# Összetett típusok

# Összetett típusok – Rekordtípus

- Rekordtípus deklarációja:

```
TYPE név IS RECORD (
  mezőnév típus [[NOT NULL] {:=|DEFAULT} kifejezés]
[, mezőnév típus [[NOT NULL] {:=|DEFAULT} kifejezés]]…
);
```

- Rekord deklarációja:

```
rekordnév rekordtípusnév;
```

- Például:

```
TYPE t_alk_rec IS RECORD (
  nev      VARCHAR2(46),
  fizetes NUMBER(8,2),
  email    VARCHAR2(25) NOT NULL);
v_fonok t_alk_rec;
```

# Összetett típusok – Kollekciók

- PL/SQL kollekciótípusai:
  - asszociatív tömb
  - beágyazott tábla
  - dinamikus tömb
- 3GL nyelvek tömb fogalmának felelnek meg
  - egydimenziós, indexe minden esetben lehet egész (asszociatív tömb esetén sztring is)
- Létrehozása két lépcsőben történik:
  - kollekciótípus létrehozása
  - kollekcióváltozó deklarálása (*`kollekciónév kollekciótípus_név;`*)
- Elemeinek típusa `REF CURSOR` kivételével tetszőleges PL/SQL típus
- Egy elemére *`kollekciónév(index)`* módon hivatkozhatunk

# Kollekciómetódusok

| *Metódus* | *Visszatérési típus* | *Tevékenység* |
|---|---|---|
| EXISTS | BOOLEAN | Igaz értéket ad, ha az adott indexű elem létezik a kollekcióban |
| COUNT | NUMBER | Visszaadja a kollekció elemeinek számát |
| LIMIT | NUMBER | Visszaadja a kollekció maximális méretét |
| FIRST | *indextípus* | Visszaadja a kollekció első elemének indexét |
| LAST | *indextípus* | Visszaadja a kollekció utolsó elemének indexét |
| NEXT | *indextípus* | Visszaadja egy megadott indexű elemet követő elem indexét |
| PRIOR | *indextípus* | Visszaadja egy megadott indexű elemet megelőző elem indexét |
| EXTEND | nincs | Bővíti a kollekciót |
| TRIM | nincs | Eltávolítja a kollekció utolsó elemeit |
| DELETE | nincs | A megadott elemeket törli a kollekcióból |

# Kollekciók – Asszociatív tömb

```
TYPE név IS TABLE OF elemtípus [NOT NULL]
  INDEX BY indextípus;
```

- Kulcs-érték párok halmaza (hashtábla)
- Csak PL/SQL programokban használható
- Az indextípus `PLS_INTEGER`, `BINARY_INTEGER`, `VARCHAR2(n)`, `STRING(n)` (vagy `LONG`) lehet.
- Az indexeknek nincs (elvi) határa
- Az `i` indexű elemnek történő értékadás létrehozza az adott elemet, ha nem létezett, és felülírja az értékét, ha létezett

