입니다.');
```

END;

/

## ■ CASE 문

자바의 switch 문과 같음.

### 예제 > 부서번호로 부서명 알아내기

```

DECLARE
    vempno EMP.EMPNO%TYPE;
    vename EMP.ENAME%TYPE;
    vdeptno EMP.DEPTNO%TYPE;
    vdname VARCHAR(20) := null;
BEGIN
    SELECT EMPNO, ENAME, DEPTNO
    INTO vempno, vename, vdeptno
    FROM EMP
    WHERE EMPNO = &EMPNO;

    vdname := CASE vdeptno
                WHEN 10 THEN 'ACCOUNT'
                WHEN 20 THEN 'RESEARCH'
                WHEN 30 THEN 'SALES'
                WHEN 40 THEN 'OPERATIONS'
    END;
```



```
DBMS_OUTPUT.PUT_LINE (VEMPNO || ' ' || VENAME || ' ' || VDEPTNO || ' ' || VDNAME);  
END;  
/
```

## PL/SQL 의 반복문

반복문은 SQL 문을 반복적으로 여러 번 실행하고자 할 때 사용함

반복문의 종류는

- 조건없이 반복 작업을 제공하기 위한 BASIC LOOP 문
- COUNT 를 기본으로 작업의 반복 제어를 제공하는 FOR LOOP 문
- 조건을 기본으로 작업의 반복 제어를 제공하기 위한 WHILE LOOP 문
- LOOP 를 종료하기 위한 EXIT 문

### ■ BASIC LOOP 문

자바의 do ~ while 문과 같은 형태임.

LOOP

반복 실행시킬 문장;

.....;

IF 반복종료조건 THEN

EXIT;

END IF;

또는

EXIT [WHEN 반복종료조건];

END LOOP;

예제> BASIC LOOP 문으로 1 부터 5 까지 출력하기

DECLARE

N NUMBER := 1;

BEGIN

LOOP

DBMS\_OUTPUT.PUT\_LINE(N);

N := N + 1;

IF N > 5 THEN

EXIT;

```
        END IF;  
    END LOOP;  
END;  
/
```

## ■ FOR LOOP 문

FOR LOOP 문에서 카운트용 변수는 자동 선언되므로, 따로 변수 선언할 필요 없음.

카운트 값은 자동으로 1 씩 증가함.

REVERSE 는 1 씩 감소함을 의미함

```
FOR 카운터용 변수 IN [REVERSE] 시작값..종료값 LOOP  
    반복실행할 문장;  
    .....;  
END LOOP;
```

예제> FOR LOOP 문으로 1 부터 5 까지 출력하기

```
DECLARE  
BEGIN  
FOR N IN 1..5 LOOP  
DBMS_OUTPUT.PUT_LINE(N);  
END LOOP;  
END;  
/
```

**실습 1**> 1 에서 10 까지 반복하여 TEST1 테이블에 저장되게 하시오.

**SCOTT 계정에서 TEST1 테이블 생성**

**SQL>** create table test1(bunho number(3), irum varchar2(10));

```
BEGIN  
FOR i IN 1..10 LOOP  
insert into test1 values(i, SYSDATE);
```

```

END LOOP;
END;
/
SQL> select * from test1;
BUNHO IRUM
-----
1      15/06/18
2      15/06/18
3      15/06/18
4      15/06/18
5      15/06/18
6      15/06/18
7      15/06/18
8      15/06/18
9      15/06/18
10     15/06/18
10 개의 행이 선택되었습니다.

```

**실습 2>** 구구단의 홀수단만 출력되게 하시오. (for 문과 if 문 혼합)

```

DECLARE
    RESULT  NUMBER;
BEGIN
    FOR DAN IN 2..9 LOOP
        IF MOD(DAN, 2) = 1 THEN
            FOR N IN 1..9 LOOP
                RESULT := DAN * N;
                DBMS_OUTPUT.PUT_LINE(DAN || ' * ' || N || ' = ' || RESULT);
            END LOOP;
            DBMS_OUTPUT.PUT_LINE(' ');
        END IF;
    END LOOP;
END;
/

```

## ■ WHILE LOOP 문

제어 조건이 TRUE 인 동안만 문장이 반복 실행됨

LOOP 를 실행할 때 조건이 처음부터 FALSE 이면 한번도 수행되지 않을 경우도 있음.

WHILE 반복시킬 조건식 LOOP

반복실행할 문장;

.....;

END LOOP;

**예제>** WHILE LOOP 문으로 1 부터 5 까지 출력하기

DECLARE

N NUMBER := 1;

BEGIN

WHILE N <= 5 LOOP

DBMS\_OUTPUT.PUT\_LINE(N);

N := N + 1;

END LOOP;

END;

/

**실습>** 구구단 2 ~ 9 단에서 결과가 홀수인 것만 출력되게 하시오.

(WHILE LOOP 사용)

DECLARE

RESULT NUMBER;

DAN NUMBER := 2;

SU NUMBER;

BEGIN

WHILE DAN <= 9 LOOP

SU := 1;

WHILE SU <= 9 LOOP

RESULT := DAN \* SU;

IF MOD(RESULT, 2) = 1 THEN

DBMS\_OUTPUT.PUT\_LINE(DAN || ' \* ' || SU || ' = ' || RESULT);

END IF;

SU := SU + 1;

END LOOP;

DBMS\_OUTPUT.PUT\_LINE(' ');

DAN := DAN + 1;

END LOOP;

