==========

<< Less 06 조합 데이터 유형 작업 >>

==========

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

<< 조합 데이터 유형(COMPOSITE DATA-TYPE) >>

사용목적: 서로 관련되어 있는 여러 개의 데이터들을
논리적으로 하나의 단위로 취급하여
PL/SQL 프로그램에서 처리하고 싶을 때 사용합니다.

• 스칼라 데이터유형과는 달리 여러 개의 값을 HOLD 할 수 있습니다.

• 스칼라 데이터유형과는 달리 [내부 구성 요소]를 포함합니다.

• 2 가지 유형이 있습니다

**1) PL/SQL Records : 여러 개의 데이터유형로 조합된 값들

2) PL/SQL Collections: 하나의 데이터 유형으로 된 값들

• INDEX BY tables or associative arrays
- INDEX BY PLS_INTEGER
- INDEX BY VARCHAR2

• Nested table

• VARRAY

==========

<< PL/SQL Records >>

==========

<< PL/SQL Records 개요 >>

테이블로 부터 SELECT 문으로 "하나의 레코드"를 반환받아 이를
변수에 Hold할 때,

• 개별적인 스칼라-데이터유형의 변수에 각각 Hold 할 수도 있고,

• 반환된 레코드를 구성하는 개별 필드의 데이터형식으로
구성된 [사용자 정의 Record 형식]의 변수를 이용하여
하나의 논리적 단위로 Hold 할 수 있습니다.

<< PL/SQL Records 특징 >>

• 하나의 데이터 유형(DataType)으로만 구성되는 Collection과는 달리
둘 이상의 다른 데이터 유형(DataType)으로 구성될 수 있습니다.

즉, Scalar 데이터 유형, Record 데이터 유형 또는
Index-by-Table 데이터 유형을 필요한 만큼 포함할 수 있습니다.

• 데이터베이스 테이블의 행과 동일하지 않습니다.

• [필드들의 Collection]을 하나의 논리적 단위로 처리합니다.

- 기본적으로 테이블에서 하나의 Data-Row를
  Fetch하여 처리하는데 편리합니다.
- PL/SQL Record에 초기값을 할당하고 NOT NULL로 정의할 수 있습니다.
- 초기 값이 없는 필드는 NULL로 초기화 됩니다.
- 필드를 정의할 때 DEFAULT 키워드를 사용할 수 있습니다.
- 중첩 레코드를 선언하고 참조할 수 있습니다.
  레코드는 다른 레코드의 구성요소가 될 수 있습니다.
- RECORD 유형을 정의하고 사용자 정의 레코드를 블록,
  서브프로그램 또는 패키지의 선언 부분에서 선언할 수 있습니다.

<< PL/SQL Record 생성 >>

- PL/SQL Record 선언방법(2단계로 수행됩니다)

  (1) 레코드 데이터 유형(DATA-TYPE)을 생성.

  (2) 선언된 데이터 유형을 사용하는 변수(식별자)를 선언.

[[Record type 생성 문법]]

```
TYPE 레코드_유형_이름 IS RECORD
        (field_declaration
        [, field_declaration]. . .);
```

• field_declaration 문법

```
필드_이름 { field_type
          | variable%TYPE
          | table.column%TYPE
          | table%ROWTYPE }
                    [ [NOT NULL]
                      {:= | DEFAULT} expr]
```

[[생성된 Record type을 이용하는 변수를 선언하는 문법]]

```
변수명   선언된_레코드_유형_이름 ;
```

(참고) 교재에서 PL/SQL record라고 하면,
record datatype을 사용하는 변수명을 의미합니다.

즉, 아래의 예제에서

```
emp_record_type은 Record-Datatype을
emp_record     는 emp_record_type의 Record를
```

각각 의미합니다.

