SQL recapitulare

SQL*

1. S-au introdus clauzele noi pentru instructiunile: SELECT, UPDATE, DELETE, INSERT, folosite în PL/SQL.
   De exemplu:
   SELECT … INTO …

   INSERT INTO …
   (…)
   VALUES (...)
   RETURNING … INTO variabile de legatura

   UPDATE … RETURNING… INTO variabile de legatura

   DELETE .. RETERNING… INTO variabile de legatura

2. Se prezintă utilitarul **SQL***

SQL

SQL (*Structured Query Language*) este un limbaj **neprocedural** pentru interogarea şi prelucrarea informațiilor din baza de date. De asemenea, *SQL* este un limbaj **declarativ**, asfel încât este suficient ca utilizatorul doar să descrie ceea ce trebuie obținut, fără a indica modul în care se ajunge la rezultat. Compilatorul limbajului *SQL* generează automat o procedură care accesează baza de date şi execută comanda dorită.

SQL permite atât definirea, prelucrarea şi interogarea datelor, cît şi controlul accesului la acestea. Comenzile *SQL* pot fi integrate în programe scrise în alte limbaje, de exemplu *Cobol*, *C*, *C++*, *Java* etc.

Principalele aspecte prin care *SQL* diferă de limbajele de programare tradiționale sunt următoarele:

- asigură accesarea automată a datelor;
- operează asupra unor mulțimi de date, şi nu asupra elementelor individuale;
- permite programarea la nivel logic, necesitatea de a considera detaliile

implementării fiind redusă.

Oracle SQL include extensii ale limbajului *SQL* standard *ANSI/ISO*, iar instrumentele şi aplicaţiile *Oracle* furnizează instrucţiuni suplimentare.

Utilitarele *SQL*Plus* şi *Oracle Enterprise Manager* permit atât executarea instrucţiunilor limbajului *SQL* standard asupra unei baze de date *Oracle*, cât şi a instrucţiunilor sau funcţiilor suplimentare disponibile.

Deşi unele utilitare şi aplicaţii *Oracle* simplifică sau maschează utilizarea *SQL*, toate operaţiile asupra bazei de date sunt realizate folosind acest limbaj.

Caracteristicile limbajului *SQL* pot fi sintetizate astfel:

- este abordabil de diferite categorii de utilizatori, inclusiv de aceia care au puţină experienţă în programare;
- este un limbaj neprocedural, de comunicare cu *server*-ul *Oracle*;
- reduce timpul necesar creării şi întreţinerii aplicaţiilor de baze de date

Consideraţii generale

În funcţie de tipul acţiunii pe care o realizează, instrucţiunile *SQL* se împart în mai multe categorii. Datorită importanţei pe care o au comenzile componente, unele dintre aceste categorii sunt evidenţiate ca limbaje în cadrul *SQL*, şi anume:

- limbajul de definire a datelor (*LDD*);
- limbajul de interogare a datelor (*LQD*);
- limbajul de prelucrare a datelor (*LMD*);
- limbajul de control al datelor (*LCD*).

Pe lângă comenzile care alcătuiesc aceste limbaje, *SQL* cuprinde:

- instrucţiuni pentru controlul sesiunii;
- instrucţiuni pentru controlul sistemului;
- instrucţiuni *SQL* încapsulate.

Limbajul de definire a datelor

În general, instrucţiunile *LDD* sunt utilizate pentru definirea structurii corespunzătoare obiectelor unei scheme. Aceste comenzi permit:

- crearea, modificarea şi suprimarea obiectelor unei scheme şi a altor obiecte ale bazei de date, inclusiv baza însăşi şi utilizatorii acesteia

(*CREATE*, *ALTER*, *DROP*);

- modificarea numelor obiectelor unei scheme (*RENAME*);
- ştergerea datelor din obiectele unei scheme, fără suprimarea structurii obiectelor respective (*TRUNCATE*);

Implicit, o instrucțiune *LDD* **permanentizează** efectul tuturor instrucțiunilor precedente şi marchează începutul unei noi tranzacții. Odată cu lansarea unei instrucțiuni *LDD*, sistemul *Oracle* declanşează implicit instrucțiuni *SQL* care modifică informația din dicționarul datelor.

Limbajul de interogare a datelor (LQD)

Exista o singura instrucțiune a limbajului LQD folosită pentru

- regăsirea datelor dintr-unul sau mai multe tabele (*SELECT*)

Limbajul de prelucrare a datelor

Instrucțiunile *LMD* sunt utile pentru interogarea şi prelucrarea datelor din obiectele unei scheme. Aceste instrucțiuni permit:

- adăugarea de înregistrări în tabele sau vizualizări (*INSERT*);
- modificarea valorilor unor coloane din înregistrările existente în tabele sau vizualizări (*UPDATE*);
- adăugarea sau actualizarea condiționată a înregistrărilor în tabele sau vizualizări (*MERGE*);
- suprimarea de înregistrări din tabele sau vizualizări (*DELETE*);

Limbajul de control al datelor

Instrucțiunile *LCD* gestionează modificările efectuate de către comenzile *LMD* şi grupează aceste comenzi în unități logice, numite tranzacții. Aceste instrucțiuni permit:

- permanentizarea modificărilor unei tranzacții (*COMMIT*);
- anularea modificărilor dintr-o tranzacție fie în întregime, fie începând de la un punct intermediar (*ROLLBACK*);
- definirea unui punct intermediar până la care tranzacția poate fi anulată (*SAVEPOINT*);
- stabilirea de proprietăți ale tranzacției (*SET TRANSACTION TO*).

Cursoare

Un **cursor reprezintă o zonă de memorie** în care se reține o instrucțiune *SQL* analizată şi informații utile procesării acesteia.

Sistemul *Oracle* gestionează în mod automat cursoarele. În dezvoltarea unei aplicații, un cursor constituie o resursă disponibilă care poate fi utilizată pentru analizarea sintactică şi semantică explicită a instrucțiunilor *SQL* încapsulate în aplicație.

Fiecare sesiune poate deschide mai multe cursoare, până la limita stabilită de parametrul de inițializare *OPEN_CURSORS*. Pentru a elibera memoria sistemului, se recomandă ca aplicațiile să prevadă închiderea cursoarelor care nu mai sunt necesare.

Execuția unui cursor plasează rezultatul cererii asociate într-o mulțime de linii (mulțime rezultat), care pot fi regăsite secvențial sau nesecvențial.

Cursoarele *scrollable* sunt cele pentru care operațiile *LMD* şi de regăsire **nu** trebuie să se desfăşoare secvențial, într-un singur sens (de la început către sfârşit). Există interfețe care permit regăsirea de linii recuperate anterior, recuperarea liniei *n* din mulțimea rezultat sau recuperarea celei de-a *n*-a linii de la poziția curentă.

Zona *SQL* partajată

Sistemul *Oracle* sesizează automat situația în care aplicațiile lansează instrucțiuni *SQL* similare. Zona *SQL* utilizată pentru procesarea unei instrucțiuni la prima sa apariție este partajată, ceea ce înseamnă că poate fi utilizată pentru procesarea aparițiilor ulterioare ale aceleiaşi instrucțiuni. Pentru o anumită instrucțiune poate exista o singură zonă *SQL* partajată. Aceste zone pot fi utilizate de către orice proces *Oracle*. Partajarea zonelor *SQL* reduce utilizarea memoriei pe *server*-ul bazei de date.

Procesarea instrucțiunilor *SQL*

În general, procesarea oricărei instrucțiuni *SQL* presupune următoarele etape: crearea unui cursor, analiza instrucțiunii, legarea variabilelor, executarea instrucțiunii şi închiderea cursorului.

Interogările solicită câteva etape suplimentare: descrierea rezultatelor, definirea modului de prezentare a acestora şi recuperarea liniilor selectate.

Un cursor este creat independent de instrucțiunea *SQL*. El este generat automat, în aşteptarea unei instrucțiuni *SQL* căreia să îi fie asociat.

În timpul analizei, instrucțiunea este transferată de la procesul utilizator la sistemul *Oracle*.

Sistemul *Oracle* analizează instrucțiunea doar dacă nu există deja o zonă

SQL partajată în *library cache*, zonă care să conțină reprezentarea instrucțiunii. În caz contrar, sistemul analizează instrucțiunea şi generează reprezentarea corespunzătoare, căreia procesul utilizator îi alocă o zonă partajată *SQL* în *library cache*. După asocierea la o zonă partajată *SQL*, o instrucțiune poate fi executată în mod repetat fără a fi necesară reanalizarea acesteia.

În etapa de analiză sunt identificate acele erori care pot fi depistate înaintea execuției instrucțiunii. Unele erori, cum ar fi cele apărute în urma conversiei datelor sau nerespectării constrângerilor de integritate, pot fi depistate şi raportate doar în faza de execuție a instrucțiunii.

Descrierea rezultatelor unei cereri determină caracteristicile rezultatului (tipuri de date, dimensiuni, nume). Dacă este necesar, sistemul efectuează conversii implicite ale valorilor returnate.

O instrucțiune poate face referință la variabile ale căror valori trebuie cunoscute în vederea execuției. Procesul obținerii acestor valori are loc în etapa de legare a variabilelor (*binding variables*).

Execuția unei instrucțiuni *UPDATE* sau *DELETE* determină blocarea liniilor afectate de aceasta. Scopul unei astfel de operații este asigurarea integrității datelor. Liniile sunt deblocate la permanentizarea sau anularea tranzacției corespunzătoare comenzii respective. Instrucțiunile *SELECT* şi *INSERT* nu determină blocări.

Etapa de recuperare a liniilor unei cereri presupune selectarea şi, eventual, ordonarea lor. Procesarea oricărei instrucțiuni *SQL* se finalizează prin închiderea cursorului asociat, adică eliberarea zonei de memorie utilizate în scopul prelucrării.

Comentarii

Comentariile pot fi asociate instrucțiunilor *SQL* sau obiectelor schemei. Ele nu afectează execuția instrucțiunilor *SQL*, dar permit ca aplicația să fie mai uşor de citit şi întreținut.

Într-o instrucțiune, un comentariu poate apărea între orice cuvinte cheie, parametri sau semne de punctuație. Includerea unui comentariu se poate efectua în două moduri.

- Textul comentat este încadrat de caracterele „/*“ şi „*/“ şi poate ocupa mai multe linii. Nu este necesară separarea de text a acestor caractere, printr-un spațiu sau o linie liberă.
- Comentariul începe cu secvența „--“, urmată de textul propriu-zis. În acest caz, comentariul se termină la sfârşitul liniei.

Limbajul de prelucrare a datelor

SQL furnizează comenzi ce permit consultarea (*SELECT*) şi actualizarea (*INSERT*, *UPDATE*, *DELETE*, *MERGE*) conţinutului bazei de date. Aceste comenzi definesc limbajul de prelucrare a datelor (*LMD*).

Comenzile limbajului *LMD* pot fi:

- formulate direct, utilizând interfaţa *SQL*PLUS* ;
- utilizate în utilitare ale sistemului *ORACLE*;
- încapsulate într-un program *PL/SQL* ;
- încapsulate într-un program scris în limbaj gazdă.

În funcţie de momentul în care se doreşte realizarea actualizărilor asupra bazei de date, utilizatorul poate folosi una din următoarele comenzi:

- *SET AUTOCOMMIT ON* – schimbările se efectuează imediat;
- *SET AUTOCOMMIT OFF* – schimbările sunt păstrate într-un *buffer* până la execuţia uneia din comenzile:
  – *COMMIT*, care are rolul de a permanentiza schimbările efectuate;
  – *ROLLBACK*, care determină renunţarea la schimbările realizate.

Limbajul de prelucrare a datelor face parte din nucleul limbajului *SQL* şi conţine mecanisme pentru interogarea şi reactualizarea obiectelor unei scheme a bazei de date.

Pentru tabele şi vizualizări, instrucţiunile *LMD* permit:

- adăugarea de noi înregistrări (*INSERT*) explicit sau ca rezultat al unor interogări;
- modificarea valorilor coloanelor din înregistrările existente (*UPDATE*);
- suprimarea de înregistrări (*DELETE*).

De asemenea, *LMD* cuprinde instrucţiuni care determină:

- adăugarea sau modificarea condiţionată de înregistrări (*MERGE*);
- vizualizarea planului de execuţie propus de către optimizor pentru o instrucţiune *SQL* (*EXPLAIN PLAN*);
- blocarea unui tabel, limitând temporar accesul altor utilizatori la acesta (*LOCK TABLE*).

Comanda *SELECT*

Una dintre cele mai importante comenzi ale limbajului de prelucrare a datelor este *SELECT*. Cu ajutorul ei pot fi extrase submulțimi de valori atât pe verticală (coloane), cât şi pe orizontală (linii) din unul sau mai multe tabele.
Sintaxa comenzii este simplă, apropiată de limbajul natural.

