Subprograme

Subprograme în *PL/SQL*

Noţiunea de subprogram (procedură sau funcţie) a fost concepută cu scopul de a grupa o mulţime de comenzi *SQL* cu instrucţiuni procedurale pentru a construi o unitate logică de tratament.

Unităţile de program ce pot fi create în *PL/SQL* sunt:

- **subprograme locale** (definite în partea declarativă a unui bloc *PL/SQL* sau a unui alt subprogram);
- **subprograme independente** (stocate în baza de date şi considerate drept obiecte ale acesteia);
- **subprograme împachetate** (definite într-un pachet care încapsulează proceduri şi funcţii).

Procedurile şi funcţiile stocate sunt unităţi de program *PL/SQL* apelabile, care există ca obiecte în schema bazei de date *Oracle*. Recuperarea unui subprogram (în cazul unei corecţii) nu cere recuperarea întregii aplicaţii. Subprogramul încărcat în memorie pentru a fi executat, poate fi partajat între obiectele (aplicaţii) care îl solicită.

Este important de făcut **distincţie între procedurile stocate şi procedurile locale** (declarate şi folosite în blocuri anonime).

- Procedurile care sunt declarate şi apelate în blocuri anonime sunt temporare. O procedură stocată (creată cu *CREATE PROCEDURE* sau conţinută într-un pachet) este permanentă în sensul că ea poate fi invocată printr-un script *iSQL**Plus, un subprogram *PL/SQL* sau un declanşator.
- Procedurile şi funcţiile stocate, care sunt compilate şi stocate în baza de date, nu mai trebuie să fie compilate a doua oară pentru a fi executate, în timp ce procedurile locale sunt compilate de fiecare dată când este executat blocul care conţine procedurile şi funcţiile respective.
- Procedurile şi funcţiile stocate pot fi apelate din orice bloc de către utilizatorul care are privilegiul *EXECUTE* asupra subprogramului, în timp ce procedurile şi funcţiile locale pot fi apelate numai din blocul care le conţine.

Când este creat un subprogram stocat, utilizând comanda *CREATE OR REPLACE*, subprogramul este depus în dicţionarul datelor. Este depus atât textul sursă, cât şi forma compilată (*p-code*). Când subprogramul este apelat, *p-code* este citit de pe disc, este depus în *shared pool*, unde poate fi accesat de mai mulţi utilizatori şi este executat dacă este necesar. El va părăsi *shared pool* conform algoritmului *LRU* (*least recently used*).

Subprogramele se pot declara în blocuri *PL/SQL*, în alte subprograme sau în pachete, dar la sfârşitul secţiunii declarative. La fel ca blocurile *PL/SQL* anonime, subprogramele conţin o parte declarativă, o parte executabilă şi opţional, o parte de tratare a erorilor.

Crearea subprogramelor stocate

1) se editează subprogramul (*CREATE PROCEDURE* sau *CREATE FUNCTION*) şi se salvează într-un *script file SQL*;
2) se încarcă şi se execută acest *script file*, este compilat codul sursă, se obţine *p-code* (subprogramul este creat);
3) se utilizează comanda *SHOW ERRORS* pentru vizualizarea eventualelor erori la compilare ale procedurii care a fost cel mai recent compilata sau *SHOW ERRORS PROCEDURE nume* pentru orice procedura compilata anterior (nu poate fi invocata o procedura care contine erori de compilare);
4) se execută subprogramul pentru a realiza acţiunea dorită (de exemplu, procedura poate fi executată de câte ori este necesar, utilizând comanda *EXECUTE* din *iSQL*Plus*) sau se invocă funcţia dintr-un bloc *PL/SQL*.

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

```
CREATE PROCEDURE add_dept IS
v_dept_id dept.department_id%TYPE;
v_dept_name dept.department_name%TYPE;
BEGIN
v_dept_id:=280;
v_dept_name:='ST-Curriculum';
INSERT INTO dept(department_id,department_name)
VALUES(v_dept_id,v_dept_name);
DBMS_OUTPUT.PUT_LINE(' Inserted '|| SQL%ROWCOUNT
||' row ');
END;
...
BEGIN
    add_dept;
END;
```

Dacă există erori la compilare şi se fac corecţiile corespunzătoare, atunci este necesară fie comanda *DROP PROCEDURE* (respectiv *DROP FUNCTION*), fie sintaxa *OR REPLACE* în cadrul comenzii *CREATE*.

Când este apelată o procedură *PL/SQL*, *server*-ul *Oracle* parcurge etapele:

1) Verifică dacă utilizatorul are privilegiul să execute procedura (fie pentru că el a creat procedura, fie pentru că i s-a dat acest privilegiu).

2) Verifică dacă procedura este prezentă în *shared pool*. Dacă este prezentă va fi executată, altfel va fi încărcată de pe disc în *database buffer cache*.

3) Verifică dacă starea procedurii este *validă* sau *invalidă*. Starea unei proceduri *PL/SQL* este *invalidă*, fie pentru că au fost detectate erori la compilarea procedurii, fie pentru că structura unui obiect s-a schimbat de când procedura a fost executată ultima oară. Dacă starea procedurii este *invalidă* atunci este recompilată automat. Dacă nici o eroare nu a fost detectată, atunci va fi executată noua versiune a procedurii.

4) Dacă procedura aparţine unui pachet atunci toate procedurile şi funcţiile pachetului sunt de asemenea încărcate în *database cache* (dacă ele nu erau deja acolo). Dacă pachetul este activat pentru prima oară într-o sesiune, atunci *server*-ul va executa blocul de iniţializare al pachetului.

```
CREATE [OR REPLACE] PROCEDURE/FUNCTION
procedure_name/function_name
[(argument1 [mode1] datatype1,
argument2 [mode2] datatype2,
...)]
IS|AS
subprogram_body;
```

Pentru a afişa codul unui subprogram, parametrii acestuia, precum şi alte informaţii legate de subprogram poate fi utilizată comanda *DESCRIBE*.

