

2. PL/SQL - Concepte generale

Cuprins

2.1. Ce este <i>PL/SQL</i>	2
2.2. Legătura cu <i>SQL</i>	2
2.3. Caracteristicile limbajului <i>PL/SQL</i>	2
2.4. Motorul <i>PL/SQL</i>	4
2.5. Evoluție	5
2.6. Comparație cu alte limbaje.....	8
2.7. Diagrama bazei de date utilizată în exemple.....	8
2.8. De ce este utilizat <i>PL/SQL</i> ?.....	9
Bibliografie.....	13

2.1. Ce este *PL/SQL*?

- Un limbaj de programare procedural creat de compania *Oracle*.
- Asigură accesarea datelor unei baze de date și permite gruparea unei multimi de comenzi într-un bloc unic de tratare a datelor.
- Poate fi utilizat doar cu o bază de date *Oracle* sau cu un utilitar *Oracle*.
- Limbaje echivalente dezvoltate de alți producători

2.2. Legătura cu *SQL*

- *SQL (Structured Query Language)*
 - Este un limbaj non-procedural, denumit și limbaj declarativ, care permite programatorilor să se axeze preponderent pe input/output și mai puțin pe pașii programului.
 - Este un limbaj *4GL (fourth-generation-programming language)*, un limbaj care este mai apropiat de limbajul natural, decât de limbajul de programare.
 - A fost standardizat de *American National Standards Institute (ANSI)*.
- *PL/SQL (Procedural Language/Structured Query Language)*
 - Reprezintă extensia procedurală a limbajului *SQL*.
 - Include atât instrucțiuni *SQL* pentru prelucrarea datelor și pentru gestiunea tranzacțiilor, cât și instrucțiuni proprii.
 - Este un limbaj *3GL (third-generation programming language)*.

Clasificarea limbajelor

2.3. Caracteristicile limbajului *PL/SQL*

- Este integrat cu *server-ul Oracle* și cu utilitarele *Oracle*.
- Este puternic integrat cu *SQL*:
 - permite utilizarea tuturor comenziilor *SQL* de prelucrare a datelor, de control al tranzacțiilor, a funcțiilor *SQL*, a operatorilor și pseudo-coloanelor;
 - suportă tipurile de date *SQL*;
 - permite atât *SQL Static* (textul complet al comenzi este cunoscut la momentul compilării), cât și *SQL Dynamic* (textul complet al comenzi este cunoscut la *run time*);
 - permite procesarea setului de rezultate al unei cereri *SQL* linie cu linie.

- Extinde *SQL* prin construcții specifice limbajelor procedurale:
 - definirea constantelor și a variabilelor;
 - declararea tipurilor;
 - definirea și utilizarea cursoarelor;
 - utilizarea structurilor de control;
 - definirea procedurilor și funcțiilor;
 - modularizarea programelor (subprograme, pachete);
 - detectarea și gestiunea erorilor de execuție și a situațiilor excepționale;
 - introducerea tipurilor obiect și a metodelor etc.
- Permite definirea și utilizarea declanșatorilor.
- Asigură securitatea informației.
- Mărește performanța aplicației:
 - permite înglobarea mai multor instrucțiuni într-un singur bloc și trimiterea acestui către baza de date, reducând-se astfel traficul dintre aplicație și baza de date.
- Are suport pentru dezvoltarea aplicațiilor *Web*.
 - Aplicațiile *Web* scrise în *PL/SQL* sunt proceduri stocate care interacționează cu un *browser Web* printr-un protocol *HTTP*.
 - Pentru a facilita dezvoltarea aplicațiilor *Web* sistemul furnizează o serie de pachete predefinite (de exemplu, pachetul *UTL_SMTP* poate fi utilizat pentru a trimite un *mail* dintr-o procedură stocată *PL/SQL*)
 - Pachetele pot fi folosite împreună cu *Oracle Internet Application Server* și *WebDB*.
 - *PL/SQL Server Pages (PSPs)* permit dezvoltarea paginilor *Web* cu conținut dinamic:
 - scripturile *PL/SQL* pot fi integrate în codul sursă *HTML*;
 - scripturile rulează atunci când un *client Web* solicită o pagină;
 - un script poate accepta parametrii, interoga sau actualiza baza de date și afișa rezultatele într-o pagină personalizată.
- Este portabil
 - Programele *PL/SQL* pot rula pe orice suport pe care există un *server Oracle*.
 - Nu depinde de platformă sau de sistemul de operare.
 - Se pot crea programe, pachete sau librării portabile care pot fi utilizate cu bazele de date *Oracle* în medii diferite.

2.4. Motorul *PL/SQL*

- Compilează și execută codul *PL/SQL*.
- Se află pe *server-ul Oracle* sau în unele utilitare *Oracle* (de exemplu, *Oracle Forms*).
 - Unele utilitare *Oracle* au propriul motor *PL/SQL*. Acesta este independent față de motorul *PL/SQL* de pe *server-ul Oracle*. Utilitarul transmite blocurile către motorul *PL/SQL* local care execută toate comenziile procedurale.
- Indiferent de mediu, motorul *PL/SQL* execută comenziile procedurale, dar trimite comenziile *SQL* către motorul *SQL* de pe *server-ul Oracle*.
 - Dacă nu ar fi incluse în blocuri *PL/SQL* comenziile *SQL* ar fi procesate separat, fiecare implicând câte un apel la *server-ul Oracle*.
- Comenziile procedurale pot fi executate pe stația *client* fără interacțiune cu *server-ul Oracle* sau în întregime pe *server-ul Oracle*.
 - Dacă utilitarul *Oracle* are motor *PL/SQL*, iar blocul *PL/SQL* nu conține comenzi *SQL*, atunci acesta este procesat local.



- ❖ Un subprogram *PL/SQL* stocat este un obiect al bazei de date și poate fi accesat de orice aplicație.
- ❖ Apelurile procedurilor care sunt stocate pe *server* sunt trimise pentru procesare motorului *PL/SQL* de pe *server*.
- ❖ Subprogramele *PL/SQL* (proceduri, funcții) declarate într-o aplicație *Developer Suite* (de exemplu, *Oracle Forms*) sunt diferite de cele stocate în baza de date. Acestea sunt procesate local.



- 1. Unde se află motorul *SQL*?**
- 2. Unde se află motorul *PL/SQL*?**
- 3. Se poate scrie cod *PL/SQL* folosind *SQL*Plus* sau *SQL Developer*?**
- 4. Utilitarul *SQL*Plus* are motor *PL/SQL*? Dar utilitarul *SQL Developer*?**
- 5. Comenziile *PL/SQL* pot fi incluse în cod *SQL*? Dar invers?**
- 6. *SQL* transmite *server-ului* de baze de date ce să facă (declarativ), nu cum să facă. (adevărat/fals)**
- 7. *PL/SQL* transmite *server-ului* de baze de date cum să facă (procedural). (adevărat/fals)**

2.5. Evoluție

În 1977 Larry Ellison împreună cu câțiva prieteni înființează compania *Software Development Laboratories*.

În 1979 numele companiei este schimbat în *Relational Software, Inc.* Primul produs lansat de această companie a fost o bază de date relațională denumită *Oracle*. Deși era prima versiune, din motive de marketing a fost denumită *Oracle V2*. Pentru accesul la date era utilizat *SQL* și nu permitea tranzacții. În acel an *Relational Software, Inc* era singura companie care producea o bază de date compatibilă cu *SQL*.

În 1982 numele companiei a fost schimbat în *Oracle Corporation*. Limbajul ales pentru baza de date a fost modelat pe *ADA*, un limbaj de programare orientat obiect de nivel înalt care este extins din *Pascal*. *Oracle* a denumit acest limbaj *PL/SQL*. Fiind descendant al limbajelor *ADA* și *Pascal*, *PL/SQL* este un limbaj bazat pe structură de blocuri.

- *Oracle V3* (1983)
 - funcționalitățile *commit* și *rollback* pentru tranzacții
- *Oracle V4* (1984)
 - consistență la citire
- *Oracle V5* (1985)
 - arhitectura *client-server*
 - suport pentru cererile distribuite
- *Oracle V6* (1988) – prima versiune *Oracle* care suportă *PL/SQL*
 - limbaj limitat, bazat pe *script-uri*, nu pe proceduri stocate
 - gestiunea erorilor primitivă
 - *SQL* complet integrat
 - blocare la nivel de linie
- *Oracle V7* (1992)
 - proceduri stocate
 - integritate referențială
 - *trigger-i*
 - tablourile *PL/SQL*
 - pachetul *UTL_FILE* pentru a putea accesa fișierele sistemului de operare
 - pachetul *DBMS_SQL* pentru *SQL* dinamic
 - *Oracle Advanced Queuing*
 - *Oracle Enterprise Manager*
 - distribuire

- *Oracle V8* (1997)
 - orientarea obiect (tipuri obiect și metode)
 - rutine externe
 - suport pentru cereri stătești
 - tipul *LOB*
 - tipuri colecție (vectori și tablouri imbricate)
 - aplicații multimedia
 - capacitatea de a realiza apele *HTTP* din baza de date
- *Oracle V8i* (1999) – devine bază de date comercială
 - *drop column*
 - partiționare și subpartiționare
 - *XML*
 - *WebDB*
 - funcții analitice în *SQL*
 - indexare *online*
 - tranzacții autonome
 - rutine externe *Java*
 - *SQL* dinamic nativ
 - operații *bulk bind*
 - *trigger*-i bază de date
 - îmbunătățire a performanței *SQL* și *PL/SQL*
 - baza de date încorporează *Java Virtual Machine* (*Oracle JVM* cunoscut și sub numele de *Aurora*)
- *Oracle V9i* (2001)
 - tabele externe
 - compilare nativă (*PL/SQL* la *C*)
 - *Oracle Streams*
 - *XML DB*
 - îmbunătățiri la nivel de *data warehouse* și *BI*
 - extindere și îmbunătățiri pentru *SQL analytic*
 - comenzi și expresii *CASE*
 - tipuri noi de date în *SQL* și *PL/SQL* (*DATETIME*, *TIMESTAMP*, colecții pe mai multe niveluri)
 - ierarhii de tipuri și subtipuri

- funcții tabel
- expresii CURSOR
- claselor *Java* în baza de date
- *Real Application Cluster*
- *Oracle V10g* (2003)
 - *recyclebin* (recuperare obiecte șterse)
 - permanentizări asincrone
 - criptarea transparentă a datelor (criptare automată la nivel de coloană)
 - îmbunătățire *SQL* analitic
 - *SQL Tuning Advisor*
 - îmbunătățire *OLAP*
 - *SQL Model Clause*
 - proceduri stocate .Net
- *Oracle V11g* (2007)



- ❖ Procedurile *PL/SQL* se execută mai rapid prin compilarea lor într-un cod nativ.
 - Procedurile sunt transluate în cod *C*, compilate cu ajutorul unui compilator *C* și apoi automat preluate în procese *Oracle*.
 - Această tehnică, care nu cere restaurarea bazei de date, poate fi utilizată pentru proceduri și pachete *Oracle*.
- ❖ *Oracle* furnizează soluții (interfețe *client-side* și *server-side*, utilitare, *JVM* integrată cu *server-ul Oracle*) dezvoltatorilor de aplicații pentru crearea, gestionarea și exploatarea aplicațiilor *Java*.
- ❖ Procedurile *Java* stocate pot fi apelate dintr-un pachet *PL/SQL*, iar proceduri *PL/SQL* existente pot fi invocate din proceduri *Java*. Datele *SQL* pot fi accesate prin două interfețe (*API*): *JDBC* și *SQLJ*. Astfel:
 - pentru a invoca o procedură *Java* din *SQL* este nevoie de interfața *Java Stored Procedures*,
 - pentru a invoca dinamic comenzi *SQL* complexe este folosit *JDBC*,
 - pentru a utiliza comenzi *SQL* statice, simple (referitoare la un tabel ale cărui coloane sunt cunoscute) dintr-un obiect *Java* este folosit *SQLJ*.

2.6. Comparație cu alte limbaje

	PL/SQL	C	JAVA
Necesită BD sau utilitar Oracle	da	nu	nu
Orientat obiect	câteva caracteristici	nu	da
Performanță asupra BD Oracle	foarte eficient	mai puțin eficient	mai puțin eficient
Portabil pe diferite SO	da	oarecum	da
Ușor de învățat	relativ ușor	mai dificil	mai dificil

2.7. Diagrama bazei de date utilizată în exemple

- Gestiunea activității unei companii comerciale
[Vezi diagrame curs](#)
- Schema simplificată

○ Entități

DEPOZITE(id_depozit#, denumire, adresa, oras, judet, orar, capacitate, valoare,
 id_director, id_tara)

SECTOARE(id_sector#, descriere, id_depozit)

ZONE (id_zona#, descriere, capacitate_maxima, capacitate_folosita, id_sector)

PRODUSE (id_produs#, denumire, descriere, stoc_curent, stoc_impus, pret_unitar,
 greutate, volum, tva, id_zona, id_um, id_categoria, data_crearii,
 data_modificarii, activ)

CARACTERISTICI (#id_caracteristica, denumire, descriere)

UNITATI_MASURA(id_um#, denumire, descriere)

CATEGORII (id_categoria#, denumire, nivel, id_parinte)

FACTURI(id_factura#, data, status, id_casa, id_client, id_adresa_livrare,
 id_adresa_facturare, id_tip_livrare, id_tip_plata, interval_livrare)

CASE(id_casa#, nume, serie, parola)

TIP_LIVRARE(id_tip_livrare#, denumire, tarif, *id_firma_t*)

ADRESE(id_adresa#, strada, oras, tara, cod_postal, id_client)

TIP_PLATA(id_tip_plata#, cod, descriere)

CLIENTI(id_client#, telefon, email, tip, oras, data_crearii, data_modificarii)

- Subentități

PERSOANE_FIZICE(id_client_f#, nume, prenume, cnp)

PERSOANE_JURIDICE(id_client_j#, denumire, persoana_contact, cui, cont, banca, cod_fiscal, numar_inregistrare)

- Tabele asociative

CLIENTI_AU_PRET_PREFERENTIAL(id_pret_pref#, id_categorie, id_client_j, discount, data_in, data_sf)

PRODUSE_AU_CARACTERISTICI (id_produs#, id_caracteristica#, valoare)

FACTURI_CONTIN_PRODUSE (id_factura#, id_produs#, cantitate, pret_facturare)

2.8. De ce este utilizat *PL/SQL*?

- În practică există numeroase situații în care *SQL* se dovedește a fi limitat.
- Exemplu – vezi explicații curs
 - Clienții companiei pot avea prețuri personalizate în funcție de anumite criterii. Comenzile se realizează *online*, clienții consultând cataloage cu prețuri personalizate.
 - În cazul în care prețul personalizat se calculează raportat la categoriile de produse este nevoie de o clasificare a clienților în funcție de categorie și numărul de produse cumpărate din categoria respectivă.
 - Tabelul *CLASIFIC_CLIENTI* (*id_client, id_categorie, nr_produse, clasificare*) conținea deja o clasificare a clienților.
 - Se decide o nouă clasificare a acestora conform tabelului de mai jos.

id_categorie	clasificare	număr produse cumpărate
1	A	> 1000
	B	≥ 500
	C	≥ 0
2	A	> 2000
	B	≥ 1000
	C	≥ 200
	D	≥ 0

- Soluție *SQL* posibilă

Varianta1

```
UPDATE clasific_clienti
SET    clasificare = 'A'
WHERE   nr_produse > 1000
AND     id_categorie = 1;
```

```
UPDATE clasific_clienti
SET    clasificare = 'B'
WHERE   nr_produse BETWEEN 500 AND 1000
AND     id_categorie = 1;
...
```

- Câte comenzi *UPDATE* sunt necesare pentru datele din tabelul anterior?
- Este eficientă o astfel de abordare?
- Există alte metode de abordare utilizând *SQL*?

Varianta2

```
UPDATE clasific_clienti
SET    clasificare = CASE WHEN nr_produse>1000
                           THEN 'A'
                           WHEN nr_produse BETWEEN 500
                           AND 1000 THEN 'B'
                           ELSE 'C' END
WHERE id_categorie = 1;
```

...

Varianta3

```
UPDATE clasific_clienti
SET    clasificare =
      CASE WHEN nr_produse>1000
            AND id_categorie=1 THEN 'A'
            WHEN nr_produse BETWEEN 500 AND 1000
            AND id_categorie = 1 THEN 'B'
            WHEN nr_produse BETWEEN 0 AND 499
            AND id_categorie=1 THEN 'C'
            WHEN nr_produse>2000
            AND id_categorie=2 THEN 'A'
            WHEN nr_produse BETWEEN 1000 AND 2000
            AND id_categorie = 2 THEN 'B'
            WHEN nr_produse BETWEEN 200 AND 999
            AND id_categorie=2 THEN 'C'
            ELSE 'D' END;
```

4. Ce se întâmplă atunci când există mai multe categorii de produse și mai multe niveluri de clasificare pentru fiecare categorie?

- Soluție *PL/SQL* posibilă

Varianta 1

```

DECLARE
    CURSOR info IS
        SELECT id_client, id_categorie, nr_produse
        FROM   clasific_clienti;

    v_clasific clasific_clienti.clasificare%type;
BEGIN
    FOR i IN info LOOP
        CASE WHEN i.nr_produse>1000
              AND i.id_categorie=1
              THEN v_clasific := 'A';
              WHEN i.nr_produse BETWEEN 500 AND 1000
              AND i.id_categorie = 1
              THEN v_clasific := 'B';
              WHEN i.nr_produse BETWEEN 0 AND 499
              AND i.id_categorie=1
              THEN v_clasific := 'C';
              WHEN i.nr_produse>2000
              AND i.id_categorie=2
              THEN v_clasific := 'A';
              WHEN i.nr_produse BETWEEN 1000 AND 2000
              AND i.id_categorie = 2
              THEN v_clasific := 'B';
              WHEN i.nr_produse BETWEEN 200 AND 999
              AND i.id_categorie=2
              THEN v_clasific := 'C';
              ELSE v_clasific := 'D';
        END CASE;
        UPDATE clasific_clienti
        SET   clasificare = v_clasific
        WHERE id_client = i.id_client
        AND   id_categorie = i.id_categorie;
    END LOOP;
END;
/

```

1. De câte ori se execută comanda *UPDATE*?
2. Este eficientă o astfel de abordare?
3. Cât timp este blocată resursa?
4. Ar fi utilă o cheie primară artificială?

Varianta2

```

DECLARE
    CURSOR info IS
        SELECT id_client, id_categorie, nr_produse
        FROM   clasific_clienti
    FOR UPDATE;
    v_clasific clasific_clienti.clasificare%type;
BEGIN
    FOR i IN info LOOP
        CASE WHEN i.nr_produse>1000
            AND i.id_categorie=1
            THEN v_clasific := 'A';
        WHEN i.nr_produse BETWEEN 500 AND 1000
            AND i.id_categorie = 1
            THEN v_clasific := 'B';
        WHEN i.nr_produse BETWEEN 0 AND 499
            AND i.id_categorie=1
            THEN v_clasific := 'C';
        WHEN i.nr_produse>2000
            AND i.id_categorie=2
            THEN v_clasific := 'A';
        WHEN i.nr_produse BETWEEN 1000 AND 2000
            AND i.id_categorie = 2
            THEN v_clasific := 'B';
        WHEN i.nr_produse BETWEEN 200 AND 999
            AND i.id_categorie=2
            THEN v_clasific := 'C';
        ELSE v_clasific := 'D';
        END CASE;
        UPDATE clasific_clienti
        SET   clasificare = v_clasific
        WHERE CURRENT OF info;
    END LOOP;
END;
/

```

5. Este mai eficientă această abordare? Cât timp este blocată resursa?
6. Există și alte metode de rezolvare a problemei?
7. Numărul de produse cumpărate de client trebuie să fie în permanență actualizat și să corespundă situației prezente. Se poate realiza *realtime* acest lucru cu *SQL*?

Bibliografie

1. *Programare avansată în Oracle9i*, I. Popescu, A. Alecu, L. Velcescu, G. Florea (Mihai), Ed. Tehnică (2004)
2. *Oracle Database PL/SQL Language Reference 11g Release 2*, Oracle Online Documentation (2012)
3. *Oracle Database 11g: PL/SQL Fundamentals, Student Guide*, Oracle University (2009)
4. *A Mini-History of Oracle and PL/SQL*, L. Cunningham (2012)
(http://www.dba-oracle.com/t_edb_pl_sql_features_release.htm)
5. *Oracle Database*, Wikipedia (2012)
(http://en.wikipedia.org/wiki/Oracle_Database)

3. PL/SQL – Blocuri. Variabile. Instrucțiuni.

CUPRINS

3. PL/SQL – Blocuri. Variabile. Instrucțiuni	1
3.1. Limbajul <i>PL/SQL</i>	2
3.2. Structura unui bloc <i>PL/SQL</i>	2
3.2.1. Separator pentru instrucțiuni	3
3.2.2. Comentarii.....	3
3.3. Operatori.....	4
3.4. Variabile	4
3.5. Blocuri <i>PL/SQL</i>	5
3.6. Comenzi <i>SQL</i> în <i>PL/SQL</i>	6
3.6.1. Comanda <i>SELECT ... INTO</i>	7
3.6.2. Comenzile <i>INSERT, UPDATE, DELETE</i>	8
3.7. Instrucțiuni <i>PL/SQL</i>	9
3.7.1. Instrucțiunea de atribuire	9
3.7.2. Instrucțiunea condițională <i>IF</i>	9
3.7.3. Instrucțiunea condițională <i>CASE</i>	11
3.7.4. Instrucțiunea iterativă <i>LOOP</i>	13
3.7.5. Instrucțiunea iterativă <i>WHILE</i>	13
3.7.6. Instrucțiunea iterativă <i>FOR</i>	14
3.7.7. Instrucțiunea vidă	15
3.7.8. Instrucțiunea de salt <i>EXIT</i>	18
3.7.9. Instrucțiunea de salt <i>CONTINUE</i>	18
3.7.10. Instrucțiunea de salt <i>GOTO</i>	19
Bibliografie.....	19

3.1. Limbajul *PL/SQL*

- Atât *PL/SQL*, cât și *server-ul Oracle* utilizează același spațiu de memorie și prin urmare nu apar supraîncărcări datorate comunicațiilor dintre acestea.
- Este un limbaj cu structură de blocuri.
- Pentru modularizarea codului *PL/SQL* se pot folosi
 - blocuri anonime
 - subprograme (proceduri și funcții)
 - funcțiile pot fi invocate direct utilizând comenzi *SQL*
 - pachete
 - *trigger-i*
 - sunt un tip special de proceduri *PL/SQL* care se execută automat la apariția unui anumit eveniment.
- Blocuri anonime versus subprograme stocate

Blocuri anonime	Subprograme stocate
Blocuri PL/SQL fără nume	Blocuri PL/SQL cu nume
Compilate de fiecare dată când aplicația este executată	Compilate o singură dată
Nu sunt stocate în BD	Sunt stocate în BD
Nu pot fi invocate de alte aplicații	Pot fi invocate de alte aplicații
Nu întorc valori	Functiile trebuie să întoarcă o valoare
Nu acceptă parametrii	Acceptă parametrii

3.2. Structura unui bloc *PL/SQL*

- Blocul este unitatea de bază a unui program *PL/SQL*.
- Mai este denumit și modul.
- Blocul *PL/SQL* conține 3 secțiuni
 - secțiunea declarativă (optională)
 - constante și variabile
 - tipuri de date locale

- cursoare
- excepții definite de utilizator
- subprograme locale (vizibile doar în bloc)
- secțiunea executabilă (obligatorie)
 - instrucțiuni *SQL* pentru prelucrarea datelor
 - instrucțiuni *PL/SQL*
 - trebuie să conțină măcar o instrucțiune
- secțiunea de tratare a excepțiilor (optională)
 - instrucțiuni efectuate atunci când apare o numită excepție/eroare
- Sintaxa generală

```
[<<nume_bloc>>]
[DECLARE
    instrucțiuni de declarare]
BEGIN
    instrucțiuni executabile SQL sau PL/SQL
[EXCEPTION
    tratarea erorilor/excepțiilor]
END [nume_bloc];
```

- Dacă blocul *PL/SQL* este executat fără erori va apărea mesajul:
anonymous block completed



Într-un bloc *PL/SQL* sunt permise instrucțiuni *SQL*Plus*?

3.2.1. Separator pentru instrucțiuni

- Caracterul „;“ este separator pentru instrucțiuni.

3.2.2. Comentarii

- Comentariile sunt ignorate de compilatorul *PL/SQL*:
 - pe o singură linie
 - sunt prefixate de simbolurile „--“
 - încep în orice punct al liniei și se termină la sfârșitul acesteia
 - pe mai multe linii
 - sunt delimitate de simbolurile „/*“ și „*/“

3.3. Operatori

- Operatorii din *PL/SQL* sunt identici cu cei din *SQL*.
- În *PL/SQL* este introdus operatorul „ ** “ pentru ridicare la putere.

Exemplul 3.1

```
SELECT POWER(3,2)
FROM   DUAL;
```

```
BEGIN
    DBMS_OUTPUT.PUT_LINE(POWER(3,2));
END;
```

```
BEGIN
    DBMS_OUTPUT.PUT_LINE(3**2);
END;
```



Care secvență poate fi executată local? În ce condiții?

3.4. Variabile

- Stochează datele în același format binar intern ca și baza de date, astfel nefiind necesare conversii suplimentare.
- Pot fi declarate doar în zona declarativă a unui bloc, unde pot fi și inițializate.

Exemplul 3.2

```
DECLARE
    nume_utilizator    VARCHAR2(30);
    data_creare_cont   DATE DEFAULT SYSDATE;
    numar_credite      NUMBER(4) NOT NULL := 1000;
    limita_inferioara_credite CONSTANT NUMBER := 100;
    limita_superioara_credite NUMBER := 5000;
    este_valid          BOOLEAN := TRUE;
```

- Fiecare variabilă se declară individual.

```
DECLARE
    -- declaratie
    -- incorrecta
    i, j INTEGER;
BEGIN
    NULL;
END;
```

```
DECLARE
    /*declaratie
    corecta*/
    i    INTEGER;
    j    INTEGER;
BEGIN
    NULL;
END;
```

- Îi se pot atribui valori noi și pot fi utilizate în zona executabilă a blocului.
- Pot fi transmise ca parametruii subprogramelor *PL/SQL*.
- Pot fi declarate pentru a menține rezultatul obținut de un subprogram *PL/SQL*.
- Sunt vizibile în blocul în care sunt declarate și în toate subblocurile declarate în acesta.
- Dacă o variabilă nu este declarată local în bloc, atunci este căutată în secțiunea declarativă a blocurilor care includ blocul respectiv.

Exemplul 3.3

```

DECLARE
    v_principal VARCHAR2(50);
BEGIN
    v_principal := 'variabila din blocul principal';

    DECLARE
        v_secundar VARCHAR2(50) :=
            'variabila din blocul secundar';
    BEGIN
        DBMS_OUTPUT.PUT_LINE('<<Bloc Secundar>>');
        DBMS_OUTPUT.PUT_LINE('Folosesc '||v_principal);
        DBMS_OUTPUT.PUT_LINE('Folosesc '||v_secundar);
        v_secundar := 'Modific '||v_secundar;
        v_principal := 'Modific '||v_principal;
        DBMS_OUTPUT.PUT_LINE(v_secundar);
        DBMS_OUTPUT.PUT_LINE(v_principal);
    END;

    DBMS_OUTPUT.PUT_LINE('<<Bloc Principal>>');
    DBMS_OUTPUT.PUT_LINE(v_secundar);
    DBMS_OUTPUT.PUT_LINE(v_principal);
END;

```



Care comandă va genera o eroare?

3.5. Blocuri *PL/SQL***Exemplul 3.4**

- Bloc fără secțiune declarativă, fără secțiune de tratare a excepțiilor.

```

BEGIN
    DBMS_OUTPUT.PUT_LINE('SGBD');
END;

```

- Bloc cu secțiune declarativă, fără secțiune de tratare a excepțiilor.

```
DECLARE
    v_data DATE := SYSDATE;
BEGIN
    DBMS_OUTPUT.PUT_LINE(v_data);
END;
```

- Bloc cu secțiune declarativă, cu secțiune de tratare a excepțiilor.

```
DECLARE
    x NUMBER := &p_x;
    y NUMBER := &p_y;
BEGIN
    DBMS_OUTPUT.PUT_LINE(x/y);
EXCEPTION
    WHEN ZERO_DIVIDE THEN
        DBMS_OUTPUT.PUT_LINE('Nu poti sa imparti la 0!');
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('Alta eroare!');
END;
```

3.6. Comenzi SQL în PL/SQL

- Comenzi *SQL* care pot fi utilizate direct în *PL/SQL*
 - *LMD* (*SELECT, INSERT, UPDATE, DELETE, MERGE*)
 - Comanda *SELECT* poate fi utilizată doar cu clauza *INTO*
 - *LCD* (*COMMIT, SAVEPOINT, ROLLBACK*)
- Comenzi *SQL* care nu pot fi utilizate direct în *PL/SQL*
 - *LDD* (*CREATE, ALTER, DROP*)
 - *LCD* (*GRANT, REVOKE*)
 - Aceste comenzi nu pot fi folosite direct în *PL/SQL* deoarece sunt construite și executate la *runtime* (sunt dinamice). De aceea pot fi utilizate în *PL/SQL* doar cu *SQL Dynamic*.
 - *SQL Static* cuprinde comenzi care sunt stabilite la momentul în care programul este compilat. Acestea pot fi utilizate direct în *PL/SQL*.

3.6.1. Comanda *SELECT ... INTO*

Exemplul 3.5

```

DECLARE
    v_clasificare  clasific_clienti.clasificare%TYPE;
    v_categorie     clasific_clienti.id_categoria%TYPE;
    v_client        clasific_clienti.id_client%TYPE
                    := &p_client;
BEGIN
    SELECT clasificare, id_categoria
    INTO   v_clasificare, v_categorie
    FROM   clasific_clienti
    WHERE  id_client = v_client;
    DBMS_OUTPUT.PUT_LINE(v_categorie || ' '
                         || v_clasificare);
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        DBMS_OUTPUT.PUT_LINE('Nicio linie!');
    WHEN TOO_MANY_ROWS THEN
        DBMS_OUTPUT.PUT_LINE('Mai multe linii!');
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('Alta eroare!');
END;

```



Comanda *SELECT ... INTO* trebuie să obțină exact o singură înregistrare?

Comanda *SELECT* poate fi utilizată în PL/SQL fără clauza *INTO*?

Exemplul 3.6

```

VARIABLE h_clasificare  VARCHAR2
VARIABLE h_categorie     NUMBER

BEGIN
    SELECT clasificare, id_categoria
    INTO   :h_clasificare, :h_categorie
    FROM   clasific_clienti
    WHERE  id_client = 82;
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        DBMS_OUTPUT.PUT_LINE('Clientul nu exista!');
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('Alta eroare!');
END;
/

PRINT h_clasificare
PRINT h_categorie

```



Comanda *SELECT* poate fi utilizată într-o procedură *Microsoft T-SQL* fără clauza *INTO*?



În *Microsoft T-SQL* comanda *SELECT* cu clauza *INTO* funcționează la fel ca și în *PL/SQL*?



Comanda *SELECT* poate fi utilizată într-o procedură *MySQL* fără clauza *INTO*?



Ce opțiuni permite comanda *SELECT ... INTO* în *MySQL*?



În cadrul acestui curs vor fi considerate corecte și punctate doar soluțiile implementate în *PL/SQL*.

3.6.2. Comenzile *INSERT, UPDATE, DELETE*

Exemplul 3.7

```
BEGIN
    DELETE FROM clasific_clienti WHERE id_client=209;
    INSERT INTO clasific_clienti VALUES (209,2,1,null);
    UPDATE clasific_clienti
    SET clasificare = 'D'
    WHERE id_client = 209;
    COMMIT;
END;
```

Exemplul 3.8 – **vezi curs**



Un bloc *PL/SQL* poate conține mai multe comenzi *COMMIT, SAVEPOINT* sau *ROLLBACK*?

3.7. Instrucțiuni PL/SQL

- Instrucțiunea de atribuire (`:=`)
- Instrucțiuni condiționale (`IF`, `CASE`)
- Instrucțiuni iterative (`LOOP`, `WHILE`, `FOR`)
- Instrucțiuni de salt (`GOTO`, `EXIT`, `CONTINUE`)
- Instrucțiunea vidă (`NULL`)

3.7.1. Instrucțiunea de atribuire

variabila `:=` expresie;

- Variabilele care sunt declarate *NOT NULL* trebuie inițializate la declarare.
 - Codul din partea stângă a exemplului de mai jos va genera eroarea *PLS-00218: a variable declared NOT NULL must have an initialization assignment*

Exemplul 3.9

```

DECLARE
    x NUMBER(2) NOT NULL;
BEGIN
    x:=2;
    DBMS_OUTPUT.PUT_LINE(x);
END;

```

```

DECLARE
    x NUMBER(2) NOT NULL :=2;
BEGIN
    DBMS_OUTPUT.PUT_LINE(x);
END;

```

3.7.2. Instrucțiunea condițională IF

```

IF expresie_booleană
    THEN comandă [comandă] ...
[ELSIF expresie_booleană
    THEN comandă [comandă] ...]
[ELSIF expresie_booleană
    THEN comandă [comandă] ...]
[ELSE comandă [comandă] ...]
END IF;

```

- Comenzile din instrucțiune sunt executate dacă expresia booleană corespunzătoare are valoare *TRUE*. În caz contrar (expresia booleană are valoarea *FALSE* sau *NULL*), secvența nu este executată.
- Instrucțiunea *IF* poate conține mai multe clauze *ELSIF*, dar o singură clauză *ELSE*. Aceasta se referă la ultima clauză *ELSIF*.

Exemplul 3.10

```

DECLARE
    v_nr NATURAL;
    v_clasificare CHAR(1) := UPPER('&p_clasificare');
BEGIN
    SELECT COUNT(*) INTO v_nr
    FROM   clasific_clienti
    WHERE  clasificare = v_clasificare;

    IF v_nr=0
    THEN
        DBMS_OUTPUT.PUT_LINE('Nu exista clienti de ' ||
                             'tipul ' || v_clasificare);
    ELSE
        DBMS_OUTPUT.PUT_LINE('Există ' || v_nr ||
                             ' clienti de tipul ' || v_clasificare);
    END IF;
END;

```

Exemplul 3.11

```

DECLARE
    v_nr NATURAL;
    v_clasificare CHAR(1) := UPPER('&p_clasificare');
BEGIN
    SELECT COUNT(*) INTO v_nr
    FROM   clasific_clienti
    WHERE  clasificare = v_clasificare
    AND    id_categorie = 1;

    IF v_nr=0
    THEN
        DBMS_OUTPUT.PUT_LINE('Nu exista clienti de ' ||
                             'tipul ' || v_clasificare);

    ELSE
        IF v_nr =1
        THEN
            DBMS_OUTPUT.PUT_LINE('Există 1 client ' ||
                                 'de tipul ' || v_clasificare);
        ELSE
            DBMS_OUTPUT.PUT_LINE('Există ' || v_nr ||
                                 ' clienti de tipul ' || v_clasificare);
        END IF;
    END IF;
END;

```

Exemplul 3.12 - vezi curs

3.7.3. Instrucțiunea condițională CASE

```
[<<eticheta>>]
CASE selector
    WHEN valoare_1_selector THEN secvență_comenzi_1;
    WHEN valoare_2_selector THEN secvență_comenzi_2;
    ...
    WHEN valoare_n_selector THEN secvență_comenzi_n;
    [ELSE secvență_comenzi;]
END CASE [eticheta];
```

- *Selectorul* este o expresie a cărei valoare este evaluată o singură dată și este utilizată pentru a selecta una dintre alternativele specificate prin clauzele *WHEN*.
 - Poate avea orice tip *PL/SQL*, cu excepția tipurilor *BLOB*, *BFILE* și tipuri definite de utilizator.
- Dacă valoarea selectorului este egală cu *valoare_k_selector*, atunci sunt executate comenziile cuprinse în *secvență_comenzi_k* și comanda *CASE* se încheie.
 - *Valoare_k_selector* poate avea orice tip *PL/SQL*, cu excepția tipurilor *BLOB*, *BFILE* și tipuri definite de utilizator.
- Secvența de comenzi din clauza *ELSE* este executată doar dacă selectorul nu are niciuna dintre valorile cuprinse în clauzele *WHEN*.
 - Clauza *ELSE* este opțională.
 - Dacă această clauză lipsește și selectorul nu are niciuna dintre valorile specificate în clauzele *WHEN*, atunci pare eroarea *CASE_NOT_FOUND*.

Exemplul 3.13

```
DECLARE
    v_nr NATURAL;
    v_clasificare CHAR(1) := UPPER('&p_clasificare');
BEGIN
    SELECT COUNT(*) INTO v_nr
    FROM   clasific_clienti
    WHERE  clasificare = v_clasificare
    AND    id_categorie = 1;

    CASE v_nr
        WHEN 0 THEN
            DBMS_OUTPUT.PUT_LINE('Nu exista clienti de ' ||
                                'tipul ' || v_clasificare);
        WHEN 1 THEN
            DBMS_OUTPUT.PUT_LINE('Există 1 client ' ||
                                'de tipul ' || v_clasificare);
    END CASE;
END;
```

```

    ELSE
        DBMS_OUTPUT.PUT_LINE('Exista ' || v_nr ||
                             ' clienti de tipul '|| v_clasificare);
    END CASE;
END;

```

- Comanda *CASE* permite o formă alternativă:

```

[<<eticheta>>]
CASE
    WHEN expresie_booleană_1 THEN secvență_comenzi_1;
    WHEN expresie_booleană_2 THEN secvență_comenzi_2;
    ...
    WHEN expresie_booleană_n THEN secvență_comenzi_n;
    [ELSE secvență_comenzi;]
END CASE [eticheta];

```

- Selectorul lipsește.
- Fiecare clauză *WHEN* conține o expresie booleană.
- Dacă expresie booleană *expresie_booleană_k* are valoarea *TRUE*, atunci sunt executate comenzile cuprinse în *secvență_comenzi_k* și comanda *CASE* se încheie.
- Secvența de comenzi din clauza *ELSE* este executată doar dacă nicio expresie booleană din clauzele *WHEN* nu are valoare *TRUE*.
- Și în acest caz clauza *ELSE* este optională. Dacă această clauză lipsește și nicio expresie booleană din clauzele *WHEN* nu are valoare *TRUE*, atunci pare eroarea *CASE_NOT_FOUND*.

Exemplul 3.14 - [vezi curs](#)



Nu confundați comanda *CASE* din *PL/SQL* cu expresia *CASE* din *SQL*.

Exemplul 3.15 - [vezi curs](#)

Expresia *CASE* are sintaxa similară comenzi *CASE*, dar:

- clauzele *WHEN* nu se termină prin caracterul „;”;
- în clauzele *WHEN* nu se realizează atribuiri;
- clauza *END* nu include cuvântul cheie *CASE*.

Exemplul 3.16 - [vezi curs](#)

3.7.4. Instrucțiunea iterativă ***LOOP***

```
LOOP
    sevență_de_comenzi;
END LOOP;
```

- Este denumită ciclare simplă.
- Comenziile incluse între cuvintele cheie ***LOOP*** și ***END LOOP*** sunt executate cel puțin o dată.
- Pentru a nu cicla la infinit trebuie utilizată comanda ***EXIT***.

Exemplul 3.17

```
DECLARE
    cod_ascii NUMBER := ASCII('A');
BEGIN
    LOOP
        DBMS_OUTPUT.PUT(CHR(cod_ascii) || ' ');
        cod_ascii := cod_ascii + 1;
        -- EXIT;
        EXIT WHEN cod_ascii > ASCII('E');
        /* IF cod_ascii > ASCII('E') THEN EXIT;
           END IF; */
    END LOOP;

    DBMS_OUTPUT.NEW_LINE;
    DBMS_OUTPUT.PUT_LINE('Iesire cand am ajuns la ' ||
                         CHR(cod_ascii));
END;
```

- Pentru a transfera controlul iterației următoare se utilizează comanda ***CONTINUE***.

Exemplul 3.18 - **vezi curs**

3.7.5. Instrucțiunea iterativă ***WHILE***

```
WHILE condiție LOOP
    sevență_de_comenzi;
END LOOP;
```

- Este denumită ciclare condiționată.
- Comenziile incluse între cuvintele cheie ***LOOP*** și ***END LOOP*** sunt executate atât timp cât condiția are valoarea ***TRUE***.
- Condiția este evaluată la începutul fiecărei iterații.

Exemplul 3.19

```

DECLARE
    i NATURAL := 1;
BEGIN
    WHILE i<=10 LOOP
        DBMS_OUTPUT.PUT(i**2|| ' ');
        i := i + 1;
    END LOOP;

    DBMS_OUTPUT.NEW_LINE;
    DBMS_OUTPUT.PUT_LINE('Iesire cand i = '|| i );
END;
/

DECLARE
    i NATURAL := 1;
BEGIN
    WHILE i<=10 LOOP
        DBMS_OUTPUT.PUT(i**2|| ' ');
        i := i + 1;
        CONTINUE WHEN i<=5;
        DBMS_OUTPUT.NEW_LINE;
    END LOOP;

    DBMS_OUTPUT.PUT_LINE('Iesire cand i = '|| i );
END;
/

```

3.7.6. Instrucțiunea iterativă *FOR*

```

FOR contor IN [REVERSE] lim_inf..lim_sup LOOP
    sevență_de_comenzi;
END LOOP;

```

- Este denumită ciclare cu pas și este utilizată dacă numărul de iterații este cunoscut.
- Comenziile incluse între cuvintele cheie *LOOP* și *END LOOP* sunt executate pentru toate valorile întregi din intervalul $[lim_inf, lim_sup]$.
- Dacă este utilizată opțiunea *REVERSE*, iterația se realizează în sens invers (de la *lim_sup* la *lim_inf*).
- Variabila *contor* nu trebuie declarată.
 - Este neidentificată în afara ciclului.
 - Implicit este de tip *BINARY_INTEGER*.
- Pasul are valoarea 1 (nu poate fi modificat).
- Limitele domeniului pot fi variabile sau expresii de tip întreg sau care pot fi convertite la întreg.

Exemplul 3.20

```

BEGIN
    FOR i IN 1..10 LOOP
        DBMS_OUTPUT.PUT(i**2||' ');
    END LOOP;

    DBMS_OUTPUT.NEW_LINE;
END;
/

BEGIN
    FOR i IN REVERSE 1..10 LOOP
        DBMS_OUTPUT.PUT(i**2||' ');
    END LOOP;

    DBMS_OUTPUT.NEW_LINE;
END;
/

BEGIN
    FOR i IN REVERSE 1..10 LOOP
        DBMS_OUTPUT.PUT(i**2||' ');
        CONTINUE WHEN i<=5;
        DBMS_OUTPUT.NEW_LINE;
    END LOOP;

    DBMS_OUTPUT.NEW_LINE;
END;
/

```

3.7.7. Instrucțiunea vidă

```
NULL;
```

- Nu există nicio corespondență între valoarea *NULL* și instrucțiunea *NULL*.
- Nu realizează nicio operație.
- Plasează controlul următoarei comenzi.
- În *PL/SQL* anumite structuri trebuie să conțină cel puțin o comandă executabilă (de exemplu, instrucțiunea *IF* sau zona de gestiune a exceptiilor).

- Un bloc care nu are nicio acțiune.

Exemplul 3.21

```
DECLARE
    x NUMBER (2) NOT NULL :=2;
BEGIN
    NULL;
END;
```

- Captarea unei excepții pentru care nu se realizează nicio acțiune.

Exemplul 3.22

```
DECLARE
    v_clasificare    clasific_clienti.clasificare%TYPE;
    v_categorie      clasific_clienti.id_categorie%TYPE;
    v_client         clasific_clienti.id_categorie%TYPE := 978;
BEGIN
    SELECT clasificare, id_categorie
    INTO   v_clasificare, v_categorie
    FROM   clasific_clienti
    WHERE  id_client = v_client;
    DBMS_OUTPUT.PUT_LINE(v_categorie || ' '
                           || v_clasificare);

EXCEPTION
    WHEN NO_DATA_FOUND THEN
        -- DBMS_OUTPUT.PUT_LINE('Nicio linie!');
        NULL;

    WHEN TOO_MANY_ROWS THEN
        DBMS_OUTPUT.PUT_LINE('Mai multe linii!');
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('Alta eroare!');
END;
```

- Salt la o etichetă după care nu urmează nicio instrucțiune executabilă (de exemplu urmează *END* sau *END IF*). Următoarele 3 exemple ilustrează opțiuni posibile.

Exemplul 3.23

```

DECLARE
    i INT(1);
BEGIN
    FOR i in 1..5 loop
        IF i=3 THEN
            GOTO eticheta;
        ELSE
            DBMS_OUTPUT.PUT_LINE('i='||i);
        END IF;
    END LOOP;
    <<eticheta>>
    --instructiunea NULL nu este necesara
    DBMS_OUTPUT.PUT_LINE('STOP cand i='||i);
END;

```

```

DECLARE
    j INT(1);
BEGIN
    FOR i in 1..5 loop
        j:=i;
        IF i=3 THEN
            GOTO eticheta;
        ELSE
            DBMS_OUTPUT.PUT_LINE('i='||i);
        END IF;
    END LOOP;
    <<eticheta>>
    --instructiunea NULL nu este necesara
    DBMS_OUTPUT.PUT_LINE('STOP cand i='||j);
END;

```

```

BEGIN
    FOR i in 1..5 loop
        IF i=3 THEN
            DBMS_OUTPUT.PUT_LINE('STOP cand i='||i);
            GOTO eticheta;
        ELSE
            DBMS_OUTPUT.PUT_LINE('i='||i);
        END IF;
    END LOOP;
    <<eticheta>>
    --instructiunea NULL este necesara
    NULL;
END;

```

- Este des utilizată în instrucțiunile condiționale pentru a sugera că într-un anumit caz nu se întâmplă nimic.

```

BEGIN
    FOR i in 1..5 loop
        IF i=3 THEN
            NULL;
        ELSE
            DBMS_OUTPUT.PUT_LINE('i='||i);
        END IF;
    END LOOP;
END;

```

3.7.8. Instrucțiunea de salt **EXIT**

```
EXIT [etichetă] [WHEN condiție];
```

- Permite ieșirea dintr-un ciclu. (Exemplele 3.17 și 3.18).
- Controlul trece fie la prima instrucțiune situată după clauza *END LOOP* corespunzătoare, fie după instrucțiunea *LOOP* având eticheta specificată.

3.7.9. Instrucțiunea de salt **CONTINUE**

```
CONTINUE [WHEN condiție];
```

- Permite transferarea controlului iterației următoare. (Exemplele 3.18, 3.19 și 3.20).

3.7.10. Instrucțiunea de salt *GOTO*

```
GOTO nume_eticheta;
```

- Permite saltul necondiționat la o instrucțiune executabilă sau la începutul unui bloc care are eticheta specificată în comandă. (Exemplul 3.23).
- Nu este permis saltul:
 - în interiorul unui bloc (subbloc);
 - în interiorul unei comenzi *IF*, *CASE* sau *LOOP*;
 - de la o clauză a comenzi *CASE*, la altă clauză a aceleiași comenzi;
 - de la tratarea unei excepții, în blocul curent;
 - în exteriorul unui subprogram.

Bibliografie

1. *Programare avansată în Oracle9i*, I. Popescu, A. Alecu, L. Velcescu, G. Florea (Mihai), Ed. Tehnică (2004)
2. *Oracle Database PL/SQL Language Reference 11g Release 2*, Oracle Online Documentation (2012)
3. *Oracle Database 11g: PL/SQL Fundamentals, Student Guide*, Oracle University (2009)
4. MySQL Online Documentation (<http://dev.mysql.com>)
5. Microsoft Online Documentation (<http://msdn.microsoft.com>)

Cuprins

4. <i>PL/SQL</i> – Tipuri de date.....	2
4.1. Tipuri de date scalare.....	4
4.1.1. Tipuri de date <i>SQL</i>	5
4.1.2. Tipuri de date <i>PL/SQL</i>	14
4.1.3. Tipuri de date și subtipurile acestora	15
4.1.4. Conversii între tipuri de date.....	17
4.1.5. Atributul <i>%TYPE</i>	17
4.2. Tipuri de date compuse.....	18
4.2.1. Atributul <i>%ROWTYPE</i>	18
4.2.2. Tipul de date înregistrare	19
4.2.3. Tipul de date colecție.....	20
4.2.4. Tablouri indexate	22
4.2.5. Tablouri imbricate.....	25
4.2.6. Vectori.....	29
4.2.7. Colecții pe mai multe niveluri.....	31
4.2.8. Compararea colecțiilor.....	32
4.2.9. Prelucrarea colecțiilor stocate în tabele	33
4.2.10. Procedeul <i>bulk collect</i>	35
4.2.11. Procedeul <i>bulk bind</i>	37
4.3. Vizualizări din dicționarul datelor.....	39
Bibliografie	40

4. PL/SQL – Tipuri de date

- Tipul de date este o mulțime de valori predefinită sau definită de utilizator.
- Constantele, variabilele și parametrii *PL/SQL* trebuie să aibă specificat un tip de date. Acesta va determina formatul de stocare, valorile și operațiile permise.

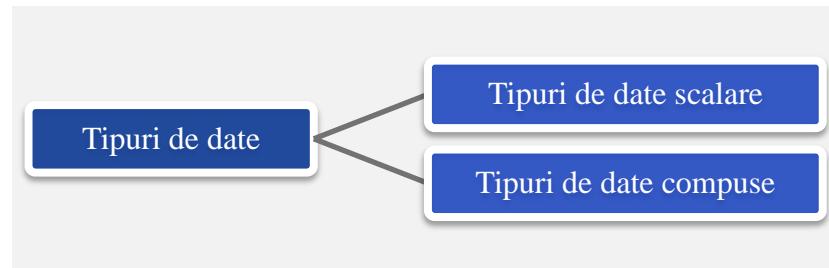


Fig. 4.1. Tipuri de date

- Există două categorii de tipuri de date:
 - tipuri de date scalare
 - pot stoca o singură valoare
 - valoarea stocată nu poate avea componente interne
 - tipuri de date compuse
 - pot stoca mai multe valori
 - valorile stocate pot avea componente interne care pot fi accesate individual

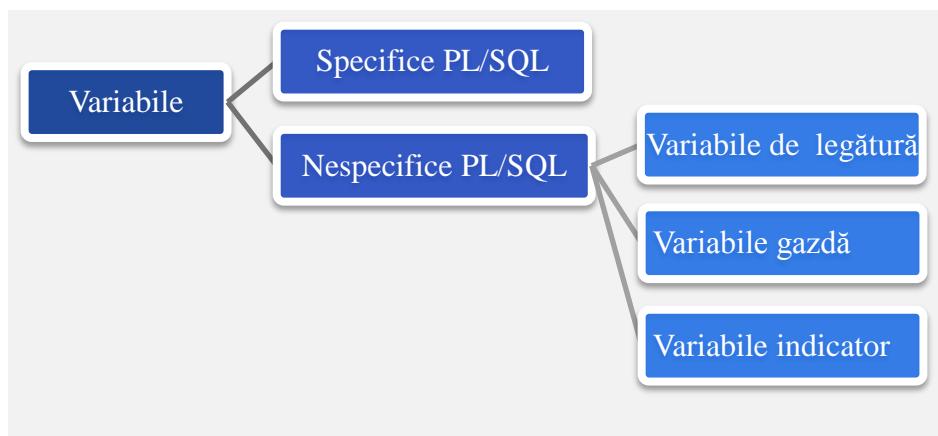


Fig. 4.2. Variabile utilizate de *Oracle*

- Variabilele folosite în *Oracle* pot fi:
 - specifice *PL/SQL*
 - nespecifice *PL/SQL*
 - variabile de legătură (*bind variables*)

- se declară într-un mediu gazdă și sunt folosite pentru a transfera la momentul execuției valori (numerice sau de tip caracter) din/ în unul sau mai multe programe *PL/SQL*
- în *SQL*Plus* se declară folosind comanda *VARIABLE*, iar pentru afișarea valorilor acestora se utilizează comanda *PRINT*; sunt referite prin prefixarea cu simbolul „::“, pentru a putea fi deosebite de variabilele declarate în *PL/SQL*
- variabile gazdă (*host variables*)
 - permit transferul de valori între un mediu de programare (de exemplu, instrucțiunile *SQL* pot fi integrate în programe *C/C++*) și instrucțiunile *SQL* care comunică cu *server-ul bazei de date Oracle*
 - în precompilatorul *Pro*C/C++* sunt declarate între directivele *EXEC SQL BEGIN DECLARE SECTION* și *EXEC SQL END DECLARE SECTION*
- variabile indicator (*indicator variables*)
 - se asociază unei variabile gazdă și permit monitorizarea acesteia
 - permit comunicarea valorii *null* între *Oracle* și un limbaj gazdă care nu are o valoare corespunzătoare pentru *null* (de exemplu, *C*)
 - se utilizează folosind una dintre formele de mai jos


```
:variabilă_gazdă INDICATOR :variabilă_indicator
sau
:variabilă_gazdă :variabilă_indicator
```
 - sunt de tip întreg (stocat 2 bytes)
 - *Oracle* poate atribui unei variabile indicator următoarele valori:
 - 0, dacă operația s-a realizat cu succes
 - 1, dacă o valoare *null* a fost întoarsă, inserată sau actualizată
 - 2, dacă într-o variabilă gazdă de tip caracter s-a întors o valoare de tip *LONG* trunchiată, fără să se poată determina lungimea originală a coloanei
 - >0, dacă rezultatul unei comenzi *SELECT* sau *FETCH* într-o variabilă gazdă de tip caracter a fost trunchiat; în acest caz valoarea indicator este dimensiunea originală a coloanei.

