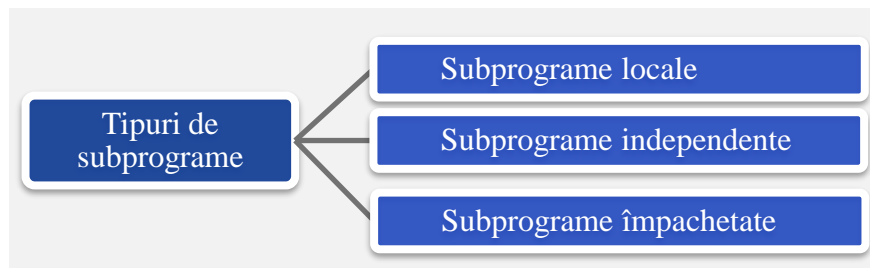


## CUPRINS

6. <i>PL/SQL</i> – Subprograme .....	2
6.1. Proceduri <i>PL/SQL</i> .....	4
6.1.1. Definirea unei proceduri .....	4
6.1.2. Apelarea unei proceduri .....	6
6.1.3. Transferul parametrilor .....	7
6.2. Funcții <i>PL/SQL</i> .....	8
6.2.1. Definirea unei funcții .....	8
6.2.2. Apelarea unei funcții .....	10
6.2.3. Utilizarea în expresii <i>SQL</i> a funcțiilor definite de utilizator .....	11
6.3. Recompilarea subprogramelor <i>PL/SQL</i> .....	12
6.4. Ștergerea subprogramelor <i>PL/SQL</i> .....	12
6.5. Subprograme <i>overload</i> .....	12
6.6. Recursivitate .....	13
6.7. Declarații <i>forward</i> .....	14
6.8. Informații referitoare la subprograme .....	15
6.9. Dependența subprogramelor .....	17
6.10. Rutine externe .....	20
Bibliografie .....	23

## 6. PL/SQL – Subprograme

- Procedurile și funcțiile *PL/SQL* sunt denumite subprograme *PL/SQL*.
- Subprogramele *PL/SQL*:
  - sunt blocuri *PL/SQL* cu nume;
  - au structura similară cu a blocurilor anonime:
    - secțiunea declarativă este opțională (cuvântul cheie *DECLARE* se înlocuiește cu *IS* sau *AS*);
    - secțiunea executabilă este obligatorie;
    - secțiunea de tratare a excepțiilor este opțională.



**Fig. 6.1.** Tipuri de subprograme

- În funcție de locul în care sunt definite subprogramele *PL/SQL* pot fi:
  - locale
    - definite în partea declarativă a unui bloc *PL/SQL* sau a unui alt subprogram
  - independente
    - stocate în baza de date și considerate drept obiecte ale acesteia
  - împachetate
    - definite într-un pachet *PL/SQL*



- ❖ Subprogramele împachetate sunt subprograme stocate?



- ❖ Un subprogram local, declarat și apelat într-un bloc anonim, este temporar sau permanent? Poate fi apelat din alte aplicații?
- ❖ Un subprogram stocat este temporar sau permanent? Poate fi apelat din alte aplicații?



- ❖ De câte ori este compilat un subprogram local?
- ❖ De câte ori este compilat un subprogram stocat?



Avantajele utilizării subprogramelor stocate:

- ❖ codul este ușor de întreținut
    - modificările necesare îmbunătățirii mai multor aplicații trebuie realizate o singură dată
    - se minimizează timpul necesar testării
  - ❖ codul este reutilizabil
    - după ce au fost compilate și validate, subprogramele pot fi reutilizate în oricât de multe aplicații
  - ❖ asigură securitatea datelor
    - acordând privilegii asupra subprogramelor, se poate permite accesul indirect asupra obiectelor bazei de date
  - ❖ asigură integritatea datelor
    - se pot grupa mai multe acțiuni înrudite care vor fi executate împreună sau niciuna
  - ❖ îmbunătățesc performanța
    - atunci când este creat un subprogram independent, în dicționarul datelor este depus atât textul sursă, cât și forma compilată (*p-code*)
    - dacă subprogramul este apelat, *p-code*-ul este citit de pe disc, este depus în *shared pool* și poate fi utilizat de mai mulți utilizatori
    - *p-code*-ul va părăsi *shared pool* conform algoritmului *LRU* (*least recently used*)
- Atunci când este apelat un subprogram stocat, *server*-ul *Oracle* parcurge etapele:
    - Verifică dacă utilizatorul are privilegiul de execuție asupra subprogramului.
    - Verifică dacă *p-code*-ul subprogramului este în *shared pool*. Dacă este prezent va fi executat, altfel va fi încărcat de pe disc în *shared pool*.
    - Verifică dacă starea subprogramului este *VALID* sau *INVALID*. Starea unui subprogram este *INVALID*, fie pentru că au fost detectate erori la compilarea acestuia, fie pentru că structura unui obiect s-a schimbat de când subprogramul a fost executat ultima oară. Dacă starea subprogramului este *INVALID*, atunci este recompilat automat. Dacă nu a fost detectată nicio eroare, atunci va fi executată noua versiune a subprogramului.