Proceduri *PL/SQL*

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

Sintaxa generală pentru crearea unei proceduri este următoarea:

```
[CREATE [OR REPLACE]] PROCEDURE nume_procedură
[(parametru[, parametru]...)] [AUTHID {DEFINER |
CURRENT_USER}] {IS | AS}
[PRAGMA AUTONOMOUS_TRANSACTION;]
[declaraţii locale]
BEGIN
partea executabilă
[EXCEPTION
partea de tratare a excepţiilor]
END [nume_procedură];
unde parametrii au următoarea formă sintactică:
nume_parametru [IN | OUT [NOCOPY] | IN OUT [NOCOPY]
tip_de_date{:= | DEFAULT} expresie]
```

Clauza *CREATE* permite ca procedura să fie stocată în baza de date. Când procedurile sunt create folosind clauza *CREATE OR REPLACE*, ele vor fi stocate în BD în formă compilată. Dacă procedura există, atunci clauza *OR REPLACE* va avea ca efect ştergerea procedurii şi înlocuirea acesteia cu noua versiune. Dacă procedura există, iar *OR REPLACE* nu este prezent, atunci comanda *CREATE* va returna eroarea "*ORA-955: Name is already used by an existing object*".

Clauza *AUTHID* specifică faptul că procedura stocată se execută cu drepturile proprietarului (implicit) sau ale utilizatorului curent. De asemenea, această clauză precizează dacă referinţele la obiecte sunt rezolvate în schema proprietarului procedurii sau a utilizatorului curent.

Clauza *PRAGMA_AUTONOMOUS_TRANSACTION* anunţă compilatorul *PL/SQL* că această procedură este autonomă (independentă). Tranzacţiile autonome permit suspendarea tranzacţiei principale, executarea unor instrucţiuni *SQL*, *commit*-ul sau *rollback*-ul acestor operaţii şi continuarea tranzacţiei principale.

Parametrii formali (variabile declarate în lista parametrilor specificaţiei subprogramului) pot să fie de tipul: *%TYPE*, *%ROWTYPE* sau un tip explicit fără specificarea dimensiunii.

Exemplu:

Să se creeze o procedură stocată care micşorează cu o cantitate dată (*cant*) valoarea poliţelor de asigurare emise de firma ASIROM.

```
CREATE OR REPLACE PROCEDURE mic (cant IN NUMBER) AS
BEGIN
  UPDATE  politaasig
  SET     valoare = valoare - cant
  WHERE   firma = 'ASIROM';
EXCEPTION
  WHEN NO_DATA_FOUND THEN
   RAISE_APPLICATION_ERROR (-20010,'nu exista ASIROM');
END;
   /
```

Dacă sunt operaţii de reactualizare în subprograme şi există declanşatori relativ la aceste operaţii care nu trebuie consideraţi, atunci înainte de apelarea subprogramului declanşatorii trebuie dezactivaţi, urmând ca ei să fie reactivaţi după ce s-a terminat execuţia subprogramului. De exemplu, în problema prezentată anterior ar trebui dezactivaţi declanşatorii referitori la tabelul *politaasig*, apelată procedura *mic* şi în final reactivaţi aceşti declanşatori.

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

***Exemplu*:**

Să se creeze o procedură locală prin care se inserează informaţii 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
      INSERTINTO 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 a unui declanşator;
- interactiv de utilizator utilizând un instrument *Oracle*;
- explicit dintr-o aplicaţie (de exemplu, *SQL*Forms* sau utilizarea de precompilatoare).

Utilizarea (apelarea) unei proceduri se poate face:

1) prin comanda:

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

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

Funcţii *PL/SQL*

Funcţia *PL/SQL* este similară unei proceduri cu excepţia că ea trebuie să întoarcă un rezultat. O funcţie fără comanda *RETURN* va genera eroare la compilare.

Când funcţia este compilată, identificatorul acesteia devine obiect în dicţionarul datelor având tipul *FUNCTION*. Algoritmul din interiorul corpului subprogramului funcţie trebuie să asigure că toate traiectoriile sale conduc la comanda *RETURN*. Dacă o traiectorie a algoritmului trimite în partea de tratare a erorilor, atunci *handler*-ul acesteia trebuie să includă o comandă *RETURN*. O funcţie trebuie să aibă un *RETURN* în antet şi cel puţin un *RETURN* în partea executabilă. Sintaxa simplificată pentru scrierea unei funcţii este următoarea:

[*CREATE* [*OR REPLACE*]] *FUNCTION* *nume_funcţie*
[(*parametru*[, *parametru*]...)]
RETURN *tip_de_date*
[***AUTHID*** {***DEFINER*** | ***CURRENT_USER***}] [***DETERMINISTIC***]
{***IS*** | ***AS***}
[***PRAGMA AUTONOMOUS_TRANSACTION****;*]
[*declaraţii locale*]
BEGIN
partea executabilă
[***EXCEPTION***
partea de mânuire a excepţiilor]
END [*nume_funcţie*];

Opţiunea *tip_de_date* specifică tipul valorii returnate de funcţie, tip care nu poate conţine specificaţii de mărime. Dacă totuşi sunt necesare aceste specificaţii se pot defini subtipuri, iar parametrii vor fi declaraţi de subtipul respectiv.

În interiorul funcţiei trebuie să apară *RETURN expresie*, unde *expresie* este valoarea rezultatului furnizat de funcţie. Pot să fie mai multe comenzi *RETURN* într-o funcţie, dar numai una din ele va fi executată, deoarece dupa ce valoarea este returnata, procesarea blocului inceteaza. Comanda *RETURN* (fără o expresie asociată) poate să apară şi într-o procedură. În acest caz, ea va avea ca efect revenirea la comanda ce urmează instrucţiunii apelante.

Opţiunea *DETERMINISTIC* ajută optimizorul *Oracle* în cazul unor apeluri repetate ale aceleaşi funcţii, având aceleaşi argumente. Ea asigură folosirea unui rezultat obţinut anterior.

În blocul *PL/SQL* al unei proceduri sau funcţii stocate (defineşte acţiunea efectuată de funcţie) nu pot fi referite variabile *host* sau variabile *bind*.

O funcţie poate accepta unul sau mai mulţi parametri, dar trebuie să returneze o singură valoare. Ca şi în cazul procedurilor, lista parametrilor este opţională. Dacă subprogramul nu are parametri, parantezele nu sunt necesare la declarare şi la apelare.

***Exemplu*:**

Să se creeze o funcţie stocată care determină numărul operelor de artă realizate pe pânză, ce au fost achiziţionate la o anumită dată.

```
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;
```

Dacă apare o eroare de compilare, utilizatorul o va corecta în fişierul editat şi apoi va trimite fişierul modificat nucleului, cu opţiunea *OR REPLACE*.

Sintaxa pentru apelul unei funcţii este:

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

O funcţie stocată poate fi apelată în mai multe moduri.