SELECT	[*ALL* | *DISTINCT*] {* | *listă de atribute selectate* | *expr AS alias*}
INTO	{ *listă de variabile declarate/variabile de legatura}*
FROM	{ [*schema.*]{*tabel* [*PARTITION* (*partition_name*)] | [*THE*] (*subquery*)} [*alias_tabel*] }
[*WHERE*	*condiție*]
[*START WITH*	*condiție*]
[*CONNECT BY*	*condiție*]
[*GROUP BY*	*listă de expresii*
[*HAVING*	*condiție*]]
[*ORDER BY*	{*expresie* | *poziție* | *c_alias*} [*ASC* | *DESC*]]
[*FOR UPDATE*	[*OF* [*schema.*]{*table* | *view*}.*coloană*] [*{***WAIT** n | **NOWAIT***}*]

Prezența clauzelor *SELECT* şi *FROM* este obligatorie deoarece acestea specifică coloanele selectate, respectiv tabelele din care se vor extrage datele.

Prezența clauzei INTO indică variabilele declarate sau variabile de legatură, care păstreaza valorile selectate în vederea recuperării lor în blocuri PL/SQL.

Tabelele specificate în clauza FROM pot fi urmate de un alias, care va reprezenta numele folosit pentru referirea tabelului respectiv în cadrul instrucțiunii.

Eliminarea duplicatelor se poate realiza folosind clauza *DISTINCT*. Dacă nu se specifică parametrul *DISTINCT*, parametrul *ALL* este implicit şi are ca efect afişarea dublurilor.

Simbolul "*" permite selectarea tuturor atributelor din tabelele asupra cărora se execută cererea. Atributele sau expresiile din lista clauzei *SELECT* pot conține *alias*-uri, care vor reprezenta numele câmpurilor respective în cadrul tabelului furnizat ca rezultat de instrucțiunea *SELECT*.

Clauza *WHERE* poate fi folosită pentru a impune anumite condiții liniilor din care se vor extrage atributele specificate în clauza *SELECT*.

Clauza *GROUP BY* grupează înregistrările după anumite câmpuri; în cazul prezenței acestei clauze, clauza *HAVING* poate impune restricții suplimentare asupra rezultatului final.

Ordonarea înregistrărilor se poate face cu ajutorul clauzei *ORDER BY*. Cu ajutorul parametrilor *ASC* şi *DESC* se poate specifica ordonarea crescătoare, respectiv descrescătoare a înregistrărilor. Pentru o secvență crescătoare valorile *null* sunt afişate ultimele. Dacă nu se face nici o specificație, atunci ordinea de returnare este la latitudinea *server*-ului.

Clauza *FOR UPDATE* permite blocarea coloanei (coloanelor) înainte de a actualiza sau şterge înregistrări din tabelele bazei de date. Prin folosirea clauzei *NOWAIT* se va genera o excepție şi nu se va mai aştepta până la ridicarea blocajelor de pe înregistrări.

***Exemplu* (SELECT cu clauza INTO)**

Să se creeze un bloc anonim în care se declară o variabilă *v_job* de tip *job_title* (%TYPE) a cărei valoare va fi titlul jobului salariatului având codul 200.

```
DECLARE
  v_job jobs.job_title%TYPE;
BEGIN
  SELECT job_title
  INTO   v_job
  FROM   employees e, jobs j
  WHERE  e.job_id=j.job_id
  AND    employee_id=200;
  DBMS_OUTPUT.PUT_LINE('jobul este '|| v_job);
END;
/
```

Varianta 2

Să se rezolve problema anterioară utilizând **variabile de legătură**. Să se afişeze rezultatul atât din bloc, cât şi din exteriorul acestuia.

```
VARIABLE rezultat VARCHAR2(35)
BEGIN
  SELECT job_title
  INTO   :rezultat
  FROM   employees e, jobs j
  WHERE  e.job_id=j.job_id AND employee_id=200;
  DBMS_OUTPUT.PUT_LINE('rezultatul este '|| :rezultat);
```

```
END;
/
PRINT rezultat
```

***Exemplu* (SELECT cu clauza INTO)**

Să se creeze un bloc anonim în care se declară o variabilă *v_job*_hiredate de tip hire_date%TYPE şi o variabil v_emp_salary de tip salary%TYPE pentru angajatul care are ID=100.

```
DECLARE
v_emp_hiredate employees.hire_date%TYPE;
v_emp_salary    employees.salary%TYPE;
BEGIN
hire_date, salary
INTO v_emp_hiredate, v_emp_salary
FROM employees
WHERE employee_id = 100;
DBMS_OUTPUT.PUT_LINE('Data_angajarii este: ' || v_emp_hiredate ||
 ' si Salariu este: ' || v_emp_salary);
END;
/
```

Funcții în *SQL*

Exista doua tipuri de functii:

- care opereaza pe o linie si returneaza un rezultat pe linie (*single row functions*);
- care opereaza pe un grup de linii si returneaza un rezultat pe grup de linii (functii grup sau *multiple row functions*).

***Single row functions* pot să fie:**

- funcții pentru prelucrarea caracterelor,
- funcții aritmetice,
- funcții pentru prelucrarea datelor calendaristice,
- funcții de conversie,
- funcții generale (*NVL*, *NVL2*, *NULLIF*, *CASE*, *DECODE* etc.).

Funcții de conversie

Conversiile pot fi făcute:

- implicit de către *server*-ul *Oracle* ;
- explicit de către utilizator.

Conversii implicite

În cazul atribuirilor, sistemul poate converti automat:

- *VARCHAR2* sau *CHAR* în *NUMBER* ;
- *VARCHAR2* sau *CHAR* în *DATE*;
- *VARCHAR2* sau *CHAR* în *ROWID*;
- *NUMBER*, *ROWID*, sau *DATE* în *VARCHAR2*.

Pentru evaluarea expresiilor, sistemul poate converti automat:

- *VARCHAR2* sau *CHAR* în *NUMBER*, dacă şirul de caractere reprezintă un număr;
- *VARCHAR2* sau *CHAR* în *DATE*, dacă şirul de caractere are formatul implicit *DD-MON-YY*;
- *VARCHAR2* sau *CHAR* în *ROWID*.

Conversii explicite

- funcția *TO_CHAR* converteşte data calendaristică sau informația numerică în şir de caractere conform unui format;
- funcția *TO_NUMBER* converteşte un şir de caractere în număr;
- funcția *TO_DATE* converteşte un şir de caractere în dată calendaristică conform unui format.

Dacă formatul este omis, convertirea se face conform unui format implicit. Funcția *TO_DATE* are forma *TO_DATE*(*şir_de_caractere* [,*'fmt'*]). Funcția este utilizată dacă se doreşte conversia unui şir de caractere care nu are formatul implicit al datei calendaristice (*DD-MON-YY*).

***Exemple*:**

```
SQL>  SELECT  TO_DATE('Feb 22, 2010','Mon dd, RRRR') as data
  2  from dual;

DATA
---------
22-FEB-10

SQL> SELECT TO_DATE('January 1, 2010','Month DD, YYYY')
  2  AS "New Year"
```

```
  3  FROM dual;

New Year
---------
01-JAN-10

SQL> SELECT TO_CHAR(SYSDATE,'Month ddth, yyyy') AS TODAY
  2  FROM dual;

TODAY
--------------------
February  21st, 2010

SQL> SELECT TO_NUMBER('232.55','999.99') AS converted
  2  from dual;

 CONVERTED
----------
    232.55
```

Funcții pentru prelucrarea caracterelor

- *LENGTH*(*string*) – returnează lungimea şirului de caractere *string*;
- *LENGTHB*(*string*) – îndeplineşte aceaşi funcție ca şi *LENGTH*, cu deosebirea că returnează numărul de octeți ocupați;
- *SUBSTR*(*string*, *start* [,*n*]) – returnează subşirul lui *string* care începe pe poziția *start* şi are lungimea *n*; dacă *n* nu este specificat, subşirul se termină la sfârşitul lui *string*;
- *LTRIM*(*string* [,*'chars'*]) – şterge din stânga şirului *string* orice caracter care apare în *chars* până la găsirea primului caracter care nu este în *chars*; dacă *chars* nu este specificat, se şterg spațiile libere din stânga lui *string*;
- *RTRIM*(*string* [,*'chars'*]) – este similar funcției *LTRIM*, cu excepția faptului că ştergerea se face la dreapta şirului de caractere;
- *LPAD*(*string*, *length* [,*'chars'*]) – adaugă *chars* la stânga şirului de caractere *string* până când lungimea noului şir devine *length*; în cazul în care *chars* nu este specificat, atunci se adaugă spații libere la stânga lui *string*;

- *RPAD*(*string, length* [*,'chars'*]) – este similar funcției *LPAD*, dar adăugarea de caractere se face la dreapta şirului;
- *REPLACE*(*string1, string2* [*,string3*]) – returnează *string1* cu toate aparițiile lui *string2* înlocuite prin *string3*; dacă *string3* nu este specificat, atunci toate aparițiile lui *string2* sunt şterse;
- *INITCAP*(*string*) – transformă primul caracter al şirului în majusculă;
- *INSTR*(*string, 'chars'* [*,start* [*,n*]]) – caută în *string*, începând de de la poziția *start*, a *n*-a apariție a secvenței *chars* şi întoarce poziția respectivă; dacă *start* nu este specificat, căutarea se face de la începutul şirului; dacă *n* nu este specificat, se caută prima apariție a secvenței *chars*;
- *UPPER*(*string*), *LOWER*(*string*) – transformă toate literele şirului de caractere *string* în majuscule, respectiv minuscule;
- *ASCII*(*char*) – returnează codul *ASCII* al unui caracter;
- *CHR*(*num*) – returnează caracterul corespunzător codului *ASCII* specificat;
- *CONCAT*(*string1, string2*) – realizează concatenarea a două şiruri de caractere;

Funcții aritmetice

Cele mai importante funcții aritmetice sunt: *ABS* (valoarea absolută), *ROUND* (rotunjire cu un număr specificat de zecimale), *TRUNC* (trunchiere cu un număr specificat de zecimale), *EXP* (ridicarea la putere a lui *e*), *LN* (logaritm natural), *LOG* (logaritm într-o bază specificată), *MOD* (restul împărțirii a două numere specificate), *POWER* (ridicarea la putere), *SIGN* (semnul unui număr), *COS* (cosinus), *COSH* (cosinus hiperbolic), *SIN*(sinus), *SQRT*(rădăcina pătrată), *TAN*(tangent), funcțiile *LEAST* şi *GREATEST*, care returnează cea mai mică, respectiv cea mai mare valoare a unei liste de expresii etc.

Funcții pentru prelucrarea datelor calendaristice

- *SYSDATE* – returnează data şi timpul curent;
- *ADD_MONTHS*(*d, count*) – returnează data care este după *count* luni de la data *d*;
- *NEXT_DAY*(*d, day*) – returnează următoarea dată după data *d*, a cărei zi a săptămânii este cea specificată prin şirul de caractere *day*;

- *LAST_DAY*(*d*) – returnează data corespunzătoare ultimei zile a lunii din care data *d* face parte;
- *MONTHS_BETWEEN*(*d2*, *d1*) – returnează numărul de luni dintre cele două date calendaristice specificate;
- *NEW_TIME*(*data*, *zona_intrare*, *zona_iesire*) – returnează ora din *zona_intrare* corespunzătoare orei din *zona_iesire*;
- *ROUND*(*d*) – dacă data *d* este înainte de miezul zilei, întoarce data *d* cu timpul setat la ora 12:00 *AM*; altfel, este returnată data corespunzătoare zilei următoare, cu timpul setat la ora 12:00 *AM*;
- *TRUNC*(*d*) – întoarce data *d*, dar cu timpul setat la ora 12:00 *AM* (miezul nopții);
- *LEAST*(*d1*, *d2*, …, *dn*), *GREATEST*(*d1*, *d2*, …, *dn*) – returnează, dintr-o listă de date calendaristice, prima, respectiv ultima dată în ordine cronologică.

***Exemplu*:**

ROUND('25-jul-95', 'MONTH') este 01-AUG-95,
ROUND('25-jul-95', 'YEAR') este 01-JAN-96,
TRUNC('25-jul-95', 'MONTH') este 01-JUL-95,
TRUNC('25-jul-95', 'YEAR') este 01-JAN-95.

