

## Limbajul PL/SQL- Colecții și înregistrări

- PRIOR(n) – indicele ce precede indicele n.
- NEXT(n) - indicele ce succede indicele n.

Dacă elementul n nu are predecesor, PRIOR(n) este NULL.

Dacă elementul n nu are succesori, NEXT(n) este NULL.

Ex. Traversarea unei tabele nested:

```
i:=cursuri.FIRST;
```

```
WHILE i IS NOT NULL LOOP
```

```
....prelucrează curs(i)..
```

```
i:=cursuri.NEXT(i);
```

```
END LOOP;
```

La traversare PRIOR și NEXT ignoră elementele șterse 91

# Limbajul PL/SQL- Colecții și înregistrări

- EXTEND – mărește dimensiunea colecției.
  - EXTEND – adaugă un element null.
  - EXTEND(n) - adaugă n elemente null.
  - EXTEND(n,i) - adaugă n copii ale elementului i.

Nu se poate folosi EXTEND pentru a initializa o colecție nulă.

Dacă impunem NOT NULL pentru tipul TABLE sau VARRAY, atunci nu trebuie folosit EXTEND de primele 2 tipuri.

DECLARE

```
TYPE lista_cursuri1 IS TABLE OF VARCHAR2(10);  
cursuri1 lista_cursuri1;
```

# Limbajul PL/SQL- Colecții și înregistrări

BEGIN

```
cursuri1:=lista_cursuri1('Algoritmi','Baze de date  
I','Limbaje formale','Tehnici de compilare');
```

```
cursuri1.DELETE(3);
```

```
cursuri1.EXTEND; - adaugă un element null.
```

```
cursuri1(5):='Programare obiect';
```

```
END;
```

Dacă se utilizează `cursuri1(i)`, unde  $i > 5$ , atunci se inițiază excepția `SUBSCRIPT_BEYOND_COUNT`

Elementele șterse sunt socotite în calculul dimensiunii.

`COUNT` pentru tabele nested returnează nr. de elemente neșterse

# Limbajul PL/SQL- Colecții și înregistrări

- TRIM are două forme:
  - TRIM – îndepărtează un element de la sfârșitul colecției
  - TRIM(n) – îndepărtează n elemente

Dacă  $n > \text{COUNT}$  atunci se lansează excepția  
SUBSCRIPT\_BEYOND\_COUNT

TRIM consideră elementele șterse.

- DELETE are trei forme:
  - DELETE – șterge toate elementele unei colecții,
  - DELETE(n) – șterge al n-lea element din tabelă nested,
  - DELETE(m,n) – șterge toate elementele cu indici m..n

Dacă  $m > n$  sau m is null sau n is null, atunci DELETE nu șterge nimic.

# Limbajul PL/SQL- Colecții și înregistrări

- Varrays sunt dense, deci nu folosim proc. DELETE
- PL/SQL ține o evidență a elementelor șterse.
- O tabelă nested sau un varray poate fi parametru formal al unei proceduri sau funcții.

Evitarea excepțiilor.

- Referirea la un element inexistent produce o excepție predefinită.

Ex. DECLARE

```
TYPE numlist IS TABLE OF NUMBER;
```

```
nums numlist; -- colecție atomic nulă.
```

```
BEGIN
```

```
nums(1):=10; -- excepția: COLLECTION_IS_NULL
```

```
nums:=numlist(15,20); --inițializarea tablei
```

# Limbajul PL/SQL- Colecții și înregistrări

```
nums(NULL):=5; -- VALUE_ERROR
```

```
nums(0):= 15; --SUBSCRIPT_OUTSIDE_LIMIT
```

```
nums(3):=40; --SUBSCRIPT_BEYOND_COUNT
```

```
nums.DELETE(1); - șterge elementul 1
```

```
IF nums(1)= 15 THEN ... --NO_DATA_FOUND
```

## Excepție

## Apare când..

COLLECTION\_is\_NULL      colecția este atomic nulă

NO\_DATA\_FOUND          indicele specifică un element șters

SUBSCRIPT\_BEYOND\_FOUND indicele depășește  
numărul de elemente din colecție.

SUBSCRIPT\_OUTSIDE\_LIMIT indice în afara  
domeniului legal.