<< PL/SQL 레코드 변수 선언 예제 >>

```
SQL> set serveroutput on
SQL>
```

```
SQL> DECLARE

          v_salary       NUMBER(8,2) ;  -- 스칼라 형식 변수

          -- 레코드 데이터 유형을 선언

          TYPE emp_record_type IS RECORD
                      (r_employee_id  NUMBER(6) NOT NULL := 100,
                       r_last_name    hr.employees.last_name%TYPE,
                       r_job_id       hr.employees.job_id%TYPE,
                       r_salary       NUMBER(8,2));


          -- 선언된 레코드 데이터 유형을 사용하는 변수를 선언.

          emp_record emp_record_type ;

     BEGIN

           -- SELECT 문의 결과-레코드를
           -- 선언된 emp_record 레코드 변수에 Hold


            select employee_id, last_name, job_id, salary
                                                 INTO emp_record
            from hr.employees2
            where  employee_id = 100 ;

                   /* RECORD 변수에 Hold된 값을 사용 시에는             */
                   /* Record의 개별 값을 [레코드변수명.필드명] 형식으로 */
                   /* 호출하여 사용함                                   */

                    DBMS_OUTPUT.PUT_LINE('----------------------');
                    DBMS_OUTPUT.PUT_LINE(emp_record.r_employee_id);
                    DBMS_OUTPUT.PUT_LINE(emp_record.r_last_name);
                    DBMS_OUTPUT.PUT_LINE(emp_record.r_job_id);
                    DBMS_OUTPUT.PUT_LINE(emp_record.r_salary);
     END;
     /
----------------------
100
King
AD_PRES
24000

PL/SQL procedure successfully completed.

SQL>



SQL> DECLARE
          TYPE emp_record_type IS RECORD
              (r_employee_id NUMBER(6) NOT NULL := 100,
               r_last_name   hr.employees.last_name%TYPE,
               r_job_id      hr.employees.job_id%TYPE,
               r_salary      NUMBER(8,2));

          emp_record emp_record_type ;

     BEGIN
          select employee_id, last_name, job_id, salary
                                                 into emp_record
          from   hr.employees2
          where  employee_id = 100 ;
```

```
        DBMS_OUTPUT.PUT_LINE('----------------------');

                /*  변수에 Hold된 값을 사용 시에 아래처럼 */
                /*  레코드명만 적으면 Error가 발생됩니다. */

                 DBMS_OUTPUT.PUT_LINE(emp_record);
            end;
            /
        ERROR at line 13:
        ORA-06550: line 13, column 5:
        PLS-00306: wrong number or types of arguments in call to 'PUT_LINE'
        ORA-06550: line 13, column 5:
        PL/SQL: Statement ignored
```

<< PL/SQL Record 구조 >>

- emp_record 변수의 내부 구성 요소 [구조]

r_employee_id	r_last_name	r_job_id	r_salary
100	King	AD_PRES	24000

- 레코드의 필드 액세스 : [record_name.field_name]

  (예) record type의 emp_record 레코드 변수의
  job_id 필드를 참조

  emp_record.job_id := 'ST_CLERK' ;

```
SQL> DECLARE
          TYPE emp_record_type IS RECORD
               (r_employee_id  NUMBER(6)  NOT NULL      := 100
               ,r_last_name    hr.employees.last_name%TYPE
               ,r_job_id       hr.employees.job_id%TYPE
               ,r_salary       NUMBER(8,2));

          emp_record emp_record_type ;
     begin
                    select employee_id, last_name, job_id, salary
                                                 into emp_record
                    from   hr.employees2
                    where  employee_id = 100 ;

                   /* emp_record 레코드 변수의 job_id 요소에   */
                   /* 기존값(AD_PRES)를 'ST_CLERK' 값으로 변경 */

                    emp_record.r_job_id := 'ST_CLERK' ;
                                        -- 값을 .
                    DBMS_OUTPUT.PUT_LINE('----------------------');
                    DBMS_OUTPUT.PUT_LINE(emp_record.r_employee_id);
                    DBMS_OUTPUT.PUT_LINE(emp_record.r_last_name);
                    DBMS_OUTPUT.PUT_LINE(emp_record.r_job_id);
                    DBMS_OUTPUT.PUT_LINE(emp_record.r_salary);
     end;
     /
----------------------
100
King
ST_CLERK    <-- 처음의 값은 'AD_PRES'가 'ST_CLERK'로 변경됨.
24000
```

<< %ROWTYPE Attribute >>

- 특정 테이블이나 view 의 행을 구성하는 전체필드 조합을 의미.
- Record-데이터 유형을 선언할 필요가 없음.

```
DECLARE
  emp_record1 hr.employees%ROWTYPE;

  TYPE emp_record_type IS RECORD
  (r_employee_id   employees.employee_id%TYPE NOT NULL :=  100,
   r_last_name     employees.last_name%TYPE,
   r_job_id        employees.job_id%TYPE,
   r_salary        employees.salary%TYPE);

  emp_record    emp_record_type ;
```

```
                ===================================
                << %ROWTYPE Attribute (6-12) >>
                ===================================
                  레코드의 특정 필드를 참조하려면,
                  아래처럼 적으면됩니다.

                예)  emp_record1.commission_pct:= .35;
```