# Asszociatív tömb - kollekciómetódusok

- EXISTS(i)
- COUNT
- LIMIT
- FIRST
- LAST
- NEXT(i)
- PRIOR(i)
- DELETE, DELETE(i), DELETE(i,j)

```
DECLARE
TYPE t_kolcsonzesek_at_binint IS TABLE OF
  kolcsonzes%ROWTYPE INDEX BY BINARY_INTEGER;

TYPE t_kolcsonzesek_at_plsint IS TABLE OF
  kolcsonzes%ROWTYPE INDEX BY PLS_INTEGER;

TYPE t_konyv_idk_at_vc2 IS TABLE OF konyv.id%TYPE
     INDEX BY konyv.isbn%TYPE; -- VARCHAR2(30)

TYPE t_vektor IS TABLE OF NUMBER
  INDEX BY BINARY_INTEGER;

TYPE t_matrix IS TABLE OF t_vektor
    INDEX BY BINARY_INTEGER;
TYPE t_at_orszagok IS TABLE OF orszagok%rowtype
     INDEX BY VARCHAR2(100);
```

```
DECLARE
  TYPE t_vektor IS TABLE OF NUMBER
    INDEX BY BINARY_INTEGER;
  v_Vektor   t_vektor;
BEGIN
  FOR i IN -2..2 LOOP
    v_Vektor(i*2) := i;
  END LOOP;

  FOR i IN -5..5 LOOP
    IF v_Vektor.EXISTS(i) THEN
      DBMS_OUTPUT.PUT_LINE(i || ' ' || v_Vektor(i));
    END IF;
  END LOOP;
END;
/
```

```
DECLARE
 TYPE t_tablazat IS TABLE OF NUMBER INDEX BY VARCHAR2(10);
 v_Tablazat   t_tablazat;
 v_Kulcs      VARCHAR2(10);
BEGIN
  FOR i IN 65..67 LOOP
    v_Kulcs := CHR(i);
    v_Tablazat(v_Kulcs) := i;
  END LOOP;
  DBMS_OUTPUT.PUT_LINE('Kulcs  Elem');
  DBMS_OUTPUT.PUT_LINE('-----  ----');
  FOR i IN 0..255 LOOP
   v_Kulcs := CHR(i);
   IF v_Tablazat.EXISTS(v_Kulcs) THEN
      DBMS_OUTPUT.PUT_LINE(v_Kulcs|| v_Tablazat(v_Kulcs));
    END IF;
  END LOOP;
END;
/
```

```
DECLARE
  TYPE t_vektor IS TABLE OF NUMBER
    INDEX BY BINARY_INTEGER;
  v_Vektor      t_vektor;
BEGIN
  FOR i IN -2..2 LOOP
    v_Vektor(i*2) := i;
  END LOOP;
  DBMS_OUTPUT.PUT_LINE('1. count: ' ||
  v_Vektor.COUNT);

  -- Hány tényleges elem esik ebbe az intervallumba?
  v_Vektor.DELETE(-1, 2);
  DBMS_OUTPUT.PUT_LINE('2. count: ' ||
  v_Vektor.COUNT);
END;
/
```

```
DECLARE
  TYPE t_tablazat IS TABLE OF NUMBER
    INDEX BY VARCHAR2(10);

  v_Tablazat   t_tablazat;
BEGIN
  v_Tablazat('a') := 1;
  v_Tablazat('A') := 2;
  v_Tablazat('z') := 3;
  v_Tablazat('Z') := 4;
  DBMS_OUTPUT.PUT_LINE('1. count: ' ||
  v_Tablazat.COUNT);
  v_Tablazat.DELETE('a','z');
  DBMS_OUTPUT.PUT_LINE('2. count: ' ||
  v_Tablazat.COUNT);
END;
/
```

```
DECLARE
  TYPE t_vektor IS TABLE OF NUMBER
    INDEX BY BINARY_INTEGER;
  v_Vektor   t_vektor;
BEGIN
  FOR i IN -2..2 LOOP
    v_Vektor(i*2) := i;
  END LOOP;

  DBMS_OUTPUT.PUT_LINE('first: '
    || NVL(TO_CHAR(v_Vektor.FIRST), 'NULL')
    || ' last: '
    || NVL(TO_CHAR(v_Vektor.LAST), 'NULL'));
END;
/
```

```
DECLARE
 TYPE t_vektor IS TABLE OF NUMBER INDEX BY BINARY_INTEGER;
 v_Vektor   t_vektor;
 i          PLS_INTEGER;
BEGIN
 FOR i IN -2..2 LOOP
   v_Vektor(i*2) := i;
 END LOOP;
 i := v_Vektor.FIRST;
 WHILE i IS NOT NULL LOOP
   DBMS_OUTPUT.PUT_LINE(i||' '||v_Vektor(i));
   i := v_Vektor.NEXT(i);
  END LOOP;

  i := v_Vektor.LAST;
  WHILE i IS NOT NULL LOOP
    DBMS_OUTPUT.PUT_LINE(i||' '|| v_Vektor(i));
    i := v_Vektor.PRIOR(i);
  END LOOP;
END;
/
```

```
DECLARE
 TYPE NickList IS TABLE OF VARCHAR2(64)
     INDEX BY VARCHAR2(32);
 nicknames NickList;
BEGIN
  nicknames('Bob') := 'Robert';
  nicknames('Buffy') := 'Esmerelda';
  nicknames('Chip') := 'Charles';
  nicknames('Dan') := 'Daniel';
  nicknames('Fluffy') := 'Ernestina';
  nicknames('Rob') := 'Robert';
  -- following deletes element denoted by this key
  nicknames.DELETE('Chip');
  -- following deletes elements with keys in this
  --alphabetic range
  nicknames.DELETE('Buffy','Fluffy');
END;
/
```

```
DECLARE
 TYPE sum_multiples IS TABLE OF PLS_INTEGER
   INDEX BY PLS_INTEGER;
 n PLS_INTEGER := 5;
 sn PLS_INTEGER := 10;
 m PLS_INTEGER := 3;
FUNCTION get_sum_multiples (multiple IN PLS_INTEGER,
   num IN PLS_INTEGER) RETURN sum_multiples IS
s sum_multiples;
BEGIN
  FOR i IN 1..num LOOP
    s(i) := multiple * ((i * (i + 1)) / 2);
  END LOOP;
 RETURN s;
END get_sum_multiples;

BEGIN
DBMS_OUTPUT.PUT_LINE ('Sum of the first ' || n || ' multiples of
   ' || m || ' is ' || get_sum_multiples (m, sn)(n));
END;
/
```

