Tema: Modularizarea aplicatiilor prin utilizarea subprogramelor

Notiunea de subprogram (procedura sau functie) a fost conceputa cu scopul de a grupa o multime de comenzi *SQL* cu instructiuni procedurale, pentru a construi o unitate logica de tratare a unei anumite probleme. În general, procedurile sunt folosite pentru a realiza o actiune, iar functiile pentru a calcula o valoare.

Unitatile de program care pot fi create în *PL/SQL* sunt:

- subprograme locale (definite în partea declarativa a unui bloc *PL/SQL* sau a unui alt subprogram);
- subprograme independente (stocate în baza de date si considerate obiecte ale acesteia);
- subprograme împachetate (definite într-un pachet care încapsuleaza proceduri si functii).

Procedurile si functiile stocate sunt unitati de program *PL/SQL* apelabile (compilate), care exista ca obiecte în schema bazei de date *Oracle*. Recuperarea unui subprogram (în cazul unei corectii) nu cere recuperarea întregii aplicatii. Subprogramul, încarcat în memorie pentru a fi executat, poate fi partajat între aplicatiile care îl solicita.

Este important de facut distinctie între procedurile stocate si procedurile locale (declarate si folosite în blocuri anonime).

- Procedurile si functiile stocate, care sunt compilate si stocate în baza de date, nu mai trebuie sa fie compilate la fiecare executie, în timp ce procedurile locale sunt compilate de fiecare data când este executat blocul care le contine.
- Procedurile declarate si apelate în blocuri anonime sunt temporare (ele nu mai exista dupa ce blocul anonim a fost executat complet). O procedura stocata (creata cu CREATE PROCEDURE sau continuta într-un pachet) este permanenta, în sensul ca ea poate fi invocata de un fisier SQL*Plus, un subprogram PL/SQL sau un declansator.
- Procedurile si functiile stocate pot fi apelate din orice bloc, de catre utilizatorul care are privilegiul EXECUTE asupra acestora, în timp ce procedurile si functiile locale pot fi apelate numai din blocul care le contine.

Când este creat un subprogram stocat, utilizând comanda *CREATE*, subprogramul este depus în dictionarul datelor. Este depus atât textul sursa, cât si forma compilata (*p-code*). Atunci când subprogramul este apelat, *p-code* este citit de pe disc, este depus în *shared pool*, unde poate fi accesat de mai multi utilizatori si este executat daca este necesar. El va parasi *shared pool* conform algoritmului *LRU* (*least recently used*).

Pachetul *DBMS_SHARED_POOL* permite pastrarea de obiecte în *shared pool*. Daca un obiect

este gestionat în aceasta maniera, el va parasi *shared pool* doar daca aceasta se cere explicit. Pentru a pastra în *shared pool* subprograme, pachete, declansatori, cursoare, clase *Java*, tipuri obiect, comenzi *SQL*, secvente este utilizata procedura *DBMS_SHARED_POOL.KEEP*. Unica modalitate de a sterge un obiect pastrat în *shared pool*, fara a reporni baza, este cu ajutorul procedurii *DBMS_SHARED_POOL.UNKEEP*.

Subprogramele se pot declara în blocuri *PL/SQL*, în alte subprograme sau în pachete, dar la sfârsitul sectiunii declarative. La fel ca blocurile *PL/SQL* anonime, subprogramele contin o parte declarativa, o parte executabila si, optional, o parte de tratare a erorilor. Partea declarativa contine declaratii de tipuri, cursoare, constante, variabile, exceptii si subprograme imbricate. Partea executabila contine instructiuni care asigneaza valori, controleaza executia programului si prelucreaza datele. Cea de-a treia parte se ocupa cu tratarea exceptiilor aparute în timpul executiei subprogramului.

Crearea subprogramelor stocate

Principalele etape pentru crearea unui subprogram stocat sunt urmatoarele:

- se editeaza subprogramul (CREATE PROCEDURE sau CREATE FUNCTION) si se salveaza într-un script file SQL;
- se încarca si se executa acest script file, se compileaza codul sursa, se obtine p-code (subprogramul este creat);
- se utilizeaza comanda SHOW ERRORS pentru vizualizarea eventualelor erori la compilare (comanda CREATE PROCEDURE sau CREATE FUNCTION depune codul sursa în dictionarul datelor chiar daca subprogramul contine erori la compilare);
- se executa subprogramul pentru a realiza actiunea dorita (de exemplu, procedura poate fi executata fie utilizând comanda EXECUTE din iSQL*Plus, fie invocând-o dintr-un bloc PL/SQL).

Când este apelat subprogramul, motorul *PL/SQL* executa *p-code*.

Daca exista erori la compilare si se fac corectiile corespunzatoare, atunci este necesara fie comanda *DROP PROCEDURE* (respectiv *DROP FUNCTION*), fie sintaxa *OR REPLACE* în cadrul comenzii *CREATE*.

Atunci când este apelata o procedura *PL/SQL*, *server*-ul *Oracle* parcurge anumite etape care vor fi detaliate în cele ce urmeaza.

- Verifica daca utilizatorul are privilegiul sa execute procedura (fie pentru ca el a creat procedura, fie pentru ca i s-a acordat acest privilegiu).
- Verifica daca procedura este prezenta în shared pool. Daca este prezenta va fi executata, altfel va fi încarcata de pe disc în database buffer cache.
- Verifica daca starea procedurii este valida (VALID) sau invalida (INVALID). Starea unei proceduri PL/SQL este invalida, fie pentru ca au fost detectate erori la compilarea procedurii,

fie pentru ca structura unui obiect s-a schimbat de când procedura a fost executata ultima oara. Daca starea este invalida atunci procedura este recompilata automat. Daca nici o eroare nu a fost detectata, atunci va fi executata noua versiune a procedurii.

- Daca procedura apartine unui pachet atunci toate procedurile si functiile pachetului sunt de asemenea încarcate în database buffer cache (daca nu erau deja acolo). Daca pachetul este activat pentru prima oara într-o sesiune, atunci server-ul va executa blocul de initializare al pachetului.

Proceduri *PL/SQL*

O procedura *PL/SQL* este un program independent care se afla compilat în schema bazei de date *Oracle*. Când procedura este compilata, identificatorul acesteia (stabilit prin comanda *CREATE PROCEDURE*) devine un nume de obiect în dictionarul datelor. Tipul obiectului este *PROCEDURE*.

Sintaxa generala pentru crearea unei proceduri este urmatoarea:

[***CREATE*** [***OR REPLACE***]] ***PROCEDURE*** *nume_procedura*

[(*parametru* [*, parametru …*])]

[***AUTHID*** {***DEFINER*** | ***CURRENT_USER***}]

{***IS*** | ***AS***}

[***PRAGMA AUTONOMOUS_TRANSACTION***;]

[*declaratii locale*]

BEGIN

partea executabila

[***EXCEPTION***

partea de tratare a exceptiilor]

END [*nume_procedura*];

Parametrii au urmatoarea forma sintactica:

nume_parametru [{***IN*** | ***OUT*** [***NOCOPY***] | ***IN OUT*** [***NOCOPY***] }]

tip_de_date [{:= | ***DEFAULT***} *expresie*]

Comanda *CREATE* permite ca procedura sa fie stocata în baza de date. Când procedurile sunt create folosind clauza *CREATE OR REPLACE*, ele vor fi stocate în baza de date în forma compilata, forma care permite executia mai rapida a acestora. Daca procedura exista, atunci clauza *OR REPLACE* va avea ca efect stergerea procedurii si înlocuirea acesteia cu noua versiune. Daca procedura exista, iar optiunea *OR REPLACE* nu este prezenta, atunci comanda *CREATE* va returna eroarea „*ORA-00955*: *Name is already used by an existing object*“.

Clauza *AUTHID* specifica faptul ca procedura stocata se executa cu drepturile proprietarului (implicit) sau ale utilizatorului curent. De asemenea, aceasta clauza precizeaza daca referintele la obiecte

sunt rezolvate în schema proprietarului procedurii sau în cea a utilizatorului curent.

Clauza *PRAGMA_AUTONOMOUS_TRANSACTION* anunta compilatorul *PL/SQL* ca aceasta procedura este autonoma (independenta). Tranzactiile autonome permit suspendarea tranzactiei principale, executarea unor instructiuni *SQL*, permanentizarea sau anularea acestor operatii si continuarea tranzactiei principale.