1) Apelarea funcţiei şi atribuirea valorii acesteia într-o variabilă de legătură:

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

Când este utilizată declaraţia *VARIABLE*, pentru variabilele *host* de tip *NUMBER* nu trebuie specificată dimensiunea, iar pentru cele de tip *CHAR* sau *VARCHAR2* valoarea implicită este 1 sau poate fi specificată o altă valoare între paranteze. *PRINT* şi *VARIABLE* sunt comenzi *iSQL*Plus.*

2) Apelarea funcţiei într-o instrucţiune *SQL*:

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

3) Apariţia numelui funcţiei într-o comandă din interiorul unui bloc *PL/SQL* (de exemplu, într-o instrucţiune de atribuire):

```
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;
/
```

Exemplu:

Să se creeze o procedură stocată care pentru un anumit tip de operă de artă (dat ca parametru) calculează numărul operelor din muzeu de tipul respectiv, numărul de specialişti care au expertizat sau au restaurat aceste opere, numărul de expoziţii în care au fost expuse, precum şi valoarea nominală totală 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;
  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;
  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  studiaza, opera
    FROM   studiaza.cod_opera = opera.cod_opera
    WHERE
    AND    opera.tip = v_tip;
    RETURN v_numar;
  END;
  FUNCTION nr_expozitii (v_tip  opera.tip%TYPE)
  RETURN NUMBER IS
    v_numar    NUMBER(3);
  BEGIN
    SELECT COUNT(DISTINCT figureaza_in.cod_expozitie)
    INTO   v_numar  figureaza_in, opera
    FROM   figureaza_in.cod_opera = opera.cod_opera
    WHERE
    AND    opera.tip = v_tip;  RETURN v_numar;
  END;
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;
```

Instrucţiunea *CALL*

O instrucţiune specifică pentru *Oracle* este comanda *CALL* care permite apelarea subprogramelor *PL/SQL* stocate (independente sau incluse în pachete) şi a subprogramelor *Java*.

CALL este o comandă *SQL* care nu este validă într-un bloc *PL/SQL*. Poate fi utilizata in PL/SQL doar dinamic, prin intermediul comenzii *EXECUTE IMMEDIATE*. Pentru executarea acestei comenzi, utilizatorul trebuie să aibă privilegiul *EXECUTE* asupra subprogramului. Comanda poate fi executată interactiv din *SQL*. Ea are sintaxa următoare:

CALL [*schema.*]*nume_subprogram* ([*lista_parametri actuali*]) [*@dblink_nume*] [***INTO*** :*variabila_host*]

Nume_subprogram este numele unui subprogram sau numele unei metode. Clauza *INTO* este folosită numai pentru variabilele de ieşire ale unei funcţii. Dacă clauza *@dblink_nume* lipseşte, sistemul se referă la baza de date locală, iar într-un sistem distribuit clauza specifică numele bazei care conţine subprogramul.

***Exemplu*:**

Sunt prezentate două exemple prin care o funcţie *PL/SQL* este apelată din *SQL*Plus*, respectiv o procedură externă *C* este apelată, 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
```

```
DECLARE
a NUMBER(7);
x VARCHAR2(10);
BEGIN
    EXECUTE IMMEDIATE 'CALL alfa_extern_procedura (:aa,
  :xx)‘ USING a, x;
  END;
```

Modificarea şi suprimarea subprogramelor *PL/SQL*

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

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

Comanda recompilează doar procedurile catalogate standard. Procedurile unui pachet se recompilează într-o altă manieră.

Ca şi în cazul tabelelor, funcţiile şi procedurile pot fi suprimate cu ajutorul comenzii *DROP*. Aceasta presupune eliminarea subprogramelor din dicţionarul datelor. *DROP* este o comandă ce aparţine limbajului de definire a datelor, astfel că se execută un *COMMIT* implicit atât înainte, cât şi după comandă.

Când este şters un subprogram prin comanda *DROP*, automat sunt revocate toate privilegiile acordate referitor la acest subprogram. Dacă este utilizată sintaxa *CREATE OR REPLACE*, privilegiile acordate asupra acestui obiect (subprogram) rămân aceleaşi.

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

Transferarea valorilor prin parametri

Lista parametrilor unui subprogram este compusă din parametri de intrare (*IN*), de ieşire (*OUT*), de intrare/ieşire (*IN OUT*), separaţi prin virgulă.

Dacă nu este specificat nimic, atunci implicit parametrul este considerat *IN*. Un parametru formal cu opţiunea *IN* poate primi valori implicite chiar în cadrul comenzii de declarare. Acest parametru este *read-only* şi deci nu poate fi schimbat în corpul subprogramului. El acţionează ca o constantă. Parametrul actual corespunzător poate fi literal, expresie, constantă sau variabilă iniţializată.

Un parametru formal cu opţiunea *OUT* este neiniţializat şi prin urmare, are automat valoarea *NULL*. În interiorul subprogramului, parametrilor cu opţiunea *OUT* sau *IN OUT* trebuie să li se asigneze o valoare explicită. Dacă nu se atribuie nici o valoare, atunci parametrul actual corespunzător va fi *NULL*. Parametrul actual trebuie să fie o variabilă, nu poate fi o constantă sau o expresie.

Dacă în procedură apare o excepţie, atunci valorile parametrilor formali cu opţiunile *IN OUT* sau *OUT* nu sunt copiate în valorile parametrilor actuali.

Implicit, transmiterea parametrilor este prin valoare în cazul parametrilor *IN* şi este prin referinţă în cazul parametrilor *OUT* sau *IN OUT*. Dacă pentru realizarea unor performanţe se doreşte transmiterea prin referinţă şi în cazul parametrilor *IN OUT* sau *OUT* atunci se poate utiliza opţiunea *NOCOPY*. Dacă opţiunea *NOCOPY* este asociată unui parametru *IN*, atunci va genera o eroare la compilare deoarece aceşti parametri se transmit de fiecare dată prin valoare.

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

- specificarea explicită prin nume;
- specificarea prin poziţie.

Exemplu:

```
CREATE PROCEDURE p1(a IN NUMBER, b IN VARCHAR2,
                    c IN DATE, d OUT NUMBER) AS…;
```

Sunt prezentate diferite moduri pentru apelarea acestei proceduri.

```
DECLARE
  var_a  NUMBER;
  var_b  VARCHAR2;
  var_c  DATE;
  var_d  NUMBER;
BEGIN
  --specificare prin poziţie
  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 şi poziţie
  p1(var_a,var_b,d=>var_d,c=>var_c);
END;
```

Exemplu:

Fie *proces_data* o procedură care procesează în mod normal data zilei curente, dar care opţional poate procesa şi alte date. Dacă nu se specifică parametrul actual corespunzător parametrului formal *plan_data*, atunci acesta va lua automat valoarea dată implicit.