# Kollekciók – Dinamikus tömb

```
TYPE név IS {VARRAY | VARYING ARRAY} (max_méret)
  OF elemtípus [NOT NULL];
```

- deklarációjakor meg kell adni a max. méretet
- Az elemek folytonosan helyezkednek el
- Az indexelés `1`-től indul
- `0` és *`max_méret`* közötti elemszámmal rendelkezhet (max. $2^{31}$)
- Speciális objektumtípus
- Az ilyen típusú változó tulajdonképpen egy referencia
  - automatikus kezdőértékadás (NULL)
  - explicit kezdőértékadás (példányosítással)

# Dinamikus tömb - kollekciómetódusok

- EXISTS(i)
- COUNT
- LIMIT
- FIRST
- LAST
- NEXT(i)
- PRIOR (i)
- EXTEND, EXTEND(n), EXTEND(n,m)
- TRIM , TRIM(n)
- DELETE

```
DECLARE
   TYPE varray_type IS VARRAY(5) OF INTEGER;
   v1 varray_type;
   v2 varray_type;
   v3 varray_type:=varray_type();
   v4 varray_type:=varray_type(10,20,30);

BEGIN
  v2 := varray_type(1, 2, 3, 4, 5);
END;
/
```

```
DECLARE
TYPE name_rec IS RECORD
    ( first_name VARCHAR2(20),
      last_name VARCHAR2(25));
TYPE names IS VARRAY(250) OF name_rec;

BEGIN
  NULL;
END;
/
```

```
DECLARE
TYPE t_nt_EmpList IS
   VARRAY(10) OF employees.employee_id%TYPE;
TYPE t_nt_Senior_Salespeople IS
   VARRAY(15) OF employees%ROWTYPE;
CURSOR c2 IS SELECT first_name, last_name
               FROM employees;
TYPE t_nt_NameList IS VARRAY(5) OF c2%ROWTYPE;

BEGIN
    NULL;
END;
/
```

```
DECLARE
TYPE dnames_var IS VARRAY(20) OF VARCHAR2(30);
dept_names dnames_var;
BEGIN
  IF dept_names IS NULL
    THEN DBMS_OUTPUT.PUT_LINE ('Before initialization,
                                the varray is null.');
   -- DBMS_OUTPUT.PUT_LINE
   -- ('It has '||dept_names.COUNT ||' elements.');
     ELSE DBMS_OUTPUT.PUT_LINE('Before initialization,
                               the varray is not null.');
  END IF;
dept_names := dnames_var();
IF dept_names IS NULL
  THEN DBMS_OUTPUT.PUT_LINE ('After initialization,
                              the varray is null.');
  ELSE DBMS_OUTPUT.PUT_LINE ('After initialization,
                              the varray is not null.');
       DBMS_OUTPUT.PUT_LINE ('It has ' ||
                     dept_names.COUNT || ' elements.');
 END IF;
END;
```

```
DECLARE
TYPE T_Szerzok IS VARRAY (10) OF VARCHAR2(50);
v_Szerzok     T_Szerzok := T_Szerzok();
BEGIN
 DBMS_OUTPUT.PUT_LINE('1.count: '||v_Szerzok.COUNT
  ||' Limit: '||NVL(TO_CHAR(v_Szerzok.LIMIT), 'NULL'));
 v_Szerzok.EXTEND; -- Egy NULL elemmel bővítünk
 DBMS_OUTPUT.PUT_LINE('2.count: '||v_Szerzok.COUNT
  ||' v_Szerzok(1): '|| NVL(v_Szerzok(1), 'NULL'));
 v_Szerzok(1) := 'Móra Ferenc';
 DBMS_OUTPUT.PUT_LINE('2. count: ' || v_Szerzok.COUNT
    || ' v_Szerzok(v_Szerzok.COUNT): '
    || NVL(v_Szerzok(v_Szerzok.COUNT), 'NULL'));
 v_Szerzok.EXTEND(3); -- 3 db NULL elemmel bővítünk
 DBMS_OUTPUT.PUT_LINE('2. count: ' || v_Szerzok.COUNT
    || ' v_Szerzok(v_Szerzok.COUNT): '
    || NVL(v_Szerzok(v_Szerzok.COUNT), 'NULL'));
…
```

```
…
v_Szerzok.EXTEND(4, 1);
DBMS_OUTPUT.PUT_LINE('2. count: ' || v_Szerzok.COUNT
    || ' v_Szerzok(v_Szerzok.COUNT): '
    || NVL(v_Szerzok(v_Szerzok.COUNT), 'NULL'));
BEGIN
    v_Szerzok.EXTEND(10);
  EXCEPTION
    WHEN SUBSCRIPT_OUTSIDE_LIMIT THEN
      DBMS_OUTPUT.PUT_LINE('Kivétel! ' || SQLERRM);
  END;

  FOR i IN 1..v_Szerzok.COUNT LOOP
    DBMS_OUTPUT.PUT_LINE(i||' '|| NVL(v_Szerzok(i), 'NULL'));
  END LOOP;
END;
/
```

```
DECLARE
  TYPE dnames_var IS VARRAY(7) OF VARCHAR2(30);
  dept_names dnames_var :=
  dnames_var('Shipping','Sales','Finance','Payroll');
BEGIN
 DBMS_OUTPUT.PUT_LINE ('dept_names has ' ||
                  dept_names.COUNT || ' elements now');
 DBMS_OUTPUT.PUT_LINE ('dept_names''s type can hold a
  maximum of ' || dept_names.LIMIT || ' elements');
 DBMS_OUTPUT.PUT_LINE ('The maximum number you can use
  with ' || 'dept_names.EXTEND() is ' ||
  (dept_names.LIMIT - dept_names.COUNT));
END;
/
```

```
DECLARE
  TYPE T_Szerzok IS VARRAY (10) OF VARCHAR2(50);
  v_Szerzok     T_Szerzok;
BEGIN
 v_Szerzok := T_Szerzok('Móricz Zsigmond','Móra
   Ferenc','Ottlik Géza','Weöres Sándor');
 DBMS_OUTPUT.PUT_LINE('1.count: '||v_Szerzok.COUNT);
 v_Szerzok.TRIM; -- Törlünk egy elemet
 DBMS_OUTPUT.PUT_LINE('2.count: '||v_Szerzok.COUNT);
 v_Szerzok.TRIM(2); -- Törlünk 2 elemet
 DBMS_OUTPUT.PUT_LINE('3.count: '||v_Szerzok.COUNT);
  BEGIN
   v_Szerzok.TRIM(10); -- Megpróbálunk túl sok elemet törölni
  EXCEPTION
    WHEN SUBSCRIPT_BEYOND_COUNT THEN
      DBMS_OUTPUT.PUT_LINE('Kivétel! ' || SQLERRM);
  END;
  DBMS_OUTPUT.PUT_LINE('4. count: ' || v_Szerzok.COUNT);
END;
/
```

```
DECLARE
  TYPE last_name_typ IS VARRAY(3) OF VARCHAR2(64);
  TYPE surname_typ IS VARRAY(3) OF VARCHAR2(64);
   -- These first two variables have the same data type.

group1 last_name_typ:= last_name_typ('Jones','Wong','Marceau');
group2 last_name_typ:= last_name_typ('Klein','Patsos','Singh');

-- This third variable has a similar declaration,
-- but is a different type.
group3 surname_typ:= surname_typ('Trevisi','Macleod','Marquez');

BEGIN
-- Allowed because they have the same data type
group1 := group2;

-- Not allowed because they have different data types
-- group3 := group2; -- fordítási hiba!
END;
/
```

```
DECLARE
 TYPE tb1 IS TABLE OF INTEGER INDEX BY PLS_INTEGER;
   -- Index-by table of index-by tables:
 TYPE ntb1 IS TABLE OF tb1 INDEX BY PLS_INTEGER;
 TYPE va1 IS VARRAY(10) OF VARCHAR2(20);
   -- Index-by table of varray elements:
 TYPE ntb2 IS TABLE OF va1 INDEX BY PLS_INTEGER;
 v1 va1 := va1('hello', 'world');
 v2 ntb1; v3 ntb2;
 v4 tb1; v5 tb1; -- empty table
BEGIN
 v4(1) := 34; v4(2) := 46456;
 v4(456) := 343; v2(23) := v4;
 v3(34) := va1(33, 456, 656, 343);
  -- assign an empty table to v2(35) and try again
 v2(35) := v5; v2(35)(2) := 78; -- it works now
END;
/
```

```
DECLARE
 TYPE t1 IS VARRAY(10) OF INTEGER;
 TYPE nt1 IS VARRAY(10) OF t1; -- multilevel varray type
 va t1 := t1(2,3,5); -- initialize multilevel varray
 nva nt1 := nt1(va, t1(55,6,73), t1(2,4), va);
 i INTEGER;
 va1 t1;
BEGIN -- multilevel access
 i := nva(2)(3); -- i will get value 73
DBMS_OUTPUT.PUT_LINE('I = ' || i); -- add a varray element to nva
 nva.EXTEND; -- replace inner varray elements
 nva(5) := t1(56, 32);
 nva(4) := t1(45,43,67,43345);
   -- replace an inner integer element
 nva(4)(4) := 1; -- replaces 43345 with 1
                 -- add an element to the 4th varray element
                 -- and store integer 89 into it.
 nva(4).EXTEND;
 nva(4)(5) := 89;
END;
/
```

