

## **Subprograme PL/SQL (funcții și proceduri)**

Un subprogram este un bloc PL/SQL cu nume (spre deosebire de blocurile anonime) care poate primi parametri și poate fi invocat dintr-un anumit mediu ( de exemplu, SQL\*Plus, Oracle Forms, Oracle Reports etc.)

Subprogramele sunt bazate pe structura de bloc PL/SQL. Similar, ele conțin o parte declarativă facultativă, o parte executabilă obligatorie și o parte de tratare de excepții facultativă.

- Exista 2 tipuri de subprograme:
  - proceduri;
  - funcții (trebuie să conțină cel puțin o comandă RETURN);
- Subprogramele pot fi :
  - locale (în cadrul altui bloc PL/SQL sau subprogram)
  - stocate (create cu comanda CREATE) - odată create, procedurile și funcțiile sunt stocate în baza de date de aceea ele se numesc subprograme stocate.

➤ Sintaxa simplificată pentru crearea unei proceduri este următoarea:

```
[CREATE [OR REPLACE] ] PROCEDURE nume_procedură [ (lista_parametri) ]  
  {IS | AS}  
  [declarații locale]  
BEGIN  
  partea executabilă  
[EXCEPTION  
  partea de tratare a excepțiilor]  
END [nume_procedură];
```

➤ Sintaxa simplificată pentru crearea unei funcții este următoarea:

```
[CREATE [OR REPLACE] ] FUNCTION nume_funcție  
  [ (lista_parametri) ]  
  
  RETURN tip_de_date  
  {IS | AS}  
  [declarații locale]  
BEGIN  
  partea executabilă  
[EXCEPTION  
  partea de tratare a excepțiilor]  
END [nume_funcție];
```

- Lista de parametri conține specificații de parametri separate prin virgulă de forma :

```
nume_parametru mod_parametru tip_parametru;  
○ mod_parametru specifică dacă parametrul este:

- de intrare (IN) – singurul care poate avea o valoare inițială
- de intrare / ieșire (IN OUT)
- de ieșire (OUT)

- mod_parametru are valoarea implicită IN.

```

- O funcție îndeplinește următoarele condiții:-

- Accepta numai parametrii de tip IN
- Accepta numai tipuri de date SQL, nu si tipuri specifice PL/SQL
- Returneaza valori de tipuri de date SQL.
- Nu modifica tabelul care este blocat pentru comanda respectiva (*mutating tables*)

- Poate fi folosită în lista de expresii a comenzii `SELECT`, clauza `WHERE` și `HAVING`, `CONNECT BY`, `START WITH`, `ORDER BY`, `GROUP BY`, clauza `VALUES` a comenzii `INSERT`, clauza `SET` a comenzii `UPDATE`.

- În cazul în care se modifică un obiect (vizualizare, tabel etc) de care depinde un subprogram, acesta este invalidat. Revalidarea se face fie prin recrearea subprogramului fie prin comanda:

```
ALTER PROCEDURE nume_proc COMPILE;
ALTER FUNCTION nume_functie COMPILE;
```

- Ștergerea unei funcții sau proceduri se realizează prin comenzile:

```
DROP PROCEDURE nume_proc;
DROP FUNCTION nume_functie;
```

- Informații despre procedurile și funcțiile deținute de utilizatorul curent se pot obține interogând vizualizarea `USER_OBJECTS` din dicționarul datelor.

```
SELECT OBJECT_NAME, OBJECT_TYPE, STATUS
FROM USER_OBJECTS
WHERE OBJECT_TYPE IN ('PROCEDURE','FUNCTION');
```

Obs: `STATUS` – starea subprogramului (validă sau invalidă).

- Codul complet al unui subprogram poate fi vizualizat folosind următoarea sintaxă:

```
SELECT TEXT
FROM USER_SOURCE
WHERE NAME = 'nume_subprogram'
ORDER BY LINE;
```

- Eroarea apărută la compilarea unui subprogram poate fi vizualizată folosind următoarea sintaxă:

```
SELECT LINE, POSITION, TEXT
FROM USER_ERRORS
WHERE NAME = 'nume';
```

- Erorile pot fi vizualizate și prin intermediul comenzii `SHOW ERRORS`.

