

## **Tipuri de date scalare în PL/SQL. Declararea variabilelor.**

### **Instrucțiuni PL/SQL**

### **Blocuri**

*PL/SQL este extensia procedurală a limbajului SQL, cu trăsături specifice limbajelor de programare.*

## **I. Tipuri de date scalare**

Nu au componente interne (conțin valori atomice). Se împart în 5 clase.

- Tipurile de date ce stochează **valori numerice** cuprind
  - tipul *NUMBER* cu subtipurile *DEC*, *DECIMAL*, *DOUBLE PRECISION*, *FLOAT*, *INTEGER*, *INT*, *NUMERIC*, *REAL*, *SMALLINT*;
  - tipul *BINARY\_INTEGER* cu subtipurile *NATURAL*, *NATURALN*, *POSITIVE*, *POSITIVEN*, *SIGNTYPE*; tipul *PLS\_INTEGER*.
- Tipurile de date ce stochează **caractere** cuprind
  - tipul *VARCHAR2* cu subtipurile *STRING*, *VARCHAR*;
  - tipul de date *CHAR* cu subtipul *CHARACTER*;
  - tipurile *LONG*, *RAW*, *LONG RAW*, *ROWID*.
- Tipurile de date ce stochează **data calendaristică și ora** cuprind tipurile *DATE*, *TIMESTAMP*, *TIMESTAMP WITH TIME ZONE*, *TIMESTAMP WITH LOCAL TIME ZONE*, *INTERVAL YEAR TO MONTH*, *INTERVAL DAY TO SECOND*.
- Tipurile de date **globalizare** ce stochează date *unicode* includ tipurile *NCHAR* și *NVARCHAR2*.
- Tipul de date *BOOLEAN* stochează **valori logice** (*true*, *false* sau *null*).

**Obs :** Mai multe informații despre tipurile de date PL/SQL la

[http://www.stanford.edu/dept/itss/docs/oracle/9i/appdev.920/a96624/03\\_types.htm](http://www.stanford.edu/dept/itss/docs/oracle/9i/appdev.920/a96624/03_types.htm)

## **II. Variabile de legătură PL/SQL**

- **O variabilă de legătură** (*bind variable*) este variabila care se declară într-un mediu gazdă și este folosită pentru transferul la execuție al valorilor numerice sau de tip caracter în/din unul sau mai multe programe *PL/SQL*.
- Variabilele declarate în mediul gazdă sau în cel apelant pot fi referite în instrucțiuni *PL/SQL* dacă acestea nu sunt în cadrul unei proceduri, funcții sau pachet.
- În *SQL\*Plus*, variabilele de legătură se declară folosind comanda **VARIABLE**, iar pentru tipărirea acestora se utilizează comanda **PRINT**. Ele sunt referite prin prefixare cu simbolul ":", pentru a putea fi deosebite de variabilele *PL/SQL* declarate.

## **III. Declararea variabilelor PL/SQL**

- Identificatorii *PL/SQL* trebuie declarați înainte să fie referiți în blocul *PL/SQL*. Dacă în declarația unei variabile apar referiri la alte variabile, acestea trebuie să fi fost declarate anterior. Orice variabilă declarată într-un bloc este accesibilă blocurilor conținute sintactic în acesta.
- În declararea variabilelor în *PL/SQL* pot fi utilizate atributele **%TYPE** și **%ROWTYPE**, care

reprezintă tipuri de date implicite.

- Atributul *%TYPE* permite definirea unei variabile având tipul unei variabile declarate anterior sau tipul unei coloane dintr-un tabel.
- Atributul *%ROWTYPE* permite definirea unei variabile având tipul unei înregistrări dintr-un tabel.