# Kollekciók – Beágyazott tábla – 1

`TYPE` *`név`* `IS TABLE OF` *`elemtípus`* `[NOT NULL];`

- Speciális objektumtípus
  - az ilyen típusú változó tulajdonképpen egy referencia
- Max. $2^{31}$ eleme lehet (1-től indexelünk)
- Az elemek szétszórtan helyezkednek el (lyukak!)

```
DECLARE
TYPE t_dt_EmpList IS
   TABLE OF employees.employee_id%TYPE;
TYPE t_dt_Senior_Salespeople IS
   TABLE OF employees%ROWTYPE;
CURSOR c2 IS SELECT first_name, last_name
               FROM employees;
TYPE t_nt_NameList IS TABLE OF c2%ROWTYPE;


BEGIN
    NULL;
END;
/
```

```
DECLARE
  TYPE nested_type IS TABLE OF VARCHAR2(30);
  v1 nested_type:=
  nested_type('Shipping','Sales','Finance','
  Payroll');
  v2 nested_type:=nested_type();
  v3 nested_type;
BEGIN
null;

END;
/
```

```
DECLARE
  TYPE t_nevek IS TABLE OF VARCHAR2(10);
  v_Nevek t_nevek;
  v_Nevek2 t_nevek:=t_nevek();
PROCEDURE URES_E(p_Nevek t_nevek) IS
 BEGIN
 IF p_Nevek IS NULL
   THEN DBMS_OUTPUT.PUT_LINE('NULL értékű kollekció');
   ELSIF p_Nevek.COUNT=0
         THEN DBMS_OUTPUT.PUT_LINE('Üres kollekció');
 END IF;
END;
BEGIN
  URES_E(v_Nevek);
  URES_E(v_Nevek2);
END;
/
```