Exemplu

```

EXEC SQL BEGIN DECLARE SECTION;
    float pret_produs;
    short indicator_pret;
EXEC SQL END DECLARE SECTION;
    ...

EXEC SQL SELECT pret
    INTO :pret_produs:indicator_pret
    FROM produse
    WHERE id_produs = 100;

IF (indicator_pret == -1)
    PRINTF("Produsul nu are pret specificat ");

ELSE
    ...

```

- un program poate atribui unei variabile indicator următoarele valori:
- 1, caz în care *Oracle* va atribui coloanei valoarea *null*, ignorând valoarea variabilei gazdă
- ≥ 0 , caz în care *Oracle* va atribui coloanei valoarea variabilei gazdă

Exemplu

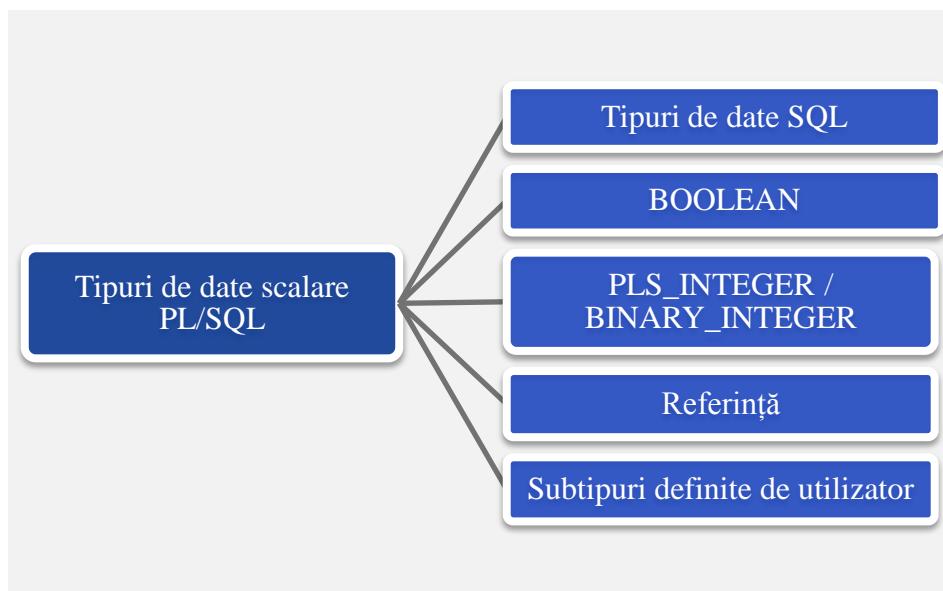
```

...
SET v_indicator = -1;
EXEC SQL INSERT INTO clienti (id_client, status)
    VALUES (:v_cod, :v_status:v_indicator);

```

4.1. Tipuri de date scalare

- Un tip de date scalar stochează o singură valoare care nu poate avea componente interne.
- Tipurile de date scalare pot avea definite subtipuri.
 - Subtipul este un tip de date care reprezintă o submulțime a unui alt tip de date, denumit tip de bază.
 - Subtipul permite aceleasi operații ca și tipul de bază.
- Pachetul *STANDARD* conține tipurile și subtipuri predefinite.
 - Utilizatorii pot defini propriile lor subtipuri.

**Fig. 4.3.** Tipuri de date scalare PL/SQL

- Tipuri de date scalare *PL/SQL*:
 - tipurile de date *SQL*
 - *BOOLEAN*
 - *PLS_INTEGER / BINARY_INTEGER*
 - referință (de exemplu, *REF CURSOR*)
 - subtipuri definite de utilizator

4.1.1. Tipuri de date *SQL*

- Dimensiunea maximă permisă de aceste tipuri de date poate fi diferită în *PL/SQL* față de *SQL*.

Tipuri CHARACTER

Tip date	Descriere	Dim max PL/SQL	Dim max SQL
CHAR [(n [BYTE CHAR])]	Dimensiune fixă - <i>n bytes</i> sau caractere (un caracter poate ocupa mai mult de 1 byte). Implicit n=1 byte.	32767 bytes	2000 bytes
VARCHAR2 (n [BYTE CHAR])	Dimensiune variabilă - <i>n bytes</i> sau caractere. Nu are valoare implicită.	32767 bytes	4000 bytes
NCHAR [(n)]	Dimensiune fixă - <i>n caractere</i> , aparținând setului național de caractere. Implicit n=1.	32767 bytes	2000 bytes
NVARCHAR2 (n)	Dimensiune variabilă, având <i>n caractere</i> , aparținând setului național de caractere. Nu are valoare implicită.	32767 bytes	4000 bytes

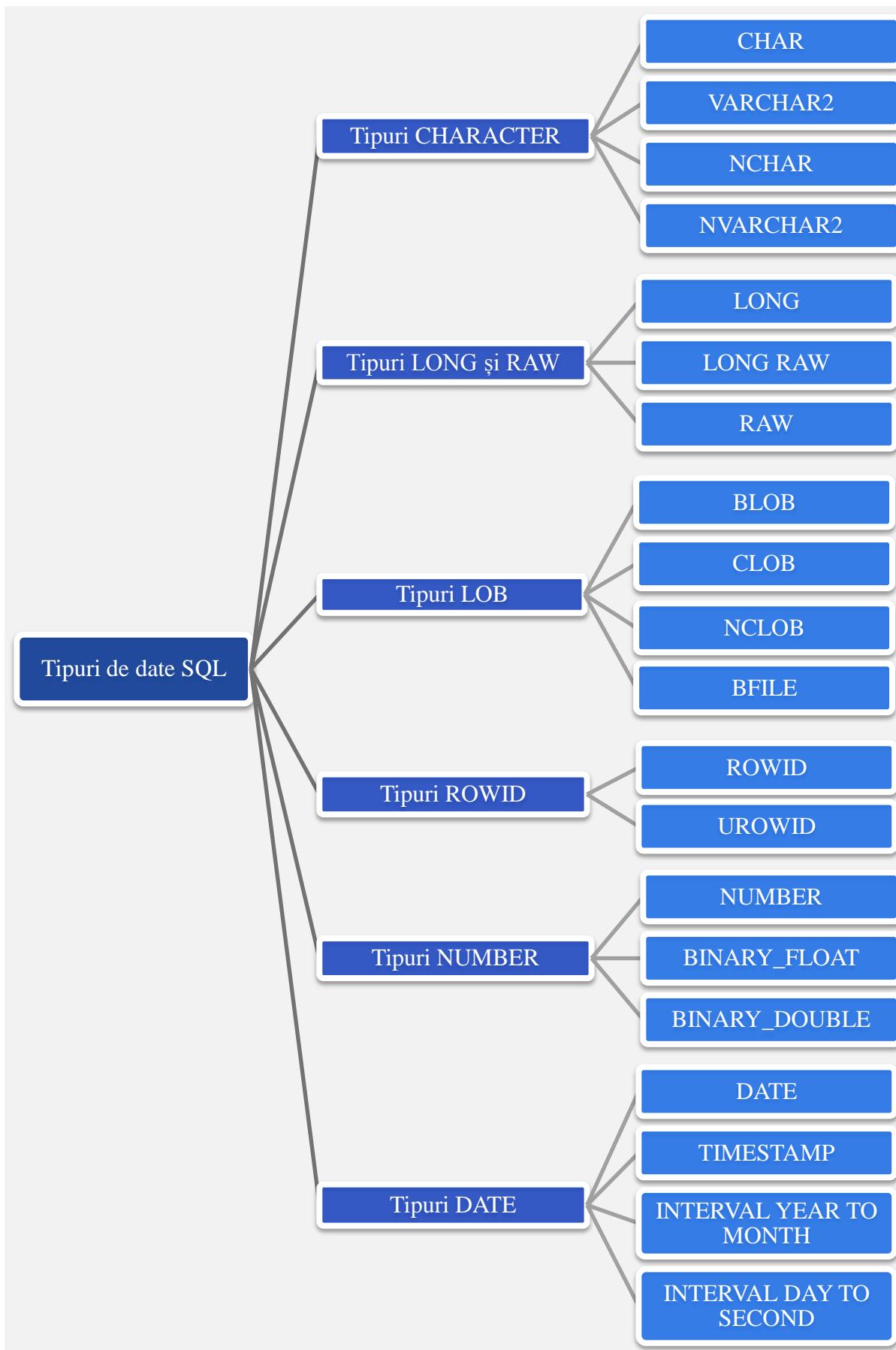


Fig. 4.4. Tipuri de date SQL

Exemplul 4.1

```

DECLARE
    sir_1 CHAR(10) := 'PL/SQL';
    sir_2 VARCHAR2(10) := 'PL/SQL';
BEGIN
    IF sir_1 = sir_2 THEN
        DBMS_OUTPUT.PUT_LINE (sir_1 || ' = ' || sir_2);
    ELSE
        DBMS_OUTPUT.PUT_LINE (sir_1 || ' != ' || sir_2 );
    END IF;
END;

```

Tipuri LONG și RAW

Tip date	Descriere	Dim max PL/SQL	Dim max SQL
LONG	Dimensiune variabilă. Păstrat doar din motive de compatibilitate cu versiunile anterioare.	32760 bytes	2GB – 1 (gigabytes)
LONG RAW	Date în format binar. Dimensiune variabilă. Păstrat doar din motive de compatibilitate cu versiunile anterioare.	32760 bytes	2GB
RAW(n)	Date în format binar sau date care sunt prelucrate byte cu byte (grafice, fișiere audio) Dimensiune variabilă. Nu are valoare implicită.	32767 bytes	2000 bytes

Tipuri LOB

Tip date	Descriere	Dim max PL/SQL	Dim max SQL
BLOB	Obiecte de tip binar de dimensiuni mari	128TB (terabytes)	(4GB-1byte) * dim_bloc (dimensiune bloc date)
CLOB	Obiecte de tip caracter de dimensiuni mari	128TB	(4GB-1byte) * dim_bloc
NCLOB	Obiecte de tip caracter de dimensiuni mari. Datele stocate corespund setului național de caractere.	128TB	(4GB-1byte) * dim_bloc
BFILE	Specifice) * dim_bloc caractere.ensiuni mari... Permite stocarea datelor binare ate, icator este dimensiunea origin	128TB	(4GB-1byte) * dim_bloc

Tipuri ROWID

Tip date	Descriere	Dim max PL/SQL	Dim max SQL
ROWID	Adresele fizice ale liniilor. (000000.FFF.BBBBBB.LLL)	obiect.fișier.bloc.linie	obiect.fișier.bloc.linie
UROWID [(n)]	Adresele logice și fizice ale liniilor. Implicit 4000bytes.	4000 bytes	4000 bytes



- ❖ ROWID-urile fizice stochează adresa liniilor din tabelele obișnuite (care nu sunt de tip *index-organized*), *cluster*-e, partiții și subpartiții ale tabelelor, indecsi, partiții și subpartiții ale indecsilor.
- ❖ ROWID-urile logice stochează adresa liniilor din tabele de tip *index-organized*.

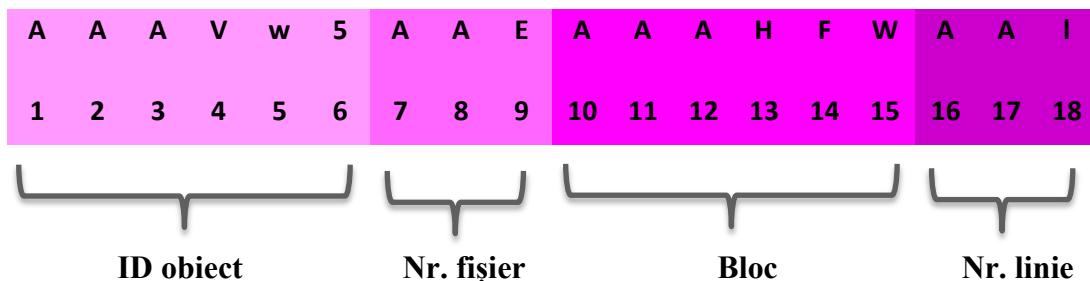


Fig. 4.5 Componentele unui ROWID



Tipul de date *UROWID* (*Universal ROWID*) permite atât adresele fizice, cât și logice ale liniilor dintr-o bază de date *Oracle*, dar și adresele liniilor din tabele externe *non-Oracle*.



Tabelele relaționale obișnuite stochează datele nesortate. Un tabel de tip *index-organized* este un tip de tabel care stochează datele într-o structură de index *B-tree*, sortate logic după cheia primară. Față de indexul normal creat automat la definirea unei chei primare, care stochează doar coloanele incluse în definiția cheii, indexul tabelului de tip *index-organized* stochează în general toate coloanele tabelului (coloanele care sunt accesate rar pot fi mutate în alte segmente față de cel principal).

Tipuri NUMBER

Tip date	Descriere	Dim max PL/SQL	Dim max SQL
NUMBER [(p[, s])]	<p>Număr cu precizia p (numărul total de cifre) și scala s (numărul de cifre ale părții zecimale dacă s este pozitiv).</p> <p>Implicit $s=0$.</p> <p>$p \in [1,38], s \in [-84,127]$</p> <p>Dacă p nu este specificat, atunci se stochează valoarea dată.</p> <p>Dacă s este pozitiv, atunci se face rotunjire a părții zecimale (de ex. dacă $s=2$, 12,474 devine 12,47, iar 12,476 devine 12,48).</p> <p>Dacă s este negativ, atunci se face rotunjire a părții întregi (de ex. dacă $s=-2$ avem rotunjire la sute; 1245 devine 1200 și 1255 devine 1300).</p> <p>Dacă $s=0$ se face rotunjire la întreg (de ex., numărul 3,45 devine 3, iar numărul 3,67 devine 4).</p> <p>Pentru a preciza doar valoarea lui s se folosește NUMBER(*,s).</p>	38 cifre	38 cifre
BINARY_FLOAT	Număr virgulă mobilă precizie simplă (32 biți)	5 bytes (1 byte pentru lungime)	5 bytes
BINARY_DOUBLE	Număr virgulă mobilă precizie dublă (64 biți)	9 bytes (1 byte pentru lungime)	9 bytes

Tipuri DATE

Tip de date	Descriere
DATE	Dată calendaristică între 01.01.4712 î.Hr. și 31.12.9999 d.Hr.
TIMESTAMP [(p)] [WITH [LOCAL] TIME ZONE]	Dată calendaristică și timp, cu precizia p pentru milisecunde ($p \in [0,9]$, implicit $p=6$). <i>WITH TIME ZONE</i> specifică diferența de fus orar. <i>LOCAL</i> implică transformarea datei calendaristice conform timpului regiunii care este setat la nivelul bazei de date.
INTERVAL YEAR [(p)] TO MONTH	Perioadă de timp specificată în ani și luni. Precizia p reprezintă numărul maxim de cifre al câmpului <i>YEAR</i> ($p \in [0,9]$, implicit $p=2$).
INTERVAL DAY [(d)] TO SECOND [(s)]	Perioadă de timp specificată în zile, ore, minute și secunde. Precizia d reprezintă numărul maxim de cifre al câmpului <i>DAY</i> ($d \in [0,9]$, implicit $d=2$).

- Oracle stochează datele de tip *DATE* folosind în întotdeauna 7 bytes. Fiecare byte stochează câte un element din dată.

Nr Byte	Descriere
1	Secol (înainte de stocare adaugă 100)
2	An (înainte de stocare adaugă 100)
3	Luna
4	Zi
5	Ora (înainte de stocare adaugă 1)
6	Minute (înainte de stocare adaugă 1)
7	Secunde (înainte de stocare adaugă 1)

Exemplul 4.2

```

CREATE TABLE test (d DATE);

INSERT INTO test
VALUES (TO_DATE('15-OCT-2012', 'DD-MON-YYYY'));

INSERT INTO test
VALUES (TO_DATE('15-OCT-2012 00:00:00',
               'DD-MON-YYYY HH24:MI:SS'));

INSERT INTO test
VALUES (TO_DATE('15.10.2012 15:22:07',
               'DD.MM.YYYY HH24:MI:SS'));

```

```
INSERT INTO test VALUES (sysdate);

SELECT DUMP(d) FROM test;

Typ=12 Len=7: 120,112,10,15,1,1,1
Typ=12 Len=7: 120,112,10,15,1,1,1
Typ=12 Len=7: 120,112,10,15,16,23,8
Typ=12 Len=7: 120,112,8,24,14,34,27
```

- Dacă se utilizează direct *SYSDATE* sau *TO_DATE* formatul se modifică:
 - Typ = 13
 - Len = 8
 - Byte-ul 8 nu este utilizat
 - anul se poate obține cu următoarea formulă: *Byte 1 + Byte 2 * 256*

Exemplul 4.3

```
SELECT DUMP(TO_DATE('15-OCT-2012 00:00:00',
    'DD-MON-YYYY HH24:MI:SS'))
FROM   DUAL;
```

Typ=13 Len=8: 220,7,10,15,0,0,0,0

```
SELECT DUMP(SYSDATE)
FROM   DUAL;
```

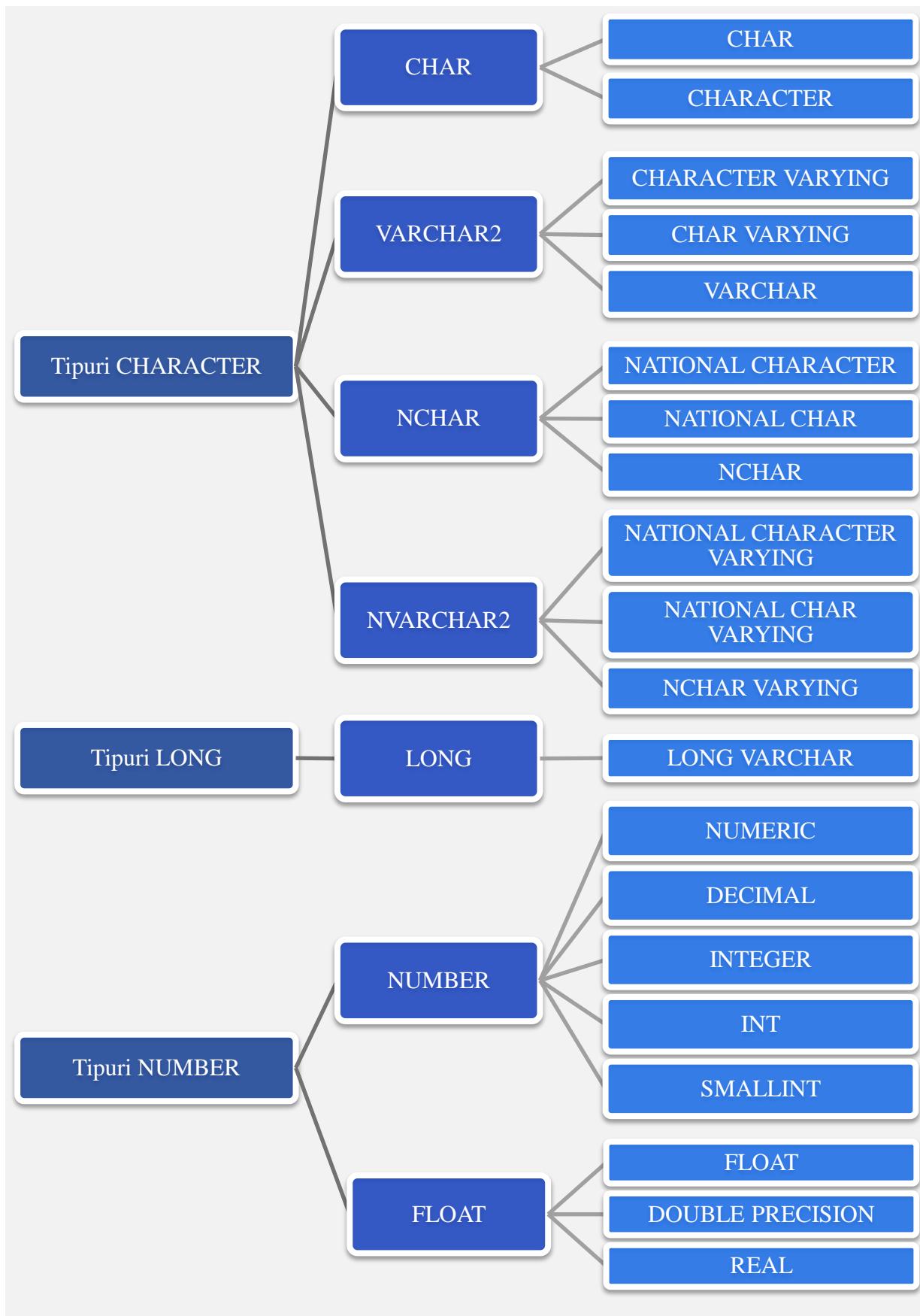
Typ=13 Len=8: 220,7,8,24,13,35,57,0

2012 = 220+7*256

Tipuri de date *SQL ANSI* sau *IBM*

- *Oracle* recunoaște numele tipurilor de date *ANSI* sau *IBM* (folosite de *SQL/DS* sau *DB2*) care diferă de numele tipurilor de date proprii.
- Atunci când este utilizat un tip de date *ANSI* sau *IBM*, acesta va fi convertit automat la tipul de date echivalent din *Oracle*.

Tip de date <i>ANSI</i>	Tip de date echivalent <i>ORACLE</i>
CHARACTER (n) CHAR (n)	CHAR (n)
CHARACTER VARYING (n) CHAR VARYING (n)	VARCHAR2 (n)
NATIONAL CHARACTER (n) NATIONAL CHAR (n) NCHAR (n)	NCHAR (n)
NATIONAL CHARACTER VARYING (n) NATIONAL CHAR VARYING (n) NCHAR VARYING (n)	NVARCHAR2 (n)
NUMERIC [(p, s)] DECIMAL [(p, s)]	NUMBER (p, s)
INTEGER INT SMALLINT	NUMBER (38)
FLOAT DOUBLE PRECISION REAL	FLOAT (126) FLOAT (126) FLOAT (63)
Tip de date <i>SQL/DS</i> sau <i>DB2</i>	Tip de date echivalent <i>ORACLE</i>
CHARACTER (n)	CHAR (n)
VARCHAR (n)	VARCHAR (n)
LONG VARCHAR	LONG
DECIMAL (p, s)	NUMBER (p, s)
INTEGER SMALLINT	NUMBER (38)

**Fig. 4.6** Tipuri de date Oracle cu tipurile ANSI/IBM echivalente

4.1.2. Tipuri de date PL/SQL

Tipul de date **BOOLEAN**

- Stochează valorile logice *true*, *false* sau valoarea *null*
- Nu are un tip *SQL* echivalent și din acest motiv nu pot fi utilizate variabile sau parametrii de tip *boolean* în:
 - comenzi *SQL*
 - funcții *SQL* predefinite
 - funcții *PL/SQL* invocate în comenzi *SQL*

Tipul de date **PLS_INTEGER / BINARY_INTEGER**

- Tipurile de date *PLS_INTEGER* și *BINARY_INTEGER* sunt identice.
- Stochează numere întregi cu semn reprezentate pe 32 biți cu valori cuprinse între -2.147.483.648 și 2.147.483.647.



Avantaje față de tipul *NUMBER* și subtipurile sale

- necesită mai puțin spațiu de stocare
- deoarece folosesc aritmetică mașinii operațiile cu acest tip sunt efectuate mai rapid decât operațiile cu tipurile *NUMBER* (care folosesc librării aritmetice).

Tipul de date referință

- Are ca valoare un *pointer* care face referință către un obiect
 - *REF CURSOR* - locația din memorie (adresa) unui cursor explicit (*Informații suplimentare în cursul despre cursoare*)

Subtipuri definite de utilizator

- Pentru a crea un subtip se utilizează comanda

```
SUBTYPE nume_subtip IS tip_de_bază [ (constrângere) ]
[NOT NULL];
```

- *tip_de_bază* poate fi un tip de date scalar sau un tip definit de utilizator
- *constrângere* se referă la precizie și scală.
- Nu se pot specifica valori implice.

Exemplul 4.4

```

DECLARE
    SUBTYPE subtip_data IS DATE NOT NULL;
    SUBTYPE subtip_email IS CHAR(15);
    SUBTYPE subtip_descriere IS VARCHAR2(1500);
    SUBTYPE subtip_rang IS PLS_INTEGER RANGE -5..5;
    SUBTYPE subtip_test IS BOOLEAN;
    v_data subtip_data := SYSDATE;
    v_email subtip_email(10);
    v_descriere subtip_descriere;
    v_rang subtip_rang := 2;
    v_test BOOLEAN;
BEGIN
    NULL;
END;

```

4.1.3. Tipuri de date și subtipurile acestora

Tip date	Subtip	Descriere
NUMBER	DEC DECIMAL NUMERIC	NUMBER cu virgulă fixă, precizie maximă 38 cifre zecimale
	FLOAT DOUBLE PRECISION	NUMBER cu virgulă mobilă, precizie maximă 126 cifre binare (aproximativ 38 cifre zecimale)
	INT INTEGER SMALLINT	Intreg, precizie maximă 38 cifre zecimale
	REAL	NUMBER cu virgulă mobilă, precizie maximă 63 cifre binare (aproximativ 18 cifre zecimale)
PLS_INTEGER	NATURAL	Valorile PLS_INTEGER nenegative
	NATURALN	Valorile PLS_INTEGER nenegative cu constrângerea NOT NULL
	POSITIVE	Valorile PLS_INTEGER pozitive
	POSITIVEN	Valorile PLS_INTEGER pozitive cu constrângerea NOT NULL
	SIGNTYPE	Valorile PLS_INTEGER -1, 0 și 1
	SIMPLE_INTEGER	Valorile PLS_INTEGER cu constrângerea NOT NULL
CHAR	CHARACTER	Același domeniu de valori ca și CHAR. Este folosit din motive de compatibilitate cu tipurile ANSI și IBM.

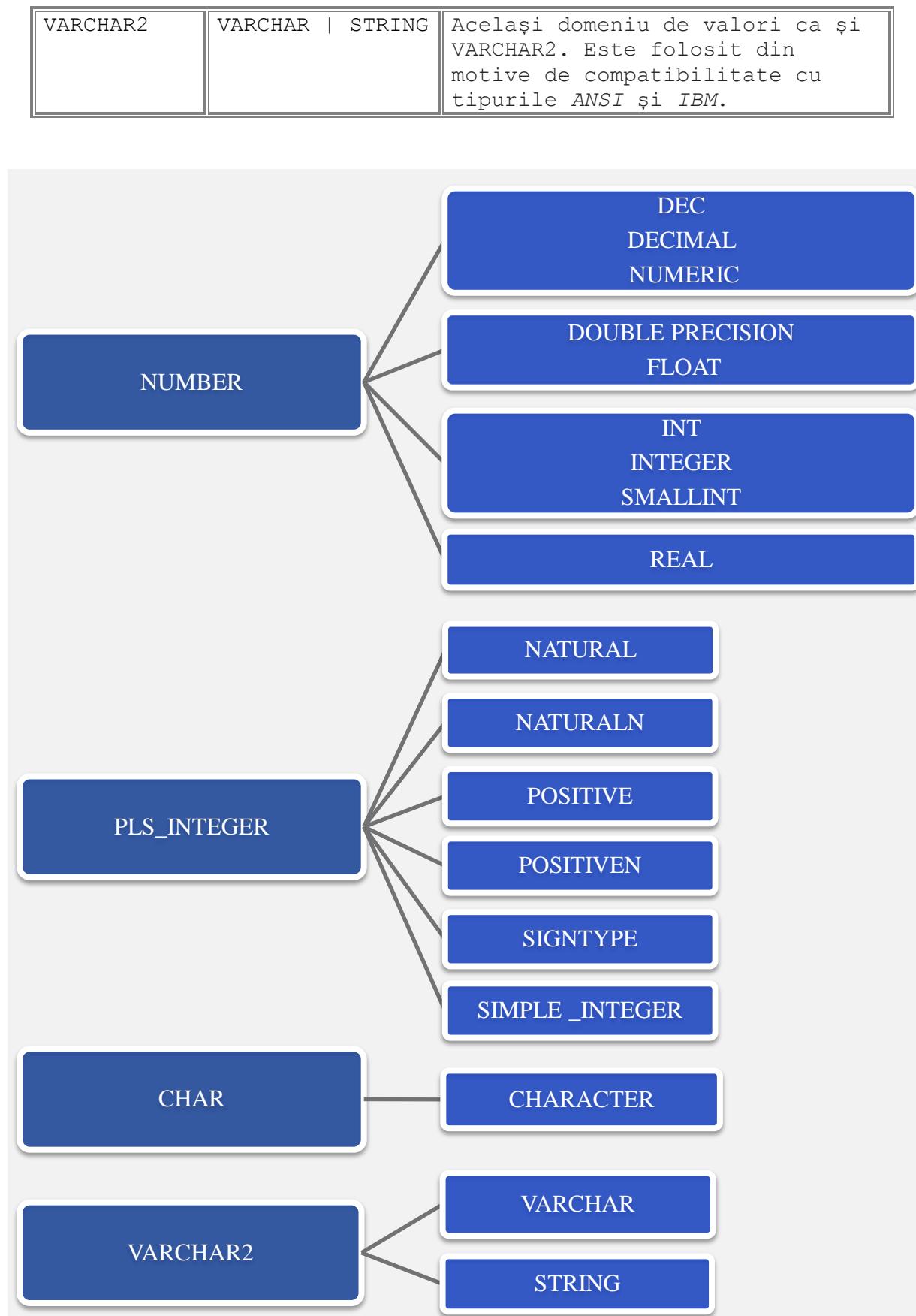


Fig. 4.7 Tipuri de date și subtipurile lor

4.1.4. Conversii între tipuri de date

- Tipuri de conversii
 - implicate (realizate automat de sistem)

Exemple de conversii implicate

	DATE	NUMBER	VARCHAR2	PLS_INTEGER
DATE	Nu se aplică	X	✓	X
NUMBER	X	Nu se aplică	✓	✓
VARCHAR2	✓	✓	Nu se aplică	✓
PLS_INTEGER	X	✓	✓	Nu se aplică

- explicite (realizate folosind explicit funcțiile de conversie)

Exemple de funcții de conversie

ASCIISTR, BFILENAME, BIN_TO_NUM, CAST, CHARTOROWID, COMPOSE, CONVERT, DECOMPOSE, HEXTORAW, NUMTODSINTERVAL, NUMTOYMINTERVAL, RAWTOHEX, RAWTONHEX, REFTOHEX, ROWIDTOCHAR, ROWIDTONCHAR, SCN_TO_TIMESTAMP, TIMESTAMP_TO_SCN, TO_BINARY_DOUBLE, TO_BINARY_FLOAT, TO_CHAR, TO_CLOB, TO_DATE, TO_DSINTERVAL, TO_LOB, TO_MULTI_BYTE, TO_NCHAR, TO_NCLOB, TO_NUMBER, TO_SINGLE_BYTE, TO_TIMESTAMP, TO_TIMESTAMP_TZ, TO_YMINTERVAL, TRANSLATE USING, UNISTR

Conversiile implicate au o serie de dezavantaje:



- pot fi lente;
- se pierde controlul asupra programului (dacă Oracle modifică regulile de conversie, atunci codul poate fi afectat);
- depind de mediul în care sunt utilizate (de exemplu, formatul datei calendaristice variază în funcție de setări; astfel, codul poate să nu ruleze pe server-e diferite);
- codul devine mai greu de înțeles.

4.1.5. Atributul %TYPE

- Este utilizat pentru a declara o variabilă cu același tip de date ca al altor variabile sau al unei coloane dintr-un tabel.

```
variabilă_2 variabilă_1%TYPE;
variabilă nume_tabel.nume_coloană%TYPE;
```



Avantaje:

- nu este necesar să se cunoască exact tipul de date al coloanei din tabel
- anumite modificări realizate asupra tipului de date al coloanei (de exemplu, se mărește dimensiunea), nu vor afecta programul

4.2. Tipuri de date compuse

- Un tip de date compus stochează mai multe valori care pot avea componente interne ce pot fi accesate individual.

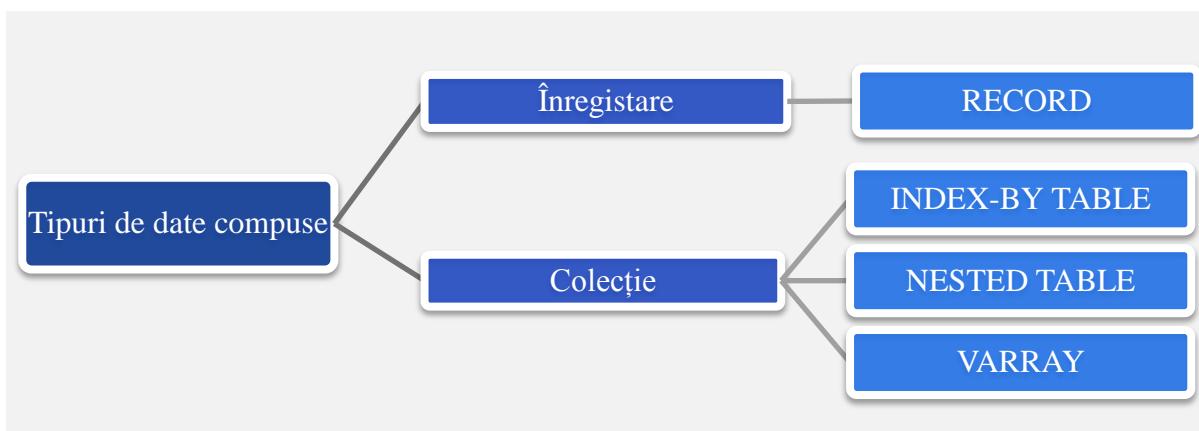


Fig. 4.8. Tipuri de date compuse

- Tipurile de date compuse
 - înregistrare (*RECORD*)
 - componentele interne pot avea tipuri de date diferite și sunt denumite câmpuri
 - colecție (*INDEX-BY TABLE*, *NESTED TABLE*, *VARRAY*)
 - componentele interne au același tip de date și sunt denumite elemente
 - fiecare element poate fi accesat folosind indexul său

4.2.1. Atributul *%ROWTYPE*

- Este utilizat pentru a declara o variabilă de tip înregistrare cu aceeași structură ca a altelor variabile de tip înregistrare, a unui tabel sau cursor.

```

variabilă_2 variabilă_1%ROWTYPE;
variabilă nume_tabel%ROWTYPE;
variabilă nume_cursor%ROWTYPE;
  
```

4.2.2. Tipul de date înregistrare

- Înregistrările se definesc în doi pași:
 - se definește un tip *RECORD*;
 - se declară variabile de acest tip.

```

TYPE nume_tip IS RECORD
  (nume_câmp1 {tip_de_date |
    variabilă%TYPE | 
    nume_tabel.coloană%TYPE | 
    nume_tabel%ROWTYPE}
  [ [NOT NULL] {:= | DEFAULT} expresie],
  nume_câmp2 {tip_de_date |
    variabilă%TYPE | 
    nume_tabel.coloană%TYPE | 
    nume_tabel%ROWTYPE}
  [ [NOT NULL] {:= | DEFAULT} expresie], ...);

variabilă nume_tip;

```

- Câmpurile unei înregistrări
 - Au implicit valoarea *null*.
 - Numărul lor nu este limitat.
 - Se referă prin prefixare cu numele înregistrării.
 - Pot fi tip scalar, *RECORD*, obiect, colecție.
 - Nu pot fi de tip *REF CURSOR*.
- Atribuirea de valori unei înregistrări se poate realiza cu
 - instrucțiunea de atribuire
 - comenzile *SELECT* sau *FETCH*
- Înregistrările
 - nu pot fi comparate (egalitate, inegalitate sau *null*).
 - pot fi parametri în subprograme.
 - pot să apară în clauza *RETURN* a unei funcții.
- Folosind direct numele înregistrării (fără a accesa individual câmpurile) se poate:
 - insera o linie în tabel (*INSERT*);
 - actualiza o linie (*UPDATE ...SET ROW*);
 - capătă o linie inserată, modificată sau ștearsă (*RETURNING*);
 - regăsi o linie (*SELECT ... INTO*).



Tipul *RECORD* nu poate fi definit decât local (într-un bloc *PL/SQL* sau pachet).

Exemplul 4.5

```

DECLARE
    TYPE rec IS RECORD
        (id    categorii.id_categoria%TYPE,
         den   categorii.denumire%TYPE,
         niv   categorii.nivel%TYPE);
    v_categ  rec;
    v_categ2 rec;
BEGIN
    v_categ.den := 'Categoria noua';
    v_categ.niv :=1;
    SELECT MAX(id_categoria)+1 INTO v_categ.id
    FROM   categorii;

    -- eroare
    -- INSERT INTO categorii(id_categoria, denumire, nivel)
    -- VALUES v_categ;
    INSERT INTO categorii(id_categoria, denumire, nivel)
    VALUES (v_categ.id, v_categ.den, v_categ.niv);

    SELECT id_categoria, denumire, nivel INTO v_categ2
    FROM   categorii
    WHERE  id_categoria= v_categ.id;

    DBMS_OUTPUT.PUT_LINE ('Ati inserat: '|| v_categ2.id ||
        ' ' || v_categ2.den || ' '|| v_categ2.niv);
END;

```

Exemplul 4.6 – vezi curs**4.2.3. Tipul de date colecție**

- Există 3 tipuri de colecții:
 - tablouri indexate (*index-by tables*), care sunt denumite și vectori asociativi (*associative arrays*)
 - sunt similare cu tabelele de dispersie (*hash tables*) din alte limbaje de programare
 - tablouri imbricate (*nested tables*)
 - sunt similare cu mulțimile (*sets, multisets*) din alte limbaje de programare
 - vectori cu dimensiune variabilă (*varray*, prescurtare de la *variable-size arrays*),
 - sunt similari cu vectorii din alte limbaje de programare
 - din motive de simplificare vor fi referiți în continuare ca *vectori*
- Declararea unei colecții se realizează în 2 pași:
 - se definește un tip colecție
 - se declară o variabilă de acel tip

- Caracteristicile tipurilor colecție

Tip colecție	Număr maxim elemente	Tip index	Dens sau împrăștiat	Loc definire
Tablouri indexate	nefixat	întreg $\in [-2^{31}-1, 2^{31}-1] = 2^{31}-1 = 2147483647$ sau sir de caractere	ambele	doar în blocuri PL/SQL
Tablouri imbricate	nefixat	întreg $\in [1, 2^{31}-1]$	initial dens, dar poate deveni împrăștiat	în blocuri PL/SQL sau la nivel de schemă
Vectori	fixat (n dat)	întreg $\in [1, n]$	dens	în blocuri PL/SQL sau la nivel de schemă

- Metodele asociate colecțiilor
 - sunt subprograme *PL/SQL* predefinite (funcții sau proceduri)
 - întorc informații despre o colecție sau operează asupra acesteia
 - pot fi apelate numai din comenzi procedurale (nu pot fi apelate în comenzi *SQL*)
 - pot fi invocate folosind forma următoare


```
nume_colecție.nume_metodă [ (parametri) ]
```
- Metodele disponibile pentru colecții sunt date în tabelul următor
 - Notațiile utilizate
 - *Tab ind* – tablou indexat
 - *Tab imb* – tablou imbricat
 - *Vec* – vector

Metodă	Descriere	Validitate		
		Tab ind	Tab imb	Vec
COUNT	Întoarce numărul curent de elemente	✓	✓	✓
DELETE	Sterge toate elementele	✓	✓	✓
DELETE (n)	Sterge elementul <i>n</i>	✓	✓	
DELETE (n, m)	Sterge toate elementele care au indexul cuprins între <i>n</i> și <i>m</i>	✓	✓	

EXISTS (n)	Întoarce <i>TRUE</i> dacă există al <i>n</i> -lea element, altfel întoarce <i>FALSE</i> (în locul excepției <i>SUBSCRIPT_OUTSIDE_LIMIT</i>)	✓	✓	✓
FIRST	Întoarce indexul primului element (cel mai mic index)	✓	✓	✓
LAST	Întoarce indexul ultimului element (cel mai mare index)	✓	✓	✓
NEXT (n)	Întoarce indexul elementului care urmează după elementul cu indexul <i>n</i> . Dacă nu există, întoarce <i>null</i> .	✓	✓	✓
PRIOR (n)	Întoarce indexul elementului care precede elementul cu indexul <i>n</i> . Dacă nu există, întoarce <i>null</i> .	✓	✓	✓
EXTEND	Adaugă un element <i>null</i> la sfârșit		✓	✓
EXTEND (n)	Adaugă <i>n</i> elemente <i>null</i> la sfârșit		✓	✓
EXTEND (n, i)	Adaugă <i>n</i> copii ale elementului de rang <i>i</i> la sfârșit		✓	✓
LIMIT	Întoarce numărul maxim de elemente specificat la declarare în cazul vectorilor, respectiv valoarea <i>null</i> în cazul tablourilor imbricate		✓	✓
TRIM	Șterge ultimul element		✓	✓
TRIM (n)	Șterge ultimele <i>n</i> elemente. Dacă <i>n</i> este mai mare decât numărul curent de elemente, atunci apare excepția <i>SUBSCRIPT_BEYOND_COUNT</i>		✓	✓

- *EXISTS* este singura metodă care poate fi aplicată unei colecții atomice *null*.
 - Orice altă metodă declanșează excepția *COLLECTION_IS_NULL*.
- *COUNT, EXISTS, FIRST, LAST, NEXT, PRIOR și LIMIT* sunt funcții, iar restul sunt proceduri *PL/SQL*.

4.2.4. Tablouri indexate

- Sunt mulțimi de perechi cheie-valoare, în care fiecare cheie este unică și utilizată pentru a putea localiza valoarea asociată.
- Atunci când este creat un tablou indexat care nu are încă elemente, acesta este vid. Nu este inițializat automat (atomic) *null*, ca în cazul celorlalte tipuri de colecții.
- Atunci când o valoare este asociată pentru prima oară unei chei, cheia este adăugată în tablou.

- Sintaxă declarare tip

```

TYPE nume_tip IS TABLE OF tip_element [NOT NULL]
  [INDEX BY { PLS_INTEGER
    | BINARY_INTEGER
    | VARCHAR2 (n)
  }
];
unde tip_element poate fi orice tip PL/SQL mai puțin REF CURSOR

{ nume_cursor%ROWTYPE
| nume_tabel%ROWTYPE | .nume_coloană%TYPE}
| nume_object%TYPE
| [REF] nume_tip_object
| nume_record[.nume_câmp]%TYPE
| nume_tip_record
| nume_tip_date_scalar
| variabilă%TYPE
}

```



Pentru indexare se pot utiliza și subtipurile *VARCHAR*, *STRING* sau *LONG*.



Tablourile indexate folosesc spațiu de stocare temporar.

Pentru a deveni persistente pe perioada sesiunii trebuie declarate într-un pachet (atât tipul, cât și variabilele de acel tip), iar valorile elementelor trebuie asignate în corpul pachetului.



Tablourile indexate

- ❖ nu au constrângeri legate de dimensiune, deci dimensiunea acestora se modifică dinamic
- ❖ nu sunt inițializate la declarare
- ❖ neinițializate sunt vide (nu au chei sau valori)
- ❖ au elemente definite doar dacă acestor elemente li se atribuie valori (dacă se încearcă utilizarea unui element căreia nu i s-a atribuit nicio valoare, se declanșează excepția *NO_DATA_FOUND*)
- ❖ permit inserarea de elemente cu chei arbitrate (nu într-o ordine prestabilită)
- ❖ nu au memorie restricționată relativ la numărul de elemente, ci la dimensiunea de memorie utilizată
- ❖ pot să apară ca parametrii în proceduri.

Exemplul 4.7

```

DECLARE
    TYPE tab_ind IS TABLE OF NUMBER INDEX BY PLS_INTEGER;
    t    tab_ind;
BEGIN
    -- atribuire valori
    FOR i IN 1..10 LOOP
        t(i):=i;
    END LOOP;
    --parcurgere
    DBMS_OUTPUT.PUT('Tabloul are ' || t.COUNT || ' elemente: ');
    FOR i IN t.FIRST..t.LAST LOOP
        DBMS_OUTPUT.PUT(t(i) || ' ');
    END LOOP;
    DBMS_OUTPUT.NEW_LINE;

    -- numar elemente
    FOR i IN 1..10 LOOP
        IF i mod 2 = 1 THEN t(i):=null;
    END IF;
    END LOOP;
    DBMS_OUTPUT.PUT('Tabloul are ' || t.COUNT || ' elemente: ');

    FOR i IN t.FIRST..t.LAST LOOP
        DBMS_OUTPUT.PUT(nvl(t(i), 0) || ' ');
    END LOOP;
    DBMS_OUTPUT.NEW_LINE;

    -- stergere elemente
    t.DELETE(t.first);
    t.DELETE(5,7);
    t.DELETE(t.last);
    DBMS_OUTPUT.PUT_LINE('Primul element are indicele ' ||
        t.first || ' si valoarea ' || nvl(t(t.first),0));
    DBMS_OUTPUT.PUT_LINE('Ultimul element are indicele ' ||
        t.last || ' si valoarea ' || nvl(t(t.last),0));
    DBMS_OUTPUT.PUT('Tabloul are ' || t.COUNT || ' elemente: ');
    FOR i IN t.FIRST..t.LAST LOOP
        IF t.EXISTS(i) THEN
            DBMS_OUTPUT.PUT(nvl(t(i), 0)|| ' ');
        END IF;
    END LOOP;
    DBMS_OUTPUT.NEW_LINE;

    t.DELETE;
    DBMS_OUTPUT.PUT_LINE('Tabloul are ' || t.COUNT
        || ' elemente.');
END;

```

Exemplul 4.8

```

DECLARE
    TYPE tab_ind IS TABLE OF produse%ROWTYPE
        INDEX BY PLS_INTEGER;
    t    tab_ind;
BEGIN
    -- atribuire valori
    SELECT * BULK COLLECT INTO t
    FROM   produse
    WHERE  ROWNUM<=10;

    --parcursere
    DBMS_OUTPUT.PUT_LINE('Tabloul are ' || t.COUNT ||
                          ' elemente: ');
    FOR i IN t.FIRST..t.LAST LOOP
        DBMS_OUTPUT.PUT_LINE(t(i).id_produs || ' ' ||
                             t(i).denumire);
    END LOOP;
END;

```

Exemplul 4.9 - vezi curs**Exemplul 4.10 - vezi curs****Exemplul 4.11 - vezi curs****4.2.5. Tablouri imbricate**

- În baza de date, tabloul imbricat este tip al unei coloane care stochează un număr nespecificat de linii, în nicio ordine particulară (stochează o mulțime de valori).
 - Atunci când tabloul imbricat din baza de date este preluat de o variabilă *PL/SQL*, sistemul atribuie liniilor indecsă consecutivi (începând cu valoarea 1). Astfel, se permite accesarea liniilor folosind indecsă, în mod asemănător cu vectorii.
 - Indecșii și ordinea liniilor dintr-un tablou imbricat ar putea să nu rămână stabilă în timp ce tabloul este stocat sau regăsit din baza de date.
- Numărul maxim de linii este dat de capacitatea maximă 2 GB.
- Inițial, tabloul este dens, dar în urma prelucrării este posibil să nu mai aibă indici consecutivi.

- Tablourile imbricate:
 - pot fi stocate în baza de date;
 - pot fi prelucrate direct în instrucțiuni *SQL*;
 - trebuie inițializate și extinse pentru a li se adăuga elemente.
- Sintaxă declarare tip

```
[CREATE [OR REPLACE]] TYPE nume_tip
IS TABLE OF tip_element [NOT NULL];
```

tip_element poate fi orice tip PL/SQL mai puțin REF CURSOR

```
{ nume_cursor%ROWTYPE
| nume_tabel%ROWTYPE | .nume_coloană%TYPE}
| nume_object%TYPE
| [REF] nume_tip_object
| nume_record[.nume_câmp]%TYPE
| nume_tip_record
| nume_tip_date_scalar
| variabilă%TYPE
}
```

- Un tablou imbricat/vector declarat, dar neinițializat, este automat inițializat (atomic) *null*.
 - Astfel, pentru verificare poate fi utilizat operatorul *IS NULL*.
 - Dacă se încearcă să se adauge un element într-un tablou imbricat/vector neinițializat (atomic *null*), se declanșează eroarea „ORA - 06531: reference to uninitialized collection“ care corespunde excepției predefinite *COLLECTION_IS_NULL*.
- Inițializarea se realizează cu ajutorul unui constructor.
 - tabelele indexate nu au constructori
- Constructorul unei colecții
 - este o funcție sistem predefinită, cu același nume ca și numele tipului colecție referite
 - întoarce o colecție de acel tip
 - se invocă folosind sintaxa


```
nume_tip_colecție ([valoare [, valoare] ... ]);
```
 - dacă pentru parametrii nu sunt specificate valori, atunci întoarce o colecție vidă (nu are elemente, dar nu este atomic *null*); altfel, întoarce o colecție care conține valorile specifice

- Dimensiunea inițială a colecției este egală cu numărul de valori specificate în constructor la inițializare.