예제)

```
SQL> DECLARE
          v_empid    number(6) ;
          v_mgrid    v_empid%TYPE ;
          v_empname  hr.employees.last_name%TYPE ;

           /* 레코드형식 선언 */

          TYPE emp_record_type IS RECORD
              (r_employee_id   NUMBER(6) NOT NULL := 100
              ,r_last_name     hr.employees.last_name%TYPE
              ,r_job_id        hr.employees.job_id%TYPE
              ,r_salary        NUMBER(8,2)
              ,r_department_id  hr.employees.department_id%TYPE);

           /* 선언된 레코드 형식을 사용하는 레코드(변수) */

          emp_record   emp_record_type ;

          dept_rec     hr.departments2%ROWTYPE ;

          emp_record2  emp_record%TYPE ;

                        /* 주의: 잘못된 코드                              */
                        /* emp_record2  emp_record%ROWTYPE;               */
                        /* %TYPE과는 다르게 앞에 선언된  Record-Type을 */
                        /* 참조할 수 없음                                 */

                      /* departments2 테이블행의 Datatype 조합을 사용하는 */
                      /* 레코드(변수)                                    */

     begin
            select employee_id, last_name, job_id
                  ,salary,department_id
                                      into emp_record2
            from   hr.employees2
            where  employee_id = 100 ;
```

```
                    /* emp_record 레코드 변수의 job_id 요소에   */
                    /* 기존값(AD_PRES)를 'ST_CLERK' 값으로 변경 */

            emp_record2.r_job_id := 'ST_CLERK' ;

            DBMS_OUTPUT.PUT_LINE('--------------------');
            DBMS_OUTPUT.PUT_LINE(emp_record2.r_employee_id);
            DBMS_OUTPUT.PUT_LINE(emp_record2.r_last_name);
            DBMS_OUTPUT.PUT_LINE(emp_record2.r_job_id);
            DBMS_OUTPUT.PUT_LINE(emp_record2.r_salary);

            select * into dept_rec
            from   hr.departments2
            where  department_id = emp_record2.r_department_id;

            DBMS_OUTPUT.PUT_LINE('====================');
            DBMS_OUTPUT.PUT_LINE(dept_rec.department_id);
            DBMS_OUTPUT.PUT_LINE(dept_rec.department_name);
            DBMS_OUTPUT.PUT_LINE(dept_rec.manager_id);
            DBMS_OUTPUT.PUT_LINE(dept_rec.location_id);
    end;
    /
--------------------
100
King
ST_CLERK
24000
====================
90
Executive
100
1700

PL/SQL procedure successfully completed.

SQL>
```

레코드에 값 할당

: SELECT 또는 FETCH 문을 사용하여
  레코드에 값 목록을 할당할 수 있습니다.

  이때 열 이름은 레코드의 필드와 동일한 순서로 표시해야 하며
  데이터 유형이 동일하면 레코드를 다른 레코드에 할당할 수도 있습니다.

  사용자 정의 레코드와 %ROWTYPE 레코드의 데이터 유형은 전혀 다릅니다.

<< %ROWTYPE 사용 잇점 >>

- 기본 데이터베이스의 열의 수나 데이터 유형을 몰라도 사용가능.
- 실행중에 기본 데이터베이스 열의 수나 데이터 유형을 변경할 수 있다.
- "SELECT * "문을 이용하여 row를 검색할 때 유용.

<< %ROWTYPE을 사용하여 하나의 레코드를 삽입 >>

1> 실습을 위해 hr.retired_emps 생성(9개의 컬럼으로 구성됨)

```
SQL> CREATE TABLE hr.retired_emps
     (EMPNO    NUMBER(4),   ENAME     VARCHAR2(10),
      JOB      VARCHAR2(9), MGR       NUMBER(4),
      HIREDATE DATE,        LEAVEDATE DATE,
      SAL      NUMBER(7,2), COMM      NUMBER(7,2),
      DEPTNO   NUMBER(2)
     )TABLESPACE example ;

Table Created.

SQL>
```

2> PL/SQL Record를 이용한 데이터 입력 (2가지 방법)

2-1> INSERT 하려는 행의 컬럼값을 모두 개별 지정하는 방법

```
SQL> DEFINE employee_number = 124
SQL>

SQL> DECLARE
        emp_rec hr.employees2%ROWTYPE;
     BEGIN
                SELECT * INTO emp_rec
                FROM  hr.employees2
                WHERE employee_id = &employee_number;

                INSERT INTO hr.retired_emps(empno, ename, job, mgr,
                                  hiredate, leavedate, sal, comm, deptno)
                VALUES (emp_rec.employee_id, emp_rec.last_name,
                        emp_rec.job_id,      emp_rec.manager_id,
                        emp_rec.hire_date,   SYSDATE,
                       emp_rec.salary,      emp_rec.commission_pct,
                        emp_rec.department_id  );
     END;
     /

PL/SQL procedure successfully completed.

SQL>
```

2-2> INSERT 하려는 행을 레코드로 지정함.

```
SQL> DEFINE employee_number = 125

SQL> DECLARE
        emp_rec hr.retired_emps%ROWTYPE;
     BEGIN
        SELECT employee_id, last_name, job_id, manager_id,
               hire_date, sysdate, salary, commission_pct,
               department_id INTO emp_rec
        FROM  hr.employees2
        WHERE employee_id = &employee_number;

        /* 레코드를 그대로 사용하여 입력 */

        INSERT INTO hr.retired_emps VALUES emp_rec;
     END;
     /

PL/SQL procedure successfully completed.
```

```
SQL>
```