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

Următoarele comenzi reprezintă apeluri corecte ale procedurii *proces_data*:

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

O declaraţie de subprogram (procedură sau funcţie) fără parametri este specificată fără paranteze. De exemplu, dacă procedura *react_calc_dur* şi funcţia *obt_date* nu au parametri, atunci:

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

Module *overload*

În anumite condiţii, două sau mai multe module pot să aibă aceleaşi nume, dar să difere prin lista parametrilor. Aceste module sunt numite module *overload* (supraîncărcate). Funcţia *TO_CHAR* este un exemplu de modul *overload*.

În cazul unui apel, compilatorul compară parametri actuali cu listele parametrilor formali pentru modulele *overload* şi execută modulul corespunzător. Toate programele *overload* trebuie să fie definite în acelaşi bloc *PL/SQL* (bloc anonim, modul sau pachet). Nu poate fi definită o versiune într-un bloc, iar altă versiune într-un bloc diferit.

Modulele *overload* pot să apară în programele *PL/SQL* fie în secţiunea declarativă a unui bloc, fie în interiorul unui pachet. Supraîncărcarea funcţiilor sau procedurilor nu se poate face pentru funcţii sau proceduri stocate, dar se poate face pentru subprograme locale, subprograme care apar în pachete sau pentru metode.

Observaţii:

- Două programe *overload* trebuie să difere, cel puţin, prin tipul unuia dintre parametri. Două programe nu pot fi *overload* dacă parametri lor formali diferă numai prin subtipurile lor şi dacă aceste subtipuri se bazează pe acelaşi tip de date.
- Nu este suficient ca lista parametrilor programelor *overload* să difere numai prin numele parametrilor formali.
- Nu este suficient ca lista parametrilor programelor *overload* să difere numai prin tipul acestora (*IN*, *OUT*, *IN OUT*). *PL/SQL* nu poate face diferenţe (la apelare) între tipurile *IN* sau *OUT*.
- Nu este suficient ca funcţiile *overload* să difere doar prin tipul datei returnate (tipul datei specificate în clauza *RETURN* a funcţiei).

Exemplu:

Următoarele subprograme nu pot fi *overload*.

```
a) FUNCTION alfa(par IN POSITIVE)…;
   FUNCTION alfa(par IN BINARY_INTEGER)…;
b) FUNCTION alfa(par IN NUMBER)…;
   FUNCTION alfa(parar IN NUMBER)…;
c) PROCEDURE beta(par IN VARCHAR2) IS…;
   PROCEDURE beta(par OUT VARCHAR2) IS…;
```

Exemplu:

Să se creeze două funcţii (locale) cu acelaşi nume care să calculeze media valorilor operelor de artă de un anumit tip. Prima funcţie va avea un argument reprezentând tipul operelor de artă, iar cea de a doua va avea două argumente, unul reprezentând tipul operelor de artă, iar celălalt reprezentând stilul operelor pentru care se calculează valoarea medie (adică funcţia va calcula media valorilor operelor de artă de un anumit tip şi care aparţin 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;
```

Procedură *versus* funcţie

Pot fi marcate câteva **deosebiri** esenţiale între funcţii şi proceduri.

- Procedura se execută ca o comandă *PL/SQL*, iar funcţia se invocă ca parte a unei expresii.
- Procedura poate returna (sau nu) una sau mai multe valori, iar funcţia trebuie să returneze (cel putin) o singură valoare.
- Procedura nu trebuie să conţină *RETURN tip_date*, iar funcţia trebuie să conţină această opţiune.

De asemenea, pot fi marcate câteva elemente esenţiale, comune atât funcţiilor cât şi procedurilor. Ambele pot:

- accepta valori implicite;
- avea secţiuni declarative, executabile şi de tratare a erorilor;
- utiliza specificarea prin nume sau poziţie a parametrilor;

Recursivitate

Un subprogram recursiv presupune că acesta se apelează pe el însuşi.

În *Oracle* o problemă delicată este legată de locul unde se plasează un apel recursiv. De exemplu, dacă apelul este în interiorul unui cursor *FOR* sau între comenzile *OPEN* şi *CLOSE*, atunci la fiecare apel este deschis alt cursor. În felul acesta, programul poate depăşi limita pentru *OPEN_CURSORS* setată în parametrul de iniţializare *Oracle*.

Exemplu:

Să se calculeze recursiv al *m*-lea termen din şirul lui Fibonacci.

```
FUNCTION fibona(m POSITIVE) RETURN INTEGER IS
BEGIN
   IF (m = 1) OR (m = 2) THEN
  RETURN 1;
  ELSE
    RETURN fibona(m-1) + fibona(m-2);
  END IF;
END fibona;
```

Declaraţii *forward*

Subprogramele sunt reciproc recursive dacă ele se apelează unul pe altul direct sau indirect. Declaraţiile *forward* permit definirea subprogramelor reciproc recursive.

În *PL/SQL*, un identificator trebuie declarat înainte de a-l folosi. De asemenea, un subprogram trebuie declarat înainte de a-l apela.

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

Procedura *beta* nu poate fi apelată deoarece nu este încă declarată. Problema se poate rezolva simplu, inversând ordinea celor două proceduri. Această soluţie nu este eficientă întotdeauna. *PL/SQL* permite un tip special de declarare a unui subprogram numit *forward*. El constă dintr-o specificare de subprogram terminată prin ";".