Exemplul 4.12 a

```

DECLARE
    TYPE tab_imb IS TABLE OF NUMBER;
    t      tab_imb := tab_imb();
BEGIN
    -- atribuire valori
    FOR i IN 1..10 LOOP
        t.EXTEND;
        t(i):=i;
    END LOOP;
    --parcurgere
    DBMS_OUTPUT.PUT('Tabloul are ' || t.COUNT || ' elemente: ');
    FOR i IN t.FIRST..t.LAST LOOP
        DBMS_OUTPUT.PUT(t(i) || ' ');
    END LOOP;
    DBMS_OUTPUT.NEW_LINE;

    -- numar elemente
    FOR i IN 1..10 LOOP
        IF i mod 2 = 1 THEN t(i):=null;
        END IF;
    END LOOP;
    DBMS_OUTPUT.PUT('Tabloul are ' || t.COUNT || ' elemente: ');

    FOR i IN t.FIRST..t.LAST LOOP
        DBMS_OUTPUT.PUT(nvl(t(i), 0) || ' ');
    END LOOP;
    DBMS_OUTPUT.NEW_LINE;

    -- stergere elemente
    t.DELETE(t.first);
    t.DELETE(5,7);
    t.DELETE(t.last);
    DBMS_OUTPUT.PUT_LINE('Primul element are indicele ' ||
        t.first || ' si valoarea ' || nvl(t(t.first),0));
    DBMS_OUTPUT.PUT_LINE('Ultimul element are indicele ' ||
        t.last || ' si valoarea ' || nvl(t(t.last),0));
    DBMS_OUTPUT.PUT('Tabloul are ' || t.COUNT || ' elemente: ');
    FOR i IN t.FIRST..t.LAST LOOP
        IF t.EXISTS(i) THEN
            DBMS_OUTPUT.PUT(nvl(t(i), 0)|| ' ');
        END IF;
    END LOOP;
    DBMS_OUTPUT.NEW_LINE;

    t.DELETE;
    DBMS_OUTPUT.PUT_LINE('Tabloul are ' || t.COUNT
        || ' elemente.');
END;

```

Exemplul 4.12_b - vezi curs

- Definirea tabelelor cu coloane de tip tablou imbricat presupune
 - definirea unui tip tablou imbricat

```
CREATE TYPE nume_tip IS TABLE OF tip_element [NOT NULL];
```

- definirea tabelului precizând pentru coloană tipul creat
 - pentru fiecare coloană de tip tablou imbricat din tabel este necesară clauza de stocare:

```
NESTED TABLE nume_coloană STORE AS nume_tabel;
```

Exemplul 4.13

```
CREATE TYPE t_imb_categ IS TABLE OF VARCHAR2(40);
/

CREATE TABLE raion_grupe_imb
( id_categorie NUMBER(4) PRIMARY KEY,
  denumire      VARCHAR2(40),
  grupe        t_imb_categ)
NESTED TABLE grupe STORE AS tab_imb_grupe;

INSERT INTO raion_grupe_imb
VALUES (1, 'r1', t_imb_categ('r11','r12'));

INSERT INTO raion_grupe_imb
VALUES (2, 'r2', t_imb_categ('r21'));
INSERT INTO raion_grupe_imb(id_categorie, denumire)
VALUES (3,'r3');

UPDATE raion_grupe_imb
SET    grupe = t_imb_categ('r31','r32')
WHERE  id_categorie =3;

SELECT * FROM raion_grupe_imb;

SELECT id_categorie, denumire, b.*
FROM   raion_grupe_imb  a, TABLE(a.grupe) b;

SELECT grupe
FROM   raion_grupe_imb
WHERE  id_categorie = 1;

SELECT *
FROM   TABLE(SELECT grupe
             FROM raion_grupe_imb
             WHERE id_categorie=1);
```

Exemplul 4.14 - vezi curs

4.2.6. Vectori

- Se utilizează în special pentru modelarea relațiilor *one-to-many*, atunci când numărul maxim de elemente *copil* este cunoscut și ordinea elementelor este importantă.
- Reprezintă structuri dense.
 - Fiecare element are un index care precizează poziția sa în vector (primul index are valoarea 1).
 - Indexul este utilizat pentru accesarea elementelor din vector.
- Vectorii:
 - față de tablourile imbricate au o dimensiune maximă specificată la declarare;
 - pot fi stocați în baza de date;
 - pot fi prelucrați direct în instrucțiuni *SQL*;
 - trebuie inițializați și extinși pentru a li se adăuga elemente.
- Sintaxă declarare tip

```
[CREATE [OR REPLACE]] TYPE nume_tip
IS {VARRAY | VARYING ARRAY} (lungime_maximă) OF tip_element
[NOT NULL];
```

tip_element poate fi orice tip PL/SQL mai puțin REF CURSOR

```
{ nume_cursor%ROWTYPE
| nume_tabel%ROWTYPE | .nume_coloană%TYPE}
| nume_object%TYPE
| [REF] nume_tip_object
| nume_record[.nume_câmp]%TYPE
| nume_tip_record
| nume_tip_date_scalar
| variabilă%TYPE
}
```

Exemplul 4.15

```
DECLARE
    TYPE tab_vec IS VARRAY(10) OF NUMBER;
    t      tab_vec := tab_vec();
BEGIN
    -- atribuire valori
    FOR i IN 1..10 LOOP
        t.EXTEND;
        t(i):=i;
    END LOOP;
    --parcuregere
    DBMS_OUTPUT.PUT('Tabloul are ' || t.COUNT || ' elemente: ');
    FOR i IN t.FIRST..t.LAST LOOP
        DBMS_OUTPUT.PUT(t(i) || ' ');
    END LOOP;
```

```

DBMS_OUTPUT.NEW_LINE;

-- numar elemente
FOR i IN 1..10 LOOP
    IF i mod 2 = 1 THEN t(i):=null;
    END IF;
END LOOP;
DBMS_OUTPUT.PUT('Tabloul are ' || t.COUNT || ' elemente: ');

FOR i IN t.FIRST..t.LAST LOOP
    DBMS_OUTPUT.PUT(nvl(t(i), 0) || ' ');
END LOOP;
DBMS_OUTPUT.NEW_LINE;
-- stergere elemente
t.DELETE;
DBMS_OUTPUT.PUT_LINE('Tabloul are ' || t.COUNT
|| ' elemente.');
END;

```

Exemplul 4.16

```

CREATE TYPE t_vect_categ IS VARRAY(10) OF VARCHAR2(40);
/

CREATE TABLE raion_grupe_vect
( id_categorie NUMBER(4) PRIMARY KEY,
  denumire      VARCHAR2(40),
  grupe        t_vect_categ);

INSERT INTO raion_grupe_vect
VALUES (1, 'r1', t_vect_categ('r11','r12'));

INSERT INTO raion_grupe_vect
VALUES (2, 'r2', t_vect_categ('r21'));

INSERT INTO raion_grupe_vect (id_categorie, denumire)
VALUES (3,'r3');

UPDATE raion_grupe_vect
SET     grupe = t_vect_categ('r31','r32')
WHERE   id_categorie =3;

SELECT * FROM raion_grupe_vect;

SELECT id_categorie, denumire, b.*
FROM   raion_grupe_vect a, TABLE(a.grupe) b;

SELECT grupe
FROM   raion_grupe_vect
WHERE  id_categorie = 1;

SELECT *
FROM   TABLE(SELECT grupe
             FROM raion_grupe_vect
             WHERE id_categorie=1);

```

4.2.7. Colecții pe mai multe niveluri

- O colecție are o singură dimensiune. Pentru a modela o colecție multidimensională se definește o colecție ale cărei elemente sunt direct sau indirect colecții (*multilevel collections*).
 - Numărul nivelurilor de imbricare este limitat doar de capacitatea de stocare a sistemului.
- Colecții pe mai multe niveluri permise:
 - vectori de vectori;
 - vectori de tablouri imbricate;
 - tablouri imbricate de tablouri imbricate;
 - tablouri imbricate de vectori;
 - tablouri imbricate sau vectori de un tip definit de utilizator care are un atribut de tip tablou imbricat sau vector.
- Pot fi utilizate ca tipuri de date pentru definirea:
 - coloanelor unui tabel relațional;
 - variabilelor *PL/SQL*;
 - atributelor unui obiect într-un tabel obiect.

Exemplul 4.17

```

DECLARE
    type t_linie is VARRAY(3) OF INTEGER;
    type matrice IS VARRAY(3) OF t_linie;
    v_linie t_linie := t_linie(4,5,6);
    a_matrice := matrice(t_linie(1,2,3), v_linie);
BEGIN
    -- se adauga un element de tip vector matricei a (linie noua)
    a.EXTEND;
    -- se adauga valori elementului nou
    a(3) := t_linie(7,8);
    -- se extinde elementul nou
    a(3).EXTEND;
    -- se adauga valoare elementului nou
    a(3)(3) := 9;

    FOR i IN 1..3 LOOP
        FOR j IN 1..3 LOOP
            DBMS_OUTPUT.PUT(a(i)(j) || ' ');
        END LOOP;
        DBMS_OUTPUT.NEW_LINE;
    END LOOP;
END;

```

4.2.8. Compararea colecțiilor

- Două variabile de tip colecție nu pot fi comparate nativ, utilizând operatorii relaționali ($<$, \leq , $=$, \geq , \leq , $>$).
 - De exemplu, pentru a determina dacă o variabilă de tip colecție este mai mică decât alta se poate defini o funcție *PL/SQL* și se poate utiliza în locul operatorului „ $<$ ”.
- Variabilele de tip tablou indexat
 - nu pot fi comparate între ele sau cu valoarea *null*
- Variabilele de tip vector
 - pot fi comparate cu valoarea *null*
- Variabilele de tip tablou imbricat
 - pot fi comparate cu valoarea *null*
 - pot fi comparate între ele (doar dacă sunt egale sau diferite) utilizând funcții și operatori *SQL multiset*
- Funcții și operatori *SQL multiset*:
 - Funcția *CARDINALITY*
 - CARDINALITY (tablou_imbricat)*
 - Întoarce numărul de elemente al unui tablou imbricat.
 - Dacă tabloul este *null* sau nu are elemente, atunci întoarce *null*.
 - Funcția *SET*
 - SET (tablou_imbricat)*
 - Întoarce un tablou imbricat (de același tip cu argumentul său) în care păstrează doar elementele distincte (elimină duplicatele).
 - Operatorul *MULTISET EXCEPT*

```
tablou_imbricat_1
MULTISET EXCEPT [ ALL | DISTINCT ]
tablou_imbricat_2
▪ Întoarce un tablou imbricat ale cărui elemente sunt în tablou_imbricat_1, dar nu și în tablou_imbricat_2.
▪ Cele două tablouri trebuie să aibă același tip.
```
 - Operatorul *MULTISET UNION*

```
tablou_imbricat_1
MULTISET UNION [ ALL | DISTINCT ]
tablou_imbricat_2
▪ Întoarce un tablou imbricat ale cărui elemente apar în tablou_imbricat_1 sau în tablou_imbricat_2.
▪ Cele două tablouri trebuie să aibă același tip.
```

- Operatorul *MULTISET INTERSECT*

```
tablou_imbricat_1
MULTISET INTERSECT [ ALL | DISTINCT ]
tablou_imbricat_2
```

- Întoarce un tablou imbricat ale cărui elemente apar atât în *tablou_imbricat_1*, cât și în *tablou_imbricat_2*.
- Cele două tablouri trebuie să aibă același tip.

- Alți operatori:

- =, <>
- IN, NOT IN
- IS [NOT] A SET
- IS [NOT] EMPTY
- MEMBER OF
- [NOT] SUBMULTISET OF

Exemplul 4.18 - vezi curs

4.2.9. Prelucrarea colecțiilor stocate în tabele

- O colecție poate fi exploataată fie în întregime (atomic) utilizând comenzi *LMD*, fie pot fi prelucrate elemente individuale dintr-o colecție (*piecewise updates*) utilizând funcții/operatori *SQL* sau anumite facilități oferite de *PL/SQL*.
- Așa cum s-a observat în exemplele anterioare, se poate utiliza:
 - comanda *INSERT* pentru a insera o colecție într-o linie a unui tabel;
 - comanda *UPDATE* pentru a modifica o colecție stocată într-un tabel;
 - comanda *DELETE* pentru a șterge o linie a unui tabel ce conține o colecție;
 - comanda *SELECT* pentru a afișa sau a regăsi în variabile *PL/SQL* o colecție stocată într-un tabel.
- Vector stocat într-un tabel
 - este prelucrat ca un întreg (nu pot fi modificate elemente individuale)
 - elementele individuale nu pot fi referite de comenzi *INSERT*, *UPDATE*, *DELETE*
 - modificarea unui element individual se poate realiza doar din *PL/SQL*
 - se selectează vectorul într-o variabilă *PL/SQL*
 - se modifică valoarea variabilei
 - se inserează înapoi în tabel

- Tablou imbricat stocat într-un tabel
 - poate fi prelucrat ca întreg
 - inserări și actualizări asupra întregii colecții
 - poate fi prelucrat la nivel de elemente individuale
 - inserarea unor elemente noi în colecție
 - ștergerea unor elemente din colecție
 - actualizarea elementelor din colecție
- Pentru a putea prelucra elementele individuale ale unui tablou imbricat stocat într-un tabel se utilizează funcția *TABLE*.
- Funcția *TABLE*
 - Poate fi aplicată:
 - unei colecții
 - unei subcereri referitoare la o colecție (lista *SELECT* din subcerere trebuie să conțină o singură coloană de tip colecție și să întoarcă o singură linie din tabel).
 - Dacă este utilizată în clauză *FROM*, atunci permite interogarea colecției în mod asemănător unui tabel (exemplele 4.13 și 4.16) .

Exemplul 4.19 – continuare exemplu 4.13

```
-- selectie elemente colectie
SELECT *
FROM TABLE (SELECT grupe
              FROM raion_grupe_imb
              WHERE id_categorie = 1);

--adaugare element in colectie
INSERT INTO TABLE (SELECT grupe
                      FROM raion_grupe_imb
                      WHERE id_categorie = 1)
VALUES ('r13');

-- adaugare elemente obtinute cu subcerere
INSERT INTO TABLE (SELECT grupe
                      FROM raion_grupe_imb
                      WHERE id_categorie = 1)
SELECT denumire
      FROM categorii
      WHERE id_parinte = 1;

-- modificare element colectie
UPDATE TABLE (SELECT grupe
                  FROM raion_grupe_imb
                  WHERE id_categorie = 1) a
SET VALUE(a) = 'r1333'
WHERE COLUMN_VALUE = 'r13';
```

```
--stergere element colectie
DELETE FROM TABLE (SELECT grupe
                     FROM raion_grupe_imb
                     WHERE id_categorie = 1) a
WHERE COLUMN_VALUE = 'r1333';
```

- Pentru prelucrarea unei colecții locale se poate folosi și funcția *CAST*.

- Funcția *CAST*

- Convertește o colecție locală la tipul colecție specificat.
- Sintaxa

CAST ({nume_colecție | MULTISET (subcerere) }
 AS tip_colecție)

- *nume_colecție* este o colecție declarată local (de exemplu, într-un bloc *PL/SQL*)
- *subcerere* este o cerere *SQL* al cărui rezultat este transformat în colecție
- *tip_colecție* este un tip colecție *SQL*

- Funcția *COLLECT*

- Are ca argument o coloană de orice tip și întoarce un tablou imbricat format din liniile selectate.

COLLECT (coloană)

- Trebuie utilizată împreună cu funcția *CAST*.

Exemplul 4.20

```
SELECT CAST (COLLECT(denumire) AS t_imb_categ)
FROM categorii
WHERE id_parinte = 1;

SELECT CAST (MULTISET(SELECT denumire
                      FROM categorii
                      WHERE id_parinte=1)
            AS t_imb_categ)
FROM DUAL;
```

4.2.10. Procedeul *bulk collect*

- Execuția comenzilor *SQL* specificate în programe determină comutări ale controlului între motorul *PL/SQL* și motorul *SQL*. Prea multe astfel de schimbări de context afectează performanța.

- Pentru a reduce numărul de comutări între cele două motoare se utilizează procedeul *bulk collect*, care permite transferul liniilor între *SQL* și *PL/SQL* prin intermediul colecțiilor.
- Procedeul *bulk collect* implică doar două comutări între cele două motoare:
 - motorul *PL/SQL* comunică motorului *SQL* să obțină mai multe linii odată și să le plaseze într-o colecție;
 - motorul *SQL* regăsește toate liniile și le încarcă în colecție, după care predă controlul motorului *PL/SQL*.
- Sintaxa:

```
comandă_clauză BULK COLLECT INTO nume_colecție
                           [,nume_colecție]...
```

unde *comandă_clauză* poate fi

- comanda *SELECT* (cursoare implicate);
- comanda *FETCH* (cursoare explicite);
- clauza *RETURNING* a comenziilor *INSERT, UPDATE, DELETE*.

Exemplul 4.21

```
DECLARE
    TYPE t_ind IS TABLE OF categorii.id_categorie%TYPE
        INDEX BY PLS_INTEGER;
    TYPE t_imb IS TABLE OF categorii.denumire%TYPE;
    TYPE t_vec IS VARRAY(10) OF categorii.nivel%TYPE;
    v_ind t_ind;
    v_imb t_imb;
    v_vec t_vec;

BEGIN
    SELECT id_categorie, denumire, nivel
    BULK COLLECT INTO v_ind, v_imb, v_vec
    FROM categorii
    WHERE ROWNUM <= 10;

    FOR i IN 1..10 LOOP
        DBMS_OUTPUT.PUT(v_ind(i) || ' ' || v_imb(i) ||
                        ' ' || v_vec(i));
        DBMS_OUTPUT.NEW_LINE;
    END LOOP;
END;
```

4.2.11. Procedeul *bulk bind*

- În exemplul următor datele menținute într-o colecție sunt inserate în tabel.
 - Colecția este parcursă folosind comanda *FOR*.
 - La fiecare iterație o comandă *INSERT* este transmisă motorului *SQL*.

Exemplul 4.22

```

DECLARE
    TYPE tab_ind IS TABLE OF tip_plata%ROWTYPE
        INDEX BY PLS_INTEGER;
    t    tab_ind;
BEGIN
    -- atribuire valori
    DELETE FROM tip_plata
    WHERE id_tip_plata NOT IN (SELECT id_tip_plata
                                FROM facturi)
    RETURNING id_tip_plata, cod, descriere BULK COLLECT INTO t;

    -- insert in tabel
    FOR i IN t.FIRST..t.LAST LOOP
        INSERT INTO tip_plata VALUES t(i);
    END LOOP;
END;
  
```

- Procedeul *bulk bind* permite transferul liniilor din colecție printr-o singură operație.
 - Este realizat cu ajutorul comenzi *FORALL*

```
FORALL index IN lim_inf..lim_sup [SAVE EXCEPTIONS]
    comandă_sql;
```

 - comandă_sql* poate fi o comandă *INSERT*, *UPDATE* sau *DELETE* care referă elementele unei colecții (de orice tip)
 - variabila *index* poate fi referită numai ca indice de colecție
 - Restricții de utilizare a comenzi *FORALL*
 - comanda poate fi folosită numai în programe *server-side*
 - comandă_sql* trebuie să refere cel puțin o colecție
 - toate elementele colecției din domeniul precizat trebuie să existe
 - indicii colecțiilor nu pot fi expresii și trebuie să aibă valori continue

Exemplul 4.23

```

DECLARE
    TYPE tab_ind IS TABLE OF tip_plata%ROWTYPE
        INDEX BY PLS_INTEGER;
    t    tab_ind;
BEGIN
    -- atribuire valori
    DELETE FROM tip_plata
    WHERE id_tip_plata NOT IN (SELECT id_tip_plata
                                FROM facturi)
    RETURNING id_tip_plata, cod, descriere BULK COLLECT INTO t;

    -- insert in tabel
    FORALL i IN t.FIRST..t.LAST
        INSERT INTO tip_plata VALUES t(i);
END;

```

- Cursorul *SQL* are un atribut compus `%BULK_ROWCOUNT` care numără liniile afectate de operațiile comenzi *FORALL*.
 - `SQL%BULK_ROWCOUNT(i)` reprezintă numărul de liniî procesate de a *i*-a execuție a comenzi *SQL*.
 - Dacă nu este afectată nici o linie, valoarea atributului este 0.
 - `%BULK_ROWCOUNT` nu poate fi parametru în subprograme și nu poate fi asignat altor colecții.

Exemplul 4.24

```

CREATE TABLE produse_copie AS SELECT * FROM PRODUSE;

DECLARE
    TYPE tip_vec IS VARRAY(3) OF NUMBER(4);
    v tip_vec := tip_vec(800, 900, 1000);
BEGIN
    FORALL i IN 1..3
        DELETE FROM produse_copie
        WHERE id_categorie = v(i);

    FOR j in 1..v.LAST LOOP
        DBMS_OUTPUT.PUT_LINE( 'Numar liniî procesate la pasul ' ||
                             j || ':' || SQL%BULK_ROWCOUNT(j));
    END LOOP;
END;
/
ROLLBACK;

```

- Dacă există o eroare în procesarea unei liniî printr-o operație *LMD* de tip *bulk*, numai acea linie va fi *rollback*.

- Clauza *SAVE EXCEPTIONS*, permite ca toate excepțiile care apar în timpul execuției comenzi *FORALL* să fie salvate și astfel procesarea poate să continue.
 - Atributul cursor *%BULK_EXCEPTIONS* poate fi utilizat pentru a vizualiza informații despre aceste excepții.
 - Atributul acționează ca un tablou *PL/SQL* și are două câmpuri:
 - *%BULK_EXCEPTIONS(i).ERROR_INDEX*, reprezentând iterarea în timpul căreia s-a declanșat excepția;
 - *%BULK_EXCEPTIONS(i).ERROR_CODE*, reprezentând codul *Oracle* al erorii respective.

Exemplul 4.25 - vezi curs

4.3. Vizualizări din dicționarul datelor

- Vizualizări care conțin informații despre tipurile de date create de utilizatori:
 - *USER_TYPES*
 - *USER_TYPE_ATTRS*

Bibliografie

1. *Programare avansată în Oracle9i*, I. Popescu, A. Alecu, L. Velcescu, G. Florea (Mihai), Ed. Tehnică (2004)
2. *Oracle Database PL/SQL Language Reference 11g Release 2*, Oracle Online Documentation (2012)
3. *Oracle Database SQL Language Reference 11g Release 2*, Oracle Online Documentation (2012)
4. *Oracle Database 11g: PL/SQL Fundamentals, Student Guide*, Oracle University (2009)
5. *Pro*C/C++ Programmer's Guide 10g Release 2 (10.2)*, Oracle Online Documentation (2012)
6. *Oracle Data Types & Subtypes*
(<http://psoug.org/reference/datatypes.html>)
7. *Oracle Conversion Functions Version 11.1*
(http://psoug.org/reference/convert_func.html)
8. *Collection extensions in 10g*
(<http://www.oracle-developer.net/display.php?id=303>)
9. MySQL Online Documentation
(<http://dev.mysql.com>)
10. Microsoft Online Documentation
(<http://msdn.microsoft.com>)

CUPRINS

5. PL/SQL – Gestiunea cursoarelor	2
5.1. Cursoare implicite.....	4
5.2. Cursoare explicite	4
5.2.1. Gestiunea cursoarelor explicite	5
5.2.2. Cursoare parametrizate	10
5.2.3. Cursoare <i>SELECT FOR UPDATE</i>	11
5.2.4. Cursoare dinamice	14
Bibliografie	17

5. PL/SQL – Gestiunea cursoarelor

- Un cursor este un pointer către o zonă de memorie (*Private SQL Area*) care stochează informații despre procesarea unei comenzi *SELECT* sau *LMD*.



În acest capitol se discută cursoarele la nivel sesiune.

- Cursoarele la nivel de sesiune:
 - sunt diferite față de cursoarele ce folosesc zonă privată *SQL* din *PGA* (*Program Global Area*);
 - există în memoria alocată sesiunii până la momentul încheierii acesteia;
- Vizualizarea *V\$OPEN_CURSOR* oferă informații despre cursoarele deschise la nivel de sesiune ale fiecărei sesiuni utilizator.



În continuare, din motive de simplificare a exprimării, pentru un cursor la nivel de sesiune se va utiliza termenul de cursor.

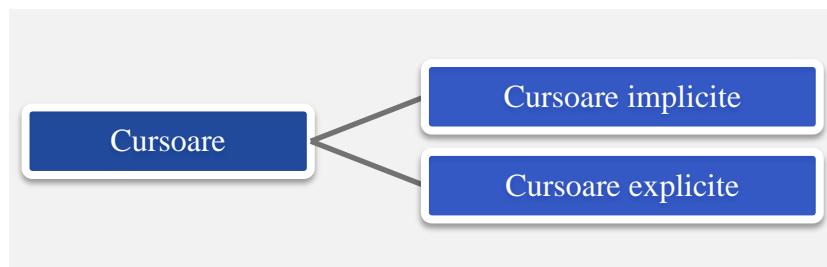


Fig. 5.1. Tipuri de cursoare

- Categorii de cursoare:
 - cursoare implicate
 - construite și gestionate automat de *PL/SQL*
 - cursoare explicite
 - construite și gestionate de către utilizatori.
- Atributele care furnizează informații despre cursoare:
 - pot fi referite doar de comenzi procedurale
 - pot fi referite utilizând sintaxa
 - pentru cursoarele implicate

SQL%nume_atribut
 - pentru cursoarele explicite

nume_cursor%nume_atribut

- lista atributelor:
 - **%ROWCOUNT**
 - este de tip întreg (*PLS_INTEGER*);
 - are valoarea *NULL* dacă nu a fost rulată nicio comandă *SELECT* sau *LMD*;
 - reprezintă numărul liniilor întoarse de ultima comandă *SELECT* sau numărul de linii afectate de ultima comandă *LMD*;
 - dacă numărul de linii este mai mare decât valoarea maximă permisă de tipul *PLS_INTEGER* (2.147.483.647), atunci întoarce o valoare negativă.
 - **%FOUND**
 - este de tip boolean;
 - are valoarea *NULL* dacă nu a fost rulată nicio comandă *SELECT* sau *LMD*;
 - în cazul cursoarelor implicite are valoarea *TRUE* dacă ultima comandă *SELECT* a întors cel puțin o linie sau ultima comandă *LMD* a afectat cel puțin o linie;
 - în cazul cursoarelor explicate are valoarea *TRUE* dacă ultima operație de încărcare (*FETCH*) dintr-un cursor a avut succes.
 - **%NOTFOUND**
 - este de tip boolean;
 - are semnificație opusă față de cea a atributului **%FOUND**.
 - **%ISOPEN**
 - este de tip boolean;
 - indică dacă un cursor este deschis;
 - în cazul cursoarelor implicite, acest atribut are întotdeauna valoarea *FALSE*, deoarece un cursor implicit este închis de sistem imediat după execuția instrucțiunii *SQL* asociate.
 - **%BULK_ROWCOUNT**
 - vezi în Capitolul 4 comanda *FORALL*
 - **%BULK_EXCEPTIONS**
 - vezi în Capitolul 4 comanda *FORALL*

5.1. Cursoare implicite

- *PL/SQL* deschide automat un cursor implicit la nivel de sesiune de fiecare dată când este rulată o comandă *SELECT* sau *LMD*.
- Mai sunt denumite și cursoare *SQL*.
- Cursorul implicit este închis automat, atunci când comanda se încheie.
- Valorile atributelor asociate cursorului rămân disponibile până când este rulată o altă comandă *SELECT* sau *LMD*.

Exemplul 5.1 – vezi curs

- 
- ❖ Atributul *SQL%NOTFOUND* nu este util în cazul comenzi *SELECT INTO*.
 - ❖ Dacă această comandă nu întoarce linii, atunci apare imediat excepția *NO_DATA_FOUND* (înainte să se poată verifica valoarea atributului *SQL%NOTFOUND*).
 - ❖ Dacă în lista *SELECT* a comenzi se utilizează funcții agregat, atunci este întoarsă întotdeauna o valoare. În acest caz, valoarea atributului *SQL%NOTFOUND* este *FALSE*.
- 
- Dacă o comandă *SELECT INTO* (nu este folosită clauza *BULK COLLECT*) întoarce mai multe linii, atunci apare imediat excepția *TOO_MANY_ROWS*. În acest caz, atributul *SQL%ROWCOUNT* are valoarea 1 (nu numărul de linii care satisfac cererea).

5.2. Cursoare explicite

- Sunt cursoare la nivel de sesiune definite și gestionate de către utilizatorii.
- Un cursor explicit are specificat un nume. Aceasta este asociat cu o comandă *SELECT* ce întoarce de obicei mai multe linii.
- Mulțimea rezultat a cererii asociate poate fi procesată folosind una dintre variantele următoare:
 - se deschide cursorul (comanda *OPEN*), se încarcă liniile cursorului în variabile (comanda *FETCH*), se închide cursorul (comanda *CLOSE*);
 - se utilizează cursorul într-o comandă *FOR LOOP*.

5.2.1. Gestiunea cursoarelor explicite

```

DECLARE
    declarare cursor
BEGIN
    deschidere cursor (OPEN)
    WHILE rămân linii de recuperat LOOP
        recuperare linie rezultat (FETCH)
    END LOOP
    închidere cursor (CLOSE)
END;

```

Declararea unui cursor explicit

- Sintaxa de declarare, fără a asocia comanda *SELECT*

```
CURSOR nume_cursor [RETURN tip];
```

- Sintaxa de declarare, cu asociere a comanda *SELECT*

```
CURSOR nume_cursor [RETURN tip]
IS comanda_SELECT;
```

Exemplul 5.2

```

DECLARE
    CURSOR c1 RETURN produse%ROWTYPE;

    CURSOR c2 IS
        SELECT id_produs, denumire FROM produse;

    CURSOR c1 RETURN produse%ROWTYPE IS
        SELECT * FROM PRODUSE;

BEGIN
    NULL;
END;

```



- ❖ Numele cursorului este un identificator unic în cadrul blocului, care nu poate să apară într-o expresie și căruia nu i se poate atribui o valoare.
- ❖ Comanda *SELECT* care apare în declararea cursorului, nu trebuie să includă clauza *INTO*.
- ❖ Dacă se cere procesarea liniilor într-o anumită ordine, atunci în cerere este utilizată clauza *ORDER BY*.

- ❖ Variabilele care sunt referite în comanda *SELECT* trebuie declarate înaintea comenzi *CURSOR*. Acestea sunt considerate variabile de legătură.
- ❖ Dacă în lista *SELECT* apare o expresie, atunci pentru expresia respectivă trebuie utilizat un alias, iar câmpul expresie se va referi prin acest alias.

Deschiderea unui cursor explicit

- Sintaxa

```
OPEN nume_cursor;
```

- Comanda *OPEN* execută cererea asociată cursorului, identifică mulțimea liniilor rezultat (mulțimea activă) și poziționează cursorul înaintea primei linii.
- Dacă se încearcă deschiderea unui cursor deja deschis, atunci pare excepția *CURSOR_ALREADY_OPEN*.
- La deschiderea unui cursor se realizează următoarele operații:
 - se alocă resursele necesare pentru a procesa cererea
 - se procesează cererea
 - se evaluează comanda *SELECT* asociată (sunt examineate valorile variabilelor de legătură ce apar în declarația cursorului)
 - se identifică mulțimea activă prin execuția cererii *SELECT*, având în vedere valorile de la pasul anterior;
 - dacă cererea include clauza *FOR UPDATE*, atunci liniile din mulțimea activă sunt blocate;
 - se poziționează *pointer*-ul înaintea primei linii din mulțimea activă.

Exemplul 5.3

```
DECLARE

    CURSOR c1 IS
        SELECT * FROM categorii WHERE id_parinte IS NULL;

    CURSOR c2 IS
        SELECT * FROM categorii WHERE 1=2;

BEGIN
    OPEN c1;
    IF c1%FOUND THEN
        DBMS_OUTPUT.PUT_LINE('c1 - cel putin o linie');
    ELSE
        DBMS_OUTPUT.PUT_LINE('c1 - nicio linie');
    END IF;

```

```

OPEN c2;
IF c2%NOTFOUND THEN
    DBMS_OUTPUT.PUT_LINE('c2 - nicio linie');
ELSE
    DBMS_OUTPUT.PUT_LINE('c2 - cel putin o linie');
END IF;

CLOSE c1;
CLOSE c2;
END;

```

Încărcarea datelor dintr-un cursor explicit

- Comanda *FETCH* regăsește liniile rezultatului din mulțimea activă.
- Sintaxa

```

FETCH nume_cursor INTO {nume_variabilă
    [, nume_variabilă] ... | nume_înregistrare};

FETCH nume_cursor BULK COLLECT INTO
    {nume_variabilă_colecție
        [, nume_variabilă_colecție]}

```

- *FETCH* realizează următoarele operații:
 - avansează *pointer*-ul la următoarea linie în mulțimea activă (*pointer*-ul poate avea doar un sens de deplasare de la prima înregistrare spre ultima);
 - citește datele liniei curente în variabile *PL/SQL*;
 - dacă *pointer*-ul este poziționat la sfârșitul mulțimii active, atunci se ieșe din bucla cursorului.



Comanda *FETCH INTO* regăsește la un moment dat o singură linie.

Comanda *FETCH BULK COLLECT INTO* încarcă la un moment mai multe linii în colecții.

Exemplul 5.4 – vezi curs

Exemplul 5.5 – vezi curs



- ❖ Atunci când un cursor încarcă o linie, acesta realizează o „schimbare de context” – controlul este preluat de motorul *SQL* care va obține datele. Motorul *SQL* plasează datele în memorie și are loc o altă „schimbare de context” – controlul este preluat înapoi de motorul *PL/SQL*. Procesul se repetă până când nu mai sunt date de încărcat. Schimbările de context sunt foarte rapide, dar numărul prea mare de astfel de operații poate implica performanță scăzută.
- ❖ Folosind metoda *BULK COLLECT* sunt obținute mai multe linii, implicând doar 2 schimbări de context.
- ❖ Începând cu *Oracle 10g*, un cursor poate determina ca *PL/SQL* să realizeze implicit operații *BULK COLLECT*, încarcând câte 100 linii la un moment dat, fără a mai fi necesară utilizarea colecțiilor. În acest caz, utilizarea colecțiilor se poate dovedi utilă, doar dacă sunt încărcate mai multe sute de linii.

Exemplul 5.6 [- vezi curs](#)

Exemplul 5.7 [- vezi curs](#)

Închiderea unui cursor explicit

- După ce a fost procesată mulțimea activă, cursorul trebuie închis.
 - *PL/SQL* este informat că programul a terminat folosirea cursorului și resursele asociate acestuia pot fi eliberate:
 - spațiul utilizat pentru memorarea mulțimii active;
 - spațiul temporar folosit pentru determinarea mulțimii active.
- Sintaxa:

```
CLOSE nume_cursor;
```

- Pentru a reutiliza cursorul este suficient ca acesta să fie redeschis.
- Dacă se încearcă încărcarea datelor dintr-un cursor închis, atunci apare excepția *INVALID_CURSOR*.



- ❖ Dacă un bloc *PL/SQL* să termină fără a închide un cursor utilizat, sistemul nu va returna o eroare sau un mesaj de avertizare.
- ❖ Se recomandă închiderea cursoarelor pentru a permite sistemului să elibereze resursele alocate.

- Valorile atributelor unui cursor explicit sunt prezentate în următorul tabel:

	OPEN		Primul FETCH		Următorul FETCH		Ultimul FETCH		CLOSE	
	Înainte	După	Înainte	După	Înainte	După	Înainte	După	Înainte	După
%ISOPEN	False	True	True	True	True	True	True	True	True	False
%FOUND	Eroare	Null	Null	True	True	True	True	Flase	False	Eroare
%NOTFOUND	Eroare	Null	Null	False	False	False	False	True	True	Eroare
%ROWCOUNT	Eroare	0	0	1	1	Depinde de date				Eroare

- După prima încărcare, dacă mulțimea rezultat este vidă, **%FOUND** va fi *FALSE*, **%NOTFOUND** va fi *TRUE*, iar **%ROWCOUNT** este 0.

Procesarea liniilor unui cursor explicit

- Se utilizează o comandă de ciclare (*LOOP*, *WHILE* sau *FOR*), prin care la fiecare iterare se va încărca o linie nouă.
- Pentru ieșirea din ciclu poate fi utilizată comanda *EXIT*.
- Utilizarea comenzii de ciclare *LOOP*
 - vezi exemplul 5.4
- Utilizarea comenzii de ciclare *WHILE*
 - vezi exemplul 5.5
- ❖ Dacă se utilizează una dintre comenziile de ciclare *LOOP* sau *WHILE*, atunci cursorul trebuie:
 1. declarat
 2. deschis
 3. parcurs, încărcând câte o linie la fiecare iterare (trebuie să se asigure ieșirea din buclă atunci când nu mai sunt linii de procesat)
 4. închis
- Utilizarea comenzii de ciclare *FOR*
 - Procesarea liniilor unui cursor explicit se poate realiza și cu ajutorul unui ciclu *FOR* special, numit **ciclu cursor**.
 - În acest caz cursorul trebuie doar declarat, operațiile de deschidere, încărcare și închidere ale acestuia fiind implicate.



Dacă se utilizează una dintre comenziile de ciclare *LOOP* sau *WHILE*, atunci cursorul trebuie:

1. declarat

2. deschis

3. parcurs, încărcând câte o linie la fiecare iterare (trebuie să se asigure ieșirea din buclă atunci când nu mai sunt linii de procesat)

4. închis

- Utilizarea comenzii de ciclare *FOR*

○ Procesarea liniilor unui cursor explicit se poate realiza și cu ajutorul unui ciclu *FOR* special, numit **ciclu cursor**.

▪ În acest caz cursorul trebuie doar declarat, operațiile de deschidere, încărcare și închidere ale acestuia fiind implicate.

- Sintaxa:

```
FOR nume_înregistrare IN nume_cursor LOOP
    secvență_de_instrucțiuni;
END LOOP;
```

- Variabila *nume_înregistrare* nu trebuie declarată.

Exemplul 5.8 - vezi curs

- Există ciclu cursoare speciale care în comanda *FOR* în loc să refere un cursor declarat, utilizează direct o subcerere (**ciclu cursor cu subcereri**).
 - În acest caz nu este necesară nici măcar declararea cursorului.

Exemplul 5.9 - vezi curs

5.2.2. Cursoare parametrizate

- Unei variabile de tip cursor îi corespunde o comandă *SELECT*, care nu poate fi modificată pe parcursul programului.
- Cursoarele parametrizate sunt cursoare ale căror comenzi *SELECT* depind de parametri ce pot fi modificați la momentul execuției.
 - Transmiterea de parametri unui cursor parametrizat se face în mod similar procedurilor stocate.

Declararea unui cursor parametrizat

- Sintaxa de declarare, fără a asocia comanda *SELECT*

```
CURSOR nume_cursor (declarare_parametru
                      [, declarare_parametru ...])
[RETURN tip];
```

- Sintaxa de declarare, cu asocierea comenzi *SELECT*

```
CURSOR nume_cursor (declarare_parametru
                      [, declarare_parametru ...])
[RETURN tip]
IS comanda_SELECT;
```

- *declarare_parametru* are sintaxa:

```
nume_parametru [IN] tip_date_scalar
[ {:= | DEFAULT} expresie]
```

- Parametrul unui cursor nu poate fi declarat *NOT NULL*.

Deschiderea unui cursor parametrizat

- Se realizează asemănător apelului unei funcții, specificând lista parametrilor actuali ai cursorului.
 - Asocierea dintre parametrii formali și cei actuali se face prin:
 - poziție (parametrii actuali sunt separați prin virgulă, respectând ordinea parametrilor formali);
 - nume (parametrii actuali sunt aranjați într-o ordine arbitrară, dar cu o corespondență de forma *parametru formal => parametru actual*).
 - Dacă în definiția cursorului, toți parametrii au valori implicate (*DEFAULT*), cursorul poate fi deschis fără a specifica vreun parametru.
- În determinarea mulțimii active se vor folosi valorile actuale ale parametrilor.
- Sintaxa

```
OPEN nume_cursor
  [ (valoare_parametru [, valoare_parametru] ...) ];
```

Procesarea liniilor unui cursor parametrizat

- Dacă pentru procesare sunt utilizate comenzi de ciclare *LOOP* sau *WHILE*, atunci nu apar modificări de sintaxă.
- Dacă este utilizat un ciclu cursor, atunci se va utiliza sintaxa:

```
FOR nume_înregistrare IN nume_cursor
  [ (valoare_parametru [, valoare_parametru] ...) ] LOOP
    secvență_de_instrucțiuni;
END LOOP;
```

Închiderea unui cursor parametrizat

- Nu apar modificări de sintaxă.

Exemplul 5.10 – vezi curs

5.2.3. Cursoare *SELECT FOR UPDATE*

- Dacă este necesară blocarea liniilor înainte ca acestea să fie șterse sau reactualizate, atunci blocarea se poate realiza cu ajutorul clauzei *FOR UPDATE* a comenzi *SELECT* din definiția cursorului.
 - Cursorul trebuie să fie deschis.

- Sintaxa

```
CURSOR nume_cursor IS
    comanda_select
FOR UPDATE [OF listă_coloane]
    [NOWAIT | WAIT n | SKIP LOCKED];
```

- Identifierul *listă_coloane* este o listă ce include câmpurile tabelului care vor fi modificate.
 - Coloanele incluse în această listă indică doar liniile cărui tabel vor fi blocate.
 - Dacă lista de coloane lipsește, atunci vor fi blocate liniile selectate din toate tabelele referite în cerere.
- Implicit comanda așteaptă până când linia necesară devine disponibilă și apoi întoarce rezultatul cererii.
- Pentru a modifica acest comportament se poate utiliza una dintre opțiunile:
 - *NOWAIT* – nu așteaptă deblocarea liniei și întoarce o eroare dacă liniile sunt deja blocate de altă sesiune;
 - *WAIT n* – așteaptă *n* secunde (*n* este de tip întreg) pentru deblocarea liniei, iar dacă linia nu este deblocată în acest interval, întoarce un mesaj de eroare.
 - *SKIP LOCKED* – se va încerca blocarea liniilor selectate de cerere, iar liniile care sunt deja blocate de o altă tranzacție vor fi sărite (opțiune utilizată de exemplu în *Oracle Streams Advanced Queuing*).

Exemplul 5.11 - [vezi explicatii curs](#)

```
--sesiune 1
SELECT * FROM produse
WHERE id_produs=10 FOR UPDATE;
--commit;

--sesiune 2
SELECT * FROM curs_plsql.produse
WHERE id_produs=10
FOR UPDATE NOWAIT;

SELECT * FROM curs_plsql.produse
WHERE id_produs=1000
FOR UPDATE WAIT 10;
```



- ❖ În momentul deschiderii unui cursor *FOR UPDATE*, liniile din mulțimea activă, determinată de clauza *SELECT*, sunt blocate pentru operații de scriere (reactualizare sau ștergere). În felul acesta este realizată consistența la citire a sistemului.
- ❖ De exemplu, această situație este utilă atunci când se reactualizează o valoare a unei linii și trebuie avută siguranță că linia nu este schimbată de un alt utilizator înaintea reactualizării. Astfel, alte sesiuni nu pot schimba liniile din mulțimea activă până când tranzacția nu este permanentizată sau anulată.
- Dacă un cursor este declarat folosind clauza *FOR UPDATE*, atunci comenziile *DELETE/UPDATE* corespunzătoare trebuie să conțină clauza *WHERE CURRENT OF nume_cursor*.
 - Clauza referă linia curentă care a fost găsită de cursor, permitând ca reactualizările și ștergerile să se efectueze asupra acestei linii, fără referirea explicită a cheii primare sau pseudocoloanei *ROWID*.
- ❖ Deoarece cursorul lucrează doar cu niște copii ale liniilor existente în tabele, după închiderea cursorului este necesară comanda *COMMIT* pentru a realiza scrierea efectivă a modificărilor.
- ❖ Deoarece blocările implicate de clauza *FOR UPDATE* vor fi eliberate de comanda *COMMIT*, nu este recomandată utilizarea comenzi *COMMIT* în interiorul ciclului în care se fac încărcări de date. Orice *FETCH* executat după *COMMIT* va eșua.
- ❖ În cazul în care cursorul nu este definit folosind *SELECT...FOR UPDATE*, nu apar probleme în acest sens și, prin urmare, în interiorul ciclului unde se fac schimbări ale datelor poate fi utilizată comanda *COMMIT*.

Exemplul 5.12 – **vezi curs**

Exemplul 5.13 – **vezi curs**

5.2.4. Cursoare dinamice

- Un cursor static este un cursor a cărui comandă *SQL* este cunoscută la momentul compilării blocului.
 - Toate exemplele anterioare se referă la cursoare statice.
- În *PL/SQL* a fost introdus conceptul de variabilă cursor, care este de tip referință.
- Variabilele cursor
 - sunt similare tipului *pointer* din limbajele *C* sau *Pascal*
 - un cursor este un obiect static, iar un cursor dinamic este un *pointer* la un cursor
 - sunt dinamice deoarece li se pot asocia diferite cereri (coloanele obținute de fiecare cerere trebuie să corespundă declarației variabilei cursor)
 - trebuie declarate, deschise, încărcate și închise în mod similar unui cursor static
 - la momentul declarării nu solicită o cerere asociată
 - pot primi valori
 - pot fi utilizate în expresii
 - pot fi utilizate ca parametrii în subprograme
 - pot fi utilizate pentru a transmite mulțimea rezultat a unei cereri între subprograme
 - pot fi variabile de legătură
 - pot fi utilizate pentru a transmite mulțimea rezultat a unei cereri între subprograme stocate și diferenți clienți
 - nu acceptă parametrii
- Sintaxa de declarare

```
TYPE tip_ref_cursor IS REF CURSOR [RETURN tip_returnat];
var_cursor tip_ref_cursor;
```

- *var_cursor* este numele variabilei cursor
- *tip_ref_cursor* este un nou tip de date ce poate fi utilizat în declarațiile următoare ale variabilelor cursor
- *tip_returnat* este un tip înregistrare sau tipul unei linii dintr-un tabel
 - corespunde coloanelor întoarse de către orice cursor asociat variabilelor cursor de tipul definit

- dacă lipsește clauza *RETURN*, cursorul poate fi deschis pentru orice cerere
- Restricții de utilizare a variabilelor cursor
 - variabilele cursor nu pot fi declarate în specificația unui pachet
 - un pachet nu poate avea definită o variabilă cursor ce poate fi referită din afară pachetului
 - valoarea unei variabile cursor nu poate fi stocată într-o colecție sau o coloană a unui tabel
 - nu pot fi utilizate operatorii de comparare pentru a testa egalitatea, inegalitatea sau valoarea *null* a variabilelor cursor
 - nu pot fi folosite cu *SQL dynamic* în *Pro*C/C++*

Utilizarea unei variabile cursor

- Comanda *OPEN...FOR* asociază o variabilă cursor cu o cerere, execută cererea, identifică mulțimea rezultat și poziționează cursorul înaintea primei linii din mulțimea rezultat.
- Sintaxa

```
OPEN {variabila_cursor | :variabila_cursor_host}
FOR {cerere_select |
      sir_dinamic [USING argument_bind [, argument_bind ...]]};
```

- *variabila_cursor* specifică o variabilă cursor declarată anterior
- *cerere_select* reprezintă cererea pentru care este deschisă variabila cursor
- *sir_dinamic* este o secvență de caractere care reprezintă cererea
 - este specifică prelucrării dinamice a comenziilor, iar posibilitățile oferite de *SQL dynamic* vor fi analizate într-un capitol separat
- *:variabila_cursor_host* reprezintă o variabilă cursor declarată într-un mediu gazdă *PL/SQL*
- Comanda *OPEN .. FOR* poate deschide același cursor pentru diferite cereri. Nu este necesară închiderea variabilei cursor înainte de a o redeschide. Dacă se redeschide variabila cursor pentru o nouă cerere, cererea anterioară este pierdută.

Exemplul 5.14 – [vezi curs](#)

Exemplul 5.15 – [vezi curs](#)

Expresii cursor

- În versiunea *Oracle9i* a fost introdus conceptul de expresie cursor, care întoarce un cursor imbricat (*nested cursor*).
- Sintaxa:

CURSOR (subcerere)
- Semnificație
 - Fiecare linie din mulțimea rezultat poate conține valori uzuale și cursoare generate de subcereri.
- Utilizare
 - *PL/SQL* acceptă cereri care au expresii cursor în cadrul unei declarații cursor, declarații *REF CURSOR* și a variabilelor cursor.
 - Expresia cursor poate să apară într-o comandă *SELECT* ce este utilizată pentru deschiderea unui cursor dinamic.
 - Expresia cursor poate fi utilizată în cereri SQL dinamice sau ca parametri actuali într-un subprogram.
 - Restricții de utilizare a unei expresii cursor
 - nu poate fi utilizată cu un cursor implicit;
 - poate să apară numai într-o comandă *SELECT* care nu este imbricată în altă cerere (exceptând cazul în care este o subcerere chiar a expresiei cursor) sau ca argument pentru funcții tabel, în clauza *FROM* a lui *SELECT*;
 - nu poate să apară în interogarea ce definește o vizualizare;
 - nu se pot efectua operații *BIND* sau *EXECUTE* cu aceste expresii.
- Încărcarea cursorului imbricat se realizează
 - automat atunci când liniile care îl conțin sunt încărcate din cursorul „părinte“.
- Închiderea cursorului imbricat are loc
 - dacă este realizată explicit de către utilizator;
 - atunci când cursorul „părinte“ este reexecutat sau închis;
 - dacă apare o eroare în timpul unei încărcări din cursorul „părinte“.

Exemplul 5.16 – **vezi curs**

Bibliografie

1. *Programare avansată în Oracle9i*, I. Popescu, A. Alecu, L. Velcescu, G. Florea (Mihai), Ed. Tehnică (2004)
2. *Oracle Database PL/SQL Language Reference 11g Release 2*, Oracle Online Documentation (2012)
3. *Oracle Database SQL Language Reference 11g Release 2*, Oracle Online Documentation (2012)
4. *Oracle Database 11g: PL/SQL Fundamentals, Student Guide*, Oracle University (2009)

CUPRINS

6. PL/SQL – Subprograme	2
6.1. Proceduri PL/SQL.....	4
6.1.1. Definirea unei proceduri	4
6.1.2. Apelarea unei proceduri	6
6.1.3. Transferul parametrilor	7
6.2. Funcții PL/SQL.....	8
6.2.1. Definirea unei funcții	8
6.2.2. Apelarea unei funcții.....	10
6.2.3. Utilizarea în expresii SQL a funcțiilor definite de utilizator.....	11
6.3. Recompilarea subprogramelor PL/SQL.....	12
6.4. Ștergerea subprogramelor PL/SQL.....	12
6.5. Subprograme overload	12
6.6. Recursivitate	13
6.7. Declarații forward.....	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.

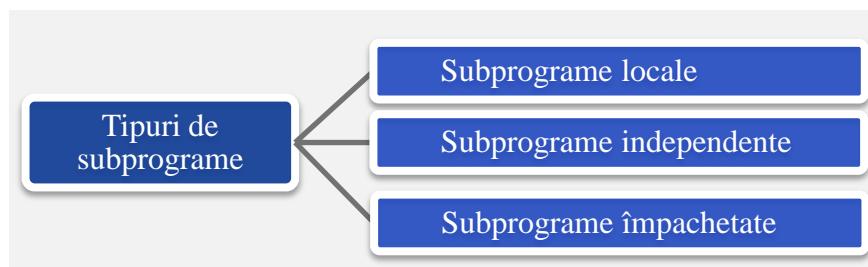


Fig. 6.1. Tipuri de subprograme

- 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 *database buffer cache*.
 - 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 recompliat 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 *database buffer cache* (dacă acestea nu erau deja acolo). Dacă pachetul este activat pentru prima oară într-o sesiune, atunci *server-ul* va executa blocul de initializare 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 implicate (*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 comenzi 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 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 traекторiile sale conduc la comanda *RETURN*. Dacă o traекторie 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 aceleși argumente la apelarea sa. Dacă sunt realizate apeluri repetitive ale funcției, având aceleși 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 comenziile *SQL* pot fi utilizate doar funcții cu parametrii 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 comenzi *SELECT*;
 - clauzele *WHERE* și *HAVING*;
 - clauzele *CONNECT BY*, *START WITH*, *ORDER BY* și *GROUP BY*;
 - clauza *VALUES* a comenzi *INSERT*;
 - clauza *SET* a comenzi *UPDATE*.

Exemplul 6.14 – vezi curs



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

- ❖ să aibă numai parametrii 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 tranzacții (nu pot utiliza comenziile *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* (suprîncărcate) au aceeași 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 formalii 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 formalii;
 - 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 comenziile *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 incorrect
  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 indirekte 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 aceeași 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*). Aceasta 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 signură 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 signatura 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 nouă 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 *;
- includând lista coloanelor în cadrul comenzi *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 distanță, atunci se face recompilare automată, dar la a doua reexecuție. Este preferabilă o recompilare manuală pentru prima reexecuție sau implementarea unei strategii de reinvoacare 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 limbiage care realizează performant acțiunile respective. În acest caz este necesară comunicarea între diferite module ale aplicației care sunt scrise în limbiage 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 comenzi *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 limbi 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 comenzi 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 signatura 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 comenzi: *CREATE JAVA instrucțiune*.

Bibliografie

1. *Programare avansată în Oracle9i*, I. Popescu, A. Alecu, L. Velcescu, G. Florea (Mihai), Ed. Tehnică (2004)
2. *Oracle Database PL/SQL Language Reference 11g Release 2*, Oracle Online Documentation (2012)
3. *Oracle Database SQL Language Reference 11g Release 2*, Oracle Online Documentation (2012)
4. *Oracle Database 11g: PL/SQL Fundamentals, Student Guide*, Oracle University (2009)

CUPRINS

7. PL/SQL – Pachete	2
7.1. Definirea pachetelor	3
7.1.1. Specificația pachetului	6
7.1.2. Corpul pachetului	6
7.1.3. Instantierea pachetului	7
7.2. Modificare,ștergerea și modificarea pachetelor	8
7.3. Pachete predefinite	11
7.3.1. Pachetul STANDARD	12
7.3.2. Pachetul DBMS_OUTPUT	13
7.3.3. Pachetul DBMS_JOB	13
7.3.4. Pachetul UTL_FILE	15
7.3.5. Pachetul DBMS_SQL	16
7.3.6. Pachetul DBMS_DDL	19
Bibliografie	22

7. PL/SQL – Pachete

- Un pachet (*package*) permite încapsularea într-o unitate logică a:
 - constantelor și variabilelor;
 - tipurilor și excepțiilor;
 - cursoarelor;
 - procedurilor și funcțiilor.
- Pachetele sunt:
 - unități de program compilate;
 - obiecte ale bazei de date care grupează tipuri, obiecte și subprograme *PL/SQL* având o legătură logică între ele.



Ce fel de subprograme integrăm într-un pachet?