Parametrii formali (variabile declarate în lista parametrilor specificatiei subprogramului) pot sa fie de tipul

%TYPE, *%ROWTYPE* sau de un tip explicit, fara specificarea dimensiunii.

***Exemplu*:**

Sa se creeze o procedura stocata care micsoreaza cu o valoare data (*cant*) politele de asigurare emise de firma *SALVAL*.

```
CREATE OR REPLACE PROCEDURE mic (cant IN NUMBER) AS BEGIN
UPDATE politaasig
SET   valoare = valoare - cant WHERE   firma = 'SALVAL';
END;
/
```

Daca în subprograme se executa operatii de reactualizare si exista declansatori relativ la aceste operatii care nu trebuie sa se execute, atunci înainte de apelarea subprogramului declansatorii trebuie dezactivati, urmând ca ei sa fie reactivati dupa ce s-a terminat executia subprogramului.

De exemplu, în problema prezentata anterior ar trebui dezactivati declansatorii referitori la tabelul

politaasig, apelata procedura *mic* si în final, reactivati acesti declansatori.

```
ALTER TABLE politaasig DISABLE ALL TRIGGERS; EXECUTE mic(10000)
ALTER TABLE politaasig ENABLE ALL TRIGGERS;
```

***Exemplu*:**

Sa se creeze o procedura locala prin care se insereaza informatii în tabelul *editata_de*.

```
DECLARE
PROCEDURE editare
(v_cod_sursa editata_de.cod_sursa%TYPE, v_cod_autor
                        editata_de.cod_autor%TYPE) IS
BEGIN
INSERT  INTO editata_de
VALUES  (v_cod_sursa,v_cod_autor); END;
BEGIN
…
```

```
editare(75643, 13579);
… END;
/
```

Procedurile stocate pot fi apelate:

- din corpul altei proceduri sau al unui declansator;
- interactiv, de catre utilizator, folosind un instrument Oracle (de exemplu, iSQL*Plus);
- explicit dintr-o aplicatie (de exemplu, Oracle Forms sau prin utilizarea de precompilatoare).

Apelarea unei proceduri se poate face în functie de mediul care o solicita:

1) în *SQL*Plus*, prin comanda

EXECUTE *nume_procedura* [(*lista_parametri_actuali*)];

2) în *PL/SQL*, prin invocarea numelui procedurii urmat de lista parametrilor actuali.

Parametrii actuali sunt variabile sau expresii referite în lista parametrilor subprogramului apelant. Ei trebuie sa fie compatibili ca tip si numar cu parametrii formali.

Functii *PL/SQL*

O functie *PL/SQL* este similara unei proceduri cu exceptia ca ea trebuie sa întoarca un rezultat. O functie fara comanda *RETURN* va genera o eroare la compilare.

Când functia este compilata, identificatorul acesteia devine obiect în dictionarul datelor având tipul *FUNCTION*. Algoritmul din interiorul corpului subprogramului functie trebuie sa asigure faptul ca toate traiectoriile sale conduc la comanda *RETURN*. Daca o traiectorie a algoritmului trimite în partea de tratare a erorilor, atunci *handler*-ul acesteia trebuie sa includa o comanda *RETURN*. Orice functie trebuie sa contina clauza *RETURN* în antet si cel putin o comanda *RETURN* în partea executabila.

Sintaxa simplificata pentru scrierea unei functii este urmatoarea:

[*CREATE* [*OR REPLACE*]] ***FUNCTION*** *nume_functie*

[(*parametru* [*, parametru …*])]

RETURN *tip_de_date*

[***AUTHID*** {***DEFINER*** | ***CURRENT_USER***}] [***DETERMINISTIC***]

{***IS*** | ***AS***}

[***PRAGMA AUTONOMOUS_TRANSACTION***;]

[*declaratii locale*]

BEGIN

partea executabila

[***EXCEPTION***

partea de tratare a exceptiilor]

END [*nume_functie*];

Optiunea *tip_de_date* specifica tipul valorii returnate de functie, tip care nu poate contine specificatii de dimensiune. Daca totusi sunt necesare aceste specificatii, se pot defini subtipuri, iar parametrii si valoarea returnata vor fi declarati de acel subtip.

În interiorul functiei trebuie sa apara instructiunea *RETURN expresie,* unde *expresie* este valoarea rezultatului furnizat de functie. Pot sa fie mai multe comenzi *RETURN* într-o functie, dar numai una din ele va fi executata. Comanda *RETURN* (fara o expresie asociata) poate sa apara si într-o procedura. În acest caz, ea va avea ca efect saltul la comanda ce urmeaza instructiunii apelante.

Optiunea *DETERMINISTIC* ajuta optimizorul *Oracle* în cazul unor apeluri repetate ale aceleasi functii, cu aceleasi argumente. Ea indica posibilitatea folosirii unui rezultat obtinut anterior.

***Observatii*:**

- În blocul PL/SQL al unei functii stocate (cel care defineste actiunea efectuata de functie) nu pot fi referite variabile host sau variabile bind.
- O functie poate accepta unul sau mai multi parametri, dar trebuie sa returneze o singura valoare. Ca si în cazul procedurilor, lista parametrilor este optionala. Daca subprogramul nu are parametri, parantezele nu sunt necesare la declarare si la apelare.
- O procedura care contine un parametru de tip OUT poate fi rescrisa sub forma unei functii.

***Exemplu*:**

Sa se creeze o functie stocata care determina numarul operelor de arta realizate pe pânza, ce au fost achizitionate la o anumita data.

```
CREATE OR REPLACE FUNCTION   numar_opere
(v_a IN opera.data_achizitie%TYPE)
RETURN NUMBER AS
alfa NUMBER;
BEGIN
SELECT COUNT(ROWID)
INTO  alfa
FROM  opera
WHERE material = 'panza' AND data_achizitie = v_a; RETURN alfa;
END numar_opere;
/
```

Daca apare o eroare de compilare, utilizatorul o va corecta în fisierul editat si apoi va trimite compilatorului (cu optiunea *OR REPLACE*) fisierul modificat.

Sintaxa pentru apelul unei functii este:

[[*schema.*]*nume_pachet.*]*nume_functie* [*@dblink*] [(*lista_parametri_actuali*)];

O functie stocata poate fi apelata în mai multe moduri. În continuare sunt prezentate trei

exemple de apelare.

1) Apelarea functiei si atribuirea valorii acesteia într-o variabila de legatura *SQL*Plus*:

```
VARIABLE val NUMBER
EXECUTE :val := numar_opere(SYSDATE) PRINT val
```

Când este utilizata declaratia *VARIABLE* pentru variabilele *host* de tip *NUMBER* nu trebuie specificata dimensiunea, iar pentru cele de tip *CHAR* sau *VARCHAR2* valoarea implicita este 1 sau poate fi specificata o alta valoare între paranteze. *PRINT* si *VARIABLE* sunt comenzi *SQL*Plus.*

2) Apelarea functiei într-o instructiune *SQL*:

```
SELECT numar_opere(SYSDATE) FROM  dual;
```

3) Aparitia numelui functiei într-o comanda din interiorul unui bloc *PL/SQL* (de exemplu, într-o instructiune de atribuire):

```
SET SERVEROUTPUT ON
ACCEPT data PROMPT 'dati data achizitionare' DECLARE
num   NUMBER;
v_data opera.data_achizitie%TYPE := '&data'; BEGIN
num := numar_opere(v_data);
DBMS_OUTPUT.PUT_LINE('numarul operelor de arta achizitionate la
data ' || TO_CHAR(v_data) || ' este '
|| TO_CHAR(num));
END;
/
SET SERVEROUTPUT OFF
```