**Sintaxa** declarării unei variabile este următoarea:

```
identificator [CONSTANT]{tip_de_date | identificator%TYPE |
  identificator%ROWTYPE} [NOT NULL]
  [{:= | DEFAULT} expresie_PL/SQL];
```

**Exemplu:**

```
v_valoare      NUMBER(15) NOT NULL := 0;
v_data_achizitie DATE DEFAULT SYSDATE;
v_material     VARCHAR2(15) := 'Matase';
c_valoare      CONSTANT NUMBER := 100000;
v_stare        VARCHAR2(20) DEFAULT 'Buna';
v_clasificare  BOOLEAN DEFAULT FALSE;
v_cod_opera    opera.cod_opera%TYPE;
v_opera        opera%ROWTYPE;
int_an_luna    INTERVAL YEAR TO MONTH :=
  INTERVAL '3-2' YEAR TO MONTH; --interval de 3 ani si 2 luni
```

**Observații:**

- Pentru ușurința referirii se convine prefixarea numelor de variabile astfel:
  - prefixarea cu litera *v* (*v\_valoare*) pentru variabilele PL/SQL
  - prefixarea cu litera *c* (*c\_valoare*) pentru constante
  - parametrii de substituție (variabilele de substituție din SQL\*Plus) se prefixează cu litera *p*
  - variabilele globale (bind variables) se prefixează cu *g*.
- Variabilele pot fi inițializate, iar dacă o variabilă nu este inițializată, valoarea implicită a acesteia este *NULL*. Dacă o variabilă este declarată *NOT NULL*, atunci ea va fi obligatoriu inițializată.
- Constantele trebuie inițializate când sunt declarate, altfel apare eroare la compilare.

**!!! Afișarea valorii variabilelor** se face cu ajutorul procedurilor:

```
DBMS_OUTPUT.PUT(sir_caractere);
```

```
DBMS_OUTPUT.PUT_LINE(sir_caractere);
```

**Obs:** se utilizează **SET SERVEROUTPUT ON** pentru activarea modului afișare.

## IV. Instrucțiuni PL/SQL

- iterative (*LOOP*, *WHILE*, *FOR*),
- de atribuire (*:=*),
- condiționale (*IF*, *CASE*),
- de salt (*GOTO*, *EXIT*),
- instrucțiunea vidă (*NULL*).

**Observații**

- Comentariile sunt ignorate de compilatorul PL/SQL. Există două tipuri de comentarii:

- pe o singură linie, prefixate de simbolurile "--", care încep în orice punct al liniei și se termină la sfârșitul acesteia și
- pe mai multe linii, care sunt delimitate de simbolurile "/\*" și "\*/".
- Caracterul ";" este separator pentru instrucțiuni.
- Operatorii din *PL/SQL*, ca și ordinea de execuție a acestora, sunt identici cu cei din *SQL*. În *PL/SQL* este introdus un nou operator (\*\*).
- Un identificator este vizibil în blocul în care este declarat și în toate subblocurile, procedurile și funcțiile încuibărite în acesta. Dacă blocul nu găsește identificatorul declarat local, atunci îl caută în secțiunea declarativă a blocurilor care includ blocul respectiv și niciodată nu caută în blocurile încuibărite în acesta.
- Comenzile *SQL \*Plus* nu pot să apară într-un bloc *PL/SQL*.
- În comanda *SELECT* trebuie specificate variabilele care recuperează rezultatul acțiunii acestei comenzi. În clauza *INTO*, care este obligatorie, pot fi folosite variabile *PL/SQL* sau variabile de legătură.
- Referirea la o variabilă de legătură se face în *PL/SQL* prin prefixarea acestei variabile utilizând caracterul ":".
- Cererea *SELECT* trebuie să întoarcă ca rezultat o singură linie. Dacă întoarce mai multe linii, atunci apare eroarea *TOO\_MANY\_ROWS*, iar dacă comanda nu găsește date se generează eroarea *NO\_DATA\_FOUND*.

!!! Pentru evaluarea unei condiții logice care apare în comenzile limbajului, trebuie remarcat că orice expresie ce conține o valoare *null* este evaluată *null*. Singura excepție o constituie operatorul de concatenare.

