SGBD-Anul 3
Laborator 4

Subprograme PL/SQL (funcții şi proceduri)

Un subprogram este un bloc PL/SQL cu nume (spre deosebire de blocurile anonime) care poate primi parametri şi 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 şi 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 şi 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;

o *mod_parametru* specifică dacă parametrul este:

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

o *mod_parametru* are valoarea implicită IN.

- O functie îndeplineste urmatoarele conditii:

- 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 folosita in lista de expresii a comenzii SELECT, clauza WHERE si 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;

- Ştergerea unei funcții sau proceduri se realizează prin comenzile:
  DROP PROCEDURE nume_proc;
  DROP FUNCTION nume_functie;

➢ Informații despre procedurile şi 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 şi 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 şi poziție

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

Exerciții:

I. [Proceduri locale]

1. Să se declare o procedură locală într-un bloc PL/SQL anonim prin care să se introducă în tabelul DEP o nouă înregistrare precizând, prin intermediul parametrilor, valori pentru toate câmpurile. Invocați procedura în cadrul blocului. Interogați tabelul DEP şi apoi anulați modificările (ROLLBACK).

2. Să se declare o procedură locală care are parametrii următori:
- p_rezultat (parametru de tip IN OUT) de tipul coloanei last_name din tabelul employees;
- p_comision (parametru de tip OUT) de tipul coloanei commission_pct din employees, inițializat cu NULL;
- p_cod (parametru de tip IN) de tipul coloanei employee_id din employees, inițializat cu NULL.

Dacă p_comision nu este NULL atunci în p_rezultat se va memora numele salariatului care are salariul maxim printre salariații având comisionul respectiv. În caz contrar, în p_rezultat se va memora numele salariatului al cărui cod are valoarea dată la apelarea procedurii.

II. [Proceduri stocate]

3. Să se creeze o procedură stocată fără parametri care afişează un mesaj "Programare PL/SQL", ziua de astăzi în formatul DD-MONTH-YYYY şi ora curentă, precum şi ziua de ieri în formatul DD-MON-YYYY. La promptul SQL apelam procedura astfel: EXECUTE first;

4. Să se şteargă procedura precedentă şi să se re-creeze, astfel încât să accepte un parametru IN de tip VARCHAR2, numit p_nume. Mesajul afişat de procedură va avea forma « <p_nume> invata PL/SQL». Invocați procedura cu numele utilizatorlui curent furnizat ca parametru.

5. a) Creați o copie JOBS2 a tabelului JOBS. Implementați constrângerea de cheie primară asupra lui JOBS2.

b) Creați o procedură ADD_JOB care inserează un nou job în tabelul JOBS2. Procedura va avea 2 parametri IN p_id şi p_title corespunzători codului şi denumirii noului job.

6. a) Creați o procedură stocată numită UPD_JOB pentru modificarea unui job existent în tabelul JOBS2. Procedura va avea ca parametri codul job-ului şi noua sa denumire (parametri IN). Se va trata cazul în care nu are loc nici o actualizare (SQL%NOTFOUND THEN RAISE_APPLICATION_ERROR).

b) Testați procedura, invocând-o astfel:

```
EXECUTE UPD_JOB('IT_DBA', 'Data Administrator');
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 şi recreând-o cu CREATE OR REPLACE PROCEDURE…

7. a) Creați o procedură stocată numită DEL_JOB care şterge un job din tabelul JOBS2. Procedura va avea ca parametru (IN) codul job-ului. Includeți o excepție corespunzătoare situației în care nici un job nu este şters.

b) Testați procedura, invocând-o astfel: DEL_JOB('IT_DBA'); DEL_JOB('IT_WEB');

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

b) Să se apeleze procedura regăsind valoarea medie a salariilor într-o variabilă gazdă. Afişați valoarea variabilei.