***Exemplu*:**

Sa se creeze o procedura stocata care pentru un anumit tip de opera de arta (dat ca parametru) calculeaza numarul operelor de tipul respectiv din muzeu, numarul de specialisti care au expertizat sau au restaurat aceste opere, numarul de expozitii în care au fost expuse, precum si valoarea nominala totala a acestora.

```
CREATE OR REPLACE PROCEDURE date_tip_opera
(v_tip opera.tip%TYPE) AS FUNCTION nr_opere (v_tip
opera.tip%TYPE) RETURN NUMBER IS
v_numar  NUMBER(3);
BEGIN
SELECT COUNT(*)
INTO   v_numar
FROM   opera
```

```
     WHERE  tip = v_tip; RETURN v_numar;
     END nr_opere;
     FUNCTION valoare_totala (v_tip opera.tip%TYPE) RETURN NUMBER IS
     v_numar  opera.valoare%TYPE; BEGIN
     SELECT SUM(valoare) INTO v_numar
     FROM   opera
     WHERE  tip = v_tip; RETURN v_numar;
     END valoare_totala;
     FUNCTION nr_specialisti (v_tip opera.tip%TYPE) RETURN NUMBER IS
     v_numar  NUMBER(3); BEGIN
     SELECT COUNT(DISTINCT studiaza.cod_specialist) INTO   v_numar
     FROM   studiaza, opera
     WHERE  studiaza.cod_opera = opera.cod_opera AND  opera.tip =
v_tip;
     RETURN v_numar; END nr_specialisti;
     FUNCTION nr_expozitii (v_tip opera.tip%TYPE) RETURN NUMBER IS
     v_numar  NUMBER(3); BEGIN
     SELECT COUNT(DISTINCT figureaza_in.cod_expozitie) INTOv_numar
     FROM   figureaza_in, opera
     WHERE  figureaza_in.cod_opera = opera.cod_opera
     AND    opera.tip = v_tip; RETURN v_numar;
     END nr_expozitii; BEGIN
     DBMS_OUTPUT.PUT_LINE('Numarul operelor de arta este '||
     nr_opere(v_tip));
     DBMS_OUTPUT.PUT_LINE('Valoarea operelor de arta este '||
     valoare_totala(v_tip));
     DBMS_OUTPUT.PUT_LINE('Numarul de specialisti este '||
     nr_specialisti(v_tip));
     DBMS_OUTPUT.PUT_LINE('Numarul de expozitii este '||
     nr_expozitii(v_tip);
     END date_tip_opera;
```

Instructiunea *CALL*

O instructiune specifica pentru *Oracle9i* este *CALL*, care permite apelarea subprogramelor *PL/SQL*

stocate (independente sau incluse în pachete) si a metodelor *Java*.

CALL este o comanda *SQL* care nu poate sa apara de sine statatoare într-un bloc *PL/SQL*. Ea poate fi utilizata în *PL/SQL* doar dinamic, prin intermediul comenzii *EXECUTE IMMEDIATE*. Pentru executarea acestei comenzi, utilizatorul trebuie sa aiba privilegiul *EXECUTE* asupra subprogramului. Instructiunea poate fi executata interactiv din *SQL*Plus*.

Comanda *CALL* are sintaxa urmatoare:

CALL [*schema.*] [{*nume_tip_obiect* | *nume_pachet*}.] *nume_subprogram*

[(*lista_parametri actuali*)] [***@****dblink_nume*] [***INTO*** :*variabila_host*]

Identificatorul *nume_subprogram* este numele unui subprogram sau al unei metode apelate. Clauza *INTO* este folosita numai pentru variabilele de iesire ale unei functii. Daca lipseste clauza *@dblink_nume*, atunci apelul se refera la baza de date locala, iar într-un sistem distribuit clauza specifica numele bazei de date ce contine subprogramul.

Exemplu:

Sunt prezentate doua exemple prin care o functie *PL/SQL* este apelata din *SQL*Plus*, respectiv o procedura externa *C* este apelata, folosind *SQL* dinamic, dintr-un bloc *PL/SQL*.

```
CREATE OR REPLACE FUNCTION apelfunctie(a IN VARCHAR2) RETURN
VARCHAR2 AS
BEGIN
DBMS_OUTPUT.PUT_LINE ('Apel functie cu ' || a); RETURN a;
END apelfunctie;
/
SQL> --apel valid
SQL> VARIABLE v_iesire VARCHAR2(20)
SQL> CALL apelfunctie('Salut!') INTO :v_iesire Apel functie cu
Salut!
Call completed
SQL> PRINT v_iesire v iesire
Salut!
DECLARE
a NUMBER(7);
x VARCHAR2(10);
BEGIN
EXECUTE IMMEDIATE 'CALL alfa_extern_procedura (:aa, :xx)'
USING a, x;
END;
/
```

Modificarea si suprimarea subprogramelor *PL/SQL*

Pentru a lua în considerare modificarea unei proceduri sau functii, recompilarea acesteia se face prin comanda:

ALTER {***FUNCTION*** | ***PROCEDURE***} [*schema.*]*nume* ***COMPILE***;

Ca si în cazul tabelelor, functiile si procedurile pot fi suprimate cu ajutorul comenzii *DROP*. Aceasta presupune eliminarea subprogramelor din dictionarul datelor. *DROP* este o comanda ce apartine limbajului de definire a datelor, astfel ca se executa un *COMMIT* implicit atât înainte, cât si dupa comanda.

Atunci când este sters un subprogram prin comanda *DROP*, automat sunt revocate toate privilegiile acordate referitor la acest subprogram. Daca este utilizata sintaxa *CREATE OR REPLACE*, privilegiile acordate acestui subprogram ramân aceleasi.

Comanda *DROP* are urmatoarea sintaxa:

DROP {***FUNCTION*** | ***PROCEDURE***} [*schema.*]*nume*;

Transferarea valorilor prin parametri

Lista parametrilor unui subprogram este compusa din parametri de intrare (*IN*), de iesire (*OUT*) sau de intrare/iesire (*IN OUT*), separati prin virgula.

Daca nu este specificat tipul parametrului, atunci implicit acesta este considerat de intrare (*IN*). Un parametru formal cu optiunea *IN* poate primi valori implicite chiar în cadrul comenzii de declarare. Acest parametru este *read-only* si deci, nu poate fi schimbat în corpul subprogramului. El actioneaza ca o constanta. Parametrul actual corespunzator poate fi literal, expresie, constanta sau variabila initializata.

Un parametru formal cu optiunea *OUT* este neinitializat si prin urmare, are automat valoarea *null*. În interiorul subprogramului, parametrilor cu optiunea *OUT* sau *IN OUT* trebuie sa li se asigneze o valoare explicita. Daca nu se atribuie nici o valoare, atunci parametrul actual corespunzator va avea valoarea *null*. Parametrul actual trebuie sa fie o variabila, nu poate fi o constanta sau o expresie.

Daca în timpul executiei procedurii apare o exceptie, atunci valorile parametrilor formali cu optiunile *IN OUT* sau *OUT* nu sunt copiate în valorile parametrilor actuali.

Implicit, transmiterea parametrilor se face prin referinta în cazul parametrilor *IN* si prin valoare în cazul parametrilor *OUT* sau *IN OUT*. Daca pentru realizarea unor performante se doreste transmiterea prin referinta si a parametrilor *IN OUT* sau *OUT*, atunci se poate utiliza optiunea *NOCOPY*. Daca optiunea *NOCOPY* este asociata unui parametru *IN*, atunci se va genera o eroare la compilare, deoarece acesti parametri se transmit de fiecare data prin referinta.

Optiunea *NOCOPY* va fi ignorata, iar parametrul va fi transmis prin valoare daca:

- parametrul actual este o componenta a unui tablou indexat (restrictia nu se aplica daca parametrul este întreg tabelul);

- parametrul actual este constrâns prin specificarea unei precizii, a unei marimi sau prin optiunea NOT NULL;
- parametrul formal si cel actual asociat sunt înregistrari care fie au fost declarate implicit ca variabile contor într-un ciclu LOOP, fie au fost declarate explicit prin %ROWTYPE, dar constrângerile pe câmpurile corespunzatoare difera;
- transmiterea parametrului actual cere o conversie implicita a tipului;
- subprogramul este o parte a unui apel de tip RPC (remote procedure call), iar în acest caz, parametrii fiind transmisi în retea, transferul nu se poate face prin referinta.

Atunci când este apelata o procedura *PL/SQL*, sistemul *Oracle* furnizeaza doua metode pentru definirea parametrilor actuali: specificarea explicita prin nume si specificarea prin pozitie.

***Exemplu*:**

Sunt prezentate diferite moduri pentru apelarea procedurii *p1*.

```
CREATE PROCEDURE p1(a IN NUMBER, b IN VARCHAR2,
c IN DATE, d OUT NUMBER) AS …;
DECLARE
var_aNUMBER; var_b  VARCHAR2; var_c     DATE; var_d    NUMBER;
BEGIN
--specificare prin pozitie p1(var_a,var_b,var_c,var_d);
--specificare prin nume
p1(b=>var_b,c=>var_c,d=>var_d,a=>var_a);
--specificare prin nume si pozitie
p1(var_a,var_b,d=>var_d,c=>var_c);
END;
```

Daca este utilizata specificatia prin pozitie, parametrii care au primit o valoare implicita trebuie sa fie plasati la sfârsitul listei parametrilor actuali.