- Descrierea specificației unui subprogram se face prin comanda `DESCRIBE`.

- Când este apelată o procedură `PL/SQL`, sistemul *Oracle* furnizează două metode pentru definirea parametrilor actuali:

- specificarea explicită prin nume;

- specificarea prin poziție.

Exemplu: subprog(a tip\_a, b\_tip\_b, c tip\_c, d tip\_d)

- specificare prin poziție:

```
subprog(var_a,var_b,var_c,var_d);
```

- specificare prin nume

```
subprog(b=>var_b,c=>var_c,d=>var_d,a=>var_a);
```

- specificare prin nume și poziție

```
subprog(var_a,var_b,d=>var_d,c=>var_c);
```

### Exerciții [Proceduri stocate]:

1. Să se creeze o procedură stocată fără parametri care afișează un mesaj "Programare PL/SQL", ziua de astăzi în formatul DD-MONTH-YYYY și ora curentă, precum și ziua de ieri în formatul DD-MON-YYYY.

```
CREATE PROCEDURE first_pnu IS
    azi DATE := SYSDATE;
    ieri azi%TYPE;
BEGIN
```

```

DBMS_OUTPUT.PUT_LINE('Programare PL/SQL') ;
DBMS_OUTPUT.PUT_LINE(TO_CHAR(azi, 'dd-month-yyyy hh24:mi:ss'));
ieri := azi -1;
DBMS_OUTPUT.PUT_LINE(TO_CHAR(ieri, 'dd-mon-yyyy'));

```

END;

/

La promptul SQL apelam procedura astfel: EXECUTE first\_pnu;

**2.** Să se șteargă procedura precedentă și să se re-creeze, astfel încât să accepte un parametru IN de tip VARCHAR2, numit p\_nume. Mesajul afișat de procedură va avea forma « <p\_nume> invata PL/SQL ». Invocați procedura cu numele utilizatorului curent furnizat ca parametru.

```
DROP PROCEDURE first_pnu ;
```

```
CREATE PROCEDURE first_pnu(p_nume VARCHAR2) IS
```

```
....
```

```
Pentru apel: EXECUTE first_pnu(USER);
```

**3. a)** Creați o copie JOBS\_pnu a tabelului JOBS. Implementați constrângerea de cheie primară asupra lui JOBS\_pnu.

```
CREATE TABLE jobs_pnu AS SELECT * FROM jobs ;
```

```
ALTER TABLE jobs_pnu ADD CONSTRAINT pk_jobs_pnu PRIMARY KEY(job_id);
```

**b)** Creați o procedură ADD\_JOB\_pnu care inserează un nou job în tabelul JOBS\_pnu. Procedura va avea 2 parametri IN p\_id și p\_title corespunzători codului și denumirii noului job.

```
CREATE OR REPLACE PROCEDURE ADD_JOB_pnu
(p_job_id IN jobs.job_id%TYPE, p_job_title IN jobs.job_title%TYPE)
```

```
IS
```

```
BEGIN
```

```
    INSERT INTO jobs_pnu (job_id, job_title)
```

```
    VALUES (p_job_id, p_job_title);
```

```
    COMMIT;
```

```
END add_job_pnu;
```

**c)** Testați procedura, invocând-o astfel:

```
EXECUTE ADD_JOB_pnu('IT_DBA', 'Database Administrator');
```

```
SELECT * FROM JOBS_pnu;
```

```
EXECUTE ADD_JOB_pnu('ST_MAN', 'Stock Manager');
```

```
SELECT * FROM JOBS_pnu;
```