3> 입력데이터 확인

```
SQL> SELECT *
     FROM hr.retired_emps;

EMPNO ENAME      JOB       MGR HIREDATE  LEAVEDATE    SAL COMM DEPTNO
----- ---------- -------- ---- --------- --------- ------ ---- ------
  124 Mourgos    ST_MAN    100 16-NOV-99 27-JUN-11   5800           50
  125 Nayer      ST_CLERK  120 16-JUL-97 27-JUN-11   3200           50

SQL>


SQL> commit ;

Commit complete.

SQL>
```

<< Record를 사용하여 테이블에 하나의 행을 갱신 >>

방법 1) ROW 키워드 및 [RECORD명]을 사용하여 행 전체를 갱신

```
SQL> DEFINE employee_number = 124

SQL> DECLARE
        emp_rec hr.retired_emps%ROWTYPE;

     BEGIN
                SELECT * INTO emp_rec
                FROM hr.retired_emps
                where empno = &employee_number;

                /* 레코드의 특정 필드를 갱신     */

                emp_rec.leavedate:=SYSDATE;

                /* 한 행 전체를 나타내는 ROW 키워드 사용하여 */
                /* 한 행 전체를 레코드의 각 필드값으로 갱신  */

                UPDATE hr.retired_emps
                SET ROW = emp_rec
                WHERE empno=&employee_number;
     END;
     /

PL/SQL procedure successfully completed.

SQL>
```

방법 2) [레코드명.필드명]를 이용하여 특정 컬럼만 갱신

```
SQL> DEFINE employee_number = 125
SQL>

SQL> DECLARE
        emp_rec hr.retired_emps%ROWTYPE;
```

```
     BEGIN
                SELECT * INTO emp_rec
                FROM hr.retired_emps
                where  empno = &employee_number;

                emp_rec.leavedate:=SYSDATE;

                /* 특정컬럼만 레코드.필드 형식으로 갱신 */

                UPDATE hr.retired_emps
                SET leavedate = emp_rec.leavedate
                WHERE empno=&employee_number;
     END;
     /

PL/SQL procedure successfully completed.

SQL>


SQL> SELECT *
     FROM hr.retired_emps
     WHERE empno in (124,125) ;

EMPNO ENAME      JOB       MGR HIREDATE  LEAVEDATE    SAL COMM DEPTNO
----- ---------- -------- ---- --------- --------- ------ ---- ------
  124 Mourgos    ST_MAN    100 16-NOV-99 27-JUN-11   5800           50
  125 Nayer      ST_CLERK  120 16-JUL-97 27-JUN-11   3200           50


SQL>
```

==========

<< PL/SQL Collections >>

==========

<< PL/SQL Collections 개요 >>

- [같은 데이터 유형]의 여러 값을 하나의 변수로 처리하고 싶을 때 사용합니다.
- 테이블의 여러 행들로 부터 반환된 특정 컬럼의 여러 값들을 하나의 단위로 처리하고 싶을 때 주로 사용합니다.

  이름 : king <-- 같은 데이터 유형을 사용하는 값들입니다.
  kochar
  홍길동

- 3 가지의 유형이 존재합니다.

  - INDEX BY-TABLES 또는 Associative arrays

    (1) INDEX BY PLS_INTEGER (10gNF)
    INDEX BY BINARY_INTEGER(~9i)

    (2) INDEX BY VARCHAR2

  - NESTED-TABLE

  - VARRAY

  (참고) 본 Lesson에서는

Less06_Composite_Datatype.txt
INDEX BY PLS_INTEGER와 NESTED TABLE 만 학습합니다.

<< INDEX BY Tables or Associative Arrays >>

- 다음 [두 개의 내부 요소]로 구성됩니다

  (1) BINARY_INTEGER나 PLS_INTEGER 데이터 유형의 기본키

  - 기본키는 순차적이진 않습니다.
  - 기본키를 사용하여 배열과 같은 방식으로 row에 액세스 가능.

  (2) 스칼라 Datatype 또는 Record-type의 열 한 개

- 크기(Hold할 수 있는 값의 갯수) 제한이 없으므로
  변수에 Hold할 수 있는 값의 갯수가 동적으로 증가될 수 있습니다.

  하지만, 기본키에 지정되는 Data-type이 Hold할 수 있는
  값들에 따라 크기가 제한됩니다.

  [[기본키로 쓰이는 Data-Type]]

  (1) BINARY_INTEGER (9i 까지)
  (2) PLS_INTEGER (10gNF)

  처리범위: -2,147,483,647 ~ 2,147,483,647 (동일함)

<< INDEX BY PLS_INTEGER Table 변수를 사용하는 방법 >>

(1) INDEX BY PLS_INTEGER Associative Arrays 형식의
    데이터 유형을 생성.

(2) 해당 유형을 사용하는 변수(식별자)를 선언.