# Beágyazott tábla - kollekciómetódusok

- EXISTS(i)
- COUNT
- LIMIT
- FIRST
- LAST
- NEXT(i)
- PRIOR (i)
- EXTEND, EXTEND(n), EXTEND(n,m)
- TRIM , TRIM(n)
- DELETE, DELETE(i), DELETE(i,j)

```
DECLARE
  TYPE NumList IS TABLE OF INTEGER;
  n NumList := NumList(1,3,5,7);
BEGIN
  n.DELETE(2); -- Delete the second element
  IF n.EXISTS(1) THEN
    DBMS_OUTPUT.PUT_LINE('OK, element #1 exists.');
  END IF;
  IF n.EXISTS(2) = FALSE THEN
     DBMS_OUTPUT.PUT_LINE('OK, element #2 was deleted.');
  END IF;
  IF n.EXISTS(99) = FALSE THEN
     DBMS_OUTPUT.PUT_LINE('OK, element #99
                          does not exist at all.');
  END IF;
END;
/
```

```
DECLARE
  TYPE NumList IS TABLE OF NUMBER;
  n NumList := NumList(1,3,5,7);
  counter INTEGER;
BEGIN
DBMS_OUTPUT.PUT_LINE('N''s first subscript is '||n.FIRST);
DBMS_OUTPUT.PUT_LINE('N''s last subscript is ' || n.LAST);
FOR i IN n.FIRST .. n.LAST LOOP
  DBMS_OUTPUT.PUT_LINE('Element #' || i || ' = ' || n(i));
 END LOOP;
n.DELETE(2); -- Delete second element.
IF n IS NOT NULL THEN
 counter := n.FIRST;
 WHILE counter IS NOT NULL LOOP
  DBMS_OUTPUT.PUT_LINE('Element #'||counter||' = '||n(counter));
  counter := n.NEXT(counter);
  END LOOP;
ELSE DBMS_OUTPUT.PUT_LINE('N is null, nothing to do.');
END IF;
END;
/
```

```
DECLARE
  TYPE NumList IS TABLE OF NUMBER;
  n NumList := NumList(1966,1971,1984,1989,1999);
BEGIN
 DBMS_OUTPUT.PUT_LINE('The element after #2 is #' ||
  n.NEXT(2));
 DBMS_OUTPUT.PUT_LINE('The element before #2 is #' ||
  n.PRIOR(2));
 n.DELETE(3);
 -- Delete an element to show how NEXT can handle gaps.
 DBMS_OUTPUT.PUT_LINE('Now the element after #2 is #' ||
  n.NEXT(2));
 IF n.PRIOR(n.FIRST) IS NULL THEN
   DBMS_OUTPUT.PUT_LINE ('Can''t get PRIOR of the first
  element.');
 END IF;
END;
/
```

```
DECLARE
TYPE NumList IS TABLE OF NUMBER;
n NumList := NumList(1,3,5,7);
counter INTEGER;
BEGIN
 n.DELETE(2);
 counter := n.FIRST;
 WHILE counter IS NOT NULL LOOP
   DBMS_OUTPUT.PUT_LINE ('Counting up: Element #'
                || counter || ' = ' || n(counter));
   counter := n.NEXT(counter);
 END LOOP;

 counter := n.LAST;
 WHILE counter IS NOT NULL LOOP
    DBMS_OUTPUT.PUT_LINE ('Counting down: Element #' ||
                          counter || ' = ' || n(counter));
    counter := n.PRIOR(counter);
 END LOOP;
END;
/
```

```
DECLARE
  TYPE NumList IS TABLE OF NUMBER;
  n NumList := NumList(2,4,6,8);
   -- Collection starts with 4 elements.
  BEGIN
   DBMS_OUTPUT.PUT_LINE ('There are '
              || n.COUNT || ' elements in N.');
   n.EXTEND(3); -- Add 3 elements at the end.
   DBMS_OUTPUT.PUT_LINE ('Now there are '
                        || n.COUNT || ' elements in N.');
   n := NumList(86,99); -- Assign a value with 2 elements.
   DBMS_OUTPUT.PUT_LINE ('Now there are ' ||
                          n.COUNT || ' elements in N.');
   n.TRIM(2); -- Remove the last 2 elements, leaving none.
   DBMS_OUTPUT.PUT_LINE ('Now there are ' ||
                          n.COUNT || ' elements in N.');
 END;
/
```

```
declare
type t is table of number(5);
v t:=t(1,2,3,4);
begin
v.delete(2);
v.extend(3,2);
end;

--no_data_found
```

```
CREATE PACKAGE personnel AS
  TYPE staff_list IS TABLE OF employees.employee_id%TYPE;
  PROCEDURE award_bonuses (empleos_buenos IN staff_list);
END personnel;
/

CREATE PACKAGE BODY personnel AS