VALUE\_ERROR indice null sau neconvertibil la un intreg 96

## Limbajul PL/SQL- Inregistrări

- Definierea:

```
TYPE tip_inreg IS RECORD (cimp1,cimp2,...);
```

cimpi are forma generală:

```
numecimp tip [[NOT NULL] {:=|DEFAULT} expresie]
```

Tipul inregistrare nu poate fi creat si memorat in baza de date. Putem folosi %TYPE si %ROWTYPE pentru a specifica tipul cimpurilor.

Tipul obiect nu poate avea ca attribute tipul RECORD.

Inregistrarile pot contine obiecte, colectii si alte inregistrari, numite inregistrari nested.

# Limbajul PL/SQL- Inregistrări

DECLARE

```
TYPE TimeRec IS RECORD (  
    seconds SMALLINT,  
    minutes SMALLINT,  
    hours    SMALLINT);
```

```
TYPE FlightRec IS RECORD (  
    flight_no INTEGER,  
    plane_id  VARCHAR2(10),  
    captain   Employee, --obiect  
    passengers PassengerList, --varray  
    depart_time TimeRec,  
    airport_code VARCHAR2(10));
```

# Limbaajul PL/SQL- Inregistrări

- Declararea:

```
var_inregistrare tip_inregistrare;
```

- var\_inregistrare poate fi parametru formal pentru proceduri sau funcții.

- Inițializarea și referirea înregistrărilor:

```
- numecimp tip [[NOT NULL] {:=|DEFAULT} expresie]
```

- Referirea:

```
var_inregistrare.numecimp
```

Dacă o funcție returnează o inregistrare, atunci referirea unui cimp se face:

```
nume_functie(lista_parametri).numecimp
```

# Limbajul PL/SQL- Inregistrări

```
DECLARE
```

```
TYPE EmpRec IS RECORD (
```

```
    emp_id  NUMBER(4),
```

```
    job_title CHAR(14),
```

```
    salary   REAL(7,2));
```

```
middle_sal REAL;
```

```
FUNCTION F1(n INTEGER) RETURN EmpRec Is
```

```
    emp_info EmpRec;
```

```
BEGIN
```

```
    ...
```

```
    RETURN emp_info;
```

```
END;
```

# Limbajul PL/SQL- Inregistrări

```
BEGIN
```

```
    middle_sal:=F1(10).salary;
```

```
END;
```

Folosim notația “.” pentru referirea la cimpuri nested și la obiecte memorate într-un cimp.

- Asignarea înregistrărilor:

```
var_inregistrare_numecimp:=expresie;
```

```
ex. emp_info.name :=UPPER(emp_info.name);
```

```
var_inreg1:=var_inreg2; -- același tip;
```

# Limbajul PL/SQL- Inregistrări

DECLARE

```
TYPE DeptRec IS RECORD (  
    dept_num NUMBER(2),  
    dept_name CHAR(14),  
    location CHAR(13));
```

```
dept1_info DeptRec;
```

```
dept2_info dept%ROWTYPE;
```

BEGIN

```
SELECT * INTO dept2_info FROM dept WHERE  
deptno=10;
```

```
dept1_info:=dept2_info; sau
```

```
SELECT * INTO dept1_info FROM dept WHERE  
deptno=10;
```

## Limbajul PL/SQL- Inregistrări

- Nu putem folosi var\_inregistrare în INSERT.
- Nu putem folosi: var\_inregistrare:=(v1,v2,...);
- Nu putem compara înregistrările, nici cu IS NULL sau IS NOT NULL.
- Prelucrarea înregistrărilor:

```
SQL> CREATE TYPE Passenger AS OBJECT (  
    flight_no    NUMBER(3),  
    name        VARCHAR2(20),  
    seat        CHAR(5));
```

```
SQL> CREATE TYPE PassengerList AS VARRAY(300)  
    OF Passenger;
```

```
SQL> CREATE TABLE flights (  

```

## Limbajul PL/SQL- Inregistrări

```
flight_no  NUMBER(3),  
gate       CHAR(5),  
departure  CHAR(15),  
arrival    CHAR(15),  
passengers PassengerList);
```

```
BEGIN
```

```
INSERT INTO flights
```

```
VALUES(100,'L25','BUC 6:50PM','IASI 7.50PM',  
       PassengerList(Passenger(100,'Popescu','3A'),  
                      Passenger(100,'Ionescu','4A'),  
                      Passenger(100,'Eftimie','5A')));
```

Ex.Program pentru citirea tabelii flights folosind o var de tip inregistrare.