단계 1) INDEX BY PLS_INTEGER Table 형식의 데이터 유형을 생성

[기본 문법]

```
TYPE Collection_데이터유형_이름  IS TABLE OF
       { datatype_type [NOT NULL]
       | variable%TYPE [NOT NULL]
       | table.column%TYPE} [NOT NULL]
       | table%ROWTYPE
              [INDEX BY
                     { PLS_INTEGER
                     | BINARY_INTEGER
                     | VARCHAR2(길이) }];
```

(참고) Collection-Type을 생성하는 위의 구문에서

(1) INDEX BY PLS_INTEGER 또는
    INDEX BY BINARY_INTEGER 가 사용되면
    ==> INDEX BY PLS_INTEGER|BINARY_INTEGER Table
        Collection-Type 이 됩니다.

(2) INDEX BY VARCHAR2(길이) 가 사용되면
    ==> INDEX BY VARCHAR2 Table Collection-Type 이
        됩니다.

(3) INDEX BY 절이 생략이 되면
==> NESTED-TABLE Collection-Type 이 됩니다

[[INDEX BY PLS_INTEGER|BINARY_INTEGER TABLE,
INDEX BY VARCHAR2 TABLE,
NESTED-TABLE 타입의 차이점]]

(1)과 (2)의 차이:
인덱스-키를 [정수 또는 문자를 사용하는가]가
다릅니다.

(1)(2)와 (3)의 차이:

(1),(2)는 인덱스-키를 사용자가 정의하지만,

(3)은 인덱스-키를 사용자가 정의 못함.

단계 2) 생성된 INDEX BY PLS_INTEGER Table형식을 이용하는 변수 생성.

[기본 문법]

변수명 선언된_컬렉션_데이터유형_이름 ;

(참고) 교재에서 PL/SQL Collection이라고 하면,
Collection 데이터유형을 사용하는 변수명을 의미합니다.

즉, 아래의 예제에서

ename_table_type은 Colleciton-Datatype을
ename_table 는 ename_table_type의 Collection을

각각 의미합니다.

<< PL/SQL Collections 생성 문법 예제 >>

1> 스칼라 형식을 사용하는
INDEX BY PLS_INTEGER 유형의 PL/SQL Collections 변수 선언

```
TYPE ename_table_type IS TABLE OF
VARCHAR2(30) INDEX BY PLS_INTEGER ;

ename_table ename_table_type;
```

2> 테이블의 특정 컬럼 형식(table.column%TYPE)을 사용하는
INDEX BY PLS_INTEGER 유형의 PL/SQL Collections 변수 선언

```
TYPE ename_table_type IS TABLE OF
hr.employees.last_name%TYPE INDEX BY PLS_INTEGER;

ename_table ename_table_type;
```

3> 테이블의 레코드 형식(table%ROWTYPE)을 사용하는
INDEX BY PLS_INTEGER 유형의 PL/SQL Collections 변수 선언(중요)

```
TYPE emp_table_type IS TABLE OF
hr.employees%ROWTYPE INDEX BY PLS_INTEGER;
```

```
        emp_table emp_table_type ;
```

4> 사용자가 생성한 특정 형식의 레코드를 사용하는
   INDEX BY PLS_INTEGER 유형의 PL/SQL Collections 변수 선언(중요)

(비교: PL/SQL-레코드 변수 선언)

```
        TYPE emp_record_type IS RECORD
            (r_employee_id   NUMBER(6) NOT NULL := 100,
             r_last_name     employees.last_name%TYPE,
             r_job_id        employees.job_id%TYPE,
             r_salary        NUMBER(8,2),
             r_department_id  employees.department_id%TYPE);

        emp_record   emp_record_type ;
```

- 위의 사용자 정의 레코드를 사용하는
  INDEX BY PLS_INTEGER 유형의 PL/SQL Collections 변수

```
        TYPE my_record_type IS TABLE OF
        emp_record_type  INDEX BY PLS_INTEGER;

        my_record_table   my_record_type ;
```

<< INDEX BY PLS_INTEGER Table 생성 및 메소드 사용 실습 >>

```
SQL> SET SERVEROUT ON
SQL>
SQL> DECLARE
        TYPE ename_table_type IS TABLE OF
        hr.employees.last_name%TYPE  -- VARCHAR2(25)
        INDEX BY PLS_INTEGER ;                  -- COLLECTION  유형선언

        TYPE hiredate_table_type IS TABLE OF
        DATE INDEX BY PLS_INTEGER ;             -- COLLECTION  유형선언

        v_ename_table      ename_table_type;    -- COLLECTION  변수선언
        v_hiredate_table   hiredate_table_type; -- COLLECTION  변수선언

     BEGIN

        -- v_ename_table 변수에 인덱스-키 값을 지정해서 값(행)을 할당

        v_ename_table(5) := 'CAMERON';

        v_ename_table(2) := 'JANGBI';

        v_ename_table(10) := 'YUBI';


        -- v_hiredate_table변수에 인덱스-키 값을 지정해서 값(행)을 할당

        v_hiredate_table(8) := SYSDATE + 7;

        IF v_ename_table.EXISTS(&indexno) THEN
                DBMS_OUTPUT.PUT_LINE('==========');
                DBMS_OUTPUT.PUT_LINE('DATA Exists');
                DBMS_OUTPUT.PUT_LINE('==========');
            ELSE
                DBMS_OUTPUT.PUT_LINE('==============');
                DBMS_OUTPUT.PUT_LINE('DATA Not Exists');
                DBMS_OUTPUT.PUT_LINE('==============');
            END IF ;
```

```
    END;
     /
Enter value for indexno: 2 <-- 2 입력 후, [Enter]
==========
DATA Exists
==========

PL/SQL procedure successfully completed.

SQL>

-- SQL*Developer에서 다시 위의 코드를 실행합니다.
-- 값 입력을 요청받으면 1을 입력합니다.

SQL> /
Enter value for indexno: 1 <-- 1 입력 후, [Enter]
==============
DATA Not Exists
==============

PL/SQL procedure successfully completed.

SQL>
```

