오라클 PL/SQL 강좌

PL/SQL 이란 ?

- PL/SQL 은 Oracle's Procedural Language extension to SQL. 의 약자 입니다.
- SQL 문장에서 변수정의, 조건처리(IF), 반복처리(LOOP, WHILE, FOR)등을 지원하며, 오라클 자체에 내장되어 있는 Procedure Language 입니다
- DECLARE 문을 이용하여 정의되며, 선언문의 사용은 선택 사항입니다.
- PL/SQL 문은 블록 구조로 되어 있고 PL/SQL 자신이 컴파일 엔진을 가지고 있습니다.

PL/SQL 의 장점

- PL/SQL 문은 BLOCK 구조로 다수의 SQL 문을 한번에 ORACLE DB 로 보내서 처리하므로 수행속도를 향상 시킬수 있습니다.
- PL/SQL 의 모든 요소는 하나 또는 두개이상의 블록으로 구성하여 모듈화가 가능하다.
- 보다 강력한 프로그램을 작성하기 위해서 큰 블록안에 소블럭을 위치시킬 수 있습니다.
- Variable, Constant, Cursor, Exception 을 정의하고, SQL 문장과 Procedural 문장에서 사용합니다.
- 단순, 복잡한 데이터형태의 변수를 선언합니다.
- 테이블의 데이터 구조와 DataBase 의 컬럼럼에 준하여 동적으로 변수를 선언 할 수 있습니다.
- Exception 처리 루틴을 이용하여 Oracle Server Error 를 처리합니다.
- 사용자 정의 에러를 선언하고 Exception 처리 루틴으로 처리 가능 합니다.

PL/SQL Block Structure

- PL/SQL 은 프로그램을 논리적인 블록으로 나누는 구조화된 블록 언어 입니다.
- PL/SQL 블록은 선언부(선택적), 실행부(필수적),예외 처리부(선택적)로 구성되어 있고, BEGIN 과 END 키워드는 반드시 기술해 주어야 합니다.
- PL/SQL 블록에서 사용하는 변수는 블록에 대해 논리적으로 선언할 수 있고 사용할 수 있습니다.

◈ Declarative Section(선언부)

- 변수, 상수, CURSOR, USER_DEFINE Exception 선언

◈ Executable Section(실행부)

- SQL, 반복분, 조건문실행
- 실행부는 BEGIN 으로 시작하고 END 로 끝납니다.
- 실행문은 프로그램 내용이 들어가는 부분으로서 필수적으로 사용되어야 합니다.

◈ Exception Handling Section(예외처리)

- 예외에 대한 처리.
- 일반적으로 오류를 정의하고 처리하는 부분으로 선택 사항입니다.

- DECLARE
  - Optional
  - Variables, cursors, user-defined exceptions
- BEGIN
  - Mandatory
  - SQL Statements
  - PL/SQL Statements
- EXCEPTION
  - Actions to perform when errors occur
- END;
  - Mandatory

PL/SQL 프로그램의 작성 요령

- PL/SQL 블록내에서는 한 문장이 종료할 때마다 세미콜론(;)을 사용합니다. .
- END 뒤에 ;을 사용하여 하나의 블록이 끝났다는 것을 명시 합니다.
- PL/SQL 블록의 작성은 편집기를 통해 파일로 작성할 수도 있고, SQL 프롬프트에서 바로 작성할 수도 있습니다.
- SLQ*PLUS 환경에서는 DELCLARE 나 BEGIN 이라는 키워드로 PL/SQL 블럭이 시작하는 것을 알 수 있습니다.
- 단일행 주석 : --
- 여러행 주석 : /* */
- PL/SQL 블록은 행에 / 가 있으면 종결됩니다.

프로시져란..

- 특정 작업을 수행할수 있고, 이름이 있는 PL/SQL 블록으로서. 매개 변수를 받을 수 있고..
  반복적으로 사용할 수 있는거죠.. 보통 연속 실행 또는 구현이 복잡한 트랜잭션을 수행하는
  PL/SQL 블록을 데이터 베이스에 저장하기 위해 생성합니다.

⊙ CREATE OR REPLACE 구문을 사용하여 생성합니다.

⊙ IS 로 PL/SQL 의 블록을 시작합니다.

⊙ LOCAL 변수는 IS 와 BEGIN 사이에 선언합니다.

```
[Syntax]
CREATE OR REPLACE procedure name
  IN argument
  OUT argument
  IN OUT argument
IS
  [변수의 선언]
BEGIN  --> 필수
  [PL/SQL Block]
  -- SQL 문장, PL/SQL 제어 문장
  [EXCEPTION]  --> 선택
  -- error 가 발생할 때 수행하는 문장
END;  --> 필수
```

프로시저 작성 예제

```
SQL>CREATE OR REPLACE PROCEDURE update_sal
    /* IN  Parameter */
    (v_empno        IN    NUMBER)
   IS
   BEGIN
    UPDATE emp
    SET sal = sal  * 1.1
    WHERE empno = v_empno;
    COMMIT;
   END update_sal;
   /
```

프로시져가 생성되었습니다.

설명..

프로시저의 이름은 update_sal 이고..

프로시저 update_sal 은 사번(v_empno)를 입력받아서 급여를 update 시켜주는 sql 문입니다.

프로시저를 끝마칠때에는 항상 "/"를 지정 합니다.

프로시저의 실행

EXECUTE 문을 이용해 프로시저를 실행합니다.

SQL> execute update_sal(7369);

PL/SQL 처리가 정상적으로 완료되었습니다.

7369 번 사원의 급여가 10% 인상됐습니다.

SELECT 문을 실행시켜보면 데이터가 수정된 것을 확인할 수 있습니다.

Parameter 란

⊙ 실행 환경과 program 사이에 값을 주고 받는 역할을 합니다.

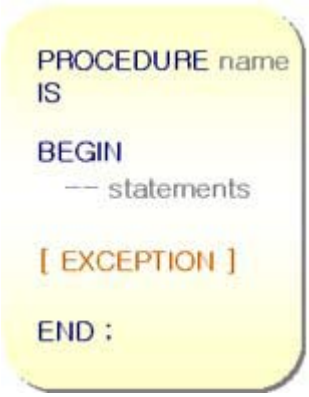
⊙ 블록 안에서의 변수와 똑같이 일시적으로 값을 저장하는 역할을 합니다.

⊙ Parameter 의 타입

- IN : 실행환경에서 program 으로 값을 전달
- OUT : program 에서 실행환경으로 값을 전달
- INOUT : 실행환경에서 program 으로 값을 전달하고,
  다시 program 에서 실행환경으로 변경된 값을 전달

Block Type(PL/SQL 블럭의 유형)

[Anonymous]



[Procedure]

[Function]

◈ Anonymous Block(익명 블록)

이름이 없는 블록을 의미합니다.

실행하기 위해 프로그램 안에서 선언되고 실행 시에 실행을 위해 PL/SQL 엔진으로 전달됩니다.

선행 컴파일러 프로그램과 SQL*Plus 또는 서버 관리자에서 익명의 블록을 내장할 수 있습니다.

◈ Procedure(프로시저)

특정 작업을 수행할 수 있는 이름이 있는 PL/SQL 블록으로서.

매개 변수를 받을 수 있고.. 반복적으로 사용할 수 있는 거죠..

보통 연속 실행 또는 구현이 복잡한 트랜잭션을 수행하는 PL/SQL블록을

데이터 베이스에 저장하기 위해 생성합니다.

◈ Function(함수)

보통 값을 계산하고 결과값을 반환하기 위해서 함수를 많이 사용합니다.

대부분 구성이 프로시저와 유사하지만 IN 파라미터만 사용 할 수 있고,

반드시 반환될 값의 데이터 타입을 RETURN문에 선언해야 합니다.

또한 PL/SQL블록 내에서 RETURN문을 통해서 반드시 값을 반환해야 합니다.

◈ 함수(Function)

- 보통 값을 계산하고 결과값을 반환하기 위해서 함수를 많이 사용 합니다.
- 대부분 구성이 프로시저와 유사 하지만 IN 파라미터만 사용 할 수 있습니다.
- 반드시 반환될 값의 데이터 타입을 RETURN 문에 선언해야 합니다.
- 또한 PL/SQL 블록 내에서 RETURN 문을 통해서 반드시 값을 반환해야 합니다.

[Syntax]

```
CREATE OR REPLACE FUNCTION function name
 [(argument...)]
RETURN  datatype
   -- Datatype 은 반환되는 값의 datatype 입니다.
IS
  [변수 선언 부분]
BEGIN
 [PL/SQL Block]
  -- PL/SQL 블록에는 적어도 한 개의 RETURN 문이 있어야 합니다.
  -- PL/SQL Block 은 함수가 수행할 내용을 정의한 몸체부분입니다.
END;
```

```
SQL> CREATE OR REPLACE FUNCTION FC_update_sal  (v_empno       IN    NUMBER)
      -- 리턴되는 변수의 데이터타입을 꼭 정의해야 합니다
      RETURN NUMBER  .
     IS
       v_sal  emp.sal%type;
     BEGIN
       UPDATE emp       SET sal  = sal  * 1.1     WHERE empno  = v_empno;
       COMMIT;
       SELECT sal  INTO v_sal    FROM emp     WHERE empno = v_empno;
     -- 리턴문이 꼭 존재해야 합니다
      RETURN v_sal;
    END;
함수가 생성되었습니다.
```

설명..