```
PROCEDURE beta ( ... );   -- declaraţie forward

..
PROCEDURE alfa ( ... ) IS
BEGIN
  beta( ... );
  ...  END;
  PROCEDURE beta ( ... ) IS
  BEGIN
  ...  END;
```

Se pot folosi declaraţii *forward* pentru a defini subprograme într-o anumită ordine logică, pentru a defini subprograme reciproc recursive, pentru a grupa subprograme într-un pachet.

Lista parametrilor formali din declaraţia *forward* trebuie să fie identică cu

cea corespunzătoare corpului subprogramului. Corpul subprogramului poate apărea oriunde după declaraţia sa *forward*, dar trebuie să rămână în aceeaşi unitate de program.

Utilizarea în expresii *SQL* a funcţiilor definite de utilizator

Începând cu *Release 7.1*, o funcţie stocată poate fi referită într-o comandă *SQL* la fel ca orice funcţie standard furnizată de sistem (*built-in function*), dar cu anumite restricţii. Funcţiile *PL/SQL* definite de utilizator pot fi apelate din orice expresie *SQL* în care pot fi folosite funcţii *SQL* standard.

Funcţiile *PL/SQL* pot să apară în:

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

Exemplu:

Să se afişeze operele de artă (titlu, valoare, stare) a căror valoare este mai mare decât valoarea medie a tuturor operelor de artă din muzeu.

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

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

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

Există restricţii referitoare la folosirea funcţiilor definite de utilizator într-o comandă *SQL*.

- funcţia definită de utilizator trebuie să fie o funcţie stocată (procedurile stocate nu pot fi apelate în expresii *SQL*), nu poate fi locală unui alt bloc;
- funcţia apelată dintr-o comandă *SELECT,* sau din comenzi paralelizate *INSERT, UPDATE* şi *DELETE* nu poate contine comenzi *LMD* care modifica tabelele bazei de date;
- funcţia apelată dintr-o comandă *UPDATE* sau *DELETE* nu poate interoga sau modifica tabele ale bazei reactualizate chiar de aceste comenzi (*table mutating*);
- funcţia apelată din comenzile *SELECT*, *INSERT*, *UPDATE* sau *DELETE* nu poate executa comenzi *LCD* (*COMMIT*), *ALTER SYSTEM*, *SET ROLE* sau comenzi *LDD* (*CREATE*);
- funcţia nu poate să apară în clauza *CHECK* a unei comenzi *CREATE/ALTER TABLE*;

- funcţia nu poate fi folosită pentru a specifica o valoare implicită pentru o coloană în cadrul unei comenzi *CREATE/ALTER TABLE*;
- funcţia poate fi utilizată într-o comandă *SQL* numai de către proprietarul funcţiei sau de utilizatorul care are privilegiul *EXECUTE* asupra acesteia;
- funcţia definită de utilizator, apelabilă dintr-o comandă *SQL*, trebuie să aibă doar parametri de tip *IN*, cei de tip *OUT* şi *IN OUT* nefiind acceptaţi;
- parametrii unei funcţii *PL/SQL* apelate dintr-o comandă *SQL* trebuie să fie specificaţi prin poziţie (specificarea prin nume nefiind permisă);
- parametrii formali ai unui subprogram funcţie trebuie să fie de tip specific bazei de date (*NUMBER*, *CHAR*, *VARCHAR2*, *ROWID*, *LONG*, *LONGROW*, *DATE*), nu tipuri *PL/SQL* (*BOOLEAN* sau *RECORD*);
- tipul returnat de un subprogram funcţie trebuie să fie un tip intern pentru *server*, nu un tip *PL/SQL* (nu poate fi *TABLE*, *RECORD* sau *BOOLEAN*);
- funcţia nu poate apela un subprogram care nu respectă restricţiile 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*. Reactualizarea este insa permisa asupra oricarui alt tabel diferit de *opera*.

Informaţii referitoare la subprograme

Informaţiile referitoare la subprogramele *PL/SQL* şi modul de acces la aceste informaţii sunt următoarele:

- codul sursă, utilizând vizualizarea *USER_SOURCE* din dicţionarul datelor (*DD*);

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

Vizualizarea *USER_OBJECTS* conţine informaţii generale despre toate obiectele manipulate în BD, în particular şi despre subprogramele stocate.

Vizualizarea *USER_OBJECTS* are următoarele câmpuri:

- *OBJECT_NAME* – numele obiectului;
- *OBJECT_TYPE*, – tipul obiectului (*PROCEDURE*, *FUNCTION* etc.);
- *OBJECT_ID* – identificator intern al obiectului;
- *CREATED* – data când obiectul a fost creat;
- *LAST_DDL_TIME* – data ultimei modificări a obiectului;
- *TIMESTAMP* – data şi momentul ultimei recompilări;
- *STATUS* – starea obiectului (*VALID* sau *INVALID*).

Pentru a verifica dacă recompilarea explicită (*ALTER*) sau implicită a avut succes se poate verifica starea subprogramelor utilizând *USER_OBJECTS*.

Orice obiect are o stare (*status*) sesizată în DD, care poate fi:

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

Exemplu:

Să se listeze procedurile şi funcţiile deţinute de utilizatorul curent, precum şi starea acestora.

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

După ce subprogramul a fost creat, codul sursă al acestuia poate fi obţinut consultând vizualizarea *USER_SOURCE* din DD, care are următoarele câmpuri:

- *NAME* – numele obiectului;
- *TYPE* – tipul obiectului;
- *LINE* – numărul liniei din codul sursă;
- *TEXT* – textul liniilor codului sursă.

Exemplu:

Să se afişeze codul complet pentru funcţia *numar_opere*.

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

Exemplu:

Să se scrie o procedură care recompilează toate obiectele invalide din schema personală.

