<< Less 03 : 실행문 작성 >>

중요 학습 내용:

- 실행부분 코팅(BIGIN..... END;)
- 함수사용방법
- 형변환 함수 주의
- 중첩 블록(Nested block) 작성 및 레이블로 변수 한정

```
==================================
        << 익명 PL/SQL Block 구조 >>
==================================
        DECLARE
                변수선언 및 초기화;
        BEGIN
                실행로직을 구현;
        END;
        /
==================================
```

- 본 문서의 실습은 별도의 언급이 없는 한, SQL*Developer를 이용하여 hr 계정으로 데이터베이스에 원격하여 수행합니다.

<< PL/SQL 블록에서 Lexical Units 분류 >>

- 식별자(Identifiers)
- 구분자(Delimiters)
- 리터럴(Literals)
- 주석(Comments)

<< PL/SQL 블록에서 SQL 함수 사용하는 방법 >>

- PL/SQL 블록 구조에서 다음의 함수는 사용할 수 없습니다.
  - -DECODE() 함수
  - -그룹 함수(SUM(), AVG(), COUNT(), MIN(), MAX())

```
DECLARE
  desc_size INTEGER(5);
  a         INTEGER (6);
  prod_description VARCHAR2(70):='This is Good';
  -- get the length of the string in prod_description
  emp_name  VARCHAR2(30) := 'SHIN' ;
...
...
BEGIN
  emp_name  := LOWER(emp_name);
  desc_size := LENGTH(prod_description);
  desc_size := SUM(a);  -- 에러입니다!! 이렇게는 못 사용합니다.
...
...
```

(참고) 집합함수 및 DECODE() 함수는 아래처럼
PL/SQL 코드 내에 포함된 SQL 문장 내에서
사용할 수는 있습니다.

```
DECLARE
  sum_sal  NUMBER(10,2);
BEGIN
  SELECT sum(salary) INTO sum_sal
  FROM employees ; -- 그룹함수는 SQL문장 안에서 사용됨
END;
/
```

<< Data Type Conversion >>

• Some conversion functions:
- TO_CHAR
- TO_DATE **
- TO_NUMBER
- TO_TIMESTAMP **
- TO_TIMESTAMP_TZ **
- TO_YMINTERVAL('YY-MM') -- YY, MM 은 정수
- TO_DSINTERVAL('DD HH24:MI:SS') -- DD,HH24,MI,SS 는 정수

<< 데이터 유형 변형 실습 >>

1> SQL*Developer를 이용하여 hr 계정으로 데이터베이스에 원격접속합니다.

2> SQL*Plus (또는 SQL*Developer)에서
DBMS_OUTPUT.put_line 프로시저의 실행 결과를 표시하기 위하여
출력 옵션 설정하고 세션의 DATE 표시형식을 다음처럼 설정합니다.

```
SQL> SET SERVEROUTPUT ON
```

3> 다음의 익명블록을 작성하여 실행합니다.

```
SQL> DECLARE
        salary       NUMBER(6)  :=6000;
        sal_hike     VARCHAR2(5):='100.7';
        total_salary salary%TYPE;
     BEGIN
        total_salary
              :=salary+sal_hike;
        DBMS_OUTPUT.PUT_LINE(total_salary);
     END;
     /
6101

PL/SQL procedure successfully completed.

SQL>
```

4> 접속 세션의 NLS_DATE_FORMAT 및 NLS_DATE_LANGUAGE를
각각 다음과 같이 설정합니다.

```
SQL> ALTER SESSION SET nls_date_format='DD-MON-YYYY' ;

Session altered.
```

```
SQL> ALTER SESSION SET nls_date_language=american ;

Session altered.

SQL>

     date_of_joining DATE := '02-Feb-2000'; <-- 자동형변환

     date_of_joining DATE := 'February 02,2000';  <-- 에러

     date_of_joining DATE := TO_DATE('February 02,2000'
                                    ,'Month DD, YYYY');
```

<< 중첩 블록(Nested Blocks) 개요 >>

```
                        DECLARE
                                ...
                        BEGIN
                                ...
                                DECLARE
                                        ...
                                BEGIN
                                        ...
                                END;
                        ... ;
                        ... ;

                        EXCEPTION
                                ...
                                DECLARE
                                        ...
                                BEGIN
                                        ...
                                END;
                       END;
                       /
```

```
================================================================
<< 변수의 범위 규칙 >>
================================================================
OUTER BLOCK에서 선언된 변수는 INNER BLOCK에서도 참조할 수 있음.
----------------------------------------------------------------
INNER BLOCK에서 선언된 변수는 OUTER BLOCK에서는 참조할 수 없음.
================================================================
-- 전역변수, 로컬변수

(중첩블록의 변수 범위 실습 예제)

1> SQL*Developer 에서
   DBMS_OUTPUT.put_line 프로시저의 실행 결과를 표시하기 위하여
   출력 옵션 설정

SQL> SET SERVEROUTPUT ON
SQL>


2> 다음의 익명블록을 작성하여 실행합니다.

SQL> DECLARE
           outer_variable VARCHAR2(20):='GLOBAL VARIABLE';
     BEGIN
           DECLARE
```

```
            inner_variable VARCHAR2(20):='LOCAL VARIABLE';
        BEGIN
            DBMS_OUTPUT.PUT_LINE('nested : '||inner_variable);
            DBMS_OUTPUT.PUT_LINE('nested : '||outer_variable);
        END;
        DBMS_OUTPUT.PUT_LINE(outer_variable);
    END;
    /
nested : LOCAL VARIABLE
nested : GLOBAL VARIABLE
GLOBAL VARIABLE

PL/SQL procedure successfully completed.

SQL>
```