이 함수에는 v_sal 이라는 %type 변수가 사용되고 있습니다.

스칼라 데이터 타입을 참고하세요.

프로지저와 마찬가지로 세미콜론(;)으로 블록을 종료한 뒤 "/"를 붙여 코드를 끝마칩니다.

함수의 실행

먼저 함수의 반환값을 저장할 변수를 선언합니다.

```
SQL> VAR salary NUMBER;
```

EXECUTE 문을 이용해 함수를 실행합니다.

```
SQL>EXECUTE :salary := FC_update_sal(7900);
PL/SQL 처리가 정상적으로 완료되었습니다.
```

오라클 SQL 에서 선언된 변수의 출력은 PRINT 문을 사용합니다.

PRINT 문으로 함수의 반환값을 저장한 salary 의 값을 확인하면 됩니다.

```
SQL>PRINT salary;
   SALARY
----------
     1045
```

결과가 이렇게 나옵니다.

변수 선언 방법

[Syntax]

```
identifier [CONSTANT] 데이터타입 [NOT NULL] [:= 상수값이나 표현식] ;
```

⊙ Identifier 의 이름은 sql 의 object 명과 동일한 규칙을 따릅니다.

⊙ Identifier 를 상수로 지정하고 싶은 경우는 CONSTANT 라는 KEYWORD 를 명시하고 반드시 초기화를 할당합니다.

⊙ NOT NULL 이 정의되어 있으면 초기값을 반드시 지정하고, 정의되어 있지 않을 때는 생략 가능합니다.

⊙ 초기값은 할당 연산자(:=)를 사용하여 정의 합니다.

⊙ 초기값을 정의하지 않으면 Identifier 는 NULL 값을 가지게 됩니다.

⊙ 일반적으로 한줄에 한 개의 Identifier 를 정의 합니다.

※ 스칼라 데이터 타입은 단수 데이터형으로 한가지의 데이터 값만 가집니다.

BINARY_INTEGER	-2147483647 에서 2147483647 사이의 정수
NUMBER[(P, S)]	고정 및 부동 소숫점 수에 대한 기본 유형
CHAR[(최대길이)]	고정 길이 문자에 대한 기본형은 32767 바이트까지 입니다. 지정하지 않는다면 디폴트 길이는 1 로 설정됩니다.
LONG	고정 길이 문자에 대한 기본형은 32760 바이트까지 입니다. LONG 데이터베이스 열의 최대 폭은 2147483647 바이트입니다.
LONG RAW	이진 데이터와 바이트 문자열에 대한 기본형은 32760Byte 까지 입니다. LONG RAW 데이터는 PL/SQL 에 의해 해석되지 않습니다.
VARCHAR2(최대길이)	3 변수 길이 문자 데이터에 대한 기본형은 32767Byte 까지 입니다. VARCHAR2 변수와 상수에 대한 디폴트 크기는 없습니다.
DATE	날짜와 시간에 대한 기본형. DATE 값은 지정 이후의 초 단위로 날에 대한 시간을 포함합니다. 날짜의 범위는 BC 4712 년 1 월 1 일부터 AD 9999 년 12 월 31 일사이 입니다.
BOOLEAN	논리연산에 사용되는 세 가지 값(TRUE, FALSE, NULL) 중 하나를 저장 하는 데이터 유형

선언 예제

```
v_price CONTANT NUMBER(4,2) := 12.34 ;     -- 상수 숫자 선언(변할 수 없다)
v_name VARCHAR22(20) ;
v_Bir_Type   CHAR(1) ;
v_flag    BOOLEAN   NOT NULL := TRUE ;      -- NOT NULL 값 TRUE 로 초기화
v_birthday DATE;
```

%TYPE 데이터형

⊙ %TYPE 데이터형은 기술한 데이터베이스 테이블의 컬럼 데이터 타입을 모를 경우 사용할 수 있고,

⊙ 또. 코딩이후 데이터베이스 컬럼의 데이터 타입이 변경될 경우 다시 수정할 필요가 없습니다.

⊙ 이미 선언된 다른 변수나 데이터베이스 컬럼의 데이터 타입을 이용하여 선언합니다.

⊙ 데이터 베이스 테이블과 컬럼 그리고 이미 선언한 변수명이 %TYPE 앞에 올 수 있습니다.

%TYPE 속성을 이용하여 얻을 수 있는 장점

- 기술한 DB column definition 을 정확히 알지 못하는 경우에 사용할 수 있습니다.
- 기술한 DB column definition 이 변경 되어도 다시 PL/SQL 을 고칠 필요가 없습니다.

예제

```
v_empno   emp.empno%TYPE := 7900 ;
v_ename emp.ename%TYPE;
```

예제 프로시져..

```
SQL>CREATE OR REPLACE PROCEDURE Emp_Info    ( p_empno IN emp.empno%TYPE )
     IS
     -- %TYPE 데이터형 변수 선언
     v_empno emp.empno%TYPE;
     v_ename emp.ename%TYPE;
     v_sal   emp.sal%TYPE;
     BEGIN
     DBMS_OUTPUT.ENABLE;
     -- %TYPE 데이터형 변수 사용
     SELECT empno, ename, sal
     INTO v_empno, v_ename, v_sal
     FROM emp
     WHERE empno = p_empno ;
     -- 결과값 출력
     DBMS_OUTPUT.PUT_LINE( '사원번호 : ' || v_empno );
     DBMS_OUTPUT.PUT_LINE( '사원이름 : ' || v_ename );
     DBMS_OUTPUT.PUT_LINE( '사원급여 : ' || v_sal );

     END;
    /
프로시져가 생성되었습니다.
SQL>SET SERVEROUTPUT ON;    -- DBMS_OUTPUT 결과값을 화면에 출력 하기위해
```

실행 결과

```
SQL> EXECUTE Emp_Info(7369);
사원번호 : 7369
사원이름 : SMITH
사원급여 : 880

PL/SQL 처리가 정상적으로 완료되었습니다.
```

복합데이터타입

하나 이상의 데이터값을 갖는 데이터 타입으로 배열과 비슷한 역할을 하고 재사용이 가능합니다.
%ROWTYPE 데이터 형과, PL/SQL 테이블과 레코드가 복합 데이터 타입에 속합니다.

%ROWTYPE

- ⊙ 테이블이나 뷰 내부의 컬럼 데이터형, 크기, 속석등을 그대로 사용할 수 있습니다.
- ⊙ %ROWTYPE 앞에 오는 것은 데이터 베이스 테이블 이름입니다.
- ⊙ 지정된 테이블의 구조와 동일한 구조를 갖는 변수를 선언할 수 있습니다.
- ⊙ 데이터베이스 컬럼들의 수나 DATATYPE 을 알지 못할 때 편리 합니다.
- ⊙ 테이블의 데이터 컬럼의 DATATYPE 이 변경될 경우 프로그램을 재 수정할 필요가 없습니다.

%ROWTYPE 예제 프로시져..

```
SQL>CREATE OR REPLACE PROCEDURE RowType_Test      ( p_empno IN emp.empno%TYPE )
  IS
      v_emp   emp%ROWTYPE ;
 BEGIN
    DBMS_OUTPUT.ENABLE;
    -- %ROWTYPE 변수 사용
    SELECT empno, ename, hiredate
    INTO v_emp.empno, v_emp.ename, v_emp.hiredate
    FROM emp    WHERE empno = p_empno;
   DBMS_OUTPUT.PUT_LINE( '사원번호 : ' || v_emp.empno );
   DBMS_OUTPUT.PUT_LINE( '사원이름 : ' || v_emp.ename );
   DBMS_OUTPUT.PUT_LINE( '입 사 일 : ' || v_emp.hiredate );
 END;
    /
 프로시져가 생성되었습니다.
```

실행 결과

```
 SQL> SET SERVEROUTPUT ON ;  -- DBMS_OUTPUT.PUT_LINE 을 출력하기 위해 사용
 SQL> EXECUTE RowType_Test(7900);
  사원번호 : 7900
  사원이름 : JAMES
  입 사 일 : 81/12/03
PL/SQL 처리가 정상적으로 완료되었습니다.
```

PL/SQL 테이블

PL/SQL 에서의 테이블은 오라클 SQL 에서의 테이블과는 다릅니다. PL/SQL 에서의 테이블은 일종의 일차원 배열이라고 생각하시면 이해하기 쉬울겁니다.

⊙ 테이블은 크기에 제한이 없으면 그 ROW 의 수는 데이터가 들어옴에 따라 자동 증가 합니다.

⊙ BINARY_INTEGER 타입의인덱스 번호로 순서가 정해집니다.

⊙ 하나의 테이블에 한 개의 컬럼 데이터를 저장 합니다.