- Dacă subprogramul aparține unui pachet atunci toate subprogramele pachetului sunt de asemenea încărcate în *shared pool* (dacă acestea nu erau deja acolo). Dacă pachetul este activat pentru prima oară într-o sesiune, atunci *server*-ul va executa blocul de inițializare al pachetului.

## 6.1. Proceduri *PL/SQL*

- O procedură este un bloc *PL/SQL* cu nume care poate accepta parametrii.
- În general procedurile sunt utilizate pentru a realiza anumite acțiuni.
- Procedurile independente sunt compilate și stocate în baza de date ca obiecte ale schemei. Tipul acestor obiecte este *procedure*.

### 6.1.1. Definirea unei proceduri

- Sintaxa

```
[CREATE [OR REPLACE]] PROCEDURE nume_procedură
    [(parametru[, parametru]...)]
    [AUTHID {DEFINER | CURRENT_USER}]
    {IS | AS}
    [PRAGMA AUTONOMOUS_TRANSACTION;]
    [declarații locale]
BEGIN
    secțiune executabilă
[EXCEPTION
    secțiune de gestiune a excepțiilor]
END [nume_procedură];
```

**BLOC PL/SQL**

- Clauza *CREATE* determină stocarea procedurii în baza de date.
- Clauza *OR REPLACE* are ca efect ștergerea procedurii cu numele specificat (dacă aceasta există deja) și înlocuirea acesteia cu noua versiune. Dacă procedura cu numele specificat în comanda *CREATE* există și se omite clauza *OR REPLACE*, atunci apare eroarea „ORA-955: Name is already used by an existing object”.
- Parametrii specificați au următoarea formă sintactică:

```
parametru IN tip_de_date {:= | DEFAULT} expresie
| { OUT | IN OUT } [NOCOPY] tip_de_date
```



Doar parametrii de tip *IN* pot avea specificate valori implicite (*DEFAULT*).

- Opțiunea *NOCOPY* poate fi utilizată doar pentru parametrii de tip *OUT* sau *IN OUT*. Determină baza de date să transmită parametrii de tip *OUT* sau *IN OUT* prin referință (parametrii de tip *IN* sunt transmiși doar prin referință, iar parametrii de *OUT* sunt transmiși implicit prin valoare). Atunci când se transmite o valoare mare (de exemplu, o colecție), această clauză poate îmbunătăți în mod semnificativ performanța.
- Tipul parametrilor poate fi specificat utilizând atributele *%TYPE*, *%ROWTYPE* sau un tip explicit fără dimensiune specificată.
- Clauza *AUTHID* specifică faptul că procedura stocată se execută cu drepturile proprietarului (implicit) sau ale utilizatorului curent. De asemenea, această clauză precizează dacă referințele către obiecte sunt rezolvate în schema proprietarului procedurii sau a utilizatorului curent.
- Clauza *PRAGMA AUTONOMOUS\_TRANSACTION* informează compilatorul *PL/SQL* că această procedură este autonomă (independentă). Tranzacțiile autonome permit suspendarea tranzacției principale, executarea unor instrucțiuni *SQL*, *commit*-ul sau *rollback*-ul acestor operații și continuarea tranzacției principale.

**Exemplul 6.1 – vezi explicații curs**

```
CREATE OR REPLACE PROCEDURE proc_ex1
    (v_id produse.id_produs%TYPE, v_procent NUMBER)
AS
BEGIN
    UPDATE  produse
    SET      pret_unitar =
              pret_unitar + pret_unitar*v_procent
    WHERE    id_produs = v_id;
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        RAISE_APPLICATION_ERROR (-20000, 'Nu exista produsul');
END;
```

**Exemplul 6.2 – vezi curs**



- ❖ Dacă subprogramul conține comenzi *LMD*, atunci execuția acestuia va determina execuția tuturor *trigger*-ilor definiți pentru aceste operații.
- ❖ Dacă se dorește evitarea declanșării acestora, atunci înainte de apelarea subprogramului *trigger*-ii trebuie dezactivați, urmând ca aceștia să fie reactivați după ce s-a terminat execuția subprogramului.