1. SQL> SELECT ROUND(SYSDATE, 'MONTH') AS "Luna viitoare"
   2 FROM DUAL;

   Luna viit

   01-MAR-10

2. SQL> SELECT TRUNC(SYSDATE, 'MONTH') AS "Inceput Luna "
   2 FROM DUAL;

   Inceput L

   01-FEB-10

3. SQL> SELECT ROUND(SYSDATE, 'YEAR') AS "Inceput An"
   2 FROM DUAL;

   Inceput A

01-JAN-10

4. SQL> SELECT TRUNC(SYSDATE, 'YEAR') AS "Inceput An"
2 FROM DUAL;

Inceput A

01-JAN-10

Observatie:
- Dacă ziua este > decat 15, atunci ROUND adauga o luna, altfel şi ROUND si TRUNC afişeaza inceputul lunii curente.
- Dacă data curenta are luna > 6 atunci ROUND adauga 1 an la data, altfel şi ROUND si TRUNC afişeaza inceputul anului curent.

Exemplu:
1. SQL> SELECT SYSDATE+1 AS tomorrow
2 FROM dual;
TOMORROW

22-FEB-10

2. Adauga 10 ani la data curenta
SQL> SELECT ADD_MONTHS(SYSDATE, 120) "Peste 10 ani"
2 FROM dual;

Peste 10

21-FEB-20

3. MONTHS_BETWEEN intorce numarul de luni dintre doua date.

SQL> SELECT last_name as NUME, hire_date as "Data_angajarii ",
2 TO_CHAR(MONTHS_BETWEEN (SYSDATE, hire_date), '999,999,999.99')
3 AS " Luni_Muncite"
4 FROM employees
5 WHERE employee_id=200;
NUME Data_anga Luni_Munc

---------- --------- ----------

Whalen 17-SEP-87 **269.16**

Utilizarea literelor mari sau mici în formatul unei date calendaristice precizează forma rezultatului. De exemplu, 'MONTH' va da rezultatul MAY, iar 'Month' va da rezultatul May.

Pentru afişarea câmpurilor de tip dată calendaristică sau pentru calcule în care sunt implicate aceste câmpuri, există funcţii specifice. Câteva din elementele care apar în formatul unei date calendaristice sunt prezentate în tabelul următor.

Format	**Descriere**	**Domeniu**
SS	Secunda relativ la minut	0-59
SSSSS	Secunda relativ la zi	0-86399
MI	Minut	0-59
HH	Ora	0-12
HH24	**Ora**	0-24
DAY	Numele zilei săptămânii	SUNDAY-SATURDAY
D	Ziua săptămânii	1-7
DD	Ziua lunii	1-31 (depinde de lună)
DDD	Ziua anului	1-366 (depinde de an)
MM	Numărul lunii	1-12
MON	Numele prescurtat al lunii	JAN-DEC
MONTH	Luna	JANUARY-DECEMBER
YY	Ultimele două cifre ale anului	de exemplu, 99
YYYY	Anul	de exemplu, 1999
YEAR	Anul în litere	
CC	Secolul	de exemplu, 17
Q	Numărul trimestrului	1-4
W	Săptămâna lunii	1-5
WW	Săptămâna anului	1-52

***Exemplu*:**

Pentru operele achiziţionate în ultimii 2 ani, să se afişeze codul galeriei în care sunt expuse, data achiziţiei, numărul de luni de la cumpărare, data primei verificări, prima zi în care au fost expuse într-o galerie şi ultima zi a lunii în care au fost achiziţionate. Se va considera că data primei verificări este după 10 luni de la achiziţionare, iar prima expunere într-o galerie a avut loc în prima zi de duminică după achiziţionare.

```
SELECT cod_galerie, data_achizitiei,
   MONTHS_BETWEEN(SYSDATE, data_achizitiei) "Numar luni",
```

```
   ADD_MONTHS(data_achizitiei, 10) "Data verificare",
   NEXT_DAY(data_achizitiei, 'SUNDAY') Expunere,
   LAST_DAY(data_achizitiei)
FROM   opera
WHERE  MONTHS_BETWEEN(SYSDATE, data_achizitiei) <= 24;
```

Funcții generale

- *DECODE*(*value*, *if1*, *then1*, *if2*, *then2*, ... , *ifN*, *thenN*, *else*) – returnează *then1* dacă *value* este egală cu *if1*, *then2* dacă value este egală cu *if2* etc.; dacă *value* nu este egală cu nici una din valorile *if*, atunci funcția întoarce valoarea *else* (selecție multiplă);
- *NVL*(*e1*, *e2*) – dacă *e1* este *NULL*, returnează *e2*; altfel, returnează *e1*;
- *NVL2*(*e1*, *e2*, *e3*) – dacă *e1* este *NOT NULL*, atunci returnează *e2*, altfel, returnează *e3*;
- *NULLIF*(*e1*, *e2*) – returneaza *null* daca *e1=e2* si returneaza *e1* daca *e1* nu este egal cu *e2*;
- *COALESCE*(*e1*, *e2*, *en*) – returneaza prima expresie care nu este *null* din lista de expresii (expresiile trebuie sa fie de acelasi tip).

***Exemplu*:**

NVL(*comision*, *0*) este 0 dacă comisionul este *null*. Prin urmare, expresia *salariu*12 + comision* nu este corectă, deoarece rezultatul său este *null* dacă comisionul este *null*. Forma corectă este *salariu*12 + NVL*(*comision*, *0*).

***Exemplu*:**

Să se afişeze preţul modificat al unor cărţi în funcţie de editură. Pentru cărţile din editura *ALL* să se dubleze preţurile, pentru cele din editura *UNIVERS* să se tripleze preţurile, iar pentru cele din editura *XXX* să se reducă la jumătate acest preţ.

```
 SELECT   pret,editura,
       DECODE(editura, 'ALL',pret*2,
                   'UNIVERS',pret*3,
                   'XXX',pret/2,
                       pret) pret_revizuit
 FROM     carte;
```

Expresia *CASE* returneaza *null* daca nu exista clauza *ELSE* si daca nici o conditie nu este indeplinita.

```
   SELECT nume, sal,
```

```
                    (CASE WHEN sal <5000 THEN 'LOW'
                    WHEN sal <10000 THEN 'MEDIUM'
                     WHEN sal <20000 THEN 'GOOD'
                      ELSE 'EXCELLENT'
                          END)  AS calificare
FROM   salariat;
```

Funcții grup

- *AVG* (media aritmetică),
- *COUNT*(*) (numărul de linii returnate de o cerere),
- *COUNT* ([*DISTINCT*] numărul valorilor unui expresii),
- *SUM* (suma valorilor unei expresii),
- *MIN* (valoarea minimă a unei expresii),
- *MAX* (valoarea maximă a unei expresii),
- *STDDEV* (deviația standard),
- *VARIANCE* (dispersia).

Operatorul *ROLLUP*

Operatorul *ROLLUP* produce o mulțime care conține liniile obținute în urma grupării obişnuite şi linii pentru subtotaluri. Acest operator furnizează valori agregat şi superagregat corespunzătoare expresiilor din clauza *GROUP BY*.

Operatorul *ROLLUP* creează grupări prin deplasarea într-o singură direcție, de la dreapta la stânga, de-a lungul listei de coloane specificate în clauza *GROUP BY*. Apoi, se aplică funcția agregat acestor grupări. Dacă sunt specificate *n* expresii în operatorul *ROLLUP*, numărul de grupări generate va fi *n* + 1. Liniile care se bazează pe valoarea primelor *n* expresii se numesc linii obişnuite, iar celelalte se numesc linii superagregat.

Daca in clauza *GROUP BY* sunt specificate *n* coloane, atunci pentru a produce subtotaluri in *n* dimensiuni ar fi necesare *n+1* operatii *SELECT* legate prin *UNION ALL*. Aceasta ar fi total ineficient, deoarece fiecare *SELECT* ar implica o parcurgere a tabelului. Operatorul *ROLLUP* are nevoie de o singura parcurgere a tabelului.

***Exemplu*:**

Să se afişeze codurile de galerii mai mici decât 50, iar pentru fiecare dintre acestea şi pentru fiecare autor care are opere expuse în galerie, să se listeze valoarea totală a lucrărilor sale. De asemenea, se cere valoarea totală a operelor expuse în fiecare galerie. Rezultatul va conține şi valoarea totală a operelor din galeriile având codul mai mic decât 50, indiferent de codul autorului.

```
SELECT   cod_galerie, cod_artist, SUM(valoare)
FROM     opera
WHERE    cod_galerie < 50
GROUP BY ROLLUP(cod_galerie, cod_artist);
```

Instrucțiunea precedentă va avea un rezultat de forma:

COD_GALERIE	COD_ARTIST	SUM(VALOARE)
10	50	14000
10	60	10000
10		24000
40	50	8080
40		8080
		32080

Operatorul *CUBE*

Operatorul *CUBE* grupează liniile selectate pe baza valorilor tuturor combinațiilor posibile ale expresiilor specificate şi returnează câte o linie totalizatoare pentru fiecare grup. El produce subtotaluri pentru toate combinațiile posibile de grupări specificate în *GROUP BY*, precum şi un total general.

Daca exista *n* coloane sau expresii in clauza GROUP BY, vor exista 2^n combinatii posibile superagregat. Matematic, aceste combinatii formeaza un cub *n*-dimensional.

Pentru producerea de subtotaluri fara ajutorul operatorului *CUBE* ar fi necesare 2^n instructiuni *SELECT* legate prin *UNION ALL*.

Exemplu:

Să se afişeze valoarea totală a operelor de artă ale unui autor, expuse în

cadrul fiecărei galerii având codul mai mic decât 50. De asemenea, să se afişeze valoarea totală a operelor din fiecare galerie având codul mai mic decât 50, valoarea totală a operelor fiecărui autor indiferent de galerie şi valoarea totală a operelor din galeriile având codul mai mic decât 50.

```
SELECT   cod_galerie, cod_artist, SUM(valoare)
FROM     opera
WHERE    cod_galerie < 50
GROUP BY CUBE(cod_galerie, cod_artist)
```

COD_GALERIE	COD_ARTIST	SUM(VALOARE)
10	50	14000
10	60	10000
10		24000
40	50	8080
40		8080
	50	22080
	60	10000
		32080

Funcția *GROUPING*

Aceasta funcție este utilă pentru:

- determinarea nivelului de agregare al unui subtotal dat, adică a grupului sau grupurilor pe care se bazează subtotalul respectiv;
- identificarea proveniență unei valori *null* a unei expresii calculate, dintr-una din liniile mulțimii rezultat.

Functia returnează valoarea 0 sau 1. Valoarea 0 poate indica fie că expresia a fost utilizată pentru calculul valorii agregat, fie că valoarea *null* a expresiei este o valoare *null* stocată.

Valoarea 1 poate indica fie că expresia nu a fost utilizată pentru calculul valorii agregat, fie că valoarea *null* a expresiei este o valoare creată de *ROLLUP* sau *CUBE* ca rezultat al grupării.

Exemplu:

```
SELECT   cod_galerie, cod_artist, SUM(valoare),
      GROUPING(cod_galerie), GROUPING(cod_artist)
FROM     opera
WHERE    cod_galerie < 50
GROUP BY ROLLUP(cod_galerie, cod_artist);
```

COD_GALERIE	COD_ARTIST	SUM (VALOARE)	GROUPING (COD_GALERIE)	GROUPING (COD_ARTIST)
10	50	14000	0	0
10	60	10000	0	0
10		24000	0	1
40	50	8080	0	0
40		8080	0	1
		32080	1	1

Pe prima linie din acest rezultat, valoarea totalizatoare reprezintă suma valorilor operelor artistului având codul 50, în cadrul galeriei 10. Pentru a calcula această valoare au fost luate în considerare coloanele *cod_galerie* şi *cod_artist*. Prin urmare, expresiile *GROUPING*(*cod_galerie*) şi *GROUPING*(*cod_artist*) au valoarea 0 pentru prima linie din rezultat.

Pe linia a treia se află valoarea totală a operelor din galeria având codul 10. Această valoare a fost calculată luând în considerare doar coloana *cod_galerie*, astfel încât *GROUPING* (*cod_galerie*) şi *GROUPING*(*cod_artist*) au valorile 0, respectiv 1.

Pe ultima linie din rezultat se află valoarea totală a operelor din galeriile având codul mai mic decât 50. Nici una dintre coloanele *cod_galerie* şi *cod_artist* nu au intervenit în calculul acestui total, prin urmare valorile corespunzătoare expresiilor *GROUPING*(*cod_galerie*) şi *GROUPING*(*cod_artist*) sunt 0.

LMD

Comanda *INSERT*

Prin intermediul comenzii *INSERT* se pot introduce înregistrări în următoarele obiecte sau tipuri de partiţii:

- tabel sau tabel de bază al unei vizualizări;
- partiţie a unui tabel partiţionat;
- tabel obiect.