```
     CREATE OR REPLACE PROCEDURE sterge 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 sterge;
```

Dacă se recompilează un obiect *PL/SQL*, atunci *server*-ul va recompila orice obiect invalid de care depinde. Dacă recompilarea automată implicită a procedurilor locale dependente are probleme, atunci starea obiectului va rămâne *INVALID* şi *server*-ul *Oracle* semnalează eroare. Prin urmare:

- este preferabil ca recompilarea să fie manuală (recompilare explicită utilizând comanda *ALTER* (*PROCEDURE*, *FUNCTION*, *TRIGGER*, *PACKAGE*) cu opţiunea *COMPILE*;
- este necesar ca recompilarea să se facă cât mai repede, după definirea unei schimbări referitoare la obiectele bazei.

Pentru a obţine valori (de exemplu, valoarea contorului pentru un *LOOP*, valoarea unei variabile înainte şi după o atribuire etc.) şi mesaje (de exemplu, părăsirea unui subprogram, apariţia unei operaţii etc.) dintr-un bloc *PL/SQL* pot fi utilizate procedurile pachetului *DBMS_OUTPUT*. Aceste informaţii se cumulează într-un *buffer* care poate fi consultat.

Dependenţa subprogramelor

Când este compilat un subprogram, toate obiectele *Oracle* care sunt referite vor fi înregistrate în dicţionarul datelor. Subprogramul este dependent de aceste obiecte. Un subprogram care are erori la compilare este marcat ca "invalid" în dicţionarul datelor. Un subprogram stocat poate deveni, de asemenea, invalid dacă o operaţie *LDD* este executată asupra unui obiect de care depinde.

Obiecte dependente:	Obiecte referite *Table,*
View, Table Procedure	*Secventa View*
Function	*Procedure Function*
Package Specification	*Synonym*
Package Body Database	*Package Specification*
Trigger	*Obiect definit de utilizator*
Obiect definit de utilizator	*Tip colectie*
Tip colectie	

Dacă se modifică definiţia unui obiect referit, obiectul dependent poate (sau nu) să continue să funcţioneze normal.

Există două tipuri de dependenţe:

- dependenţă directă, în care obiectul dependent (de exemplu, *procedure* sau *function*) face referinţă direct la un *table*, *view*, *sequence*, *procedure*, *function*.
- dependenţă indirectă, în care obiectul dependent (*procedure* sau *function*) face referinţă indirect la un *table*, *view*, *sequence*, *procedure*, *function* prin intermediul unui *view*, *procedure* sau *function*.

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

În cazul dependenţelor la distanţă, procedurile stocate local şi toate obiectele dependente vor fi invalidate. Ele nu vor fi recompilate automat la următorul apel.

Exemplu:

Se presupune că procedura *filtru* va referi direct tabelul *opera* şi că procedura *adaug* va reactualiza tabelul *opera* prin intermediul unei vizualizări *nou_opera.*

Pentru aflarea dependenţelor directe se poate utiliza vizualizarea *USER_DEPENDENCIES* din dicţionarul datelor.

```
SELECT NAME, TYPE, REFENCED_NAME, REFENCED_TYPE
FROM USER_DEPENDENCIES
WHEREREFENCED_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
```

Dependenţele indirecte pot fi afişate utilizând vizualizările *DEPTREE* şi *IDEPTREE*. Vizualizarea *DEPTREE* afişează o reprezentare a tuturor obiectelor dependente (direct sau indirect). Vizualizarea *IDEPTREE* afişează o reprezentare a aceleaşi informaţii, sub forma unui arbore.

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

1. executat scriptul *UTLDTREE*;
2. executată 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
FROMDEPTREE
ORDER    BY SEQ#;
   NESTED_LEVEL   TYPE         NAME
              0   Table        opera
              1   View         nou_opera
              2   Procedure    adaug
              1   Procedure    filtru

SELECT  *
FROM    IDEPTREE;

DEPENDENCIES
TABLE   nume_schema.opera
VIEW nume_schema.nou_opera
PROCEDURE nume_schema.adaug
PROCEDURE nume_schema.filtru
```

Dependenţele la distanţă sunt manipulate prin una din modalităţile alese de utilizator: modelul *timestamp* (implicit) sau modelul *signature.*

Fiecare unitate *PL/SQL* are un *timestamp* care este setat când unitatea este modificată (creata sau recompilata) şi care este depus în câmpul *LAST_DDL_TIME* din dicţionarul datelor. Modelul *timestamp* realizează compararea momentelor ultimei modificări a celor două obiecte analizate. Dacă obiectul (referit) bazei are momentul ultimei modificări mai recent ca cel al obiectului dependent, atunci obiectul dependent va fi recompilat.

Modelul *signature* determină momentul la care obiectele bazei distante trebuie recompilate. Când este creată o procedură, o *signature* este depusă în dicţionarul datelor, alături de *p-code*. Aceasta conţine: numele construcţiei *PLSQL* (*PROCEDURE*, *FUNCTION*, *PACKAGE*), tipurile parametrilor, ordinea parametrilor, numărul acestora şi modul de transmitere (*IN*, *OUT*, *IN OUT*). Dacă parametrii se schimbă, atunci evident *signature* se schimbă. Daca signatura nu se schimba, atunci executia continua.

Recompilarea procedurilor şi funcţiilor dependente este fără succes dacă:

- obiectul referit este distrus (*DROP*) sau redenumit (*RENAME*);
- tipul coloanei referite este schimbat;
- coloana referita este stearsa;
- o vizualizare referită este înlocuită printr-o vizualizare având alte coloane;
- lista parametrilor unei proceduri referite este modificată.

Recompilarea procedurilor şi funcţiilor dependente este cu succes dacă:

- tabelul referit are noi coloane;
- nici o coloana nou definita nu are restrictia NOT NULL;
- tipul coloanelor referite nu s-a schimbat;
- un tabel "private" este sters, dar exista un tabel "public" avand acelasi nume si structura;
- toate comenzile *INSERT* contin efectiv lista coloanelor;
- corpul *PL/SQL* a unei proceduri referite a fost modificat şi recompilat cu succes.

Cum pot fi minimizate erorile datorate dependenţelor?

- utilizând comenzi *SELECT* cu opţiunea *;
- incluzând lista coloanelor in cadrul comenzii *INSERT*;
- declarând variabile cu atributul *%TYPE*;
- declarând înregistrări cu atributul *%ROWTYPE.*

În concluzie:

- Dacă procedura depinde de un obiect local, atunci se face recompilare automată la prima reexecuţie.
- Dacă procedura depinde de o procedură distantă, atunci se face recompilare automată, dar la a doua reexecuţie. Este preferabilă o recompilare manuală pentru prima reexecuţie sau implementarea unei strategii de reinvocare a ei (a doua oară).
- Dacă procedura depinde de un obiect distant, dar care nu este procedură, atunci nu se face recompilare automată.

Rutine externe (opțional)

PL/SQL a fost special conceput pentru *Oracle* şi este specializat pentru procesarea tranzacţiilor *SQL*.

Totuşi, într-o aplicaţie complexă pot să apară cerinţe şi funcţionalităţi care sunt mai eficient de implementat în *C*, *Java* sau alt limbaj de programare. Dacă aplicaţia trebuie să efectueze anumite acţiuni care nu pot fi implementate optim

utilizând *PL/SQL*, atunci este preferabil să fie utilizate alte limbaje care realizează performant acţiunile respective. În acest caz este necesară comunicarea între diferite module ale aplicaţiei care sunt scrise în limbaje diferite.

Până la versiunea *Oracle8*, singura modalitate de comunicare între *PL/SQL* şi alte limbaje (de exemplu, limbajul *C*) a fost utilizarea pachetelor *DBMS_PIPE* şi/sau *DBMS_ALERT*.