***Exemplu*:**

Fie *proces_data* o procedura care proceseaza în mod normal data zilei curente, dar care optional poate procesa si alte date. Daca nu se specifica parametrul actual corespunzator parametrului formal *plan_data*, atunci acesta va lua automat valoarea data implicit (data curenta a sistemului).

```
PROCEDURE proces_data(data_in IN NUMBER,
plan_data                     IN DATE := SYSDATE) IS …
```

Urmatoarele comenzi reprezinta apeluri corecte ale procedurii *proces_data*:

```
proces_data(10); proces_data(10,SYSDATE+1);
proces_data(plan_data=>SYSDATE+1,data_in=>10);
```

O declaratie de subprogram (procedura sau functie) fara parametri este specificata fara paranteze.

De exemplu, daca procedura *react_calc_dur* si functia *obt_date* nu au parametri, atunci:

```
react_calc_dur;
-- apel corect react_calc_dur();
-- apel incorect data_mea := obt_date;
-- apel corect
```

Module *overload*

Doua sau mai multe module pot sa aiba aceleasi nume, dar sa difere prin lista parametrilor. Aceste module sunt numite module *overload* (supraîncarcate). Functia *TO_CHAR* este un exemplu de modul *overload*. Exista o singura functie, *TO_CHAR*, pentru a converti date numerice si calendaristice în date de tip caracter.

În cazul unui apel, compilatorul compara parametrii actuali cu listele parametrilor formali pentru modulele *overload* si executa modulul corespunzator. Toate programele *overload* trebuie sa fie definite în acelasi bloc *PL/SQL* (bloc anonim, modul sau pachet).

Modulele *overload* pot sa apara în programele *PL/SQL* fie în sectiunea declarativa a unui bloc, fie în interiorul unui pachet. Supraîncarcarea subprogramelor nu se poate face pentru functii sau proceduri stocate, dar este permisa pentru subprograme locale, subprograme care apar în pachete sau pentru metode.

Observatii:

- Doua subprograme overload trebuie sa difere, cel putin prin tipul unuia dintre parametri. Doua subprograme nu pot fi overload daca parametrii lor formali difera numai prin tipurile lor si daca acestea sunt niste subtipuri care se bazeaza pe acelasi tip de date.
- Nu este suficient ca lista parametrilor subprogramelor overload sa difere numai prin numele parametrilor formali.
- Nu este suficient ca lista parametrilor subprogramelor overload sa difere numai prin tipul acestora (IN, OUT, IN OUT). PL/SQL nu poate face diferenta (la apelare) între tipurile IN si OUT.
- Nu este suficient ca functiile overload sa difere doar prin tipul de date returnat (tipul de date specificat în clauza RETURN a functiei).

Exemplu:

Urmatoarele subprograme nu pot fi *overload*.

1) `FUNCTION alfa(par IN POSITIVE)…; FUNCTION alfa(par IN BINARY_INTEGER)…;`

2) `FUNCTION alfa(par IN NUMBER)…; FUNCTION alfa(parar IN NUMBER)…;`

3) `PROCEDURE beta(par IN VARCHAR2) IS…; PROCEDURE beta(par OUT`

```
VARCHAR2) IS…;
```

***Exemplu*:**

Sa se creeze doua functii (locale) cu acelasi nume care sa calculeze media valorilor operelor de arta de un anumit tip. Prima functie va avea un argument reprezentând tipul operelor de arta, iar cea de-a doua va avea doua argumente, unul reprezentând tipul operelor de arta, iar celalalt reprezentând stilul operelor pentru care se calculeaza valoarea medie (functia va calcula media valorilor operelor de arta de un anumit tip si care apartin unui stil specificat).

```
DECLARE
medie1 NUMBER(10,2); medie2 NUMBER(10,2);
FUNCTION valoare_medie (v_tip opera.tip%TYPE)
RETURN NUMBER IS medie NUMBER(10,2);
BEGIN
SELECT AVG(valoare) INTO medie
FROM   opera
WHERE  tip = v_tip; RETURN medie;
END;
FUNCTION valoare_medie (v_tip  opera.tip%TYPE,
                   v_stil    opera.stil%TYPE)
RETURN NUMBER IS medie NUMBER(10,2);
BEGIN
SELECT AVG(valoare) INTO medie
FROM   opera
WHERE  tip = v_tip AND stil = v_stil; RETURN medie;
END;
BEGIN
medie1 := valoare_medie('pictura');
DBMS_OUTPUT.PUT_LINE('Media valorilor picturilor din muzeu
este ' || medie1);
medie2 := valoare_medie('pictura ', 'impresionism ');
DBMS_OUTPUT.PUT_LINE('Media valorilor picturilor
impresioniste din muzeu este ' || medie2);
END;
```

Procedura *versus* functie

Dupa cum am mai subliniat, în general, procedura este utilizata pentru realizarea unor actiuni, iar functia este folosita pentru calculul unei valori.

Pot fi marcate câteva deosebiri esentiale între functii si proceduri.

- Procedura se executa ca o comanda PL/SQL, iar functia se invoca în cadrul unei expresii.
- Procedura poate returna (sau nu) una sau mai multe valori, iar functia trebuie sa returneze o singura valoare.
- Procedura nu trebuie sa contina clauza RETURN expresie, iar functia trebuie sa contina aceasta optiune.

De asemenea, pot fi remarcate câteva elemente esentiale, comune atât functiilor cât si procedurilor.

Ambele pot:

- accepta valori implicite;
- contine sectiuni declarative, executabile si de tratare a erorilor;
- utiliza specificarea prin nume sau pozitie a parametrilor;
- accepta parametri NOCOPY.

Tratarea exceptiilor

Daca apare o eroare într-un subprogram, atunci este declansata o exceptie. Daca exceptia este tratata în subprogram, atunci blocul se termina si controlul trece la sectiunea de tratare a erorilor din subprogramul respectiv. Daca nu exista o tratare a exceptiei în subprogram, atunci controlul trece la programul apelant de nivel imediat superior (în partea de tratare a erorilor), respectând regulile de propagare a exceptiilor. În acest caz, valorile parametrilor de tip *OUT* sau *IN OUT* nu sunt returnate parametrilor actuali. Acestia vor avea aceleasi valori, ca si cum subprogramul nu ar fi fost apelat.

Recursivitate

Recursivitatea este o tehnica importanta pentru simplificarea modelarii algoritmilor. Un subprogram recursiv presupune ca acesta se apeleaza pe el însusi.

În *Oracle*, o problema delicata este legata de locul unde se plaseaza un apel recursiv. De exemplu, daca apelul este în interiorul unei comenzi *FOR* specifice cursoarelor sau între comenzile *OPEN* si *CLOSE*, atunci la fiecare apel este deschis alt cursor. În felul acesta, programul poate depasi limita admisa de cursoare deschise la un moment dat (*OPEN_CURSORS*), setata în parametrul de initializare *Oracle*.

***Exemplu*:**

Sa se calculeze recursiv al *m*-lea termen din sirul lui Fibonacci.

```
CREATE OR REPLACE FUNCTION fibo(m POSITIVE)
RETURN INTEGER AS BEGIN
IF (m = 1) OR (m = 2) THEN RETURN 1;
ELSE
```

```
RETURN fibo(m-1) + fibo(m-2); END IF;
END fibona;
```

***Exemplu*:**

Sa se calculeze iterativ al *m*-lea termen din sirul lui Fibonacci.

```
CREATE OR REPLACE FUNCTION fibo(m POSITIVE)
RETURN INTEGER AS ter1   INTEGER := 1;
ter2 INTEGER := 0;
valoare INTEGER; BEGIN
IF (m = 1) OR (m = 2) THEN
RETURN 1; ELSE
valoare := ter1 + ter2;
FOR i IN 3..m  LOOP
ter2 := ter1; ter1 := valoare;
valoare := ter1 + ter2; END LOOP;
RETURN valoare;
END IF;
END fibo;
```

Declaratii *forward*

Subprogramele se numesc reciproc recursive daca se apeleaza unul pe altul, în mod direct sau indirect.

Declaratiile *forward* permit definirea subprogramelor reciproc recursive.