```
VARIABLE g_medie NUMBER;
EXECUTE pb (:g_medie)
PRINT g_medie
```

9. a) Să se creeze o procedură stocată care primeşte printr-un parametru salariul unui angajat şi returnează prin intermediul aceluiaşi parametru salariul actualizat astfel: dacă salariul este mai mic decât 3000, valoarea lui creşte cu 20%, dacă este cuprins între 3000 şi 7000 valoarea lui creşte cu 15%, dacă este mai mare decât 7000 va fi mărit cu 10%, iar dacă este null va lua valoarea 1000.

b) Să se declare o variabilă gazdă g_sal (VARIABLE). Să se scrie un bloc anonim PL/SQL prin care se va atribui variabilei g_sal valoarea unei variabile de substituție citite de la tastatură (ACCEPT). Să se apeleze procedura pentru această valoare şi să se afişeze valoarea returnată.

10. a) Creați o procedură numită GET_EMP care întoarce salariul şi denumirea job-ului pentru un angajat al cărui cod este transmis ca parametru.

b) Executați procedura utilizând câte o variabilă gazdă pentru cei doi parametri OUT.

c) testați procedura atât pentru coduri existente cât şi pentru coduri inexistente în tabelul EMP. Ce se întâmplă atunci când o invocăm pentru un cod inexistent ?

III. [Funcții locale]

11. Să se creeze o procedură stocată care pentru un anumit cod de departament (dat ca parametru) calculează prin intermediul unor funcții locale numărul de salariați care lucrează în el, suma salariilor şi numărul managerilor salariaților care lucrează în departamentul respectiv.

12. Să se creeze două funcții (locale) supraîncărcate (overload) care să calculeze media salariilor astfel:

- prima funcție va avea ca argument codul departamentului, adică funcția calculează media salariilor din departamentul specificat;
- a doua funcție va avea două argumente, unul reprezentând codul departamentului, iar celălalt reprezentând job-ul, adică funcția va calcula media salariilor dintr-un anumit departament şi care aparțin unui job specificat.

IV. [Funcții stocate]

13. a) Creați o funcție stocată numită GET_JOB care returnează titlul unui job al cărui cod este specificat ca parametru.

b) Creați o variabilă gazdă de tip VARCHAR2(35), numită g_titlu. Invocați funcția, returnând rezultatul în variabila g_titlu. Tipăriți valoarea acestei variabile.

14. 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 prin diferite modalități:

- printr-o variabilă de legătură;
- folosind comanda CALL (CALL pb (*cod*) INTO :variabila);
- printr-o comandă SELECT;
- într-un bloc PL/SQL.

15. a) Creați o funcție numită GET_ANUAL care returnează salariul anual calculat pe baza valorilor salariului annual şi a comisionului, introduse ca parametri (pSalary * 12 * (1 + pComision)). Parametrii pot avea valoarea null, dar funcția nu va îbtoarce niciodată null.

b) Utilizați funcția într-o comandă SELECT pentru afişarea codurilor, numelor şi salariilor anuale ale angajaților din departamentul 50.

16. a) Creați o funcție numită VALID_DEPTID pentru validarea unui cod de departament specificat ca parametru. Funcția va întoarce o valoare booleana (TRUE dacă departamentul există).

b) - Creați o procedură numită ADD_EMP care adaugă un angajat în tabelul EMP. Linia respectivă va fi adăugată în tabel doar dacă departamentul specificat este valid, altfel utilizatorul va primi un mesaj adecvat.

- Procedura va avea următorii parametrii, cu valorile DEFAULT specificate între paranteze : first_name, last_name, email, job_id (SA_REP), manager_id (145), salary (1000), commission_pct (0), department_id (30).

- Pentru codul angajatului se va utiliza o secvență EMP_SEQ (emp_seq.NEXTVAL) , iar data angajării se consideră a fi TRUNC(SYSDATE).

c) Testați procedura, adăugând un angajat pentru care se specifică numele, prenumele, codul departamentului = 15, iar restul parametrilor se lasă DEFAULT.

Adăugați un angajat pentru care se specifică numele, prenumele, codul departamentului =80, iar restul parametrilor rămân la valorile DEFAULT.

Adăugați un angajat precizând valori pentru toți parametrii procedurii.

```
EXECUTE add_emp(p_lname => 'Harris', p_fname=>'Jane', p_email =>'JHarris',
        p_dept_id=>15);
```

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

18. Să se afişeze numele, job-ul şi salariul angajaților al căror salariu este mai mare decât media salariilor din tabelul employees (calculata cu ajutorul unei functii).

19. a) Analizați subprogramele create anterior în vizualizarea din dicționarul datelor USER_OBJECTS.

b) Regăsiți codul unuia dintre subprogramele create anterior în vizualizarea USER_SOURCE.

c) Aflați tipul parametrilor uneia dintre procedurile create anterior utilizând comanda DESCRIBE.

Exerciții propuse

1. Să se creeze o procedură stocată care măreşte salariile angajaților care nu au comision şi au media mai mică decât cea a departamentului în care lucrează cu o valoare transmisă ca parametru.

2. Să se creeze o funcție stocată care determină numărul de salariați care au fost angajați după toți salariații ai unui manager al cărui cod este dat ca parametru. Să se apeleze această funcție într-un bloc PL/SQL.

3. Să se declare o procedură locală prin care să se introducă în tabelul locations o nouă înregistrare.

4. Să se declare o procedură locală care are următorii parametri:

- p_rezultat (parametru de ieşire) de tip NUMBER;
- p_job_id (parametru de intrare) de tip job_id din jobs2, inițializat cu NULL;
- p_titlu (parametru de intrare) de tip job_title din jobs2, inițializat cu NULL.

Dacă job_id nu este NULL atunci în rezultat se va memora numărul de angajați având codul job-ului specificat ca parametru. În caz contrar, în rezultat se va memora numărul de angajați care au titlul job-ului dat de al treilea parametru din procedura. Tratați excepțiile care pot apărea

6. Să se creeze trei funcții locale cu acelaşi nume care să calculeze numărul de salariați astfel:

- prima funcție va avea ca argument codul departamentului, adică funcția calculează numărul de salariați din departamentul specificat;
- a doua funcție va avea două argumente, unul reprezentând codul departamentului, iar celălalt reprezentând anul angajării, adică funcția va calcula numărul de salariați din departament şi care au fost angajați într-un anumit an;
- a treia funcție va avea trei argumente, unul reprezentând codul departamentului, unul reprezentând anul angajării, iar celălalt grila de salarizare, adică funcția va calcula numărul de salariați din departament, care au fost angajați într-un anumit an şi au salariul într-o anumită grilă de salarizare.

7. Să se creeze o funcție pentru calculul recursiv al combinărilor.

8. Să se modifice salariul unui angajat al cărui cod este introdus ca parametru astfel încât să devină media salariilor angajaților care câştigă comision dintr-un departament dat ca parametru.

9. Să se scrie o funcție care să întoarcă, pentru un angajat al cărui cod este specificat ca parametru, vechimea exprimată în luni. Să se utilizeze funcția într-o instrucțiune SELECT care să întoarcă numele angajaților, salariul şi numărul de luni lucrate.