Importanța pachetelor

- Atunci când este referențiat un pachet (atunci când este apelată pentru prima dată o construcție a pachetului), întregul pachet este încărcat în SGA (zona globală sistem) și este pregătit pentru execuție.
- Plasarea pachetului în SGA reprezintă avantajul vitezei de execuție, deoarece *server-ul* nu mai trebuie să aducă informația despre pachet de pe disc, aceasta fiind deja în memorie.
 - Apelurile ulterioare ale unor construcții din același pachet, nu solicită operații *I/O* de pe disc.
 - De aceea, ori de câte ori apare cazul unor proceduri și funcții înrudite care trebuie să fie executate împreună, este convenabil ca acestea să fie grupate într-un pachet stocat.
 - În memorie există o singură copie a unui pachet (pentru toți utilizatorii).
- Subprogramelor *overload* pot deveni stocate prin intermediul pachetelor.

Diferența față de subprograme

- Spre deosebire de subprograme, pachetele nu pot:
 - fi apelate;
 - transmite parametri;
 - fi imbricate.

7.1. Definirea pachetelor

- Un pachet conține două părți, fiecare fiind stocată separat în dicționarul datelor:
 - specificația (*package specification*)
 - este partea „vizibilă“ a pachetului, adică interfața cu aplicațiile sau cu alte unități program;
 - poate conține declarații de tipuri, constante, variabile, excepții, cursoare și subprograme (care vor fi vizibile altor aplicații);
 - se declară prima (înaintea corpului pachetului).
 - corpul (*package body*)
 - este partea „ascunsă“ a pachetului, mascată de restul aplicației;
 - conține codul care implementează obiectele definite în specificație (cursoarele și subprogramele) și, de asemenea, obiecte și declarații proprii;
 - poate conține obiecte publice (declarate în specificație) sau private (nu sunt declarate în specificație).

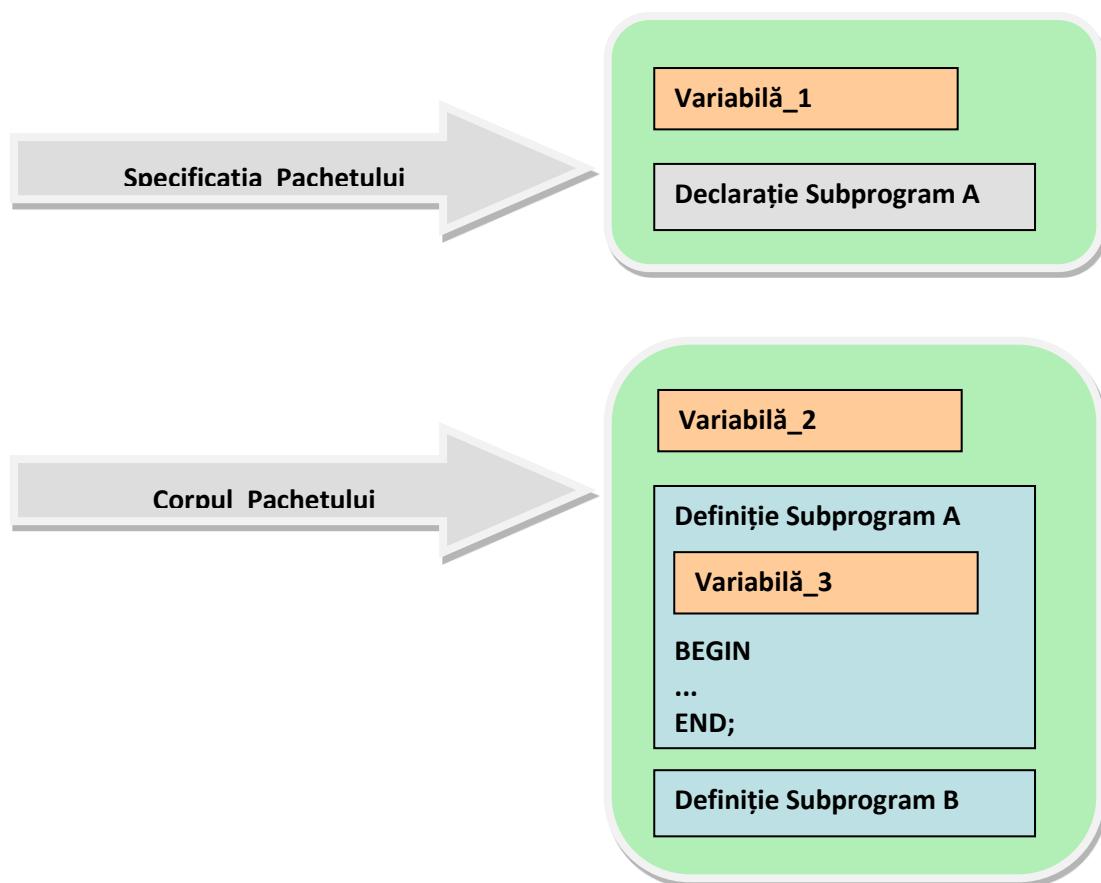


Fig. 7.1. Pachete PL/SQL

- Definirea unui pachet se realizează în două etape:
 - crearea specificației pachetului;
 - crearea corpului pachetului.



❖ Se pot defini pachete care cuprind doar partea de specificație?



❖ Se pot defini pachete care cuprind doar corpul pachetului?

❖ Ce situații impun definirea atât a specificației, cât și a corpului pachetului?



❖ Corpul pachetului poate fi schimbat fără a modifica specificația acestuia?



❖ Dacă specificația este schimbată, aceasta invalidează automat corpul pachetului, deoarece corpul depinde de specificație.



❖ Specificația și corpul pachetului sunt unități compilate separat.

❖ Corpul poate fi compilat doar după ce specificația a fost compilată cu succes.

- Pentru a crea un pachet în schema personală este necesar privilegiul sistem *CREATE PROCEDURE*, iar pentru a crea un pachet în altă schemă este necesar privilegiul sistem *CREATE ANY PROCEDURE*.
- Un pachet definit este disponibil pentru utilizatorul care l-a creat sau orice utilizator căruia i s-a acordat privilegiul *EXECUTE* asupra pachetului respectiv.

- Sintaxa:

```

CREATE PACKAGE nume_pachet
{IS | AS} - specificația pachetului
    -- declarații de tipuri și obiecte publice,
    -- specificații de cursoare și subprograme
END [nume_pachet];
/

CREATE PACKAGE BODY nume_pachet
{IS | AS} -- corpul pachetului
/* declarații de obiecte și tipuri private,
corpurile cursoarelor și subprogramelor precizate
în specificație */

[BEGIN
/* instrucțiuni de initializare, executate o singură
dată, atunci când pachetul este invocat prima
dată în sesiunea utilizatorului */
]

END [nume_pachet];
/

```

- Procesul de creare a specificației și corpului unui pachet urmează același algoritm ca cel întâlnit în crearea subprogramelor *PL/SQL* independente:
 - sunt verificate erorile sintactice și semantice, iar modulul este depus în dicționarul datelor;
 - sunt verificate instrucțiunile *SQL* individuale, adică dacă obiectele referite există și dacă utilizatorul le poate accesa;
 - sunt comparate declarațiile de subprograme din specificația pachetului cu cele din corpul pachetului (dacă au același număr și tip de parametri).
- Orice eroare detectată la compilarea specificației sau a corpului pachetului este marcată în dicționarul datelor.
- După ce specificația și corpul pachetului sunt compilate, acestea devin obiecte în schema curentă.
 - În vizualizarea *USER_OBJECTS* din dicționarul datelor, vor apărea două noi linii:

OBJECT_TYPE	OBJECT_NAME
PACKAGE	nume_pachet
PACKAGE BODY	nume_pachet

7.1.1. Specificația pachetului

- Specificația unui pachet cuprinde declararea procedurilor, funcțiilor, constantelor, variabilelor și excepțiilor care pot fi accesibile aplicațiilor, adică declararea obiectelor de tip *PUBLIC* din pachet.
 - Acestea pot fi utilizate în comenzi sau proceduri care nu aparțin pachetului.
 - Este necesar privilegiul *EXECUTE* asupra pachetului.
 - ◊ Variabilele declarate în specificația unui pachet sunt globale pachetului și sesiunii.
 - ◊ Acestea sunt inițializate (implicit) cu valoarea *NULL*, evident dacă nu este specificată explicit o altă valoare.
- Sintaxa:

```
CREATE [OR REPLACE] PACKAGE [schema.]nume_pachet
  [AUTHID {CURRENT_USER | DEFINER}]
  {IS | AS}
  specificație_PL/SQL;
```

- *Specificație_PL/SQL* poate include declarații de tipuri, variabile, cursoare, excepții, funcții, proceduri etc.
 - În secțiunea declarativă, un obiect trebuie declarat înainte de a fi referit.
- Opțiunea *OR REPLACE* este specificată dacă există deja corpul pachetului.
- Clauzele *IS* și *AS* sunt echivalente, dar dacă se folosește *PROCEDURE BUILDER* este necesară opțiunea *IS*.
- Clauza *AUTHID* specifică faptul ca subprogramele pachetului se execută cu drepturile proprietarului (implicit) sau ale utilizatorului curent.
 - De asemenea, această clauză precizează dacă referințele la obiecte sunt rezolvate în schema proprietarului subprogramului sau a utilizatorului curent.

7.1.2. Corpul pachetului

- Corpul unui pachet conține codul *PL/SQL* pentru obiectele declarate în specificația acestuia și obiectele private pachetului.
- De asemenea, corpul poate include o secțiune declarativă în care sunt specificate definiții locale de tipuri, variabile, constante, proceduri și funcții locale.
 - Obiectele private sunt vizibile numai în interiorul corpului pachetului și pot fi accesate numai de către funcțiile și procedurile din pachetul respectiv.

- Corpul pachetului este opțional și nu este necesar să fie creat dacă specificația pachetului nu conține declarații de proceduri sau funcții.



- ❖ Ordinea în care subprogramele sunt definite în interiorul corpului pachetului este importantă.
- ❖ O variabilă trebuie declarată înainte de a fi referită de altă variabilă sau subprogram, iar un subprogram privat trebuie declarat sau definit înainte de a fi apelat de alte subprograme.

- Sintaxa:

```
CREATE [OR REPLACE] PACKAGE BODY [schema.]nume_pachet
{ IS | AS }
    corp_pachet;
```

7.1.3. Instanțierea pachetului

- Un pachet este instanțiat atunci când este apelat prima dată.
 - Pachetul este citit de pe disc, depus în memorie și este executat codul compilat al subprogramului apelat.
 - În acest moment, memoria este alocată tuturor variabilelor definite în pachet.
- În multe cazuri, atunci când pachetul este instanțiat prima dată într-o sesiune, sunt necesare anumite inițializări.
 - Aceasta se realizează prin adăugarea unei secțiuni de inițializare (optională) în corpul pachetului secțiune încadrată între cuvintele cheie *BEGIN* și *END*.
 - Secțiunea conține un cod de inițializare care este executat atunci când pachetul este invocat pentru prima dată.
- Referința la o declarație sau la un obiect specificat în pachet se face prefixând numele obiectului cu numele pachetului.
 - În corpul pachetului, obiectele din specificație sunt referite fără a specifica numele pachetului.

7.2. Modificarea, stergerea și utilizarea pachetelor

Modificarea pachetului

- Dacă se dorește modificarea sursei pachetului, atunci utilizatorul poate recrea pachetul (cu opțiunea *OR REPLACE*) pentru a-l înlocui pe cel existent.
 - ❖ Schimbarea corpului pachetului nu impune recompilarea construcțiilor dependente.
 - ❖ Schimbările în specificația pachetului necesită recompilarea fiecărui subprogram stocat care referențiază pachetul.
- Folosind comanda *ALTER PACKAGE* se pot recompile explicit specificația pachetului, corpul pachetului sau ambele module.
 - Recompilarea explicită elimină necesitatea recompilării implicită a pachetului la *run-time*.
 - Deoarece într-un pachet toate obiectele sunt stocate ca o unitate, comanda *ALTER PACKAGE* determină recompilarea tuturor obiectelor pachetului.
 - Dacă în timpul recompilării apar erori, atunci este întors un mesaj, iar pachetul devine invalid.
 - Pentru a consulta mesajele erorilor se poate utiliza comanda *SQL*Plus SHOW ERRORS*.



Funcțiile și procedurile dintr-un pachet nu pot fi recompilate individual folosind comenzile *ALTER FUNCTION* sau *ALTER PROCEDURE*.

- Sintaxa:

```
ALTER PACKAGE [schema.]nume_pachet
  COMPILE { PACKAGE | SPECIFICATION | BODY };
```

- Clauza *PACKAGE* determină recompilarea atât a specificației, cât și a corpului pachetului.
 - Este opțiune implicită.
- Clauza *SPECIFICATION* determină recompilarea specificației pachetului.
 - După modificarea specificației unui pachet se poate dori recompilarea acesteia, pentru a verifica dacă apar erori de compilare.

- Atunci când se recompilează specificația pachetului, baza de date invalidează orice obiect local care depinde de acea specificație (ca de exemplu, proceduri care invocă proceduri sau funcții din pachet).
- Corpul pachetului depinde de asemenea de specificație. În lipsa unei recomplări explicate, baza de date va recompile implicit obiectele dependente.
- Clauza *BODY* determină doar recomplarea corpului pachetului.
 - Recompilarea corpului pachetului nu invalidează obiectele care depind de specificație.

Stergerea pachetului

- Sintaxa:

```
DROP PACKAGE [BODY] [schema.]nume_pachet;
```

- Dacă în cadrul comenzi apare opțiunea *BODY*, atunci este șters doar corpul pachetului.
- Dacă se omite opțiunea *BODY*, atunci sunt șterse atât specificația, cât și corpul pachetului.
- Dacă pachetul este șters, toate obiectele dependente de acesta devin invalide. Dacă este șters numai corpul, toate obiectele dependente de acesta rămân valide. În schimb, nu pot fi apelate subprogramele declarate în specificația pachetului, până când nu este recreat corpul pachetului.
- Pentru ca un utilizator să poată șterge un pachet trebuie ca pachetul să aparțină schemei utilizatorului sau utilizatorul să posede privilegiul sistem *DROP ANY PROCEDURE*.

Utilizarea pachetului

- Se realizează în funcție de mediul (*SQL* sau *PL/SQL*) care solicită un obiect din pachetul respectiv.
 - Referirea unui obiect din pachet se realizează prefixând numele acestuia cu numele pachetului.
 - De exemplu, invocarea unei proceduri definite într-un pachet se realizează în:
 - *PLSQL/SQL*, prin comanda


```
nume_pachet.nume_procedură [(listă_argumete)];
```
 - *SQL*Plus*, prin comanda:


```
EXECUTE nume_pachet.nume_procedură [(listă_argumete)]
```

Exemplul 7.1 [- vezi curs](#)

Exemplul 7.2 [- vezi curs](#)

Comenzile **LCD - COMMIT, ROLLBACK, SAVEPOINT**

-  ❖ Un *trigger* nu poate apela un subprogram care conține comenzi *COMMIT*, *ROLLBACK*, *SAVEPOINT*. Prin urmare, pentru flexibilitatea apelului de către *trigger*-i a subprogramelor conținute în pachete, niciun subprogram al pachetului nu trebuie să conțină aceste comenzi.
- ❖ Într-un pachet nu pot fi referite variabile gazdă.
- ❖ Într-un pachet sunt permise declarații *forward*.

Invalidarea modulelor **PL/SQL**

-  ❖ Dacă un subprogram independent apelează un subprogram definit într-un pachet, atunci apar următoarele situații:
 - atunci când corpul pachetului este modificat, dar specificația acestuia nu, subprogramul care referă o construcție a pachetului rămâne valid;
 - dacă specificația pachetului este modificată, atunci subprogramul care referă o construcție a pachetului, precum și corpul pachetului devin invalide.
- ❖ Dacă un subprogram independent referit de un pachet se modifică, atunci întregul corp al pachetului devine invalid, dar specificația pachetului rămâne validă.

Exemplul 7.3 [- vezi curs](#)

Exemplul 7.4 [- vezi curs](#)

Exemplul 7.5 [- vezi curs](#)

Exemplul 7.6 [- vezi curs](#)

-  ❖ Un cursor declarat în specificația unui pachet este un tip de variabilă globală și respectă aceleași reguli privind persistența ca și celelalte variabile.
- ❖ Statusul unui cursor nu este definit de o singură valoare (ca în cazul variabilelor), ci din următoarele atrbute:
 - *%ISOPEN* (dacă este deschis sau închis);
 - *%ROWCOUNT* (dacă este deschis, numărul de linii încărcate);
 - *%FOUND* sau *%NOTFOUND* (dacă ultimul *FETCH* a avut succes).

7.3. Pachete predefinite

- *PL/SQL* conține o serie de pachete predefinite utile în dezvoltarea aplicațiilor.
- Pachetele predefinite adaugă noi funcționalități limbajului, protocoale de comunicație, acces la fișierele sistemului etc.
- Exemple de pachete predefinite:
 - *STANDARD*
 - definește mediul *PL/SQL*
 - conține funcțiile predefinite
 - *DBMS_STANDARD*
 - facilități ale limbajului utile pentru interacțiunea aplicației cu *server-ul Oracle*
 - *DBMS_OUTPUT*
 - permite afișarea de informații
 - *DBMS_DDL*
 - permite accesarea anumitor comenzi *LDD* din *PL/SQL*; în plus, oferă operații speciale de administrare
 - *DBMS_JOB*
 - permite planificarea și gestiunea *job-urilor*
 - *DBMS_SQL*
 - oferă o interfață pentru a putea utiliza *SQL* dinamic
 - *DBMS_PIPE*
 - permite operații de comunicare între două sau mai multe sesiuni conectate la aceeași instanță *Oracle*
 - *DBMS_LOCK*
 - este utilizat pentru a cere, a modifica sau a elibera blocările din baza de date
 - permite folosirea exclusivă sau partajată a unei resurse
 - *DBMS_MVIEW / DBMS_SNAPSHOT*
 - permite exploatarea vizualizărilor materializate
 - *DBMS_UTILITY*
 - oferă utilități *DBA*, permite analiza obiectelor unei scheme, verifică dacă *server-ul* lucrează în mod paralel etc.
 - *DBMS_LOB*
 - oferă mecanisme de acces și prelucrare a datelor de tip *LOB*
 - permite compararea datelor de tip *LOB*, adăugarea de date la un *LOB*, copierea datelor dintr-un *LOB* în altul, ștergerea unor porțiuni din date *LOB*, deschiderea, închiderea și regăsirea de informații din date *BFILE* etc.

- *UTL_FILE*
 - permite citirea/scrierea din/în fișierele text ale sistemului de operare
- *UTL_MAIL*
 - permite crearea și trimitera unui *e-mail*
- *UTL_HTTP*
 - permite utilizarea protocolului *HTTP* pentru a accesa date de pe *Internet*
- *UTL_TCP*
 - permite aplicațiilor *PL/SQL* să comunice cu *server-e* externe utilizând protocolul *TCP/IP*

7.3.1. Pachetul *STANDARD*

- Este un pachet predefinit fundamental în care se declară tipurile, excepțiile, subprogramele care sunt utilizabile automat în programele *PL/SQL*.
- Conține funcțiile predefinite (de exemplu, *UPPER*, *ABS*, *TO_CHAR* etc.).
- Conținutul acestui pachet este vizibil tuturor aplicațiilor.
- Pentru referirea componentelor acestui pachet nu trebuie utilizată prefixarea cu numele pachetului.

Exemplul 7.7 - vezi explicații curs

```
SELECT STANDARD.ABS (-1), ABS (-1)
FROM   DUAL;
```



- ❖ Se pot defini funcții cu același nume ca și cele predefinite?
- ❖ Funcția din exemplul 7.8 se poate defini?

Exemplul 7.8 - vezi explicații curs

```
CREATE OR REPLACE FUNCTION ABS (x NUMBER)
RETURN VARCHAR2
IS
BEGIN
  IF X<0 THEN RETURN 'Rezultatul intors este: '||-1*X;
  ELSE    RETURN 'Rezultatul intors este: '||X;
  END IF;
END;
/
SELECT ABS (-1)
FROM   DUAL;
```

7.3.2. Pachetul ***DBMS_OUTPUT***

- Trimit mesaje text dintr-un bloc *PL/SQL* într-o zonă privată de memorie, din care mesajele pot fi afișate pe ecran.
- Pachetul este utilizat preponderent:
 - în timpul testării și depanării programelor;
 - pentru a afișa mesaje și rapoarte în *SQL*DBA* sau *SQL*Plus*;
 - pentru a urmări pașii de execuție a unui program.
- Subprograme definite în pachet:
 - *PUT* – depune text în *buffer* (pe o singură linie);
 - *NEW_LINE* – trimit conținutul *buffer*-ului pe ecran (adaugă în *buffer* un sfârșit de linie);
 - *PUT_LINE* – depune text în *buffer* (*PUT*) și trimit conținutul *buffer*-ului pe ecran (*NEW_LINE*);
 - *GET_LINE* – citește din *buffer* o singură linie;
 - *GET_LINES* – citește din *buffer* mai multe linii (le depune într-o colecție);
 - *DISABLE* – dezactivează apelurile procedurilor *PUT*, *NEW_LINE*, *PUT_LINE*, *GET_LINE*, *GET_LINES* și elimină informația depusă în *buffer*;
 - *ENABLE* – activează apelurile procedurilor *PUT*, *NEW_LINE*, *PUT_LINE*, *GET_LINE*, *GET_LINES* și permite specificarea dimensiunii *buffer*-ului.

Exemplul 7.9 **- vezi curs**

Exemplul 7.10 **- vezi curs**

Exemplul 7.11 **- vezi curs**

7.3.3. Pachetul ***DBMS_JOB***

- Pachetul *DBMS_JOB* este utilizat pentru planificarea programelor *PL/SQL* în vederea execuției.
- Cu ajutorul acestui pachet:
 - se pot executa programe *PL/SQL* la momente determinate de timp;
 - se pot șterge sau suspenda programe din lista de planificări în vederea execuției;
 - se pot rula programe de întreținere a sistemului în perioadele de timp în care acesta este mai puțin solicitat (de exemplu, noaptea) etc.

- Subprograme definite în pachet:
 - *SUBMIT* – adaugă un nou *job* în coada de aşteptare;
 - *REMOVE* – șterge un *job* specificat din coada de aşteptare;
 - *RUN* – execută imediat un *job* specificat;
 - *BROKEN* – dezactivează execuția unui *job* și îl setează *broken* (implicit, orice *job* este *not broken*, iar un *job* marcat *broken* nu se execută);
 - *WHAT* – descrie un *job* specificat;
 - *NEXT_DATE* – specifică momentul următoarei execuții a unui *job*;
 - *INTERVAL* – specifică intervalul de timp scurs dintre două execuții consecutive ale unui *job*;
 - *CHANGE* – modifică argumentele *WHAT*, *NEXT_DATE*, *INTERVAL*.
- Fiecare dintre subprogramele pachetului are argumente specifice.
 - Procedura *SUBMIT* are ca argumente:
 - *JOB* – de tip *OUT*, un identificator pentru *job* (*BINARY_INTEGER*);
 - *WHAT* – de tip *IN*, codul *PL/SQL* care va fi executat ca un *job* (*VARCHAR2*);
 - *NEXT_DATE* – de tip *IN*, data următoarei execuții a *job*-ului (implicit este *SYSDATE*);
 - *INTERVAL* – de tip *IN*, funcție care furnizează intervalul dintre execuțiile *job*-ului (*VARCHAR2*, implicit este *null*);
 - *NO_PARSE* – de tip *IN*, variabilă logică care indică dacă *job*-ul trebuie analizat gramatical (*BOOLEAN*, implicit este *FALSE*).

Exemplul 7.12 – vezi curs

- Vizualizarea *DBA_JOBS* din dicționarul datelor furnizează informații referitoare la starea tuturor *job*-urilor din coada de aşteptare.
- Vizualizarea *DBA_JOBS_RUNNING* conține informații despre *job*-urile care sunt în curs de execuție.

Exemplul 7.13

```
SELECT JOB, LOG_USER, NEXT_DATE, BROKEN, WHAT
FROM DBA_JOBS;
```

```
SELECT *
FROM DBA_JOBS_RUNNING;
```

7.3.4. Pachetul *UTL_FILE*

- Pachetul *UTL_FILE* permite programului *PL/SQL* scrierea în fișiere text definite la nivelul sistemului de operare, respectiv citirea din aceste fișiere.
- Tipuri de date definite pachetul *UTL_FILE*
 - o *FILE_TYPE*
 - Specificația tipului:

```
TYPE file_type IS RECORD (
    id      BINARY_INTEGER,
    datatype BINARY_INTEGER);
```
- Subprograme definite în pachetul *UTL_FILE*
 - o funcții
 - *FOPEN*
 - Deschide un fișier și întoarce un *handler* care va fi utilizat în următoarele operații *I/O*.
 - Specificația funcției:

```
UTL_FILE.FOPEN (
    location      IN VARCHAR2,
    filename      IN VARCHAR2,
    open_mode     IN VARCHAR2,
    max_linesize IN BINARY_INTEGER DEFAULT 1024)
RETURN file_type;
```

 - Parametrul *open_mode* este un string care specifică pentru ce operații a fost deschis fișierul: r (*read text*), w (*write text*), a (*append text*), rb (*read byte mode*), wb (*write byte mode*) sau ab (*append byte mode*).
 - *IS_OPEN*
 - Întoarce valoarea *TRUE* dacă fișierul este deschis, altfel întoarce *FALSE*.
 - Specificația funcției:

```
UTL_FILE.IS_OPEN(file IN FILE_TYPE)
RETURN BOOLEAN;
```
 - o proceduri
 - *GET_LINE*
 - Citește o linie din fișierul deschis pentru citire și o plasează într-un *buffer* de tip șir de caractere.
 - *PUT și PUT_LINE*
 - Permit scrierea textului din *buffer* în fișierul deschis pentru scriere sau adăugare.

- *PUTF*
 - Este asemănătoare funcției *printf()*.
 - Este o procedură *PUT* cu format.
- *NEW_LINE*
 - Scrie în fișier un caracter sfârșit de linie specific fișierelor sistemului de operare.
- *FCLOSE*
 - Închide un fișier
- *FCLOSEALL*
 - Închide toate *handler*-urile fișierului deschis.
- Utilizarea componentelor acestui pachet pentru procesarea fișierelor sistemului de operare poate declanșa excepții, printre care:
 - *INVALID_PATH* – numele sau locația fișierului sunt invalide;
 - *INVALID_MODE* – parametrul *OPEN_MODE* (prin care se specifică dacă fișierul este deschis pentru citire, scriere, adăugare) este invalid;
 - *INVALID_FILEHANDLE* – *handler*-ul de fișier obținut în urma deschiderii este invalid;
 - *INVALID_OPERATION* – operație invalidă asupra fișierului;
 - *READ_ERROR* – o eroare a sistemului de operare a apărut în timpul operației de citire;
 - *WRITE_ERROR* – o eroare a sistemului de operare a apărut în timpul operației de scriere;
 - *INTERNAL_ERROR* – o eroare nespecificată a apărut în *PL/SQL*.

Exemplul 7.14 - vezi curs

7.3.5. Pachetul **DBMS_SQL**

- Permite utilizarea dinamică a comenziilor *SQL* în proceduri stocate sau în blocuri anonime.
 - Comenziile dinamice nu sunt încorporate în programul sursă, ci sunt depuse în siruri de caractere.
- O comandă *SQL* dinamică este o instrucțiune *SQL* care conține variabile ce se pot schimba în timpul execuției.
 - De exemplu, pot fi utilizate instrucțiuni *SQL* dinamice pentru:
 - a crea o procedură care operează asupra unui tabel al cărui nume nu este cunoscut decât în momentul execuției;

- a scrie și executa o comandă *LDD*;
- a scrie și executa o comandă *GRANT*, *ALTER SESSION* etc.



- ❖ În *PL/SQL* comenzi date ca exemplu anterior nu pot fi executate static.
 - ❖ Pachetul *DBMS_SQL* permite, de exemplu, ca într-o procedură stocată să fie utilizată o comandă *DROP TABLE*.
 - ❖ Utilizarea pachetului *DBMS_SQL* pentru a executa comenzi *LDD* poate genera interblocări. De exemplu, pachetul este utilizat pentru a șterge o procedură care însă este utilizată.
 - ❖ *SQL* dinamic suportă toate tipurile de date *SQL*, dar nu suportă tipurile de date specifice *PL/SQL*.
- Orice comandă *SQL* trebuie să treacă prin anumite etape (unele putând fi evitate):
 - analizarea sintactică;
 - validarea;
 - asigurarea că toate referințele la obiecte sunt corecte;
 - asigurarea că există privilegiile referitoare la acele obiecte (*parse*);
 - obținerea de valori pentru variabilele de legătură din comanda (*binding variables*);
 - executarea comenzi (*execute*);
 - selectarea linilor rezultatului;
 - încărcarea liniilor rezultatului (*fetch*).
 - Dintre subprogramele pachetului *DBMS_SQL*, care permit implementarea etapelor amintite anterior, se remarcă:
 - *OPEN_CURSOR* (deschide un nou cursor, adică se stabilește o zonă de memorie în care este procesată comanda *SQL*);
 - *PARSE* (stabilește validitatea comenzi *SQL*, adică se verifică sintaxa instrucțiunii și se asociază cursorului deschis);
 - *BIND_VARIABLE* (leagă valoarea dată de variabila corespunzătoare din comanda *SQL* analizată)
 - *EXECUTE* (execută comanda *SQL* și întoarce numărul de linii procesate);
 - *FETCH_ROWS* (regăsește o linie pentru un cursor specificat, iar pentru mai multe linii folosește un *LOOP*);
 - *CLOSE_CURSOR* (închide cursorul specificat).

Exemplul 7.15 [- vezi curs](#)

Exemplul 7.16 [- vezi curs](#)

SQL Dinamic nativ

- Comanda de bază utilizată pentru procesarea dinamică nativă a comenziilor *SQL* și a blocurilor *PL/SQL* este *EXECUTE IMMEDIATE*, care are următoarea sintaxă:

```
EXECUTE IMMEDIATE sir_dinamic
[[BULK COLLECT] INTO {def_variabila [, def_variabila ...] |  

record} ]
[USING [IN | OUT | IN OUT] argument_bind
[, [IN | OUT | IN OUT] argument_bind ...] ]
[ {RETURNING | RETURN}
[BULK COLLECT] INTO argument_bind [, argument_bind ...] ];
```

- *sir_dinamic* este un sir de caractere care reprezintă o comandă *SQL* (fără caracter de terminare “;”) sau un bloc *PL/SQL* (fără caracter de terminare “/”).
- *def_variabila* reprezintă variabila în care se stochează valoarea coloanei selectate.
- *record* reprezintă înregistrarea în care se depune o linie selectată.
- *argument_bind*, dacă se referă la valori de intrare (*IN*) este o expresie (comandă *SQL* sau bloc *PL/SQL*), iar dacă se referă la valori de ieșire (*OUT*) este o variabilă ce va conține valoarea selectată de comanda *SQL* sau de blocul *PL/SQL*.
- Clauza *INTO* este folosită pentru cereri care întorc o singură linie, iar clauza *USING* pentru a reține argumentele de legătură.
- Pentru procesarea unei cereri care întoarce mai multe linii sunt necesare instrucțiunile *OPEN...FOR*, *FETCH* și *CLOSE*.
- Prin clauza *RETURNING* sunt precizate variabilele care conțin rezultatele.

Exemplul 7.17 [- vezi curs](#)

Exemplul 7.18 [- vezi curs](#)

Exemplul 7.19 [- vezi curs](#)

Exemplul 7.20 [- vezi curs](#)

SQL Dinamic nativ versus pachetul DBMS_SQL

- Pentru execuția dinamică a comenziilor *SQL* în *PL/SQL* există două tehnici:
 - utilizarea pachetului *DBMS_SQL*;
 - *SQL* dinamic nativ.
- Dacă s-ar face o comparație între *SQL* dinamic nativ și funcționalitatea pachetului *DBMS_SQL*, se poate sublinia că *SQL* dinamic nativ:
 - este mai ușor de utilizat;
 - solicită mai puțin cod;
 - este mai rapid;
 - poate încărca liniile direct în înregistrări *PL/SQL*;
 - suportă toate tipurile acceptate de *SQL* static în *PL/SQL*, inclusiv tipuri definite de utilizator.
- Față de *SQL* dinamic nativ pachetul *DBMS_SQL*:
 - suportă comenzi *SQL* mai mari de 32 KB;
 - permite încărcarea înregistrărilor (procedura *FETCH_ROWS*);
 - acceptă comenzi cu clauza *RETURNING* pentru reactualizarea și ștergerea de linii multiple;
 - suportă posibilitățile oferite de comanda *DESCRIBE* (procedura *DESCRIBE_COLUMNS*);
 - analizează validitatea unei comenzi *SQL* o singură dată (procedura *PARSE*), permitând ulterior mai multe utilizări ale comenzi pentru diferite mulțimi de argumente.

7.3.6. Pachetul DBMS_DDL

- Permite accesul la anumite comenzi *LDD* care pot fi folosite în subprograme *PL/SQL* stocate.
 - De exemplu, prin intermediul acestui pachet pot utilizați în *PL/SQL* comenziile *ALTER* sau *ANALYZE*.

Procedura ALTER_COMPILE

- Permite recompilarea programului modificat (procedură, funcție, declanșator, pachet, corp pachet).

- Sintaxa

```
ALTER_COMPILE (tip_object, nume_schema, nume_object);
```

- Instrucțiune echivalentă *SQL*

```
ALTER PROCEDURE | FUNCTION | PACKAGE [nume_schema.]nume  
COMPILE [ PACKAGE | SPECIFICATION | BODY ];
```

Vezi Curs SGBD6 PL/SQL - Exemplul 6.19 (recompilare subprograme invalide)

Procedura *ANALYZE_OBJECT*

- Permite colectarea statisticilor pentru obiecte de tip *table*, *cluster* sau *index* care vor fi utilizate pentru optimizarea planului de execuție a comenziilor *SQL* care accesează obiectele analizate.
 - De exemplu, despre un tabel se pot obține următoarele informații: numărul de linii, numărul de blocuri, lungimea medie a unei linii, numărul de valori distincte ale unei coloane, numărul elementelor *null* dintr-o coloană, distribuția datelor (histograma) etc.

- Sintaxa

```
ANALYZE_OBJECT (tip_object, nume_schema, nume_object,  
metoda, număr_linii_estimate, procent, opțiune_metoda,  
nume_partiție);
```

- *Metoda* poate fi *COMPUTE*, *ESTIMATE* sau *DELETE*.
 - *DELETE* determină ștergerea statisticilor din dicționarul datelor referitoare la obiectul analizat.
 - *COMPUTE* calculează statisticile referitoare la un obiect analizat și le depune în dicționarul datelor.
 - *ESTIMATE* estimatează statistici.
- Dacă *nume_schema* este *null*, atunci se presupune că este vorba de schema curentă.
- Dacă *tip_object* este diferit de *table*, *index* sau *cluster*, se declanșează o eroare.
- Parametrul *procent* reprezintă procentajul liniilor de estimat și este ignorat dacă este specificat numărul liniilor de estimat (*număr_linii_estimate*). Implicit, ultimele patru argumente ale procedurii au valoarea *null*.
- Argumentul *opțiune_metoda* poate avea forma:

```
[FOR TABLE] [FOR ALL INDEXES]  
[FOR ALL [INDEXED] COLUMNS] [SIZE n]
```

- Pentru metoda *ESTIMATE* trebuie să fie prezentă una dintre aceste opțiuni.
- Instrucțiune echivalentă *SQL*

```
ANALYZE TABLE | CLUSTER | INDEX  
[nume_schema.]nume_object [metoda]  
    STATISTICS [SAMPLE n] [ROWS | PERCENT]
```

Exemplul 7.21 - **vezi curs**

Bibliografie

1. *Programare avansată în Oracle9i*, I. Popescu, A. Alecu, L. Velcescu, G. Florea (Mihai), Ed. Tehnică (2004)
2. *Oracle Database PL/SQL Language Reference 11g Release 2*, Oracle Online Documentation (2012)
3. *Oracle Database SQL Language Reference 11g Release 2*, Oracle Online Documentation (2012)
4. *Oracle Database 11g: PL/SQL Fundamentals, Student Guide*, Oracle University (2009)

CUPRINS

8. PL/SQL – Trigger-i	2
8.1. Trigger-i LMD	5
8.1.1. Trigger-i LMD la nivel de comandă	7
8.1.2. Trigger-i LMD la nivel de linie	7
8.1.3. Ordinea de execuție a trigger-ilor LMD	8
8.1.4. Predicate condiționale	11
8.2. Trigger-i INSTEAD OF	12
8.3. Trigger-i sistem	13
8.3.1. Trigger-i pentru comenzi LDD	14
8.3.2. Trigger-i pentru evenimente sistem	14
8.4. Modificarea și stergerea trigger-ilor	15
8.5. Informații despre trigger-i	16
8.6. Privilegii sistem	17
8.7. Tabele <i>mutating</i>	18
Bibliografie	19

8. PL/SQL – Trigger-i

- Un *trigger* (declanșator) este un bloc *PL/SQL* cu nume, stocat în baza de date (independent), care se execută automat ori de câte ori are loc evenimentul „declanșator“ de care este asociat.
- Evenimentul declanșator poate consta din:
 - modificarea unui tabel sau a unei vizualizări;
 - acțiuni sistem;
 - acțiuni utilizator.
- Un *trigger* poate fi asociat cu un eveniment care are loc asupra unui tabel, unei vizualizări, unei scheme sau unei baze de date.



❖ Față de subprogramele stocate *trigger-ii*:

- pot fi activați sau dezactivați;
- nu pot fi invocați explicit;
- nu pot conține comenzi *LCD COMMIT, SAVEPOINT* sau *ROLLBACK*.

❖ Un *trigger* activ (*enable*) va fi invocat automat de către sistem ori de câte ori are loc evenimentul asociat acestuia.

❖ Un *trigger* dezactivat (*disable*) nu va fi declanșat, chiar dacă evenimentul asociat are loc.



În mod asemănător pachetelor, *trigger-i* nu pot fi definiți local în blocuri *PL/SQL* sau pachete.



❖ Ca și în cazul subprogramelor independente sau al pachetelor, atunci când un *trigger* este depus în dicționarul datelor alături de codul sursă este depusă și și forma compilată (*p-codul*).

❖ Dacă *trigger-ul* este valid, atunci va fi apelat fără recomplilare.

❖ *Trigger-ii* pot fi invalidați în aceeași manieră ca pachetele și subprogramele.

Dacă *trigger-ul* este invalidat, acesta va fi recompliat la următoarea activare.



Când utilizăm *trigger-i*?

Atunci când dorim ca efectuarea unei anumite operații să implice întotdeauna execuția unor acțiuni asociate.



- ❖ Nu trebuie definiți *trigger-i* care duplică sau înlocuiesc acțiuni ce pot fi implementate mai simplu. De exemplu, nu are sens să fie definiți *trigger-i* care să implementeze regulile de integritate ce pot fi definite prin constrângeri declarative.
- ❖ Utilizarea excesivă a *trigger-ilor* poate determina interdependențe complexe ce pot fi dificil de menținut.
- ❖ Atunci când sunt definiți *trigger-i* trebuie să se țină cont de recursivitate și de efectele în cascadă.
- *Trigger-ii* pot fi definiți la nivel de:
 - aplicație (*application triggers*);
 - bază de date (*database triggers*).

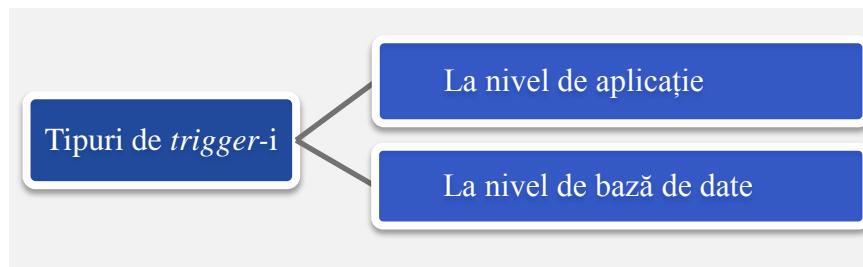


Fig. 8.1. Tipuri de *trigger-i*

Trigger-i aplicație

- Se pot executa automat ori de câte ori apare un anumit eveniment într-o aplicație (de exemplu, o aplicație dezvoltată cu *Developer Suite*).
 - *Form Builder* (utilitar *Developer Suite*) utilizează frecvent acest tip de *trigger-i* (*form builder triggers*). Aceștia pot fi declanșați prin apăsarea unui buton, prin navigarea pe un numit câmp etc.

Trigger-i bază de date

- Se pot executa automat ori de câte ori are loc:
 - o comandă *LMD* asupra datelor unui tabel;
 - o comandă *LMD* asupra datelor unei vizualizări;
 - o comandă *LDD* (*CREATE*, *ALTER*, *DROP*) referitoare la anumite obiecte ale schemei sau ale bazei de date;
 - un eveniment sistem (*SHUTDOWN*, *STARTUP*);

- o acțiune a utilizatorului (*LOGON*, *LOGOFF*);
- o eroare (*SERVERERROR*, *SUSPEND*).

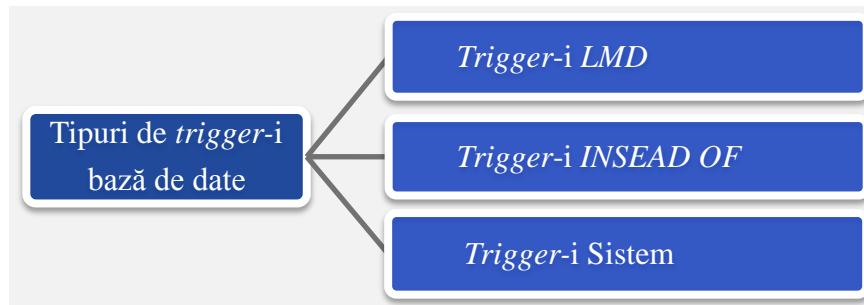


Fig. 8.2. Tipuri de trigger-i bază de date

- *Trigger-ii* bază de date sunt de trei tipuri:
 - *trigger-i LMD*
 - sunt activați de comenzi *LMD* (*INSERT*, *UPDATE* sau *DELETE*) executate asupra unui tabel al bazei de date
 - *trigger-i INSTEAD OF*
 - sunt activați de comenzi *LMD* executate asupra unei vizualizări (relaționale sau obiect)
 - *trigger-i* sistem
 - sunt activați de un eveniment sistem (oprirea sau pornirea bazei), de comenzi *LDD* (*CREATE*, *ALTER*, *DROP*), de conectarea/deconectarea unui utilizator
 - sunt definiți la nivel de schemă sau la nivel de bază de date
- !**
- ❖ *Trigger-ii* asociați unui tabel vor acționa indiferent de aplicația care a efectuat operația *LMD*.
 - ❖ Dacă operația *LMD* se referă la o vizualizare, *trigger-ul INSTEAD OF* definește acțiunile care vor avea loc, iar dacă aceste acțiuni includ comenzi *LMD* referitoare la tabele, atunci *trigger-ii* asociați acestor tabele sunt și ei, la rândul lor, activați.
 - ❖ *Trigger-ii* asociați unei baze de date se declanșează pentru fiecare eveniment, pentru toți utilizatorii.
 - ❖ *Trigger-ii* asociați unei scheme sau unui tabel se declanșează numai dacă evenimentul declanșator implică acea schemă sau acel tabel.



În acest capitol se face referire doar la *trigger-ii* bază de date.

8.1. Trigger-i LMD

- Sunt activați de comenzi *LMD* (*INSERT*, *UPDATE* sau *DELETE*) executate asupra unui tabel al bazei de date.
- În comanda de creare a unui *trigger* pot fi precizate mai multe comenzi declanșatoare diferite, dar care se execută asupra unui singur tabel.
- Sintaxa:

```

CREATE [OR REPLACE] TRIGGER [schema.]nume_trigger
--momentul când este declanșat
{ BEFORE | AFTER }
--comanda/comenzile care îl declanșează
{ DELETE|INSERT|UPDATE [OF coloana[, coloana ...] ] }
[OR {DELETE|INSERT|UPDATE [OF coloana[, coloana ...] ] ...}]
ON [schema.]nume_tabel

[REFERENCING {OLD [AS] vechi NEW [AS] nou
              | NEW [AS] nou OLD [AS] vechi } ]
[FOR EACH ROW]
[WHEN (condiție) ]

        corp_trigger (Bloc anonim PL/SQL sau comanda CALL);
    
```

- Numele *trigger*-ului:
 - trebuie să fie unic printre numele *trigger*-ilor din cadrul aceleiași scheme;
 - poate să coincidă cu numele altor obiecte ale schemei în care este definit (de exemplu, tabele, vizualizări sau proceduri).
- Momentul declanșării *trigger*-ului:
 - rezintă momentul în care va fi executat corpul *trigger*-ului;
 - poate fi înainte (*BEFORE*) sau după (*AFTER*) comanda declanșatoare;
- Comanda declanșatoare:
 - poate fi specificată o singură comandă *LMD* (*INSERT*, *DELETE* sau *UPDATE*) sau o combinație disjunctivă celor trei comenzi *LMD* (folosind operatorul logic *OR*);
 - poate fi urmată de o enumerare a coloanele a căror actualizare va declanșa *trigger*-ul (comanda declanșatoare este un *UPDATE*).

- Tabelul asupra căruia este executată comanda declanșatoare poate fi:
 - un tabel (*table*);
 - un tablou imbricat (*nested table*).
- Valorile coloanelor înainte și după modificarea unei linii:
 - Înainte de modificare valoarea unei coloane este referită prin atributul *OLD*, iar după modificare este referită prin atributul *NEW*.
 - În interiorul blocului *PL/SQL*, coloanele prefixate prin *OLD* sau *NEW* sunt considerate variabile externe, deci trebuie prefixate cu ":".
- Clauza *REFERENCING* permite redenumirea atributelor *NEW* și *OLD*.
- Clauza *FOR EACH ROW* declară un *trigger* la nivel de linie.
 - Lipsa acestei clauze determină definirea unui *trigger* la nivel de instrucție.
- Clauza *WHEN* precizează o expresie *booleană* care este verificată pentru fiecare linie corespunzătoare din tabel.
 - Este valabilă doar pentru *trigger*-ii la nivel de linie.
- Corpul *trigger*-ului
 - nu poate depăși 32KB;
 - pentru a evita depășirea dimensiunii maxime se pot defini proceduri stocate ce pot fi invocate din corpul *trigger*-ului;
 - poate consta dintr-un bloc *PL/SQL*;
 - poate consta dintr-o singură comandă *CALL*;
 - comanda *CALL* poate apela un subprogram *PL/SQL* stocat, o rutină *C* sau o metodă *Java*;
 - în acest caz, comanda *CALL* nu poate conține clauza *INTO* care este specifică funcțiilor;
 - pentru a referi coloanele tabelului asociat *trigger*-ului, acestea trebuie prefixate de atributele *:NEW* sau *:OLD*;
 - în expresia parametrilor nu pot să apară variabile *bind*.



- ❖ Un *trigger* poate activa alt *trigger*, iar acesta la rândul său poate activa alt *trigger* etc (*trigger*-i în cascadă).
- ❖ Sistemul permite maximum 32 de *trigger*-i în cascadă.
- ❖ Numărul acestora poate fi limitat (utilizând parametrul de inițializare *OPEN_CURSORS*), deoarece pentru fiecare execuție a unui *trigger* trebuie deschis un nou cursor.

- Trigger-ii LMD pot fi:
 - la nivel de comandă (*statement level trigger*);
 - la nivel de linie (*row level trigger*).

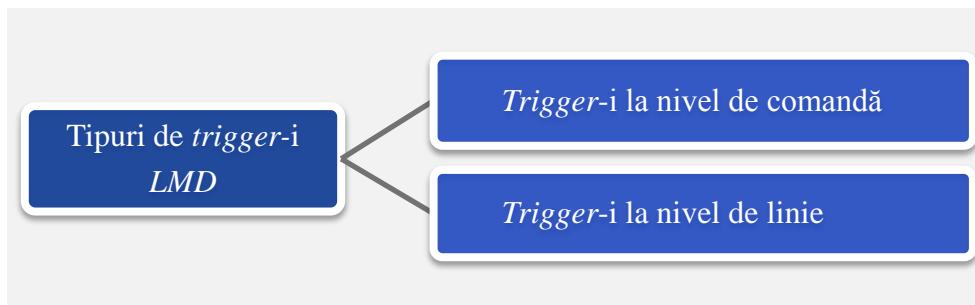


Fig. 8.3. Tipuri de trigger-i LMD

8.1.1. Trigger-i LMD la nivel de comandă

- Sunt execuți o singură dată pentru comanda declanșatoare, indiferent de numărul de linii afectate (chiar dacă nicio linie nu este afectată).
- Sunt utilizați atunci când acțiunea *trigger*-ului nu depinde de informațiile menținute în liniile afectate de comandă.

Exemplul 8.1 – **vezi curs**

Exemplul 8.2 – **vezi curs**

8.1.2. Trigger-i LMD la nivel de linie

- Sunt creați atunci când în comanda *CREATE TRIGGER* este precizată clauza *FOR EACH ROW*.
 - Lipsa acestei clauze determină definirea unui *trigger* la nivel de instrucțiune.
- Sunt execuți pentru fiecare linie afectată de instrucțiunea declanșatoare.
 - Dacă instrucțiunea declanșatoare nu afectează nicio linie, atunci nu sunt execuți.
- Restricțiile acestor tipuri de trigger-i pot fi specificate în clauza *WHEN* (printr-o expresie *booleană*).
 - Expressia *booleană* este evaluată de *trigger* pentru fiecare linie afectată de comanda declanșatoare.
 - *Trigger*-ul este executat pentru o anumită linie, doar dacă expresia este adevărată pentru acea linie.
 - În expresia *booleană* nu sunt permise funcții definite de utilizator sau subcereri *SQL*.



- ❖ Trigger-ii la nivel linie nu sunt performanți dacă se fac frecvent reactualizări pe tabele de dimensiuni foarte mari.

Exemplul 8.3 – **vezi curs**

- Accesul la vechile, respectiv noile valori ale coloanelor liniei curente, afectată de evenimentul ce a declanșat trigger-ul, se realizează prin:
 - OLD.numă_colonă
 - NEW.numă_colonă
- !
 - ❖ În interiorul blocului *PL/SQL*, coloanele tabelului prefixate cu *OLD* sau *NEW* sunt considerate variabile externe și deci, trebuie precedate de caracterul „:“.
 - ❖ În expresia booleană din clauza *WHEN* nu trebuie utilizată prefixarea cu „:“ pentru *OLD* sau *NEW*.
- În cazul celor trei comenzi *LMD*, aceste valori devin:

COMANDA	NEW.numă_colonă	OLD.numă_colonă
INSERT	nouă valoare	null
DELETE	null	vechea valoare
UPDATE	nouă valoare	vechea valoare

Exemplul 8.4 – **vezi curs**

8.1.3. Ordinea de execuție a trigger-ilor LMD

- *PL/SQL* permite definirea a 12 tipuri de trigger-i *LMD* care sunt obținuți prin combinarea proprietăților ce pot fi specificate în comanda de definire a acestora:
 - momentul declanșării (*BEFORE* sau *AFTER*);
 - nivelul la care acționează (nivel comandă sau nivel linie – *FOR EACH ROW*);
 - comanda declanșatoare (*INSERT*, *UPDATE* sau *DELETE*).
- !
 - ❖ De exemplu, un trigger *BEFORE INSERT* acționează o singură dată, înaintea execuției unei instrucțiuni *INSERT*, iar un trigger *BEFORE INSERT FOR EACH ROW* acționează înainte de inserarea fiecărei noi înregistrări.