**Exemplul 6.3**

```
ALTER TABLE produse DISABLE ALL TRIGGERS;  
EXECUTE proc_ex2(37,0.1)  
ALTER TABLE produse ENABLE ALL TRIGGERS;
```

### 6.1.2. Apelarea unei proceduri

- Procedurile stocate pot fi apelate:
  - din corpul altei proceduri sau al unui declanșator;
  - interactiv, de utilizator folosind un utilitar *Oracle* (de exemplu, *SQL\*Plus*);
  - explicit, dintr-o aplicație (de exemplu, *Oracle Forms* sau un precompilator).
- Apelarea unei proceduri se realizează
  - în *SQL\*Plus*, prin comanda:  
`EXECUTE nume_procedură [(lista_parametri_actuali)];`
  - în *PL/SQL*, prin apariția numelui procedurii urmat de lista parametrilor actuali:  
`nume_procedură [(lista_parametri_actuali)];`



O procedură stocată poate fi invocată într-o comandă *SQL* (de exemplu, în comanda ***SELECT***)?

- La apelarea unei proceduri, parametrii actuali pot fi definiți specificându-i
  - explicit, prin nume;
  - prin poziție .

**Exercițiu 6.4 – temă**

Definiți un bloc *PL/SQL* în care procedura *proc\_ex2* este apelată pentru fiecare produs din categoria „Sisteme de operare” (nivel 5). Prețul acestor produse va fi micșorat cu 5%.

**Exemplul 6.5 – vezi curs**

### 6.1.3. Transferul parametrilor

- Parametrii formali ai unei proceduri pot fi:
  - parametri de intrare (*IN*);
  - parametri de ieșire (*OUT*);
  - de intrare/ieșire (*IN OUT*).
- Dacă nu este specificat tipul parametrului, atunci implicit este considerat *IN*.
- Parametrul formal de tip *IN*
  - Poate primi valori implicite în cadrul comenzii de declarare.
  - Este *read-only* și deci valoarea sa nu poate fi modificată în corpul subprogramului.
  - Parametrul actual corespunzător poate fi literal, expresie, constantă sau variabilă inițializată.
- Parametrul formal de tip *OUT*
  - Este neinițializat și prin urmare, are automat valoarea *null*.
  - În interiorul subprogramului, parametrilor cu opțiunea *OUT* sau *IN OUT* trebuie să li se asigneze o valoare explicită. Dacă nu se atribuie nicio 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 excepție, atunci valorile parametrilor formali cu opțiunile *OUT* sau *IN OUT* nu sunt copiate în valorile parametrilor actuali.
- Implicit, transmiterea parametrilor este:
  - prin referință, în cazul parametrilor *IN*;
  - prin valoare în cazul parametrilor *OUT* sau *IN OUT*.
    - Dacă din motive de performanță se dorește transmiterea prin referință și în cazul parametrilor *IN OUT* sau *OUT*, atunci se poate utiliza opțiunea *NOCOPY*.
    - Dacă opțiunea *NOCOPY* este asociată unui parametru *IN*, atunci va genera o eroare la compilare, deoarece acești parametri se transmit de fiecare dată prin referință.

**Exemplul 6.6** – **vezi curs**

**Exemplul 6.7** – **vezi curs**

## 6.2. Funcții PL/SQL

- O funcție *PL/SQL* este un bloc *PL/SQL* cu nume care trebuie să întoarcă un rezultat (o singură valoare).
- Funcțiile independente sunt compilate și stocate în baza de date ca obiecte ale schemei. Tipul acestor obiecte este *function*.



O funcție stocată poate fi invocată într-o comandă *SQL* (de exemplu, în comanda *SELECT*)?

### 6.2.1. Definirea unei funcții

- Sintaxa

```
[CREATE [OR REPLACE]] FUNCTION nume_funcție
    [(parametru[, parametru]...)]
RETURN tip_de_date
    [AUTHID {DEFINER | CURRENT_USER}]
    [DETERMINISTIC]
    {IS | AS}
    [PRAGMA AUTONOMOUS_TRANSACTION;]
    [declarații locale]
BEGIN
    secțiune executabilă
    [EXCEPTION
        secțiune de gestiune a excepțiilor]
END [nume_funcție];
```

**BLOC PL/SQL**

- Clauza *RETURN* este utilizată în locul parametrilor de tip *OUT*.
  - O funcție trebuie să conțină clauza *RETURN* în antet și cel puțin o comandă *RETURN* în partea executabilă.
    - În interiorul funcției trebuie să apară *RETURN expresie*, unde *expresie* este valoarea rezultatului furnizat de funcție.
    - Într-o funcție pot să apară mai multe comenzi *RETURN*, dar numai una din acestea va fi executată, deoarece după ce valoarea este întoarsă, procesarea blocului încetează.