În *PL/SQL*, un identificator trebuie declarat înainte de a fi folosit. De asemenea, un subprogram trebuie declarat înainte de a fi apelat.

***Exemplu*:**

```
PROCEDURE alfa ( ... ) IS BEGIN
beta( ... );       -- apel incorect
… END;
PROCEDURE beta ( ... ) IS BEGIN
… END;
```

În acest exemplu, procedura *beta* nu poate fi apelata deoarece nu este înca declarata. Problema se poate rezolva simplu în acest caz, inversând ordinea celor doua proceduri. Aceasta solutie nu este eficienta întotdeauna (de exemplu, daca si procedura *beta* contine un apel al procedurii *alfa*).

PL/SQL permite un tip special, numit *forward*, de declarare a unui subprogram. El consta dintr-o specificare a antetului unui subprogram, terminata prin caracterul „;“. O declaratie de tip *forward* pentru procedura *beta* are forma:

```
PROCEDURE beta ( ... );  -- declaratie forward
…
PROCEDURE alfa ( ... ) IS BEGIN
beta( ... );
… END;
PROCEDURE beta ( ... ) IS BEGIN
… END;
```

Declaratiile *forward* pot fi folosite pentru a defini subprograme într-o anumita ordine logica, pentru a defini subprograme reciproc recursive sau pentru a grupa subprograme într-un pachet.

Lista parametrilor formali din declaratia *forward* trebuie sa fie identica cu cea corespunzatoare corpului subprogramului. Corpul subprogramului poate aparea oriunde dupa declaratia sa *forward*, dar sa ramâna în aceeasi unitate de program.

Utilizarea în expresii *SQL* a functiilor definite de utilizator

O functie stocata poate fi referita într-o comanda *SQL* la fel ca orice functie standard furnizata de sistem (*built-in function*), dar cu anumite restrictii.

Functiile *PL/SQL* definite de utilizator pot fi apelate din orice expresie *SQL* în care se pot folosi functii *SQL* standard.

Functiile *PL/SQL* pot sa apara în:

- lista de câmpuri a comenzii *SELECT*;
- conditia clauzelor *WHERE* si *HAVING*;
- clauzele *CONNECT BY*, *START WITH*, *ORDER BY* si *GROUP BY*;
- clauza *VALUES* a comenzii *INSERT*;
- clauza *SET* a comenzii *UPDATE*.

***Exemplu*:**

Sa se afiseze operele de arta (titlu, valoare, stare) a caror valoare este mai mare decât valoarea medie a tuturor operelor de arta din muzeu.

```
CREATE OR REPLACE FUNCTION valoare_medie RETURN NUMBER AS
v_val_mediu opera.valoare%TYPE; BEGIN
SELECT AVG(valoare) INTO v_val_mediu FROM    opera;
RETURN v_val_mediu;
END;
```

Referirea acestei functii într-o comanda *SQL* se poate face prin secventa:

```
SELECT titlu, valoare, stare FROM  opera
WHERE     valoare >= valoare_medie;
```

Exista restrictii referitoare la folosirea functiilor definite de utilizator într-o comanda *SQL*. Câteva dintre acestea, care s-au pastrat si pentru *Oracle9i*, vor fi enumerate în continuare:

- functia definita de utilizator trebuie sa fie o functie stocata (procedurile stocate nu pot fi apelate în expresii SQL), nu poate fi locala altui bloc;
- functia definita de utilizator trebuie sa fie o functie linie si nu una grup (restrictia dispare în Oracle9i);
- functia apelata dintr-o comanda SELECT sau din comenzi paralelizate INSERT, UPDATE si DELETE nu poate modifica tabelele bazei de date;
- functia apelata dintr-o comanda UPDATE sau DELETE nu poate interoga sau modifica tabele ale bazei reactualizate chiar de aceste comenzi (table mutating);
- functia apelata din comenzile SELECT, INSERT, UPDATE sau DELETE nu poate contine comenzi LCD (COMMIT), ALTER SYSTEM, SET ROLE sau comenzi LDD (CREATE);
- functia nu poate aparea în clauza CHECK a unei comenzi CREATE/ALTER TABLE;
- functia nu poate fi folosita pentru a specifica o valoare implicita pentru o coloana în cadrul unei comenzi CREATE/ALTER TABLE;
- functia poate fi utilizata într-o comanda SQL numai de catre proprietarul functiei sau de utilizatorul care are privilegiul EXECUTE asupra acesteia;
- parametrii unei functii PL/SQL apelate dintr-o comanda SQL trebuie sa fie specificati prin pozitie (specificarea prin nume nefiind permisa);
- functia definita de utilizator, apelabila dintr-o comanda SQL, trebuie sa aiba doar parametri de tip IN, cei de tip OUT si IN OUT nefiind acceptati;
- parametrii formali ai unui subprogram functie trebuie sa fie de tip specific bazei de date (NUMBER, CHAR, VARCHAR2, ROWID, LONG, LONGROW, DATE etc.) si nu de tipuri PL/SQL (BOOLEAN, RECORD etc.);
- tipul returnat de un subprogram functie trebuie sa fie un tip intern pentru server, nu un tip PL/SQL;
- functia nu poate apela un subprogram care nu respecta restrictiile anterioare.

***Exemplu*:**

```
CREATE OR REPLACE FUNCTION calcul (p_val NUMBER)
RETURN NUMBER IS
BEGIN
INSERT INTO opera(cod_opera, tip, data_achizitie, valoare)
VALUES (1358, 'gravura', SYSDATE, 700000);
RETURN (p_val*7); END;
/
```

```
UPDATE    opera
SET valoare = calcul (550000) WHERE    cod_opera = 7531;
```

Comanda *UPDATE* va returna o eroare deoarece tabelul *opera* este *mutating.* Operatia de reactualizare este însa permisa asupra oricarui alt tabel diferit de *opera*.

Informatii referitoare la subprograme

Informatiile referitoare la subprogramele *PL/SQL* si modul de acces la aceste informatii sunt urmatoarele:

- codul sursa, utilizând vizualizarea USER_SOURCE din dictionarul datelor;
- informatii generale, utilizând vizualizarea USER_OBJECTS din dictionarul datelor;
- tipul parametrilor (IN, OUT, IN OUT), utilizând comanda DESCRIBE din SQL*Plus;
- p-code (nu este accesibil utilizatorilor);
- erorile la compilare, utilizând vizualizarea USER_ERRORS din dictionarul datelor sau comanda SHOW ERRORS;
- informatii de depanare, utilizând pachetul DBMS_OUTPUT.

Atunci când este creat un subprogram stocat, informatiile referitoare la subprogram sunt depuse în dictionarul datelor.

Vizualizarea *USER_OBJECTS* contine informatii generale despre toate obiectele prelucrate în baza de date si, în particular, despre subprogramele stocate.

Vizualizarea *USER_OBJECTS* are urmatoarele câmpuri:

- OBJECT_NAME – numele obiectului;
- OBJECT_TYPE – tipul obiectului (PROCEDURE, FUNCTION etc.);
- OBJECT_ID – identificator intern al obiectului;
- CREATED – data la care a fost creat obiectul;
- LAST_DDL_TIME – data ultimei modificari a obiectului;
- TIMESTAMP – data si momentul ultimei recompilari;
- STATUS – starea de validitate a obiectului.

Pentru a verifica daca recompilarea explicita (*ALTER*) sau implicita a avut succes se poate verifica starea subprogramelor utilizând coloana *STATUS* din vizualizarea *USER_OBJECTS*.

Orice obiect are o stare (*status*) sesizata în dictionarul datelor, care poate fi:

- *VALID* (obiectul a fost compilat si poate fi folosit atunci când este referit);
- *INVALID* (obiectul trebuie compilat înainte de a fi folosit).

***Exemplu*:**

Sa se listeze în ordine alfabetica, procedurile si functiile detinute de utilizatorul curent, precum si starea acestora.

```
SELECTOBJECT_NAME, OBJECT_TYPE, STATUS FROM USER_OBJECTS
WHERE OBJECT_TYPE IN ('PROCEDURE','FUNCTION')
ORDER BY  OBJECT_NAME;
```

Dupa ce subprogramul a fost creat, codul sursa al acestuia poate fi obtinut consultând vizualizarea *USER_SOURCE* din dictionarul datelor. Vizualizarea are urmatoarele câmpuri: *NAME* (numele obiectului), *TYPE* (tipul obiectului), *LINE* (numarul liniei din codul sursa), *TEXT* (textul liniilor codului sursa).