## 1) Instrucțiunea de atribuire

**variabila := expresie;**

**Obs:** Nu poate fi asignată valoarea *NULL* unei variabile care a fost declarată *NOT NULL*.

## 2) Instrucțiunea IF

```

IF condiție1 THEN
    secvența_de_comenzi_1
[ELSIF condiție2 THEN
    secvența_de_comenzi_2]
...
[ELSE
    secvența_de_comenzi_n]
END IF;

```

Este permis un număr arbitrar de opțiuni *ELSIF*, dar poate fi cel mult o clauză *ELSE*. Aceasta se referă la ultimul *ELSIF*.

## 3) Instrucțiunea CASE

*Oracle9i* furnizează o nouă comandă (*CASE*) care permite implementarea unor condiții multiple. Instrucțiunea are următoarea formă sintactică:

```

[<<eticheta>>]
CASE test_var
    WHEN valoare_1 THEN secvența_de_comenzi_1;
    WHEN valoare_2 THEN secvența_de_comenzi_2,
    ...

```

```

WHEN valoare_k THEN secvența_de_comenzi_k;
[ELSE alta_secvența;]
END CASE [eticheta];

```

Sau următoarea formă, în care fiecare clauză *WHEN* conține o expresie booleană.

```

[<<eticheta>>]
CASE
  WHEN condiție_1 THEN secvența_de_comenzi_1;
  WHEN condiție_2 THEN secvența_de_comenzi_2,
  ...
  WHEN condiție_k THEN secvența_de_comenzi_k;
[ELSE alta_secvența;]
END CASE [eticheta];

```

#### 4) Instrucțiuni iterative

Instrucțiunile de ciclare pot fi:

- încuibărite pe multiple niveluri;
- etichetate;
- ieșirea din ciclare se poate realiza cu ajutorul comenzii *EXIT*.

##### a) *LOOP*

```

      secvența_de_comenzi
END LOOP;

```

Comanda se execută cel puțin o dată. Dacă nu este utilizată comanda *EXIT*, ciclarea ar putea continua la infinit.

##### b) *WHILE condiție LOOP*

```

      secvența_de_comenzi
END LOOP;

```

Dacă condiția este evaluată ca fiind *FALSE* sau *NULL*, atunci secvența de comenzi nu este executată și controlul trece la instrucțiunea după *END LOOP*.

Instrucțiunea repetitivă *FOR* (ciclare cu pas) permite executarea unei secvențe de instrucțiuni pentru valori ale variabilei *contor* cuprinse între două limite, *lim\_inf* și *lim\_sup*. Dacă este prezentă opțiunea *REVERSE*, iterația se face (în sens invers) de la *lim\_sup* la *lim\_inf*.

##### c) *FOR contor\_ciclu IN [REVERSE] lim\_inf..lim\_sup LOOP*

```

      secvența_de_comenzi
END LOOP;

```

Variabila *contor\_ciclu* nu trebuie declarată, ea fiind implicit de tip *BINARY\_INTEGER* și este neidentificată în afara ciclului. Pasul are implicit valoarea 1 și nu poate fi modificat. Limitele domeniului pot fi variabile sau expresii, dar care pot fi convertite la întreg.

#### 5) Instrucțiuni de salt

Instrucțiunea *EXIT* permite ieșirea dintr-un ciclu. Controlul trece fie la prima instrucțiune situată după *END LOOP*-ul corespunzător, fie la instrucțiunea având eticheta *nume\_eticheta*.

```

EXIT [nume_eticheta] [WHEN condiție];

```

Numele etichetelor urmează aceleași reguli ca cele definite pentru identificatori. Eticheta se plasează înaintea comenzii, fie pe aceeași linie, fie pe o linie separată. Etichetele se definesc prin intercalare între "<<" și ">>".