- Algoritmul din interiorul corpului subprogramului funcție 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 funcție fără comanda *RETURN* va genera eroare la compilare.
- Comanda *RETURN* (fără o expresie asociată) poate să apară și într-o procedură. În acest caz, ea va avea ca efect revenirea la comanda ce urmează instrucțiunii apelante.
- Opțiunea *tip\_de\_date* specifică tipul valorii întoarse de funcție, tip care nu poate conține specificații de mărime.
  - Dacă totuși sunt necesare aceste specificații se pot defini subtipuri, iar parametrii vor fi declarați de subtipul respectiv.
- Opțiunea *DETERMINISTIC* indică optimizorului că funcția va întoarce același rezultat dacă sunt folosite aceleași argumente la apelarea sa. Dacă sunt realizate apeluri repetate ale funcției, având aceleași argumente, atunci optimizorul poate utiliza un rezultat obținut anterior.
- O funcție poate accepta unul sau mai mulți parametri. Ca și în cazul procedurilor, lista parametrilor este opțională. Dacă subprogramul nu are parametri, parantezele nu sunt necesare la declarare și la apelare.



Funcțiile acceptă toate cele 3 tipuri de parametri (*IN*, *OUT* sau *IN OUT*).

În comenzile *SQL* pot fi utilizate doar funcții cu parametri de tip *IN*.



Ce tipuri de proceduri pot fi transformate în funcții?

**Exemplul 6.8** – **vezi curs**

### 6.2.2. Apelarea unei funcții

- O funcție independentă poate fi apelată în mai multe moduri, folosind sintaxa:

`nume_funcție [(lista_parametri_actuali)]`

- într-o comandă *SQL*;

#### Exemplul 6.9

```
SELECT func_ex8(100,2007)
FROM   DUAL;
```

- în *SQL\*PLUS* (apelarea funcției și atribuirea valorii acesteia într-o variabilă de legătură);

#### Exemplul 6.10

```
VARIABLE rezultat NUMBER
EXECUTE :rezultat := func_ex8(100,2007);
PRINT rezultat
```

- în *PL/SQL*;

#### Exemplul 6.11

```
BEGIN
  DBMS_OUTPUT.PUT_LINE(func_ex8(100,2007));
END;
```

- La apelarea unei funcții, parametrii actuali pot fi definiți specificându-i

- explicit, prin nume;
- prin poziție.

#### Exemplul 6.12

```
DECLARE
  n NUMBER := func_ex8(p_id=>100);
BEGIN
  DBMS_OUTPUT.PUT_LINE(n);
END;
```

### Comanda *CALL*

- Permite apelarea subprogramelor *PL/SQL* stocate (independente sau incluse în pachete) și a subprogramelor *Java*.
- Este o comandă *SQL* care nu este validă într-un bloc *PL/SQL*.
  - În *PL/SQL* poate fi utilizată doar dinamic, prin intermediul comenzii *EXECUTE IMMEDIATE*.

- Sintaxa:

```
CALL nume_subprogram([lista_parametri actuali])  
[INTO :variabila_host]
```

- *nume\_subprogram* reprezintă numele unui subprogram sau numele unei metode.
- Clauza *INTO* este folosită numai pentru variabilele de ieșire ale unei funcții.

**Exemplul 6.13**

```
SQL> VARIABLE rezultat NUMBER  
SQL> CALL func_ex8(100,2007) INTO :rezultat;  
  
Call completed.  
  
SQL> PRINT rezultat  
  
      REZULTAT  
-----  
           863
```

### 6.2.3. Utilizarea în expresii *SQL* a funcțiilor definite de utilizator

- O funcție stocată poate fi referită într-o comandă *SQL* la fel ca orice funcție standard furnizată de sistem, dar cu anumite restricții.
- Funcțiile definite de utilizator pot fi apelate din orice expresie *SQL* în care pot fi folosite funcții *SQL* standard.
- Funcțiile definite de utilizator pot să apară în:
  - clauza *SELECT* a comenzii *SELECT*;
  - clauzele *WHERE* și *HAVING*;
  - clauzele *CONNECT BY*, *START WITH*, *ORDER BY* și *GROUP BY*;
  - clauza *VALUES* a comenzii *INSERT*;
  - clauza *SET* a comenzii *UPDATE*.

**Exemplul 6.14 – vezi curs**

Funcțiile ce pot fi utilizate în comenzi *SQL* trebuie:

- ❖ să aibă numai parametri de tip *IN*, al căror tip de date este un tip valid *SQL*;
- ❖ să întoarcă o valoare al cărei tip să fie un tip valid *SQL*, cu dimensiunea maximă admisă în *SQL*.