***Exemplu*:**

Sa se afiseze codul complet pentru functia *numar_opere*.

```
SELECTTEXT
FROM  USER_SOURCE
WHERE NAME = 'NUMAR_OPERE' ORDER BY    LINE;
```

***Exemplu*:**

Sa se scrie o procedura care recompileaza toate obiectele invalide din schema personala.

```
CREATE OR REPLACE PROCEDURE recompileaza IS CURSOR obj_curs IS
SELECT   OBJECT_TYPE, OBJECT_NAME FROM USER_OBJECTS
WHERE    STATUS = 'INVALID'
AND      OBJECT_TYPE IN
('PROCEDURE', 'FUNCTION', PACKAGE', 'PACKAGE BODY', 'VIEW');
BEGIN
FOR obj_rec IN obj_curs LOOP
DBMS_DDL.ALTER_COMPILE(obj_rec.OBJECT_TYPE,
USER, obj_rec.OBJECT_NAME);
END LOOP;
END recompileaza;
```

Atunci când se recompileaza un obiect *PL/SQL*, *server*-ul va recompila orice obiect invalid de care depinde acesta.

Daca la recompilarea automata implicita a procedurilor locale dependente apar probleme, atunci starea obiectului va ramâne *INVALID* si *server*-ul *Oracle* va semnala o eroare. Prin urmare:

- este preferabil ca recompilarea sa fie manuala, explicita utilizând comanda ALTER (PROCEDURE, FUNCTION, TRIGGER, PACKAGE) cu optiunea COMPILE;
- este necesar ca dupa o schimbare referitoare la obiectele bazei, recompilarea sa se faca cât mai repede.

Vizualizarea *USER_ERRORS* afiseaza textul tuturor erorilor de compilare. Câmpurile acesteia (*NAME*, *TYPE*, *SEQUENCE*, *LINE*, *POSITION*, *TEXT*) sunt analizate în capitolul referitor la tratarea

exceptiilor.

Pentru a obtine valori (de exemplu, valoarea contorului pentru un *LOOP*, valoarea unei variabile înainte si dupa o atribuire etc.) si mesaje (de exemplu, parasirea unui subprogram, aparitia unei operatii etc.) dintr-un bloc *PL/SQL* pot fi utilizate procedurile pachetului *DBMS_OUTPUT*. Aceste informatii se cumuleaza într-un *buffer* care poate fi consultat ulterior.

Dependenta subprogramelor

Atunci când este compilat un subprogram, toate obiectele *Oracle* care sunt referite vor fi înregistrate în dictionarul datelor. Subprogramul este dependent de aceste obiecte. Un subprogram care are erori la compilare este marcat ca *INVALID* în dictionarul datelor. Un subprogram stocat poate deveni, de asemenea, invalid dupa executia unei operatii *LDD* asupra unui obiect de care depinde.

Obiecte dependente	Obiecte referite
View	*Table*
Procedure	*View*
Function	*Procedure*
Package Specification	*Function*
Package Body	*Synonym*
Database Trigger	*Package Specification*

Modificarea definitiei unui obiect referit poate sa influenteze (sau nu) functionarea normala a obiectului dependent.

Exista doua tipuri de dependente:

- dependenta directa, în care obiectul dependent (procedure sau function) face referinta direct la un obiect de tip table, view, sequence, procedure, function;
- dependenta indirecta, în care obiectul dependent (procedure sau function) face referinta indirect la un obiect de tip table, view, sequence, procedure, function prin intermediul unui view, procedure sau function.

În cazul dependentelor locale, atunci când un obiect referit este modificat, obiectele dependente sunt invalidate. La urmatorul apel al obiectului invalidat, acesta va fi recompilat automat de catre *server*-ul *Oracle*.

În cazul dependentelor la distanta, procedurile stocate local si toate obiectele dependente vor fi invalidate. Ele nu vor fi recompilate automat la urmatorul apel.

***Exemplu*:**

Se presupune ca procedura *filtru* va referi direct tabelul *opera* si ca procedura *adaug* va reactualiza tabelul *opera* prin intermediul unei vizualizari *nou_opera*. Pentru aflarea dependentelor directe se poate utiliza vizualizarea *USER_DEPENDENCIES* din dictionarul datelor.

```
SELECT NAME, TYPE, REFENCED_NAME, REFENCED_TYPE FROM
USER_DEPENDENCIES
WHERE     REFENCED_NAME IN ('opera', 'nou_opera');
```

NAME	TYPE	REFENCED NAME	REFENCED TYPE
filtru	Procedure	opera	Table
adaug	Procedure	nou_opera	View
nou_opera	View	opera	Table

Dependentele indirecte pot fi afisate utilizând vizualizarile *DEPTREE* si *IDEPTREE*. Vizualizarea *DEPTREE* afiseaza o reprezentare a tuturor obiectelor dependente (direct sau indirect). Vizualizarea *IDEPTREE* afiseaza o reprezentare a aceleasi informatii, sub forma unui arbore.

Pentru a utiliza aceste vizualizari furnizate de sistemul *Oracle* trebuie:

- executat scriptul *UTLDTREE*;
- executata procedura *DEPTREE_FILL* (are trei argumente: tipul obiectului referit, schema obiectului referit, numele obiectului referit).

***Exemplu*:**

```
@UTLDTREE
EXECUTE DEPTREE_FILL ('TABLE', 'SCOTT', 'opera')
SELECT NESTED_LEVEL, TYPE, NAME
FROM DEPTREE ORDER  BY SEQ#;
```

NESTED LEVEL	TYPE	NAME
0	Table	opera
1	View	nou_opera
2	Procedure	adaug
1	Procedre	filtru

```
SELECT    *
FROMIDEPTREE;
DEPENDENCIES
TABLE nume_schema.opera
VIEW nume_schema.nou_opera PROCEDURE nume_schema.adaug
PROCEDURE nume_schema.filtru
```

Dependentele la distanta sunt tratate printr-una dintre modalitatile alese de utilizator, modelul *timestamp* (implicit) sau modelul *signature.*

Fiecare unitate *PL/SQL* are un *timestamp* (eticheta de timp) care este setat atunci când unitatea este creata sau recompilata si care este depus în câmpul *LAST_DDL_TIME* din dictionarul datelor.

Modelul *timestamp* realizeaza compararea momentelor ultimei modificari a celor doua obiecte analizate. Daca obiectul bazei are momentul ultimei modificari mai recent decât cel al obiectului dependent, atunci obiectul dependent va fi recompilat.

Modelul *signature* (semnatura) determina momentul la care obiectele bazei distante trebuie recompilate. Când este creat un subprogram, o *signature* este depusa în dictionarul datelor, alaturi de *p-code*. Aceasta contine: numele constructiei *PL/SQL* (*PROCEDURE, FUNCTION, PACKAGE*), tipurile parametrilor, ordinea parametrilor, numarul acestora si modul de transmitere (*IN, OUT, IN OUT*). Daca parametrii se schimba, atunci evident *signature* se schimba.

Pentru a folosi modelul *signature* este necesara setarea parametrului

REMOTE_DEPENDENCIES_MODE la *SIGNATURE*. Aceasta se poate realiza prin:

1) comanda *ALTER SESSION*, care va afecta doar sesiunea curenta:

ALTER SESSION SET REMOTE_DEPENDENCIES_MODE = SIGNATURE;

2) comanda *ALTER SYSTEM*, care va afecta întreaga baza de date (toate sesiunile), dar trebuie avut privilegiul sistem pentru a utiliza aceasta instructiune:

ALTER SYSTEM SET REMOTE_DEPENDENCIES_MODE = SIGNATURE;

3) adaugarea în fisierul de initializare a unei linii de forma:

REMOTE_DEPENDENCIES_MODE = SIGNATURE;

Recompilarea procedurilor si a functiilor dependente este fara succes daca:

- obiectul referit este suprimat (DROP) sau redenumit (RENAME);
- tipul coloanei referite este schimbat;
- o vizualizare referita este înlocuita printr-o vizualizare ce contine alte coloane;
- lista parametrilor unei proceduri referite este modificata.

Recompilarea procedurilor si functiilor dependente este cu succes daca:

- tabelul referit are noi coloane;
- corpul PL/SQL al unei proceduri referite a fost modificat si recompilat cu succes.