```
END;  
/
```

## PL/SQL 예외 처리

PL/SQL에서는 실행문에서 발생한 에러를 오라클 PL/SQL 엔진이 오류를 발생시켜 잡을 수 있다.

예외처리 기본문법은 아래와 같으며 예외의 종류는 시스템 오류 예외처리와 프로그래머 정의 예외처리 2 가지로 나뉜다.

### ■ 예외 처리 영역

```
DECLARE  
--선언영역  
BEGIN  
--실행영역  
EXCEPTION  
-- 예외처리 영역  
END;  
/
```

### ■ 예외 처리 구문

```
EXCEPTION  
    WHEN 예외이름 1 THEN 처리문장 1          -- EXCEPTION HANDLER  
    WHEN 예외이름 2 OR 예외이름 3 THEN 처리문장 2  
    WHEN OTHERS THEN 처리문장 3  
END;  
/
```

### ■ 시스템 오류 예외처리

시스템 오류( 예를 들면, 메모리 초과, 인덱스 중복 키 등 )는 오라클이 정의하는 예외로 보통 PL/SQL 실행 엔진이 오류조건을 탐지하여 발생시키는 예외이다. Exception 이름을 아는 경우와 모르는 경우에 대하여 사용하는 방법은 다르다.

### ● Exception 이름을 아는 경우

오라클에서 미리 정의해 놓은 Exception 임

ACCESS_INTO_NULL	초기화되지 않은 오브젝트에 값을 할당하려고 할 때
CASE_NOT_FOUND	CASE 문장에서 ELSE 구문도 없고 WHEN 절에 명시된 조건을 만족하는 것이 없을 경우
COLLECTION_IS_NULL	초기화되지 않은 중첩 테이블이나 VARRAY 같은 컬렉션을 EXISTS 외의 다른 메소드로 접근을 시도할 경우
CURSOR_ALREADY_OPEN	이미 오픈된 커서를 다시 오픈하려고 시도하는 경우
DUP_VAL_ON_INDEX	UNIQUE 인덱스가 설정된 컬럼에 중복 데이터를 입력할 경우
INVALID_CURSOR	허용되지 않은 커서에 접근할 경우 (OPEN 되지 않은 커서를 닫으려고 할 경우)
INVALID_NUMBER	SQL 문장에서 문자형 데이터를 숫자형으로 변환할 때 제대로 된 숫자로 변환되지 않을 경우
LOGIN_DENIED	잘못된 사용자명이나 비밀번호로 접속을 시도할 때
NO_DATA_FOUND	SELECT INTO 문장의 결과로 선택된 행이 하나도 없을 경우

예> UNIQUE INDEX 가 설정된 컬럼에 중복값을 입력한 경우의 예외처리

```
BEGIN
    INSERT INTO EXAM_MEMBERS(MID, PWD, NAME) VALUES('javaking', '111', '자바킹');
EXCEPTION
    WHEN DUP_VAL_ON_INDEX THEN
        DBMS_OUTPUT.PUT_LINE('아이디가 중복되었습니다.');
```

END;  
/

### ● Exception 이름을 모를 경우

오라클에서 지정한 Exception 이름을 모를 경우에는 선언 영역에 프로그램 지시자인 PRAGMA 키워드와 EXCEPTION\_INIT() 함수로 예외를 직접 정의하여 사용함.

EXCEPTION\_INIT() 함수는 컴파일 시간 명령어 또는 예외명과 내부 오류코드를 연관 짓는 프라그마로 컴파일러에게 exception 으로 선언된 식별자를 지정된 오류번호와 연관짓게 한다.

사용방법 :

```
DECLARE
    예외변수 EXCEPTION;
    PRAGMA EXCEPTION_INIT(예외변수명, 에러코드);
BEGIN

EXCEPTION
    WHEN 예외변수명 THEN 처리내용;
```

예>

```
DECLARE
    TOOLONG_NAME EXCEPTION;          -- EXCEPTION 선언
    PRAGMA EXCEPTION_INIT(TOOLONG_NAME, -12899);
    --에러코드와 선언한 EXCEPTION 매핑
BEGIN
    INSERT INTO EXAM_MEMBERS(MID, PWD, NAME)
VALUES('javaking ooooooooooooooooooooooooooooooooooooo', '111', '자바킹');
EXCEPTION
    WHEN TOOLONG_NAME THEN
        DBMS_OUTPUT.PUT_LINE('사용자 아이디 이름이 길이 허용 범위를 벗어났습니다.');
```

```
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('알 수 없는 오류가 발생하였습니다.');
```

```
END;
/
```



실습> EMP 테이블의 ENAME 컬럼에 INSERT 하는 쿼리문이 실행되게 PL/SQL 문을 작성하시오. 단, ENAME 컬럼에 지정된 바이트 크기를 초과한 경우에 대한 예외를 처리하시오.