- O comandă declanșatoare sau o comandă din corpul unui *trigger* poate determina verificarea mai multor constrângeri de integritate. De asemenea, *trigger*-ii pot conține comenzi care pot determina declanșarea altor *trigger*-i (*trigger*-i în cascadă).
- Pentru a menține o secvență adecvată de declanșare a *trigger*-ilor și de verificare a constrângерilor de integritate, sistemul *Oracle* utilizează următorul model de execuție a *trigger*-lor *LMD* multipli:
 1. Se execută toți *trigger*-ii *BEFORE* comandă care sunt declanșați de comanda *LMD*.
 2. Pentru fiecare linie afectată de comanda *LMD*:
 - 2.1. se execută toți *trigger*-ii *BEFORE* linie care sunt declanșați de comanda *LMD*;
 - 2.2. se blochează și se modifică linia afectată de comanda *LMD* (se rulează comanda *LMD*); se verifică constrângările de integritate (blocarea rămâne valabilă până în momentul în care tranzacția este permanentizată);
 - 2.3. se execută toți *trigger*-ii *AFTER* linie care sunt declanșați de comanda *LMD*.
 3. Se execută toți *trigger*-ii *AFTER* comandă care sunt declanșați de comanda *LMD*.
- Definiția modelului de execuție este recursivă.
 - De exemplu, o comandă *SQL* poate determina execuția unui *trigger BEFORE* linie și verificarea unei constrângeri de integritate. Acel *trigger BEFORE* linie poate realiza o actualizare (*UPDATE*) care la rândul său determină verificarea unei constrângeri de integritate și execuția unui *trigger AFTER* comandă. *Trigger*-ul *AFTER* comandă determină la rândul său o verificare a unei constrângeri de integritate.
 - În acest caz, modelul de execuție rulează pașii recursiv, după cum urmează:
Este lansată comanda *SQL* declanșatoare.
 1. Se execută *trigger*-ii *BEFORE* linie.
 - 1.1. Se execută *trigger*-ii *AFTER* comandă declanșați de comanda *UPDATE* din corpul *trigger*-ului *BEFORE* linie.
 - 1.1.1. Se execută comenzi din corpul *trigger*-ilor *AFTER* comandă.
 - 1.1.2. Se verifică dacă sunt îndeplinite constrângările de integritate definite pentru tabelele modificate de *trigger*-ii *AFTER* comandă.
 - 1.2. Se execută comenzi din corpul *trigger*-ilor *BEFORE* linie.
 - 1.3. Se verifică dacă sunt îndeplinite constrângările de integritate definite pentru tabelele modificate de *trigger*-ii *BEFORE* linie.

- 2. Se execută comanda *SQL*.
 - 3. Se verifică dacă sunt îndeplinite constrângerile de integritate ce ar putea fi încălcate de comanda *SQL*.
 - Există două excepții privind recursivitatea.
 - Atunci când o comandă modifică un tabel (cheia primară sau cheia externă) care face parte dintr-o constrângere referențială și declanșează un *trigger* ce modifică celălalt tabel referit în constrângere, doar comanda declanșatoare va determina verificarea constrângerii de integritate. Astfel, se permite *trigger-ilor* la nivel de linie să forțeze integritatea referențială.
 - *Trigger-ii* la nivel de comandă declanșați datorită opțiunilor *DELETE CASCADE* și *DELETE SET NULL* sunt execuții înainte și după lansarea comenzi *DELETE* de către utilizator. În acest mod se previne apariția erorilor *mutating*.
-  ❖ O proprietate importantă a modelului de execuție a *trigger-ilor* este acela că toate acțiunile și verificările realizate datorită execuției unei comenzi declanșatoare trebuie să se termine cu succes.
-  ❖ Dacă apare o excepție în interiorul unui *trigger* și aceasta nu este explicit tratată, atunci toate acțiunile realizate ca rezultat al comenzi declanșatoare, incluzând toate acțiunile realizate de *trigger-ii* declanșați de comandă vor fi anulate (*rollback*).
-  ❖ În acest mod, *trigger-i* nu pot compromite constrângerile de integritate.
-  ❖ *Trigger-ii* de tipuri diferite sunt execuții într-o ordine specifică.
-  ❖ *Trigger-ii* de același tip definiți pentru aceeași comandă nu au garantată o ordine specifică. De exemplu, toți *trigger-ii* *BEFORE* linie definiți pentru o singură comandă *LMD* nu sunt declanșați mereu în aceeași ordine. Din acest motiv, în aplicații nu se recomandă utilizarea mai multor *trigger-i* de același tip care sunt declanșați de aceeași comandă.
-  ❖ Se poate specifica ordinea de execuție a *trigger-ilor* de același tip definiți pentru aceeași comandă?
-  ❖ *Trigger-ii* bază de date pot fi definiți numai pe tabele (excepție, *trigger-ul INSTEAD OF* care este definit pe o vizualizare).

- ❖ Totuși, dacă o comandă *LMD* este aplicată unei vizualizări, pot fi activați *trigger-ii* asociați tabelelor care definesc vizualizarea.



Restricții:

- ❖ În versiunile *Oracle* anterioare corpul unui *trigger* nu poate conține o interogare sau o reactualizare a unui tabel aflat în plin proces de modificare, pe timpul acțiunii *trigger-ului (mutating table)*. Începând cu versiunea *Oracle 11g* această problemă este rezolvată prin utilizarea *trigger-ilor compuși (compound triggers)*.
- ❖ Blocul *PL/SQL* care descrie acțiunea *trigger-ului* nu poate conține comenzi pentru gestiunea tranzacțiilor (*COMMIT*, *ROLLBACK*, *SAVEPOINT*).
 - Controlul tranzacțiilor este permis, însă, în procedurile stocate.
 - Dacă un *trigger* apelează o procedură stocată care execută o comandă referitoare la controlul tranzacțiilor, atunci va apărea o eroare la execuție și tranzacția va fi anulată.
- ❖ Comenzile *DDL* nu pot să apară decât în *trigger-ii* sistem.
- ❖ În corpul *trigger-ului* pot fi referite și utilizate coloane *LOB*, dar nu pot fi modificate valorile acestora.
- ❖ În corpul *trigger-ului* se pot insera date în coloanele de tip *LONG* și *LONGRAW*, dar nu pot fi declarate variabile de acest tip.
- ❖ Dacă un tabel este șters, automat sunt șterși toți *trigger-ii* asociați acestuia.

8.1.4. Predicate condiționale

- În interiorul unui *trigger* care poate fi executat pentru diferite tipuri de instrucțiuni *LMD* se pot folosi trei funcții *booleene* (din pachetul *DBMS_STANDARD*) prin care se stabilește tipul operației executate:
 - *INSERTING* – întoarce valoarea *TRUE* atunci când comanda declanșatoare este o comandă *INSERT*;
 - *DELETING* – întoarce valoarea *TRUE* atunci când comanda declanșatoare este o comandă *DELETE*;
 - *UPDATING* – întoarce valoarea *TRUE* atunci când comanda declanșatoare este o comandă *UPDATE*;
 - *UPDATING('nume_coloană')* întoarce *TRUE* atunci când comanda declanșatoare este o comandă *UPDATE* asupra coloanei *nume_coloană*.



- ❖ Utilizând aceste predicate, în corpul *trigger*-ului se pot executa secvențe de instrucțiuni diferite, în funcție de tipul operației *LMD* declanșatoare.

Exemplul 8.5 – vezi curs

8.2. Trigger-i INSTEAD OF

- Oferă o modalitate de actualizare a vizualizărilor obiect și a celor relaționale.
- Sintaxa:

```

CREATE [OR REPLACE] TRIGGER [schema.]nume_trigger
--momentul când este declanșat
    INSTEAD OF
--comanda/comenzi care îl declanșează
    { DELETE|INSERT|UPDATE [OF coloana[, coloana ...] ] }
    [OR {DELETE|INSERT|UPDATE [OF coloana[, coloana ...] ] ...}]
    ON
        [schema.]nume_vizualizare
    [REFERENCING {OLD [AS] vechi NEW [AS] nou
                  | NEW [AS] nou OLD [AS] vechi } ]
FOR EACH ROW
    [WHEN (condiție) ]
    corp_trigger (bloc anonim PL/SQL sau comanda CALL);

```

- *Trigger*-ul *INSTEAD OF* permite reactualizarea unei vizualizări prin comenzi *LMD*.
- Datorită regulilor stricte existente pentru reactualizarea vizualizărilor, nu orice tip de vizualizare permite reactualizări *LMD*.
 - De exemplu, o vizualizare care este definită pe baza *join*-ului mai multor tabele nu permite reactualizarea tuturor acestor tabelelor.
 - O vizualizare nu poate fi modificată prin comenzi *LMD* dacă aceasta conține operatori pe mulțimi, funcții grup, clauzele *GROUP BY*, *CONNECT BY*, *START WITH* sau operatorul *DISTINCT*.
- *Trigger*-ul *INSTEAD OF* este utilizat pentru a executa operații *LMD* direct pe tabelele de bază ale vizualizării.
 - Comenziile *LMD* lansate asupra unei vizualizări, sunt preluate de *trigger*-ul *INSTEAD OF* (care poate lansa comenziile direct pe tabelele de bază).

- Trigger-ul *INSTEAD OF* poate fi definit asupra vizualizărilor ce au drept câmpuri tablouri imbricate, trigger-ul furnizând o modalitate de reactualizare a elementelor tabloului imbricat.
 - În acest caz, trigger-ul se declanșează doar în cazul în care comenziile *LMD* operează asupra tabloului imbricat (numai când elementele tabloului imbricat sunt modificate folosind clauzele *THE()* sau *TABLE()*) și nu atunci când comanda *LMD* operează doar asupra vizualizării.
 - Atributele *:OLD* și *:NEW* se referă la liniile tabloului imbricat, iar pentru a referi linia curentă din tabloul „părinte“ s-a introdus atributul *:PARENT*.



- ❖ Spre deosebire de trigger-ii *LMD*, trigger-ii *INSTEAD OF* se execută în locul instrucțiunii *LMD (INSERT, UPDATE, DELETE)* specificate.
- ❖ Opțiunea *UPDATE OF* nu este permisă pentru acest tip de trigger.
- ❖ Trigger-ii *INSTEAD OF* se definesc pentru o vizualizare, nu pentru un tabel.
- ❖ Trigger-ii *INSTEAD OF* acționează la nivel de linie.

Exemplul 8.6 – vezi curs

8.3. Trigger-i sistem

- Pot fi definiți la nivelul:
 - bazei de date;
 - schemei.
- Sunt declanșați de:
 - comenzi *LDD (CREATE, DROP, ALTER)*;
 - evenimente sistem (*STARTUP, SHUTDOWN, LOGON, LOGOFF, SUSPEND, SERVERERROR*).
- Sintaxa:

```
CREATE [OR REPLACE] TRIGGER [schema.]nume_trigger
  {BEFORE | AFTER}
  {comenzi_LDD | evenimente_sistem}
  ON {DATABASE | SCHEMA}
  [WHEN (condiție) ]
  corp_trigger;
```

- Pentru trigger-ii sistem se pot utiliza funcții speciale care permit obținerea de informații referitoare la evenimentul declanșator.

- Sunt funcții *PL/SQL* stocate care trebuie prefixate de numele proprietarului (*SYS*):
 - *SYSEVENT* – întoarce evenimentul sistem care a declanșat *trigger-ul* (este de tip *VARCHAR2(20)* și este aplicabilă oricărui eveniment);
 - *DATABASE_NAME* – întoarce numele bazei de date curente (este de tip *VARCHAR2(50)* și este aplicabilă oricărui eveniment);
 - *SERVER_ERROR* – întoarce codul erorii a cărei poziție în stiva erorilor este dată de argumentul de tip *NUMBER* al funcției (este de tip *NUMBER* și este aplicabilă evenimentului *SERVERERROR*);
 - *LOGIN_USER* – întoarce identificatorul utilizatorului care a declanșat *trigger-ul* (este de tip *VARCHAR2(30)* și este aplicabilă oricărui eveniment);
 - *DICTIONARY_OBJ_NAME* – întoarce numele obiectului referit de comanda *LDD* care a declanșat *trigger-ul* (este de tip *VARCHAR2(30)* și este aplicabilă evenimentelor *CREATE, ALTER, DROP*).

8.3.1. Trigger-i pentru comenzi LDD

- Sunt declanșați de comenzi *LDD (CREATE, ALTER, DROP)*.
 - *ON DATABASE* determină declanșarea *trigger-ului* de comenzi *LDD* aplicate asupra obiectelor din orice schemă a bazei de date;
 - *ON SCHEMA* determină declanșarea *trigger-ului* de comenzi *LDD* aplicate asupra obiectelor din schema personală.
- Există restricții asupra expresiilor din condiția clauzei *WHEN*.
 - De exemplu, *trigger-ii LDD* pot verifica tipul și numele obiectelor definite, identificatorul și numele utilizatorului.

Exemplul 8.7 – **vezi curs**

8.3.2. Trigger-i pentru evenimente sistem

- Sunt declanșați de anumite evenimente sistem (*STARTUP, SHUTDOWN, LOGON, LOGOFF, SUSPEND, SERVERERROR*).
 - *ON DATABASE* determină declanșarea *trigger-ului* de evenimente sistem pentru orice schemă a bazei de date;
 - *ON SCHEMA* determină declanșarea *trigger-ului* de evenimente sistem din schema personală.

- Există restricții asupra expresiilor din condiția clauzei *WHEN*.
 - De exemplu, *trigger-ii LOGON* și *LOGOFF* pot verifica doar identificatorul (*userid*) și numele utilizatorului (*username*).

Exemplul 8.8 – vezi curs

Exemplul 8.9 – vezi curs

Exemplul 8.10 – vezi curs

8.4. Modificarea și ștergerea trigger-ilor

Ștergerea trigger-ilor

- Similar procedurilor și pachetelor, un *trigger* poate fi:
 - șters folosind comanda


```
DROP TRIGGER [schema.]nume_trigger;
```

 - recreat folosind clauza *OR REPLACE* din cadrul comenzii *CREATE TRIGGER*
 - clauza permite schimbarea definiției unui *trigger* existent fără suprimarea acestuia.

Modificarea trigger-ilor

- Modificarea unui *trigger* poate consta din:
 - recompilare (*COMPILE*);
 - redenumire (*RENAME*);
 - dezactivare (*DISABLE*);
 - activare (*ENABLE*).

- Sintaxa:

```
ALTER TRIGGER [schema.]nume_trigger
{ENABLE | DISABLE | COMPILE | RENAME TO nume_nou};
```

- Uneori în loc de ștergerea unui *trigger* este preferabilă doar dezactivarea sa temporară.
 - Un *trigger* este activat implicit atunci când acesta este creat.
 - Un *trigger* dezactivat continuă să existe ca obiect în dicționarul datelor, dar este nu va mai fi executat de sistem (până când nu este din nou activat).
- Activarea, respectiv dezactivarea tuturor *trigger-ilor* asociați unui tabel se poate realiza utilizând comanda următoare:

```
ALTER TABLE [schema.]nume_tabel
{ENABLE | DISABLE } ALL TRIGGERS;
```



- ❖ Comanda *ALTER TRIGGER* permite activarea, respectiv dezactivarea unui singur *trigger* (al cărui nume este specificat în comandă).
- ❖ Comanda *ALTER TABLE* permite activarea, respectiv dezactivarea tuturor *trigger-ilor* asociati unui tabel.



- ❖ Comanda *DROP TRIGGER* permite ştergerea unui singur *trigger* (al cărui nume este specificat în comandă).
- ❖ Comanda *DROP TABLE* determină implicit ştergerea tuturor *trigger-ilor* asociati unui tabel.

8.5. Informații despre trigger-i

- Vizualizări din dicționarul datelor ce conțin informații despre *trigger-i*:
 - *USER_OBJETS*
 - atunci când este creat un *trigger* în vizualizare apare o linie nouă cu informații despre acesta (nume, tip, id, data creării, data ultimei modificări, stare etc);
 - tipul obiectului creat este *trigger*.
 - *USER_TRIGGER*
 - conține informații complete despre *trigger* (codul sursă detaliat, starea);
 - *USER_TRIGGER_COLS*
 - conține informații despre coloanele utilizate în *trigger*;
 - *USER_ERRORS*
 - conține informații despre erorile apărute la compilare *trigger-ului*;
 - *USER_DEPENDENCIES*
 - este utilizată pentru a determina dependențele *trigger-ilor*.

Vizualizarea *USER_TRIGGER*

- Vizualizarea include următoarele informații:
 - numele *trigger-ului* (*TRIGGER_NAME*);
 - tipul *trigger-ului* (*TRIGGER_TYPE*);
 - evenimentul declanșator (*TRIGGERING_EVENT*);

- numele proprietarului tabelului (*TABLE_OWNER*);
- numele tabelului pe care este definit *trigger*-ul (*TABLE_NAME*);
 - dacă obiectul referit de *trigger* nu este un tabel sau o vizualizare, atunci *TABLE_NAME* este are valoarea *null*;
- clauza *WHEN* (*WHEN_CLAUSE*);
- corpul *trigger*-ului (*TRIGGER_BODY*);
- antetul (*DESCRIPTION*);
- starea *trigger*-ului (*STATUS*)
 - poate să fie *ENABLED* sau *DISABLED*;
 - numele utilizate pentru a referi parametrii *OLD* și *NEW* (*REFERENCING_NAMES*).
- În operațiile de gestiune a bazei de date este necesară uneori reconstruirea instrucțiunilor *CREATE TRIGGER*, atunci când codul sursă original nu mai este disponibil.
 - Aceasta se poate realiza utilizând informațiile din vizualizarea *USER_TRIGGERS*.

Exemplul 8.11 – vezi curs

8.6. Privilegii sistem

- Sistemul furnizează privilegii sistem pentru gestiunea *trigger*-ilor:
 - *CREATE TRIGGER* (permite crearea *trigger*-ilor în schema personală);
 - *CREATE ANY TRIGGER* (permite crearea *trigger*-ilor în orice schemă cu excepția celei corespunzătoare utilizatorului *SYS*);
 - *ALTER ANY TRIGGER* (permite activarea, dezactivarea sau compilarea *trigger*-ilor din orice schemă cu excepția utilizatorului *SYS*);
 - *DROP ANY TRIGGER* (permite suprimarea *trigger*-ilor la nivel de bază de date din orice schemă cu excepția celei corespunzătoare utilizatorului *SYS*);
 - *ADMINISTER DATABASE TRIGGER* (permite crearea sau modificarea unui *trigger* sistem referitor la baza de date);
 - *EXECUTE* (permite referirea, în corpul *trigger*-ului, a procedurilor, funcțiilor sau pachetelor din alte scheme).

8.7. Tabele *mutating*

- Un tabel *mutating* este un tabel care este modificat curent de o comandă *LMD* (tabelul este aflat în proces de modificare).
 - Un *trigger* la nivel de linie nu poate obține informații (*SELECT*) dintr-un tabel *mutating*, deoarece ar “vedea” date inconsistente (datele din tabel ar fi în proces de modificare în timp ce *trigger*-ul ar încerca să le consulte).
-  ❖ Tabelele nu sunt considerate *mutating* pentru *trigger*-ii la nivel de comandă.
-  ❖ Vizualizările nu sunt considerate *mutating* în *trigger*-ii *INSTEAD OF*.
-  ❖ Atunci când este definit un *trigger* trebuie respectată următoarea regulă: Comenzile *SQL* din corpul *trigger*-ului nu pot consulta sau modifica date dintr-un tabel *mutating*.
- ❖ Chiar tabelul pe care este definit *trigger*-ul este un tabel *mutating*.
- ❖ Există o sigură excepție:
Dacă o comandă *INSERT* afectează numai o înregistrare, *trigger*-ii la nivel de linie pentru înregistrarea respectivă nu tratează tabelul ca fiind *mutating*.
- ❖ Comanda *INSERT INTO tabel SELECT ...* consideră tabelul *mutating* chiar dacă cererea întoarce o singură înregistrare.
-  ❖ Fiecare versiune nouă a bazei de date *Oracle* reduce impactul erorilor *mutating*.
- ❖ Dacă un *trigger* determină o astfel de eroare, atunci singura opțiune este ca acesta să fie rescris (o soluție este utilizarea *trigger*-ilor la nivel de comandă).

Exemplul 8.12 – [vezi curs](#)

Exemplul 8.13 – [vezi curs](#)

Exemplul 8.14 – [vezi curs](#)

Exemplul 8.15 – [vezi curs](#)

Exemplul 8.16 – [vezi curs](#)

Exemplul 8.17 – [vezi curs](#)

Implementați cu ajutorul unui *trigger* următoarea restricție: un client poate beneficia într-un an de cel mult 3 perioade cu prețuri preferențiale.

Bibliografie

1. *Programare avansată în Oracle9i*, I. Popescu, A. Alecu, L. Velcescu, G. Florea (Mihai), Ed. Tehnică (2004)
2. *Oracle Database Concepts 10g Release 2*, Oracle Online Documentation (2012)
3. *Oracle Database PL/SQL User's Guide and Reference 10g Release 2*, Oracle Online Documentation (2012)
4. *Oracle Database PL/SQL Language Reference 11g Release 2*, Oracle Online Documentation (2012)
5. *Oracle Database SQL Reference 10g Release 2*, Oracle Online Documentation (2012)
6. *Oracle Database 11g: PL/SQL Fundamentals, Student Guide*, Oracle University (2009)

CUPRINS

9. PL/SQL – Gestiunea excepțiilor.....	2
9.1. Secțiunea de tratare a excepțiilor.....	4
9.2. Funcții pentru identificarea excepțiilor.....	5
9.3. Excepții interne predefinite	5
9.4. Excepții interne nepredefinite.....	7
9.5. Excepții externe	8
9.6. Cazuri speciale în tratarea excepțiilor	11
9.7. Activarea excepțiilor.....	13
9.8. Propagarea excepțiilor	14
9.8.1 Excepție declanșată în secțiunea executabilă.....	15
9.8.2 Excepție declanșată în secțiunea declarativă	15
9.8.3 Excepție declanșată în secțiunea <i>EXCEPTION</i>	16
9.9. Informații despre erori	17
Bibliografie.....	18

9. PL/SQL – Gestiunea exceptiilor

- Mecanismul de gestiune a exceptiilor permite utilizatorului să definească și să controleze comportamentul programului atunci când acesta generează o eroare.

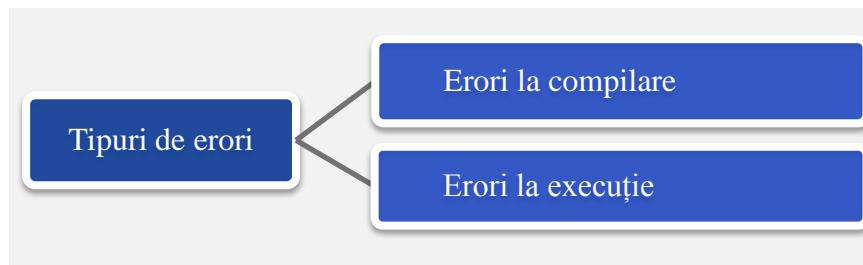


Fig. 9.1. Tipuri de erori

- Într-un program *PL/SQL* pot să apară:
 - erori la compilare
 - sunt detectate de motorul *PL/SQL*;
 - programul nu poate trata aceste erori deoarece nu a fost încă executat;
 - programatorul trebuie să corecteze erorile și să execute din nou programul;
 - erori la execuție
 - sunt denumite exceptii;
 - pot apărea datorită deficiențelor de proiectare, defecțiunilor la nivel *hardware*, greșelilor de cod etc.;
 - în program trebuie prevăzută apariția unei astfel de erori și specificat modul concret de tratare a acesteia;
 - atunci când apare eroarea este declanșată o excepție, iar controlul trece la secțiunea de tratare a exceptiilor, unde va avea loc tratarea erorii;
 - dacă excepția nu este tratată, atunci aceasta se va propaga în mediul din care a fost lansat programul;
 - nu pot fi anticipate toate excepțiile posibile, dar prin mecanismul de tratare a exceptiilor se poate permite programului să își continue execuția și în prezența unumitor erori.
- Exceptiile pot fi definite și tratate la nivelul fiecărui bloc din program (bloc principal, funcții și proceduri, blocuri interioare acestora).
 - Execuția unui bloc se termină întotdeauna atunci când apare o excepție, dar se pot executa acțiuni ulterioare apariției acesteia, într-o secțiune specială de tratare a exceptiilor.

- Posibilitatea de a da nume fiecărei excepții, de a izola tratarea erorilor într-o secțiune particulară, de a declanșa automat erori (în cazul excepțiilor interne) îmbunătățește lizibilitatea și fiabilitatea programului. Prin utilizarea excepțiilor și a rutinelor de tratare a excepțiilor, un program *PL/SQL* devine robust și capabil să trateze atât erorile așteptate, cât și cele neașteptate ce pot apărea în timpul execuției.

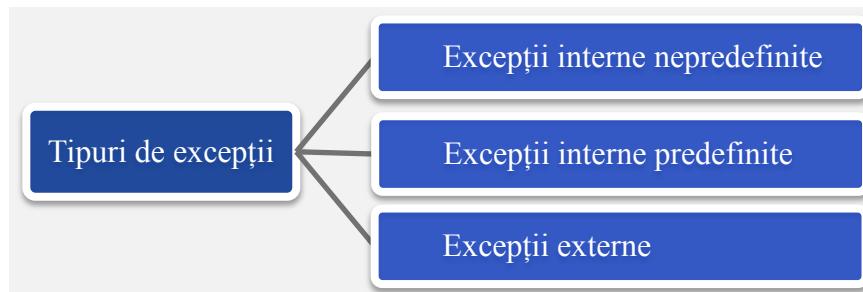


Fig. 9.2. Tipuri de excepții

- Există următoarele tipuri de excepții:
 - excepții interne nepredefinite
 - au un cod de eroare, dar nu au nume asociat decât dacă acesta este precizat de către utilizator;
 - excepții interne predefinite
 - sunt excepții interne care au nume asociat de către sistem;
 - excepții externe
 - sunt excepții definite de utilizator în blocuri *PL/SQL* anonime, subprograme sau pachete.
- Excepțiile interne:
 - se produc atunci când un bloc *PL/SQL* nu respectă o regulă *Oracle* sau depășește o limită a sistemului de operare;
 - pot fi independente de structura bazei de date sau pot să apară datorită nerespectării constrângerilor statice implementate în structură (*PRIMARY KEY*, *FOREIGN KEY*, *NOT NULL*, *UNIQUE*, *CHECK*);
- Atunci când apare o eroare *Oracle*, excepția asociată acesteia se declanșează implicit.
 - De exemplu, dacă apare eroarea *ORA-01403* (deoarece o comandă *SELECT* nu întoarce nicio linie) atunci implicit *PL/SQL* activează excepția *NO_DATA_FOUND* (al cărui cod este 100).
 - Cu toate că fiecare astfel de excepție are asociat un cod specific, ele trebuie referite prin nume.

	Excepții interne nepredefinite	Excepții interne predefinite	Excepții externe
Proprietar	Sistemul	Sistemul	Utilizatorul
Cod asociat	Da	Da	Doar dacă utilizatorul îl atribuie
Nume asociat	Doar dacă utilizatorul îl atribuie	Da	Da
Declanșare automată	Da	Da	Nu
Declanșare explicită	Opțional	Opțional	Da

Fig. 9.3. Caracteristicile excepțiilor

9.1. Secțiunea de tratare a excepțiilor

- Tratarea excepțiilor se realizează în secțiunea *EXCEPTION* a unui bloc *PL/SQL*.
- Sintaxa:

```
EXCEPTION
  WHEN nume_excepție1 [OR nume_excepție2 ...] THEN
    secvența_de_instrucțiuni_1;
  [WHEN nume_excepție3 [OR nume_excepție4 ...] THEN
    secvența_de_instrucțiuni_2;
  ...
  [WHEN OTHERS THEN
    secvența_de_instrucțiuni_n;]
END;
```

- Clauza *WHEN OTHERS* trebuie să fie ultima clauză specificată și trebuie să fie unică.
 - Toate excepțiile care nu au fost specificate explicit vor fi captate prin această clauză.

9.2. Funcții pentru identificarea exceptiilor

- Funcția *SQLCODE*:
 - obține codul exceptiei;
 - întoarce o valoare de tip numeric;
 - codul exceptiei este:
 - un număr negativ, în cazul unei erori interne;
 - numărul +100, în cazul exceptiei *NO_DATA_FOUND*;
 - numărul 0, în cazul unei execuții normale (fără exceptii);
 - numărul 1, în cazul unei exceptii definite de utilizator.
- Funcția *SQLERRM*:
 - obține mesajul asociat exceptiei;
 - întoarce un sir de caractere;
 - lungimea maximă a mesajului este de 512 caractere;
 - mesajul asociat exceptiei declanșate poate fi furnizat și de funcția *DBMS_UTILITY.FORMAT_ERROR_STACK*.

Exemplul 9.1 – **vezi curs**

9.3. Exceptii interne predefinite

- Exceptiile interne predefinite (erori de tip *ORA-n*):
 - nu trebuie declarate în secțiunea declarativă a blocului *PL/SQL*;
 - sunt declanșate implicit de către *server-ul Oracle*;
 - sunt referite prin numele asociat lor;
 - *PL/SQL* declară aceste exceptii în pachetul *STANDARD*.

Nume exceptie	Cod	Descriere
ACCESS_INTO_NULL	-6530	Asignare de valori atributelor unui obiect neinitializat.
CASE_NOT_FOUND	-6592	Nu este selectată nici una din clauzele <i>WHEN</i> ale lui <i>CASE</i> și nu există nici clauza <i>ELSE</i> .
COLLECTION_IS_NULL	-6531	Aplicarea unei metode (diferite de <i>EXISTS</i>) unui tabel imbricat sau unui vector neinitializat.

CURSOR_ALREADY_OPEN	-6511	Deschiderea unui cursor care este deja deschis.
DUP_VAL_ON_INDEX	-1	Detectarea unei dubluri într-o coloană unde acestea sunt interzise.
INVALID_CURSOR	-1001	Operație ilegală asupra unui cursor.
INVALID_NUMBER	-1722	Conversie nepermisă de la tipul sir de caractere la număr.
LOGIN_DENIED	-1017	Nume sau parolă incorecte.
NO_DATA_FOUND	+100	Comanda <i>SELECT</i> nu întoarce nicio linie.
NOT_LOGGED_ON	-1012	Programul <i>PL/SQL</i> apelează baza de date fără să fie conectat la <i>Oracle</i> .
PROGRAM_ERROR	-6501	<i>PL/SQL</i> are o problemă internă.
ROWTYPE_MISMATCH	-6504	Incompatibilitate între parametrii actuali și formali, la deschiderea unui cursor parametrizat.
SELF_IS_NULL	-30625	Apelul unei metode când instanța este <i>NULL</i> .
STORAGE_ERROR	-6500	<i>PL/SQL</i> are probleme cu spațiul de memorie.
SUBSCRIPT_BEYOND_COUNT	-6533	Referire la o componentă a unui tablou imbricat sau vector, folosind un index mai mare decât numărul elementelor colecției respective.
SUBSCRIPT_OUTSIDE_LIMIT	-6532	Referire la o componentă a unui tabel imbricat sau vector, folosind un index care este în afara domeniului (de exemplu, -1).
SYS_INVALID_ROWID	-1410	Conversia unui sir de caractere într-un <i>ROWID</i> nu se poate face deoarece sirul nu reprezintă un <i>ROWID</i> valid.
TIMEOUT_ON_RESOURCE	-51	Expirarea timpului de așteptare pentru eliberarea unei resurse.
TRANSACTION_BACKED_OUT	-61	Tranzacția este anulată datorită unei interblocări.
TOO_MANY_ROWS	-1422	Comanda <i>SELECT</i> întoarce mai multe linii.

VALUE_ERROR	-6502	Apariția unor erori în conversii, constrângeri sau erori aritmetice.
ZERO_DIVIDE	-1476	Sesizarea unei împărțiri la zero.



- ❖ Aceeași excepție poate să apară în diferite circumstanțe.
- ❖ De exemplu, excepția *NO_DATA_FOUND* poate fi generată fie pentru că o interogare nu întoarce un rezultat, fie pentru că se referă un element al unui tablou *PL/SQL* care nu a fost definit (nu are atribuită o valoare).
- ❖ Dacă într-un bloc *PL/SQL* apar ambele situații, este greu de stabilit care dintre ele a generat eroarea și este necesară restructurarea blocului, astfel încât acesta să poată diferenția cele două situații.

Exemplul 9.2 – [vezi curs](#)



- ❖ Deși excepțiile interne sunt lansate implicit (automat) de către sistem, sunt cazuri în care utilizatorul le poate invoca explicit.

Exemplul 9.3 – [vezi curs](#)

9.4. Excepții interne nepredefinite

- Excepțiile interne nepredefinite se declară în secțiunea declarativă a blocului *PL/SQL* și sunt declanșate implicit de către *server-ul Oracle*.
- Diferențierea acestor erori este posibilă doar cu ajutorul codului asociat lor.
- Există două metode de tratare a excepțiilor interne nepredefinite, folosind:
 - clauza *WHEN OTHERS* din secțiunea *EXCEPTION* a blocului;
 - directiva de compilare (pseudo-instrucțiune) *PRAGMA EXCEPTION_INIT*.
- Directiva *PRAGMA EXCEPTION_INIT*
 - permite asocierea unui nume pentru o excepție al cărui cod de eroare intern este specificat;
 - având un nume specificat pentru excepție, permite referirea acesteia în secțiunea *EXCEPTION* a blocului;
 - este procesată în momentul compilării (nu la execuție);

- trebuie să apară în partea declarativă a unui bloc, pachet sau subprogram, după definirea numelui excepției;
 - poate să apară de mai multe ori într-un program; de asemenea, pot fi asignate mai multe nume pentru același cod de eroare.
- Pentru a trata o eroare folosind directiva *PRAGMA EXCEPTION_INIT* trebuie urmări pașii de mai jos:
 - 1) se declară numele excepției în partea declarativă a blocului:
`nume_excepție EXCEPTION;`
 - 2) se asociază numele excepției cu un cod de eroare standard *Oracle*:
`PRAGMA EXCEPTION_INIT (nume_excepție, cod_eroare);`
 - 3) se referă excepția în secțiunea *EXCEPTION* a blocului (excepția este tratată automat, fără a fi necesară comanda *RAISE*):
`WHEN nume_excepție THEN set_de_instrucțiuni`

Exemplul 9.4 - vezi curs

9.5. Excepții externe

- Excepțiile externe:
 - sunt excepții definite de utilizator;
 - sunt declarate în secțiunea declarativă a unui bloc, subprogram sau pachet;
 - sunt activate explicit în partea executabilă a blocului (folosind comanda *RAISE* însotită de numele excepției);
 - pot să apară în toate secțiunile unui bloc, subprogram sau pachet;
 - nu pot să apară în instrucțiuni de atribuire sau în comenzi *SQL*;
 - în mod implicit, au asociat același cod (+1) și același mesaj (*User-Defined Exception*).
- Sintaxa de declarare și prelucrare a excepțiilor externe :

```

DECLARE
    nume_excepție EXCEPTION; -- declarare excepție
BEGIN
    ...
    RAISE nume_excepție; -- activare excepție
    -- execuția este întreruptă și se transferă
    -- controlul în secțiunea EXCEPTION
  
```

```

...
EXCEPTION
    WHEN nume_excepție THEN
        -- definire mod de tratare a erorii
        ...
END;

```

Exemplul 9.5 - vezi curs

- Activarea unei excepții externe poate fi făcută și cu ajutorul procedurii *RAISE_APPLICATION_ERROR*, furnizată de pachetul *DBMS_STANDARD*.
 - Această procedură poate fi utilizată pentru a întoarce un mesaj de eroare unității care o apelează, mesaj mai descriptiv (non standard) decât identificatorul erorii.
 - Are următoarea specificație:

```

RAISE_APPLICATION_ERROR (num NUMBER,
                         msg VARCHAR2, keeperrorstack BOOLEAN);

```

 - parametrul *num* reprezintă codul asociat erorii, un număr cuprins între -20000 și -20999;
 - parametrul *msg* reprezintă mesajul asociat erorii, un sir de caractere de maxim 2048 bytes;
 - parametrul *keeperrorstack* este opțional; dacă are valoarea *TRUE*, atunci noua eroare se va adăuga listei erorilor existente, iar dacă este *FALSE* (valoare implicită) atunci noua eroare va înlocui lista curentă a erorilor (se reține ultimul mesaj de eroare).
-  O aplicație poate apela procedura *RAISE_APPLICATION_ERROR* numai dintr-un subprogram stocat (sau metodă).
 - Dacă procedura *RAISE_APPLICATION_ERROR* este apelată, atunci subprogramul se termină și sunt întoarce codul și mesajul asociate erorii respective.
- Procedura *RAISE_APPLICATION_ERROR* poate fi apelată în secțiunea executabilă, în secțiunea de tratare a excepțiilor sau simultan în ambele secțiuni.
 - În secțiunea executabilă:

```

DELETE FROM produse WHERE denumire = 'produs';
IF SQL%NOTFOUND THEN
    RAISE_APPLICATION_ERROR(-20001,'Date incorecte');
END IF;

```

- În secțiunea de tratare a excepțiilor:

```
EXCEPTION
  WHEN exceptie THEN
    RAISE_APPLICATION_ERROR(-20002,'info invalida');
END;
```

- În ambele secțiuni:

```
DECLARE
  exceptie EXCEPTION;
  PRAGMA EXCEPTION_INIT (exceptie, -20000);
BEGIN
  DELETE produse WHERE denumire = 'produs cautat';
  IF SQL%NOTFOUND THEN
    RAISE_APPLICATION_ERROR(-20000,'Date incorecte');
  END IF;
EXCEPTION
  WHEN exceptie THEN
    DBMS_OUTPUT.PUT_LINE('Mesajul exceptiei: '||SQLERRM);
    DBMS_OUTPUT.PUT_LINE('Codul exceptiei: '||SQLCODE);
END;
```



- ❖ Procedura *RAISE_APPLICATION_ERROR* facilitează comunicația dintre *client* și *server*, transmițând aplicației *client* erori specifice aplicației de pe *server* (de obicei, un *trigger*).
- ❖ Procedura *RAISE_APPLICATION_ERROR* este doar un mecanism folosit pentru comunicația dintre *server* și *client* a unei erori definite de utilizator, care permite ca procesul *client* să trateze excepția.

Exemplul 9.6

```
CREATE OR REPLACE TRIGGER trig
  BEFORE UPDATE OF serie ON case
  FOR EACH ROW
  WHEN (NEW.serie <> OLD.serie)
BEGIN
  RAISE_APPLICATION_ERROR (-20145,
    'Nu puteti modifica seria casei fiscale!');
END;
/

--blocul urmator detecteaza si trateaza eroarea
DECLARE
  -- declarare exceptie
  exceptie EXCEPTION;
  -- asociere un nume codului de eroare folosit in trigger
  PRAGMA EXCEPTION_INIT(exceptie,-20145);
```

```

BEGIN
    -- lansare comanda declasatoare
    UPDATE case
    SET serie = serie||'_';
EXCEPTION
    -- tratare exceptie
    WHEN exceptie THEN
        -- se afiseaza mesajul erorii specificat in trigger
        -- in procedura RAISE_APPLICATION_ERROR
        DBMS_OUTPUT.PUT_LINE(SQLERRM);
END;
/

```

9.6. Cazuri speciale în tratarea excepțiilor

- Dacă se declanșează o excepție într-un bloc simplu atunci:
 - execuția blocului este întreruptă (setul de comenzi care urmează după comanda care a declanșat excepția nu se mai execută);
 - controlul este transferat în secțiunea de tratare a excepțiilor;
 - se tratează excepția (se execută comenziile specificate în *handler*-ul excepției respective);
 - se ieșe din bloc.

- ❖ Dacă după o eroare se dorește totuși continuarea prelucrării datelor, este suficient ca instrucțiunea care a declanșat excepția să fie inclusă într-un subbloc.
- ❖ În acest caz, după tratarea excepției și ieșirea din subbloc, se continuă secvența de instrucțiuni din blocul principal.

```

BEGIN
    comanda_1;  -- declanseaza exceptia E
    comanda_2;  -- nu se mai executa
EXCEPTION
    WHEN E THEN
        set_comenzi; -- se executa
END;

```

- Dacă se dorește execuția comenzi *comanda_2* chiar și atunci când *comanda_1* declanșează o excepție, atunci *comanda_1* se include într-un subbloc, iar excepția declanșată de aceasta este tratată în acel subbloc.

```

BEGIN
    BEGIN
        comanda_1; -- declanseaza exceptia E
    EXCEPTION
        WHEN E THEN
            set_comenzi; -- se executa
    END;

    comanda_2; -- se executa
    EXCEPTION
        ...
    END;

```

- Uneori este dificil de aflat care comandă *SQL* a determinat o anumită eroare, deoarece există o singură secțiune pentru tratarea erorilor unui bloc.

Variante de rezolvare a acestor situații:

- utilizarea unui contor care să identifice instrucțiunea *SQL* care a declanșat excepția.

```

DECLARE
    contor NUMBER(1),
BEGIN
    contor :=1;
    comanda_SELECT_1;
    contor :=2;
    comanda_SELECT_2;
    contor :=3;
    comanda_SELECT_3;
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        DBMS_OUTPUT.PUT_LINE('comanda SELECT '|| contor);
END;

```

- Introducerea fiecărei instrucțiuni *SQL* într-un subbloc

```

BEGIN
    BEGIN
        comanda_SELECT_1;
    EXCEPTION
        WHEN NO_DATA_FOUND THEN
            DBMS_OUTPUT.PUT_LINE('comanda SELECT 1');
    END;

    BEGIN
        comanda_SELECT_2;
    EXCEPTION
        WHEN NO_DATA_FOUND THEN
            DBMS_OUTPUT.PUT_LINE('comanda SELECT 2');
    END;

```

```

BEGIN
    comanda_SELECT_3;
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        DBMS_OUTPUT.PUT_LINE('comanda SELECT 3');
END;
END;

```

9.7. Activarea excepțiilor

- Există două metode de activare a unei excepții:
 - activarea explicită a excepției (definită de utilizator sau predefinită) în interiorul blocului, cu ajutorul comenzi *RAISE*;
 - activarea automată a excepției asociate unei erori *Oracle*.
- Excepțiile pot fi generate în oricare dintre secțiunile unui bloc *PL/SQL* (în secțiunea declarativă, în secțiunea executabilă sau în secțiunea de tratare a excepțiilor).
 - La aceste niveluri ale programului, o excepție poate fi gestionată în moduri diferite.
- Pentru a reinvoca o excepție, după ce a fost tratată în blocul curent, se folosește instrucțiunea *RAISE*, dar fără a fi însorită de numele excepției.
 - În acest fel, după executarea instrucțiunilor corespunzătoare tratării excepției, aceasta se transmite și blocului „părinte“.
 - Pentru a fi recunoscută ca atare de către blocul „părinte“, excepția trebuie să nu fie definită în blocul curent, ci în blocul „părinte“ (sau chiar mai sus în ierarhie), în caz contrar ea putând fi captată de către blocul „părinte“ doar la categoria *OTHERS*.
- Pentru a executa același set de acțiuni în cazul mai multor excepții nominalizate explicit, în secțiunea de tratare a excepțiilor se poate utiliza operatorul *OR* (*WHEN exceptie_1 OR exceptie_2 THEN ...*).
- Pentru a evita tratarea fiecărei erori în parte, se folosește secțiunea *WHEN OTHERS* care va cuprinde acțiuni pentru fiecare excepție care nu a fost tratată, adică pentru captarea excepțiilor neprevăzute sau necunoscute.
 - Această secțiune trebuie utilizată cu atenție deoarece poate masca erori critice sau poate împiedica aplicația să răspundă în mod corespunzător.

9.8. Propagarea excepțiilor

- Dacă este declanșată o eroare în secțiunea executabilă, iar
 - blocul curent are un *handler* pentru tratarea acesteia, atunci blocul curent se termină cu succes și controlul este transmis blocului imediat exterior;
 - blocul curent nu are un *handler* pentru tratarea acesteia, atunci excepția se propagă spre blocul „părinte“, iar blocul curent se termină fără succes;
 - procesul se repetă până când fie se găsește într-un bloc modalitatea de tratare a erorii, fie se oprește execuția și se semnalează situația apărută (*unhandled exception error*).
- Dacă este declanșată o eroare în secțiunea declarativă a blocului sau în secțiunea de tratare a erorilor, atunci aceasta este propagată către blocul imediat exterior, chiar dacă există un *handler* al acesteia în blocul curent.
- La un moment dat, într-o secțiune *EXCEPTION*, poate fi activă numai o singură excepție.



- ❖ Instrucțiunea *GOTO* nu permite:
 - saltul la secțiunea de tratare a unei excepții;
 - saltul de la secțiunea de tratare a unei excepții, în blocul curent.
- ❖ Comanda *GOTO* permite totuși saltul de la secțiunea de tratare a unei excepții la un bloc care include blocul curent.

Exemplul 9.7

```

DECLARE
  v_den  produse.denumire%TYPE;
  v_id   produse.id_produs%TYPE := &p_id;
BEGIN
  SELECT denumire
  INTO   v_den
  FROM   produse
  WHERE   id_produs = v_id;
  <<eticheta>>
  DBMS_OUTPUT.PUT_LINE('Denumirea produsului este '||v_den);
EXCEPTION
  WHEN NO_DATA_FOUND THEN v_den := ' ';
    GOTO eticheta; --salt ilegal in blocul curent
END;
/

```

9.8.1 Excepție declanșată în secțiunea executabilă

- Excepția este produsă și tratată în subbloc. După tratarea excepției controlul este transmis blocului exterior.

```

DECLARE
  A EXCEPTION;
BEGIN
  ...
  BEGIN
    RAISE A; -- in subbloc se produse exceptia A
  EXCEPTION
    WHEN A THEN ...-- exceptia A este tratata in subbloc
    ...
  END;
  -- aici este reluat controlul
END;

```

- Excepția este produsă în subbloc, dar nu este tratată în acesta. Excepția se propagă spre blocul exterior. Regula poate fi aplicată de mai multe ori.

```

DECLARE
  A EXCEPTION;
  B EXCEPTION;
BEGIN
  BEGIN
    RAISE B; -- in subbloc se produse exceptia B
  EXCEPTION
    WHEN A THEN ...
      --exceptia B nu este tratata in subbloc
  END;
  EXCEPTION
    WHEN B THEN ...
      /*exceptia B s-a propagat spre blocul exterior unde este
       tratata, apoi controlul este dat in exteriorul blocului */
  END;

```

9.8.2 Excepție declanșată în secțiunea declarativă

- Dacă în secțiunea declarativă este generată o excepție, atunci aceasta se propagă către blocul exterior, unde are loc tratarea acesteia. Chiar dacă există un *handler* pentru excepție în blocul curent, acesta nu este executat.

Exemplul 9.8

```

BEGIN
  DECLARE
    nr_produse NUMBER(10) := ' ';
    -- este generata eroarea VALUE_ERROR
  BEGIN
    SELECT COUNT (DISTINCT id_produs)
    INTO   nr_produse
    FROM   facturi_produse;
  EXCEPTION
    WHEN VALUE_ERROR THEN
      -- eroarea nu este captata si tratata in blocul intern
      DBMS_OUTPUT.PUT_LINE('Eroare bloc intern: ' || SQLERRM);
  END;
  EXCEPTION
    WHEN VALUE_ERROR THEN
      -- eroarea este captata si tratata in blocul extern
      DBMS_OUTPUT.PUT_LINE('Eroare bloc extern: ' || SQLERRM);
  END;
/

```

9.8.3 Excepție declanșată în secțiunea *EXCEPTION*

- Dacă excepția este declanșată în secțiunea *EXCEPTION*, atunci aceasta se propagă imediat spre blocul exterior.

```

BEGIN
  DECLARE
    A EXCEPTION;
    B EXCEPTION;
  BEGIN
    RAISE A; -- este generata exceptia A
  EXCEPTION
    WHEN A THEN
      RAISE B; -- este generata exceptia B
    WHEN B THEN ...
      /* exceptia este propagata spre blocul exterior
         cu toate ca exista aici un handler pentru ea */
  END;
  EXCEPTION
    WHEN B THEN ...
      --exceptia B este tratata in blocul exterior
  END;

```

9.9. Informații despre erori

- Pentru a obține textul corespunzător erorilor apărute la **compilare**, poate fi utilizată vizualizarea *USER_ERRORS* din dicționarul datelor.
 - Pentru informații adiționale referitoare la erori pot fi consultate vizualizările *ALL_ERRORS* sau *DBA_ERRORS*.
 - Vizualizarea *USER_ERRORS* oferă informații despre obiectele care au generat erori la compilare; de exemplu:
 - numele obiectului (*NAME*);
 - tipul obiectului (*TYPE*);
 - numărul liniei din codul sursă la care a apărut eroarea (*LINE*);
 - poziția din linie (*POSITION*);
 - mesajul asociat erorii (*TEXT*).
-  ♦ *LINE* specifică numărul liniei în care apare eroarea, dar acesta nu corespunde liniei efective din fișierul text (se referă la codul sursă depus în *USER_SOURCE*).

Exemplul 9.9

```

CREATE OR REPLACE FUNCTION f_test
RETURN NUMBER;
IS
BEGIN
    RETURN 1;
END;
/
FUNCTION F_TEST compiled
Errors: check compiler log

SELECT LINE, POSITION, TEXT
FROM   USER_ERRORS
WHERE  NAME = UPPER('f_test');

LINE    POSITION    TEXT
----  -----
3        1          PLS-00103: Encountered the symbol "IS"

```

Bibliografie

1. *Programare avansată în Oracle9i*, I. Popescu, A. Alecu, L. Velcescu, G. Florea (Mihai), Ed. Tehnică (2004)
2. *Oracle Database PL/SQL Language Reference 11g Release 2*, Oracle Online Documentation (2012)
3. *Oracle Database SQL Language Reference 11g Release 2*, Oracle Online Documentation (2012)
4. *Oracle Database 11g: PL/SQL Fundamentals, Student Guide*, Oracle University (2009)

CUPRINS

10. Structura bazei de date	2
10.1. Structura fizică a bazei de date	2
10.1.1. Fișiere de date	2
10.1.2. Fișiere de reluare.....	3
10.1.3. Fișiere de control.....	4
10.2. Structura logică a bazei de date	5
10.2.1. Blocuri de date	5
10.2.2. Extensii	7
10.2.3. Segmente.....	9
10.2.4. Spații tabel	11
10.2.5. Scheme de obiecte.....	15
10.3. Dicționarul datelor.....	16
Bibliografie	20

10. Structura bazei de date

- Baza de date *Oracle*:
 - este o colecție de date tratate unitar;
 - este creată pentru a asigura stocarea și extragerea informațiilor.
- Orice bază de date are o structură fizică și o structură logică. Deoarece structura fizică este separată de cea logică, stocarea fizică a datelor poate fi administrată fără să afecteze accesul la structurile logice.

10.1. Structura fizică a bazei de date

- Structura fizică a bazei de date *Oracle* presupune existența unui set de fișiere binare prin intermediul cărora se realizează stocarea fizică a datelor.
- Aceste fișiere sunt împărțite în următoarele categorii:
 - fișiere de date (*Datafiles*);
 - fișiere de reluare (*Redo Log Files*);
 - fișiere de control (*Control Files*).

10.1.1. Fișiere de date

- Sunt fișiere fizice ale sistemului de operare care stochează:
 - datele tuturor structurilor logice ale bazei (obiecte create de utilizator);
 - informația necesară funcționării sistemului.
 - Primul fișier de date creat este cel care stochează dicționarul datelor. Dimensiunea acestui fișier este de cel puțin 150 MB.
- Caracteristicile fișierelor de date:
 - dimensiunea fișierelor de date poate fi schimbată ulterior creării acestora și poate fi extinsă automat până la o valoare specificată, atunci când spațiul alocat este depășit (alocare dinamică a spațiului);
 - unul sau mai multe fișiere de date formează o unitate logică numită spațiu tabel (*tablespace*);
 - un fișier de date poate fi asociat unui singur spațiu tabel și unei singure baze de date.



- ❖ Informațiile din fișierele de date pot fi accesate în timpul operațiilor uzuale asupra bazei și stocate în memoria *cache*.
- ❖ Dacă un utilizator interoghează date dintr-un tabel al bazei și informațiile cererii solicitate nu sunt încă în memoria *cache*, atunci acestea vor fi preluate din fișierul de date și stocate în memorie.
- Alocarea unui fișier de date se face conform sintaxei următoare:

```
DATAFILE ['nume_fișier']
[SIZE întreg [ {K | M} ] ] [REUSE]
[clauza_autoextend]
```

- În cazul spațiilor tabel *undo*, pentru fiecare fișier de date asociat trebuie specificată o dimensiune.
- Pentru celelalte spații tabel, clauza *SIZE* poate fi omisă dacă fișierul există deja.
- Opțiunea *REUSE* permite sistemului să reutilizeze fișierele deja existente. Dacă fișierul specificat nu există, sistemul *Oracle* va ignora această clauză și îl va crea.
- Dacă se dorește ca fișierele de date să fie auto-extensibile, atunci se precizează *clauza_autoextend*.