[Syntax]

예제

```
TYPE prdname_table IS TABLE OF VARCHAR2(30)
INDEX BY BINARY_INTEGER;

-- prdname_table 테이블타입으로 prdname_tab 변수를 선언해서 사용
prdname_tab   prdname_table ;
```

PL/SQL 테이블 예제 프로시져..

```
SQL>CREATE OR REPLACE PROCEDURE Table_Test
  (v_deptno IN emp.deptno%TYPE)
  IS
     TYPE empno_table IS TABLE OF emp.empno%TYPE
     INDEX BY BINARY_INTEGER;
     TYPE ename_table IS TABLE OF emp.ename%TYPE
     INDEX BY BINARY_INTEGER;
     TYPE sal_table IS TABLE OF emp.sal%TYPE
     INDEX BY BINARY_INTEGER;
     -- 테이블타입으로 변수를 선언해서 사용
     empno_tab  empno_table ;
     ename_tab  ename_table ;
     sal_tab    sal_table;
     i BINARY_INTEGER := 0;
  BEGIN
     DBMS_OUTPUT.ENABLE;
     FOR emp_list IN(SELECT empno, ename, sal FROM emp WHERE deptno = v_deptno) LOOP
      /* emp_list 는 자동선언되는 BINARY_INTEGER 형 변수로 1 씩 증가합니다.
         emp_list 대신 다른 문자열 사용가능 */
          i := i + 1;
         -- 테이블 변수에 검색된 결과를 넣습니다
          empno_tab(i) := emp_list.empno ;
          ename_tab(i) := emp_list.ename ;
          sal_tab(i)   := emp_list.sal ;
     END LOOP;
     -- 1 부터 i 까지 FOR 문을 실행
     FOR cnt IN 1..i LOOP
       -- TABLE 변수에 넣은 값을 뿌려줌
       DBMS_OUTPUT.PUT_LINE( '사원번호 : ' || empno_tab(cnt) );
       DBMS_OUTPUT.PUT_LINE( '사원이름 : ' || ename_tab(cnt) );
       DBMS_OUTPUT.PUT_LINE( '사원급여 : ' || sal_tab(cnt) );
     END LOOP;
  END;
  /
```

프로시져가 생성되었습니다.

실행 결과

```
SQL> SET SERVEROUTPUT ON ; -- (DBMS_OUTPUT.PUT_LINE 을 출력하기 위해 사용)
SQL> EXECUTE Table_Test(10);
사원번호 : 7782
사원이름 : CLARK
사원급여 : 2450
사원번호 : 7839
사원이름 : KING
사원급여 : 5000
사원번호 : 7934
사원이름 : MILLER
사원급여 : 1300
PL/SQL 처리가 정상적으로 완료되었습니다.
```

emp 테이블에 있는 데이터의 입력한 부서에 해당하는 사원번호, 사원이름, 사원급여를 뿌려주는 프로시져 입니다

PL/SQL 레코드

여러 개의 데이터 타입을 갖는 변수들의 집합입니다.

⊙ 스칼라, RECORD, 또는 PL/SQL TABLE datatype 중 하나 이상의 요소로 구성됩니다.

⊙ 논리적 단위로서 필드 집합을 처리할 수 있도록 해 줍니다.

⊙ PL/SQL 테이블과 다르게 개별 필드의 이름을 부여할 수 있고, 선언시 초기화가 가능합니다.


```
예제   TYPE record_test IS RECORD
   ( record_empno    NUMBER,
     record_ename    VARCHAR2(30),
     record_sal      NUMBER);
     prd_record      record_test;
```

PL/SQL RECORD 예제 프로시져..

```
SQL> CREATE OR REPLACE PROCEDURE Record_Test   ( p_empno IN emp.empno%TYPE )
  IS
     TYPE emp_record IS RECORD
     (v_empno    NUMBER,    v_ename    VARCHAR2(30),    v_hiredate   DATE );
     emp_rec   emp_record ;
 BEGIN
     DBMS_OUTPUT.ENABLE;
     SELECT empno, ename, hiredate
     INTO emp_rec.v_empno, emp_rec.v_ename, emp_rec.v_hiredate
     FROM emp    WHERE empno = p_empno;
    DBMS_OUTPUT.PUT_LINE( '사원번호 : ' || emp_rec.v_empno );
    DBMS_OUTPUT.PUT_LINE( '사원이름 : ' || emp_rec.v_ename );
    DBMS_OUTPUT.PUT_LINE( '입 사 일 : ' || emp_rec.v_hiredate );
 END;
/
```

프로시져가 생성되었습니다.

실행 결과

```
SQL> SET SERVEROUTPUT ON ;  -- (DBMS_OUTPUT.PUT_LINE 을 출력하기 위해 사용)
SQL> EXECUTE Record_Test(7369);
사원번호 : 7369
사원이름 : SMITH
입 사 일 : 80/12/17
PL/SQL 처리가 정상적으로 완료되었습니다.     %ROWTYPE 예제와 비교해 보세요
```

PL/SQL Table of Record

⊙ PL/SQL TABLE 변수 선언과 비슷하며 데이터타입을 %ROWTYPE 으로 선언하면 됩니다.

⊙ PL/SQL TABLE 과 RECORD 의 복합 기능을 합니다.

[xojnyℓ]

```
DECLARE
TYPE dept_table_type IS TABLE OF dept%ROWTYPE
INDEX BY BINARY_INTEGER;
-- Each element of dept_table  is a record
dept_table    dept_table_type ;
```

PL/SQL TABLE OF RECORD 예제 프로시져..

```
SQL> CREATE OR REPLACE PROCEDURE Table_Test
  IS
       i BINARY_INTEGER := 0;
    -- PL/SQL Table of Record 의 선언
   TYPE dept_table_type IS TABLE OF dept%ROWTYPE
    INDEX BY BINARY_INTEGER;
    dept_table dept_table_type;
BEGIN
   FOR dept_list IN (SELECT * FROM dept) LOOP
       i:= i+1;
      dept_table(i).deptno := dept_list.deptno ;
      dept_table(i).dname := dept_list.dname ;
      dept_table(i).loc    := dept_list.loc ;
   END LOOP;
   FOR cnt IN 1..i LOOP
      DBMS_OUTPUT.PUT_LINE( '부서번호 : ' || dept_table(cnt).deptno ||
                            '부서명 : ' ||  dept_table(cnt).dname ||
                            '위치 : ' || dept_table(cnt).loc );
   END LOOP;
 END;
/
```

실행결과

```
SQL>set serveroutput on;
SQL>exec Table_test;
부서번호 : 10 부서명 : ACCOUNTING 위치 : NEW_YORK
부서번호 : 20 부서명 : RESEARCH 위치 : DALLAS
부서번호 : 30 부서명 : 인사과위치 : CHICAGO
부서번호 : 40 부서명 : OPERATIONS 위치 : BOS%TON
PL/SQL 처리가 정상적으로 완료되었습니다.
```

Insert 문

PL/SQL 에서의 INSERT 문은 SQL 과 비슷합니다.

사원 등록 예제 프로시져..

```
SQL> CREATE OR REPLACE PROCEDURE Insert_Test
     ( v_empno  IN emp.empno%TYPE,
       v_ename  IN emp.ename%TYPE,
       v_deptno IN emp.deptno%TYPE )
     IS
    BEGIN
       DBMS_OUTPUT.ENABLE;
       INSERT INTO emp(empno, ename, hiredate, deptno)
       VALUES(v_empno, v_ename, sysdate, v_deptno);
       DBMS_OUTPUT.PUT_LINE( '사원번호 : ' || v_empno );
       DBMS_OUTPUT.PUT_LINE( '사원이름 : ' || v_ename );
       DBMS_OUTPUT.PUT_LINE( '사원부서 : ' || v_deptno );
       DBMS_OUTPUT.PUT_LINE( '데이터 입력 성공 ' );
     END ;
    /
프로시져가 생성되었습니다.
```

실행 결과

```
SQL> SET SERVEROUTPUT ON ;  -- (DBMS_OUTPUT.PUT_LINE 을 출력하기 위해 사용)
SQL>  EXECUTE Insert_Test(1000, 'brave', 20);
사원번호 : 1000
사원이름 : brave
사원부서 : 20
데이터 입력 성공

PL/SQL 처리가 정상적으로 완료되었습니다.
```

UPDATE

상품 수정 예제 프로시저..

※ 특정 사원의 급여를 일정%센트 인상/인하하는 프로시져

```
SQL>CREATE OR REPLACE PROCEDURE Update_Test
    ( v_empno IN    emp.empno%TYPE,      -- 급여를 수정한 사원의 사번
      v_rate    IN    NUMBER )             -- 급여의 인상/인하율
      IS
      v_emp  emp%ROWTYPE ;
      BEGIN
      DBMS_OUTPUT.ENABLE;
      UPDATE emp   SET sal = sal+(sal * (v_rate/100))   -- 급여를 계산
      WHERE empno = v_empno ;
      DBMS_OUTPUT.PUT_LINE( '데이터 수정 성공 ' );
      -- 수정된 데이터 확인하기 위해 검색
      SELECT empno, ename, sal    INTO v_emp.empno, v_emp.ename, v_emp.sal
      FROM emp    WHERE empno = v_empno ;
      DBMS_OUTPUT.PUT_LINE( ' **** 수 정 확 인 **** ');
      DBMS_OUTPUT.PUT_LINE( '사원번호 : ' || v_emp.empno );
      DBMS_OUTPUT.PUT_LINE( '사원이름 : ' || v_emp.ename );
      DBMS_OUTPUT.PUT_LINE( '사원급여 : ' || v_emp.sal );
      END ;
      /
프로시저가 생성되었습니다.
```