**4. a)** Creați o procedură stocată numită UPD\_JOB\_pnu pentru modificarea unui job existent în tabelul JOBS\_pnu. Procedura va avea ca parametri codul job-ului și noua sa denumire (parametri IN). Se va trata cazul în care nu are loc nici o actualizare.

```
CREATE OR REPLACE PROCEDURE UPD_JOB_pnu
(p_job_id IN jobs.job_id%TYPE, p_job_title IN jobs.job_title%TYPE)
```

```
IS
```

```
BEGIN
```

```
    UPDATE jobs_pnu
```

```
    SET job_title = p_job_title
```

```
    WHERE job_id = p_job_id;
```

```
    IF SQL%NOTFOUND THEN
```

```
        RAISE_APPLICATION_ERROR(-20202, 'Nici o actualizare');
```

```
        -- sau doar cu afisare mesaj
```

```
        -- DBMS_OUTPUT.PUT_LINE('Nici o actualizare');
```

```
    END IF;
```

```
END upd_job_pnu;
```

**b)** Testați procedura, invocând-o astfel:

```
EXECUTE UPD_JOB_pnu('IT_DBA', 'Data Administrator');
```

```
SELECT * FROM job_pnu
WHERE UPPER(job_id) = 'IT_DBA'
EXECUTE UPD_JOB('IT_WEB', 'Web master');
```

*Obs :* A doua invocare va conduce la apariția excepției. Analizați ce s-ar fi întâmplat dacă nu prevedeam această excepție, punând între comentarii liniile aferente din procedură și recreând-o cu CREATE OR REPLACE PROCEDURE...

5. a) Creați o procedură stocată numită DEL\_JOB\_pnu care șterge un job din tabelul JOBS\_pnu. Procedura va avea ca parametru (IN) codul job-ului. Includeți o excepție corespunzătoare situației în care nici un job nu este șters.

b) Testați procedura, invocând-o astfel:

```
DEL_JOB_pnu('IT_DBA');
DEL_JOB_pnu('IT_WEB');
```

6. a) Să se creeze o procedură stocată care calculează salariul mediu al angajaților, returnându-l prin intermediul unui parametru de tip OUT.

```
CREATE OR REPLACE PROCEDURE p8l4_pnu (p_salAvg OUT employees.salary%TYPE)
AS
BEGIN
    SELECT AVG(salary)
    INTO p_salAvg
    FROM employees;
END;
/
```

### [Funcții stocate]

7. Să se creeze o funcție stocată care determină numărul de salariați din employees angajați după 1995, într-un departament dat ca parametru. Să se apeleze această funcție.

```
CREATE OR REPLACE FUNCTION p14l4_pnu (p_dept employees.department_id%TYPE)
RETURN NUMBER
IS
    rezultat NUMBER;
BEGIN
    SELECT COUNT(*)
    INTO rezultat
    FROM employees
    WHERE department_id=p_dept
    AND TO_CHAR(hire_date,'yyyy')>1995;
    RETURN rezultat;
END p14l4_pnu;
/
```

```
1) VARIABLE nr NUMBER
EXECUTE :nr := p14l4_pnu (80);
PRINT nr
```

```
2) SELECT p14l4_pnu (80)
FROM dual;
```

8. Să se calculeze recursiv numărul de permutări ale unei mulțimi cu n elemente, unde n va fi transmis ca parametru.

```
CREATE OR REPLACE FUNCTION permutari_pnu(p_n NUMBER)
```

```

        RETURN INTEGER IS
BEGIN
    IF (n=0) THEN
        RETURN 1;
    ELSE
        RETURN p_n*permutari_pnu (n-1);
    END IF;
END permutari_pnu;
/
VARIABLE g_n NUMBER
EXECUTE :g_n := permutari_pnu (5);
PRINT g_n

```

**9.** Să se afișeze numele, job-ul și salariul angajaților al căror salariu este mai mare decât media salariilor din tabelul employees.

```

CREATE OR REPLACE FUNCTION medie_pnu
RETURN NUMBER IS
medie NUMBER;
BEGIN
    SELECT AVG(salary)
    INTO    medie
    FROM    employees;
    RETURN medie;
END;
/
SELECT last_name, job_id, salary
FROM    employees
WHERE   salary >= medie_pnu;

```