10.1.2. Fișiere de reluare

- Înregistrează toate modificările care au loc asupra datelor bazei (indiferent dacă au fost permanentizate sau nu) și care nu au fost scrise încă în fișierele de date.
- O bază de date *Oracle* conține două sau mai multe fișiere de reluare.
 - Specificarea acestora se face în momentul creării sau modificării bazei.
 - Fișierele de reluare sunt utilizate în manieră circulară. Cele care au fost folosite în întregime, pot fi arhivate până când sistemul le va reutiliza.
- Fișierele de reluare asigură protecția bazei de date în cazul defecțiunilor.
 - De exemplu, dacă operațiile bazei sunt întrerupte din cauza unei avarii de curent, datele din memorie nu pot fi scrise în fișierele de date și sunt pierdute temporar. Atunci când baza de date este repornită, informațiile vor putea fi recuperate din cele mai recente fișiere de reluare.
 - Deoarece și aceste fișiere pot fi alterate, sistemul *Oracle* permite utilizarea fișierelor de reluare multiplexate. Aceasta presupune că pentru fiecare fișier de reluare se creează un grup de copii ale sale care sunt localizate pe discuri separate.

Orice modificare făcută asupra unuia dintre membrii grupului se propagă la întregul grup. Dacă un fișier de reluare este pierdut sau afectat, poate fi utilizată o copie a acestuia de pe alt disc.

- Vizualizările *V\$LOG* și *V\$LOGFILE* din dicționarul datelor furnizează informații despre grupurile fișierelor de reluare și membrii acestora.

10.1.3. Fișiere de control

- Sunt fișiere binare de dimensiune redusă, necesare pentru pornirea și funcționarea bazei de date.
 - Fiecare fișier de control este asociat unei singure baze de date și conține informații despre structura fizică a acesteia (numele și data creării bazei, informații despre spațiile tabel, numele și locația fișierelor de date și a fișierelor de reluare, informații despre operațiile de *backup* sau despre *checkpoint*-urile efectuate etc.).
- Orice bază de date *Oracle* deține cel puțin un fișier de control, care este creat odată cu aceasta. Ca și în cazul fișierelor de reluare, sistemul *Oracle* permite existența fișierelor de control multiplexate. În mod implicit, sistemul realizează cel puțin o copie a fișierului de control pe durata creării bazei de date.
- La pornirea unei instanțe *Oracle*, sistemul folosește fișierul de control pentru a identifica baza și a determina dacă aceasta este în stare validă pentru utilizare. De asemenea, sunt identificate fișierele de reluare necesare execuției operațiilor bazei de date.
- Fișierele de control reflectă automat schimbările (creare, redenumire sau ștergere) care au loc la nivelul fișierelor de date sau de reluare. Informațiile din fișierele de control pot fi modificate doar de *server-ul Oracle*.
- Dacă un fișier de control devine inaccesibil, baza de date nu va funcționa la parametrii corespunzători. Dacă toate copiile fișierelor de control ale bazei sunt pierdute, atunci aceasta trebuie recuperată (*recovery*) înainte de a fi deschisă.
- Interogând anumite vizualizări din dicționarul datelor se pot obține următoarele informații despre fișierele de control:
 - numele și starea fișierelor asociate instanței (*V\$CONTROLFILE*);
 - starea și localizarea tuturor parametrilor (*V\$PARAMETER*);
 - înregistrările conținute (*V\$CONTROLFILE_RECORD_SECTION*).

10.2. Structura logică a bazei de date

- Utilizarea adecvată a structurilor logice determină alocarea optimă a spațiului de pe disc în vederea obținerii unei baze de date eficiente.
- Componentele structurii logice a unei baze de date *Oracle* sunt:
 - blocurile de date (*data block*);
 - extensiile (*extent*);
 - segmentele (*segment*);
 - spațiile tabel (*tablespace*);
 - obiectele schemei (*schema object*).
- Gestiona dinamică a spațiului de pe disc pe măsura utilizării bazei de date, se realizează prin trei niveluri de granularitate:
 - segmentul;
 - extensia;
 - blocul.
- Spațiile tabel, segmentele, extensiile și blocurile permit definirea logică a organizării fizice a bazei de date și efectuarea legăturii dintre nivelul fizic și nivelul logic al acestora.
- Obiectele schemei (tabele, vizualizări, vizualizări materializate, secvențe, unități de program, sinonime, indecsi, grupări, dimensiuni, legături de baze de date) sunt structuri logice care referă în mod direct datele bazei.

10.2.1. Blocuri de date

- Sistemul administrează spațiul de stocare al fișierelor de date prin unități logice numite blocuri de date.
- Blocul reprezintă cea mai mică unitate *I/O* folosită de baza de date, corespunzătoare unui bloc fizic de octeți de pe disc.
- Dimensiunea blocului de date este definită în momentul creării bazei și poate fi modificată ulterior.
 - Aceasta trebuie să reprezinte un multiplu al dimensiunii blocurilor fizice de la nivelul sistemului de operare.
 - Modificarea dimensiunii blocurilor logice se face prin intermediul parametrului *DB_BLOCK_SIZE*.

- Din punct de vedere structural, blocul de date *Oracle* cuprinde:
 - un antet (*header*), care conține informații generale referitoare la bloc:
 - adresa;
 - tipul segmentului căruia îi aparține;
 - un catalog al tabelelor (*table directory*);
 - cuprinde informații despre tabelele care au date (înregistrări) stocate în blocul respectiv;
 - un catalog al liniilor (*row directory*);
 - conține informații despre liniile situate în bloc (de exemplu, adresa acestora);
 - odată ce s-a alocat spațiu pentru catalogul liniilor, acesta nu poate fi revendicat de celelalte porțiuni ale blocului de date; sistemul îl va utiliza doar atunci când sunt introduse liniile noi în bloc;
 - un spațiu pentru date (*data space*):
 - este format din liniile de date care conțin informațiile stocate în tabele sau indecsă;
 - un bloc poate cuprinde una sau mai multe liniile de date; dacă o linie nu încape într-un bloc, atunci aceasta poate fi stocată în două sau mai multe blocuri înlántuite (de exemplu, dacă tabelul conține o coloană de tip *LOB*);
 - un spațiu liber (*free space*):
 - este alocat pentru inserarea de noi liniile sau actualizarea liniilor care necesită spațiu suplimentar;
 - alegerea blocului în care va fi inserată o linie nouă depinde de spațiul liber al acestuia și de valorile parametrilor *PCTFREE* și *PCTUSED*:
 - parametrul *PCTFREE* reprezintă procentul minim din blocul de date care trebuie păstrat liber pentru actualizările liniilor deja existente în bloc; valoarea implicită a acestui parametru este 10%;
 - parametrul *PCTUSED* reprezintă procentul minim al spațiului utilizat din bloc care trebuie atins pentru a permite din nou inserarea unor liniile de date; valoarea sa implicită este 40%;
 - într-un bloc, se pot introduce date atât timp cât dimensiunea spațiului liber este mai mare decât limita fixată de parametrul *PCTFREE*; sistemul *Oracle* va considera acest bloc indisponibil pentru inserarea de noi liniile, până când procentajul spațiului utilizat coboară sub valoarea dată de *PCTUSED*.

- spațiul liber poate fi administrat automat de către sistem sau manual de către administrator;
 - pentru anumite tipuri de segmente (de date sau index), sistemul menține una sau mai multe liste de blocuri de date care au dimensiunea spațiului liber mai mare decât valoarea parametrului *PCTFREE*; acestea se numesc liste libere (*free list*); blocurile de date care compun listele libere sunt folosite pentru comenzi *INSERT*.

10.2.2. Extensii

- Extensia este o unitate logică de alocare a spațiului bazei de date, compusă dintr-o mulțime contiguă de blocuri de date (din același fișier de date).
- Una sau mai multe extensii formează un segment.
 - Inițial, segmentul are o singură extensie (*initial extent*);
 - De exemplu, la crearea unui tabel, sistemul alocă acestuia un segment cu o extensie inițială care conține un anumit număr de blocuri de date.
 - Blocurile ce corespund extensiei inițiale sunt rezervate pentru liniile tabelului.
 - Dacă spațiul de stocare alocat segmentului este complet folosit, sistemul *Oracle* alocă pentru acesta o nouă extensie (*next extent*).
- O extensie este alocată atunci când este creat sau extins un segment și este dezalocată când segmentul este șters sau trunchiat.
- Atunci când extensiile sunt eliberate, sistemul modifică fișierele de date (dacă administrarea spațiilor tabel se face local) sau reactualizează dicționarul datelor (dacă administrarea spațiilor tabel se face prin dicționar). În acest fel, dimensiunea spațiului de stocare disponibil este cunoscută în orice moment. Eliberarea unei extensii implică ștergerea datelor existente în blocurile de date alocate acesteia. Blocurile respective vor fi reutilizate pentru extensiile noi create.
- Parametrii care permit definirea dimensiunilor și a limitelor extensiilor în cadrul segmentelor sunt definiți cu ajutorul clauzei *STORAGE*.
 - Aceasta poate fi specificată în momentul creării sau modificării obiectelor bazei de date (tabele, vizualizări materializate, *cluster*-e, indecsi, partiții).
 - Clauza *STORAGE* are următoarea sintaxă:

```
STORAGE ( { INITIAL întreg [ {K | M} ]  
          | NEXT întreg [ {K | M} ]  
          | MINEXTENTS întreg
```

```

| MAXEXTENTS { întreg | UNLIMITED}
| PCTINCREASE întreg
| FREELISTS întreg
| FREELIST GROUPS întreg
| OPTIMAL [ { întreg [ {K | M} ] | NULL} ]
| BUFFER_POOL {KEEP | RECYCLE | DEFAULT} );

```

- *INITIAL* specifică dimensiunea în octeți a extensiei inițiale. Sistemul alocă spațiu pentru această extensie atunci când creează un segment.
- *NEXT* precizează dimensiunea în octeți a următoarei extensii care va fi alocată segmentului. Valoarea maximă a acesteia depinde de sistemul de operare.
- *MINEXTENTS* reprezintă numărul minim de extensii care trebuie alocat atunci când se creează un segment, iar *MAXEXTENTS* numărul total de extensii. Opțiunea *UNLIMITED* determină alocarea automată a extensiilor.
- *PCTINCREASE* specifică procentul de creștere a dimensiunii unei extensii (începând cu a treia extensie), față de dimensiunea extensiei anterioare. Valoarea minimă pentru *PCTINCREASE* este 0, ceea ce presupune că, exceptând extensia inițială, toate extensiile alocate au aceeași dimensiune.
- *FREELISTS* specifică numărul de componente ale fiecărui grup de liste libere pentru un tabel, o partitie, o grupare sau un index (valoarea minimă este 1, iar valoarea maximă depinde de dimensiunea blocurilor de date).
- *FREELIST GROUPS* precizează numărul de grupuri de liste libere pentru obiectele create.
- *OPTIMAL* precizează dimensiunea optimă în octeți pentru segmentele de revenire, opțiunea fiind relevantă doar pentru aceste tipuri de segmente. Sistemul menține dimensiunea specificată prin dezalocarea dinamică a extensiilor care nu mai sunt folosite. Opțiunea *NULL* presupune că nu se specifică o dimensiune optimă pentru segmentele de revenire. În acest caz, sistemul nu va dezaloca niciodată extensiile alocate acestor tipuri de segmente.
- Clauza *BUFFER_POOL* permite specificarea unei zone de memorie *cache* (*buffer pool*)隐含的 pentru un obiect al schemei. Toate blocurile de date alocate obiectului vor fi stocate în această zonă. Opțiunea *KEEP* permite sistemului să mențină obiectul în memorie, evitând astfel operațiile *I/O*. Opțiunea *RECYCLE* determină depunerea blocurilor de date alocate segmentului într-o zonă de memorie *RECYCLE*. Opțiunea *DEFAULT* indică o zonă *buffer pool* implicită.

10.2.3. Segmente

- Segmentul este un set de extensii care conține toate datele unei structuri logice dintr-un spațiu tabel.
 - Un segment conține cel puțin o extensie.
 - Fiecare obiect fizic stocat îi corespunde un segment.
- Deoarece nu toate obiectele unei baze de date sunt stocate fizic, există obiecte cărora nu le corespunde niciun segment (de exemplu, vizualizările).
- Segmentul folosește blocuri de date care se găsesc în același spațiu tabel.
- Sistemul permite administrarea automată a spațiului liber din interiorul segmentelor bazei de date. În acest fel, spațiul liber, respectiv cel utilizat al segmentului este monitorizat prin intermediul unor obiecte numite *bitmap*-uri.
- Baza de date *Oracle* conține diferite tipuri de segmente:
 - segmente de date (*data segment*);
 - segmente index (*index segment*);
 - segmente temporare (*temporary segment*);
 - segmente de revenire (*undo segment*).

Segmentele de date

- Sunt definite atunci când este folosită comanda de creare a unui tabel sau a unei grupări.
- Un singur segment de date este folosit pentru stocarea tuturor datelor dintr-un tabel nepartitionat care nu face parte din nici o grupare, dintr-o partitie a unui tabel partitionat sau dintr-o grupare de tabele.
- Una sau mai multe coloane ale unui tabel pot stoca obiecte de tip *LOB* (de exemplu, documente text, imagini, videoclipuri). În acest caz, *server*-ul *Oracle* stochează datele în segmente separate, numite segmente *LOB*.
- coloană a unui tabel poate fi definită de tip tablou imbricat. Valorile unei coloane de tip tablou imbricat sunt stocate în segmente diferite.
- Parametrii de stocare pentru un tabel sau o grupare, specificați prin clauza *STORAGE*, determină modul de alocare a extensiilor segmentului. Aceștia sunt importanți, deoarece pot afecta eficiența stocării și a accesului la date.
- Pentru vizualizările materializate sistemul creează segmentele de date în aceeași manieră ca și în cazul tabelelor sau grupărilor.

Segmentele index

- Sunt folosite pentru a stoca datele unui index. Fiecare index nepartitionat este conținut într-un singur segment. În cazul indecsilor partionatați, fiecărei partiții i se asociază câte un segment index.
- Sistemul creează un segment index de fiecare dată când este folosită comanda *CREATE INDEX*. În această comandă se pot specifica parametrii de stocare pentru extensiile asociate segmentului index și spațiul tabel în care este creat acesta. Spațiul tabel respectiv poate fi diferit de cel care conține segmentul de date al tabelului asupra căruia a fost creat indexul.

Segmentele temporare

- Sunt utilizate de sistem pentru analiza și execuția comenzilor *SQL* care necesită un spațiu temporar de stocare.
- Sistemul alocă în mod automat segmente temporare atunci când este necesar și le suprimă după execuția comenzii *SQL*.
- Segmentele temporare sunt alocate în majoritatea cazurilor de sortare (atunci când operația respectivă nu se poate face în memorie sau dacă folosirea indecsilor nu presupune o soluție mai eficientă).
- Comenzile *CREATE INDEX* și *SELECT* (cu opțiunile *ORDER BY*, *DISTINCT*, *GROUP BY* și operațiile *UNION*, *INTERSECT* sau *MINUS*) pot determina alocarea unui segment temporar. O singură comandă *SQL* poate necesita mai multe segmente temporare (de exemplu, o interogare ce conține simultan clauzele *DISTINCT*, *GROUP BY* și *ORDER BY*).

Segmentele de revenire

- Înregistrează valorile anterioare ale datelor care au fost modificate de către tranzacții (permanent sau nu), permitând astfel revenirea la forma inițială a acestora.
- O bază de date conține unul sau mai multe segmente de revenire. Ele sunt folosite pentru a oferi consistență la citire, a anula acțiunea tranzacțiilor sau a efectua operațiile de recuperare a bazei de date.
- Segmentele de revenire nu pot fi accesate de către utilizatorii sau administratorii bazei de date. Ele pot fi scrise și citite doar de către sistem. Atunci când o tranzacție devine activă, sunt modificate blocurile de date din segmentul de revenire asociat. Sistemul înregistrează în fișierele de reluare toate schimbările care au loc asupra blocurilor de date și informațiile pentru revenire. Astfel, în cazul eventualelor defecțiuni, *Oracle*

poate restaura automat baza de date la starea anterioară, folosind segmentele de revenire corespunzătoare tranzacțiilor care erau active în momentul defectiunii.

- În versiunile anterioare, administrarea segmentelor de revenire se realiza manual. Începând cu versiunea *Oracle9i* se permite administrarea acestora și în mod automat (*Automatic Undo Management*). Server-ul *Oracle* gestionează crearea, alocarea și optimizarea segmentelor de revenire, menținând valorile vechi ale datelor în spații tabel de revenire.

10.2.4. Spații tabel

- O bază de date este compusă dintr-o mulțime de unități logice de stocare numite spații tabel. Acestea pot fi asociate unei singure baze de date și sunt formate dintr-unul sau mai multe segmente.
- Spațiile tabel sunt utilizate pentru a grupa logic o mulțime de obiecte.
 - De exemplu, pentru a simplifica unele operații de administrare, spațiile tabel pot grupa toate obiectele unei anumite aplicații.
 - Fiecare obiect al bazei are specificat un spațiu tabel în care trebuie să fie creat. Datele care alcătuiesc obiectul sunt apoi stocate în fișierele de date alocate spațiului tabel respectiv. Un fișier de date poate fi alocat unui singur spațiu tabel.
- În orice bază de date *Oracle*, primul spațiu tabel creat este *SYSTEM*, căruia îi va fi alocat automat (în timpul creării bazei de date) primul fișier de date. Pe lângă spațiul tabel *SYSTEM*, baza de date conține și alte spații tabel (non-*SYSTEM*), pentru a stoca logic obiectele sale.

Spațiul tabel *SYSTEM*

- Conține dicționarul datelor, inclusiv unitățile de program stocate.
- Conține segmentul de revenire *SYSTEM*.
- Nu trebuie să conțină date ale utilizatorilor, deși este permis acest lucru.

Spațiile tabel non-*SYSTEM*

- Permit administrarea flexibilă a bazei de date.
- Separă segmentele de revenire, segmentele temporare, segmentele de date și segmentele index.
- Separă datele dinamice de cele statice.
- Controlează spațiul alocat pentru obiectele utilizatorilor.

- Sistemul permite atribuirea implicită a spațiilor tabel.
- De asemenea, pentru fiecare utilizator se poate aloca explicit un spațiu tabel, în care vor fi stocate toate obiectele create de acesta. Folosirea mai multor spații tabel permite flexibilitate în execuția operațiilor bazei de date.
- Spațiile tabel pot fi *online* (accesibile) sau *offline* (neaccesibile).
 - În mod normal, spațiile tabel sunt *online* pentru a permite utilizatorilor accesul la informațiile stocate în ele.
 - Spațiul tabel *SYSTEM* nu poate fi niciodată *offline*.
- Comanda *CREATE TABLESPACE* permite crearea unui spațiu tabel:

```

CREATE [UNDO] TABLESPACE nume_spațiu_tabel
    [DATAFILE specificație_fișier [clauza_autoextend]
        [, specificație_fișier [clauza_autoextend]... ] ]
    [MINIMUM EXTENT întreg [ {K | M} ] ]
    [BLOCKSIZE întreg [K] ]
    [ {LOGGING | NOLOGGING} ]
    [DEFAULT clauza_storage]
    [ {ONLINE | OFFLINE} ]
    [ {PERMANENT | TEMPORARY} ]
    [clauza_administrare_extensie]
    [clauza_administrare_segment];
  
```

- Opțiunea *UNDO* permite crearea unui spațiu tabel de revenire (*undo*). Spațiile tabel *undo* sunt rezervate sistemului, deci utilizatorii nu pot crea obiecte ale bazei de date în acestea. Dacă sunt definite mai multe spații tabel *undo*, numai unul dintre ele poate fi activ la un moment dat. Un spațiu tabel *undo* poate fi suprimit numai dacă nu este activ.
- Clauza *MINIMUM EXTENT* specifică dimensiunea minimă a extensiilor care vor fi alocate spațiului tabel. Această dimensiune se va aplica pentru toate extensiile care vor fi asociate segmentelor din spațiul tabel respectiv. Clauza nu este relevantă pentru spațiile tabel temporare administrate prin dicționarul datelor.
- Pentru specificarea dimensiunii nonstandard a blocurilor de date alocate spațiului tabel se folosește clauza *BLOCKSIZE*. Spațiile tabel temporare nu permit blocuri de date cu dimensiune nonstandard.

- Prin opțiunea *LOGGING* se specifică dacă toate schimbările ce au loc asupra tabelelor, indecsilor și partișilor care vor avea alocat spațiul tabel definit sunt înregistrate implicit în fișierele de reluare. În modul *NOLOGGING*, datele sunt modificate cu jurnalizare minimă adică, doar pentru a marca extensiile noi cu *INVALID* și pentru a înregistra modificările de la nivelul dicționarului datelor. Clauza *DEFAULT* permite specificarea parametrilor implicați de stocare pentru toate obiectele create în spațiul tabel definit.



- ❖ Valoarea unui parametru de stocare specificat la nivel de segment are prioritate față de cea precizată pentru acesta la nivelul spațiului tabel, exceptând parametrii *MINIMUM EXTENT* și *UNIFORM SIZE*.
- ❖ Dacă la nivel de segment parametru de stocare nu au fost specificați explicit, atunci aceștia vor avea valorile specificate pentru spațiul tabel în care se găsesc segmentele respective.
- ❖ Atunci când parametrii de stocare nu sunt specificați explicit pentru un spațiu tabel, *server-ul Oracle* folosește valorile sistem implice.
- ❖ Dacă parametrii de stocare sunt modificați, noile opțiuni se aplică doar pentru extensiile care nu au fost deja alocate.
- ❖ Există parametri de stocare ce nu pot fi specificați la nivel de spațiu tabel, ci numai la nivel de segment.
- Dacă se folosește opțiunea *ONLINE*, spațiul tabel devine disponibil imediat după ce a fost creat pentru toți utilizatorii care au dreptul de a-l accesa. Această opțiune este implicită. *OFFLINE* presupune că spațiul tabel nu este disponibil după creare, până când opțiunea este modificată la valoarea *ONLINE* prin comanda *ALTER TABLESPACE*.
- Clauza *PERMANENT* este specificată dacă spațiul tabel va fi folosit pentru a stoca obiecte permanente. Dacă spațiul tabel trebuie să stocheze obiecte temporare (de exemplu, segmentele folosite pentru sortări) atunci este utilizată clauza *TEMPORARY*. În acest caz nu se pot specifica clauzele *EXTENT MANAGEMENT LOCAL* sau *BLOCKSIZE*. Pentru a crea spații tabel temporare gestionate local se utilizează comanda *CREATE TEMPORARY TABLESPACE*.
- Specificarea modului de administrare a extensiilor alocate spațiului tabel se face prin clauza *_administrare_extensie*.
- Pentru spațiile tabel permanente, administrate local se poate specifica clauza *_administrare_segment*.

Aceasta are următoarea sintaxă:

```
SEGMENT SPACE MANAGEMENT {MANUAL | AUTO}
```

- Opțiunea *MANUAL* presupune că sistemul administrează spațiul liber folosind liste libere. Dacă se specifică opțiunea *AUTO*, sistemul *Oracle* va administra spațiul liber al segmentelor spațiului tabel folosind un *bitmap*.
- Spațiile tabel pot fi modificate sau eliminate. Modificarea unui spațiu tabel se face prin comanda *ALTER TABLESPACE* care permite adăugarea unor fișiere de date, transferarea fișierelor, schimbarea parametrilor de stocare, schimbarea stării din *online* în *offline* și reciproc, recuperarea spațiului tabel respectiv etc.
- Spațiile tabel pot fi suprimate chiar atunci când conțin date. Comanda prin care se suprimă un spațiu tabel este *DROP TABLESPACE*. Spațul tabel *SYSTEM* nu poate fi eliminat deoarece conține dicționarul datelor. De asemenea, nu pot fi eliminate spațiile tabel care conțin segmente active. Sistemul *Oracle9i* șterge odată cu un spațiu tabel și fișierele de date asociate acestuia.
- Informații despre spațiile tabel definite în baza de date se obțin interogând vizualizările *DBA_TABLESPACES* și *V\$TABLESPACE* din dicționarul datelor.



Relația dintre baze de date, spații tabel și fișiere de date presupune că:

- ❖ fiecare bază de date este împărțită din punct de vedere logic în unul sau mai multe spații tabel;
- ❖ unul sau mai multe fișiere de date sunt create explicit pentru fiecare spațiu tabel, cu scopul de a stoca fizic datele din structurile sale logice;
- ❖ suma mărimilor tuturor fișierelor de date asociate unui spațiu tabel reprezintă capacitatea totală de stocare a spațiului tabel;
- ❖ suma capacitaților de stocare a spațiilor tabel ale unei baze de date reprezintă capacitatea totală de stocare a bazei.
- Vizualizări din dicționarul datelor care oferă informațiile despre spații tabel, fișiere de date, segmente și extensii:
 - *DBA_TABLESPACES*;
 - *DBA_DATA_FILES*;
 - *DBA_SEGMENTS*;
 - *DBA_EXTENTS*;
 - *DBA_FREE_SPACE*.

10.2.5. Scheme de obiecte

- O schemă reprezintă mulțimea obiectelor bazei de date, aflate în posesia unui utilizator (fiecare utilizator deține o singură schemă). Numele schemei este același cu numele utilizatorului.
- Între spațiile tabel și schemele de obiecte nu există o corespondență biunivocă. Obiectele aceleiași scheme pot fi în spații tabel diferite, iar un spațiu tabel poate conține obiecte din mai multe scheme.
- Pentru a accesa un obiect din propria schemă, utilizatorul poate folosi doar numele acestuia. Pentru referirea unui obiect din schema altui utilizator, trebuie specificat atât numele obiectului, cât și schema din care face parte, prin folosirea notăției *schema.obiect*.
- Obiecte ale schemei
 - Tabelul (*table*) este unitatea fundamentală pentru stocarea datelor unei baze *Oracle*. Sistemul permite stocarea partaționată a tabelelor. Dacă un tabel este partaționat, atunci fiecare partație a să corespunde unui segment.
 - Vizualizarea (*view*) este o reprezentare logică a datelor din unul sau mai multe tabele sau vizualizări. O vizualizare poate fi privită ca o „interrogare stocată”. Vizualizarea nu stochează înregistrări, ea doar referă datele unor tabele de bază (*master table*) care, la rândul lor, pot fi tabele sau vizualizări.
 - O vizualizare materializată (*materialized view*) permite accesul indirect la datele unui tabel, prin stocarea rezultatelor unei interrogări. Diferența față de o vizualizare obișnuită, care fiind virtuală nu stochează date, este că vizualizarea materializată necesită alocarea unui spațiu de memorie pentru stocarea liniilor rezultate în urma interrogării asociate.
 - Secvența (*sequence*) este un obiect al bazei de date care permite utilizatorilor să genereze automat o listă serială de numere întregi unice. Numerele secvenței sunt independente de tabele, astfel că aceeași secvență poate fi folosită pentru mai multe tabele.
 - Unitățile de program (*program unit*) reprezintă subprograme (proceduri și funcții stocate), pachete, declanșatori și clase *Java*.
 - Sinonimele (*synonym*) sunt nume alternative pentru tabele, vizualizări, secvențe sau unități de program.

- Indexul (*index*) este o structură opțională, asociată tabelelor bazei de date, care conține adresa fizică a fiecărei linii dintr-un tabel.
- Grupările (*cluster*) sunt structuri opționale pentru stocarea datelor din tabele create cu scopul de a permite acces rapid la acestea. O grupare conține unul sau mai multe tabele.
 - Din punct de vedere logic, tabelele unei grupări sunt stocate unitar (în același segment) și, de obicei, sunt folosite împreună, deoarece au coloane comune. Aceste coloane sunt numite chei *cluster* și trebuie indexate, astfel încât liniile grupării să fie recuperate cu minimum de operații *I/O*.
 - Asemenea indecșilor, grupările nu afectează proiectarea aplicațiilor. Datele stocate într-o grupare pot fi accesate prin comenzi *SQL*, în același mod ca și cele dintr-un tabel obișnuit.
- Obiectul dimensiune (*dimension*) definește ierarhia („părinte“/„copil“) dintre perechi sau seturi de coloane. Fiecare valoare de pe un nivel „copil“ este asociată cu o valoare unică de pe nivelul „părinte“.
 - Obiectul dimensiune nu conține date, ci relații logice între tabele. Coloanele unui obiect dimensiune pot proveni din același tabel (denormalizat) sau de la mai multe tabele (în întregime sau parțial normalize).
- O legătură de baze de date (*database link*) este un obiect al unei scheme dintr-o bază care permite accesarea obiectelor din altă bază. Atunci când se lucrează cu mai multe instanțe, legăturile dintre bazele de date sunt folosite pentru a transmite date între aplicații.

10.3. Dicționarul datelor

- Una dintre cele mai importante componente ale bazei de date *Oracle* este dicționarul datelor.
- Dicționarul datelor include tabele și vizualizări *read-only* ce conțin informații despre baza de date:
 - definițiile tuturor obiectelor din schemele bazei;
 - cantitatea de spațiu alocat pentru obiectele schemelor și cantitatea de spațiu utilizat de acestea la momentul curent;
 - valorile implicite ale coloanelor;
 - constrângerile de integritate;

- numele utilizatorilor *Oracle*;
- privilegiile și *role*-urile acordate fiecărui utilizator;
- informații de auditare etc.
- Dicționarul datelor:
 - este generat automat la crearea bazei de date;
 - este reactualizat de către *server*-ul *Oracle* după fiecare comandă de definire a datelor sau de control al accesului la acestea;
 - reflectă imaginea bazei de date (structura fizică și logică) la un moment dat;
 - este deținut de către utilizatorul *SYS*;
 - se află în spațiul tabel *SYSTEM*.
- Sistemul poate accesa dicționarul datelor pentru a obține informații despre utilizatori, despre obiecte sau despre structurile de stocare. Orice utilizator poate consulta dicționarul datelor pentru a afla informații despre baza de date (documentare sau administrare). Accesul la informațiile din dicționar se face utilizând instrucțunea *SELECT* din *SQL*.
- Din punct de vedere structural, dicționarul datelor este compus din tabele de bază și vizualizări publice asupra acestora.
 - Tabelele de bază stochează informațiile asociate bazei de date, fiind primele obiecte create. În general, doar sistemul are acces direct la scrierea și citirea datelor din aceste tabele. Majoritatea datelor din tabelele de bază sunt stocate în format criptat și sunt dificil de interpretat direct de către utilizatori.
 - Dicționarul datelor conține un tabel numit *DUAL*, pe care aplicațiile sistemului sau cele create de utilizatori îl pot referi pentru a garanta un rezultat cunoscut. Acest tabel este format dintr-o singură coloană (*DUMMY*) și o singură linie care conține valoarea *X*.
 - Vizualizările decodifică informațiile stocate în tabelele de bază și le sintetizează pentru a fi disponibile utilizatorilor. Vizualizările dicționarului pot fi relative la:
 - toate obiectele din baza de date (prefix *DBA_*);
 - obiectele accesibile unui utilizator (prefix *ALL_*);
 - obiectele unui utilizator (prefix *USER_*);
 - performanțe (prefix *V\$* sau *GV\$*).
- Utilizatorii fără privilegii de administrare pot accesa doar vizualizările prefixate de *USER_* sau *ALL_*. Pentru a obține lista vizualizărilor disponibile se poate interoga vizualizarea *DICTIONARY* care are sinonimul *DICT*.

- Vizualizările prefixate de *DBA_* prezintă o imagine globală asupra bazei de date. Acestea pot fi interogate doar de utilizatorii care au privilegiul *SELECT ANY DICTIONARY*. Pentru a referi o astfel de vizualizare, trebuie utilizată notația cu punct și prefixat numele acesteia cu numele proprietarului său (*SYS*).
- Vizualizările prefixate de *ALL_* reflectă baza de date din perspectiva utilizatorului curent. Acestea oferă informații despre toate obiectele schemei la care are acces utilizatorul curent.
- Vizualizările prefixate de *USER_* conțin informații despre utilizatorul curent, obiectele create de acesta, privilegiile sistem pe care le deține, privilegiile pe care le-a acordat altor utilizatori etc. Exceptând coloana *OWNER*, vizualizările prefixate de *USER_* conțin aceleași coloane ca și celelalte vizualizări.
- În funcție de valorile sufixului (precizat între paranteze), aceste vizualizări furnizează informații despre:
 - utilizatori (*USERS*);
 - coloane specificate în constrângeri (*CONS_COLUMNS*);
 - tabele ale bazei (*TABLES*) și tabele externe (*EXTERNAL_TABLES*);
 - vizualizări (*VIEWS*) și vizualizări materializate (*MVIEWS*);
 - grupări (*CLUSTERS*) și indecsi (*INDEXES*);
 - declanșatori (*TRIGGERS*);
 - sinonime (*SYNONIMS*) și secvențe (*SEQUENCES*);
 - obiecte (*OBJECTS*);
 - biblioteci (*LIBRARIES*);
 - politici de securitate (*POLICIES*);
 - privilegii obiect asupra tabelelor (*TAB_PRIVS*);
 - tipuri obiect (*TYPES*) și colecție (*COLL_TYPES*) etc.
- Vizualizarea *DBA_OBJECTS* este utilă pentru obținerea de informații referitoare la toate categoriile de obiecte ale unei scheme. Coloanele acestei vizualizări sunt următoarele:
 - *OWNER* (proprietarul obiectului);
 - *OBJECT_NAME* (numele obiectului);
 - *SUBOBJECT_NAME* (numele subobiectelor);
 - *OBJECT_ID* (identificatorul obiectului);
 - *DATA_OBJECT_ID* (identificatorul segmentului ce conține obiectul);
 - *OBJECT_TYPE* (tipul obiectului);

- *CREATED* (data creării obiectului);
- *LAST_DDL_TIME* (data ultimei modificări structurale a obiectului);
- *TIMESTAMP* (specificarea formatului datei);
- *STATUS* (starea obiectului);
- *TEMPORARY* (precizarea că obiectul este temporar);
- *GENERATED* (specificarea modului de generare al obiectului);
- *SECONDARY* (precizarea că obiectul este secundar, creat prin metoda *ODCIIndexCreate* a lui *Oracle Data Cartridge*).

Exemplul 10.1

```
-- proprietarul, numele și tipul obiectelor din baza de date.
SELECT OWNER, OBJECT_NAME, OBJECT_TYPE
FROM   SYS.DBA_OBJECTS;

-- proprietarul, numele și tipul tuturor obiectelor accesibile
utilizatorului curent.
SELECT OWNER, OBJECT_NAME, OBJECT_TYPE
FROM   ALL_OBJECTS;

-- numele și tipul obiectelor create de utilizatorul curent.
SELECT OBJECT_NAME, OBJECT_TYPE
FROM   USER_OBJECTS;
```

- Vizualizările prefixate de *V\$* oferă informații despre performanțele bazei de date.
 - Se numesc vizualizări fixe, deoarece administratorii bazei nu le pot modifica sau suprima.
 - Fiecare instanță are asociat un set propriu de vizualizări prefixate de *V\$*.
 - Fiind virtuale, acestea există în memorie doar în timp ce baza de date este pornită.
 - Prin interogarea acestor vizualizări se obțin informații despre:
 - sesiunile curente (*V\$SESSION*) și procesele active (*V\$PROCESS*);
 - obiectele care sunt blocate la un moment dat și sesiunile care le accesează (*V\$ACCES*);
 - tranzacțiile active (*V\$TRANSACTION*);
 - procesele *background* (*V\$BGPROCESS*);
 - fișierele de reluare care trebuie arhivate (*V\$ARCHIVE*);
 - numele și numărul tuturor spațiilor tabel asociate fișierelor de control (*V\$TABLESPACE*);
 - numele și starea fișierelor de control asociate instanței bazei de date (*V\$CONTROLFILE*) etc.

Bibliografie

1. *Programare avansată în Oracle9i*, I. Popescu, A. Alecu, L. Velcescu, G. Florea (Mihai), Ed. Tehnică (2004)
2. *Oracle Database PL/SQL Language Reference 11g Release 2*, Oracle Online Documentation (2012)
3. *Oracle Database SQL Language Reference 11g Release 2*, Oracle Online Documentation (2012)
4. *Oracle Database 11g: PL/SQL Fundamentals, Student Guide*, Oracle University (2009)

CUPRINS

11. Arhitectura internă a sistemului <i>Oracle</i>	2
11.1. Nivelul fișiere	2
11.2. Arhitectura proceselor	3
11.2.1. Procese <i>user</i>	3
11.2.2. Procese <i>Oracle</i>	4
11.3. Arhitectura memoriei.....	9
11.3.1. Zona globală sistem	9
11.3.2. Zona globală program	11
Bibliografie	14

11. Arhitectura internă a sistemului *Oracle*

- Din punct de vedere al arhitecturii interne, sistemul *Oracle* este structurat pe trei niveluri:
 - nivelul procese
 - corespunde diverselor procese de sistem care asigură gestiunea datelor;
 - nivelul memorie
 - este compus dintr-o mulțime de zone tampon alocate pentru a stoca date și anumite informații de control;
 - nivelul fișiere
 - corespunde structurii bazei de date și modului în care sunt stocate datele.

11.1. Nivelul fișiere

- Server-ul *Oracle* poate folosi fișiere care nu fac parte din baza de date.
 - Acestea permit configurarea instanței, autentificarea utilizatorilor și recuperarea bazei de date.
- Fișierele de parole (*password file*):
 - sunt fișiere binare folosite pentru autentificarea utilizatorilor bazei de date;
 - sunt localizate în directorul *DATABASE*, iar numele lor depinde de identificatorul *SID* al instanței bazei de date (*psdSID.ora*);
 - pot fi folosite atât pentru conexiunile locale la o bază de date, cât și pentru conexiunile la distanță;
 - în mod automat, sunt ascunse (*hidden*);
 - pot deveni vizibile prin intermediul comenziilor sau utilitarelor specifice fiecărui sistem de operare.
- Fișierul parametrilor de inițializare (*parameter file*)
 - reprezintă principalul mijloc de configurare a sistemului și este utilizat pentru definirea caracteristicilor unei instanțe *Oracle*;
 - conține o colecție de valori ale unor parametri care controlează sau modifică anumite aspecte din modul de funcționare al bazei de date sau al instanței;
 - în versiunile mai vechi, sistemul stoca valorile parametrilor de inițializare doar într-un fișier de tip text (*PFILE*); începând cu versiunea *Oracle9i*, acestea pot fi menținute în fișierul de parametri *server* (*SPFILE*), care este construit prin

comanda *CREATE SPFILE*, pe baza fișierului *PFILE*; sistemul asigură o interfață pentru vizualizarea și modificarea parametrilor de inițializare.

- Fișierele *archived redo log*:
 - sunt copii *offline* ale fișierelor de reluare, folosite pentru recuperarea bazei de date în cazul defecțiunilor *hardware*.
- Fișierele istorice (*trace file* și *alert file*):
 - conțin toate mesajele, erorile și evenimentele importante;
 - dacă un proces de tip *server* sau *background* detectează o eroare internă, atunci acesta va scrie informații despre eroarea respectivă în fișierul *trace* asociat; în general, numele fișierelor *trace* asociate cu procesele *background* conțin numele proceselor care le-au generat;
 - orice bază de date conține câte un fișier *alert* asociat fiecărei instanțe;
 - în loc să afișeze informațiile pe consolă, sistemul *Oracle* înregistrează cronologic în acest fișier toate mesajele și erorile care apar.

11.2. Arhitectura proceselor

- Pentru a accesa o instanță a unei baze de date *Oracle*, se execută:
 - aplicație sau un utilitar *Oracle*, prin intermediul cărora se lansează comenzi *SQL* asupra bazei de date;
 - un cod *Oracle server*, cu ajutorul căruia sunt interpretate și procesate comenziile *SQL*.
- Procesele:
 - execută aplicațiile și comenziile interne;
 - sunt mecanisme ale sistemului care permit executarea unor operații de calcul sau operații *I/O*;
 - au alocate zone private de memorie individuale;
 - sunt de două tipuri:
 - procese *user*, care execută aplicațiile;
 - procese *Oracle* (procese *server* și procese *background*), care asigură gestiunea informațiilor dintr-o bază de date.

11.2.1. Procese *user*

- Sistemul creează un proces *user* pentru a executa codul unei aplicații program sau în urma lansării unui utilitar *Oracle*.

- Procesul se execută pe stația *client* (stația de pe care utilizatorul s-a conectat la *server-ul Oracle*).
 - Procesul începe și se termină odată cu aplicația utilizatorului respectiv.
 - Procesul *user* nu interacționează în mod direct cu *server-ul Oracle*.
 - Aceasta generează mesaje printr-un program interfață (*User Program Interface*) care creează o sesiune și pornește un proces *server*.
 - Programul interfață prelucrează cererile, captează și întoarce erorile, execută conversii și decodificări de date, realizează analize sintactice.
-  ♦ Trebuie făcută distincția dintre conexiune și sesiune.
- Conexiunea este o cale de comunicare între un proces *user* și o instanță *Oracle*.
 - Sesiunea este o conexiune la o instanță *Oracle* realizată printr-un proces *user*.

11.2.2. Procese *Oracle*

- Procesele *Oracle* execută instrucțiunile interne ale *server-ului Oracle*.
 - Sunt invocate de alte procese pentru a îndeplini anumite operații în favoarea acestora.
 - Există două tipuri de procese *Oracle*:
 - procese *server* (*server process*);
 - procese de fundal (*background process*).

Procesul *server*

- Procesul *server*:
 - interacționează cu procesele *user* și comunică în mod direct cu *server-ul Oracle* pentru a transmite cererile acestora;
 - este pornit atunci când procesul *user* inițiază o sesiune;
 - comunică cu *server-ul Oracle* prin *Oracle Program Interface (OPI)*.
- În cazul configurației *server* dedicate, un proces *server* poate gestiona cererile unui singur proces *user*. Sistemul *Oracle* poate fi configurat astfel încât un singur proces *server* să deservească unul sau mai multe procese *user*. O configurație *server* partajată permite mai multor procese *user* să împartă un număr mic de procese *server*, minimizând numărul de procese *server* și maximizând folosirea resurselor sistem.

- Există sisteme în care procesele *user* și procesele *server* sunt separate și sisteme în care acestea pot fi integrate într-un singur proces. Dacă un sistem folosește un *server* partajat sau dacă procesele *user*, respectiv *server* rulează pe stații diferite, atunci procesele *user* și cele *server* trebuie separate. Sistemele *client/server* separă procesele *user* de cele *server* și le execută pe stații diferite.
-  ❖ *Server-ul Oracle* este un sistem de administrare a bazelor de date relaționale orientate pe obiecte și constă din:
- o instanță *Oracle*;
 - o bază de date *Oracle*.

Instanța *Oracle*

- Conține dintr-o structură de memorie numită *SGA* (*System Global Area*) și procese *background*.
- Pentru a putea accesa baza de date, trebuie pornită o instanță.
- Instanța poate fi asociată unei singure baze de date.
- Atunci când baza de date este pornită, se localizează structura *SGA* și sunt lansate procesele *background* (*DBWn*, *LGWR*, *CKPT*, *SMON*, *PMON*, *Dnnn*, *ARCn*, *RECO*, *LMS*, *QMNn* etc.).
- Dacă unul dintre aceste procese se termină anormal, atunci instanța se oprește.

Procesele *background*

- Sunt folosite pentru a îmbunătăți performanțele unui sistem multiprocesor, în prezența mai multor utilizatori.
- *Server-ul Oracle* creează câte un set de procese *background* pentru fiecare instanță.
 - Acestea reunesc funcțiile executate pentru fiecare proces *user*.
 - De asemenea, procesele *background* execută operațiile *I/O* asincrone și monitorizează alte procese *Oracle*.
 - Procesele *background* *DBWn*, *LGWR*, *CKPT*, *SMON* și *PMON* sunt obligatorii pentru funcționarea sistemului *Oracle*.
- Tipuri de procese *background*
 - *Database Writer* (*DBWn*)
 - Scrie blocurile modificate, din zonele de memorie tampon (*database buffer cache*) în fișierele de date. Această operație este numită *checkpoint* și determină punctul din care începe recuperarea instanței.

- Menține numărul necesar de *buffer*-e libere din *database buffer cache*. Atunci când o zonă tampon din *database buffer cache* se modifică, aceasta este marcată *dirty*. Procesul *DBWn* scrie pe disc zonele tampon *dirty* dacă:
 - un proces *server* are nevoie și nu găsește un *buffer* disponibil;
 - numărul de *buffer*-e *dirty* depășește valoarea admisă;
 - se realizează un *checkpoint* al bazei de date;
 - se elimină sau se trunchiază un tabel;
 - un spațiu tabel devine *offline* sau *read only* etc.
- Nu trebuie să scrie blocurile modificate de fiecare dată când o tranzacție devine permanentă (*commit*). De obicei, *DBWn* se activează atunci când mai multe date trebuie incluse în SGA și sunt prea puține zone tampon libere. Datele care nu au fost folosite recent sunt scrise primele în fișierele de date.
- Deși un singur proces *database writer* (*DBW0*) este suficient pentru majoritatea sistemelor, se pot configura procese adiționale (de la *DBW1* până la *DBW9*) cu scopul de a obține performanțe mărite la scriere. Parametrul *DB_WRITER_PROCESSES* specifică numărul de procese *DBWn*.

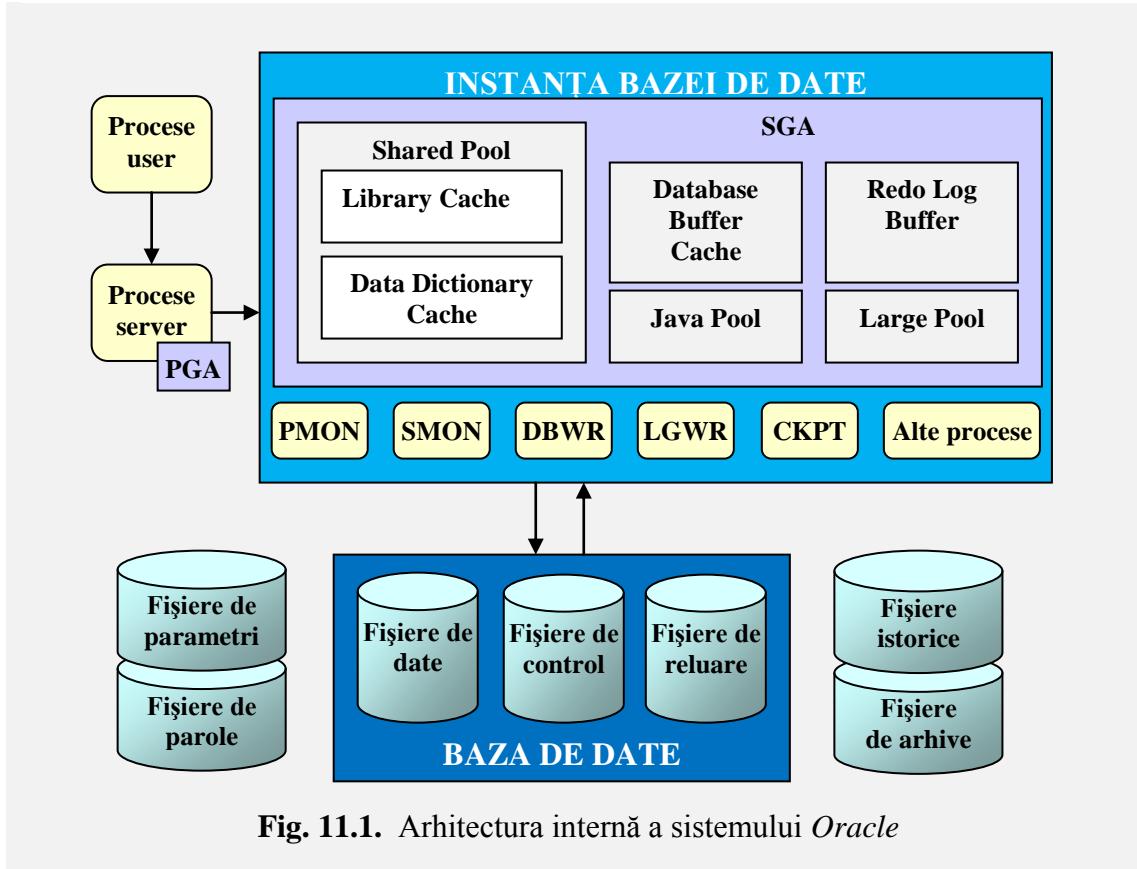


Fig. 11.1. Arhitectura internă a sistemului *Oracle*

- *Log Writer (LGWR)*

- Scrie zonelor tampon de reluare (*redo log buffer*) în fișierele de reluare (*redo log file*):
 - atunci când o tranzacție este permanentizată;
 - înainte ca procesul *DBWn* să scrie blocurile modificate din zonele de memorie tampon în fișierele de date;
 - la fiecare 3 secunde etc.
- Zonele tampon de reluare se găsesc în memoria *SGA* și prin urmare, acest proces înregistrează pe disc modificările care au loc în memoria partajată. Dacă baza de date are fișiere de reluare multiplexate, atunci *LGWR* scrie zonele tampon de reluare într-un grup de astfel de fișiere.

- *Checkpoint (CKPT)*

- Anunță procesul *DBWn* că va avea loc o operație *checkpoint* și modifică *header*-ele fișierelor de date și de control, pentru a înregistra detalii despre aceasta.
- Deoarece procesul *DBWn* scrie blocurile modificate în fișierele de date conform algoritmului *LRU* (*Least Recently Used*), un bloc de date care este folosit frecvent poate să nu fie scris niciodată pe disc dacă nu se realizează o operație *checkpoint*.
- Procesele *CKPT* pot iniția operații *checkpoint* din următoarele motive:
 - pentru a asigura că blocurile de date modificate din memorie sunt scrise pe disc în mod regulat, astfel încât datele să nu se piardă în cazul întreruperilor la nivelul sistemului sau al bazei de date;
 - pentru a reduce timpul necesar recuperării instanței (în acest caz, este necesară doar procesarea fișierelor de reluare);
 - pentru a asigura că toate datele permanentizate sunt scrise în fișierele de date în timpul procesului de închidere al bazei de date.

- *System Monitor (SMON)*

- Inițiază procesul de recuperare a datelor la pornirea unei instanțe care a eșuat anterior.
- Procesul de recuperare al unei instanțe se realizează în mai mulți pași.
 - 1) Sunt recuperate datele care nu au fost înregistrate în fișierele de date, dar care au fost înregistrate în fișierele de reluare. În timpul acestei operații, procesul *SMON* citește fișierele de reluare și modifică blocurilor de date în

conformitate cu acestea. Deoarece tranzacțiile permanentizate au fost scrise în fișierele de reluare, procesul recuperează complet aceste tranzacții.

2) Se deschide baza de date, astfel încât utilizatorii să se poată conecta. Toate datele care nu sunt blocate de tranzacțiile nerecuperate devin imediat disponibile.

3) Se anulează (*rollback*) tranzacțiile nepermanentizate.

- De asemenea, procesul *SMON* îndeplinește și funcții de întreținere a spațiului. Acesta reunește zonele adiacente de spații libere din fișierele de date și asigură eliberarea segmentelor temporare care nu mai sunt utilizate.

- *Dispatcher (Dnnn)*

- Este un proces *background* opțional, prezent atunci când este folosită o configurație *server* partajată.
- Pentru fiecare protocol de comunicare este creat cel puțin un proces dispecer.
- Fiecare proces dispecer este responsabil cu direcționarea cererilor venite din partea unor procese *user*, cu validarea proceselor *server* și transmiterea răspunsului către procesele *user*.

- *Process Monitor (PMON)*

- Asigură recuperarea proceselor *user*, dacă acestea eșuează. În acest sens, procesul *PMON* este responsabil pentru anularea tranzacției curente, eliberarea memoriei *cache* și a resurselor utilizate de procesul respectiv, rezolvarea blocărilor curente intervenite la nivel de tabel sau linie.
- De asemenea, *PMON* verifică procesele *server* și procesele dispecer pe care le repornește dacă acestea eșuează.

- *Recoverer (RECO)*

- Este folosit pentru a rezolva problema tranzacțiilor distribuite dintr-o rețea și căderile sistemului unei baze de date distribuite.
- Procesul *RECO* se conectează automat la bazele de date distante ale căror tranzacții distribuite se derulează anormal și execută operații de permanentizare sau de anulare pe porțiuni locale din tranzacțiile distribuite în așteptare.
- Dacă conexiunea eșuează, atunci procesul încearcă, la anumite intervale de timp, să stabilească o nouă legătură cu bazele de date implicate.

- *Archiver (ARCn)*

- Produce copia arhivată a fișierelor de reluare, înregistrând toate modificările făcute în baza de date.