Restricții de utilizare a funcțiilor în comenzi *SQL*:

- ❖ funcțiile invocate într-o comandă *SELECT* nu pot conține comenzi *LMD*;
- ❖ funcțiile invocate într-o comandă *UPDATE* sau *DELETE* asupra unui tabel *T* nu pot utiliza comenzi *SELECT* sau *LMD* care referă același tabel *T* (*table mutating*);
- ❖ nu pot termina tranzații (nu pot utiliza comenzile *COMMIT* sau *ROLLBACK*);
- ❖ nu pot utiliza comenzi *LDD* (de exemplu, *CREATE TABLE*) sau *LCD* (de exemplu, *ALTER SESSION*), deoarece acestea realizează *COMMIT* automat;
- ❖ nu pot să apară în clauza *CHECK* a unei comenzi *CREATE/ALTER TABLE*;
- ❖ nu pot fi folosite pentru a specifica o valoare implicită a unei coloane în cadrul unei comenzi *CREATE/ALTER TABLE*;
- ❖ nu pot utiliza subprograme care nu respectă restricțiile enumerate anterior.

### 6.3. Recompilarea subprogramelor *PL/SQL*

- Pentru a recompila subprogramele independente invalide se utilizează comanda

```
ALTER {FUNCTION | PROCEDURE} nume_subprogram COMPILE;
```

- Recompilarea explicită elimină recompilarea implicită la *run-time* și previne apariția erorilor de compilare la acel moment.

### 6.4. Ștergerea subprogramelor *PL/SQL*

- Pentru a șterge un subprogram independent se utilizează comanda:

```
DROP {FUNCTION | PROCEDURE} nume_subprogram;
```

### 6.5. Subprograme *overload*

- Subprogramele *overload* (supraîncărcate) au aceleași nume, dar diferă prin lista parametrilor.
  - *Exemplu*: funcția predefinită *TO\_CHAR*.
- În cazul unui apel, compilatorul compară parametri actuali cu listele parametrilor formali ale subprogramelor *overload* și execută modulul corespunzător.



- ❖ Toate subprogramele *overload* trebuie definite în același bloc *PL/SQL* (bloc anonim sau pachet).
- ❖ Subprogramele *overload* pot să apară în programele *PL/SQL*:
  - în secțiunea declarativă a unui bloc;
  - în interiorul unui pachet.
- ❖ Subprogramele independente nu pot fi *overload*.
- ❖ Două programe *overload* trebuie să difere, cel puțin, prin tipul unuia dintre parametri. Două programe nu pot fi *overload* dacă parametri lor formali diferă numai prin subtipurile lor și dacă aceste subtipuri se bazează pe același tip de date.
- ❖ Pentru ca două subprograme să fie *overload* nu este suficient ca:
  - lista parametrilor să difere numai prin numele parametrilor formali;
  - lista parametrilor să difere numai prin tipul acestora (*IN*, *OUT*, *IN OUT*); *PL/SQL* nu poate face diferențe (la apelare) între tipurile *IN* sau *OUT*;
  - să difere doar prin tipul datei returnate (tipul datei specificate în clauza *RETURN* a unei funcții).



Următoarele subprograme nu pot fi *overload*:

- a) `FUNCTION alfa(v POSITIVE) ...;`  
`FUNCTION alfa(v PLS_INTEGER) ...;`
- b) `FUNCTION alfa(x NUMBER) ...;`  
`FUNCTION alfa(y NUMBER) ...;`
- c) `PROCEDURE beta(v IN VARCHAR2) ...;`  
`PROCEDURE beta(v OUT VARCHAR2) ...;`

**Exemplul 6.15** – **vezi curs**

## 6.6. Recursivitate

- Un subprogram recursiv se apelează pe el însuși.
- Fiecare invocare recursivă a subprogramului creează câte o instanță pentru fiecare componentă declarată în subprogram și pentru fiecare comandă *SQL* executată de subprogram. Dacă apelul este în interiorul unui cursor *FOR* sau între comenzile *OPEN* și *CLOSE*, atunci la fiecare apel este deschis alt cursor. Programul poate determina depășirea limitei impuse de parametrul *OPEN\_CURSORS*.

**Exemplul 6.16** – **vezi curs**

## 6.7. Declarații *forward*

- Două subprograme sunt reciproc recursive dacă ele se apelează unul pe altul direct sau indirect.
- Declarațiile *forward* permit:
  - definirea subprogramelor reciproc recursive;
  - declararea subprogramelor în ordine alfabetică sau într-o anumită ordine logică (în blocuri *PL/SQL* sau pachete).
- În *PL/SQL*, un identificator trebuie declarat înainte de a-l folosi. De asemenea, un subprogram trebuie declarat înainte de a-l apela.