*Exemplu:*

```

DECLARE
  v_contor  BINARY_INTEGER := 1;
  raspuns   VARCHAR2(10);
  alt_raspuns VARCHAR2(10);
BEGIN
  ...
  <<exterior>>
  LOOP
    v_contor := v_contor + 1;
    EXIT WHEN v_contor > 70;
    <<interior>>
    LOOP
      ...
      EXIT exterior WHEN raspuns = 'DA';
      -- se parasesc ambele cicluri
      EXIT WHEN alt_raspuns = 'DA';
      -- se paraseste ciclul interior
      ...
    END LOOP interior;
    ...
  END LOOP exterior;
END;
```

**GOTO** *nume\_eticheta*;

Nu este permis saltul:

- în interiorul unui bloc (subbloc);
- în interiorul unei comenzi *IF*, *CASE* sau *LOOP*;
- de la o clauză a comenzii *CASE*, la altă clauză aceleași comenzi;
- de la tratarea unei excepții, în blocul curent;
- în exteriorul unui subprogram.

**7) Instrucțiunea vidă.** *NULL* este instrucțiunea care nu are nici un efect. Nu trebuie confundată instrucțiunea *NULL* cu valoarea *null*!

## V. Blocuri PL/SQL

*PL/SQL* este un limbaj cu structura de **bloc**, adică programele sunt compuse din blocuri care pot fi complet separate sau încuibărite unul în altul.

Un program poate cuprinde unul sau mai multe blocuri. Un bloc poate fi anonim sau neanonim.

- **Blocurile anonime** sunt blocuri *PL/SQL* fără nume, care sunt construite dinamic și sunt executate o singură dată. Acest tip de bloc nu are argumente și nu returnează un rezultat.
- **Blocurile neanonime** sunt fie blocuri având un nume (etichetate), care sunt construite static sau dinamic și sunt executate o singură dată, fie subprograme, pachete sau declanșatori..

**Structura unui bloc *PL/SQL*** este compusă din trei secțiuni distincte:

Blocul *PL/SQL* are următoarea structură generală:

```

[<<nume_bloc>>]
[DECLARE
  instrucțiuni de declarare]
BEGIN
```

*instrucțiuni executabile (SQL sau PL/SQL)*

**[EXCEPTION**

*tratarea erorilor]*

**END** [nume\_bloc];

Dacă blocul *PL/SQL* este executat fără erori, invariant va apare mesajul:

*PL/SQL procedure successfully completed*

## Compatibilitate SQL

Din punct de vedere al compatibilității *PL/SQL versus SQL* există următoarele reguli de bază:

- *PL/SQL* furnizează toate comenzile *LMD* ale lui *SQL*, comanda *SELECT* cu clauza *INTO*, comenzile *LCD*, funcțiile, pseudo-coloanele și operatorii *SQL*;
- *PL/SQL* nu furnizează comenzile *LDD*.
- Majoritatea funcțiilor *SQL* sunt disponibile în *PL/SQL*.
- Există funcții noi, specifice *PL/SQL*, cum sunt funcțiile *SQLCODE* și *SQLERRM*.
- Există funcții *SQL* care nu sunt disponibile în instrucțiuni procedurale (de exemplu, **DECODE**, **NULLIF**, **funcțiile grup**), dar care sunt disponibile în instrucțiunile *SQL* dintr-un bloc *PL/SQL*. *SQL* nu poate folosi funcții sau atribute specifice *PL/SQL*.

! Funcțiile grup trebuie folosite cu atenție, deoarece instrucțiunea *SELECT ... INTO* nu poate conține clauza *GROUP BY*.