프로시저 실행

```
SQL> SET SERVEROUTPUT ON ;  -- (DBMS_OUTPUT.PUT_LINE 을 출력하기 위해 사용)
SQL> EXECUTE Update_Test(7900, -10);
데이터 수정 성공
**** 수 정 확 인 ****
사원번호 : 7900
사원이름 : JAMES
사원급여 : 855
PL/SQL 처리가 정상적으로 완료되었습니다.
7900 번 사원의 급여를 10% 인하했습니다.
```

DELETE

사원 삭제 예제 프로시저..

```
SQL> CREATE OR REPLACE PROCEDURE Delete_Test  ( p_empno IN  emp.empno%TYPE )
     IS
     -- 삭제 데이터를 확인하기 레코드 선언
     TYPE del_record IS  RECORD
     ( v_empno      emp.empno%TYPE,
       v_ename      emp.ename%TYPE,
       v_hiredate    emp.hiredate%TYPE) ;
       v_emp  del_record ;
     BEGIN
     DBMS_OUTPUT.ENABLE;
      -- 삭제된 데이터 확인용 쿼리
      SELECT empno, ename, hiredate
      INTO v_emp.v_empno, v_emp.v_ename, v_emp.v_hiredate
      FROM emp    WHERE empno = p_empno ;
     DBMS_OUTPUT.PUT_LINE( '사원번호 : ' || v_emp.v_empno );
     DBMS_OUTPUT.PUT_LINE( '사원이름 : ' || v_emp.v_ename );
     DBMS_OUTPUT.PUT_LINE( '입 사 일 : ' || v_emp.v_hiredate );
     -- 삭제 쿼리
     DELETE    FROM emp   WHERE empno = p_empno ;
     DBMS_OUTPUT.PUT_LINE( '데이터 삭제 성공 ' );
    END;
/
프로시저가 생성되었습니다.
```

프로시저 실행 (결과화면)

```
SQL> SET SERVEROUTPUT ON ;  -- (DBMS_OUTPUT.PUT_LINE 을 출력하기 위해 사용)
SQL> EXECUTE Delete_Test(7900);
사원번호 : 7900
사원이름 : JAMES
입 사 일 : 81/12/03
데이터 삭제 성공
PL/SQL 처리가 정상적으로 완료되었습니다.
```

※ 7900 사원을 삭제했습니다.

FOR LOOP 문

```
FOR index IN [REVERSE] 시작값 .. END값 LOOP
  statement 1
  statement 2
  ....
END LOOP;
```

- index 는 자동 선언되는 binary_integer 형 변수이고, 1 씩 증가합니다.
- reverse 옵션이 사용될 경우 index 는 upper_bound 에서 lower_bound 로 1 씩 감소합니다.
- IN 다음에는 coursor 나 select 문이 올 수 있습니다.

FOR 문 예제

```
DECLARE
 -- 사원 이름을 출력하기 위한 PL/SQL 테이블 선언
  TYPE ename_table IS TABLE OF emp.ename%TYPE
  INDEX BY BINARY_INTEGER;
 -- 사원 급여를 출력하기 위한 PL/SQL 테이블 선언
  TYPE sal_table IS TABLE OF emp.sal%TYPE
  INDEX BY BINARY_INTEGER;
  ename_tab    ename_table;
  sal_tab      sal_table;
  i BINARY_INTEGER := 0;
BEGIN
 DBMS_OUTPUT.ENABLE;
 FOR emp_list IN  (SELECT ename, sal FROM emp WHERE deptno = 10) LOOP
    i := i +1 ;
    ename_tab(i) := emp_list.ename;     -- 테이블에 사원 이름을 저장
    sal_tab(i)     := emp_list.sal;       -- 테이블에 사원 급여를 저장
 END LOOP;
 FOR cnt IN  1..i  LOOP       -- 화면에 출력
   DBMS_OUTPUT.PUT_LINE('사원이름 : ' || ename_tab(cnt));
   DBMS_OUTPUT.PUT_LINE('사원급여 : ' || sal_tab(cnt));
 END LOOP;
END;
/
사원이름 : CLARK
사원급여 : 2450
```

```
사원이름 : KING
사원급여 : 5000
사원이름 : MILLER
사원급여 : 1300
PL/SQL 처리가 정상적으로 완료되었습니다.
```

Loop 문 while 문

EXIT 문이 사용되었을 경우, 무조건 LOOP 문을 빠져나갑니다,

EXITH WHEN 이 사용될 경우 WHEN 절에 LOOP 를 빠져 나가는 조건을 제어할 수 있습니다.

LOOP 문 예제

```
SQL> SET SERVEROUTPUT ON ;  -- (DBMS_OUTPUT.PUT_LINE 을 출력하기 위해 사용)
SQL>  DECLARE
        v_cnt number(3) := 100;
      BEGIN
        DBMS_OUTPUT.ENABLE ;
       LOOP
        INSERT INTO emp(empno, ename , hiredate)
        VALUES(v_cnt, 'test'||to_char(v_cnt),  sysdate);
        v_cnt := v_cnt+1;
        EXIT WHEN v_cnt > 110;
       END LOOP;
       DBMS_OUTPUT.PUT_LINE(v_cnt-100 || '개의 데이터가 입력되었습니다');
      END;
     /
 11 개의 데이터가 입력되었습니다
PL/SQL 처리가 정상적으로 완료되었습니다.
```

WHILE LOOP 문

WHILE LOOP문은 FOR 문과 비슷하며 조건이 TRUE일 경우만 반복되는 LOOP문 입니다.

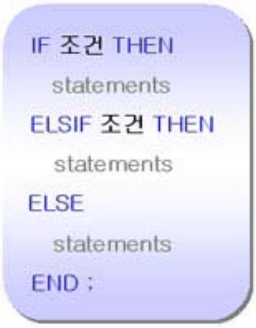
예제

```
WHILE cnt < 10 LOOP
  INSERT INTO emp(empno, ename , hiredate)
  VALUES(emp_seq.nextval, 'test',  sysdate);
  cnt := cnt + 1 ;
END LOOP ;
```

cnt가 10이면 반복 While Loop를 탈출

EXIT WHEN 조건 => 조건이 만족할 때 반복 loop를 탈출합니다. .

조건제어



IF 문 예제 프로시저..

```
SQL>CREATE OR REPLACE PROCEDURE Dept_Search  (p_empno IN emp.empno%TYPE )
IS
  v_deptno emp.deptno%type ;
BEGIN
  DBMS_OUTPUT.ENABLE;
  SELECT deptno INTO v_deptno FROM emp WHERE empno = p_empno ;
  IF v_deptno <=  7000 THEN
     DBMS_OUTPUT.PUT_LINE( ' ACCOUNTING 부서 사원입니다. ' );
  ELSIF v_deptno < 7900 THEN
     DBMS_OUTPUT.PUT_LINE( ' RESEARCH 부서 사원입니다. ' );
  ELSE
     DBMS_OUTPUT.PUT_LINE( ' 부서가 없네요... ' );
  END IF ;
END ;
/
프로시저가 생성되었습니다.
```

프로시저 실행

```
SQL> SET SERVEROUTPUT ON ; -- (DBMS_OUTPUT.PUT_LINE 을 출력하기 위해 사용)
SQL> EXECUTE Dept_Search(7900);
부서가 없네요...
PL/SQL 처리가 정상적으로 완료되었습니다.
SQL> EXECUTE Dept_Search(7369);
RESEARCH 부서 사원입니다.
PL/SQL 처리가 정상적으로 완료되었습니다.
```

암시적인 커서는 오라클이나 PL/SQL 실행 메커니즘에 의해 처리되는 SQL 문장이 처리되는 곳에 대한 익명의 에드레스입니다. 오라클 데이터 베이스에서 실행되는 모든 SQL 문장은 암시적인 커서이며 그것들과 함께 모든 암시적인 커서 속성이 사용될 수 있습니다.

-암시적 커서의 속성

◈ SQL%ROWCOUNT : 해당 SQL 문에 영향을 받는 행의 수

◈ SQL%FOUND : 해당 SQL 영향을 받는 행의 수가 1 개 이상일 경우 TRUE

◈ SQL%NOTFOUND : 해당 SQL 문에 영향을 받는 행의 수가 없을 경우 TRUE

◈ SQL%ISOPEN : 항상 FALSE, 암시적 커서가 열려 있는지의 여부 검색

(암시적 커서는 SQL 문이 실행되는 순간 자동으로 열림과 닫힘 실행)

암시적 커서 예제

```
CREATE OR REPLACE PROCEDURE Implicit_Cursor (p_empno emp.empno%TYPE)
is
   v_sal   emp.sal%TYPE;
   v_update_row NUMBER;
BEGIN
   SELECT sal INTO v_sal FROM emp   WHERE empno = p_empno ;
 -- 검색된 데이터가 있을경우
   IF  SQL%FOUND THEN
      DBMS_OUTPUT.PUT_LINE('검색한 데이터가 존재합니다 : '||v_sal);
   END IF;
   UPDATE emp SET sal = sal*1.1   WHERE empno = p_empno;
 -- 수정한 데이터의 카운트를 변수에 저장
   v_update_row := SQL%ROWCOUNT;
   DBMS_OUTPUT.PUT_LINE('급여가 인상된 사원 수 : '|| v_update_row);
END;
프로시저가 생성되었습니다.
```