Începând cu *Oracle8*, comunicarea este simplificată prin utilizarea rutinelor externe. O rutină externă este o procedură sau o funcţie scrisă într-un limbaj diferit de *PL/SQL*, dar apelabilă dintr-un program *PL/SQL*. *PL/SQL* extinde funcţionalitatea *server*-ului *Oracle*, furnizând o interfaţă pentru apelarea rutinelor externe. Orice bloc *PL/SQL* executat pe *server* sau pe *client* poate apela o rutină externă. Singurul limbaj acceptat pentru rutine externe în *Oracle8* era limbajul *C*.

Pentru a marca apelarea unei rutine externe în programul *PL/SQL* este definit un punct de intrare (*wrapper*) care direcţionează spre codul extern (program *PL/SQL* → *wrapper* → cod extern). O clauză specială (*AS EXTERNAL*) este utilizată (în cadrul comenzii *CREATE OR REPLACE PROCEDURE*) pentru crearea unui *wrapper*. De fapt, clauza conţine informaţii referitoare la numele bibliotecii în care se găseşte subprogramul extern (clauza *LIBRARY*), numele rutinei externe (clauza *NAME*) şi corespondenţa (*C* <–> *PL/SQL*) între tipurile de date (clauza *PARAMETERS*). Ultimele versiuni renunta la clauza *AS EXTERNAL*.

Rutinele externe (scrise în *C*) sunt compilate, apoi depuse într-o bibliotecă dinamică (*DLL – dynamic link library*) şi sunt încărcate doar când este necesar acest lucru. Dacă se invocă o rutină externă scrisă în *C*, trebuie setată conexiunea spre această rutină. Un proces numit *extproc* este declanşat automat de către *server*. La rândul său, procesul *extproc* va încărca biblioteca identificată prin clauza *LIBRARY* şi va apela rutina respectivă.

Oracle9i permite utilizarea de rutine externe scrise în *Java*. De asemenea. prin utilizarea clauzei *AS LANGUAGE*, un *wrapper* poate include specificaţii de apelare. De fapt, aceste specificaţii permit apelarea rutinelor externe scrise în orice limbaj. De exemplu, o procedură scrisă într-un limbaj diferit de *C* sau *Java* poate fi utilizată în *SQL* sau *PL/SQL* dacă procedura respectivă este apelabilă din

C. În felul acesta, biblioteci standard scrise în alte limbaje de programare pot fi apelate din programe *PL/SQL*.

Procedura *PL/SQL* executată pe *server*-ul *Oracle* poate apela o rutină externă scrisă în *C* care este depusă într-o bibliotecă partajată.

Procedura *C* se execută într-un spaţiu adresă diferit de cel al *server*-ului *Oracle*, în timp ce unităţile *PL/SQL* şi metodele *Java* se execută în spaţiul de adresă al *server*-ului. *JVM* (*Java Virtual Machine*) de pe pe *server* va executa metoda *Java* direct, fără a fi necesar procesul *extproc*.

Maniera de a încărca depinde de limbajul în care este scrisă rutina (*C* sau *Java*).

- Pentru a apela rutine externe *C*, *server*-ul trebuie să cunoască poziţionarea bibliotecii dinamice *DLL*. Acest lucru este furnizat de *alias*-ul bibliotecii din clauza *AS LANGUAGE*.
- Pentru apelarea unei rutine externe *Java* se va încărca clasa *Java* în baza de date. Este necesară doar crearea unui *wrapper* care direcţionează spre codul extern. Spre deosebire de rutinele externe *C*, nu este necesară nici biblioteca şi nici setarea conexiunii spre rutina externă.

Clauza *LANGUAGE* din cadrul comenzii de creare a unui subprogram, specifică limbajul în care este scrisă rutina (procedură externă *C* sau metodă *Java*) şi are următoarea formă:

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

Pentru o procedură *C* sunt date informaţii referitoare la numele acesteia (clauza *NAME*); *alias*-ul bibliotecii în care se găseşte (clauza *LIBRARY*); opţiuni referitoare la tipul, poziţia, lungimea, modul de transmitere (prin valoare sau prin referinţă) al parametrilor (clauza *PARAMETERS*); posibilitatea ca rutina externă să acceseze informaţii despre parametri, excepţii, alocarea memoriei utilizator (clauza *WITH CONTEXT*).

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

Pentru o metodă *Java*, în clauză trebuie specificată doar signatura metodei (lista tipurilor parametrilor în ordinea apariţiei).

***Exemplu*:**

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

Scrierea "c_calc" este corecta, iar " " implica ca stocarea este *case sensitive*, altfel implicit se depune numele cu litere mari.

Procedura poate fi apelata dintr-un bloc *PL/SQL*:

```
DECLARE
   emp_id    NUMBER;
   procent   NUMBER;
BEGIN
…
calc(emp_id, procent);
…  END;
```

Rutina externă nu este apelată direct, ci se apelează subprogramul *PL/SQL* care referă rutina externă.

Apelarea poate să apară în: blocuri anonime, subprograme independente sau aparţinând unui pachet, metode ale unui tip obiect, declanşatori bază de date, comenzi *SQL* care apelează funcţii (în acest caz trebuie utilizată pragma *RESTRICT_REFERENCES*).

De remarcat că o metodă *Java* poate fi apelată din orice bloc *PL/SQL*, subprogram sau pachet.

JDBC (*Java Database Connectivity*), care reprezintă interfaţa *Java* standard pentru conectare la baze de date relaţionale şi *SQLJ* permit apelarea de blocuri *PL/SQL* din programe *Java*. *SQLJ* face posibilă incorporarea operaţiilor

SQL în codul *Java*. Standardul *SQLJ* acoperă doar operaţii *SQL* statice. *Oracle9i SQLJ* include extensii pentru a suporta direct *SQL* dinamic.

O altă modalitate de a încărca programe *Java* este folosirea interactivă în *iSQL*Plus* a comenzii: *CREATE JAVA instrucţiune*.

Funcţii tabel

O funcţie tabel (*table function*) returnează drept rezultat un set de linii (de obicei, sub forma unei colecţii). Această funcţie poate fi interogată direct printr-o comandă *SQL*, ca şi cum ar fi un tabel al bazei de date. În felul acesta, funcţia poate fi utilizată în clauza *FROM* a unei cereri.

O funcţie tabel conductă (*pipelined table function*) este similară unei funcţii tabel, dar returnează datele iterativ, pe măsură ce acestea sunt obţinute, nu toate deodată. Aceste funcţii sunt mai eficiente deoarece informaţia este returnată imediat cum este obţinută.