- Deși pentru cele mai multe sisteme este suficient un singur proces *archiver* (*ARC0*), se pot specifica mai multe astfel de procese, prin intermediul parametrului de inițializare *LOG_ARCHIVE_MAX_PROCESSES*.
- *Lock Manager Server (LMS)*
 - Gestioneză inter-blocările dintre instanțele bazei de date în *Oracle Real Application Clusters*.
- *Queue Monitor (QMNn)*
 - Este un proces *background* opțional care monitorizează mesajele din lista creată prin *Oracle Advanced Queuing*.

11.3. Arhitectura memoriei

- Toate structurile de memorie se găsesc în memoria centrală. Sistemul creează și folosește structurile de memorie pentru a depozita codul programelor executate, datele necesare în timpul execuției acestora, datele folosite în comun de mai multe procese *Oracle*, informațiile referitoare la sesiunile curente etc.
- Din punct de vedere structural, memoria este compusă din:
 - zona globală sistem (*SGA - System Global Area*);
 - zona globală program (*PGA - Program Global Area*).

11.3.1. Zona globală sistem

- Este un grup de structuri partajate de memorie care conțin date și informații de control relative la o instanță.
- Este alocată atunci când se pornește instanța și este eliberată în momentul opririi acesteia.
 - Fiecare instanță are propria sa *SGA*.
 - Datele conținute în *SGA* sunt folosite în comun de către utilizatorii conectați la instanță.
 - O parte din *SGA* este folosită pentru stocarea informațiilor despre starea bazei de date și a instanței. Informațiile sunt accesate de către procesele *background*. Această zonă este numită *SGA fixă* și nu poate conține date ale utilizatorilor.
- Trebuie să fie dimensionată astfel încât să permită stocarea cât mai multor date în memorie și să reducă numărul de operații *I/O*.

- Începând cu versiunea *Oracle9i*, sistemul poate configura în mod dinamic zona *SGA* în timp ce instanța este pornită (*SGA* dinamic).
- Informațiile conținute în *SGA* sunt repartizate în diferite zone (care sunt alocate la pornirea instanței):
 - *database buffer cache*,
 - *redo log buffer*,
 - *shared pool*,
 - *large pool*,
 - *Java pool*.

Database buffer cache

- Este o mulțime de zone tampon care stochează blocurile de date (modificate sau nu) folosite cel mai recent.
- Aceste zone sunt organizate în două structuri:
 - listă de blocuri scrise (*write list*)
 - este formată din *buffer*-ele *dirty*, care conțin datele modificate, dar nu scrise pe disc;
 - listă de blocuri care au fost accesate recent (*least recently used list*).
 - este formată din *buffer*-e libere, *buffer*-e din memoria *cache* și *buffer*-e *dirty* care nu au fost încă mutate în lista de blocuri scrise.
- Atunci când este executată o cerere, procesul *server* caută blocurile de care are nevoie în *database buffer cache*. Dacă nu găsește un bloc, procesul *server* îl citește din fișierul de date și plasează o copie a acestuia în *database buffer cache*. Întrucât datele cel mai frecvent folosite sunt păstrate în memoria *cache*, se micșorează numărul de operații *I/O* și se îmbunătățesc performanțele bazei de date.

***Redo log buffer* (zona tampon de reluare)**

- Este un *buffer* circular care conține informațiile referitoare la modificările blocurilor de date ale bazei.
- Aceste informații pot fi necesare atât pentru recuperarea bazei de date, cât și pentru întoarcerea la forma anterioară a modificărilor realizate prin operații de tip *LMD* sau *LLD*.
- Conținutul *buffer*-ului este scris de procesul *LGWR* în fișierul de reluare.

Shared pool

- Este o porțiune din *SGA* formată din trei zone partajate de memorie:
 - *library cache*
 - conține informații despre cele mai recente comenzi *SQL* (text, versiune compilată, plan de execuție) și unități de program *PL/SQL*;
 - o cerere executată de mai multe ori poate utiliza versiunea compilată și planul de execuție deja existent de la rularea anterioară;
 - *data dictionary cache*
 - conține cele mai recente informații accesate din dicționarul datelor;
 - această zonă de memorie poartă și denumirea de *row cache*, deoarece datele sunt stocate sub formă de linii (nu sub formă de *buffer*-e care conțin blocuri întregi de date);
 - *buffer*-e pentru mesaje relative la execuțiile paralele și la structurile de control.

Large pool

- Este o zonă opțională *SGA*, configurată doar pentru *server*-ele partajate.

Java pool

- Este o zonă de memorie opțională, necesară atunci când este instalat și utilizat limbajul *Java*.

11.3.2. Zona globală program

- *PGA* este o zonă de memorie care conține date și informații de control relative la un singur proces *Oracle*.
 - ❖ Spre deosebire de *SGA*, care este utilizată de mai multe proceze (zonă de memorie partajată), *PGA* este folosită numai de unul singur (zonă de memorie nepartajată).
 - ❖ Zona globală program este alocată la crearea procesului și eliberată la terminarea acestuia.
- Conținutul zonei de memorie *PGA* depinde de modul de configurare a sistemului (*server* dedicat sau partajat), singurul care poate accesa această zonă.



❖ Spre deosebire de *SGA*, care este utilizată de mai multe proceze (zonă de memorie partajată), *PGA* este folosită numai de unul singur (zonă de memorie nepartajată).

❖ Zona globală program este alocată la crearea procesului și eliberată la terminarea acestuia.

• Conținutul zonei de memorie *PGA* depinde de modul de configurare a sistemului (*server* dedicat sau partajat), singurul care poate accesa această zonă.

- În general, zona globală program este formată din
 - zonă privată *SQL*;
 - conține date (de exemplu, informații de legătură) și structuri de memorie necesare rulării comenziilor;
 - pentru fiecare comandă *SQL* inițiată de o sesiune se asociază o zonă privată *SQL*;
 - în cazul *server*-elor partajate, zona privată *SQL* este menținută în *SGA*;
 - administrarea zonelor private *SQL* este responsabilitatea procesului *user*;
 - numărul de zone private *SQL* pe care le poate aloca procesul *user* este limitat de valoarea parametrului de initializare *OPEN_CURSORS*;
 - zonă de memorie alocată sesiunii;
 - zone de lucru *SQL*.

Scenariu

- Următoarea succesiune de pași ilustrează o configurare *Oracle* în care procesul *user* și procesul *server* asociat se derulează pe mașini diferite.
 - 1) Se pornește o instanță pe stația care găzduiește sistemul *Oracle* (cunoscută sub denumirea de gazdă sau *server* de baze de date).
 - 2) O stație de lucru *client* rulează o aplicație care determină crearea unui proces *user*. Aplicația *client* încearcă să stabilească o conexiune cu *server*-ul de baze de date.
 - 3) *Server*-ul detectează cererea de conectare a aplicației și creează un proces *server* dedicat procesului *user*. Procesului *server* i se alocă o zonă de memorie *PGA*.
 - 4) Utilizatorul execută o comandă *SQL* și apoi instrucțiunea *COMMIT* pentru a permanentiza schimbările și a încheia tranzacția.
 - 5) Procesul *server* primește comanda și verifică dacă în *shared pool* (*library cache*) există o zonă partajată *SQL* care conține comenzi similare. Dacă este găsită o astfel de zonă, procesul *server* verifică privilegiile de acces ale utilizatorului la datele interogate și folosește zona respectivă pentru procesarea comenzi. În caz contrar, pentru comandă este alocată o nouă zonă partajată *SQL*, unde aceasta va fi analizată și procesată.
 - 6) Procesul *server* extrage datelor necesare cererii. Dacă datele necesare au fost recent folosite, atunci procesul *server* le va găsi și extrage din *database buffer cache*. În caz contrar, datele vor fi extrase din fișierele de date și o copie a acestora va fi depusă în *database buffer cache*.

- 7) Procesul *server* modifică datele din *database buffer cache*. Informațiile referitoare la modificările blocurilor de date sunt înregistrate în *redo log buffer*. Deoarece tranzacția a fost permanentizată, procesul *LGWR* o înregistrează imediat într-un fișier de reluare. La un moment dat procesul *CKPT* anunță procesul *DBWn* că va avea loc o operație *checkpoint* și modifică *header*-ele fișierelor de date și de control, pentru a înregistra detalii despre aceasta. *DBWn* scrie blocurile de date modificate din *database buffer cache* în fișierele de date.
- 8) Dacă tranzacția s-a derulat cu succes, procesul *server* trimite un mesaj corespunzător către aplicație. Altfel, este transmis un mesaj de eroare.
- Pe întregul parcurs al procedurii, rulează și alte procese *background*, care pot interveni în condiții excepționale. În plus, *server*-ul bazei de date administrează și alte tranzacții, prevenind inter-blocarea celor care necesită aceleași date.

Bibliografie

1. *Programare avansată în Oracle9i*, I. Popescu, A. Alecu, L. Velcescu, G. Florea (Mihai), Ed. Tehnică (2004)
2. *Oracle Database PL/SQL Language Reference 11g Release 2*, Oracle Online Documentation (2012)
3. *Oracle Database SQL Language Reference 11g Release 2*, Oracle Online Documentation (2012)
4. *Oracle Database 11g: PL/SQL Fundamentals, Student Guide*, Oracle University (2009)

CUPRINS

12. Gestiunea bazei de date	2
12.1. Crearea unei baze de date	2
12.2. Pornirea unei baze de date	8
12.3. Închiderea unei baze de date.....	9
12.4. Starea de repaus a unei baze de date.....	11
Bibliografie	12

12. Gestiunea bazei de date

- O bază de date *Oracle* este disponibilă utilizatorilor din momentul deschiderii și până la închiderea ei.
- Un utilizator *Oracle* este o persoană care posedă un cont înregistrat în *server-ul Oracle* și pentru care au fost acordate privilegii corespunzătoare de acces la date.
- Fiecare bază de date are un administrator. Acesta este un utilizator cu privilegii speciale care trebuie:
 - să evalueze modul de folosire a resurselor mașinii pe care va fi instalat sistemul *Oracle* și aplicațiile sale (discuri disponibile pentru sistem și baze de date, *driver-e* dedicate, memoria necesară pentru instanțe);
 - să instaleze sistemul;
 - să creeze planul bazei de date, adică structurile logice de stocare, modelul general al bazei și strategiile de restaurare a acesteia (planificarea modului în care structurile logice de stocare ale bazei de date vor afecta funcționarea mașinii pe care rulează sistemul, performanțele bazei în timpul operațiilor de acces la date și eficiența procedurilor de recuperare și restaurare a datelor);
 - să creeze și să deschidă baza de date;
 - să asigure strategii de recuperare a bazei de date în cazul pierderilor;
 - să înregistreze utilizatorii bazei de date și să definească domenii de securitate pentru aceștia;
 - să controleze și să monitorizeze accesul utilizatorilor la baza de date;
 - să optimizeze performanțele bazei de date;
 - să modifice structura bazei, la cererea dezvoltatorilor de aplicații etc.
- Administratorul bazei de date sau utilizatorii posesori ai unui privilegiu *DBA* (*Database Administrator*) pot să pornească sau să oprească baza de date.

12.1. Crearea unei baze de date

- Crearea unei baze de date constă în definirea și implementarea celor două tipuri de structuri (logică și fizică) ce compun baza.
- O bază de date poate fi creată nu numai în momentul instalării sistemului, ci și ulterior.

- Modalitățile de creare a unei baze de date sunt următoarele:
 - în mod automat, la instalarea lui *Oracle Database Server*, utilizând *Oracle Universal Installer*;
 - folosind comanda *CREATE DATABASE* din *SQL*;
 - utilizând instrumentul *Oracle Database Configuration Assistant*;
 - în timpul migrării de la o versiune anterioară a bazei deja existente, folosind *Oracle Data Migration Assistant* etc.
- În continuare sunt prezentate succint pașii care trebuie urmați atunci când baza de date este creată manual.

Pas 1

- Se alege un identificator unic (*SID*) al instanței.
 - În felul acesta, sunt evitate confuziile cu instanțele care vor fi create ulterior și vor rula concurent cu aceasta.
 - Comanda prin care se setează identificatorul instanței este următoarea:
`%identificator_instanță ORACLE_SID nume_bază;`

Pas 2

- Se stabilesc metodele de autentificare pentru administratorul bazei de date.

Pas 3

- Se creează fișierul parametrilor de inițializare.
 - Acest fișier este necesar pentru definirea caracteristicilor instanței.
 - Denumirea și locația fișierului depinde de platforma pe care rezidă *server-ul Oracle*.
 - De exemplu, în mediul *WINDOWS*, denumirea fișierului parametrilor de inițializare este *init\$ORACLE_SID.ora* și este localizat în directorul *\$ORACLE_HOME\database*.

Pas 4

- Se realizează conectarea la o instanță.
 - De exemplu, aceasta presupune pornirea utilitarului *SQL*Plus* și conectarea la instanța *Oracle* ca *SYSDBA*:
`CONNECT SYS/parola AS SYSDBA;`

Pas 5

- Se pornește instanța fără a monta baza de date.
 - În acest moment este definită zona de memorie *SGA* și sunt pornite procesele *background* necesare creării bazei de date.
 - Comanda are următoarea sintaxă:

```
STARTUP NOMOUNT;
```

Pas 6

- Se selectează numele global al bazei de date, care reprezintă numele și domeniul acesteia.
 - Numele bazei de date este precizat prin intermediul parametrului de inițializare *DB_NAME*.
 - Valoarea acestui parametru trebuie să fie aceeași cu cea a parametrului *SID*.
- Domeniul bazei de date reprezintă locația acesteia în structura rețelei, fiind specificat prin intermediul parametrului de inițializare *DB_DOMAIN*.

Pas 7

- Se creează efectiv baza de date.
 - Aceasta presupune definirea fișierelor de date, de control și de reluare, a spațiului tabel *SYSTEM* și a segmentului de revenire *SYSTEM*, a dicționarului datelor, alegerea setului de caractere pentru stocarea datelor de tip text în baza de date, setarea zonei de timp (*timezone*) pentru bază.
 - Comanda are următoarea sintaxă:

```
CREATE DATABASE [nume_bază]
[CONTROLFILE REUSE]
[LOGFILE GROUP întreg specificație_fișier
[, specificație_fișier...]
[, GROUP întreg specificație_fișier
[, specificație_fișier...]]]
[MAXLOGFILES întreg]
[MAXLOGMEMBERS întreg]
[MAXLOGHISTORY întreg]
[MAXDATAFILES întreg]
[MAXINSTANCES întreg]
[ {ARCHIVELOG | NOARCHIVELOG} ]
[CHARACTER SET set_caractere]
```

```
[NATIONAL CHARACTER SET set_caractere]
[DATAFILE specificație_fișier [clauza_autoextend]
[, specificație_fișier [clauza_autoextend]... ] ]
[specificație_spățiu_tabel_implicit]
[specificație_spățiu_tabel_de_revenire]
[SET TIME_ZONE = '{ [+ | -] hh : mi | timp_regiune}' ];
```

- Numele bazei de date poate avea lungimea de cel mult 8 octeți și conține doar caractere *ASCII*. Sistemul scrie acest nume în fișierul de control. Atunci când numele bazei nu este specificat, se folosește parametrul de inițializare *DB_NAME*.
- Clauza *CONTROLFILE REUSE* permite reutilizarea fișierelor de control identificate prin parametrul de inițializare *CONTROL_FILES*. În general, clauza este folosită doar dacă baza de date este creată din nou.
- Specificarea fișierelor de reluare se realizează prin clauza *LOGFILE GROUP*. Fiecare fișier de reluare declarat prin *specificație_fișier* își asociază un grup corespunzător, care poate conține unul sau mai mulți membri (copii ale fișierului de reluare respectiv). O bază de date trebuie să conțină cel puțin două grupuri de fișiere de reluare.
- Clauza *MAXLOGFILES* precizează numărul maxim de grupuri de fișiere de reluare care pot fi create pentru baza de date. Valorile implicate, minime și maxime ale acestui parametru depind de sistemul de operare.
- Prin clauza *MAXLOGMEMBERS* se specifică numărul maxim de membri ai unui grup.
- Parametrul *MAXDATAFILES* precizează în fișierul de control, dimensiunea inițială (la momentul creării bazei de date sau a fișierului de control) a zonei alocate pentru fișierele de date.
- Parametrul *MAXINSTANCES* specifică numărul maxim de instanțe simultane corespunzătoare aceleiași baze de date (care este montată și deschisă). Valoarea minimă a acestui parametru este 1, iar valorile maxime și implicate depind de sistemul de operare pe care rulează *Oracle*.
- Atunci când se dorește menținerea unor copii *offline* ale fișierelor de reluare, înainte ca acestea să fie reutilizate, se folosește opțiunea *ARCHIVELOG*. În caz contrar, se precizează opțiunea *NOARCHIVELOG*.

- Parametrul *MAXLOGHISTORY* este util doar dacă baza de date este folosită cu *Real Application Clusters*, în mod *ARCHIVELOG*. Acest parametru specifică numărul maxim de fișiere de reluare arhive.
- Prin clauza *CHARACTER SET* se precizează setul de caractere *Unicode* folosit de baza de date (*AL32UFT8*, *UTF8*, *UTFE* etc.).
- Clauza *NATIONAL CHARACTER SET* precizează setul de caractere *Unicode* (*AL16UTF16*, *UTF8* etc.) folosit pentru a stoca datele din coloanele de tip *NCHAR*, *NCLOB* sau *NVARCHAR2*.
- Clauza *DATAFILE* specifică fișierelor de date din spațiul tabel *SYSTEM*.
- Clauza *SET TIME_ZONE* stabilește zona de timp pentru baza de date. Dacă această clauză este omisă, sistemul *Oracle* încearcă să preia zona de timp a sistemului de operare gazdă.
- Intervalul folosit pentru *hh:mi* este de la -12:00 la +14:00. Specificațiile posibile pentru *temp_regiune* se găsesc în coloana *TZNAME* a vizualizării *V\$TIMEZONE_NAMES* din dicționarul datelor.

Exemplul 12.1

```
CREATE DATABASE baza_depozit
CONTROLFILE REUSE
LOGFILE
    GROUP 1 ('C:/baza_depozit/redolog1.log',
              'D:/baza_depozit/redolog1.log') SIZE 25K
    GROUP 2 ('C:/baza_depozit/redolog2.log',
              'D:/baza_depozit/redolog2.log') SIZE 25K
MAXLOGFILES 6
MAXLOGMEMBERS 2
MAXDATAFILES 50
MAXINSTANCES 1
CHARACTER SET US7ASCII
DATAFILE 'C:/baza_depozit/datafile1.dbf' AUTOEXTEND ON
UNDO TABLESPACE undo_tbs
DEFAULT TEMPORARY TABLESPACE temp_tbs;
```

Pas 8

- Se creează spații tabel adiționale pentru utilizatori, astfel încât baza de date să devină funcțională.

Exemplul 12.2

Se creează spațiul tabel *users_tbs*, având un fișier de date reutilizabil de dimensiune 200MB autoextensibilă. Dimensiunea următorului spațiu de pe disc ce va fi alocat

pentru extinderea fișierului de date este 1000KB, iar dimensiunea maximă a spațiului de pe disc care poate fi alocat este nelimitată.

```
CREATE TABLESPACE users_tbs
  DATAFILE 'C:/baza_depozit/users1.dbf'
  SIZE 200M REUSE
  AUTOEXTEND ON NEXT 1000K
  MAXSIZE UNLIMITED;
```

Pas 9

- Se execută *script*-urile de generare a vizualizărilor dicționarului datelor și de instalare a obiectelor folosite de *PL/SQL*.

```
$ORACLE_HOME/rdbms/admin/catalog.sql;
$ORACLE_HOME/rdbms/admin/catproc.sql;
```

Pas 10

- Se creează fișierul de parametri *server* pentru a putea gestiona dinamic parametrii de inițializare.
 - Acesta este definit pe baza fișierului parametrilor de inițializare.

Exemplul 12.3

```
CREATE SPFILE='/oracle/dbs/spfile_depozit.ora'
FROM PFILE='/oracle/admin/baza_depozit/pfile/init.ora';
```

Pas 11

- Se dezvoltă strategiile de recuperare a bazei de date.
 -  Baza de date poate fi utilizată doar după ce a fost creată și pornită.
 - Orice utilizator cu privilegii de administrare poate opri sau porni baza de date. În general, doar administratorii au aceste privilegii.
 - De asemenea, aceștia sunt singurii utilizatori care pot accesa baza de date atunci când ea este închisă.

12.2. Pornirea unei baze de date

- Presupune crearea unei instanțe asociate și alegerea modului în care baza de date va rula.
- Există mai multe metode de a porni o bază de date, în funcție de utilitarul folosit (*SQL*Plus, RMAN, OEM*).
- Pornirea unei instanțe presupune pregătirea contextului necesar pentru utilizarea bazei de date (allocarea spațiului de memorie *SGA* și crearea proceselor *background*, acestea nefind încă asociate bazei).
- O instanță poate fi pornită în mai multe moduri:
 - fără montarea bazei de date (*STARTUP NOMOUNT*)
 - accesul la baza de date nu este permis;
 - acest procedeu se realizează în vederea creării bazei de date sau pentru recrearea fișierelor de control;
 - cu montarea bazei de date, dar fără deschiderea acesteia (*STARTUP MOUNT*)
 - se permite accesul la baza de date numai pentru anumite operații de administrare (recuperarea bazei de date, adăugarea, ștergerea sau redenumirea fișierelor de reluare, redenumirea fișierelor de date etc.);
 - cu montarea și deschiderea bazei de date
 - se permite accesul la date;
 - dacă deschiderea bazei se realizează în mod restrictiv (*STARTUP RESTRICT*), atunci aceasta este disponibilă doar pentru administrator;
 - accesul tuturor utilizatorilor este permis numai dacă baza de date este deschisă în mod nerestrictiv.
- Pornirea unei instanțe include următoarele etape:
 - citirea fișierului parametrilor de inițializare;
 - alocarea memoriei *SGA*;
 - pornirea proceselor *background*;
 - deschiderea fișierelor istorice (*trace file* și *alert file*).
- Montarea bazei de date include următorii pași:
 - asocierea bazei de date la instanța pornită anterior;
 - localizarea și deschiderea fișierelor de control specificate în fișierul parametrilor de inițializare;

- consultarea fișierelor de control pentru a obține numele și starea fișierelor de date și de reluare (nu se verifică existența acestor fișiere).
- Procesul de deschidere al bazei de date presupune deschiderea fișierelor de date și a fișierelor de reluare. Dacă unul dintre aceste fișiere nu există, atunci sistemul va returna un mesaj de eroare.
 - Server-ul *Oracle* verifică dacă toate fișierele de date și cele de reluare pot fi deschise și, apoi, controlează starea bazei de date. Dacă este necesar, procesul *background SMON* inițiază recuperarea instanței.
 - Deschiderea bazei de date se face cu ajutorul comenzi *STARTUP*.
 - Atunci când instanța este deja pornită, se folosește comanda *ALTER DATABASE OPEN*.
 - Baza de date poate fi deschisă astfel încât să permită doar consultarea datelor (*ALTER DATABASE OPEN READ ONLY*) sau atât citirea, cât și scrierea acestora (*ALTER DATABASE OPEN READ WRITE*).
 - Baza de date poate fi încărcată în mod dedicat sau partajat. Sistemul *Oracle* permite mai multor instanțe să fie asociate concurent unei singure baze de date. Prin setarea parametrului de inițializare *CLUSTER_DATABASE*, baza de date devine disponibilă mai multor instanțe.
 - Comanda pentru pornirea unei instanțe are următoarea sintaxă simplificată:


```
STARTUP [FORCE] [RESTRICT]
[ {OPEN [RECOVER] [bază_de_date] | MOUNT | NOMOUNT} ] ;

```

 - Clauza *FORCE* permite abandonarea forțată a unei instanțe pornite.
 - Clauza *RECOVER* asigură desfășurarea procesului de recuperare înainte de pornirea bazei.

12.3. Închiderea unei baze de date

- Pentru a iniția operația de închidere a bazei de date se folosește comanda *SHUTDOWN*.
 - Este necesară conectarea ca *SYSDBA* sau *SYSOPER*.
- Utilizatorii care încearcă să se conecteze la baza de date în timp ce închiderea acesteia este în derulare, vor primi mesajul de eroare „*ORA-01090: shutdown in progress - connection is not permitted*“.

- Procesul de închidere a unei baze de date presupune trei etape:
 - închiderea bazei de date
 - datele din *SGA* sunt înregistrate în fișierele de date și în cele de reluare care apoi vor fi închise, iar fișierele de control rămân deschise;
 - demontarea bazei de date
 - baza de date este disociată de instanță și sunt închise fișierele de control;
 - oprirea instanței
 - se eliberează memoria utilizată de *SGA* și se opresc toate procesele *Oracle*, apoi se închid fișierele istorice.
- Închiderea unei baze de date se poate realiza în mai multe moduri:
 - normal;
 - imediat;
 - tranzacțional;
 - renunțare (*abort*).

Închiderea bazei de date în modul normal

- Este opțiune implicită.
- Se realizează prin comanda *SHUTDOWN NORMAL*.
- Aceasta presupune că nu mai sunt permise conexiuni noi.
- Sistemul așteaptă ca toți utilizatorii curenti să se deconecteze, apoi demarează procesul de închidere a bazei de date.
- Baza de date și fișierele de reluare sunt scrise pe disc.
- Repornirea bazei nu necesită recuperarea instanței.

Închiderea bazei de date în modul imediat

- Se realizează prin comanda *SHUTDOWN IMMEDIATE*.
- Se folosește pentru a iniția o operație automată de recuperare a bazei, atunci când este anunțată o întrerupere imediată de curent sau atunci când baza de date, respectiv una dintre aplicațiile sale funcționează anormal și utilizatorii nu se pot deconecta.
- Presupune că nu sunt permise conexiuni sau tranzacții noi.
- Tranzacțiile care nu sunt permanentizate se derulează înapoi (*rollback*).
- Sistemul deconectează automat toți utilizatorii și începe procesul de oprire a bazei de date. Următoarea pornire a bazei nu va necesita recuperarea instanței.

Închiderea bazei de date în modul tranzacțional

- Se realizează prin comanda *SHUTDOWN TRANSACTIONAL*.
- Este utilizată pentru a permite finalizarea tranzacțiilor active, înainte de a se activa procesul de închidere a bazei.
 - Prin urmare, utilizatorul va fi deconectat numai după ce se termină tranzacția curentă a acestuia cu baza de date.
- Nu sunt permise conexiuni sau tranzacții noi.
- Nici în acest caz, următoarea pornire a bazei de date nu va necesita recuperarea instanței.

Închiderea bazei de date în modul renunțare

- Se realizează prin comanda *SHUTDOWN ABORT*.
- Presupune că baza de date se închide instantaneu, prin renunțarea la instanță.
- Nu sunt permise conexiuni sau tranzacții noi.
- Comenzile *SQL* curente nu sunt procesate.
- Tranzacțiile care nu sunt permanentizate, nu se derulează înapoi (*rollback*).
- Sistemul deconectează automat toți utilizatorii.
- Următoarea pornire a bazei de date necesită operația de recuperare a instanței, care se realizează în mod automat.

12.4. Starea de repaus a unei baze de date

- Administratorii bazei trebuie să izoleze uneori anumite acțiuni de cele concurente realizate de utilizatorii obișnuiți (tranzacții, interogări sau comenzi *PL/SQL*).
 - De exemplu, în timpul modificării structurii unui tabel poate fi necesară interzicerea tranzacțiilor concurente care accesează datele acestui tabel.
- Procesul de izolare a bazei de date presupune închiderea acesteia și redeschiderea sa în mod restrictiv (*STARTUP RESTRICT*). În cazul sistemelor care trebuie să funcționeze continuu, oprirea temporară a bazei de date poate cauza probleme importante.
- *Oracle9i* a introdus posibilitatea de izolare a bazei de date prin trecerea sa în stare de repaus, fără a deconecta utilizatorii. În acest fel, administratorul poate întreprinde în siguranță acele acțiuni a căror execuție cere izolare față de utilizatorii obișnuiți. De asemenea, se mărește disponibilitatea pentru sistemele care trebuie să funcționeze continuu.

Bibliografie

1. *Programare avansată în Oracle9i*, I. Popescu, A. Alecu, L. Velcescu, G. Florea (Mihai), Ed. Tehnică (2004)
2. *Oracle Database PL/SQL Language Reference 11g Release 2*, Oracle Online Documentation (2012)
3. *Oracle Database SQL Language Reference 11g Release 2*, Oracle Online Documentation (2012)
4. *Oracle Database 11g: PL/SQL Fundamentals, Student Guide*, Oracle University (2009)

CUPRINS

13. Securitatea bazei de date	2
13.1. Administrarea utilizatorilor și a resurselor	5
13.1.1. Metode de autentificare a utilizatorilor.....	6
13.1.2. Administrarea utilizatorilor.....	10
13.1.3. Administrarea parolelor și a resurselor utilizând profiluri.....	14
13.1.4. Informații despre utilizatori și profiluri	20
13.2. Administrarea privilegiilor și a role-urilor	21
13.2.1. Privilegii.....	21
13.2.2. Role-uri	22
13.2.3. Acordarea privilegiilor și a <i>role</i>-urilor.....	25
13.2.4. Revocarea privilegiilor și a <i>role</i>-urilor	28
13.2.5. Informații despre privilegii și role-uri.....	29
13.4. Auditarea	30
13.4.1. Opțiuni de audit.....	30
13.4.2. Mecanisme pentru audit.....	33
13.4.3. Informații despre audit.....	35
Bibliografie	36

13. Securitatea bazei de date

- Securitatea unei baze de date presupune monitorizarea acțiunilor realizate de utilizatori asupra acesteia sau a obiectelor sale.
 - În acest sens, sistemul folosește scheme de obiecte și domenii de securitate.
 - Un utilizator este un nume definit în baza de date care poate accesa obiectele acesteia.
 - Obiectele bazei de date sunt conținute în scheme.
 - Definirea utilizatorilor și a schemelor de obiecte ajută administratorii să asigure securitatea bazei de date.
- Accesul tuturor utilizatorilor la anumite obiecte este reglementat prin acordarea de privilegii și *role-uri*.
 - Un privilegiu reprezintă permisiunea de a accesa un obiect într-o manieră predefinită.
 - Un *role* este un grup de privilegii care poate fi acordat utilizatorilor bazei de date sau altor *role-uri*.
- Atunci când este creat un utilizator, administratorul bazei de date trebuie să definească un domeniu de securitate pentru acesta.
 - Sunt specificate metodele de autentificare, spațiile tabel implicate și cele temporare, profilul pentru restricționarea folosirii resurselor bazei de date, privilegiile și *role-urile* acordate utilizatorului respectiv.

Securitate la nivel de sistem

- Presupune administrarea adecvată a utilizatorilor, alegerea metodelor de autentificare a acestora și stabilirea securității sistemului de operare gazdă.
- Fiecare bază de date are unul sau mai mulți administratori care sunt responsabili pentru asigurarea politicilor de securitate (administratori pentru securitate).
 - Dacă baza de date are dimensiuni reduse, administratorul bazei poate avea și responsabilități de securitate.
- Utilizatorii sunt cei care accesează informațiile din baza de date și de aceea, este deosebit de important modul de administrare al acestora.
 - În general, administratorul pentru securitate este singurul utilizator al bazei care are dreptul de administrare a celorlalți utilizatori (creare, modificare, eliminare).

- Sistemul furnizează mai multe metode de autentificare a utilizatorilor bazei de date, folosind:
 - parole;
 - sistemul de operare gazdă;
 - servicii de rețea sau protocolul *SSL (Secure Sockets Layer)*;
 - autentificare și autorizare de tip *proxy*.
- La nivelul sistemului de operare pe care rezidă *server-ul* și aplicațiile de baze de date trebuie ca:
 - administratorii bazei de date să aibă privilegii de creare, respectiv de ștergere a fișierelor;
 - utilizatorii obișnuiți ai bazei să nu dețină privilegii de creare sau de ștergere a fișierelor utile funcționării bazei de date;
 - administratorii pentru securitate să dețină privilegii de modificare a domeniului de securitate pentru conturile disponibile în sistemul de operare (dacă identificarea *role-urilor* pentru utilizatorii bazei de date *Oracle* se realizează prin sistemul de operare).

Securitate la nivel de date

- Include mecanisme de control al accesului la date și al gradului de utilizare a obiectelor bazei.
 - Se stabilesc utilizatorii care au acces la un anumit obiect și tipurile de acțiuni permise asupra sa.
 - În acest sens, se acordă utilizatorilor bazei de date anumite privilegii sistem și obiect, care pot fi grupate în *role-uri*.
 - De asemenea, pentru fiecare obiect al unei scheme sunt definite acțiunile care vor fi auditate.
- O altă modalitate de stabilire a securității la acest nivel este folosirea vizualizărilor.
 - Acestea pot restricționa accesul la datele unui tabel, prin excluderea unor coloane sau linii.
 - Sistemul permite implementarea politicilor de securitate prin folosirea unor funcții și asocierea acestora la tabele sau vizualizări (*fine-grained access control*).
 - O astfel de funcție generează automat o condiție *WHERE* într-o instrucțiune *SQL* și astfel se restricționează accesul la anumite linii de date.

Securitate la nivel de utilizator

- Există două modalități generale de stabilire a securității la nivel de utilizatori:
 - prin intermediul parolelor;
 - prin acordarea privilegiilor.
- Dacă autentificarea utilizatorilor este administrată de baza de date, atunci administratorii trebuie să dezvolte politici de securitate pentru parole.
 - De exemplu, să impună utilizatorilor bazei de date să-și schimbe parolele la anumite intervale de timp, lungimea parolelor să fie suficient de mare, iar în componența acestora să intre atât litere, cât și cifre.
- Problema administrării privilegiilor pentru diferite tipuri de utilizatori este de o deosebită importanță.
 - Pentru bazele de date cu mulți utilizatori, administrarea privilegiilor este mai eficientă dacă se folosesc *role-uri*.
 - În schimb, atunci când numărul de utilizatori este scăzut, se recomandă să se acorde privilegii explicate pentru fiecare utilizator și să se evite folosirea *role-urilor*.
 - Administratorul pentru securitate trebuie să decidă care sunt categoriile de grupuri de utilizatori și să atribuie *role-uri* fiecărui grup în parte. De asemenea, acesta trebuie să decidă ce privilegii trebuie acordate nominal utilizatorilor.
- Administratorii pentru securitate
 - Trebuie să dezvolte politici de securitate inclusiv pentru administratorii bazei de date.
 - Pentru baze foarte mari, care necesită mai mulți administratori, administratorul pentru securitate trebuie să decidă care sunt grupurile de privilegii de administrare și să le includă în *role-uri* de administrare.
 - Pentru baze de date mici este suficientă crearea unui singur *role* specific și acordarea acestuia tuturor administratorilor bazei.
 - Trebuie să creeze politici de securitate pentru dezvoltatorii de aplicații care se conectează la baza de date.
 - Pentru crearea obiectelor, se pot acorda privilegii direct dezvoltatorilor sau numai administratorilor bazei, care să definească obiectele la cererea acestora.
 - Dezvoltatorii de aplicații sunt singurii utilizatori ai bazei de date care necesită grupuri speciale de privilegii. Spre deosebire de utilizatorii finali, aceștia au nevoie de privilegii sisteme specifice activității lor.

- Privilegiul sistem *CREATE* este acordat dezvoltatorilor, pentru ca aceștia să poată crea obiectele necesare în aplicații. Dacă dezvoltatorii de aplicații au privilegii pentru crearea obiectelor, administratorul pentru securitate trebuie să controleze limitele de folosire a spațiului bazei de date pentru fiecare dezvoltator în parte.
- Trebuie să definească proceduri de verificare (audit) pentru baza de date.
 - Se poate impune ca aceste proceduri să fie active doar pentru anumite acțiuni sau se pot face verificări prin selecție aleatoare.
 - Administratorul trebuie să decidă care este nivelul minimal la care se fac aceste verificări.
- În cazul sistemelor mari de baze de date, pe care rulează multe aplicații, trebuie să existe administratori pentru aplicații. Aceștia sunt responsabili pentru:
 - crearea și administrarea obiectelor folosite de aplicațiile bazei;
 - crearea *role*-urilor corespunzătoare aplicațiilor și gestiunea privilegiilor asociate fiecarui astfel de *role*;
 - menținerea și modificarea codului aplicațiilor, procedurilor și pachetelor *Oracle*, dacă acest lucru este necesar.
- De cele mai multe ori, administratorul pentru aplicații este unul dintre dezvoltatorii care analizează și proiectează aplicația.

13.1. Administrarea utilizatorilor și a resurselor

- În funcție de licențele primite pentru sistemul *Oracle*, trebuie limitat numărul de sesiuni concurente și de utilizatori conectați la baza de date.
 - Aceasta se realizează prin setarea parametrilor de inițializare de tip *LICENSE*.
- Licențele pentru folosirea concurentă limitează numărul de sesiuni care pot fi conectate simultan la baza de date.
 - Numărul maxim de sesiuni concurente (*LICENSE_MAX_SESSIONS*) poate fi precizat înainte de a porni instanța și modificat în timp ce baza de date este pornită.
 - Atunci când această limită este atinsă, sistemul va trimite utilizatorilor un mesaj prin care îi anunță acest lucru.
 - În acest caz, se pot conecta la baza de date numai utilizatorii care au privilegiul *RESTRICTED SESSION*.

- Limitarea numărului de utilizatori restricționează numărul autorizațiilor individuale de folosire a sistemului.
 - Precizarea numărului maxim de utilizatori care pot fi creați în bază se face înainte de pornirea unei instanțe (*LICENSE_MAX_USERS*).
 - Această limită poate fi modificată în timpul funcționării instanței, prin folosirea comenzi *ALTER SYSTEM*.
 - După depășirea ei, nu mai pot fi creați noi utilizatori. În acest caz, sistemul va trimite un mesaj prin care anunță că numărul maxim de utilizatori permisi a fost atins.
- Vizualizarea *V\$LICENSE* din dicționarul datelor permite identificarea setărilor curente privind limitările impuse, numărul curent de sesiuni utilizator și numărul maxim de sesiuni concurente atins de la momentul pornirii instanței.

13.1.1. Metode de autentificare a utilizatorilor

- Pentru a preveni folosirea neautorizată a unui cont de utilizator (*username*), sistemul *Oracle* realizează validarea utilizatorilor prin diferite metode, înainte ca aceștia să inițieze o sesiune de lucru cu baza de date:
 - autentificare prin baza de date, folosind parole;
 - autentificare externă, prin sistemul de operare sau servicii de rețea;
 - autentificare globală, prin protocolul de securitate *SSL*;
 - autentificare *proxy*, dacă utilizatorii se conectează la baza de date printr-un *server* de aplicații.
- În general, este folosită aceeași metodă pentru autentificarea tuturor utilizatorilor bazei de date. Totuși, sistemul *Oracle* permite abordarea tuturor metodelor de autentificare, în cadrul aceleiași instanțe a bazei.
- Sistemul necesită proceduri speciale de autentificare pentru administratorii bazei de date, deoarece ei pot executa operații importante asupra acestora.

Autentificarea prin baza de date

- Administrarea conturilor, respectiv a parolelor și validarea utilizatorilor sunt realizate în întregime de către sistem.
- Pentru fiecare utilizator se definesc parole de acces. Din motive de securitate acestea pot fi stocate în format criptat.

Autentificarea externă

- Conturile utilizatorilor sunt întreținute de sistem, iar administrarea parolelor și autentificarea utilizatorilor se fac printr-un serviciu extern (sistemul de operare sau un serviciu rețea, de exemplu *Oracle Net*).
- Conturile utilizatorilor bazei de date vor fi formate dintr-un prefix urmat de numele conturilor acestora din sistemul de operare.
 - Prefixul este setat prin parametrul de inițializare *OS_AUTHENT_PREFIX*.
 - Dacă valoarea acestuia este modificată, atunci conturile care folosesc vechiul prefix sunt invalide.
 - Valoarea implicită a parametrului este *OPS\$*.
- Un utilizator care încearcă să se conecteze la baza de date *Oracle*, va fi autentificat de sistemul de operare.
 - Dacă în acest sistem utilizatorul are contul *nume*, atunci conectarea la baza de date este permisă doar dacă sistemul *Oracle* conține contul corespondent al utilizatorului la nivelul bazei de date (*OPS\$nume*).
- Beneficiile autentificării prin **sistemele de operare** sunt:
 - utilizatorii se pot conecta la sistemul *Oracle* în mod convențional, fără să folosească un cont și o parolă (de exemplu, un utilizator poate să invoce utilitarul *SQL*Plus*, lansând direct comanda *SQLPLUS*);
 - controlul autorizării utilizatorilor este centralizat în sistemul de operare (în baza de date nu trebuie stocate și administrate parolele utilizatorilor).
- Sistemul *Oracle* poate utiliza un **serviciu de rețea** (*DCE*, *Kerberos*, *SESAME* etc.) pentru autentificarea utilizatorilor. Pentru aceasta trebuie utilizate facilitățile aduse de *Oracle Advanced Security*.
- Serviciul de autentificare externă prin rețea folosește infrastructuri cu chei publice.
 - Elementele fundamentale ale acestora sunt certificatele digitale, autoritatele de certificare și facilitățile de administrare a certificatelor.
 - În funcționarea sistemelor cu chei publice (*PKI*) este necesar un sistem de generare, circulație și autentificare a cheilor folosite de utilizatori.
 - Autoritatele de certificare distribuie certificate de chei autentificate.
 - Certificatul reprezintă o asociere imposibil de falsificat dintre o cheie publică și un anumit atribut al posesorului său. El poartă semnătura digitală a unei autorități de certificare care, în acest fel, confirmă identitatea subiectului.

- Sistemele de autentificare care folosesc infrastructuri cu chei publice eliberează utilizatorilor certificate digitale, pentru autentificarea directă la *server*.
- Infrastructura cu chei publice folosită de *Oracle* are componentele:
 - protocolul *SSL* (independent de platformă și de aplicație, care furnizează servicii de autentificare, compresie de date, criptare și integritate a datelor pentru o serie de protocoale *Internet* la nivel aplicație), pentru autentificarea și gestiunea sigură a cheilor;
 - utilitarul *Oracle Call Interface* și funcții *PL/SQL* pentru folosirea semnăturii digitale și verificarea acestora utilizând certificatul asociat;
 - certificate pentru utilizatori, eliberate de o autoritate de certificare care oferă un nivel ridicat de încredere;
 - portofele virtuale, care conțin cheia privată a utilizatorului, certificatul de autentificare și lista certificatelor de bază prin care utilizatorul obține un nivel ridicat de încredere;
 - *Oracle Wallet Manager*, o aplicație *Java* folosită pentru gestiunea și editarea scrisorilor de acreditare din portofelele virtuale (protejează cheile utilizatorilor, gestionează certificatele X.509v3 pe *server*-ele *Oracle*, generează perechi de chei publice și private, creează cereri de certificare către autoritatea de certificare, instalează certificate, configerează certificate de încredere, deschide un portofel *Oracle* pentru a accesa un *PKI*, creează portofele care pot fi deschise folosind *Oracle Enterprise Login Assistant*);
 - certificate de tip X.509v3, obținute de la autorități de certificare externe sistemului *Oracle*;
 - instrumentul *Oracle Enterprise Security Manager*, pentru gestiunea centralizată a privilegiilor;
 - serviciul *Oracle Internet Directory*, care permite configurarea și administrarea centralizată a utilizatorilor, incluzând privilegii și atrbute de securitate pentru autentificarea acestora cu certificate X.509;
 - utilitarul *Oracle Enterprise Login Assistant*, pentru deschiderea sau închiderea portofelului virtual al unui utilizator și activarea sau dezactivarea comunicațiilor unei aplicații securizate prin *SSL*.

Autentificarea globală

- Utilizatorii sunt identificați în baza de date ca utilizatori globali, folosind protocolul *SSL*.
- Administrarea utilizatorilor se face în afara bazei de date, prin centralizarea acestora într-un director, numit director de servicii (*directory service*).
- *Role-urile globale* sunt definite în baza de date, dar autorizațiile pentru acestea se acordă prin directorul de servicii.
- Avantajul gestionării centralizate constă în posibilitatea definirii de utilizatori și *role-uri* la nivel de companie.
- Singurul dezavantaj este că utilizatorul trebuie recreat în directorul de servicii, pentru fiecare bază de date la care trebuie să aibă acces. Soluția este crearea de utilizatori independenți de schemă, ceea ce le permite acestora să folosească o schemă în comun.
- Procesul de creare a unui utilizator independent de schemă presupune:
 - crearea unei scheme partajate la nivelul bazei de date, folosind comanda `CREATE USER nume_schemă IDENTIFIED BY 'specificație_identificare';`
 - crearea utilizatorilor globali în directorul de servicii și asocierea lor la această schemă.

Autentificarea proxy

- În cazul configurației *multitier*, autentificarea *proxy* presupune verificarea dreptului de acces al *server-ului* de aplicații la baza de date, protejând identitatea și privilegiile *client-ilor* de-a lungul tuturor nivelurilor și verificând doar acțiunile realizate în favoarea acestora. De asemenea, sunt controlate acțiunile pe care *server-ul* de aplicații le inițiază în nume propriu.
- Sistemul *Oracle* oferă două forme de autentificare *proxy*:
 - dacă *client-ul* este o aplicație sau un utilizator global, atunci el va fi autentificat de către *server-ul* de aplicații;
 - dacă *client-ul* este un utilizator al bazei de date, atunci el nu va fi autentificat de către *server-ul* de aplicații (identitatea *client-ilor* și parolele trec prin *server-ul* de aplicații către *server-ul* de baze de date, unde are loc autentificarea);
- Pentru a putea să acționeze în favoarea *client-ului*, *server-ul* de aplicații este autorizat de către administrator.

- Principalul avantaj al arhitecturii *multitier* este acela că permite mai multor *client*-i să aibă acces la *server*-ul de date, printr-un *server* de aplicații, fără să fie necesare conexiuni separate.
- Într-un mediu *multitier* autentificarea se realizează în trei etape:
 - aplicația *client* trimite dovada autenticității către *server*-ul de aplicații (parolă sau certificat);
 - *server*-ul de aplicații verifică autenticitatea *client*-ului și apoi se autentifică și el la *server*-ul de baze de date;
 - *server*-ul de baze de date verifică autenticitatea *server*-ului de aplicații, existența *client*-ului și faptul că *server*-ul de aplicații are dreptul de conectare pentru *client*-ul respectiv.

13.1.2. Administrarea utilizatorilor

- Fiecare bază de date *Oracle* conține o listă de utilizatori. Pentru a accesa baza de date, un utilizator trebuie să execute o aplicație a bazei și să se conecteze la o instanță a acesteia, folosind un cont valid definit în bază.
- Crearea unui utilizator se realizează prin comanda *CREATE USER*.
 - Pentru a avea dreptul de folosire a acestei comenzi este necesar privilegiul sistem *CREATE USER*.
 - În general, administratorul pentru securitate este singurul care are acest privilegiu.
 - Crearea unui utilizator constă în definirea identității sale (nume și parolă), specificarea profilului, a spațiului tabel implicit, a cotei de folosire a spațiului tabel și a spațiului tabel temporar în care sunt create segmentele temporare.
- Forma simplificată a comenzi *CREATE USER* este următoarea:

```
CREATE USER nume_utilizator
  IDENTIFIED
    { BY parolă | EXTERNALLY
    | GLOBALLY AS
      'CN = nume_user, alte_atribute_de_identificare' }
  [DEFAULT TABLESPACE nume_spațiu_tabel]
  [TEMPORARY TABLESPACE nume_spațiu_tabel]
  [QUOTA { întreg [{K | M} ] | UNLIMITED}
    ON nume_spațiu_tabel]
  [PROFILE nume_profil]
  [PASSWORD EXPIRE]
  [ACCOUNT {LOCK | UNLOCK} ] };
```

- Numele unui utilizator trebuie să fie unic.
- Un utilizator și un *role* nu pot avea același nume.
- Modul de autentificare poate fi realizat prin baza de date (*IDENTIFIED BY parolă*), extern (*IDENTIFIED EXTERNALLY*) sau prin protocolul SSL (*IDENTIFIED GLOBALLY AS 'specificație_identificare'*).
- Sirul *specificație_identificare* furnizează un mod de identificare la nivelul directorului de servicii.
- Fiecare utilizator trebuie să aibă asociat un spațiu tabel implicit (*DEFAULT TABLESPACE*) în care sistemul stochează obiectele create de utilizator, dacă nu se specifică un alt spațiu tabel pentru acestea. Spațiul tabel implicit este *SYSTEM*.
- Pentru fiecare utilizator se poate specifica un spațiu tabel temporar. Acest spațiu tabel este folosit pentru stocarea segmentelor temporare care sunt necesare comenzilor SQL inițiate de către utilizator. Dacă nu este specificat un spațiu tabel temporar, sistemul alocă în mod implicit utilizatorului spațiul tabel temporar care a fost definit la crearea bazei de date. Dacă nici acesta nu a fost specificat, spațiul tabel temporar implicit este *SYSTEM*.
- Cu scopul de a preveni consumul excesiv al spațiului bazei de date se pot preciza limite de folosire (cote) pentru spațiile tabel la care are acces utilizatorul. Aceste limite sunt specificate prin clauza *QUOTA*. Dacă limita spațiului tabel este 0, atunci utilizatorul nu mai poate crea obiecte noi, iar pentru obiectele existente în spațiul tabel respectiv acesta nu mai poate aloca spațiu suplimentar. Opțiunea *UNLIMITED* presupune folosirea nelimitată a spațiului tabel respectiv.
- Privilegiul sistem *UNLIMITED TABLESPACE* permite utilizatorilor să dețină spațiu tabel nelimitat. Acest privilegiu are prioritate față de toate limitele specificate pentru spațiile tabel alocate utilizatorului respectiv.
- De asemenea, la crearea unui utilizator se poate specifica profilul acestuia. Profilurile facilitează administrarea parolelor și a limitelor de folosire a resurselor. Dacă nu este specificat nici un profil, se asociază unul implicit.
- Clauza *PASSWORD EXPIRE* presupune că parola utilizatorului trebuie schimbată la prima conectare la baza de date.
- Pentru blocarea sau deblocarea contului unui utilizator este folosită clauza *ACCOUNT*, cu opțiunile *LOCK*, respectiv *UNLOCK*.

Exemplul 13.1

- a) Se creează utilizatorul *student1*, identificat prin parola *s1t2u3d*. Utilizatorului i se asociază spațiul tabel *users_tbs* cu posibilitatea de a folosi maxim *5MB* din acesta și spațiul tabel temporar *temp_tbs* din care poate folosi maxim *3MB*. Limitarea folosirii resurselor bazei de date este definită de profilul *user_p*.

```
CREATE USER student1
  IDENTIFIED BY s1t2u3d
  DEFAULT TABLESPACE users_tbs
  QUOTA 5M ON users_tbs
  TEMPORARY TABLESPACE temp_tbs
  QUOTA 3M ON temp_tbs
  PROFILE user_p;
```

- b) Se creează utilizatorul *student2* cu opțiunea de autentificare externă.

```
CREATE USER student2
  IDENTIFIED EXTERNALLY
  DEFAULT TABLESPACE users_tbs
  QUOTA 15M ON users_tbs
  PROFILE user_p;
```

- c) Se creează un utilizator al bazei de date care să fie autentificat extern, prin sistemul de operare. Contul corespunzător de la nivelul sistemului de operare este *student3*.

```
CREATE USER ops$student3
  IDENTIFIED EXTERNALLY
  DEFAULT TABLESPACE users_tbs
  QUOTA 10M ON users_tbs
  PROFILE user_p;
```

- d) Se creează utilizatorul global *student*, având drept atribut de identificare jobul (*JB*), departamentul (*DP*) și specializarea (*SP*).

```
CREATE USER student
  IDENTIFIED GLOBALLY AS
    'JB=programator, DP=informatica, SP=oracle'
  DEFAULT TABLESPACE users_tbs
  QUOTA 20M ON users_tbs
  PROFILE user_p;
```

- Pentru a modifica o opțiune a domeniului de securitate al unui utilizator (modul de autentificare, limitele unui spațiu tabel, revocarea unui spațiu tabel, profilul) este necesar privilegiul sistem *ALTER USER*.