<< 코드 설명 >>

v_ename_table(5) := 'CAMERON';

==> v_ename_table 변수에 [인덱스-키가 5]인 요소의 값으로 'CAMERON'을 할당.

v_ename_table.EXISTS(5)

==> [인덱스-키가 5]인 요소가 v_ename_table 변수에 존재하면 TRUE를 반환합니다.

<< INDEX BY PLS_INTEGER Table 구조 >>

Unique key	Value
...	...
1	Jones
5	Smith
3	Maduro
...	...
PLS_INTEGER	Scalar

<< INDEX BY PLS_INTEGER Table 사용 상의 숙지사항

- index by table에는 하나의 열과 이 열에 대한 고유 식별자가 있으며 모두 이름 지정이 불가합니다.
- index by table은 선언 시에 초기화될 수 없습니다.

  즉, 테이블을 선언할 때 채워지지 않으며 키나 값을 포함하지 않습니다.

index by table에 값을 채우려면 실행부에서
명시적으로 할당문(:=)을 사용해야 합니다.

- 실제 값이 저장되는 컬럼은
scalar 또는 레코드 등의 datatype을 가질 수 있지만,

기본키가 되는 컬럼은 PLS_INTEGER 와 BINARY_INTEGER, VARCHAR2
중 하나만 지정가능합니다.

<< 선언된 INDEX BY PLS_INTEGER Table 변수의 Methods 사용하기 >>

메소드 사용방법 : 컬렉션변수이름.메소드이름[(인덱스키)]

사용 예 : v_ename_table.exists(1)

메소드이름	설명
EXISTS(n)	PL/SQL 테이블에 n 번 인덱스키를 가진 데이터 요소가 존재하면 TRUE를 반환. v_ename_table.exists(5)
COUNT	현재 PL/SQL 테이블에 있는 요소 수를 반환. v_ename_table.COUNT
FIRST LAST	PL/SQL 테이블의 첫번째(가장 작은)와 마지막(가장 큰) 인덱스 번호를 반환. - PL/SQL 변수가 비어 있으면 NULL을 반환 v_ename_table.FIRST: 가장 작은 인덱스 번호 v_ename_table.LAST : 가장 큰 인덱스 번호
PRIOR(n)	PL/SQL 테이블에서 인덱스 n 앞의 인덱스 번호를 반환. v_ename_table.PRIOR(5): 인덱스 키가 5보다 작은 가장 큰 값
NEXT(n)	PL/SQL 테이블에서 인덱스 n 다음의 인덱스 번호를 반환. v_ename_table.NEXT(5): 인덱스키가 5보다 큰 가장 작은 값
DELETE	PL/SQL 테이블에서 모든 요소를 제거. v_ename_table.DELETE
DELETE(n)	PL/SQL 테이블에서 n 번째 요소를 제거. v_ename_table.DELETE(5): 인덱스키가 5인 값을 제거
DELETE(m, n)	PL/SQL 테이블에서 범위 m...n 사이에 있는 모든요소를 제거 v_ename_table.DELETE(5, 12): 인덱스키가 5와 12 사이 값을 제거

<< INDEX BY Table of Records >>

- "INDEX BY Table of Records" 를 사용하여
  [질의에 의해 검색된 컬럼들로 구성된 레코드들]을 처리할 수 있습니다.
- 허용된 PL/SQL 데이터 유형을 사용하여 Table 변수를 정의합니다.