 PROCEDURE award_bonuses (empleos_buenos staff_list) IS
  BEGIN
   FOR i IN empleos_buenos.FIRST..empleos_buenos.LAST
    LOOP
     UPDATE employees
     SET salary = salary + 100
     WHERE employees.employee_id = empleos_buenos(i);
    END LOOP;
   END;
 END;
 /
```

```
DECLARE
  good_employees personnel.staff_list;
BEGIN
good_employees:=
  personnel.staff_list(100,103,107);

 personnel.award_bonuses(good_employees);
END;
/
```

```
DECLARE
  TYPE Roster IS
   TABLE OF VARCHAR2(15);
names Roster :=
   Roster('D Caruso','J Hamil','D Piro','R Singh');

PROCEDURE verify_name(the_name VARCHAR2) IS
  BEGIN
   DBMS_OUTPUT.PUT_LINE(the_name);
  END;
BEGIN
  FOR i IN names.FIRST .. names.LAST LOOP
   IF names(i) = 'J Hamil' THEN
     DBMS_OUTPUT.PUT_LINE(names(i));
   END IF;
   END LOOP;
verify_name(names(3));
END; /
```

```
DECLARE
  TYPE NumList IS TABLE OF NUMBER;
  n NumList := NumList(1,2,3,5,7,11);
PROCEDURE print_numlist(the_list NumList) IS
  output VARCHAR2(128);
BEGIN
  IF n.COUNT = 0 THEN
    DBMS_OUTPUT.PUT_LINE('No elements in collection.');
   ELSE FOR i IN the_list.FIRST .. the_list.LAST LOOP
            output := output ||
               NVL(TO_CHAR(the_list(i)),'NULL') || ' ';
        END LOOP;
        DBMS_OUTPUT.PUT_LINE(output);
  END IF;
END;
…
```

```
…
BEGIN
  print_numlist(n);
  n.TRIM(2);  print_numlist(n);
  n.TRIM; print_numlist(n);
  n.TRIM(n.COUNT); -- Remove all remaining elements.
  print_numlist(n);
BEGIN
  n := NumList(1,2,3);
  n.TRIM(100);
  EXCEPTION WHEN SUBSCRIPT_BEYOND_COUNT
    THEN DBMS_OUTPUT.PUT_LINE ('There weren''t 100 elements
   to be trimmed.');
END;
n := NumList(1,2,3,4);
n.DELETE(3);
n.TRIM(2); print_numlist(n);
END;
/
```

```
DECLARE
 TYPE CourseList IS TABLE OF VARCHAR2(10);
 courses CourseList;
BEGIN
 courses := CourseList('Biol 4412', 'Psyc 3112',
            'Anth 3001');
 courses.DELETE(courses.LAST);
 courses.TRIM(courses.COUNT);
 DBMS_OUTPUT.PUT_LINE(courses(1));
END;
/

Result:
Biol 4412
```

```
DECLARE
 TYPE NumList IS TABLE OF NUMBER;
 n NumList := NumList(10,20,30,40,50,60,70,80,90,100);
BEGIN
  n.DELETE(2); -- deletes element 2
  n.DELETE(3,6); -- deletes elements 3 through 6
  n.DELETE(7,7); -- deletes element 7
  n.DELETE(6,3); -- does nothing since 6 > 3
  n.DELETE; -- deletes all elements
END;
/
```

```
DECLARE
  TYPE t_nevek IS TABLE OF VARCHAR2(10);
  v_Nevek t_nevek:=t_nevek('A1`,'B2`,'C3`,'D4`,'E5',
  'F6','G7','H8','I9','J10');
  i  PLS_INTEGER;
BEGIN
 DBMS_OUTPUT.PUT_LINE('1. count: ' || v_Nevek.COUNT);
 v_Nevek.DELETE(3); -- Törlünk pár elemet
 v_Nevek.DELETE(6, 8);
 v_Nevek.DELETE(10, 12);
 v_Nevek.DELETE(60);
 DBMS_OUTPUT.PUT_LINE('2. count: ' || v_Nevek.COUNT);
 i := v_Nevek.FIRST;
 WHILE i IS NOT NULL LOOP
   DBMS_OUTPUT.PUT_LINE(i||'  '||v_Nevek(i));
   i := v_Nevek.NEXT(i);
  END LOOP;
END;
/
```

```
DECLARE
 TYPE tb1 IS TABLE OF VARCHAR2(20);
 TYPE Ntb1 IS TABLE OF tb1; -- table of table elements
 TYPE Tv1 IS VARRAY(10) OF INTEGER;
 TYPE ntb2 IS TABLE OF tv1; -- table of varray elements
 vtb1 tb1 := tb1('one', 'three');
 vntb1 ntb1 := ntb1(vtb1);
 vntb2 ntb2 := ntb2(tv1(3,5), tv1(5,7,3));
   -- table of varray elements
BEGIN
 vntb1.EXTEND;
 vntb1(2) := vntb1(1);
   -- delete the first element in vntb1
 vntb1.DELETE(1); -- delete the first string
             -- from the second table in the nested table
 vntb1(2).DELETE(1);
END;
/
```