==========

(비교) OUTER-블록에서 INNER-블록에 선언된 변수를 사용하려고 시도하면
아래처럼 에러가 발생됩니다.

```
SQL> DECLARE
        outer_variable VARCHAR2(20):='GLOBAL VARIABLE';
    BEGIN
        DECLARE
            inner_variable VARCHAR2(20):='LOCAL VARIABLE';
        BEGIN
            DBMS_OUTPUT.PUT_LINE('nested : '||inner_variable);
            DBMS_OUTPUT.PUT_LINE('nested : '||outer_variable);
        END;
        DBMS_OUTPUT.PUT_LINE(outer_variable);
        DBMS_OUTPUT.PUT_LINE(inner_variable);
    END;
    /
ERROR at line 11:
ORA-06550: line 11, column 27:
PLS-00201: identifier 'INNER_VARIABLE' must be declared
ORA-06550: line 11, column 6:
PL/SQL: Statement ignored


SQL>
```

<< 변수 범위(Scope) 및 가시성(Visibility) >>

```
-- SQL*Plus (또는 SQL*Developer)에서
   DBMS_OUTPUT.put_line 프로시저의 실행 결과를 표시하기 위하여
   출력 옵션 설정하고 세션의 DATE 표시형식을 다음처럼 설정합니다.

SQL> SET SERVEROUTPUT ON
SQL>

SQL> ALTER SESSION SET nls_date_format='DD-MON-YYYY' ;

Session altered.

SQL>
SQL> ALTER SESSION SET nls_date_language=american ;

Session altered.
```

```
SQL>

SQL> DECLARE
        father_name   VARCHAR2(20) := 'Patrick';
        date_of_birth DATE         := '20-Apr-1972';
     BEGIN
          DECLARE
            child_name    VARCHAR2(20):= 'Mike';
            date_of_birth DATE:= '12-Dec-2002';
          BEGIN
            DBMS_OUTPUT.PUT_LINE('Father''s Name: '||father_name);
            DBMS_OUTPUT.PUT_LINE('Date of Birth: '||date_of_birth);
            DBMS_OUTPUT.PUT_LINE('Child''s Name: '||child_name);
          END;
        DBMS_OUTPUT.PUT_LINE('Date of Birth: '||date_of_birth);
     END;
     /
Father's Name: Patrick
Date of Birth: 12-DEC-2002
Child's Name: Mike
Date of Birth: 20-APR-1972

PL/SQL procedure successfully completed.

SQL>
        ============================================================
        하나의 블록에서 같은 이름의 변수는 사용할 수 없습니다.
        그러나 2개의 다른 블록(중첩블록들)에서는 같은 이름을 가지는
        변수를 선언할 수 있습니다.
        ============================================================



<< 식별자의 명확한 지정( Qualify an Identifier) >>


SQL> SET SERVEROUTPUT ON
SQL>

SQL> <<o_block1>>
     DECLARE
        father_name VARCHAR2(20):='Patrick';
        date_of_birth DATE:='20-Apr-1972';
     BEGIN
       <<n_block2>>
          DECLARE
            child_name VARCHAR2(20):='Mike';
            date_of_birth DATE:='12-Dec-2002';
          BEGIN
            DBMS_OUTPUT.PUT_LINE('Father''s Name : '||o_block 1.father_name);
            DBMS_OUTPUT.PUT_LINE('Date of Birth : ' ||o_block 1.date_of_birth);
            DBMS_OUTPUT.PUT_LINE('Child''s Name  : '||child_name);
            DBMS_OUTPUT.PUT_LINE('Date of Birth : ' ||n_block 2.date_of_birth);
          END;
      DBMS_OUTPUT.PUT_LINE('Date of Birth : '||o_block1.date_of_birth);
    END;
    /
Father's Name : Patrick
Date of Birth : 20-APR-1972
Child's Name  : Mike
Date of Birth : 12-DEC-2002
Date of Birth : 20-APR-1972

PL/SQL procedure successfully completed.

SQL>
```