실행결과

```
SQL> SET SERVEROUTPUT ON ;  -- (DBMS_OUTPUT.PUT_LINE 을 출력하기 위해 사용)
SQL> EXECUTE Implicit_Cursor(7369);
검색한 데이터가 존재합니다 : 880
급여가 인상된 사원 수 : 1
PL/SQL 처리가 정상적으로 완료되었습니다
```

▣ 커서란 무엇인가? 명시적 커서(Explicit Cursor)

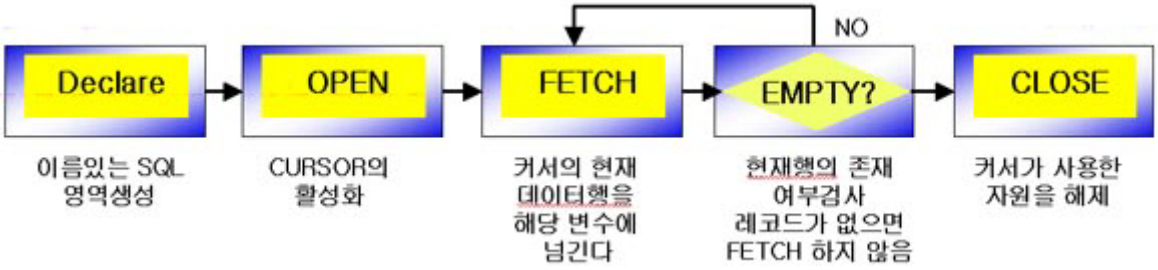
◈커서는 Private SQL 의 작업영역 입니다.

◈오라클 서버에 의해 실행되는 모든 SQL 문은 연관된 각각의 커서를 소유하고 있습니다.

◈ 커서의 종류

- 암시적 커서 : 모든 DML 과 PL/SQL SELECT 문에 대해 선언됩니다.
- 명시적 커서 : 프로그래머에 의해 선언되며 이름이 있는 커서입니다.

▣ Explicit Cursor 의 흐름도?



▣ 문법(Syntax)

```
CURSOR cursor_name IS
SELECT statement
```

▣ 커서 열기(OPEN)

◈커서의 열기는 OPEN 문을 사용합니다.

◈커서안의 검색이 실행되며 아무런 데이터행을 추출하지 못해도 에러가 발생하지 않습니다.

```
OPEN  cursor_name;
```

▣ 커서 패치(FETCH)

◈커서의 FETCH 는 현재 데이터 행을 OUTPUT 변수에 반환합니다.

◈커서의 SELECT 문의 컬럼의 수와 OUTPUT 변수의 수가 동일해야 합니다.

◈커서 컬럼의 변수의 타입과 OUTPUT 변수의 데이터 타입도 동일해야 합니다.

◈커서는 한 라인씩 데이터를 패치 합니다.

```
FETCH  cursor_name INTO variable1, variable2 ;
```

▣ 커서 닫기(CLOSE)

◈사용을 마친 커서는 반드시 닫아 주어야 합니다.

◈필요하다면 커서를 다시 열 수 있습니다.

◈커서를 닫은 상태에서 FETCH 를 할 수 없습니다.

```
CLOSE  cursor_name;
```

Explicit Cursor 예제

특정 부서의 평균급여와 사원수를 출력..

```
SQL>CREATE OR REPLACE PROCEDURE ExpCursor_Test
      (v_deptno    dept.deptno%TYPE)
      IS
      CURSOR dept_avg IS
      SELECT b.dname, COUNT(a.empno) cnt, ROUND(AVG(a.sal),3) salary
      FROM emp a, dept b
      WHERE a.deptno = b.deptno
         AND b.deptno = v_deptno
      GROUP BY b.dname ;

      -- 커서를 패치하기 위한 변수 선언
       v_dname    dept.dname%TYPE;
       emp_cnt     NUMBER;
       sal_avg     NUMBER;
    BEGIN
      -- 커서의 오픈
      OPEN dept_avg;
      -- 커서의 패치
      FETCH dept_avg INTO v_dname, emp_cnt, sal_avg;
        DBMS_OUTPUT.PUT_LINE('부서명 : ' || v_dname);
        DBMS_OUTPUT.PUT_LINE('사원수 : ' || emp_cnt);
        DBMS_OUTPUT.PUT_LINE('평균급여 : ' || sal_avg);
     -- 커서의 CLOSE
     CLOSE dept_avg;
    EXCEPTION

     WHEN OTHERS THEN

       DBMS_OUTPUT.PUT_LINE(SQLERRM||'에러 발생 ');

    END;
    /
```

```
프로시져가 생성되었습니다.

실행 결과
SQL> SET SERVEROUTPUT ON ; -- (DBMS_OUTPUT.PUT_LINE 을 출력하기 위해 사용)

SQL> EXECUTE ExpCursor_Test(30);
부서명 : SALES
사원수 : 6
평균급여 : 1550.833

PL/SQL 처리가 정상적으로 완료되었습니다.
```

FOR 문에서 커서 사용(Cursor FOR Loops)

◈ FOR 문을 사용하면 커서의 OPEN, FETCH, CLOSE 가 자동 발생하므로 따로 기술할 필요가 없습니다

◈ 레코드 이름도 자동 선언되므로 따로 선언할 필요가 없습니다.

FOR 문에서 커서 사용 예제

부서별 사원수와 급여 합계를 구하는 프로시저입니다.

```
SQL> CREATE OR REPLACE PROCEDURE ForCursor_Test
     IS
      CURSOR dept_sum IS
      SELECT b.dname, COUNT(a.empno) cnt, SUM(a.sal) salary
      FROM emp a, dept b
      WHERE a.deptno = b.deptno
      GROUP BY b.dname;
   BEGIN
    -- Cursor 를 FOR 문에서 실행시킨다
    FOR emp_list IN dept_sum LOOP
      DBMS_OUTPUT.PUT_LINE('부서명 : ' || emp_list.dname);
      DBMS_OUTPUT.PUT_LINE('사원수 : ' || emp_list.cnt);
      DBMS_OUTPUT.PUT_LINE('급여합계 : ' || emp_list.salary);
    END LOOP;
  EXCEPTION
    WHEN OTHERS THEN
      DBMS_OUTPUT.PUT_LINE(SQLERRM||'에러 발생 ');
  END;
  /
```

프로시져가 생성되었습니다.

실행 결과

SQL> SET SERVEROUTPUT ON ; -- (DBMS_OUTPUT.PUT_LINE 을 출력하기 위해 사용)

```
SQL> EXECUTE ForCursor_Test;
부서명 : ACCOUNTING
사원수 : 3
급여합계 : 8750
부서명 : RESEARCH
사원수 : 6
급여합계 : 10875
부서명 : SALES
사원수 : 6
급여합계 : 9305

PL/SQL 처리가 정상적으로 완료되었습니다.
```

▣ 명시적 커서의 속성(Explicit Cursor Attributes)

◈%ISOPEN

- 커서가 OPEN 되어 있으면 TRUE
- %ISOPEN 속성을 이용하여 커서가 열려있는지 알 수 있습니다.

◈%NOTFOUND

- 패치한 데이터가 행을 반환하지 않으면 TRUE
- %NOTFOUND 속성을 이용하여 루프를 종료할 시점을 찾습니다.

◈%FOUND

- 패치한 데이터가 행을 반환하면 TRUE

◈ %ROWCOUNT

- 현재까지 반환된 모든 데이터 행의 수
- %ROWCOUNT 속성을 이용하여 정확한 숫자만큼의 행을 추출합니다.

커서의 속성 예제

```
SQL>CREATE OR REPLACE PROCEDURE AttrCursor_Test
IS
    v_empno     emp.empno%TYPE;
```

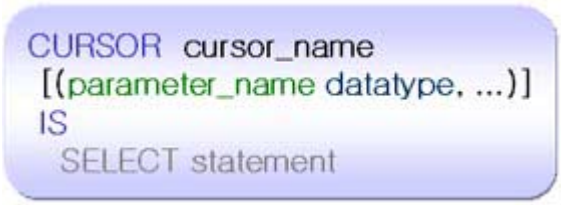
```
    v_ename     emp.ename%TYPE;
    v_sal       emp.sal%TYPE;
  CURSOR emp_list IS
    SELECT empno, ename, sal
    FROM emp;
BEGIN
  DBMS_OUTPUT.ENABLE;
  OPEN emp_list;
  LOOP
    FETCH emp_list INTO v_empno, v_ename, v_sal;
      -- 데이터를 찾지 못하면 빠져 나갑니다
      EXIT WHEN emp_list%NOTFOUND;
  END LOOP;
  DBMS_OUTPUT.PUT_LINE('전체데이터 수 ' || emp_list%ROWCOUNT);
  CLOSE emp_list;
 EXCEPTION
   WHEN OTHERS THEN
     DBMS_OUTPUT.PUT_LINE('ERR MESSAGE : ' || SQLERRM);
END;
/

프로시져가 생성되었습니다.

실행 결과
SQL> SET SERVEROUTPUT ON ; -- (DBMS_OUTPUT.PUT_LINE 을 출력하기 위해 사용)

SQL> EXECUTE AttrCursor_Test;
전체데이터 수 15

PL/SQL 처리가 정상적으로 완료되었습니다.
```

파라미터가 있는 커서(Cursors with Parameters)

◈ 커서가 열리고 질의가 실행되면 매개 변수 값을 커서에 전달한다.