- Beágyazott tábla és dinamikus tömb NULL értéke tesztelhető.
- Beágyazott táblák egyenlősége is vizsgálható akkor, ha azonos típusúak és az elemek is összehasonlíthatók egyenlőség szerint.
- A rekordot tartalmazó beágyazott tábla és bármilyen elemű dinamikus tömb vagy asszociatív tömb egyenlőségvizsgálata fordítási hibát eredményezne.

```
DECLARE
TYPE dnames_tab IS TABLE OF VARCHAR2(30);
dept_names dnames_tab:=dnames_tab('Shipping','Sales',
   'Finance','Payroll');  -- Initialized to non-null value
empty_set dnames_tab; -- Not initialized, therefore null
PROCEDURE print_dept_names_status IS
 BEGIN
 IF dept_names IS NULL
   THEN DBMS_OUTPUT.PUT_LINE('dept_names is null.');
   ELSE DBMS_OUTPUT.PUT_LINE('dept_names is not null.');
 END IF;
END print_dept_names_status;
BEGIN
 print_dept_names_status;
 dept_names := empty_set;
  -- Assign null collection to dept_names.
 print_dept_names_status;
dept_names := dnames_tab('Shipping','Sales','Finance','Payroll');
   -- Re-initialize dept_names print_dept_names_status;
END;
/
```

```
DECLARE
 TYPE Foursome IS VARRAY(4) OF VARCHAR2(15);
 team Foursome;
BEGIN
 IF team IS NULL
   THEN DBMS_OUTPUT.PUT_LINE('team IS NULL');
   ELSE DBMS_OUTPUT.PUT_LINE('team IS NOT NULL');
 END IF;
END;
/
Result:
team IS NULL
```

```
DECLARE
 TYPE dnames_tab IS TABLE OF VARCHAR2(30);
 dept_names1 dnames_tab:= dnames_tab('Shipping','Sales',
   'Finance','Payroll');
 dept_names2 dnames_tab := dnames_tab('Sales','Finance',
   'Shipping','Payroll');
 dept_names3 dnames_tab := dnames_tab('Sales',
     'Finance','Payroll');
BEGIN
 IF dept_names1 = dept_names2
    THEN DBMS_OUTPUT.PUT_LINE('dept_names1 = dept_names2');
 END IF;
 IF dept_names2 != dept_names3
   THEN DBMS_OUTPUT.PUT_LINE('dept_names2 != dept_names3');
 END IF;
END;
/
```