Sintaxa generală a instrucțiunii este următoarea:

INSERT {*inserare _tabel_singular* | *inserare_multi_tabel*};

Clauza **inserare_tabel_singular** permite introducerea uneia sau mai multor înregistrări într-un singur tabel. Sintaxa corespunzătoare este următoarea

INTO clauza_obiect [***AS*** *alias*] [(*nume_coloană* [, *nume_coloană* ...])]
{***VALUES*** ({*expr* | ***DEFAULT***} [, {*expr* | ***DEFAULT***} ...])
[*clauza_returning*] | *subcerere*}

Opțiunea ***clauza_obiect*** are următoarea formă sintactică:

{ [*nume_schema.*] {*nume_tabel* | {*nume_vizualizare* }
[@ *legatura_baza_de_ date*]
| (*subcerere* [*clauza_restricționare_subcerere*])
| *expresie_colecție_tabel*}

Sintaxa pentru *clauza_restricționare_subcerere* este următoarea:

WITH {***READ ONLY*** | ***CHECK OPTION***}
[***CONSTRAINT*** *nume_constrângere*] }

Opțiunea *clauza_returning* are următoarea formă:

RETURNING *expr* [, *expr*...] ***INTO*** *element* [, *element*...]

Sintaxa clauzei *expresie_colecție_tabel* este următoarea:

TABLE (*expresie_colecție*) [(+)]

Prin intermediul opțiunii *clauza_obiect* se specifică obiectele în care se introduc date. Dacă se specifică o vizualizare sau o vizualizare obiect, sistemul *Oracle* va insera liniile în tabelul de bază al acesteia. În continuare sunt menționate câteva restricții ale acestei clauze.

- Într-o vizualizare care a fost creată utilizând clauza *WITH CHECK OPTION* se pot introduce numai linii care sunt selectate de cererea din definiția acesteia.
- Prezența clauzei *ORDER BY* în subcererea din *clauza_obiect* garantează doar ordonarea liniilor ce vor fi inserate şi numai în cadrul fiecărei extensii a tabelului. Nu se asigură ordonarea liniilor noi printre cele deja

existente.

- Dacă o vizualizare a fost creată utilizând un singur tabel de bază este posibilă inserarea de linii, ale căror valori pot fi regăsite ulterior, prin utilizarea clauzei ***RETURNING***.
- Într-o vizualizare pot fi inserate înregistrări doar prin declanşatorii *INSTEAD OF*, dacă cererea care o defineşte conține una din următoarele construcții: un operator pe mulțimi, operatorul *DISTINCT*, o funcție agregat, clauzele *GROUP BY, ORDER BY, CONNECT BY* sau *START WITH*, o expresie colecție într-o listă *SELECT*, o subcerere într-o listă *SELECT, join*-uri.

.

Pentru tabel, vizualizare sau subcerere se poate specifica un *alias*. Acesta nu este permis în cadrul unei inserări multiple.

Clauza *VALUES* specifică valorile ce vor fi introduse în tabel sau vizualizare. Pentru a insera mai multe linii prin aceeaşi instrucțiune *INSERT*, în locul acestei clauze se va preciza o subcerere.

În linia introdusă, fiecărei coloane a listei din clauza *INSERT INTO* i se atribuie o valoare din clauza ***VALUES*** sau din **subcerere**. Dacă se omite precizarea valorii unei coloane din listă, se va considera valoarea sa implicită.

Dacă se introduce o linie care conține valori pentru fiecare coloană, nu este necesară precizarea listei de coloane în clauza *INTO*. În absența listei de coloane, clauza *VALUES* sau subcererea trebuie să precizeze valori pentru toate coloanele din tabel, în ordinea în care au fost definite.

În ceea ce priveşte valorile inserate, se impun următoarele restricții:

- nu se poate inițializa un atribut intern *LOB* al unui obiect cu altă valoare decât *empty* sau *null*;
- nu poate fi inserată o valoare *BFILE*, dacă locatorul acesteia nu a fost inițializat, fie şi cu valoarea *null*;
- pentru un tabel partiționat de tip listă, nu se poate introduce o valoare în coloana corespunzătoare cheii de partiționare, dacă aceasta nu există deja în lista uneia dintre partiții;
- la inserarea într-o vizualizare nu este permisă precizarea cuvântului cheie *DEFAULT*.

Opțiunea *WITH READ ONLY* indică faptul că tabelul sau vizualizarea nu pot

fi actualizate. Clauza *WITH CHECK OPTION* determină interzicerea modificărilor asupra tabelului sau vizualizării care ar putea produce linii ce nu sunt incluse în subcerere. Acestei constrângeri i se poate atribui un nume prin intermediul opțiunii *CONSTRAINT*. În absența acesteia, sistemul îi va atribui automat un nume de forma *SYS_Cn,* unde *n* este un număr întreg ce asigură unicitatea identificatorului în cadrul bazei de date.

Opțiunea *legătură_bază_de_date* specifică un nume complet sau parțial al unei legături către o bază de date distantă, în care se află tabelul sau vizualizarea. În absența acestei clauze, se presupune că tabelul sau vizualizarea se află în baza de date locală.

Clauza *expresie_colecție_tabel* determină tratarea valorii din *expresie_colecție* ca un tabel, în cadrul cererilor şi operațiilor *LMD*. În *expresie_colecție* poate fi specificată o subcerere, o coloană, o funcție sau un constructor de colecție. Indiferent de forma sa, aceasta trebuie să returneze o valoare colecție, adică o valoare de tip tablou imbricat sau vector. Procesul de extragere al elementelor unei colecții este numit distribuirea colecțiilor.

Clauza *RETURNING* recuperează liniile afectate de o instrucțiune *LMD*. Această clauză poate fi specificată pentru tabele şi vizualizări având un singur tabel de bază. Atunci când operează asupra unei singure linii, o instrucțiune *LMD* ce conține clauza *RETURNING* poate recupera valori ale coloanelor, valori *ROWID* şi valori referință corespunzătoare liniei afectate. Aceste valori pot fi stocate în variabile gazdă sau variabile ***PL/SQL***. Instrucțiunile *LMD* care operează asupra mai multor linii pot recupera aceste valori în tablouri de legătură (*bind array*). În *expr* poate fi specificată orice expresie validă, cu excepția expresiilor de tip subcerere scalară.

Opțiunea *INTO* indică faptul că valorile liniilor modificate urmează să fie stocate în variabilele specificate. Fiecare element din clauză este o variabilă gazdă sau *PL/SQL* care stochează o valoare recuperată (*expr*). Fiecărei expresii din lista *RETURNING* trebuie să i se asocizeze în lista *INTO* o variabilă gazdă sau *PL/SQL,* compatibilă din punct de vedere al tipului de date.

Clauza *RETURNING* nu poate fi specificată pentru inserări multitabel, operații *LMD* paralele, recuperarea tipurilor *LONG* sau vizualizări asupra cărora a fost definit un declanşator de tip *INSTEAD OF*.

Subcererea specificată în comanda *INSERT* returnează linii care vor fi adăugate în tabel. Dacă în tabel se introduc linii prin intermediul unei subcereri, coloanele din lista *SELECT* trebuie să corespundă, ca număr şi tip, celor precizate în clauza *INTO*. În absența unei liste de coloane în clauza *INTO,* subcererea trebuie să furnizeze valori pentru fiecare atribut al obiectului destinație, respectând ordinea în care acestea au fost definite.

Observații:

- Pentru claritate, este recomandată utilizarea unei liste de coloane în clauza *INSERT*.
- În clauza *VALUES*, valorile de tip caracter şi dată calendaristică trebuie incluse între apostrofuri. Nu se recomandă includerea între apostrofuri a valorilor numerice, întrucât aceasta ar determina conversii implicite la tipul *NUMBER*.
- Pentru introducerea de valori speciale în tabel, pot fi utilizate funcții.

Adăugarea unei linii care va conține valori *null* se poate realiza în mod:

- implicit, prin omiterea numelui coloanei din lista de coloane;
- explicit, prin specificarea în lista de valori a cuvântului cheie *null* sau a şirului vid în cazul şirurilor de caractere sau datelor calendaristice.

***Exemplu*:**

Să se adauge prin metoda explicită, respectiv prin metoda implicită, două înregistrări în tabelul *artist*, precizând numai codul, numele şi prenumele artistului.

```
INSERT INTO artist(cod_artist, nume, prenume)
VALUES (50, 'Grigorescu', 'Nicolae');

INSERT INTO artist
VALUES (70, 'Andreescu', 'Ion', NULL, NULL, NULL, NULL);
```

***Exemplu*:**

```
INSERT INTO opera (cod_opera, tip, titlu, cod_artist,
            data_achizitiei, cod_galerie, material)
VALUES (110, 'sculptura', 'Rugaciune', 60,
    TO_DATE('20 JAN, 1997', 'DD MON, YYYY'),
    cod_galerie_secv.CURRVAL, 'bronz');
```

Server-ul *Oracle* aplică automat toate tipurile de date, domeniile de valori şi constrângerile de integritate. La introducerea sau actualizarea de înregistrări, pot apărea erori în următoarele situații:

- nu a fost specificată o valoare pentru o coloană *NOT NULL*;
- există valori duplicat care încalcă o constrângere de unicitate;
- a fost încălcată constrângerea de cheie externă sau o constrângere de tip *CHECK*;
- există incompatibilitate în privința tipurilor de date;

- s-a încercat inserarea unei valori având o dimensiune mai mare decât a coloanei corespunzătoare.

***Exemplu*:**

Următoarea instrucțiune va genera eroarea „*ORA-01400*: *cannot insert NULL into ...*“, întrucât se încearcă inserarea unei linii în tabelul *opera* fără a preciza valoarea cheii primare.

```
INSERT INTO opera(titlu, data_achizitiei)
VALUES ('Flori de camp', SYSDATE);
```

***Exemplu*:**

Se presupune că există un tabel *opera_3000* care are aceeaşi structură ca şi tabelul *opera*. Să se insereze în acest tabel codul şi titlul operelor a căror valoare, incluzând polițele de asigurare, depăşeşte 3000.

```
INSERT INTO   opera_3000(cod_opera, titlu)
  SELECT cod_opera, titlu
  FROM   opera o,
        (SELECT    cod_opera, sum(valoare) val_polite
         FROM      polita_asig
         GROUP BY  cod_opera) x
            WHERE  x.cod_opera = o.cod_opera
              AND    o.valoare + x.val_polite > 3000;
```

***Exemplu*:**

```
INSERT INTO domeniu
VALUES ('&cod', '&intdom');--  inserare prin parametrizare
```

***Exemplu*:**

Să se introducă o înregistrare în tabelul *opera*. Presupunând că au fost declarate variabilele de legătură *bind1* şi *bind2*, să se returneze valoarea operei mărită cu 20% şi codul acesteia.

```
INSERT INTO opera
    (cod_opera, titlu, cod_autor, valoare, data_achizitiei)
VALUES (178, 'Abis', 80, 4500, SYSDATE)
RETURNING valoare*1.20, cod_opera INTO :bind1, :bind2;
```

Inserări multitabel

O inserare multitabel presupune introducerea de linii calculate pe baza

rezultatelor unei subcereri, în unul sau mai multe tabele. Acest tip de inserare, introdus de *Oracle9i*, este util în mediul *data warehouse*. Astfel, datele extrase dintr-un sistem sursă, pot fi transformate utilizând instrucțiuni *INSERT* multitabel, spre a fi încărcate în obiectele bazei de date.

Pentru o astfel de inserare, în versiunile anterioare lui *Oracle9i* erau necesare *n* operații independente *INSERT INTO…SELECT…*, unde *n* reprezintă numărul tabelelor destinație. Aceasta presupunea *n* procesări ale aceleiaşi surse de date şi, prin urmare, creşterea de *n* ori a timpului necesar procesului.

Sintaxa clauzei *inserare_multi_tabel* este următoarea:

{***ALL INTO***…[***VALUES***…] [***INTO***…[***VALUES***…] …]
| *inserare_condiționată*} *subcerere*

Clauza *inserare_condiționată* are forma următoare:

[***ALL*** | ***FIRST***]
WHEN *condiție* ***THEN INTO***…[***VALUES***…]
[***INTO***…[***VALUES***…] …]
[***WHEN*** *condiție* ***THEN INTO***…[***VALUES***…]
[***INTO***…[***VALUES***…] …] …]
[***ELSE INTO***…[***VALUES***…]
[***INTO***…[***VALUES***…] …] …]

Pentru a efectua o inserare multitabel necondiționată, sistemul va executa câte o instrucțiune *INSERT…INTO* pentru fiecare linie returnată de subcerere.

Utilizând clauza *inserare_condiționată*, decizia inserării unei linii depinde de condiția specificată prin intermediul opțiunii *WHEN*. Expresiile prezente în aceste condiții trebuie să facă referință la coloane returnate de subcerere. O instrucțiune de inserare multitabel poate conține maxim 127 clauze *WHEN*.

Specificarea opțiunii *ALL* determină evaluarea tuturor condițiilor din clauzele *WHEN*. Pentru cele a căror valoare este *TRUE*, se inserează înregistrarea specificată în opțiunea *INTO* corespunzătoare.