## Exerciții

1. Care dintre următoarele declarații nu sunt corecte și explicați de ce:

- a) DECLARE  
v\_id NUMBER(4);
- b) DECLARE  
v\_x, v\_y, v\_z VARCHAR2(10);
- c) DECLARE  
v\_birthdate DATE NOT NULL;
- d) DECLARE  
v\_in\_stock BOOLEAN := 1;
- e) DECLARE  
TYPE name\_table\_type IS TABLE OF VARCHAR2(20)  
INDEX BY BINARY\_INTEGER;  
dept\_name\_table name\_table\_type;

2. Determinați tipul de date al rezultatului în fiecare din atribuirile următoare:

- a) v\_days\_to\_go := v\_due\_date - SYSDATE;
- b) v\_sender := USER || ' ' || TO\_CHAR(v\_dept\_no);
- c) v\_sum := \$100,000 + \$250,000;
- d) v\_flag := TRUE;
- e) v\_n1 := v\_n2 > (2 \* v\_n3);
- f) v\_value := NULL;

3. Se consideră următorul bloc *PL/SQL*:

```
<<bloc>>
DECLARE
  v_cantitate NUMBER(3) := 300;
  v_mesaj VARCHAR2(255) := 'Produs 1';
```

```

BEGIN
  <<subbloc>>
  DECLARE
    v_cantitate NUMBER(3) := 1;
    v_mesaj    VARCHAR2(255) := 'Produs 2';
    v_locatie   VARCHAR2(50) := 'Europa';
  BEGIN
    v_cantitate := v_cantitate + 1;
    v_locatie   := v_locatie || 'de est';
  END;
  v_cantitate:= v_cantitate + 1;
  v_mesaj := v_mesaj || ' se afla in stoc';
  v_locatie := v_locatie || 'de est' ;
END;
/

```

Evaluati:

- valoarea variabilei v\_cantitate în subbloc; (2)
- valoarea variabilei v\_locatie la poziția în subbloc ; (Europe de est, având tipul VARCHAR2)
- valoarea variabilei v\_cantitate în blocul principal ; (601, iar tipul este NUMBER)
- valoarea variabilei v\_mesaj în blocul principal ; ('Produs 1 se afla in stoc')
- valoarea variabilei v\_locatie în blocul principal. (nu este corectă ; v\_locatie nu este vizibilă în afara subblocului)

4. Să se creeze un bloc anonim în care se declară o variabilă v\_oras de tipul coloanei city (locations.city%TYPE). Atribuiți acestei variabile numele orașului în care se află departamentul având codul 30. Afișați în cele două moduri descrise anterior.

<pre> SET SERVEROUTPUT ON DECLARE   v_oras locations.city%TYPE; BEGIN   SELECT city   INTO v_oras   FROM departments d, locations l   WHERE d.location_id=l.location_id AND   department_id=30;   DBMS_OUTPUT.PUT_LINE('Orașul este '     v_oras); END; / SET SERVEROUTPUT OFF </pre>	<pre> VARIABLE g_oras VARCHAR2(20) BEGIN   SELECT city   INTO :g_oras   FROM departments d, locations l   WHERE d.location_id=l.location_id AND   department_id=30;   END; / PRINT g_oras </pre>
---	--

5. Să se creeze un bloc anonim în care să se afle media salariilor pentru angajații al căror departament este 50. Se vor folosi variabilele v\_media\_sal de tipul coloanei salary și v\_dept (de tip NUMBER).

```

SET SERVEROUTPUT ON
DECLARE
  v_media_sal employees.salary%TYPE;
  v_dept NUMBER:=50;
BEGIN
  SELECT AVG(salary)
  INTO v_media_sal
  FROM employees
  WHERE department_id= v_dept;

```

```

DBMS_OUTPUT.PUT_LINE('media salariilor este '|| v_media_sal);
END;
/
SET SERVEROUTPUT OFF