### Exemplul 6.17

```
DECLARE

  PROCEDURE alfa IS
  BEGIN
    beta('apel beta din alfa');    -- apel incorect
  END;

  PROCEDURE beta (x VARCHAR2) IS
  BEGIN
    DBMS_OUTPUT.PUT_LINE(x);
  END;
BEGIN
  alfa;
END;
```

- Eroarea apare deoarece procedura *beta* este apelată înainte de a fi declarată. Problema se poate rezolva simplu, inversând ordinea celor două proceduri. Această soluție nu este eficientă întotdeauna.

```
DECLARE

  PROCEDURE beta (x VARCHAR2) IS
  BEGIN
    DBMS_OUTPUT.PUT_LINE(x);
  END;

  PROCEDURE alfa IS
  BEGIN
    beta('apel beta din alfa');    -- apel corect
  END;
BEGIN
  alfa;
END;
```

- *PL/SQL* permite un tip special de declarare a unui subprogram numit *forward*. Acesta constă dintr-o specificare de subprogram terminată prin “;”.

```
DECLARE

  PROCEDURE beta (x VARCHAR2);  -- declaratie forward

  PROCEDURE alfa IS
  BEGIN
    beta('apel beta din alfa'); -- apel corect
  END;

  PROCEDURE beta (x VARCHAR2) IS
  BEGIN
    DBMS_OUTPUT.PUT_LINE(x);
  END;
BEGIN
  alfa;
END;
```

- Lista parametrilor formali din declarația *forward* trebuie să fie identică cu cea corespunzătoare corpului subprogramului. Corpul subprogramului poate apărea oriunde după declarația sa *forward*, dar trebuie să rămână în aceeași unitate de program.

## 6.8. Informații referitoare la subprograme

- Informațiile referitoare la subprogramele *PL/SQL* și modul de acces la acestea:
  - informații generale - vizualizarea *USER\_OBJECTS*;
  - codul sursă - vizualizarea *USER\_SOURCE*;
  - tipul parametrilor (*IN*, *OUT*, *IN OUT*) - comanda *DESCRIBE* din *SQL\*Plus*;
  - erorile la compilare - vizualizarea *USER\_ERRORS* sau comanda *SHOW ERRORS* din *SQL\*Plus*.

### Vizualizarea *USER\_OBJECTS*

- Conține informații generale despre toate obiectele bazei de date, în particular și despre subprogramele stocate.
- Vizualizarea *USER\_OBJECTS* conține informații despre:
  - *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 și momentul ultimei recompilări;
  - *STATUS* – starea obiectului (*VALID* sau *INVALID*) etc.
- Pentru a verifica dacă recompilarea explicită (*ALTER*) sau implicită a avut succes se poate consulta starea subprogramelor utilizând *USER\_OBJECTS*.
    - Orice obiect are o stare (*status*) sesizată în dicționarul datelor, care poate fi:
      - *VALID* - obiectul a fost compilat și poate fi folosit când este referit;
      - *INVALID* - obiectul trebuie compilat înainte de a fi folosit.

**Exemplul 6.18**

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

**Exemplul 6.19**

```
CREATE OR REPLACE PROCEDURE proc_ex19 IS
BEGIN
  FOR i IN (SELECT  OBJECT_TYPE, OBJECT_NAME
             FROM    USER_OBJECTS
             WHERE   STATUS = 'INVALID'
             AND     OBJECT_TYPE IN ('PROCEDURE',
                                     'FUNCTION')) LOOP
    --recompileaza toate subprogramele invalide
    --din schema curenta
    DBMS_DDL.ALTER_COMPILE(i.OBJECT_TYPE, USER,
                           i.OBJECT_NAME);
  END LOOP;
END;
```

- 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* și *server*-ul *Oracle* semnalează eroare.
  - Este preferabil ca recompilarea să fie manuală (recompilare explicită utilizând comanda *ALTER (PROCEDURE, FUNCTION, TRIGGER, PACKAGE)* cu opțiunea *COMPILE*.



- Este necesar ca recompilarea să se facă cât mai repede, după definirea unei schimbări referitoare la obiectele bazei.

### Vizualizarea *USER\_SOURCE*

- După ce subprogramul a fost creat, codul sursă al acestuia poate fi obținut consultând vizualizarea *USER\_SOURCE* din dicționarul datelor, care are următoarele câmpuri:
  - *NAME* – numele obiectului;
  - *TYPE* – tipul obiectului;
  - *LINE* – numărul liniei din codul sursă;
  - *TEXT* – textul liniilor codului sursă.

#### Exemplul 6.20

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

## 6.9. Dependența subprogramelor

- Atunci când un subprogram este compilat, Atunci când un subprogram este compilat, în dicționarul datelor se vor înregistra informații despre toate obiectele referite.
  - Subprogramul este dependent de aceste obiecte.
- Un subprogram care are erori la compilare este marcat ca “*invalid*” în dicționarul datelor.
  - Un subprogram stocat poate deveni, de asemenea, invalid dacă o operație *LDD* este executată asupra unui obiect de care depinde.
- Dacă se modifică definiția unui obiect referit, obiectul dependent poate (sau nu) să continue să funcționeze normal.
- Există două tipuri de dependențe:
  - dependență directă
    - obiectul dependent (de exemplu, *procedure* sau *function*) face referință direct la un *table*, *view*, *sequence*, *procedure* sau *function*
  - dependență indirectă
  - obiectul dependent (de exemplu, *procedure* sau *function*) face referință indirect la un *table*, *view*, *sequence*, *procedure*, *function* prin intermediul unui *view*, *procedure* sau *function*

- În cazul dependențelor locale, atunci 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 dependențelor la distanță, procedurile stocate local și toate obiectele dependente vor fi invalidate. Acestea nu vor fi recompilate automat la următorul apel.
- Vizualizarea *USER\_DEPENDENCIES* oferă informații despre obiectele referite de un obiect dependent.