Opțiunea *FIRST* determină inserarea corespunzătoare primei clauze *WHEN* a cărei condiție este evaluată *TRUE*. Toate celelalte clauze *WHEN* sunt ignorate.

Dacă nici o condiție din clauzele *WHEN* nu este *TRUE,* atunci sistemul execută clauza *INTO* corespunzătoare opțiunii *ELSE*, iar dacă aceasta nu există, nu efectuează nici o acțiune.

Inserările multitabel pot fi efectuate numai asupra tabelelor, nu şi asupra vizualizărilor sau vizualizărilor materializate. De asemenea, acest tip de inserare nu se poate efectua asupra tabelelor distante. Subcererea dintr-o instrucțiune corespunzătoare unei inserări multitabel nu poate utiliza o secvență.

***Exemplu*:**

Se presupune că licitațiile pentru achiziționarea operelor de artă au loc numai în zilele de miercuri. Fie tabelul *opera_intrare* (*data_inceput*, *sapt_1*, *sapt_2*, *sapt_3*, *sapt_4*), în care *data_inceput* reprezintă data primei zile de miercuri din lună, iar *sapt_i* furnizează valoarea operelor achiziționate în săptămâna respectivă. Considerând că în fiecare săptămână se achiziționează o singură lucrare, să se insereze liniile corespunzătoare în tabelul *opera*.

```
INSERT ALL
  INTO    opera (cod_opera, valoare, data_achizitiei)
  VALUES  (secv_cod_opera.NEXT_VALUE, sapt_1, data_inceput)
  INTO    opera (cod_opera, valoare, data_achizitiei)
  VALUES  (secv_cod_opera.NEXT_VALUE, sapt_2, data_inceput + 7)
  INTO    opera (cod_opera, valoare, data_achizitiei)
  VALUES  (secv_cod_opera.NEXT_VALUE, sapt_3, data_inceput + 14)
  INTO    opera (cod_opera, valoare, data_achizitiei)
  VALUES  (secv_cod_opera.NEXT_VALUE, sapt_4,  data_inceput + 21)
SELECT  data_inceput, sapt_1, sapt_2, sapt_3, sapt_4
FROM    opera_intrare;
```

***Exemplu*:**

Se consideră trei tabele care conțin informații referitoare la galerii, în funcție de numărul operelor găzduite de către acestea. Tabelele se numesc *galerie_mica*, *galerie_medie*, *galerie_mare* după cum numărul de opere este mai mic decât 30, cuprins între 31 şi 100, respectiv mai mare decât 101. Se presupune că structura fiecăruia dintre tabele este alcătuită din coloanele *cod_galerie*, *nume*, *număr_opere*. Să se insereze înregistrări în aceste tabele.

```
INSERT ALL
  WHEN nr_opere <= 30 THEN
    INTO galerie_mica
  WHEN nr_opere > 30 AND nr_opere <= 100 THEN
    INTO galerie_medie
  WHEN nr_opere > 100 THEN
    INTO galerie_mare
SELECT  g.cod_galerie, nume_galerie, nr_opere
FROM    galerie g, (SELECT    cod_galerie, count(*) nr_opere
                    FROM      opera
```

```
                GROUP BY  cod_galerie) x
WHERE   g.cod_galerie = x.cod_galerie;
```

Comanda *UPDATE*

Pentru modificarea valorilor existente într-un tabel sau într-un tabel de bază al unei vizualizări, se utilizează comanda *UPDATE*, care are următoarea sintaxă generală:

UPDATE
{***clauza_obiect*** } [***AS*** *alias*]
SET { { (*nume_coloană* [, *nume_coloană…*]) = (*subcerere*)
| *nume_coloană* = {*expr* | (*subcerere*) | ***DEFAULT***} }
[, { (*nume_coloană* [, *nume_coloană …*]) = (*subcerere*)
| *nume_coloană* = {*expr* | (*subcerere*) | ***DEFAULT***} } …]}
[***WHERE*** *condiție*] [***clauza_returning***];

Sintaxa şi semnificația opțiunilor *clauza_obiect, alias, clauza_returning* au fost prezentate explicit în cadrul comenzii *INSERT*.

. Prin *nume_coloană* se precizează coloanele ale căror valori vor fi modificate. Valorile coloanelor care nu apar în această listă rămân nemodificate. O coloană care se referă la un atribut al unui obiect *LOB* va trebui inițializată *empty* sau *null*.

Subcererea trebuie să returneze câte o linie pentru fiecare înregistrare actualizată. De asemenea, numărul şi tipul coloanelor returnate de subcerere trebuie să corespundă cu cel al coloanelor actualizate. Dacă subcererea nu returnează nici o linie, atunci coloanei respective i se va atribui valoarea *null*.

Dacă se modifică o singură linie, este recomandată utilizarea valorii cheii primare pentru a identifica înregistrarea supusă actualizării. În absența clauzei *WHERE*, sunt modificate toate liniile din tabel.

***Exemplu*:**

a) Să se transfere în galeria 10 opera având codul 110.

```
UPDATE opera
SET    cod_galerie = 10
WHERE  cod_opera = 110;
```

b) Să se modifice informaţiile referitoare la opera având codul 120, considerând că se află expusă în aceeaşi galerie şi a fost achiziţionată la aceeaşi dată ca şi opera al cărei cod este 110.

```
UPDATE opera
SET    (cod_galerie, data_achizitiei) =
                  (SELECT cod_galerie, data_achizitiei
                   FROM   opera
                   WHERE  cod_opera = 110)
WHERE  cod_opera = 120;
```

c) Să se modifice *copie_opera* pe baza valorilor din tabelul *opera*. Se consideră că operele care au acelaşi autor ca şi opera având codul 100 sunt expuse în galeria ce conţine lucrarea al cărei cod este 110.

```
UPDATE copie_opera
SET    cod_galerie = (SELECT cod_galerie
                      FROM   opera
                      WHERE  cod_opera = 110)
WHERE  cod_autor = (SELECT cod_autor
                    FROM   opera
                    WHERE  cod_opera = 100);
```

Cazurile în care instrucţiunea *UPDATE* nu poate fi executată sunt similare celor în care eşuează instrucţiunea *INSERT*. Acestea au fost menţionate anterior.

***Exemplu*:**

```
UPDATE opera
SET    cod_galerie = 47
WHERE  cod_galerie = 40;
```

Deoarece galeria având codul 47 nu există în tabelul „părinte“ (*galerie*), instrucţiunea precedentă va genera eroarea „*ORA-02291*: *integrity constraint* (*STUDENT.SYS_C002773*) *violated - parent key not found*“.

***Exemplu*:**

Să se transfere în galeria având codul 50 toate operele din galeria 60, care sunt create de Nicolae Grigorescu. Să se mărească cu 10% valoarea acestor opere.

```
UPDATE opera o
SET    cod_galerie = 50,
       valoare = valoare * 1.10
WHERE  (SELECT INITCAP(nume) ||' '||INITCAP(prenume)
        FROM   artist
```

```
      WHERE  cod_artist = o.cod_artist)=
                          'Nicolae Grigorescu'
AND    o.cod_galerie = 60;
```

***Exemplu*:**

Să se actualizeze operele al căror autor este Ştefan Luchian astfel: codul galeriei devine codul galeriei în care este expusă cea mai scumpă operă a artistului, valoarea fiecărei opere va fi mărită cu 10% din media valorilor operelor din galerie, iar data achiziției va fi considerată data celei mai recente achiziții din galerie.

```
UPDATE opera o
SET    cod_galerie = (SELECT cod_galerie
                      FROM   artist a, opera b
                      WHERE  a.cod_artist = b.cod_artist
                      AND    INITCAP(nume) = 'Luchian '
                      AND    INITCAP(prenume) = 'Stefan'
                      AND    valoare =
                           (SELECT MAX(valoare)
                            FROM   opera
                            WHERE  cod_artist =
                                 b.cod_artist)),
     (valoare, data_achizitiei) =
                     (SELECT o.valoare + AVG(o2.valoare)*0.10,
                           MAX(o2.data_achizitiei)
                      FROM   opera o2
                      WHERE  o.cod_opera = o2.cod_opera)
WHERE cod_artist = (SELECT cod_artist
                    FROM   artist
                    WHERE  INITCAP(nume) = 'Luchian '
                    AND    INITCAP(prenume) = 'Stefan');
```

***Exemplu*:**

Să se mărească cu 1000 valoarea operei având codul 100 şi să se **returneze** titlul, codul artistului şi vechea valoare în variabilele de legătură *bind1*, *bind2*, respectiv *bind3*.

```
UPDATE    opera
SET       valoare = valoare + 1000
WHERE     cod_opera = 100
RETURNING titlu, cod_artist, valoare – 1000
INTO      :bind1, :bind2, :bind3;
```

Oracle9i introduce o nouă funcționalitate, reprezentată de posibilitatea utilizării valorilor implicite (*DEFAULT*) în instrucțiunile *INSERT* şi *UPDATE*. Unei coloane i se atribuie valoarea implicită definită la crearea sau modificarea structurii tabelului dacă:

- nu se precizează nici o valoare;
- se precizează cuvântul cheie *DEFAULT* în comenzile *INSERT* sau *UPDATE*.

Dacă nu a fost definită nici o valoare implicită pentru coloana respectivă, sistemul îi atribuie valoarea *null*. Cuvântul cheie *DEFAULT* nu poate fi specificat la actualizarea vizualizărilor.

Exemplu:

Să se creeze tabelul *test*, având o coloană căreia i se specifică o valoare implicită. Ulterior, să se modifice această valoare. Să se insereze şi să se actualizeze câte o înregistrare din tabel, utilizând valoarea implicită.

```
CREATE TABLE test(
  cod NUMBER       PRIMARY KEY,
  nume VARCHAR2(30) DEFAULT 'NECUNOSCUT');
ALTER TABLE test MODIFY (nume DEFAULT 'NEDEFINIT');
INSERT INTO test VALUES (1, DEFAULT);
UPDATE test SET nume = DEFAULT WHERE cod = 2;
```

Comanda *DELETE*

Ştergerea unor linii dintr-un tabel (simplu, partiționat sau tabel de bază al unei vizualizări) se realizează prin intermediul comenzii *DELETE*, care are următoarea sintaxă:

DELETE
[***FROM***] {*clauza_obiect* } [***AS*** *alias*]
[***WHERE*** *condiție*] [***clauza_returning***];

Sintaxa şi semnificația clauzelor care sunt prezente în instrucțiunea *DELETE*

sunt similare celor expuse în cadrul instrucțiunii *INSERT*.

Clauza *WHERE* determină ştergerea liniilor identificate prin condiția respectivă. În absența clauzei *WHERE* sunt şterse toate liniile din tabel. Pentru a şterge linii identificate cu ajutorul valorilor din alte tabele, se utilizează subcereri.

***Exemplu*:**

a) Să se suprime înregistrarea corespunzătoare operei având codul 120.

```
DELETE FROM opera
WHERE  cod_opera = 120;
```

Următoarea instrucțiune are acelaşi efect, dar utilizează o subcerere:

```
DELETE FROM (SELECT * FROM opera)
WHERE  cod_opera = 120;
```

b) Să se şteargă întregul conținut al tabelului *copie_opera*.

```
DELETE FROM copie_opera;
```

c) Să se şteargă toate operele care se află expuse într-o galerie al cărei nume conține şirul de caractere „FLOARE“.

```
DELETE FROM opera
WHERE  cod_galerie = (SELECT cod_galerie
                      FROM   galerie
                      WHERE  UPPER(nume_galerie)
                             LIKE '% FLOARE %');
```

Dacă se încearcă ştergerea unei înregistrări care conține o valoare implicată într-o constrângere de integritate, atunci va fi returnată o eroare.

***Exemplu*:**

```
DELETE FROM galerie
WHERE  cod_galerie = 40;
```

În urma execuției acestei instrucțiuni sistemul generează eroarea „*ORA-02292*: *integrity constraint* (*STUDENT.SYS_C002773*) *violated - child record found*“, datorată calității de cheie externă a coloanei *cod_galerie* în tabelul *opera*. Există opere în galeria având codul 40 şi de aceea aceasta nu poate fi suprimată.

În cazul în care constrângerea de integritate referențială a fost definită utilizând opțiunea *ON DELETE CASCADE*, atunci instrucțiunea *DELETE* va şterge atât liniile indicate, cât şi liniile „copil“ din tabelele corespunzătoare.