# Kollekciók – Kivételek

- `COLLECTION_IS_NULL`: NULL értékű kollekcióra metódus meghívása (az `EXISTS` kivételével)
- `SUBSCRIPT_BEYOND_COUNT`: az elemszámnál nagyobb indexű elemre hivatkozáskor (dinamikus tömb vagy beágyazott tábla esetén)
- `SUBSCRIPT_OUTSIDE_LIMIT`: érvényes tartományon kívüli indexhivatkozás esetén (pl. `-1`) (dinamikus tömb vagy beágyazott tábla esetén)
- `NO_DATA_FOUND`: nem létező elemre történő hivatkozáskor
- `VALUE_ERROR`: ha az index NULL, vagy nem konvertálható a kulcs típusára

```
DECLARE
 TYPE WordList IS TABLE OF VARCHAR2(5);
 words WordList;
 err_msg VARCHAR2(100);
PROCEDURE display_error IS
 BEGIN
 err_msg := SUBSTR(SQLERRM, 1, 100);
 DBMS_OUTPUT.PUT_LINE('Error message = ' || err_msg);
 END;
BEGIN
 BEGIN
 words(1) := 10;
 EXCEPTION WHEN OTHERS THEN display_error;
 END;
words := WordList('1st', '2nd', '3rd');
words(3) := words(1) || '+2';
BEGIN
 words(3) := 'longer than 5 characters';
 EXCEPTION WHEN OTHERS THEN display_error;
END;
…
```

```
…
BEGIN
 words('B') := 'dunno';
 EXCEPTION WHEN OTHERS THEN display_error;
END;
BEGIN
 words(0) := 'zero';
 EXCEPTION WHEN OTHERS THEN display_error;
END;
BEGIN
 words(4) := 'maybe';
 EXCEPTION WHEN OTHERS THEN display_error;
END;
BEGIN
 words.DELETE(1);
 IF words(1) = 'First' THEN NULL;
 END IF;
 EXCEPTION WHEN OTHERS THEN display_error;
END;
END;
```

```
DECLARE
  TYPE NumList IS TABLE OF NUMBER;
  nums NumList := NumList(10,20,30);
   -- initialize table
BEGIN
  nums.DELETE(-1);
    -- does not raise SUBSCRIPT_OUTSIDE_LIMIT
  nums.DELETE(3); -- delete 3rd element
  DBMS_OUTPUT.PUT_LINE(nums.COUNT); -- prints 2
  nums(3) := 30;
   -- allowed; does not raise NO_DATA_FOUND
  DBMS_OUTPUT.PUT_LINE(nums.COUNT); -- prints 3
END;
/
Result:
2
3
```

```
declare
type t_dt is varray(5) of number(3); v_dt t_dt;
begin
 begin
  dbms_output.put_line(v_dt(1));
  exception when others
     then dbms_output.put_line(sqlcode||' '||sqlerrm);
     --collection_is_null
 end;

 begin
 v_dt:=t_dt(1,5,7);  v_dt.extend(3,2);
  exception when others
     then dbms_output.put_line(sqlcode||' '||sqlerrm);
     --subscript_outside_limit
 end;
 …
```

…

```
 begin
 v_dt:=t_dt(1,5,7); dbms_output.put_line(v_dt(4));
  exception when others then
  dbms_output.put_line(sqlcode||' '||sqlerrm);
     --subscript_beyond_count
 end;
 begin
 v_dt:=t_dt(1,5,7);
  dbms_output.put_line(v_dt(-1));
  exception when others
     then dbms_output.put_line(sqlcode||' '||sqlerrm);
     --subscript_outside_limit
 end;
 begin
 v_dt:=t_dt(1,5,7);
  dbms_output.put_line(v_dt('a'));
  exception when others
     then dbms_output.put_line(sqlcode||' '||sqlerrm);
     --value_error
 end; end; /
```

```
declare
type t_at is table of number(3) index by pls_integer;
v_at t_at;
begin
 begin
  dbms_output.put_line(v_at(1));
  exception when others
     then dbms_output.put_line(sqlcode||' '||sqlerrm);
     --no_data_found
 end;

 begin
 v_at(5):=1; v_at(4):=2; v_at('a'):=3;
  exception when others
     then dbms_output.put_line(sqlcode||' '||sqlerrm);
     --value_error
 end;
end;
```