◈ 다른 active set 을 원할때 마다 explicit 커서를 따로 선언해야 한다

▣ 문법(Syntax)



파라미터가 있는 커서 예제

```
SQL> CREATE OR REPLACE PROCEDURE ParamCursor_Test
     (param_deptno   emp.deptno%TYPE)
     IS
      v_ename     emp.ename%TYPE;
    -- Parameter가 있는 커서의 선언
     CURSOR emp_list(v_deptno emp.deptno%TYPE) IS
     SELECT ename       FROM emp       WHERE deptno = v_deptno;
    BEGIN
     DBMS_OUTPUT.ENABLE;
     DBMS_OUTPUT.PUT_LINE(' ****** 입력한 부서에 해당하는 사람들 ****** ');
    -- Parameter변수의 값을 전달(OPEN될 때 값을 전달한다)
     FOR emplst IN emp_list(param_deptno) LOOP
       DBMS_OUTPUT.PUT_LINE('이름 : ' || emplst.ename);
     END LOOP;
     EXCEPTION
       WHEN OTHERS THEN
         DBMS_OUTPUT.PUT_LINE('ERR MESSAGE : ' || SQLERRM);
     END;
     /
프로시져가 생성되었습니다.
```

실행 결과

```
SQL> SET SERVEROUTPUT ON ; -- (DBMS_OUTPUT.PUT_LINE을 출력하기 위해 사용)
SQL> EXECUTE ParamCursor_Test(10);
****** 입력한 부서에 해당하는 사람들 ******
이름 : CLARK
이름 : KING
이름 : MILLER

PL/SQL 처리가 정상적으로 완료되었습니다.
```

◆ WHERE CURRENT OF

- ROWID 를 이용하지 않고도 현재 참조하는 행을 갱신하고 삭제할 수 있게 합니다.
- 추가적으로 FETCH 문에 의해 가장 최근에 처리된 행을 참조하기 위해서
  "WHERE CURRENT OF 커서이름 " 절로 DELETE 나 UPDATE 문 작성이 가능합니다..
- 이 절을 사용할 때 참조하는 커서가 있어야 하며,
  FOR UPDATE 절이 커서 선언 query 문장 안에 있어야 합니다.
  그렇지 않으면 error 가 발생합니다..

WHERE CURRENT OF 예제

```
SQL> SET SERVEROUTPUT ON ;  -- DBMS_OUTPUT.PUT_LINE 을 출력하기 위해 사용
SQL>CREATE OR REPLACE PROCEDURE where_current
      IS
     CURSOR emp_list IS
        SELECT empno
        FROM emp
        WHERE empno = 7934
        FOR UPDATE;

    BEGIN

     --DBMS_OUTPUT.PUT_LINE 명령을 사용하기 위해서
     DBMS_OUTPUT.ENABLE;

     FOR emplst IN emp_list LOOP
        --emp_list 커서에 해당하는 사람의 직업을  SALESMAN 으로 업데이트 시킵니다.
        UPDATE emp
        SET job = 'SALESMAN'
        WHERE CURRENT OF emp_list;
       DBMS_OUTPUT.PUT_LINE('수정 성공');
     END LOOP;
     EXCEPTION
       WHEN OTHERS THEN
         -- 에러 발생시 에러 메시지 출력
         DBMS_OUTPUT.PUT_LINE('ERR MESSAGE : ' || SQLERRM);
END;
```

```
--먼저 데이터를 확인해 보세용
SQL> SELECT job FROM emp WHERE empno = 7934;
JOB
---------
CLERK
--PLSQL 을 실행시키고..
SQL> EXECUTE where_current;
수정 성공  --DBMS_OUTPUT.PUT_LINE 명령으로 출력한거..

PL/SQL 처리가 정상적으로 완료되었습니다.
-- 다시 데이터를 확인하면 변경된 것을 볼 수 있습니다.
SQL>  SELECT job FROM emp WHERE empno = 7934;

JOB
---------
SALESMAN
```

▣ 예외(Exception)란?

◈ 오라클 PL/SQL 의 오류를 예외라고 부릅니다.

◈ 오류는 PL/SQL 을 컴파일 할때 문법적인 오류로 발생하는 컴파일 타임 오류와,
프로그램을 실행할때 발생하는 실행타임 오류로 구분할 수 있습니다.

▣ PL/SQL 오류의 종류

예 외	설 명	처 리
미리 정의된 오라클 서버 오류 (Predefined Oracle Server)	PL/SQL 에서 자주 발생하는 약 20 개의 오류	선언할 필요도 없고, 발생시에 예외 절로 자동 트랩(Trap)된다.
미리 정의되지 않은 오라클 서버 오류 (Non-Predefined Oracle Server)	미리 정의된 오라클 서버 오류를 제외한 모든 오류	선언부에서 선언해야 하고 발생시 자동 트랩된다.
사용자 정의 오류 (User-Defined)	개발자가 정한 조건에 만족하지 않을경우 발생하는 오류	선언부에서 선언하고 실행부에서 RAISE 문을 사용하여 발생시켜야 한다

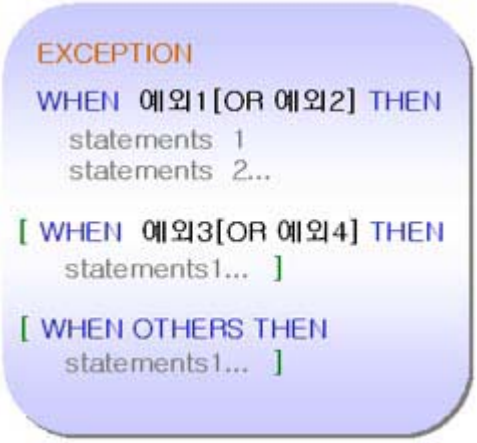
▣ Execption 문법(Syntax)

◈ WHEN OTHERS 절은 맨 마지막에 옵니다.

◈ 예외 처리절은 EXCEPTION 부터 시작합니다.

◈ 허용합니다.

◈ 예외가 발생하면 여러 개의 예외 처리부 중에 하나의 예외 처리부에 트랩(Trap)됩니다.



미리 정의된 예외(Predefined Exceptions)

◈ 오라클 PL/SQL 은 자주 일어나는 몇가지 예외를 미리 정의해 놓았으며, 이러한 예외는 개발자가 따로 선언할 필요가 없습니다.

▣ 미리 정의된 예외의 종류?

◈ NO_DATA_FOUND : SELECT 문이 아무런 데이터 행을 반환하지 못할때

◈ TOO_MANY_ROWS : 하나만 리턴해야하는 SELECT 문이 하나 이상의 행을 반환할 때

◈ INVALID_CURSOR : 잘못된 커서 연산

◈ ZERO_DIVIDE : 0 으로 나눌때

◈ DUP_VAL_ON_INDEX : UNIQUE 제약을 갖는 컬럼에 중복되는 데이터가 INSERT 될때

이 외에도 몇 개가 더 있습니다.

미리 정의된 예외 예제

```
SQL> CREATE OR REPLACE PROCEDURE PreException_test
       (v_deptno  IN emp.empno%TYPE)
  IS
     v_emp   emp%ROWTYPE;
  BEGIN
    DBMS_OUTPUT.ENABLE;
    SELECT empno, ename, deptno
    INTO v_emp.empno, v_emp.ename, v_emp.deptno
    FROM emp
    WHERE deptno = v_deptno ;
    DBMS_OUTPUT.PUT_LINE('사번 : ' || v_emp.empno);
    DBMS_OUTPUT.PUT_LINE('이름 : ' || v_emp.ename);
    DBMS_OUTPUT.PUT_LINE('부서번호 : ' || v_emp.deptno);
  EXCEPTION
    WHEN  DUP_VAL_ON_INDEX  THEN
      DBMS_OUTPUT.PUT_LINE('데이터가 존재 합니다.');
      DBMS_OUTPUT.PUT_LINE('DUP_VAL_ON_INDEX 에러 발생');

    WHEN  TOO_MANY_ROWS  THEN

     DBMS_OUTPUT.PUT_LINE('TOO_MANY_ROWS 에러 발생');
    WHEN  NO_DATA_FOUND  THEN
```

```
      DBMS_OUTPUT.PUT_LINE('NO_DATA_FOUND 에러 발생');
    WHEN  OTHERS  THEN
      DBMS_OUTPUT.PUT_LINE('기타 에러 발생');
 END;
 /
프로시저가 생성되었습니다.
```

프로시저 실행

```
SQL> SET SERVEROUTPUT ON ;  -- (DBMS_OUTPUT.PUT_LINE 을 출력하기 위해 사용)
SQL> EXECUTE PreException_Test(20);
TOO_MANY_ROWS 에러 발생
PL/SQL 처리가 정상적으로 완료되었습니다.
```

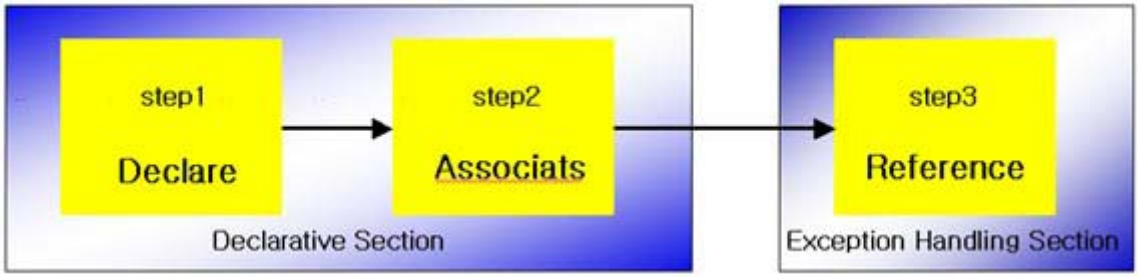
※ TOO_MANY_ROWS 에러를 타는 이유?