***Exemplu*:**

Să se şteargă ultima operă de artă achiziționată. Să se rețină valoarea acesteia într-o variabilă de legătură.

```
DELETE FROM opera
WHERE  data_achizitiei = (SELECT MAX(data_achizitiei)
                  FROM   opera)
RETURNING valoare INTO :bind1;
```

***Exemplu*:**

Să se şteargă cartea cea mai scumpă şi să se rețină valoarea acesteia într-o variabilă de legătură.

```
DELETE  FROM  carte
WHERE   pret = (SELECT  MAX(pret)
                 FROM    carte
RETURNING pret INTO :aaa;
```

Utilizarea subcererilor în instrucțiunile *LMD*

În exemplele prezentate anterior se observă că subcererile pot fi utilizate pentru furnizarea valorilor care identifică liniile afectate de o instrucțiune *LMD*. În continuare, vor fi prezentate şi alte situații în care subcererile pot fi folosite în instrucțiuni *LMD*.

O subcerere poate fi folosită pentru a identifica tabelul şi coloanele referite de o instrucțiune *LMD*. De exemplu, subcererile pot fi folosite în locul numelui tabelului din clauza *INTO* a instrucțiunii *INSERT*. Lista *SELECT* a acestei subcereri trebuie să conțină acelaşi număr de coloane ca şi lista corespunzătoare clauzei *VALUES*. Pentru ca instrucțiunea *INSERT* să fie executată cu succes, trebuie să fie respectate toate regulile impuse asupra coloanelor tabelului de bază. Astfel, nu se poate specifica o valoare duplicat pentru cheia primară şi nu se poate lăsa neprecizată valoarea unei coloane având constrângerea *NOT NULL*.

***Exemplu*:**

```
INSERT INTO (SELECT cod_opera, titlu, data_achizitiei,
                valoare, cod_galerie
          FROM   opera
          WHERE  cod_galerie = 40)
VALUES (150, 'Intelepciunea Pamantului',
      TO_DATE('10-JUL-80', 'DD-MON-RR'), 10000, 40);
```

Specificarea opțiunii *WITH CHECK OPTION* într-o subcerere utilizată în

locul tabelului corespunzător unei instrucțiuni *LMD* are ca efect interzicerea operațiilor care produc linii ce nu vor fi incluse în rezultatul subcererii.

***Exemplu*:**

```
INSERT INTO (SELECT cod_opera, titlu,
                    data_achizitiei, valoare
             FROM   opera
             WHERE  cod_galerie = 40 WITH CHECK OPTION)
VALUES (160, 'Portretul unei femei',
        TO_DATE('26-MAR-82', 'DD-MON-RR'),15000);
```

Subcererea din exemplul anterior identifică operele expuse în galeria având codul 40. Întrucât coloana *cod_galerie* nu se află în lista *SELECT*, nu se precizează nici o valoare a acesteia pentru linia introdusă. Aceasta înseamnă că noua înregistrare ar avea valoarea *null* pentru coloana *cod_galerie* şi, prin urmare, nu ar apărea în rezultatul subcererii. Execuția instrucțiunii determină generarea erorii „*ORA-01402*: *view WITH CHECK OPTION where-clause violation*“.

LIMBAJUL PENTRU CONTROLUL DATELOR

Controlul unei baze de date cu ajutorul *SQL*-ului se refera la:

- asigurarea confidentialitatii si securitatii datelor;
- organizarea fizica a datelor;
- realizarea unor performante;
- reluarea unor actiuni in cazul unei defectiuni;
- garantarea coerentei datelor in cazul prelucrarii concurente.

Sistemul de gestiune trebuie:

- să pună la dispoziția unui număr mare de utilizatori o mulțime coerentă de date;
- să garanteze coerența datelor în cazul manipulării simultane de către diferiți utilizatori.

Coerența este asigurată cu ajutorul conceptului de **tranzacție**.

Tranzacția este unitatea logică de lucru constând din una sau mai multe instrucțiuni *SQL*, care trebuie să fie executate **atomic** (ori se execută toate, ori nu

se execută nici una!), asigurând astfel trecerea BD dintr-o stare coerentă în altă stare coerentă.

Dacă toate operațiile ce constituie tranzacția sunt executate şi devin efective, spunem că tranzacția este **validată**, iar modificările aduse de tranzacție devin definitive.

Controlul tranzacțiilor se realizează prin utilizarea instrucțiunilor limbajului de control al datelor: *COMMIT*, *ROLLBACK*, *SET TRANSACTION*, *SAVEPOINT*.

Tranzacțiile pot fi de tip *LMD*, *LDD* sau *LCD*. Tranzacțiile *LDD* şi *LCD* constau dintr-o singură instrucțiune *LDD*, respectiv *LCD*. Tranzacțiile *LMD* constau dintr-o succesiune de instrucțiuni *LMD* care determină o modificare consistentă a datelor. De exemplu, un transfer de fonduri între două conturi bancare presupune debitarea unui cont şi creditarea celuilalt cu aceeaşi sumă. Ambele acțiuni trebuie fie să eşueze, fie să se încheie cu succes. Creditarea nu trebuie salvată fără să fie salvată şi operația de debitare.

Deci o tranzacție constă:

- dintr-o singură instrucțiune *LDD*;
- dintr-o singură instrucțiune *LCD*;
- din instrucțiuni *LMD* care fac schimbări consistente în date.

Procesarea tranzacțiilor

Modificările făcute asupra datelor în timpul unei tranzacții sunt temporare până când tranzacția este salvată. Operațiile de prelucrare a datelor afectează inițial un *buffer* al bazei. Prin urmare, starea precedentă a datelor poate fi recuperată.

Tranzacția începe:

- când este executată prima instrucțiune LMD SQL.
- după o comandă COMMIT,
- după o comandă ROLLBACK,
- după conectarea inițială la Oracle,

şi se termină la:

- apariția unei instrucțiuni *LDD*;
- emiterea unei instrucțiuni *LCD*;
- părăsirea mediului *SQL*Plus*;
- defectarea stației de lucru sau la înregistrarea unei întreruperi a sistemului.

După ce se termină o tranzacție, prima instrucțiune SQL executabilă va genera automat începutul unei noi tranzacții.

Execuția unei instrucțiuni *LDD* determină salvarea automată a modificărilor survenite pe perioada tranzacției. *Server*-ul *Oracle* generează o operație *COMMIT* implicită înainte şi după orice instrucțiune *LDD*. Aşadar, chiar dacă instrucțiunea *LDD* nu este executată cu succes, instrucțiunea anterioară acesteia nu mai poate fi anulată întrucât *server*-ul a efectuat deja operația *COMMIT*.

Instrucțiunile *COMMIT* şi *ROLLBACK* încheie tranzacția curentă prin definitivarea, respectiv anularea tuturor modificărilor aflate în aşteptare. Aceste instrucțiuni permit:

- asigurarea consistenței datelor;
- previzualizarea modificărilor asupra datelor înainte ca acestea să devină permanente;
- gruparea logică a operațiilor.

Ieşirea din mediul *SQL*Plus* fără lansarea unei instrucțiuni *COMMIT* sau *ROLLBACK* are ca efect salvarea automată a tranzacției curente.

Atunci când intervine o anomalie (cădere) a sistemului sau închiderea anormală a sesiunii *SQL*Plus*, întreaga tranzacție curentă este anulată automat (*ROLLBACK*). Acest fapt împiedică eroarea să cauzeze modificări nedorite ale datelor şi determină revenirea la starea din momentul ultimei operații *COMMIT*.

O ieşire normală din *iSQL*Plus* are loc prin apăsarea butonului *Exit*. Terminarea normală a unei sesiuni *SQL*Plus* are loc prin execuția comenzii *EXIT*. În *SQL*Plus*, închiderea ferestrei este interpretată ca ieşire anormală.

Interfața *SQL*Plus* pune la dispoziție variabila de mediu *AUTOCOMMIT*, care poate avea valorile *ON* sau *OFF*. Dacă această variabilă are valoarea *ON*, atunci efectul fiecărei instrucțiuni *LMD* se definitivează imediat ce instrucțiunea a fost executată. Dacă variabila *AUTOCOMMIT* are valoarea *OFF*, definitivarea unei tranzacții va avea loc la execuția comenzii *COMMIT* sau în cazurile de salvare automată a tranzacțiilor, prezentate anterior.

După încheierea unei tranzacții, următoarea instrucțiune *SQL* executabilă marchează automat începutul unei noi tranzacții.

Comanda *COMMIT*

Instrucțiunea *COMMIT* determină încheierea tranzacției curente şi permanentizarea modificărilor care au intervenit pe parcursul acesteia. Instrucțiunea suprimă toate punctele intermediare definite în tranzacție şi eliberează blocările tranzacției. De asemenea, instrucțiunea poate fi utilizată pentru

terminarea unei tranzacții *read-only* începută printr-o comandă *SET TRANSACTION*.

Sistemul lansează implicit o comandă *COMMIT* înainte şi după orice instrucțiune *LDD*. Sintaxa corespunzătoare comenzii *COMMIT* este următoarea:

COMMIT [***WORK***] [***COMMENT*** '*text*' | ***FORCE*** '*text*' [, *nr_întreg*]];

Opțiunea *WORK* a fost introdusă din motive de conformitate cu standardul *SQL*. Instrucțiunile *COMMIT* şi *COMMIT WORK* sunt echivalente.

Clauza *COMMENT* permite specificarea unui comentariu care va fi asociat tranzacției curente. Textul care îi urmează poate ocupa maxim 255 octeți şi va fi stocat în vizualizarea *DBA_2PC_PENDING* din dicționarul datelor.

Într-un sistem distribuit, clauza *FORCE* permite salvarea manuală a unei tranzacții distribuite *in-doubt* (în care operația *COMMIT* a fost întreruptă de o cădere a sistemului sau a rețelei). Textul care îi urmează conține identificatorul local sau global al tranzacției. Pentru a afla aceşti identificatori se poate consulta, de asemenea, vizualizarea *DBA_2PC_PENDING* din dicționarul datelor.

O instrucțiune *COMMIT* care conține clauza *FORCE* permanentizează numai tranzacția specificată şi nu o afectează pe cea curentă. Prezența acestei clauze nu este permisă în *PL/SQL*.

***Exemplu*:**

Să se insereze o înregistrare în tabelul *artist* şi să se permanentizeze modificarea.

```
INSERT INTO artist(cod_artist, nume, prenume)
VALUES (189, 'Pallady', 'Theodor');
COMMIT;
```

Înainte de operația *COMMIT*, utilizatorul curent poate vizualiza rezultatele comenzilor *LMD* prin interogarea tabelelor. Efectele acestor operații nu sunt vizibile celorlalți utilizatori. *Server*-ul *Oracle* asigură consistența la citire, astfel încât fiecare utilizator vizualizează datele în starea corespunzătoare ultimei operații *COMMIT* efectuate asupra lor. Liniile afectate de tranzacția curentă sunt blocate, nefiind posibilă modificarea lor de către ceilalți utilizatori.

Dacă mai mulți utilizatori modifică simultan acelaşi tabel, fiecare dintre aceştia poate consulta numai propriile modificări. Pe măsură ce operația *COMMIT* este executată de către utilizatori, actualizările efectuate de aceştia devin vizibile.

În urma execuției instrucțiunii *COMMIT*, modificările asupra datelor sunt scrise în baza de date, iar starea precedentă a datelor este pierdută definitiv. În

acest fel, rezultatele tranzacției pot fi vizualizate de către toți utilizatorii. Blocările asupra liniilor afectate sunt eliberate, astfel că înregistrările devin disponibile celorlalți utilizatori pentru a efectua noi actualizări. După operația *COMMIT*, toate punctele intermediare (*SAVEPOINT*) ale tranzacției respective sunt șterse.

Comanda *ROLLBACK*

Atunci când o linie este modificată, valorile anterioare ale coloanelor actualizate sunt salvate într-un segment de reluare. Dacă tranzacția este anulată, *server*-ul *Oracle* va rescrie valorile din acest segment în linia tabelului.

Pentru a renunța la modificările efectuate se utilizează instrucțiunea *ROLLBACK*. În urma execuției acesteia, se încheie tranzacția, se anulează modificările asupra datelor, se restaurează starea lor precedentă și se eliberează blocările asupra liniilor.

O parte a tranzacției poate fi anulată automat printr-o operație *ROLLBACK* implicită dacă a fost detectată o eroare în timpul execuției unei instrucțiuni. Dacă o singură instrucțiune *LMD* eșuează în timpul execuției unei tranzacții, efectul său este anulat de un *ROLLBACK* la nivel de instrucțiune, dar schimbările efectuate de instrucțiunile *LMD* precedente nu sunt anulate. Acestea din urmă pot fi salvate sau anulate explicit de către utilizator.

Sintaxa instrucțiunii *ROLLBACK* este următoarea:

ROLLBACK [***WORK***]
  [***TO*** [***SAVEPOINT***] *pct_intermediar* | ***FORCE*** '*text*'];

Semnificațiile opțiunilor *WORK* și *FORCE* sunt similare celor prezentate în cadrul instrucțiunii *COMMIT*.