Conceptul de funcţie tabel conductă a fost introdus în versiunea *Oracle9i*. Utilizatorul poate să definească astfel de funcţii. De asemenea, este posibilă execuţia paralelă a funcţiilor tabel (evident şi a celor clasice). În acest caz, funcţia trebuie să conţină în declaraţie opţiunea *PARALLEL_ENABLE*.

Funcţia tabel conductă acceptă orice argument pe care îl poate accepta o funcţie obişnuită şi trebuie să returneze o colecţie (*nested table* sau *varray*). Un parametru input poate fi vector, tabel *PL/SQL*, *REF CURSOR*. Ea este declarată specificând cuvântul cheie *PIPELINED* în comanda *CREATE OR REPLACE FUNCTION*. Funcţia tabel conductă trebuie să se termine printr-o comandă *RETURN* simplă, care nu întoarce nici o valoare.

Pentru a returna un element individual al colecţiei este folosită comanda *PIPE ROW*, care poate să apară numai în corpul unei funcţii tabel conductă, în caz contrar generându-se o eroare. Comanda poate fi omisă dacă funcţia tabel conductă nu returnează nici o linie.

După ce funcţia a fost creată, ea poate fi apelată dintr-o cerere *SQL*

utilizând operatorul *TABLE*. Cererile referitoare la astfel de funcţii pot să includă cursoare şi referinţe la cursoare, respectându-se semantica de la cursoarele clasice.

Funcţia tabel conductă nu poate să apară în comenzile *INSERT*, *UPDATE*, *DELETE*. Totuşi, pentru a realiza o reactualizare, poate fi creată o vizualizare relativă la funcţia tabel şi folosit un declanşator *INSTEAD OF*.

***Exemplu*:**

```
CREATE FUNCTION ff(p SYS_REFCURSOR)  RETURN cartype
        PIPELINED IS
  PRAGMA AUTONOMOUS_TRANSACTION;  BEGIN    …    END;
```

In timpul executiei paralele, fiecare instanta a functiei tabel va crea o tranzactie independenta.

Urmatoarele comenzi sunt incorecte.

```
  UPDATE ff(CURSOR (SELECT * FROM tab))
SET col = valoare;  INSERT INTO ff(…)
  VALUES('orice', 'vrei');
```

***Exemplu*:**

Să se obţină o instanţă a unui tabel ce conţine informaţii referitoare la denumirea zilelor săptămânii.

Problema este rezolvată în două variante. Prima reprezintă o soluţie clasică, iar a doua variantă implementează problema cu ajutorul unei funcţii tabel conductă.

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;
```

Funcţia *calc1* poate fi invocată 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 observă că tabelul este implicat doar în tipul rezultatului. Pentru apelarea funcţiei *calc2* este folosită sintaxa următoare:

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

Funcţiile tabel sunt folosite frecvent pentru conversii de tipuri de date. *Oracle9i* introduce posibilitatea de a crea o funcţie tabel care returnează un tip *PL/SQL* (definit într-un bloc). Funcţia tabel care furnizează (la nivel de pachet) drept rezultat un tip de date trebuie să fie de tip conductă. Pentru apelare este utilizată sintaxa simplificată (fără *CAST*).

Procesarea tranzacţiilor autonome

Tranzacţia este o unitate logică de lucru, adică o secvenţă de comenzi care trebuie să se execute ca un întreg pentru a menţine consistenţa bazei. În mod uzual, o tranzacţie poate să cuprindă mai multe blocuri, iar într-un bloc pot să fie mai multe tranzacţii.

O **tranzacţie autonomă** este o tranzacţie independentă începută de altă tranzacţie, numită tranzacţie principală. Tranzacţia autonomă permite suspendarea tranzacţiei principale, executarea de comenzi *SQL*, *commit*-ul şi *rollback*-ul acestor operaţii.

Odată începută, tranzacţia autonomă este independentă în sensul că nu împarte blocări, resurse sau dependenţe cu tranzacţia principală.

În felul acesta, o aplicaţie nu trebuie să cunoască operaţiile autonome ale unei proceduri, iar procedura nu trebuie să cunoască nimic despre tranzacţiile aplicaţiei.

Pentru definirea unei tranzacţii autonome trebuie să se utilizeze pragma *AUTONOMOUS_TRANSACTION* care informează compilatorul *PL/SQL* să marcheze o rutină ca fiind autonomă. Prin rutină se înţelege: bloc anonim de cel mai înalt nivel (nu încuibărit); procedură sau funcţie locală, independentă sau împachetată; metodă a unui tip obiect; declanşator bază de date.

```
CREATE PACKAGE exemplu AS
  …
  FUNCTION autono(x INTEGER) RETURN real;
  END exemplu;
  CREATE PACKAGE BODY exemplu AS
    …
  FUNCTION autono(x INTEGER) RETURN real IS  PRAGMA
    AUTONOMOUS_TRANSACTION;
  z real;
BEGIN
…  END;
END exemplu;
```

Codul *PRAGMA AUTONOMOUS_TRANSACTION* poate marca numai rutine individuale ca fiind independente. Nu pot fi marcate toate subprogramele unui pachet sau toate metodele unui tip obiect ca autonome. Prin urmare, *pragma* nu poate să apară în partea de specificaţie a unui pachet. Codul *PRAGMA AUTONOMOUS_TRANSACTION* se specifica în partea declarativă a rutinei.

Observaţii:

- Declanşatorii autonomi, spre deosebire de cei clasici pot conţine comenzi *LCD* (de exemplu, *COMMIT*, *ROLLBACK*).
- Excepţiile declanşate în tranzacţii autonome generează un *rollback* la nivel de tranzacţie, nu la nivel de instrucţiune.
- Când se intră în secţiunea executabilă a unei tranzacţii autonome, tranzacţia principală se suspendă.

Cu toate că o tranzacţie autonomă este începută de altă tranzacţie, ea **nu** este o tranzacţie încuibărită deoarece:

- nu partajează resurse cu tranzacţia principală;
- nu depinde de tranzacţia principală (de exemplu, dacă tranzacţia principală este *rollback*, atunci tranzacţiile încuibărite sunt de asemenea *rollback*, dar tranzacţia autonomă nu este *rollback*);
- schimbările *commit* din tranzacţii autonome sunt vizibile imediat altor tranzacţii, pe când cele de la tranzacţii încuibărite sunt vizibile doar după ce tranzacţia principală este *commit*.