<< INDEX BY Table of Records 생성 문법 예제 >>

```
DECLARE
-- 1. Table명%ROWTYPE 요소를 갖는 INDEX BY Table 형식 선언
        TYPE dept_table_type IS TABLE OF
        hr.departments%ROWTYPE INDEX BY PLS_INTEGER ;

-- 2. 위의 형식을 사용하는 변수 선언
        dept_table dept_table_type;

-- dept_table 변수의 각 요소(element)는 하나의 record 입니다.
```

<< Table of Records를 호출하는 법 >>

```
[형식]   table(index).field

        -- table : table 유형의 collection변수명
        -- index : 인덱스 번호
        -- field : 레코드의 필드 이름(즉, 테이블 컬럼명)

[예]     dept_table(15).location_id := 1700;

        ==> dept_table 변수의 15번 인덱스 키 요소에 있는
            레코드의 location_id 속성에 1700을 할당합니다.

(용어) 속성: Attributes ==> 필드 : 레코드의 하나의 값
       요소: Elements           : 컬렉션의 값
```

<< INDEX BY Table of Records 예제 >>

```
SQL> SET SERVEROUTPUT ON
SQL>

SQL> DECLARE
        TYPE dept_record_type IS RECORD
            (dept_id     NUMBER(6) NOT NULL := 100,
             dept_name   departments.department_name%TYPE
            );

        TYPE dept_table_type IS TABLE OF
        dept_record_type       /* RECORD TYPE 을 사용 */
        INDEX BY PLS_INTEGER ;

        TYPE emp_table_type IS TABLE OF
```

```
        hr.employees%ROWTYPE INDEX BY BINARY_INTEGER ;


        my_dept_table dept_table_type ;  /* RECORD-COLLECTION */

        my_emp_table emp_table_type;     /* RECORD-COLLECTION */

        max_count    NUMBER(3):= 104;    /* 변수 */

        v_count      number(3);

BEGIN
        SELECT * INTO my_emp_table(102)
        FROM hr.employees
        WHERE employee_id=102 ;

        SELECT * INTO my_emp_table(100)
        FROM hr.employees
        WHERE employee_id=100 ;

        SELECT * INTO my_emp_table(104)
        FROM hr.employees
        WHERE employee_id=104 ;

        SELECT * INTO my_emp_table(101)
        FROM hr.employees
        WHERE employee_id=101 ;

        SELECT * INTO my_emp_table(103)
        FROM hr.employees
        WHERE employee_id=103 ;

        SELECT * INTO my_emp_table(107)
        FROM hr.employees
        WHERE employee_id=107 ;

        SELECT * INTO my_emp_table(105)
        FROM hr.employees
        WHERE employee_id=105 ;

        SELECT * INTO my_emp_table(106)
        FROM hr.employees
        WHERE employee_id=106 ;


        DBMS_OUTPUT.PUT_LINE (
             'FIRST_index: '||my_emp_table.FIRST||' , '||
            'LAST_index : '||my_emp_table.LAST
             );


        DBMS_OUTPUT.PUT_LINE (
             'prior_index: '||my_emp_table.prior(101)||' , '||
            'next_index : '||my_emp_table.next(101)
             );

        DBMS_OUTPUT.PUT_LINE (
             'prior_index: '||my_emp_table.prior(100)||' , '||
            'next_index : '||my_emp_table.next(104)
             );

        my_emp_table.delete(107) ;

        DBMS_OUTPUT.PUT_LINE (
             'FIRST_index: '||my_emp_table.FIRST||' , '||
            'LAST_index : '||my_emp_table.LAST
             );
```

```
        if my_emp_table.exists(90) then
             DBMS_OUTPUT.PUT_LINE ('Exist (90) elements');
           else  DBMS_OUTPUT.PUT_LINE ('NOT Exist (90) elements');
        end if;

        FOR i IN my_emp_table.FIRST..my_emp_table.LAST
        LOOP
             v_count :=my_emp_table.NEXT(i);
             DBMS_OUTPUT.PUT_LINE (i||'  NEXT index :'|| v_count);
             DBMS_OUTPUT.PUT_LINE (my_emp_table(i).employee_id
                              ||','||my_emp_table(i).last_name);

        END LOOP;
     END;
     /
FIRST_index: 100 , LAST_index : 107
prior_index: 100 , next_index : 102
prior_index:  , next_index : 105
FIRST_index: 100 , LAST_index : 106
NOT Exist (90) elements
100  NEXT index :101
100,King
101  NEXT index :102
101,Kochhar
102  NEXT index :103
102,De Haan
103  NEXT index :104
103,Hunold
104  NEXT index :105
104,Ernst
105  NEXT index :106
105,Austin
106  NEXT index :
106,Pataballa

PL/SQL procedure successfully completed.

SQL>
```

<< 중첩 테이블(NESTED-TABLE) >>

- 기능은 INDEX BY tables와 유사
- 기본키가 PLS_INTEGER 가 아니며 음수값이 될 수 없다.
- 첫번째 컬럼이 키로서 참조되지만 기본키라는 의미는 없다.
- 중첩 테이블은 데이터베이스에 저장객체로 생성될 수 있지만,

  INDEX BY tables 은 데이터베이스에 저장객체로 생성될 수 없다.

(참고) 데이터베이스 내장 객체로 생성된 중첩 테이블

중첩 테이블은 값 집합을 보유합니다.

달리 말하면 테이블 내에 테이블이 있는 것입니다.

- 데이터베이스 내장 객체로 생성된 NESTED-TABLE

  중첩 테이블은 범위가 제한되지 않아 동적으로 증가할 수 있습니다.