```
DECLARE
    TOOLONG_NAME EXCEPTION;
    PRAGMA EXCEPTION_INIT(TOOLONG_NAME, -12899);
BEGIN
    INSERT INTO EMP (EMPNO, ENAME)
    VALUES (7777, 'JAVAKINGOOOOOOOOOOOOOOOOO');
EXCEPTION
    WHEN TOOLONG_NAME THEN
        DBMS_OUTPUT.PUT_LINE('ENAME 컬럼에 지정된 바이트 크기를 초과하였습니다.');
```

WHEN OTHERS THEN  
DBMS\_OUTPUT.PUT\_LINE('알 수 없는 예외가 발생하였습니다.');

END;  
/

## ■ 프로그래머 정의 예외

프로그래머가 정의하는 예외로 오라클 지정함수 RAISE\_APPLICATION\_ERROR()를 사용하여 오류코드 -20000 ~ -20999 의 범위 내에서 사용자 정의 예외를 만들거나, RAISE(자바의 throw)를 사용하여 예외를 발생 시킬 수 있다.

사용방법 :

```
RAISE_APPLICATION_ERROR(에러코드, 메세지);
RAISE 예외;
```

- 사용자정의 예외 - 에러코드를 던짐
- -20000 과 -20999 사이에 있어야만 함 (사용하지 않는 코드번호 이용)

예>

```
DECLARE
    V_PWD VARCHAR2(50);
    TOOLONG_NAME EXCEPTION;
    TOOSHORT_PWD EXCEPTION;
```

```

PRAGMA EXCEPTION_INIT(TOOLONG_NAME, -12899);
PRAGMA EXCEPTION_INIT(TOOSHORT_PWD, -20001);
BEGIN
    V_PWD := '&PWD';
    IF LENGTH(V_PWD) < 7 THEN
        --RAISE TOOSHORT_PWD; -- 예외발생
        RAISE_APPLICATION_ERROR(-20001, '비밀번호가 너무 작은 오류');
    END IF;
    INSERT INTO EXAM_MEMBERS(MID, PWD, NAME)
VALUES('javaking ooooooooooooooooooooooooooooooooo', '111', '자바킹');
EXCEPTION
    WHEN DUP_VAL_ON_INDEX THEN
        DBMS_OUTPUT.PUT_LINE('아이디가 중복되었습니다.');
```

실습> 아이디와 암호를 입력받아,  
 -- 아이디와 암호는 글자갯수가 10 글자이상 15 글자 미만일 때만 정상 출력하고  
 -- 글자 갯수가 10 글자 미만이면, TOOSHORT 예외 발생시키고  
 -- 글자 갯수가 15 글자 이상이면, TOOLONG 예외 발생시키도록 PL/SQL 구문을 작성하시오.  
 -- TOOLONG 은 제공되는 오류코드 -12899 에 대해 매핑하고,  
 -- TOOSHORT 는 오류코드를 -20001 로 새로 정하고, 메세지는 '글자갯수 부족'으로 처리함  
 -- 아이디는 같은 값 두번 입력못하게 중복 예외 적용함.  
 -- 아이디 중복 예외 테스트용 초기값 대입한 아이디 변수 따로 준비해 놓음  
 -- 입력된 아이디와 준비된 변수값이 같으면 중복 예외 발생시킴

```

DECLARE
    VID VARCHAR2(14) := 'STUDENT0123';
    V_ID VARCHAR2(14);
    V_PWD VARCHAR2(14);
```

```
TOOLONG EXCEPTION;
TOOSHORT EXCEPTION;
PRAGMA EXCEPTION_INIT(TOOLONG, -12899);
PRAGMA EXCEPTION_INIT(TOOSHORT, -20001);
BEGIN
    V_ID := '&ID';
    V_PWD := '&PWD';

    IF (LENGTH(V_ID) < 10 OR LENGTH(V_PWD) < 10) THEN
        RAISE_APPLICATION_ERROR(-20001, '글자 갯수 부족');
    END IF;

    IF VID = V_ID THEN
        RAISE DUP_VAL_ON_INDEX;
    END IF;

EXCEPTION
    WHEN DUP_VAL_ON_INDEX THEN
        DBMS_OUTPUT.PUT_LINE('아이디가 중복되었습니다.');
```

WHEN TOOLONG THEN

```
        DBMS_OUTPUT.PUT_LINE('글자 갯수 범위 초과');
```

WHEN TOOSHORT THEN

```
        DBMS_OUTPUT.PUT_LINE('글자 갯수 부족.');
```

WHEN OTHERS THEN

```
        DBMS_OUTPUT.PUT_LINE('알수 없는 오류가 발생하였습니다.');
```

END;

/