- SELECT 문의 결과가 1 개 이상의 행을 리턴하기 때문이다..
- TOO_MANY_ROWS 를 피하기 위해서는 FOR 문이나 LOOP 문으로 SELECT 문을 처리해야 합니다.

아래와 같이 바꾸면 에러가 발생하지 않습니다.

```
   FOR  emp_list  IN
      (SELECT empno, ename, deptno
      FROM emp
      WHERE deptno = v_deptno)  LOOP

      DBMS_OUTPUT.PUT_LINE('사번 : ' || emp_list.empno);
      DBMS_OUTPUT.PUT_LINE('이름 : ' || emp_list.ename);
      DBMS_OUTPUT.PUT_LINE('부서번호 : ' || emp_list.deptno);

   END LOOP;
```

미리 정의되지 않은 예외(Non-Predefiend Exceptions)



◈ STEP 1 : 예외의 이름을 선언(선언절)

◈ STEP 2 : PRAGMA EXCEPTION_INIT 문장으로 예외의 이름과 오라클 서버 오류 번호를 결합(선언절)

◈ STEP 3 : 예외가 발생할 경우 해당 예외를 참조한다(예외절)

미리 정의되지 않은 예외 예제

```
SQL> CREATE OR REPLACE PROCEDURE NonPreException_Test
     IS
       not_null_test EXCEPTION;    -- STEP 1
       /* not_null_test 는 선언된 예외 이름
          -1400 Error 처리번호는 표준 Oracle7 Server Error 번호 */
       PRAGMA EXCEPTION_INIT(not_null_test, -1400);      -- STEP 2

     BEGIN

       DBMS_OUTPUT.ENABLE;

     -- empno 를 입력하지 않아서 NOT NULL 에러 발생
     INSERT INTO emp(ename, deptno)
     VALUES('tiger', 30);

     EXCEPTION

     WHEN not_null_test THEN    -- STEP 3

        DBMS_OUTPUT.PUT_LINE('not null 에러 발생 ');

     END;
     /
```

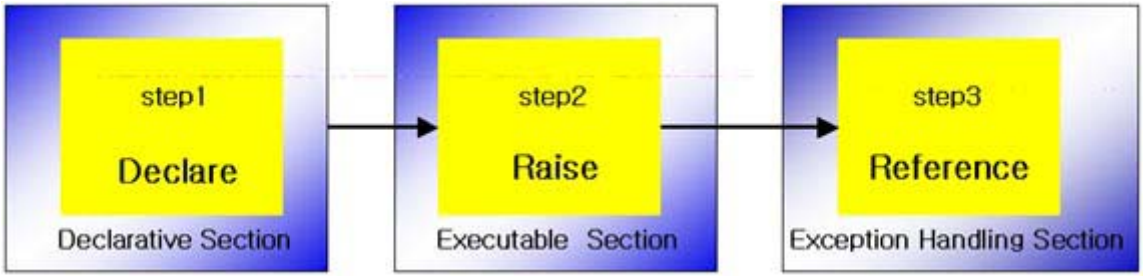
프로시져가 생성되었습니다.

실행 결과

```
SQL> SET SERVEROUTPUT ON ;    -- (DBMS_OUTPUT.PUT_LINE 을 출력하기 위해 사용)

SQL> EXECUTE NonPreException_Test;
not null 에러 발생

PL/SQL 처리가 정상적으로 완료되었습니다.
```

사용자 정의 예외(User-Defined Exceptions)

◈ 오라클 저장함수 RAISE_APPLICATION_ERROR 를 사용하여 오류코드 -20000 부터 -20999 의 범위 내에서 사용자 정의 예외를 만들수 있습니다.



◈ STEP 1 : 예외의 이름을 선언(선언절)

◈ STEP 2 : RAISE 문을 사용하여 직접적으로 예외를 발생시킨다(실행절)

◈ STEP 3 : 예외가 발생할 경우 해당 예외를 참조한다(예외절)

사용자 정의 예외 예제 Procedure

입력한 부서의 사원이 5 명보다 적으면 사용자 정의 예외가 발생하는 예제 입니다.

```
SQL>CREATE OR REPLACE PROCEDURE User_Exception
      (v_deptno IN emp.deptno%type )
    IS
     -- 예외의 이름을 선언
     user_define_error EXCEPTION;     -- STEP 1
     cnt     NUMBER;
    BEGIN
     DBMS_OUTPUT.ENABLE;
     SELECT COUNT(empno)
     INTO cnt
     FROM emp
     WHERE deptno = v_deptno;
     IF cnt < 5 THEN
      -- RAISE 문을 사용하여 직접적으로 예외를 발생시킨다
       RAISE user_define_error;        -- STEP 2
     END IF;

    EXCEPTION
     -- 예외가 발생할 경우 해당 예외를 참조한다.
     WHEN user_define_error THEN      -- STEP 3
```

```
        RAISE_APPLICATION_ERROR(-20001, '부서에 사원이 몇명 안되네요..');

  END;
 /


프로시져가 생성되었습니다.
```

실행 결과

```
SQL> SET SERVEROUTPUT ON ; -- (DBMS_OUTPUT.PUT_LINE 을 출력하기 위해 사용)

SQL> EXECUTE user_exception(10);
BEGIN user_exception(10); END;
 *
1 행에 오류:
ORA-20001: 부서에 사원이 몇명 안되네요..
ORA-06512: "SCOTT.USER_EXCEPTION", 줄 17 에서
ORA-06512: 줄 1 에서
```

10 부서의 사원이 5 보다 적기 때문에 사용자 정의 예외가 발생했습니다.

```
SQL> EXECUTE user_exception(20);
PL/SQL 처리가 정상적으로 완료되었습니다.
```

20 부서로 실행을 하면 에러가 발생하지 않는 것 을 알 수 있습니다..

SQLCODE, SQLERRM

◈WHEN OTHERS 문으로 트랩(Trap)되는 오류들의 실제 오류 코드와 설명을 볼때 사용한다.

◈ SQLCODE : 실행된 프로그램이 성공적으로 종료하였을때는 오류번호 0 을 포함하며,
그렇지 못할 경우에는 해당 오류코드 번호를 포함한다.

◈ SQLERRM : SQLCODE 에 포함된 오라클 오류 번호에 해당하는 메시지를 가진다.

SQLCODE Value	Description
0	오류 없이 성공적으로 종료
1	사용자 정의 예외 번호
+100	NO_DATA_FOUND 예외 번호
음수	위에 것을 제외한 오라클 서버 에러 번호

SQLCODE, SQLERRM 예제 프로시저

```
SQL> CREATE OR REPLACE PROCEDURE Errcode_Exception
      (v_deptno IN emp.deptno%type )
      IS
       v_emp   emp%ROWTYPE ;
      BEGIN
        DBMS_OUTPUT.ENABLE;
        -- ERROR 발생 for 문을 돌려야 됨
        SELECT *
        INTO v_emp
        FROM emp
        WHERE deptno = v_deptno;
      DBMS_OUTPUT.PUT_LINE('사번 : ' || v_emp.empno);
      DBMS_OUTPUT.PUT_LINE('이름 : ' || v_emp.ename);
    EXCEPTION

     WHEN OTHERS THEN

       DBMS_OUTPUT.PUT_LINE('ERR CODE : ' || TO_CHAR(SQLCODE));
       DBMS_OUTPUT.PUT_LINE('ERR MESSAGE : ' || SQLERRM);
```

```
END;
/
```

프로시져가 생성되었습니다.

실행 결과

```
SQL> SET SERVEROUTPUT ON ; -- (DBMS_OUTPUT.PUT_LINE 을 출력하기 위해 사용)

SQL> EXECUTE Errcode_Exception(30);
RR CODE : -1422
ERR MESSAGE : ORA-01422: 실제 인출은 요구된 것보다 많은 수의 행을 추출합니다

PL/SQL 처리가 정상적으로 완료되었습니다.
```

위와 같이 SQLCODE, SQLERRM 을 사용하면 에러 코드와 에러 메시지를 볼 수 있습니다.

package?

◆ 패키지(package)는 오라클 데이터베이스에 저장되어 있는 서로 관련있는 PL/SQL 프로지져와 함수들의 집합 입니다

◆ 패키지는 선언부와 본문 두 부분으로 나누어 집니다.

패키지 선언부

- 선언절은 패키지에 포함될 PL/SQL 프로시저나, 함수, 커서, 변수, 예외절을 선언 합니다.
- 패키지 선언부에서 선언한 모든 요소들은 패키지 전체에 적용됩니다.
- 즉 선언부에서 선언한 변수는 PUBLIC 변수로 사용 됩니다.