**Exemplul 6.21**

```
SELECT NAME, TYPE, REFERENCED_NAME, REFERENCED_TYPE
FROM   USER_DEPENDENCIES
WHERE  REFERENCED_TYPE IN
        ('TYPE', 'TABLE', 'PROCEDURE', 'FUNCTION', 'VIEW')
AND    NAME NOT LIKE 'BIN%'
ORDER BY 1;
```

- Dependențele indirecte pot fi afișate utilizând vizualizările *DEPTREE* și *IDEPTREE*.
  - Vizualizarea *DEPTREE* afișează o reprezentare a tuturor obiectelor dependente (direct sau indirect).
  - Vizualizarea *IDEPTREE* afișează sub forma unui arbore aceleași informații.
- Pentru a utiliza aceste vizualizări furnizate de sistemul *Oracle* trebuie:
  1. conectare ca administrator;
  2. executat scriptul *UTLDTREE*;
  3. executată procedura *DEPTREE\_FILL* (are trei argumente: tipul obiectului referit, schema obiectului referit, numele obiectului referit).

**Exemplul 6.22**

```
--conectare ca administrator
EXECUTE DEPTREE_FILL ('TABLE', 'CURS_PLSQL', 'FACTURI');

SELECT NESTED_LEVEL, TYPE, NAME
FROM   DEPTREE
ORDER BY SEQ#;

SELECT *
FROM   IDEPTREE;
```

- Dependențele la distanță pot fi manipulate folosind modelul *timestamp* (implicit) sau modelul *signature*.
- Ori de câte ori o unitate *PL/SQL* este modificată (creată sau recompilată) este înregistrat momentul de timp la care are loc modificarea (*timestamp*). Acesta poate fi observat interogând vizualizarea *USER\_OBJECTS*, câmpul *LAST\_DDL\_TIME*. Modelul *timestamp* realizează compararea momentelor ultimei modificări a celor două obiecte analizate. Dacă un obiect (referit) are momentul ultimei modificări mai recent decât cel al obiectului dependent, atunci obiectul dependent va fi recompilat.
- Modelul *signature* determină momentul la care obiectele bazei distante trebuie recompilate. Atunci când este creată o unitate *PL/SQL*, o semnătură este depusă în dicționarul datelor, alături de *p-code*. Aceasta conține numele blocului *PLSQL* (*PROCEDURE*, *FUNCTION*, *PACKAGE*), tipurile parametrilor, ordinea parametrilor, numărul acestora și modul de transmitere (*IN*, *OUT*, *IN OUT*), tipul de date întors de funcție. Dacă semnătura nu este modificată, atunci execuția continuă (fără a fi necesară recompilarea).



Recompilarea procedurilor și funcțiilor dependente este fără succes dacă:

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



Recompilarea procedurilor și funcțiilor dependente este cu succes dacă:

- ❖ tabelul referit are coloane noi;
- ❖ nicio coloană nou definită nu are restricția *NOT NULL*;
- ❖ tipul coloanelor referite nu s-a schimbat;
- ❖ un tabel privat este șters, dar există un tabel public cu același nume și structură;
- ❖ comenzile *INSERT* conțin efectiv lista coloanelor;
- ❖ un subprogram referit a fost modificat și recompilat cu succes.



Cum pot fi minimizate erorile datorate dependențelor?

- ❖ utilizând comenzi *SELECT* cu opțiunea \*;
- ❖ incluzând lista coloanelor în cadrul comenzii *INSERT*;
- ❖ declarând variabile cu atributul *%TYPE*;
- ❖ declarând înregistrări cu atributul *%ROWTYPE*.



- ❖ Dacă procedura depinde de un obiect local, atunci se face recompilare automată la prima reexecuție.
- ❖ Dacă procedura depinde de o procedură distantă, atunci se face recompilare automată, dar la a doua reexecuție. Este preferabilă o recompilare manuală pentru prima reexecuție 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.