Erorile datorate dependentelor pot fi minimizate:

- utilizând comenzi SELECT cu optiunea „*“;
- incluzând lista coloanelor în comanda INSERT;
- declarând variabile cu atributul %TYPE;
- declarând înregistrari cu atributul %ROWTYPE;

Concluzii:

- Daca procedura depinde de un obiect local, atunci se face recompilare automata la prima reexecutie.
- Daca procedura depinde de o procedura distanta, atunci se face recompilare automata, dar la a doua reexecutie. Este preferabila o recompilare manuala pentru prima reexecutie sau

implementarea unei strategii de reinvocare a ei (a doua oara).

- Daca procedura depinde de un obiect distant, dar care nu este procedura, atunci nu se face recompilare automata.

Rutine externe

PL/SQL a fost special conceput pentru *Oracle* si este specializat pentru procesarea tranzactiilor *SQL*.

Totusi, într-o aplicatie complexa pot sa apara cerinte si functionalitati care sunt mai eficient de implementat în *C, Java* sau alt limbaj de programare. De exemplu, *Java* este un limbaj portabil cu un model de securitate bine definit, care lucreaza excelent pentru aplicatii *Internet*.

Daca aplicatia trebuie sa efectueze anumite actiuni care nu pot fi implementate optim utilizând *PL/SQL*, atunci este preferabil sa fie utilizate alte limbaje care realizeaza performant actiunile respective. În acest caz este necesara comunicarea între diferite module ale aplicatiei care sunt scrise în limbaje diferite.

Pâna la versiunea *Oracle8*, modalitatea de comunicare între *PL/SQL* si alte limbaje (de exemplu, limbajul

C) a fost utilizarea pachetelor *DBMS_PIPE* si/sau *DBMS_ALERT*.

Începând cu *Oracle8*, comunicarea este simplificata prin utilizarea rutinelor externe. O rutina externa este o procedura sau o functie scrisa într-un limbaj diferit de *PL/SQL*, dar apelabila dintr-un program scris în *PL/SQL*. *PL/SQL* extinde functionalitatea *server*-ului *Oracle*, furnizând o interfata pentru apelarea rutinelor externe. Orice bloc *PL/SQL* executat pe *server* sau pe *client* poate apela o rutina externa. Singurul limbaj acceptat pentru rutine externe în *Oracle8* este limbajul *C*.

Pentru a marca apelarea unei rutine externe în programul *PL/SQL* este definit un punct de intrare (*wrapper*) care directioneaza spre codul extern (program *PL/SQL* ? *wrapper* ? cod extern). Pentru crearea unui *wrapper* este utilizata o clauza speciala (*AS EXTERNAL*) în cadrul comenzii *CREATE OR REPLACE PROCEDURE*. De fapt, clauza contine informatii referitoare la numele bibliotecii în care se gaseste subprogramul extern (clauza *LIBRARY*), numele rutinei externe (clauza *NAME*) si corespondenta (*C <–> PL/SQL*) dintre tipurile de date (clauza *PARAMETERS*). În ultimele versiuni s-a renuntat la clauza *AS EXTERNAL*.

Rutinele externe (scrise în *C*) sunt compilate, apoi depuse într-o biblioteca dinamica (*DLL – dynamic link library*) si sunt încarcate doar atunci când este necesar. Daca se invoca o rutina externa scrisa în *C*, trebuie setata conexiunea spre aceasta rutina. Un proces numit *extproc* este declansat automat de catre *server*. La rândul sau, procesul *extproc* va încarca biblioteca identificata prin clauza *LIBRARY* si va apela rutina respectiva.

Oracle8i permite utilizarea de rutine externe scrise în *Java*. De asemenea, un *wrapper* poate include specificatii de apelare, prin utilizarea clauzei *LANGUAGE*. De fapt, aceste specificatii permit

apelarea rutinelor externe scrise în orice limbaj. De exemplu, o procedura scrisa într-un limbaj diferit de *C* sau *Java* poate fi utilizata în *SQL* sau *PL/SQL* daca procedura respectiva este apelabila din *C*. În felul acesta, biblioteci standard scrise în alte limbaje de programare pot fi apelate din programe *PL/SQL*.

Procedura *PL/SQL* executata pe *server*-ul *Oracle* poate apela o rutina externa scrisa în limbajul *C*, depusa într-o biblioteca partajata. Procedura *C* se executa într-un spatiu adresa diferit de cel al *server*-ului *Oracle*, în timp ce unitatile *PL/SQL* si metodele *Java* se executa în spatiul adresa al *server*-ului. *JVM* (*Java Virtual Machine*) de pe *server* va executa metoda *Java* în mod direct, fara a fi necesar procesul *extproc*.

Maniera de încarcare depinde de limbajul în care este scrisa rutina.

- Pentru a apela rutine externe C, server-ul trebuie sa cunoasca pozitionarea bibliotecii dinamice DLL. Acest lucru este furnizat de alias-ul bibliotecii din clauza AS LANGUAGE.
- Pentru apelarea unei rutine externe Java se va încarca clasa Java în baza de date. Este necesara doar crearea unui wrapper care directioneaza catre codul extern. Spre deosebire de rutinele externe C, nu este necesara nici biblioteca si nici setarea conexiunii spre rutina externa.

Clauza *LANGUAGE* din cadrul comenzii de creare a unui subprogram, specifica limbajul în care este scrisa rutina (procedura externa *C* sau metoda *Java*) si are urmatoarea forma:

{***IS*** | ***AS***} ***LANGUAGE*** {***C*** | ***JAVA***}

Pentru o procedura *C* sunt date informatii referitoare la numele acesteia (clauza *NAME*); *alias*-ul bibliotecii în care se gaseste (clauza *LIBRARY*); optiuni referitoare la tipul, pozitia, lungimea, modul de transmitere (prin valoare sau prin referinta) al parametrilor (clauza *PARAMETERS*); posibilitatea ca rutina externa sa acceseze informatii despre parametri, exceptii, alocarea memoriei utilizator (clauza *WITH CONTEXT*).

LIBRARY *nume_biblioteca* [***NAME*** *nume_proc_c*] [***WITH CONTEXT***] [***PARAMETERS*** (*parametru_extern* [, *parametru_extern ...*])]

Pentru o metoda *Java*, în clauza trebuie specificata doar signatura metodei (lista tipurilor parametrilor în ordinea aparitiei).

***Exemplu*:**

```
CREATE OR REPLACE FUNCTION calc (x IN REAL) RETURN NUMBER AS
LANGUAGE C
LIBRARY biblioteca NAME "c_calc"
PARAMETERS (x BY REFERENCES);
```

O rutina externa nu este apelata direct, ci se apeleaza subprogramul *PL/SQL* care refera rutina externa. Apelarea poate sa apara în: blocuri anonime, subprograme independente sau care apartin unui pachet, metode ale unui tip obiect, declansatori baza de date, comenzi *SQL* care apeleaza functii (în acest caz, trebuie utilizata clauza *PRAGMA RESTRICT_REFERENCES*).

De remarcat ca o metoda *Java* poate fi apelata din orice bloc *PL/SQL*, subprogram sau pachet. *JDBC* (*Java Database Connectivity*), care reprezinta interfata *Java* standard pentru conectare la baze de date relationale, si *SQLJ* permit apelarea de blocuri *PL/SQL* din programe *Java*. *SQLJ* face posibila incorporarea operatiilor *SQL* în codul *Java*. Standardul *SQLJ* acopera doar operatii *SQL* statice. *Oracle9i SQLJ* include extensii pentru a suporta direct *SQL* dinamic.

O alta modalitate de a încarca metode *Java* este folosirea interactiva în *SQL*Plus* a comenzii *CREATE JAVA instructiune*.

Functii tabel

O functie tabel (*table function*) returneaza drept rezultat un set de linii (de obicei, sub forma unei colectii). Aceasta functie poate fi interogata direct printr-o comanda *SQL*, ca si cum ar fi un tabel al bazei de date. În felul acesta, functia poate fi utilizata în clauza *FROM* a unei cereri.

O functie tabel conducta (*pipelined table function*) este similara unei functii tabel, dar returneaza datele iterativ, pe masura ce acestea sunt obtinute, nu toate deodata. Aceste functii sunt mai eficiente deoarece informatia este returnata imediat cum este obtinuta.