[Syntax]

```
CREATE [OR REPLACE] PACKAGE package_name IS | AS
    [ 변수선언절 ]
    [ 커서선언절 ]
    [ 예외선언절 ]
    [ Procedure 선언절 ]
    [ Function 선언절 ]
END package_name ;
```

패키지 본문

- 패키지 본문은 패키지에서 선언된 부분의 실행을 정의 합니다.
- 즉 실제 프로시져나 함수의 내용에 해당하는 부분이 옵니다.

[Syntax]

```
CREATE [OR REPLACE] PACKAGE BODY package_name IS | AS
    [ 변수선언절 ]
    [ 커서선언절 ]
    [ 예외선언절 ]
    [ Procedure 선언절 ]
    [ Function 선언절 ]
END package_name ;
```

아주 간단한 패키지 예제입니다.

4 개의 프로시저가 존재하고 있습니다.

프로시저명	프로시저 기능	보기
all_emp_info	모든 사원의 사원 정보 (사번, 성명, 입사일)	프로시저보기
all_sal_info	모든 사원의 급여 정보 (평균급여, 최고급여, 최소급여)	프로시저보기

dept_emp_info	특정 부서의 사원 정보 (사번, 성명, 입사일)	프로시저보기
dept_sql_info	특정 부서의 급여 정보 (평균급여, 최고급여, 최소급여)	프로시저보기

위 4 개의 프로시저를 가지고 패키지를 생성하겠습니다.

선언부를 먼저 생성 합니다.

package 예제 (선언부)

```
SQL>CREATE OR REPLACE PACKAGE emp_info AS
        PROCEDURE all_emp_info;                              -- 모든 사원의 사원 정보
        PROCEDURE all_sal_info;                              -- 모든 사원의 급여 정보
        PROCEDURE dept_emp_info (v_deptno IN  NUMBER) ;   -- 특정 부서의 사원 정보
        PROCEDURE dept_sal_info (v_deptno IN  NUMBER) ;    -- 특정 부서의 급여 정보
     END emp_info;
Package created.
```

선언부를 생성 하고 나서 본문 부분을 생성 합니다.

package 예제 (본문)

```
 SQL>CREATE OR REPLACE PACKAGE BODY emp_info AS
          -- 모든 사원의 사원 정보
          PROCEDURE all_emp_info
          IS
               CURSOR emp_cursor IS
               SELECT empno, ename, to_char(hiredate, 'RRRR/MM/DD') hiredate
               FROM emp
               ORDER BY hiredate;
          BEGIN
              FOR  aa  IN emp_cursor LOOP
                   DBMS_OUTPUT.PUT_LINE('사번 : ' || aa.empno);
                    DBMS_OUTPUT.PUT_LINE('성명 : ' || aa.ename);
                    DBMS_OUTPUT.PUT_LINE('입사일 : ' || aa.hiredate);
               END LOOP;
          EXCEPTION
               WHEN OTHERS THEN
                    DBMS_OUTPUT.PUT_LINE(SQLERRM||'에러 발생 ');
           END all_emp_info;
          -- 모든 사원의 급여 정보
          PROCEDURE all_sal_info
```

```
IS
    CURSOR emp_cursor IS
    SELECT round(avg(sal),3) avg_sal, max(sal) max_sal, min(sal) min_sal
    FROM emp;
        BEGIN
    FOR  aa  IN emp_cursor LOOP
        DBMS_OUTPUT.PUT_LINE('전체급여평균 : ' || aa.avg_sal);
        DBMS_OUTPUT.PUT_LINE('최대급여금액 : ' || aa.max_sal);
        DBMS_OUTPUT.PUT_LINE('최소급여금액 : ' || aa.min_sal);
    END LOOP;
EXCEPTION
    WHEN OTHERS THEN
         DBMS_OUTPUT.PUT_LINE(SQLERRM||'에러 발생 ');
END all_sal_info;
--특정 부서의  사원 정보
PROCEDURE dept_emp_info (v_deptno IN  NUMBER)
IS
    CURSOR emp_cursor IS
    SELECT empno, ename, to_char(hiredate, 'RRRR/MM/DD') hiredate
    FROM emp
    WHERE deptno = v_deptno
    ORDER BY hiredate;
BEGIN
     FOR  aa  IN emp_cursor LOOP
        DBMS_OUTPUT.PUT_LINE('사번 : ' || aa.empno);
        DBMS_OUTPUT.PUT_LINE('성명 : ' || aa.ename);
        DBMS_OUTPUT.PUT_LINE('입사일 : ' || aa.hiredate);
    END LOOP;
EXCEPTION
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE(SQLERRM||'에러 발생 ');
END dept_emp_info;
--특정 부서의  급여 정보
PROCEDURE dept_sal_info (v_deptno IN  NUMBER)
IS
    CURSOR emp_cursor IS
```

```
            SELECT round(avg(sal),3) avg_sal, max(sal) max_sal, min(sal) min_sal
            FROM emp
            WHERE deptno = v_deptno;
        BEGIN
            FOR  aa  IN emp_cursor LOOP
                DBMS_OUTPUT.PUT_LINE('전체급여평균 : ' || aa.avg_sal);
                DBMS_OUTPUT.PUT_LINE('최대급여금액 : ' || aa.max_sal);
                DBMS_OUTPUT.PUT_LINE('최소급여금액 : ' || aa.min_sal);
            END LOOP;

        EXCEPTION
            WHEN OTHERS THEN
                DBMS_OUTPUT.PUT_LINE(SQLERRM||'에러 발생 ');

        END dept_sal_info;

    END emp_info;
    /
Package body created.
```

패키지의 실행

패키지의 실행은 패키지 명 다음에 .을 찍고 프로시저냐 함수 명을 써 줍니다.

먼저 set serveroutput on 을 실행한후..

```
SQL> SET SERVEROUTPUT ON ; -- (DBMS_OUTPUT.PUT_LINE 을 출력하기 위해 사용)
```

다음 명령들을 실행해 보세요..

```
SQL> exec emp_info.all_emp_info;
SQL> exec emp_info.all_sal_info;
SQL> exec emp_info.dept_emp_info(10);
SQL> exec emp_info.dept_sal_info(10);
```

트리거란?

◆ INSERT, UPDATE, DELETE 문이 TABLE 에 대해 행해질 때 묵시적으로 수행되는 PROCEDURE 입니다.

◆ Trigger 는 TABLE 과는 별도로 DATABASE 에 저장됩니다.

◆ Trigger 는 VIEW 에 대해서가 아니라 TABLE 에 관해서만 정의될 수 있습니다.

[Syntax]

- BEFORE : INSERT, UPDATE, DELETE 문이 실행되기 전에 트리거가 실행됩니다.
- AFTER : INSERT, UPDATE, DELETE 문이 실행된 후 트리거가 실행됩니다.
- trigger_event : INSERT, UPDATE, DELETE 중에서 한 개 이상 올 수 있습니다.
- FOR EACH ROW : 이 옵션이 있으면 행 트리거가 됩니다.

-- 행 트리거 : 컬럼의 각각의 행의 데이터 행 변화가 생길 때마다 실행되며, 그 데이터 행의 실제 값을 제어할 수 있습니다.

-- 문장 트리거 : 트리거 사건에 의해 단 한번 실행되며, 컬럼의 각 데이터 행을 제어할 수 없습니다.

간단한 행 트리거 예제

```
SQL>CREATE OR REPLACE TRIGGER triger_test
BEFORE
UPDATE ON dept
FOR EACH ROW
BEGIN
DBMS_OUTPUT.PUT_LINE('변경 전 컬럼 값 : ' || : old.dname);
DBMS_OUTPUT.PUT_LINE('변경 후 컬럼 값 : ' || : new.dname);
END;
/
```

```
SQL> SET SERVEROUTPUT ON ; -- (DBMS_OUTPUT.PUT_LINE 을 출력하기 위해 사용)
-- UPDATE 문을 실행시키면..
```

```
SQL>UPDATE dept
SET dname = '총무부'
WHERE deptno = 30
-- 트리거가 자동 실행되어 결과가 출력됩니당.
변경 전 컬럼 값 : 인사과
변경 후 컬럼 값 : 총무부
1 행이 갱신되었습니다.
```

간단한 행 트리거 예제 2 (PLSQL BLOCK 이 있는 트리거)

```
SQL>CREATE OR REPLACE trigger sum_trigger
 BEFORE
 INSERT OR UPDATE ON emp
 FOR EACH ROW
DECLARE
  -- 변수를 선언할 때는 DECLARE 문을 사용해야 합니다
  avg_sal NUMBER;
BEGIN
  SELECT ROUND(AVG(sal),3)  INTO avg_sal  FROM emp;
  DBMS_OUTPUT.PUT_LINE('급여 평균 : ' || avg_sal);
END;
/
트리거가 생성되었습니다.
```

```
SQL> SET SERVEROUTPUT ON ; -- (DBMS_OUTPUT.PUT_LINE 을 출력하기 위해 사용)
-- INSERT 문을 실행합니다..
 SQL> INSERT INTO EMP(EMPNO, ENAME, JOB, HIREDATE, SAL)
 VALUES(1000, 'LION', 'SALES', SYSDATE, 5000);
-- INSERT 문을 실행되기 전까지의 급여 평균이 출력됩니다.
 급여 평균 : 2073.214
 1 개의 행이 만들어졌습니다.
```