## 6.10. Rutine externe

- *PL/SQL* a fost special conceput pentru *Oracle* și este specializat pentru procesarea tranzacțiilor *SQL*.
- Totuși, într-o aplicație complexă pot să apară cerințe și funcționalități care sunt mai eficient de implementat în *C*, *Java* sau alt limbaj de programare. Dacă aplicația trebuie să efectueze anumite acțiuni care nu pot fi implementate optim utilizând *PL/SQL*, atunci este preferabil să fie utilizate alte limbaje care realizează performant acțiunile respective. În acest caz este necesară comunicarea între diferite module ale aplicației care sunt scrise în limbaje diferite.
- Rutinele externe:
  - sunt subprograme scrise într-un limbaj diferit de *PL/SQL*;
  - sunt apelabile dintr-un program *PL/SQL*.
- *PL/SQL* extinde funcționalitatea server-ului *Oracle*, furnizând o interfață pentru apelarea rutinelor externe. Orice bloc *PL/SQL* executat pe server sau pe client poate apela o rutină externă.
- Pentru a marca apelarea unei rutine externe în programul *PL/SQL* este definit un punct de intrare (*wrapper*) care direcționează 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 conține informații referitoare la numele bibliotecii în care se găsește subprogramul extern (clauza *LIBRARY*), numele rutinei externe (clauza *NAME*) și corespondența (*limbaj* ↔ *PL/SQL*) între tipurile de date (clauza *PARAMETERS*). Ultimele versiuni renunță la clauza *AS EXTERNAL*.

- Rutinele externe (scrise în *C*) sunt compilate, apoi depuse într-o bibliotecă dinamică (*DLL – dynamic link library*) și 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 declanșat automat de către *server*. La rândul său, procesul *extproc* va încărca biblioteca identificată prin clauza *LIBRARY* și va apela rutina respectivă.
- *Oracle8i* permite utilizarea de rutine externe scrise în *Java*. De asemenea, prin utilizarea clauzei *AS LANGUAGE*, un *wrapper* poate include specificații de apelare. De fapt, aceste specificații 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 spațiu adresă diferit de cel al *server*-ului *Oracle*, în timp ce unitățile *PL/SQL* și metodele *Java* se execută în spațiul 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 poziționarea 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 direcționează spre codul extern. Spre deosebire de rutinele externe *C*, nu este necesară nici biblioteca și 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*) și are următoarea formă:  

```
{IS | AS} LANGUAGE {C | JAVA}
```
- Pentru o procedură *C* sunt date informații referitoare la numele acesteia (clauza *NAME*); *alias*-ul bibliotecii în care se găsește (clauza *LIBRARY*); opțiuni referitoare la tipul, poziția, lungimea, modul de transmitere (prin valoare sau prin referință) al

parametrilor (clauza *PARAMETERS*); posibilitatea ca rutina externă să acceseze informații despre parametri, excepții, 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 semnătura metodei (lista tipurilor parametrilor în ordinea apariției).

#### Exemplul 6.23

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

-- apelare din PL/SQL
BEGIN
  calc(100);
END;
```

- Apelarea rutinei externe poate să apară în:
  - blocuri anonime;
  - subprograme independente sau aparținând unui pachet;
  - metode ale unui tip obiect;
  - declanșatori bază de date;
  - comenzi *SQL* care apelează funcții (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 interfața *Java* standard pentru conectare la baze de date relaționale și *SQLJ* permit apelarea de blocuri *PL/SQL* din programe *Java*. *SQLJ* face posibilă incorporarea operațiilor *SQL* în codul *Java*. Standardul *SQLJ* acoperă doar operații *SQL* statice. *Oracle9i SQLJ* include extensii pentru a suporta direct *SQL* dinamic.
- Altă modalitate de a încărca programe *Java* este folosirea în *SQL\*Plus* a comenzii: *CREATE JAVA instrucțiune*.

## Bibliografie

1. Connolly T.M., Begg C.E., *Database Systems: A Practical Approach to Design, Implementation and Management*, 5th edition, Pearson Education, 2005
2. Dollinger R., Andron L., *Baze de date și gestiunea tranzacțiilor*, Editura Albastră, Cluj-Napoca, 2004
3. Oracle and/or its affiliates, *Oracle Database Concepts*, 1993, 2017
4. Oracle and/or its affiliates, *Oracle Database Performance Tuning Guide*, 2013, 2017
5. Oracle and/or its affiliates, *Oracle Database SQL Language Reference*, 1996, 2017
6. Oracle and/or its affiliates, *Oracle Database PL/SQL Language Reference*, 1996, 2017
7. Oracle and/or its affiliates, *Oracle Database Administrator's Guide*, 2001, 2010
8. Oracle and/or its affiliates, *Pro\*C/C++ Programmer's Guide*, 1996, 2014
9. Oracle University, *Oracle Database 11g: PL/SQL Fundamentals, Student Guide*, 2009
10. Popescu I., Alecu A., Velcescu L., Florea (Mihai) G., *Programare avansată în Oracle9i*, Ed. Tehnică, 2004