```

6. Să se specifice dacă un departament este mare, mediu sau mic după cum numărul angajaților săi este mai mare ca 30, cuprins între 10 și 30 sau mai mic decât 10. Codul departamentului va fi cerut utilizatorului.

```

ACCEPT p_cod_dep PROMPT 'Introduceti codul departamentului '
DECLARE
  v_cod_dep departments.department_id%TYPE := &p_cod_dep;
  v_numar    NUMBER(3) := 0;
  v_comentariu VARCHAR2(10);
BEGIN
  SELECT COUNT(*)
  INTO   v_numar
  FROM   employees
  WHERE  department_id = v_cod_dep;
  IF v_numar < 10 THEN
    v_comentariu := 'mic';
  ELSIF v_numar BETWEEN 10 AND 30 THEN
    v_comentariu := 'mediu';
  ELSE
    v_comentariu := 'mare';
  END IF;
  DBMS_OUTPUT.PUT_LINE('Departamentul avand codul' || v_cod_dep
    || 'este de tip' || v_comentariu);
END;
/

```

7. Stocați într-o variabilă de substituție p\_cod\_dep valoarea unui cod de departament. Definiți și o variabilă p\_com care reține un număr din intervalul [0, 100). Pentru angajații din departamentul respectiv care nu au comision, să se atribuiască valoarea lui p\_com câmpului commission\_pct. Afișați numărul de linii afectate de această actualizare. Dacă acest număr este 0, să se scrie « Nici o linie actualizata ».

```

SET SERVEROUTPUT ON
SET VERIFY OFF
DEFINE p_cod_dep= 50
DEFINE p_com =10
DECLARE
  v_cod_dep emp_pnu.department_id%TYPE:= &p_cod_dep;
  v_com      NUMBER(2);
BEGIN
  UPDATE emp_pnu
  SET      commission_pct = &p_com/100
  WHERE  department_id= v_cod_dep;
  IF SQL%ROWCOUNT = 0 THEN
    DBMS_OUTPUT.PUT_LINE('Nici o linie actualizata');
  ELSE DBMS_OUTPUT.PUT_LINE(SQL%ROWCOUNT ||' linii actualizate ');
  END IF;
END;
/
SET VERIFY ON

```



SET SERVEROUTPUT OFF

**Obs:** (vom reveni în laboratorul despre cursoare)

Atributele cursoarelor implicite :

- SQL%ROWCOUNT – Numarul de linii afectate de cea mai recenta comanda SQL;
- SQL%FOUND – Atribut boolean ce returneaza TRUE daca ultima comanda SQL a afectat cel putin o linie;
- SQL%NOTFOUND – Atribut boolean ce returneaza TRUE daca ultima comanda SQL nu a afectat nici o linie
- SQL%ISOPEN – Atribut boolean ce returneaza TRUE daca cursorul implicit asociat ultimei comenzi a ramas deschis. Nu e niciodata true pentru ca serverul inchide automat cursorul la terminarea comenzii SQL.

8. În structura tabelului *emp\_pnu* se va introduce un nou câmp (*stea* de tip *VARCHAR2(200)*). Să se creeze un bloc *PL/SQL* care va reactualiza acest câmp, introducând o steluță pentru fiecare 100\$ din salariul unui angajat al cărui cod este specificat de către utilizator.

```
ALTER TABLE emp_pnu
ADD stea VARCHAR2(200);
```

```
SET VERIFY OFF
ACCEPT p_cod_ang PROMPT 'Dati codul unui angajat'
DECLARE
  v_cod_ang emp_pnu.employee_id%TYPE := &p_cod_ang;
  v_salariu emp_pnu.salary%TYPE;
  v_stea emp_pnu.stea%TYPE:= NULL;
BEGIN
  SELECT NVL(ROUND(salary/100),0)
  INTO    v_salariu
  FROM    emp_pnu
  WHERE   employee_id = v_cod_ang;
  FOR i IN 1..v_salariu LOOP
    v_stea := v_stea || '*'
  END LOOP;
  UPDATE emp_pnu
  SET     stea = v_stea
  WHERE   employee_id = v_cod_ang;
  COMMIT;
END;
/
SET VERIFY ON
```