(주의) 블록이름을 변수에 이용하더라도
외부블록에서 내부블록의 변수를
호출하는 것은 안됩니다.

```
SQL> <<o_block1>>
     DECLARE
        father_name VARCHAR2(20):='Patrick';
        date_of_birth DATE:='20-Apr-1972';
     BEGIN
        <<n_block2>>
         DECLARE
           child_name VARCHAR2(20):='Mike';
           date_of_birth DATE:='12-Dec-2002';
         BEGIN
           DBMS_OUTPUT.PUT_LINE('Father''s Name: '||father_name);
           DBMS_OUTPUT.PUT_LINE('Date of Birth: '||o_block1.date_of_birth);
           DBMS_OUTPUT.PUT_LINE('Child''s Name: '||child_name);
           DBMS_OUTPUT.PUT_LINE('Date of Birth: '||n_block2.date_of_birth);
         END;
      DBMS_OUTPUT.PUT_LINE('Date of Birth: '||n_block2.date_of_birth);
     END;
     /
      DBMS_OUTPUT.PUT_LINE('Date of Birth: '||n_block2.date_of_birth);
                                                 *
ERROR at line 16:
ORA-06550: line 16, column 56:
PLS-00219: label 'N_BLOCK2' reference is out of scope
ORA-06550: line 16, column 7:
PL/SQL: Statement ignored


SQL>
```

<< Variable Scope 결정 >>

```
SQL> <<outer>>
     DECLARE
                 sal  NUMBER(7,2) := 60000;
                 comm NUMBER(7,2) := sal * 0.20;
                 message VARCHAR2(255) := ' eligible for commission';
     BEGIN
                 DECLARE
                     sal  NUMBER(7,2) := 50000;
                     comm NUMBER(7,2) := 0;
                     total_comp NUMBER(7,2) := sal + comm;
                 BEGIN
                       DBMS_OUTPUT.PUT_LINE('01: '||sal);
                       DBMS_OUTPUT.PUT_LINE('02: '||comm);
                       DBMS_OUTPUT.PUT_LINE('03: '||total_comp);
                       DBMS_OUTPUT.PUT_LINE('04: '||outer.sal);
                       DBMS_OUTPUT.PUT_LINE('05: '||outer.comm);
                       DBMS_OUTPUT.PUT_LINE('06: '||outer.message);
                     message := 'CLERK not'||message;
                     outer.comm := sal * 0.30;
                       DBMS_OUTPUT.PUT_LINE('07: '||sal);
                       DBMS_OUTPUT.PUT_LINE('08: '||comm);
                       DBMS_OUTPUT.PUT_LINE('09: '||total_comp);
                       DBMS_OUTPUT.PUT_LINE('10: '||outer.sal);
                       DBMS_OUTPUT.PUT_LINE('11: '||outer.comm);
                       DBMS_OUTPUT.PUT_LINE('12: '||outer.message);
                   END;
```

```
                message := 'SALESMAN '||message;
                DBMS_OUTPUT.PUT_LINE(' 13: '||message);
     END;
     /
01: 50000  <--child sal
02: 0      <--child comm
03: 50000  <--child sal+child comm
04: 60000  <--parent sal
05: 12000  <--parent sal*parent comm
06:  eligible for commission  <--parent message
07: 50000  <--child sal
08: 0      <--child comm
09: 50000  <--child sal+child comm
10: 60000  <--parent sal
11: 15000  <--child sal * 0.3
12: CLERK not eligible for commission <--child 에서 변경
13: SALESMAN CLERK not eligible for commission

PL/SQL procedure successfully completed.

SQL>
```

<< PL/SQL 블록의 실행부(BEGIN...END;)에서 연산자(Operators) >>

- Increment the counter for a loop.

```
loop_count := loop_count + 1;
```

- Set the value of a Boolean flag.

```
good_sal := sal BETWEEN 50000 AND 150000;
```

- Validate whether an employee number contains a value.

```
valid := (empno IS NOT NULL);
```

==========
<< NULL 처리시 주의할 규칙 >>
==========

- NULL을 포함하는 비교들은 항상 NULL을 산출한다
- NULL에 NOT 연산자를 적용하는 것은 NULL을 산출한다
- [IF--THEN]문장(conditional control statements)에서
  만약 IF 절의 조건이 NULL을 산출하면
  THEN 절에 명시된 일련의 관련 문장들은
  실행되지 않습니다.

==========

<< Programming Guidelines >>

- 주석 처리
- 대소문자 규정을 정하고 이를 준수
- identifiers와 다른 객체들에 대하여 이름-지정-규칙 준수
- indenting(들여쓰기)을 사용해서 코드의 가독성을 향상하세요.(3-21)

```
BEGIN
IF x=0 THEN
y:=1;
END IF;
END;
/



BEGIN
      IF x=0 THEN
                  y:=1;
      END IF;
END;
/
```

```
SQL> SET SERVEROUTPUT ON
SQL>
SQL> DECLARE
        v_test NUMBER(10) := 0 ;
     BEGIN
        v_test := 4**2 ;
        dbms_output.put_line(
                        'SHIN'||CHR(10)||
                        'ORACLE'||CHR(10)||
                        'FIGHTING'||'----'||v_test) ;
     END ;
     /
SHIN
ORACLE
FIGHTING----16

PL/SQL procedure successfully completed.

SQL>
```