  중첩 테이블은 PL/SQL과 데이터베이스 모두에서 사용할 수 있습니다.

PL/SQL에서 중첩 테이블은 크기가 동적으로 증가할 수 있는
1차원 배열과 같습니다.

데이터베이스에서 중첩 테이블은 값 집합을 보유하는 열(또는 열들)의
유형입니다.

오라클 데이터베이스는 특별한 순서 없이 중첩 테이블의
행을 저장합니다.

중첩 테이블을 데이터베이스에서 PL/SQL 변수로 읽어올 경우에는
행에 1부터 시작되는 연속적인 하위 스크립트가 제공됩니다.

이로써 개별 행에 대해 배열과 같은 액세스가 가능합니다.
중첩 테이블은 처음에 조밀하지만 삭제를 통해 희소해지면서
비연속적인 하위 스크립트를 가지게 됩니다.

- INDEX BY table이 초기화 되지 않으면, Empty 상태임.
- NESTED TABLE이 초기화 되지 않으면,
  자동적으로 NULL로 초기화 됩니다.

<< PL/SQL 코드에서의 중첩테이블 생성 문법 예제 >>

```
DECLARE
        TYPE location_type IS TABLE OF
        locations.city%TYPE ;

        offices location_type;

BEGIN
        offices := location_type( 'Bombay'
                                , 'Tokyo'
                                , 'Singapore'
                                , 'Oxford'    );
```

<< PL/SQL 코드에서의 중첩테이블 사용 예제 >>

```
SQL> SET SERVEROUTPUT ON
SQL>

SQL> DECLARE
        TYPE location_type IS TABLE OF
        hr.locations.city%TYPE;

        offices location_type ;

        table_count NUMBER ;

     BEGIN

        offices := location_type( 'Bombay'
                                , 'Tokyo'
                                , 'Singapore'
                                , 'Oxford'    );

        table_count := offices.count;

        DBMS_OUTPUT.PUT_LINE('table_count: '||table_count);


        FOR i in 1..table_count
        LOOP
            DBMS_OUTPUT.PUT_LINE(i||'=='||offices(i));
        END LOOP;
```

```
     END;
     /
table_count: 4
1==Bombay
2==Tokyo
3==Singapore
4==Oxford

PL/SQL procedure successfully completed.

SQL>
```

-- 아래의 코드를 분석해보세요.

```
SQL> CREATE TABLE hr.test
     (col1 NUMBER
     ,col2 NUMBER)
     TABLESPACE example ;

Table created.

SQL>

SQL> SET SERVEROUT ON
SQL>

SQL> DECLARE
        TYPE test_record_type IS TABLE OF
        test%rowtype ;

        v_test_record test%rowtype ;
        v_test test_record_type ;
     BEGIN
        v_test_record.col1 := '10';
        v_test_record.col2 := '20';
        v_test := test_record_type(v_test_record) ;
        dbms_output.put_line(v_test(1).col1||'--'||v_test(1).col2);
     END ;
     /
10--20

PL/SQL procedure successfully completed.

SQL>
```

<< VARRAY >>

가변 크기의 배열 또는 VARRAY도 런타임 시
요소 수를 변경할 수 있지만 고정 개수의 요소를
보유하는 동종 요소로 된 컬렉션입니다.

VARRAY는 일련 번호를 하위 스크립트로 사용합니다.
상응하는 SQL 유형을 정의하면 VARRAY를
데이터베이스 테이블에 저장할 수 있습니다.
SQL을 통해 저장하고 검색할 수 있지만
중첩 테이블보다는 유연성이 낮습니다.
배열 연산에 대한 개별 요소를 참조하거나
전체적으로 컬렉션을 조작할 수 있습니다.

VARRAY는 언제나 범위가 제한적이고 희소해지지 않습니다.

해당 유형 정의에서 VARRAY의 최대크기를 지정할 수 있습니다.

인덱스는 고정된 하한값 1을 가지고
확장 가능한 상한값을 가집니다.

VARRAY는 0(비어 있는 경우)부터 해당 유형 정의에서 지정한
최대값까지 다양한 수의 요소를 포함할 수 있습니다.

요소를 참조하기 위해 표준 하위 스크립트 구문을 사용할 수 있습니다.

Variable-size arrays (VARRAY) are similar to PL/SQL tables
 except that a VARRAY is constrained in size.

The maximum size of a VARRAY is
    2 gigabytes (GB) as in nested tables.

The distinction between nested table and VARRAY is
the physical storage mode.

The elements of a VARRAY are stored contiguously
in memory and not in the database.

<< PL/SQL VARRAY type 생성 예제 >>

```
TYPE location_type IS VARRAY(3) OF
locations.city%TYPE;

v_offices location_type ;
```

위의 예에서 3개 초과를 해서 값을 할당(initialize)하면
[Subscript outside of limit] 에러메세지가 표시됩니다.