În clauza *TO SAVEPOINT* se poate specifica punctul intermediar până la care se dorește anularea tranzacției. În absența acestei clauze, întreaga tranzacție este anulată. O tranzacție *in-doubt* nu poate fi anulată manual până la un punct intermediar.

Dacă a fost definit un punct intermediar prin instrucțiunea *SAVEPOINT nume*, instrucțiunea *ROLLBACK TO SAVEPOINT nume* determină întoarcerea tranzacției curente la punctul intermediar specificat.

În felul acesta se revine într-o stare anterioară a tranzacției și se anulează modificările care au survenit după definirea punctului intermediar. De asemenea, sunt șterse punctele intermediare ulterioare acestuia și sunt eliberate toate blocările asupra tabelelor sau liniilor, efectuate după punctul intermediar respectiv.

Comanda *SAVEPOINT*

Instrucțiunea *SAVEPOINT* marchează un punct intermediar în procesarea tranzacției. În acest mod este posibilă împărțirea tranzacției în subtranzacții. Această instrucțiune nu face parte din standardul *ANSI* al limbajului *SQL*.

Punctele intermediare definite în procesarea unei tranzacții nu sunt obiecte ale schemei şi nu pot fi referite în dicționarul datelor. Nu există nici o modalitate de a lista punctele intermediare definite. Dacă este creat un al doilea punct intermediar având acelaşi nume cu un punct intermediar precedent, acesta din urmă este şters.

Instrucțiunea *SAVEPOINT* are sintaxa:

SAVEPOINT *nume_pct_intermediar*;

***Exemplu*:**

Să se mărească prin 20%, respectiv 50% valoarea operelor care au codurile 100, respectiv 150. Să se verifice că valoarea totală a operelor din galerie nu depăşeşte 7 000 000, iar apoi să se reactualizeze valoarea operei având codul 150, mărind-o cu 40%.

```
UPDATE opera
SET    valoare = valoare * 1.2
WHERE  cod_opera = 100;
SAVEPOINT val_100;
UPDATE opera
SET    valoare = valoare * 1.5
WHERE  cod_opera = 150;
SAVEPOINT val_150;
SELECT SUM(valoare)
FROM   opera;
ROLLBACK TO SAVEPOINT val_100;
UPDATE opera
SET    valoare = valoare * 1.4
WHERE cod_opera = 150;
COMMIT;
```

*INTERFATA SQL*PLUS*

***SQL* este un limbaj** compus din comenzi care permit comunicarea cu *server*-ul *Oracle*, din orice utilitar sau aplicație.

SQL*Plus* este un utilitar *Oracle, având comenzi proprii specifice, care recunoaşte instrucțiunile *SQL* şi le trimite *server*-ului *Oracle* pentru execuție.

Oracle9i a introdus interfața *iSQL*Plus*, la *SQL*Plus*. Aceasta se bazează pe un *browser Web*, prin intermediul căruia este permisă conectarea la sistemul *Oracle9i* şi efectuarea acțiunilor care sunt posibile prin *SQL*Plus*, cu unele excepții. Din *iSQL*Plus*, pot fi lansate comenzi *SQL*Plus*, *SQL* şi *PL/SQL*.

Dintre funcționalitățile mediului *SQL*Plus*, se pot enumera:

- editarea, executarea, salvarea şi regăsirea instrucțiunilor *SQL* şi a blocurilor *PL/SQL*;
- calculul, stocarea şi afişarea rezultatelor furnizate de cereri;
- listarea structurii tabelelor;
- accesarea şi copierea de informații dintr-o bază de date în alta;
- administrarea bazei de date.

Variabile de substituție

O variabilă de substituție este definită de utilizator. O comandă în care apare o variabilă de substituție va fi executată de *SQL*Plus* ca şi cum ar conține o valoare efectivă. Variabilele de substituție pot fi utilizate oriunde în comenzile *SQL* şi *SQL*Plus*, dar nu pot apărea ca prim cuvânt la *prompt*-ul de comandă.

Când *SQL*Plus* întâlneşte într-o comandă o variabilă de substituție nedefinită, va solicita utilizatorului introducerea unei valori. Aceasta poate fi orice şir de caractere şi poate conține *blank*-uri sau semne de punctuație. Dacă variabila este de tip caracter şi nu a fost inclusă între apostrofuri în comanda *SQL*, utilizatorul va trebui să includă între apostrofuri valoarea introdusă.

*SQL*Plus* citeşte de la tastatură valoarea introdusă şi listează forma liniei ce conține variabila de substituție, înainte şi după înlocuirea valorii introduse. Această listare poate fi suprimată prin comanda *SET VERIFY OFF*.

Dacă valoarea furnizată unei variabile de substituție coincide cu numele altei variabile, atunci conținutul acesteia va fi utilizat în locul valorii respective.

Dacă o variabilă de substituție apare de mai multe ori într-o comandă şi este precedată de un singur caracter „&“, valoarea ei va fi solicitată utilizatorului de tot atâtea ori. Pentru a evita acest lucru, se vor utiliza două caractere „&“ în fața variabilelor de substituție. *SQL*Plus* definește automat (ca şi când ar fi folosită comanda *DEFINE*) variabilele de substituție precedate de „&&“, dar nu şi pe cele precedate de „&“. Dacă *var* este o variabilă definită prin comanda *DEFINE*, atunci

*SQL*Plus* foloseşte valoarea acesteia în locul fiecărei variabile de substituţie care face referinţă la *var* fie că este precedată de „&“, fie de „&&“. *SQL*Plus* nu va mai solicita valoarea lui *var* în sesiunea curentă, până la execuţia unei comenzi *UNDEFINE var*.

Variabilele de substituţie nu pot fi folosite în comenzile de editare a *buffer*-ului (*APPEND, CHANGE, DEL* şi *INPUT*). Aceste comenzi tratează textul care începe cu simbolul „&“ sau „&&“ ca pe un şir de caractere.

Caracteristici ale mediului *SQL*Plus*

***SQL*Plus* stochează** ultima comandă *SQL* sau ultimul bloc *PL/SQL* introdus într-o zonă de memorie numită ***buffer SQL*.** Conţinutul *buffer*-ului *SQL* se schimbă la introducerea următoarei comenzi *SQL* sau bloc *PL/SQL*.

În *buffer*-ul *SQL*, **nu** se stochează caracterele **„;“ şi „/“** utilizate pentru execuţia unei comenzi.

Comenzile *SQL*Plus* **nu** sunt păstrate în *buffer*-ul *SQL*.

Activarea interfeţei *SQL*Plus*

1. Din *WINDOWS*:

Se selecteaza: *START > PROGRAMS > ORACLE > APPLICATION DEVELOPMENT > SQL*Plus*

Se da *username, password* si numele bazei.

2. Prin linia de comanda:

SQLPLUS [*nume_utiliz/parola*][*@nume_baza_de_date*] [*@nume_fisier*]

Conexiune la *SQL*Plus*

După ce utilizatorul se conectează la SQL*Plus, sistemul afişează un prompt (SQL>) şi aşteaptă comenzile utilizatorului. Utilizatorul poate da:

comenzi SQL pentru accesarea bazei de date;

blocuri PL/SQL pentru accesarea bazei de date;

comenzi SQL*Plus.

După ce utilizatorul se conectează la *SQL*Plus*, sistemul afişează un *prompt* (***SQL>*)** şi aşteaptă comenzile utilizatorului. Utilizatorul poate da:

- comenzi *SQL* pentru accesarea bazei de date;
- blocuri *PL/SQL* pentru accesarea bazei de date

Închiderea sesiunii de lucru *SQL*Plus* şi preluarea controlului sistemului de operare al calculatorului gazdă se realizează cu *QUIT* sau *EXIT*.

Tabelul următor evidenţiază diferenţele dintre instrucţiunile *SQL* şi cele *SQL*Plus*:

SQL	***SQL*Plus***
Este un limbaj de comunicare cu *server*-ul *Oracle* pentru accesarea datelor.	Este un utilitar care recunoaşte instrucţiunile *SQL* şi le transferă *server*-ului *Oracle*.
Se bazează pe standardul *ANSI* pentru *SQL*.	Este o interfaţă specifică sistemului *Oracle* pentru execuţia instrucţiunilor *SQL*.
SQL*Plus stochează ultima comandă SQL sau ultimul bloc PL/SQL introdus într-o zonă de memorie numită **buffer SQL**	Comenzile SQL*Plus nu sunt depuse în buffer-ul SQL;
Prelucrează date şi defineşte obiecte din baza de date.	Nu permite prelucrarea informaţiilor din baza de date.
Nu are un caracter de continuare.	Acceptă „–“ drept caracter de continuare pentru comenzile scrise pe mai multe linii.
Instrucţiunile nu pot fi abreviate.	Comenzile pot fi abreviate.
Utilizează funcţii pentru a efectua formatări.	Utilizează comenzi pentru formatarea datelor.

Pentru a personaliza mediul *SQL*Plus* (de exemplu, ora curentă să apară ca parte a *prompt*-ului de comandă) şi a păstra, în sesiunile viitoare, caracteristicile stabilite, se utilizează un fişier al sistemului de operare gazdă, numit ***login.sql*.** În acesta, pot fi adăugate instrucţiuni *SQL*, comenzi *SQL*Plus* sau blocuri *PL/SQL*. La pornirea utilitarului *SQL*Plus*, sunt rulate automat comenzile din acest fişier.

Setări în *SQL*Plus*

Comanda care permite controlul sesiunii *SQL*Plus* este ***SET*.** Efectul acesteia se păstrează până la sfârşitul sesiunii în care a fost executată. Pentru a păstra unele setări, comanda trebuie adăugată în fişierul *login.sql*.

Sintaxa generică a acestei comenzi este:

***SET** variabila_sistem valoare*

Componenta variabila_sistem controlează un aspect al mediului în care se desfăşoară sesiunea.

Parametrul *variabila_sistem* poate lua oricare din valorile care apar la execuția comenzii SHOW ALL.

În continuare, sunt prezentate câteva variabile ale sistemului şi semnificația acestora.

- *ARRAY*[*SIZE*] *nr_întreg* stabileşte numărul de linii (*batch*) pe care *SQL*Plus* le recuperează simultan din baza de date.
- *FEED*[*BACK*] {*nr_întreg* | *OFF* | *ON*} afişează numărul de linii returnate de o cerere, atunci când aceasta selectează un număr de înregistrări mai mare decât valoarea variabilei.
- *HEA*[*DING*] {*OFF* | *ON*} determină afişarea capetelor de coloană în rapoarte.
- *LONG* {*nr_întreg*} determină lățimea maximă pentru afişarea valorilor *LONG*, *CLOB*, *NCLOB* sau *XMLType*.
- *LINESIZE nr_întreg* setează lățimea, în caractere, a paginii pe care sunt afişate rezultatele interogărilor.
- *PAGESIZE nr_întreg* setează numărul de linii al unei pagini.
- *NUMFORMAT format* stabileşte formatul implicit pentru afişarea valorilor numerice în rezultatele interogărilor.
- *PAUSE* {*text* | *ON* | *OFF*} decide oprirea la începutul fiecărei pagini de rezultate, urmând ca defilarea să fie reluată după apăsarea tastei *enter*. Dacă se specifică *text*, acesta va fi afişat la fiecare oprire a defilării.
- *TIME* {*ON* | *OFF*} determină afişarea orei curente înaintea fiecărui *prompt* de comandă.
- *SUFFIX* permite stabilirea extensiei fişierelor *script*. Implicit, *SQL*Plus* adaugă numelui fişierului extensia *.sql*.
- *DEFINE* setează caracterul de substituție, care implicit este „&“.
- *ESCAPE* defineşte un caracter care, utilizat înainte de caracterul de substituție, determină tratarea acestuia drept un caracter obişnuit. Implicit, caracterul *escape* este „\“.
- *VERIFY* {*ON* | *OFF*} determină listarea fiecărei linii din fişierul de comenzi, înainte şi după substituție.
- *CONCAT* defineşte caracterul care separă numele unei variabile sau parametru de substituție de caracterele care îi urmează imediat. Implicit, acest caracter este „.“.
- *AUTOTRACE* {*OFF* | *ON* [*EXPLAIN* | *STATISTICS*] | *TRACEONLY*} determină generarea automată a unui raport asupra planului de execuție furnizat de optimizor şi a statisticilor asupra execuției instrucțiunilor

LMD. Opțiunile *ON EXPLAIN*, *ON STATISTICS* afişează numai planul de execuție al optimizorului, respectiv numai statisticile asupra execuției instrucțiunilor *SQL*. Clauza *TRACEONLY* are acelaşi efect ca şi *ON*, dar suprimă listarea rezultatului cererii utilizatorului.

- UND[ERLINE] {_ | c | OFF | ON} – specifică caracterul folosit pentru sublinierea numelor coloanelor.