Conceptul de functie tabel conducta a fost introdus în versiunea *Oracle9i*. Utilizatorul poate sa defineasca astfel de functii. De asemenea, este posibila executia paralela a functiilor tabel (evident si a celor clasice). În acest caz, functia trebuie sa contina în declaratie optiunea *PARALLEL_ENABLE*.

Functia tabel conducta accepta orice argument pe care îl poate accepta o functie obisnuita si trebuie sa returneze o colectie (*nested table* sau *varray*). Ea este declarata specificând cuvântul cheie *PIPELINED* în comanda *CREATE OR REPLACE FUNCTION*. Functia tabel conducta trebuie sa se termine printr-o comanda *RETURN* simpla, care nu întoarce nici o valoare.

Pentru a returna un element individual al colectiei este folosita comanda *PIPE ROW*, care poate sa apara numai în corpul unei functii tabel conducta, în caz contrar generându-se o eroare. Comanda poate fi omisa daca functia tabel conducta nu returneaza nici o linie.

Dupa ce functia a fost creata, ea poate fi apelata dintr-o cerere *SQL* utilizând operatorul *TABLE*. Cererile referitoare la astfel de functii pot sa includa cursoare si referinte la cursoare, respectându-se semantica de la cursoarele clasice.

Functia tabel conducta nu poate sa apara în comenzile *INSERT, UPDATE, DELETE*. Totusi, pentru a realiza o reactualizare, poate fi creata o vizualizare relativa la functia tabel si folosit un declansator *INSTEAD OF*.

***Exemplu*:**

Sa se obtina o instanta a unui tabel ce contine informatii referitoare la denumirea zilelor saptamânii.

Problema este rezolvata în doua variante. Prima reprezinta o solutie clasica, iar a doua varianta

implementeaza problema cu ajutorul unei functii tabel conducta.

Varianta 1:

```
CREATE TYPE t_linie AS OBJECT (
idl NUMBER, sir VARCHAR2(20));
CREATE TYPE t_tabel AS TABLE OF t_linie;
CREATE OR REPLACE FUNCTION calc1 RETURN t_tabel AS
v_tabelt_tabel; BEGIN
v_tabel := t_tabel (t_linie (1, 'luni'));
FOR j IN 2..7 LOOP
v_tabel.EXTEND;
IF j = 2
THEN v_tabel(j) := t_linie (2, 'marti'); ELSIF j = 3
THEN v_tabel(j) := t_linie (3, 'miercuri'); ELSIF j = 4
THEN v_tabel(j) := t_linie (4, 'joi'); ELSIF j = 5
THEN v_tabel(j) := t_linie (5, 'vineri'); ELSIF j = 6
THEN v_tabel(j) := t_linie (6, 'sambata'); ELSIF j = 7
THEN v_tabel(j) := t_linie (7, 'duminica'); END IF;
END LOOP;
RETURN v_tabel;
END calc1;
```

Functia *calc1* poate fi invocata în clauza *FROM* a unei comenzi *SELECT*:

```
SELECT *
FROM TABLE (CAST (calc1 AS t_tabel));
```

Varianta 2:

```
CREATE OR REPLACE FUNCTION calc2 RETURN t_tabel PIPELINED AS
v_liniet_linie; BEGIN
FOR j IN 1..7 LOOP
v_linie := CASE j
WHEN 1 THEN t_linie (1, 'luni')
WHEN 2 THEN t_linie (2, 'marti')
WHEN 3 THEN t_linie (3, 'miercuri')
WHEN 4 THEN t_linie (4, 'joi')
WHEN 5 THEN t_linie (5, 'vineri')
WHEN 6 THEN t_linie (6, 'sambata')
WHEN 7 THEN t_linie (7, 'duminica') END;
PIPE ROW (v_linie); END LOOP;
```

```
RETURN;
END calc2;
```

Se observa ca tabelul este implicat doar în tipul rezultatului. Pentru apelarea functiei *calc2* este folosita sintaxa urmatoare:

```
SELECT *
FROM TABLE (calc2);
```

Functiile tabel sunt folosite frecvent pentru conversii de tipuri de date. *Oracle9i* introduce posibilitatea de a crea o functie tabel care returneaza un tip *PL/SQL* (definit într-un bloc). Functia tabel care furnizeaza (la nivel de pachet) drept rezultat un tip de date trebuie sa fie de tip conducta. Pentru apelare este utilizata sintaxa simplificata (fara *CAST*).

***Exemplu*:**

```
CREATE OR REPLACE PACKAGE exemplu IS
TYPE t_linie IS RECORD (idl NUMBER, sir VARCHAR2(20));
TYPE t_tabel IS TABLE OF t_linie;
END exemplu;
CREATE OR REPLACE FUNCTION calc3 RETURN exemplu.t_tabel
PIPELINED AS
v_linieexemplu.t_linie; BEGIN
FOR j IN 1..7 LOOP CASE j
WHEN 1 THEN v_linie.idl := 1; v_linie.sir := 'luni';
WHEN 2 THEN v_linie.idl := 2; v_linie.sir := 'marti';
WHEN 3 THEN v_linie.idl := 3; v_linie.sir := 'miercuri';
WHEN 4 THEN v_linie.idl := 4; v_linie.sir := 'joi';
WHEN 5 THEN v_linie.idl := 5; v_linie.sir := 'vineri';
WHEN 6 THEN v_linie.idl := 6; v_linie.sir := 'sambata';
WHEN 7 THEN v_linie.idl := 7; v_linie.sir := 'duminica';
END CASE;
PIPE ROW (v_linie); END LOOP;
RETURN;
END calc3;
```

Procesarea tranzactiilor autonome

Tranzactia este o unitate logica de lucru, adica o secventa de comenzi care trebuie sa se execute ca un întreg pentru a mentine consistenta bazei de date. În mod uzual, o tranzactie poate sa cuprinda mai multe blocuri, iar într-un bloc pot sa fie mai multe tranzactii.

O tranzactie autonoma este o tranzactie independenta lansata de alta tranzactie, numita tranzactie

principala. Tranzactia autonoma permite suspendarea tranzactiei principale, executarea de comenzi *SQL*, permanentizarea (*commit*) si anularea (*rollback*) acestor operatii.

Odata începuta, tranzactia autonoma este independenta, în sensul ca nu partajeaza blocari, resurse sau dependente cu tranzactia principala. În felul acesta, o aplicatie nu trebuie sa cunoasca operatiile autonome ale unei proceduri, iar procedura nu trebuie sa cunoasca nimic despre tranzactiile aplicatiei. Tranzactia autonoma are totusi toate functionalitatile unei tranzactii obisnuite (permite cereri paralele, procesari distribuite etc.).

Pentru definirea unei tranzactii autonome se utilizeaza clauza *PRAGMA AUTONOMOUS_TRANSACTION* care informeaza compilatorul *PL/SQL* ca trebuie sa marcheze o rutina ca fiind autonoma. Prin rutina se întelege: bloc anonim de cel mai înalt nivel (nu imbricat); procedura sau functie locala, independenta sau împachetata; metoda a unui tip obiect; declansator baza de date.

Codul *PRAGMA AUTONOMOUS_TRANSACTION* se specifica în partea declarativa a rutinei.

Codul *PRAGMA AUTONOMOUS_TRANSACTION* marcheaza numai rutine individuale ca fiind independente. Nu pot fi marcate toate subprogramele unui pachet sau toate metodele unui tip obiect ca autonome. Prin urmare, clauza nu poate sa apara în partea de specificatie a unui pachet.

***Observatii*:**

- Declansatorii autonomi, spre deosebire de cei clasici pot contine comenzi LCD (de exemplu, COMMIT, ROLLBACK).
- Exceptiile declansate în cadrul tranzactiilor autonome genereaza un rollback la nivel de tranzactie si nu la nivel de instructiune.
- Când se intra în sectiunea executabila a unei tranzactii autonome, tranzactia principala este suspendata.

Cu toate ca o tranzactie autonoma este initiata de alta tranzactie, ea nu este o tranzactie imbricata deoarece:

- nu partajeaza resurse cu tranzactia principala;
- nu depinde de tranzactia principala (de exemplu, daca tranzactia principala este anulata, atunci tranzactiile imbricate sunt de asemenea anulate, în timp ce tranzactia autonoma nu este anulata);
- schimbarile permanentizate din tranzactii autonome sunt vizibile imediat altor tranzactii, pe când cele din tranzactii imbricate sunt vizibile doar dupa ce tranzactia principala este permanentizata.