- De obicei, acesta este deținut doar de administratorul pentru securitate.
- Comanda folosită este *ALTER USER*, iar modificările specificate prin aceasta nu afectează sesiunea curentă a utilizatorului.
- Singura opțiune pe care fiecare utilizator o poate modifica pentru propriul cont este parola. Pentru aceasta nu este necesar un privilegiu sistem special, altul decât cel de conectare la baza de date. Comanda folosită are următoarea sintaxă:

```
ALTER USER nume_utilizator IDENTIFIED BY parolă;
```
- Ștergerea unui utilizator implică eliminarea acestuia și a schemei asociate, din dicționarul datelor, ceea ce determină ștergerea imediată a tuturor obiectelor conținute în schemă.
 - Dacă se dorește păstrarea schemei unui utilizator, iar acesta trebuie să piardă dreptul de a accesa baza de date, i se va revoca utilizatorului respectiv privilegiul *CREATE SESSION*.
 - Nu se poate elimina un utilizator care este conectat la baza de date.
 - Pentru a putea elmina un utilizator este necesar privilegiul sistem *DROP USER*.
 - Dacă schema utilizatorului conține obiecte, trebuie folosită opțiunea *CASCADE* pentru a șterge atât utilizatorul, cât și toate obiectele asociate, inclusiv cheile externe care referă tabelele deținute de utilizator.
 - Dacă utilizatorul are obiecte asociate și nu este utilizată această clauză, atunci sistemul returnează un mesaj de eroare, utilizatorul nefiind eliminat. Sintaxa comenzi folosite pentru ștergerea unui utilizator este:

```
DROP USER nume_utilizator [CASCADE];
```
- Fiecare bază de date conține un grup de utilizatori numit *PUBLIC*.
 - Deoarece orice utilizator al bazei de date face parte în mod automat din grupul *PUBLIC*, privilegiile și *role*-urile acordate acestui grup sunt accesibile tuturor utilizatorilor.
 - Ca membru al acestui grup utilizatorul poate consulta toate tabelele din dicționarul datelor prefixate de *USER* sau *ALL*.
 - Se pot acorda sau revoca grupului *PUBLIC* orice privilegii sistem, privilegii obiect sau *role*-uri. Din motive de securitate, este recomandabil să se acorde doar privilegiile sau *role*-urile care îi interesează pe toți membrii grupului. Acordarea sau revocarea unor privilegii sistem sau obiect grupului *PUBLIC* poate conduce la recompilarea vizualizărilor, procedurilor, funcțiilor, pachetelor și declanșatorilor.

Grupul *PUBLIC* are următoarele restricții:

- nu se pot asocia cote spațiilor tabel deținute de grup, însă se poate acorda acestui privilegiu sistem *UNLIMITED TABLESPACE*;
- se pot crea legături de baze de date sau sinonime publice (prin comanda *CREATE PUBLIC {DATABASE LINK | SYNONYM}*), acestea fiind singurele obiecte deținute de grupul *PUBLIC*.

13.1.3. Administrarea parolelor și a resurselor utilizând profiluri

- Un profil este un set de limitări de resurse care poate fi atribuit unui utilizator al bazei de date. Fiecare bază *Oracle* permite definirea unui număr nelimitat de profiluri. Acestea trebuie create și administrate doar dacă politica de securitate a bazei de date cere ca utilizarea resurselor să fie limitată. Pentru a putea folosi profilurile, mai întâi se creează categorii de grupuri de utilizatori similari.
- Profilurile pot fi atribuite fiecărui utilizator în parte (folosind comanda *CREATE USER* sau *ALTER USER*) sau se pot defini profiluri implicate care sunt asociate tuturor utilizatorilor care nu au un profil specific.
- Pentru a crea un profil este necesar privilegiul sistem *CREATE PROFILE*. În momentul creării se pot explicita limitele folosirii unor resurse particulare sau parametrii pentru parole.
- Comanda prin care se creează un profil este următoarea:

```
CREATE PROFILE nume LIMIT
  [SESSIONS_PER_USER valoare]
  [CPU_PER_SESSION valoare]
  [CPU_PER_CALL valoare]
  [CONNECT_TIME valoare]
  [IDLE_TIME valoare]
  [LOGICAL_READS_PER_SESSION valoare]
  [LOGICAL_READS_PER_CALL valoare]
  [PRIVATE_SGA valoare]
  [COMPOSITE_LIMIT valoare]
  [FAILED_LOGIN_ATTEMPTS valoare]
  [PASSWORD_LIFE_TIME valoare]
  [PASSWORD_REUSE_TIME valoare]
  [PASSWORD_REUSE_MAX valoare]
  [PASSWORD_LOCK_TIME valoare]
  [PASSWORD_GRACE_TIME valoare]
  [PASSWORD_VERIFY_FUNCTION {funcție | NULL | DEFAULT} ];
```

- Pentru fiecare parametru care apare în comandă se poate preciza o valoare întreagă sau opțiunea *UNLIMITED*, respectiv *DEFAULT*. Opțiunea *UNLIMITED* indică

posibilitatea de folosire nelimitată a resursei respective. Dacă se folosește *DEFAULT*, atunci limita de folosire a resursei va fi preluată din profilul implicit.

- Fiecare bază de date are un profil implicit. Toate limitările de resurse nespecificate pentru profilurile particulare vor fi setate automat la valorile implicate. La crearea bazei de date *server-ul Oracle* definește profilul implicit *DEFAULT*.
- Sistemul *Oracle* poate autentifica utilizatorii folosind informații stocate în baza de date. Cea mai importantă informație de autentificare o reprezintă parola asociată unui utilizator. Aceasta este criptată și stocată în dicționarul datelor. Utilizatorul poate oricând să-și schimbe propria parolă.
- Pentru a asigura confidențialitatea parolelor, sistemul permite criptarea acestora în timpul conexiunilor din rețea (*client/server* sau *server/server*). Dacă este activată această funcționalitate atât pe mașina *client*, cât și pe *server*, sistemul codifică parolele înainte de a le trimite în rețea, folosind o versiune modificată a algoritmului de criptare *DES (Data Encryption Standard)*.
- Dacă un utilizator introduce parola greșit de un număr specificat de ori, sistemul blochează contul asociat acestuia. În funcție de modul de configurare, contul poate fi deblocat automat, după un interval specificat de timp, sau manual, de administratorul bazei de date. Prin parametrul *FAILED_LOGIN_ATTEMPTS* se precizează numărul de conectări care pot eșua înainte de blocarea contului, iar prin parametrul *PASSWORD_LOCK_TIME* numărul de zile în care acesta va fi blocat.
- Administratorul bazei de date poate bloca manual un cont, folosind comanda *ALTER USER*. În acest caz, contul nu mai poate fi deblocat automat, administratorul trebuind să-l deblocheze explicit.
- Prin parametrul *PASSWORD_LIFE_TIME* se poate specifica durata de viață a unei parole. După această perioadă, parola expiră și trebuie schimbată. La prima încercare de conectare după expirare, apare un mesaj prin care utilizatorul este atenționat că trebuie să-și schimbe parola într-un anumit număr de zile (perioadă în care are încă dreptul de conectare). Această perioadă este precizată prin parametrul *PASSWORD_GRACE_TIME*. Dacă parola nu este schimbată până la terminarea perioadei de grație, atunci contul se blochează automat.
- Se recomandă ca schimbarea parolelor să nu se facă folosind comanda *ALTER USER*, deoarece aceasta nu realizează verificarea completă a complexității parolei. Este

preferabil ca pentru schimbarea unei parole să se utilizeze funcția *OCIPasswordChange()*.

- Pentru a verifica dacă o parolă nou specificată nu a mai fost utilizată anterior este menținută o istorie a parolelor. Prin parametrul *PASSWORD_REUSE_TIME* se poate preciza intervalul de timp (exprimat în număr de zile) în care utilizatorii nu pot reutiliza o parolă. De asemenea, se permite și precizarea numărului de schimbări consecutive ale parolei până se poate reutiliza o parolă anterioară (*PASSWORD_REUSE_MAX*).
- Fiecare parolă trebuie să aibă o complexitate minimă pentru a asigura protecția sistemului împotriva eventualilor intruși care încearcă să o descopere. În timpul execuției script-ului *utlpwdmg.sql*, server-ul *Oracle* creează în schema *SYS* funcția implicită *VERIFY_FUNCTION* care asigură verificarea complexității unei parole.
 - Aceasta verifică dacă parola îndeplinește următoarele condiții:
 - are lungimea de cel puțin 4 caractere;
 - nu coincide cu *username*-ul;
 - include cel puțin o literă, o cifră și un caracter special;
 - nu este un cuvânt simplu din lista de noțiuni interne (de exemplu, numele contului, al bazei de date sau al utilizatorului etc.);
 - diferă de parola anterioară cu cel puțin 3 caractere.
- Rutina de verificare a complexității parolelor poate fi modificată sau se poate crea o rutină nouă, în schema *SYS*. Comanda *CREATE PROFILE* permite, prin intermediul parametrului *PASSWORD_VERIFY_FUNCTION*, precizarea unei funcții *PL/SQL* pentru verificarea complexității parolelor.
- Pentru fiecare utilizator, sistemul permite specificarea unei limite de folosire a resurselor disponibile, în cadrul domeniului său de securitate. Astfel, se controlează consumul resurselor importante ale sistemului. Limitarea resurselor este o metodă deosebit de utilă în cazul sistemelor *multiuser*, unde acestea sunt costisitoare. Consumul excesiv al resurselor, de către unul sau mai mulți utilizatori, poate fi în detrimentul celorlalți utilizatori ai bazei de date.
- Sistemul poate controla utilizarea resurselor la nivel de sesiune (*session level*), la nivel de apel (*call level*) sau la ambele niveluri.
- În momentul conexiunii unui utilizator la o bază de date se creează o sesiune. Fiecare sesiune consumă timp *CPU* (timp de încărcare al procesorului) și memorie. Dacă

utilizatorul depășește limita de consum a resurselor, sistemul anulează comanda curentă și returnează un mesaj prin care indică depășirea limitei de consum a resurselor. Toate comenziile anterioare tranzacției curente rămân intacte și sunt permise doar operații de tip *COMMIT*, *ROLLBACK* sau deconectare. Toate celelalte operații produc erori.

- Sistemul permite și limitarea altor resurse la nivel de sesiune: numărul de sesiuni concurente pentru fiecare utilizator, timpul în care o sesiune poate rămâne inactivă, timpul de conectare consumat pentru fiecare sesiune, spațiul *SGA* (folosit de zonele private *SQL*) al unei sesiuni etc.
- Pentru procesarea comenziilor *SQL* sau a altor tipuri de apeluri către sistem este necesar un timp *CPU*. În medie, apelurile nu sunt mari consumatoare de timp *CPU*. În schimb, o comandă *SQL* poate implica multe interogări care pot fi mari consumatoare ale acestei resurse. Pentru a preveni folosirea inadecvată a timpului *CPU*, se pot fixa limite pentru fiecare apel în parte sau pentru apelurile din cadrul unei anumite sesiuni.
- Operațiile *I/O* sunt unele dintre cele mai costisitoare operații într-un sistem de baze de date. De aceea, sistemul permite limitarea citirilor blocurilor logice de date, la nivel de apel sau de sesiune. Citirea blocurilor logice de date include citirea blocurilor de date din memorie și de pe disc. Limitările presupun numărarea blocurilor citite în timpul unui apel sau pe parcursul unei sesiuni.
- Opțiunile de limitare a resurselor care intervin în comanda *CREATE PROFILE* sunt următoarele:
 - *SESSIONS_PER_USER* (numărul maxim de sesiuni concurente);
 - *CPU_PER_SESSION* (timpul de încărcare al procesorului pentru o sesiune, exprimat în secunde);
 - *CPU_PER_CALL* (timpul de încărcare al procesorului pentru un apel, exprimat în secunde);
 - *CONNECT_TIME* (timpul total al unei sesiuni, exprimat în minute);
 - *IDLE_TIME* (timpul de inactivitate continuă pe parcursul unei sesiuni, exprimat în minute);
 - *LOGICAL_READS_PER_SESSION* (numărul de blocuri de date citite într-o sesiune, inclusiv blocurile citite din memorie sau de pe disc);
 - *LOGICAL_READS_PER_CALL* (numărul de blocuri de date citite pe parcursul unui apel pentru a procesa o comandă *SQL*);

- *PRIVATE_SGA* (dimensiunea spațiului SGA alocat unei sesiuni);
- *COMPOSITE_LIMIT* (costul total al resurselor pentru o sesiune).
- Înainte de crearea profilurilor și specificarea limitelor de folosire a resurselor asociate lor, trebuie să se determine valorile estimative pentru fiecare limită de resurse. Estimarea acestor limite se poate face evaluând tipul de operații pe care le execută o categorie de utilizatori. De exemplu, dacă o categorie de utilizatori, în mod normal, nu face un număr mare de citiri din blocurile logice de date, atunci valorile parametrilor referitor la acest tip de resursă pot fi mari. De obicei, cea mai bună cale de a determina valoarea aproximativă a limitei de folosire a resurselor pentru un profil al unui utilizator dat, este consultarea informațiilor istorice cu privire la ocuparea fiecărui tip de resurse.
- Pentru a obține statistici referitoare la folosirea resurselor se poate utiliza componenta de monitorizare a utilitarului *OEM* sau *SQL*Plus*. De exemplu, administratorul pentru securitate poate folosi clauza *AUDIT SESSION* pentru a obține informații despre limitele *CONNECT_TIME*, *LOGICAL_READS_PER_SESSION* și *LOGICAL_READS_PER_CALL*.
- Pentru ca profilurile să aibă efect trebuie să fie pornită opțiunea de limitare a resurselor pentru baza de date. Activarea sau dezactivarea acestei opțiuni se poate face înaintea pornirii bazei de date sau în timp ce aceasta este deschisă. În acest sens, este folosit parametrul de inițializare *RESOURCE_LIMIT* (*TRUE* pentru activare, *FALSE* pentru dezactivare).

Exemplul 13.1

Se activează opțiunea de limitare a resurselor presupunând că baza de date este deschisă.

```
ALTER SYSTEM SET RESOURCE_LIMIT = TRUE;
```

- După cum se observă, comanda *CREATE PROFILE* permite limitarea costului total al resurselor. Astfel, utilizatorul poate folosi orice resurse dorește, dar suma cantităților resurselor consumate nu poate depăși această limită. Un profil poate folosi limitări explicite de resurse, concomitent cu limitarea costului total al resurselor. Atingerea uneia dintre limite determină oprirea activității utilizatorului în sesiune. Folosirea limitărilor compuse permite mai multă flexibilitate. Valoarea corectă a unei limite compuse de resurse depinde de cantitatea totală de resurse folosită în medie de un utilizator.

- Limitările de resurse la nivel de apel (*LOGICAL_READS_PER_CALL*, *CPU_PER_CALL*) sunt impuse fiecărui apel inițiat în timpul execuției unei comenzi *SQL*. Depășirea uneia dintre aceste limite determină întreruperea procesării comenзii și anularea acesteia, sesiunea utilizatorului fiind menținută.
- După ce profilul a fost creat, el poate fi asociat utilizatorilor bazei de date. Nu este posibil ca un utilizator să aibă concomitent mai multe profiluri. Dacă un profil este atribuit unui utilizator care are deja unul, noul profil îl va înlocui pe cel vechi. Asocierea profilurilor nu afectează sesiunea curentă. Profilurile pot fi atribuite numai utilizatorilor, și nu unui *role* sau unui alt profil.

Exemplul 13.2

Se creează profilul *u_profil*, specificând parametrii în mod corespunzător.

```
CREATE PROFILE u_profil LIMIT
  SESSIONS_PER_USER          UNLIMITED
  CPU_PER_SESSION             UNLIMITED
  CPU_PER_CALL                1000
  CONNECT_TIME                 50
  LOGICAL_READS_PER_SESSION   DEFAULT
  LOGICAL_READS_PER_CALL      DEFAULT
  PRIVATE_SGA                  25K
  COMPOSITE_LIMIT              DEFAULT
  FAILED_LOGIN_ATTEMPTS        3
  PASSWORD_LIFE_TIME           50
  PASSWORD_REUSE_TIME          50
  PASSWORD_REUSE_MAX           DEFAULT
  PASSWORD_VERIFY_FUNCTION     DEFAULT
  PASSWORD_LOCK_TIME           1/24
  PASSWORD_GRACE_TIME          15;
```

- Folosind comanda *ALTER PROFILE* se pot modifica limitările de resurse ale unui profil. Pentru aceasta este necesar privilegiul sistem *ALTER PROFILE*. Orice utilizator care deține acest privilegiu poate modifica limitările profilului implicit. Inițial, nu există limitări de resurse în profilul implicit (sunt *UNLIMITED*). Pentru a preveni consumul nelimitat de resurse, administratorul pentru securitatea sistemului va trebui să modifice acest profil.
- Modificările făcute asupra unui profil nu vor afecta sesiunea curentă. Acestea vor deveni active pentru sesiunile ulterioare.

Exemplul 13.3

Se modifică profilul creat anterior, astfel încât după două încercări de conectare eşuate contul utilizatorului să se blocheze, parola să expire după 25 de zile și să nu se poată reutiliza aceeași parolă pentru cel puțin 60 de zile.

```
ALTER PROFILE u_profil LIMIT
  FAILED_LOGIN_ATTEMPTS      2
  PASSWORD_LIFE_TIME          30
  PASSWORD_REUSE_TIME         60
  PASSWORD_REUSE_MAX          UNLIMITED;
```

- Stergerea unui profil necesită privilegiul sistem *DROP PROFILE*. Comanda *SQL* folosită are următoarea sintaxă:

```
DROP PROFILE nume_profil [CASCADE];
```

- Opțiunea *CASCADE* se folosește dacă profilul este asociat mai multor utilizatori. Aceasta determină disocierea profilului respectiv de toți utilizatorii care îl aveau alocat. În mod automat, sistemul va atribui acestora profilul implicit.

13.1.4. Informații despre utilizatori și profili

- Sistemul menține o serie de vizualizări în dicționarului datelor care conțin informații despre utilizatorii bazei de date și despre profili:
 - *DBA_TS_QUOTAS* (descrie cotele spațiilor tabel pentru utilizatori);
 - *USER_PASSWORD_LIMITS* (descrie valorile parametrilor referitor la parole, setați prin comanda *CREATE PROFILE*);
 - *DBA_PROFILES* (listază toate profilurile împreună cu limitele lor);
 - *USER_RESOURCE_LIMITS* (afișează limitările de resurse pentru utilizatorul curent);
 - *RESOURCE_COST* (afișează costul fiecărei resurse);
 - *V\$SESSTAT* (listază statistici despre sesiunile utilizatorilor);
 - *V\$STATNAM* (afișează numele statisticilor listate prin vizualizarea anterioară);
 - *PROXY_USERS* (descrie utilizatorii bazei de date care își pot asuma identitatea altor utilizatori) etc.

13.2. Administrarea privilegiilor și a role-urilor

13.2.1. Privilegii

- Un privilegiu este dreptul de a executa anumite comenzi *SQL*.
- Privilegiile pot include dreptul de: conectare la baza de date, creare de tabele, selectare a linilor din tabelul altui utilizator, execuție a procedurilor stocate create de alt utilizator etc.
- Privilegiile trebuie să fie acordate utilizatorilor numai dacă sunt absolut necesare în activitatea acestora. Acordarea excesivă de privilegii poate compromite securitatea bazei de date. Privilegiile pot fi de tip sistem sau obiect.
- **Un privilegiu sistem** reprezintă dreptul de a executa anumite acțiuni asupra bazei de date sau de a grupa aceste acțiuni. De exemplu, privilegiul de a crea spații tabel sau de a șterge linii din orice tabel al bazei sunt privilegii sistem. Există peste 100 de privilegii sistem distințe.
- Privilegiile sistem pot fi clasificate astfel:
 - privilegii specifice sistemului (de exemplu, *CREATE SESSION*, *DROP TABLESPACE*, *ALTER TABLESPACE*);
 - privilegii pentru gestiunea obiectelor create de utilizator (de exemplu, *CREATE TABLE*, *SELECT VIEW*, *EXECUTE PROCEDURE*);
 - privilegii pentru gestiunea obiectelor din orice schemă (de exemplu, *CREATE ANY TABLE*, *DROP ANY INDEX*).
- Deoarece privilegiile sistem sunt puternice, se recomandă să nu se acorde utilizatorilor obișnuiți privilegii de tip *ANY* asupra tabelelor dicționarului datelor (de exemplu, *UPDATE ANY TABLE*). Pentru a securiza accesul la dicționarul datelor, parametrul de inițializare *O7_DICTIONARY_ACCESSIBILITY* trebuie să fie *FALSE*. În acest caz, accesul la obiectele schemei *SYS* este permis doar utilizatorului care deține schema *SYS* sau care se conectează ca administrator. Privilegiile sistem acordate utilizatorilor vor permite accesul la obiectele din alte scheme, dar nu la cele din schema *SYS*.
- Pentru administratorii bazei de date există două privilegii sistem speciale, *SYSOPER* și *SYSDBA*. Aceste privilegii permit accesul la instanța bazei de date, chiar dacă baza nu este deschisă. Atunci când un utilizator având privilegiul *SYSOPER* sau *SYSDBA* se conectează la baza de date îi este asociată o schemă obiectuală implicită, nu cea corespunzătoare *username*-ului său. Pentru *SYSDBA* această schemă este *SYS*, iar

pentru *SYSOPER* este *PUBLIC*. Privilegiul *SYSOPER* permite folosirea comenzilor de bază, dar fără posibilitatea de a vizualiza datele utilizatorilor (*STARTUP*, *SHUTDOWN*, *ALTER DATABASE*, *CREATE SPFILE* etc.). Privilegiul *SYSDBA* permite utilizatorului să execute orice tip de operație asupra bazei de date și a obiectelor sale.

- **Privilegiile obiect** sunt drepturi de a executa anumite acțiuni asupra unui obiect specific al schemei: tabel, vizualizare, secvență, procedură, funcție, pachet. De exemplu, *ALTER TABLE*, *DELETE TABLE*, *SELECT TABLE* și *UPDATE TABLE* sunt privilegii obiect. Unele obiecte, ca de exemplu grupările, indecșii, declanșatorii, legăturile de baze de date, nu au asociate privilegii obiect. Folosirea lor este controlată de privilegiile sistem. De exemplu, pentru a modifica o grupare utilizatorul trebuie să fie proprietarul grupării sau să dețină privilegiul sistem *ALTER ANY CLUSTER*.
- În ceea ce privește respectarea privilegiilor, obiectul schemei este echivalent cu sinonimul său. Dacă se acordă un privilegiu pentru un obiect, acesta este valabil și pentru sinonimul său și reciproc. Dacă sinonimul este eliminat, toate privilegiile obiectului au în continuare efect, chiar dacă privilegiile au fost acordate doar pentru sinonimul său.

13.2.2. Role-uri

- Un *role* este un grup de privilegii înrudite care pot fi acordate sau revocate simultan utilizatorilor sau altor *role*-uri.
- Folosirea *role*-urilor permite:
 - simplificarea administrării privilegiilor (în loc să se acorde mai multe privilegii în mod explicit unui grup de utilizatori înrudiți, se poate crea un *role* care să conțină toate privilegiile necesare și apoi să se acorde acest *role* fiecărui membru al grupului);
 - administrarea dinamică a privilegiilor (dacă privilegiile unui grup de utilizatori trebuie schimbate, modificarea lor se va face la nivelul *role*-ului care le conține și aceasta se propagă automat pentru fiecare utilizator care are asociat *role*-ul respectiv);
 - activarea selectivă a privilegiilor (*role*-urile pot fi activate sau dezactivate selectiv, astfel încât se permite un control ridicat al privilegiilor acordate utilizatorilor).

- Se pot crea aplicații care să interogheze dicționarul datelor și să activeze/dezactiveze automat unele *role-uri*, în funcție de utilizatorul care încearcă să execute aplicația respectivă.
- *Role-urile* au următoarele caracteristici:
 - pot fi acordate sau revocate utilizatorilor folosind aceleași comenzi ca și în cazul privilegiilor sistem;
 - pot cuprinde atât privilegii sistem, cât și privilegii obiect;
 - pot fi protejate prin folosirea parolelor;
 - trebuie să aibă un nume unic, diferit de conturile utilizatorilor și de celealte *role-uri* din baza de date;
 - nu sunt conținute în schema nici unui utilizator;
 - caracteristicile lor pot fi regăsite în dicționarul datelor.
- La crearea bazei de date *Oracle* se definesc automat *role-uri* predefinite. Acestea li se pot acorda sau revoca alte privilegii sau *role-uri*. Un exemplu de *role* predefinit este *CONNECT* care include privilegiile sistem *CREATE SESSION*, *ALTER SESSION*, *CREATE TABLE* etc.
- Pentru a permite utilizatorilor care nu dețin un *role DBA* accesul la vizualizările dicționarului datelor, sistemul oferă *role-urile* *DELETE_CATALOG_ROLE*, *EXECUTE_CATALOG_ROLE* și *SELECT_CATALOG_ROLE*.
- Crearea unui *role* necesită privilegiul sistem *CREATE ROLE* și se realizează prin comanda cu același nume. După creare, *role-ul* nu are privilegii sau *role-uri* asociate, acestea putând fi acordate ulterior.
- Comanda prin care se creează un *role* are următoarea sintaxă:

```
CREATE ROLE nume_role
  {NOT IDENTIFIED | IDENTIFIED tip_autorizare};
```

- Dacă este folosită clauza *NOT IDENTIFIED* atunci nu se cere nici o autorizare pentru *role-ul* respectiv. Această opțiune este implicită.
- Clauza *tip_autorizare* specifică următoarelor categorii de autorizări:
 - prin baza de date (*IDENTIFIED BY parolă*);
 - prin intermediul unei aplicații care utilizează pachete specifice (*IDENTIFIED USING [schema.]nume_pachet*);
 - externă, realizându-se prin sistemul de operare, rețea sau alte surse externe (*IDENTIFIED EXTERNALLY*);

- globală (*IDENTIFIED GLOBALLY*).
- Dacă autorizarea *role*-ului se face prin sistemul de operare, atunci informațiile pentru fiecare utilizator trebuie configurate la acest nivel. În cazul autorizării prin rețea, dacă utilizatorii se conectează la baza de date prin *Oracle Net*, atunci *role*-urile nu pot fi autorizate de sistemul de operare.
- Autorizarea globală presupune existența *role*-urilor globale. Un *role* global se definește în baza de date, dar nu poate fi acordat de la acest nivel altui utilizator sau *role*. Atunci când un utilizator global încearcă să se conecteze la bază este interrogat directorul de servicii pentru a se obține *role*-urile globale asociate.
- Schimbarea modului de autorizare a unui *role* se face folosind comanda *ALTER ROLE*. Pentru aceasta este necesar privilegiul sistem *ALTER ANY ROLE* sau un *role* cu opțiuni de administrare.
- Unui utilizator îi pot fi acordate mai multe *role*-uri. La fiecare conectare a utilizatorului este activat în mod automat un *role* implicit, chiar dacă acesta nu are asociat explicit nici un *role*. Un *role* implicit este specificat folosind clauza *DEFAULT ROLE* din comanda *ALTER USER*:

```
ALTER USER nume_utilizator DEFAULT ROLE
    {role [, role ...] |
     ALL [EXCEPT role [, role ...] ] | NONE};
```

- Folosind opțiunea *ALL*, toate *role*-urile acordate utilizatorului devin implicite, cu excepția celor care apar în clauza *EXCEPT*.
- Opțiunea *NONE* este utilizată dacă utilizatorul nu trebuie să dețină *role*-uri implicite. În acest caz, la conectare utilizatorul va beneficia doar de privilegiile care i-au fost acordate în mod explicit.
- Clauza *DEFAULT ROLE* nu poate fi folosită pentru *role*-luri:
 - care nu au fost acordate utilizatorului;
 - acordate prin intermediul altor *role*-uri;
 - administrate prin servicii externe.
- Deoarece *role*-urile trebuie inițial acordate unui utilizator și apoi declarate implicite, nu se pot seta *role*-uri implicite folosind comanda *CREATE USER*.
- Atunci când un *role* este activ, utilizatorul poate folosi privilegiile acordate prin intermediul acestuia. Dacă *role*-ul este inactiv, atunci utilizatorul poate folosi doar privilegiile care i-au fost acordate în mod explicit sau care fac parte din alte *role*-uri deținute de acesta. Activarea sau dezactivarea *role*-urilor persistă doar pentru sesiunea

currentă. În sesiunile următoare, utilizatorul va avea activate din nou *role*-urile implicate.

- Pentru activarea sau dezactivarea *role*-urilor este utilizată comanda *SET ROLE*. Aceasta are următoarea sintaxă:

```
SET ROLE {role [IDENTIFIED BY parolă]
           [, role [IDENTIFIED BY parolă]...]
           | ALL [EXCEPT role [, role ...] ] | NONE};
```

- Clauza *IDENTIFIED BY* precizează parola necesară pentru autorizarea *role*-ului.
- Folosind opțiunea *ALL* sunt activate toate *role*-urile acordate utilizatorului curent, cu excepția celor care apar în clauza *EXCEPT*.
- Opțiunea *NONE* asigură dezactivarea tuturor *role*-urilor care erau acordate utilizatorului respectiv.
- În unele cazuri este necesară suprimarea unui *role* din baza de date. Ștergerea acestuia implică suprimarea sa din toate *role*-urile cărora le-a fost acordat. Deoarece crearea obiectelor nu este dependentă de privilegiile primite prin *role*-uri, tabelele și celealte obiecte nu vor fi șterse în momentul suprimării *role*-ului. Comanda *SQL* de ștergere a unui *role* este *DROP ROLE*. Pentru folosirea acestei comenzi este necesar privilegiul sistem *DROP ANY ROLE* sau un *role* cu opțiuni de administrare.

13.2.3. Acordarea privilegiilor și a *role*-urilor

- Privilegiile și *role*-urile pot fi acordate utilizatorilor sau altor *role*-uri folosind comanda *GRANT* sau utilitarul *Oracle Enterprise Manager*.
- Nu pot acorda privilegii sistem sau *role*-uri decât utilizatorii cărora le-au fost acordate acestea cu opțiunea *WITH ADMIN OPTION* a comenzi *GRANT* sau cei care dețin privilegiul sistem *GRANT ANY ROLE*.
- Utilizatorii nu pot acorda *role*-uri autorizate global. Acordarea, respectiv revocarea unui astfel de *role* este controlată în întregime prin directorul de servicii.
- Comanda *SQL* prin care se acordă privilegii sistem sau *role*-uri unui utilizator are următoarea sintaxă:

```
GRANT {privilegiu_sistem | role} [, {privilegiu_sistem | role}...]
       TO {nume_utilizator | role | PUBLIC}
            [, {nume_utilizator | role | PUBLIC}...]
       [WITH ADMIN OPTION];
```

- Opțiunea *PUBLIC* permite acordarea privilegiilor sistem tuturor utilizatorilor bazei de date.
- Clauza *WITH ADMIN OPTION* permite utilizatorilor să acorde mai departe privilegiile sistem și *role*-urile respective altor utilizatori sau *role*-uri.
- Privilegiile obiect nu pot fi acordate împreună cu privilegii sistem sau cu *role*-uri, în cadrul aceleiași comenzi *GRANT*. Acordarea de privilegii obiect utilizatorilor sau *role*-urilor poate fi făcută de proprietarul obiectului sau de un utilizator căruia i s-a acordat privilegiul obiect respectiv, cu opțiunea *WITH GRANT OPTION*.
- Comanda folosită pentru acordarea unui privilegiu obiect este:

```
GRANT {privilegiu_object [ (listă_coloane) ]
        [, privilegiu_object [ (listă_coloane)...]
         | ALL [PRIVILEGES] ]
ON [nume_schemă.]obiect
TO {nume_utilizator | role | PUBLIC}
    [, {nume_utilizator | role | PUBLIC}...]
[WITH GRANT OPTION];
```

- Prin opțiunea *listă_coloane* se precizează coloanele unui tabel sau ale unei vizualizări pentru care se acordă privilegiul.
- Această opțiune poate fi folosită atunci când sunt acordate privilegii obiect de tip *INSERT*, *REFERENCES* sau *UPDATE*.
- Opțiunea *ALL* permite acordarea tuturor privilegiilor obiect pentru care utilizatorul ce inițiază comanda *GRANT* are opțiunea *WITH GRANT OPTION*. Clauza *ON [nume_schemă.]obiect* precizează obiectul relativ la care este acordat privilegiul. Opțiunea *PUBLIC* permite acordarea privilegiilor obiect tuturor utilizatorilor bazei de date.
- Clauza *WITH GRANT OPTION* permite utilizatorilor să acorde privilegii obiect altor utilizatori sau *role*-uri. Această clauză nu poate fi utilizată atunci când privilegiul obiect este acordat unui *role*.
- În ceea ce privește privilegiile referitoare la comenzi *LMD*, acestea sunt acordate pentru operațiile *DELETE*, *INSERT*, *SELECT* și *UPDATE* asupra unui tabel sau unei vizualizări, doar utilizatorilor sau *role*-urilor care trebuie să interogheze sau să prelucreze datele respective.
- Privilegiile *INSERT* și *UPDATE* se pot restricționa pentru anumite coloane ale tabelului. În cazul unui privilegiu *INSERT* restricționat pentru anumite coloane, inserarea unei linii permite inserarea de valori doar pentru coloanele accesibile.

- Coloanele restricționate primesc valori implicate sau *null*. În cazul unui privilegiu *UPDATE* restricționat, vor putea fi modificate doar coloanele pentru care utilizatorul are acest drept.
- Privilegiile *ALTER*, *INDEX* și *REFERENCES* permit executarea de operații *LDD* asupra unui tabel. Un utilizator care încearcă să execute o comandă *LDD* trebuie să aibă anumite privilegii sistem sau obiect. De exemplu, pentru a crea un declanșator asupra unui tabel, utilizatorul trebuie să dețină privilegiul obiect *ALTER TABLE* și privilegiul sistem *CREATE TRIGGER*. Privilegiul *REFERENCES* poate fi acordat unei anumite coloane a unui tabel. Astfel, tabelul respectiv este folosit ca tabel „părinte“ pentru orice cheie externă care trebuie creată.

Exemplul 13.4

- a) Se acordă utilizatorului *u1* dreptul de a porni și opri baza de date, fără a-i se permite crearea unei baze de date.

```
GRANT SYSOPER TO u1;
```

- b) Se acordă utilizatorilor *u2* și *u3* privilegiile obiect *SELECT* și *INSERT* asupra coloanelor tabelului *produse*, cu posibilitatea ca aceștia să acorde privilegiile și altor utilizatori sau *role-uri*.

```
GRANT SELECT, INSERT ON produse TO u2, u3
WITH GRANT OPTION;
```

- c) Se acordă utilizatorului *u4* privilegiul obiect *INSERT* doar pentru coloanele *id_produs* și *denumire*.

```
GRANT INSERT (id_produs, denumire) ON produse TO u4;
```

Exemplul 13.5

- a) Se creează un *role* numit *u_role*, care să permită utilizatorilor să creeze tabele și vizualizări.

```
CREATE ROLE u_role;
GRANT CREATE TABLE, CREATE VIEW TO u_role;
```

- b) Se atribuie *role-ul* creat anterior și *role-ul* predefinit *RESOURCE*, utilizatorului *u5*. *Role-ul RESOURCE* trebuie să fie activat în mod automat atunci când utilizatorul se conectează.

```
GRANT user_role, RESOURCE TO u5;
ALTER USER u1
DEFAULT ROLE RESOURCE;
```

- c) Se permite utilizatorului *u5* să consulte vizualizările dicționarului.

```
GRANT SELECT_CATALOG_ROLE TO u5;
```

- d) Se afișează segmentele *undo* care sunt utilizate de instanța curentă (se presupune că utilizatorul curent este *u5*).

```
SET ROLE SELECT_CATALOG_ROLE;
SELECT SEGMENT_NAME
FROM   DBA_ROLLBACK_SEGS
WHERE  STATUS = 'ONLINE';
```

13.2.4. Revocarea privilegiilor și a *role*-urilor

- Pentru a revoca privilegii sau *role*-uri se poate folosi comanda *REVOKE* sau utilitarul *Oracle Enterprise Manager*.
- Un utilizator care are opțiunea de administrare, de acordare a unui privilegiu sau *role*, le poate revoca pe acestea oricărui *role* sau utilizator al bazei de date. Un utilizator care deține privilegiul *GRANT ANY ROLE* poate revoca orice *role*.
- Sintaxa generală a comenzii de revocare a unui privilegiu sistem sau *role* este următoarea:

```
REVOKE {privilegiu_sistem | role}
      [, {privilegiu_sistem | role}...]
  FROM    {nume_utilizator | role | PUBLIC};
```

- Opțiunea *PUBLIC* permite revocarea privilegiilor sistem sau a *role*-urilor tuturor utilizatorilor bazei de date.
- Sintaxa generală a comenzi de revocare a unui privilegiu obiect este:

```
REVOKE {privilegiu_object [, privilegiu_object ...]
          | ALL [PRIVILEGES] }
  ON [nume_schemă.]objec
  FROM {nume_utilizator | role | PUBLIC}
        [, {nume_utilizator | role | PUBLIC}...]
  [CASCADE CONSTRAINTS];
```

- Opțiunea *ALL* permite revocarea tuturor privilegiilor obiect acordate utilizatorului.
- Prin clauza *ON* se identifică obiectul la care se referă privilegiul ce trebuie revocat.
- Clauza *FROM* identifică utilizatorii sau *role*-urile pentru care este revocat privilegiul obiect.
- Opțiunea *PUBLIC* determină revocarea unor privilegii obiect tuturor utilizatorilor bazei de date.
- Clauza *CASCADE CONSTRAINTS* permite suprimarea constrângerilor de integritate referențială definite folosindu-se privilegiile *REFERENCES* sau *ALL*.

- Definițiile obiectelor care depind de privilegii *LMD* sistem sau obiect pot fi afectate dacă privilegiile respective sunt revocate.
 - De exemplu, dacă privilegiul sistem *SELECT ANY TABLE* a fost acordat unui utilizator care apoi a creat proceduri sau vizualizări ce folosesc un tabel din altă schemă, atunci revocarea acestui privilegiu determină invalidarea procedurilor sau vizualizărilor respective.
 - De asemenea, dacă o procedură include comenzi *SQL* prin care sunt selectate datele unui tabel și este revocat privilegiul obiect *SELECT* asupra tabelului respectiv, atunci procedura nu se mai execută cu succes.
- Definițiile obiectelor pentru care sunt necesare privilegiile obiect *ALTER* și *INDEX* nu sunt afectate dacă aceste privilegii sunt revocate.
 - De exemplu, dacă privilegiul *INDEX* este revocat unui utilizator care a creat un index asupra unui tabel al altui utilizator, indexul respectiv va continua să existe și după revocare.
- Revocarea unui privilegiu obiect poate determina efectul de revocare în cascadă a acestuia.
 - De exemplu, dacă utilizatorului *u1* i se acordă privilegiul obiect *SELECT* asupra unui tabel, cu opțiunea *WITH GRANT OPTION*, iar acesta îl acordă utilizatorului *u2*, atunci revocarea privilegiului utilizatorului *u1* va determina automat revocarea acestui privilegiu și pentru utilizatorul *u2*.

13.2.5. Informații despre privilegii și role-uri

- Sistemul menține o serie de vizualizări în dicționarului datelor, care conțin informații despre privilegii și *role-uri*:
 - *DBA_SYS_PRIVS* (privilegiile sistem acordate utilizatorilor sau altor *role-uri*);
 - *SESSION_PRIVS* (privilegiile sistem și obiect disponibile utilizatorului în sesiunea curentă);
 - *DBA_TAB_PRIVS* (privilegiile acordate asupra obiectelor bazei);
 - *DBA_COL_PRIVS* (coloanele obiectelor care sunt referite în privilegii);
 - *DBA_ROLES* (*role-urile definite* în baza de date);
 - *DBA_ROLES_PRIVS* (*role-urile acordate* utilizatorilor sau altor *role-uri*);
 - *ROLE_ROL_PRIVS* (*role-urile acordate* altor *role-uri*);
 - *SESSION_ROLES* (*role-urile active* pentru utilizator în sesiunea curentă);

- *ROLE_SYS_PRIVS* (privilegiile sistem acordate *role*-urilor);
- *ROLE_TAB_PRIVS* (privilegiile obiect acordate *role*-urilor) etc.

13.4. Auditarea

- Auditarea presupune monitorizarea și înregistrarea selectivă a acțiunilor utilizatorilor unei baze de date. Auditarea este un proces folosit pentru:
 - investigarea activităților suspecte (de exemplu, dacă un utilizator neautorizat șterge date din tabele, administratorul pentru securitate trebuie să verifice toate conexiunile la bază și operațiile de ștergere a liniilor din tabelele bazei de date pentru a-l identifica);
 - monitorizarea și gruparea datelor pe categorii de activități specifice în cadrul bazei de date (de exemplu, administratorul bazei trebuie să colecteze statistici despre tabelele care au fost modificate, numărul de operații *I/O* executate, durata medie a unei sesiuni, privilegiile sistem folosite, numărul de utilizatori care s-au conectat simultan la diferite intervale de timp etc.).

13.4.1. Opțiuni de audit

- Controlul acțiunilor întreprinse asupra elementelor unei baze de date este realizat prin comanda *AUDIT*, iar rezultatele verificărilor sunt înregistrate într-un tabel (*AUD\$*) al dicționarului datelor, cunoscut sub denumirea de *audit trail*. Pentru a preveni completarea totală a spațiului tabel asociat, periodic se șterg înregistrări din tabelul *AUD\$*. Se recomandă ca tabelul *AUD\$* să nu aparțină spațiului tabel *SYSTEM*. De asemenea, acesta trebuie protejat împotriva accesului neautorizat.
- Sintaxa simplificată a comenzi *AUDIT* este următoarea:

```
AUDIT { comandă_SQL [,comandă_SQL ...]
        | privilegiu_sistem, [, privilegiu_sistem ...]
          [BY utilizator]
          | ON [nume_schemă.]obiect}
[BY {SESSION | ACCESS} ]
[WHENEVER [NOT] SUCCESSFUL];
```

- Sistemul *Oracle* suportă trei tipuri generale de audit, la nivel de comandă, privilegiu sau obiect ale unei scheme. Operațiile de audit pot fi definite la nivel de sesiune sau acces. Pentru a dezactiva opțiunile de audit inițiate printr-o comandă *AUDIT* se utilizează comanda *NOAUDIT* asupra celor opțiuni.

- Auditul la nivel de comandă se referă la verificări selective asupra comenziilor *SQL*, care se găsesc în una dintre următoarele două categorii:
 - comenzi *LDD* relativ la un anumit obiect al schemei (de exemplu, *AUDIT TABLE* verifică toate comenziile *CREATE* și *DROP TABLE*);
 - comenzi *LMD* referitoare la anumite obiecte ale schemei, dar fără să se specifice numele acestora (de exemplu, *AUDIT SELECT TABLE* controlează toate operațiile *SELECT* asupra tabelelor și vizualizărilor).
- Sistemul *Oracle* permite verificarea selectivă a execuțiilor comenziilor *SQL*. Execuțiile eşuate se pot verifica doar dacă structura *SQL* este validă, iar eşuarea s-a produs din cauza referirii unui obiect inexistent sau a lipsei unei autorizații adecvate. Comenziile care eşuează pentru că nu au fost valide, nu pot fi verificate.
- Pentru verificarea exclusivă a execuțiilor cu succes se utilizează clauza *WHENEVER SUCCESSFUL* a comenzi *AUDIT*, iar pentru verificarea doar a execuțiilor eşuate clauza *WHENEVER NOT SUCCESSFUL* a aceleiași comenzi. Dacă nu este folosită nici una dintre clauzele menționate mai sus, verificarea se realizează în ambele cazuri.
- Auditul la nivel de privilegii asigură verificarea selectivă a comenziilor care necesită privilegii sistem.
 - De exemplu, verificarea privilegiului sistem *SELECT ANY TABLE* permite monitorizarea comenziilor care se execută folosind acest privilegiu.
- Privilegiile obiect sunt verificate înaintea privilegiilor sistem. Dacă sunt setate opțiuni de audit pentru comenzi și privilegii similare, atunci este generată o singură înregistrare pentru audit.
 - De exemplu, dacă comanda *CREATE TABLE* și privilegiul sistem *CREATE TABLE* sunt ambele verificate, atunci se generează o singură înregistrare de audit, de fiecare dată când este creat un tabel.
- Auditul la nivel de obiect al schemei constă în verificarea comenziilor *LMD* specific și a comenziilor *GRANT* sau *REVOKE* pentru obiectele schemei.
- Se pot verifica acele comenzi care referă tabele, vizualizări, secvențe, proceduri stocate, funcții, pachete. P
 - Procedurile din pachete nu pot fi verificate individual.
 - De asemenea, comenziile care referă grupări, indecsi sau sinonime, nu pot fi verificate direct.

- Totuși, se poate verifica accesul la aceste obiecte în mod indirect, prin verificarea operațiilor care afectează tabelul de bază.

Exemplul 13.6

Se consideră următoarea secvență de comenzi *SQL*:

```
AUDIT SELECT ON produse;

CREATE VIEW produse_categorii AS
    SELECT p.id_produs, p.denumire, c.denumire, c.nivel
    FROM produse p, categorii c
    WHERE p.id_produs = c.id_produs;

AUDIT SELECT ON produse_categorii;

SELECT * FROM produse_categorii;
```

Ca urmare a interogării vizualizării *produse_categorii* sunt generate două înregistrări de auditare, corespunzătoare interogării vizualizării *produse_categorii*, respectiv tabelului de bază *produse*. Interrogarea tabelului de bază *categorii* nu generează înregistrări de verificare, deoarece opțiunea de *AUDIT SELECT* pentru acest tabel nu este activată.

- Opțiunile de audit pot fi specificate la nivel de sesiune sau acces, folosind clauzele *BY SESSION*, respectiv *BY ACCESS* ale comenzi *AUDIT*.
- Auditarea la nivel de sesiune are ca rezultat inserarea unei singure înregistrări în tabelul *AUD\$* pentru fiecare utilizator și obiect al schemei, în timpul sesiunii care include o acțiune auditată.
- Acțiunea de audit la nivel de acces presupune inserarea unei înregistrări în tabelul *AUD\$* pentru fiecare execuție a unei operații care este monitorizată.
 - Operațiile de audit la nivel de comenzi sau de privilegii, aplicate pentru comenzi *LDD* nu pot fi precizate decât la nivel de acces.
- Opțiunile de audit la nivel de comenzi sau privilegii permit ca monitorizarea să se realizeze pentru un anumit utilizator sau pentru un grup de utilizatori.
 - Prin utilizarea operației de audit asupra unu grup de utilizatori se poate minimiza numărul de înregistrări din tabelul *AUD\$*.

Exemplul 13.7

- a) Se dispune auditarea comenzi *SELECT TABLE* folosite de utilizatorii *u1* și *u2*.

```
AUDIT SELECT TABLE
BY U1, U2;
```

- b) Se presupune că:

- auditarea se realizează la nivel de sesiune pentru toate comenzi *SELECT TABLE*;
- utilizatorul *u1* se conectează la baza de date, execută trei comenzi *SELECT* asupra tabelului *produse* și apoi se deconectează;
- utilizatorul *u2* se conectează la baza de date, execută două comenzi *SELECT* asupra tabelului *categorii* și apoi se deconectează.

În tabelul *AUD\$* se vor insera două înregistrări, câte una pentru fiecare sesiune ce conține comenzi auditate.

Dacă același utilizator execută interogări asupra unor tabele distințe, atunci în tabelul *AUD\$* se vor insera mai multe înregistrări, câte una pentru fiecare tabel.

- c) Se presupune că:

- auditarea se realizează la nivel de acces pentru toate comenzi *SELECT TABLE*;
- utilizatorul *u1* se conectează la baza de date, execută trei comenzi *SELECT* asupra tabelului *produse* și apoi se deconectează;
- utilizatorul *u2* se conectează la baza de date, execută patru comenzi *SELECT* asupra tabelului *produse* și apoi se deconectează.

În tabelul *AUD\$* se vor insera șapte înregistrări, câte una pentru fiecare comandă *SELECT*.

13.4.2. Mecanisme pentru audit

- Înregistrarea informațiilor pentru audit poate fi activată sau dezactivată. Dacă opțiunea de auditare este activă, atunci se permite oricărui utilizator autorizat să specifiche opțiuni pentru audit, rezervând administratorului pentru securitate dreptul de control asupra înregistrării informațiilor de audit.
- Informațiile de audit conțin numele contului, identificatorul sesiunii, identificatorul mașinii *client*, numele schemei de obiecte care a fost accesată, operațiile executate sau inițiate, codul rezultat în urma compilării operației, privilegiile sistem folosite etc.
- Activarea sau dezactivarea operațiilor de audit pentru o instanță se realizează prin parametrul de initializare *AUDIT_TRAIL*. Acest parametru poate avea una din următoarele valori:

- *TRUE* sau *DB* (se activează operațiile de audit și informațiile despre acestea sunt înregistrate în tabelul *AUD\$* al bazei de date);
 - *OS* (se activează operațiile de audit și informațiile despre acestea sunt înregistrate în tabelul *audit trail* al sistemului de operare);
 - *FALSE* sau *NONE* (se dezactivează operațiile de audit).
- Atunci când opțiunea de audit devine activă se generează verificări asupra înregistrărilor pe durata derulării fazelor de execuție a comenzi. Comenzile *SQL* din interiorul unităților de program *PL/SQL* sunt verificate individual.
 - Chiar dacă auditul bazei de date nu este activat, sistemul *Oracle* înregistrează întotdeauna în tabelul *audit trail* al sistemului de operare, informații despre următoarele acțiuni care au loc asupra bazei:
 - pornirea instanței (utilizatorul care a pornit instanța, identificatorul mașinii *client*, data și momentul de timp etc.);
 - oprirea instanței (utilizatorul care a închis instanța, identificatorul mașinii *client*, data și momentul de timp etc.);
 - sesiunile utilizatorilor cu privilegii de administrare.
 - Opțiunile de audit la nivel de comenzi *SQL* sau privilegii au efect pe timpul conexiunii unui utilizator la baza de date (pe toată durata sesiunii). Schimbările opțiunilor de audit la nivel de comenzi și privilegii nu afectează sesiunea curentă. Modificările au efect numai asupra noilor sesiuni inițiate. În schimb, modificările opțiunilor de audit la nivel de obiecte ale schemei au efect imediat, inclusiv pentru sesiunea curentă.
 - Operația de audit a bazei de date nu permite monitorizări la nivel de coloană. Dacă acest lucru este necesar, atunci se pot folosi rutine speciale pentru auditare (de exemplu, proceduri stocate, declanșatori). Baza de date permite mecanisme de audit integrate, astfel încât monitorizările pot fi realizate automat, fără intervenția utilizatorilor.
 - Sistemul *Oracle* oferă procedee de auditare granulară (*fine-grained auditing*) prin care se permite monitorizarea accesului la date. Administratorul pentru securitate poate utiliza pachetul *DBMS_FGA* cu scopul de a crea politici de audit pentru un anumit tabel.
 - Dacă fiecare linie returnată de o interogare îndeplinește condițiile de audit, atunci se înregistrează în tabelul *AUD\$* un eveniment de audit care include *username*-ul, codul *SQL*, variabilele de legătură, identificatorul sesiunii, momentul de timp etc.

- Opțional, administratorii pot defini modalități de prelucrare și procesare a evenimentelor de audit. De exemplu, la prelucrarea unui eveniment de audit se pot trimite mesaje de avertizare administratorului. Toate informațiile relevante sunt livrate printr-un mecanism asemănător declanșatorilor.

13.4.3. Informații despre audit

- Informații despre opțiunile de audit și evenimentele de audit înregistrate se pot obține consultând dicționarul datelor. Dintre cele mai importante vizualizări referitoare la auditul bazei de date se remarcă:
 - *ALL_DEF_AUDIT_OPTS* (opțiunile implicite pentru audit);
 - *DBA_STMT_AUDIT_OPTS* (opțiunile de audit la nivel de comandă);
 - *DBA_PRIV_AUDIT_OPTS* (opțiunile de audit la nivel de privilegiu);
 - *DBA_OBJ_AUDIT_OPTS* (opțiunile de audit la nivel de obiect);
 - *DBA_AUDIT_TRAIL* (toate înregistrările din *audit trail*);
 - *DBA_AUDIT_OBJECT* (înregistrările din *audit trail* referitoare la obiectele schemei);
 - *DBA_AUDIT_SESSION* (înregistrările din *audit trail* pentru operațiile de audit la nivel de sesiune);
 - *DBA_AUDIT_STATEMENT* (înregistrările din *audit trail* referitoare la comenzi etc.

Bibliografie

1. *Programare avansată în Oracle9i*, I. Popescu, A. Alecu, L. Velcescu, G. Florea (Mihai), Ed. Tehnică (2004)
2. *Oracle Database PL/SQL Language Reference 11g Release 2*, Oracle Online Documentation (2012)
3. *Oracle Database SQL Language Reference 11g Release 2*, Oracle Online Documentation (2012)
4. *Oracle Database 11g: PL/SQL Fundamentals, Student Guide*, Oracle University (2009)