Exista multe alte setari (în special cele folosite la formatarea rapoartelor) care pot fi vizualizate cu comanda SHWO ALL

Execuția instrucțiunilor *SQL* şi a blocurilor *PL/SQL*

În *SQL*Plus*, lansarea în execuție a unei instrucțiuni *SQL* sau a unui bloc *PL/SQL* este posibilă prin intermediul comenzilor prezentate în continuare.

- „/“ execută, fără a lista, instrucțiunea *SQL* sau blocul *PL/SQL* stocat curent în *buffer*-ul *SQL*.
- *R*[*UN*] listează şi execută instrucțiunea *SQL* sau blocul *PL/SQL* stocat curent în *buffer*-ul *SQL*.
- *EXEC*[*UTE*] *instrucțiune* execută o singură instrucțiune *PL/SQL* sau rulează o procedură stocată.
- *TIMI*[*NG*] [*START text* | *SHOW* | *STOP*] colectează şi afişează date asupra resurselor utilizate pentru execuția uneia sau mai multor instrucțiuni sau blocuri. Comanda colectează informații pentru o perioadă de timp încheiată, salvându-le într-un *timer*. Pentru ştergerea tuturor acestor *timer*-e, se utilizează comanda *CLEAR TIMING*.
- @ sau *STA*[*RT*]{*url* | *nume_fişier*[.*ext*]} [*lista_argumente*] lansează în execuție comenzile din fişierul specificat. Acesta se poate afla în sistemul de fişiere local sau pe un *server Web*.
- @@ *nume_fişier*[.*ext*] este o comandă similară celei precedente. Ea este utilă pentru execuția fişierelor imbricate de comenzi întrucât caută fişierul de comenzi specificat, în aceeaşi cale sau *URL* în care se află fişierul de comenzi din care a fost apelat.

Nu se pot utiliza parametri atunci când se rulează o instrucțiune prin intermediul comenzii *RUN* sau „/“. Instrucțiunea trebuie stocată într-un fişier, care va fi rulat prin comanda *START* sau „@“.

Editarea instrucțiunilor *SQL* şi a blocurilor *PL/SQL*

Anumite acțiuni asupra *buffer*-ului *SQL* determină ca o anumită linie să devină linia curentă a acestuia. Astfel, în urma:

- afişării unei linii prin comanda *LIST*, aceasta devine linia curentă;
- execuției comenzii LIST sau RUN asupra buffer-ului, ultima linie devine linia curentă;
- obținerii unui mesaj de eroare, linia conținând eroarea devine automat linia curentă.

Comenzile prezentate în continuare acționează asupra liniei curente din *buffer*-ul *SQL*.

- *A*[*PPEND*] *text* - adaugă textul specificat la sfârşitul liniei curente din *buffer*-ul *SQL*. Pentru a separa textul de caracterele precedente, se introduc două spații între *APPEND* şi *text*. Pentru a introduce un text care se termină cu „;“, comanda va fi încheiată prin două caractere „;“ (utilitarul *SQL*Plus* consideră un singur caracter „;“ drept terminator de comandă).
- *C*[*HANGE*] *car_separ text_vechi* [*car_separ* [*text_nou* [*car_separ*]]] modifică textul de pe linia curentă din *buffer*-ul *SQL*. Drept caracter de separare, poate fi utilizat orice caracter care nu este alfanumeric. Spațiul dintre cuvântul cheie *CHANGE* şi primul caracter de separare poate fi omis.
- *DEL* [*n* | *n m* | *n* *| *n LAST* | * | * *n* |* *LAST* | *LAST*] şterge una sau mai multe linii din *buffer*-ul *SQL*. Caracterul „*” indică linia curentă. Se poate omite spațiul dintre cuvântul cheie *DEL* şi *n* sau *, dar nu cel dintre *DEL* şi *LAST*. Comanda *DEL* fără clauze determină ştergerea liniei curente din *buffer*.
- *I*[*NPUT*] [*text*] adaugă una sau mai multe linii de text după linia curentă din *buffer*-ul *SQL*.
- *L*[*IST*] [*n* |*n m* | *n* * | *n LAST* | * | * *n* | * *LAST* | *LAST*] listează una sau mai multe linii din *buffer*-ul *SQL*. Sunt valabile precizările făcute la comanda *DEL*.
- *CLEAR SCREEN* determină ştergerea conținutului ecranului. Această comandă poate fi utilă, de exemplu, înainte de afişarea unui raport.
- *n text* – inlocuieste linia *n* prin *text*;

Exemplu :

>SQL select * from jobs
where jobtitle='Prezident'
i order by job_title;`
>SQL / -- executa
>SQL del 2sterge linia 2
>2 order by order by job_title – inlocuieste filtru where cu order

Crearea şi modificarea fişierelor de comenzi (*script*)

În mediul *SQL*Plus*, prelucrarea fişierelor de comenzi este permisă prin intermediul instrucțiunilor prezentate în continuare.

- ***ED*[*IT*] [*nume_fişier*[*.ext*]]** invocă un editor de text al sistemului de operare gazdă pentru a deschide fişierul de comenzi specificat sau pentru a edita *buffer*-ul *SQL*.
- *GET nume_fişier*[*.ext*] încarcă un fişier al sistemului de operare gazdă în *buffer*-ul *SQL*.
- ***SAV*[*E*]** *file_name*[*.ext*] salvează conținutul *buffer*-ului *SQL* într-un fişier al sistemului de operare gazdă.

```
SAVE    alfa
EDIT    alfa
@alfa
```

- ***STORE SET nume_fişier*[*.ext*]** salvează parametrii sesiunii *SQL*Plus* curente într-un fişier al sistemului de operare gazdă.
- *WHENEVER OSERROR* {*EXIT* | *CONTINUE*} poate determina ieşirea din mediul *SQL*Plus* dacă apare o eroare a sistemului de operare (de exemplu, o eroare de intrare/ieşire în/din fişier).
- *WHENEVER SQLERROR* {*EXIT* | *CONTINUE*} poate determina ieşirea din mediul *SQL*Plus* dacă o comandă *SQL* sau un bloc *PL/SQL* generează o eroare.

Într-un fişier de comenzi, comentariile pot fi plasate în trei moduri.

- Comanda *REM*[*ARK*] marchează începutul unui comentariu pe o singură linie într-un fişier *script*.
- Delimitatorii „/*...*/“ permit introducerea de comentarii, conținând una sau mai multe linii, în instrucțiunile *SQL* sau în blocurile *PL/SQL*.
- **Comentariile *ANSI/ISO* sunt marcate de caracterele „--“** şi permit introducerea unui comentariu pe o singură linie, în cadrul unei

instrucțiuni *SQL* sau bloc *PL/SQL*.

- Pentru a obține informații referitoare la structura tabelelor, vizualizărilor sau sinonimelor, a procedurilor, funcțiilor sau pachetelor fără a fi necesară consultarea cataloagelor de sistem, se utilizează comanda:

  DESC[RIBE] **[***nume_schema.*]*nume_obiect*

  Ex. Desc tabs

Interactivitate în *SQL*Plus*

Comenzile prezentate în continuare permit interacțiunea dintre mediul *SQL*Plus* şi utilizator.

- ***ACC*[*EPT*]** *variabila* [*tip*] [*PROMPT text*] citeşte o linie de intrare şi o stochează într-o variabilă utilizator.
- ***DEF*[*INE*]** [*variabila*] | [*variabila = text*] specifică o variabilă utilizator, căreia i se poate atribui o valoare de tipul *CHAR*. Fără argumente, comanda listează valorile şi tipurile tuturor variabilelor. Sistemul *Oracle9i* a introdus variabila *CONNECT_IDENTIFIER* care conține *SID*-ul corespunzător conexiunii utilizatorului. Aceasta permite ca informația asupra conectării să poată fi accesată ca şi oricare altă variabilă specificată prin comanda *DEFINE*.
- *PAU*[*SE*] [*text*] afişează o linie vidă, urmată de o linie conținând text, apoi aşteaptă ca utilizatorul să acționeze tasta *enter*.
- ***PROMPT*** [*text*] afişează mesajul specificat sau o linie vidă pe ecranul utilizatorului.
- *UNDEF*[*INE*] *variabila* permite ştergerea variabilelor definite de utilizator fie explicit, prin comanda *DEFINE*, fie implicit, printr-un argument în comanda *START*.

***Exemplu*:**

```
ACCEPT alfa PROMPT 'Numarul de exemplare:'
ACCEPT beta PROMPT 'Numele autorului:'
SELECT *
FROM   carte
WHERE  nrex = &alfa
AND    autor = '&beta';
```

Variabilele de substituție (*&nume*)

Aceste variabile sunt utilizate pentru stocarea temporară a unor valori. Variabilele pot să apară în comenzi SQL sau SQL*Plus. Interfața cere utilizatorului să dea valori de fiecare dată când întâlneşte o variabilă nedefinită. Dacă variabila este precedată de simbolurile "&&", doar la prima apelare se va solicita o valoare.

Exemplu:

```
SELECT    nume, &&salariu
FROM      salariat
ORDER BY  &salariu;
```

Pentru variabilele de tip caracter sau de tip dată calendaristică este obligatorie folosirea ghilimelelor. Variabilele *de substituție pot să apară în condiția WHERE, în clauza ORDER BY, în* expresia unei coloane, în numele unui tabel, în locul unei întregi comenzi *SELECT*.

***Exemplu*:**

```
SELECT   &coloana
FROM     &tabel
WHERE    &conditie
ORDER BY &ordine;
```

*Comanda SET VER[IFY] {ON | OFF} permite listarea (sau nu) textului unei comenzi SQL sau PL/SQL, inainte si dupa ce SQL*Plus inlocuieste variabilele de substitutie cu valori efective.*

*SQL*Plus permite definirea variabilelor utilizator de tip CHAR prin:*

DEFINE variabilă = valoare

*Variabila rămâne definită până când fie se părăseşte SQL*Plus, fie se dă comanda UNDEFINE pentru variabila respectivă.*

Tipărirea tuturor variabilelor utilizator, a valorilor şi tipurilor acestora se obține prin forma DEFINE.

Exemplu:

```
SQL> DEFINE autor1 = Zola
SQL> DEFINE autor2 = Blaga
SQL> SELECT  titlu, nrex
  2  FROM    carte
  3  WHERE   autor = '&autor1'
  4  OR      autor = '&autor2';
```

Crearea şi afişarea variabilelor de legătură

Variabilele de legătură (*bind variables*) se creează în *SQL*Plus* şi pot fi referite în *SQL* sau *PL/SQL*. Ele au următoarele funcționalități:

- permit afişarea în *SQL*Plus* a valorilor utilizate în blocuri *PL/SQL* (variabilele declarate în blocurile *PL/SQL* nu pot fi afişate în *SQL*Plus*);
- pot fi utilizate în mai multe blocuri *PL/SQL*, permițând comunicarea între acestea.

Mediul *SQL*Plus* furnizează comenzi utile în lucrul cu astfel de variabile.

- ***PRI*[*NT*] [*variabila*] afişează valoarea unei variabile de legătură.**
- ***VAR*[*IABLE*] [***variabila tip***] declară o variabilă de legătură care poate fi referită în blocurile *PL/SQL*. Dacă nu se specifică nici un argument, comanda *VARIABLE* listează toate variabilele de legătură create în sesiunea curentă.

Referirea unei variabile de legătură se realizează în *PL/SQL* precedând numele variabilei prin caracterul „**:**“.

Comenzi *SQL*Plus* diverse

Utilizatorul dispune de comenzi pentru conectarea sau deconectarea unui utilizator de la mediul *SQL*Plus*, descrierea obiectelor bazei de date, părăsirea sesiunii curente, afişarea unor informații totalizatoare sau a valorilor pentru variabilele sistem sau de mediu.

- Instrucțiunea *DESC*[*RIBE*] { [*schema*.]*nume_obiect*[*@id_conectare*] } listează definițiile coloanelor pentru tabelul, vizualizarea sau sinonimul specificat. Pentru o funcție sau procedură, această comandă listează specificația obiectului respectiv.
- {*EXIT* | *QUIT*} [*COMMIT* | *ROLLBACK*] salvează sau anulează modificările aflate în aşteptare, deconectează utilizatorul de la *Oracle*, închide *SQL*Plus* şi predă controlul sistemului de operare. Opțiunea *COMMIT* este implicită.
- Comanda *SHO*[*W*] *opțiune* afişează valorile pentru variabilele sistem *SQL*Plus* sau cele ale mediului *SQL*Plus* curent.

Opțiunile care pot fi prezente în comanda *SHOW* sunt următoarele: numele unei variabile a sistemului, *ALL*, *BTI*[*TLE*], *ERRORS*, *LNO*, *PNO*, *REL*[*EASE*], *REPF*[*OOTER*], *REPH*[*EADER*], *SGA*, *SQLCODE*, *TTI*[*TLE*], *USER*.

Clauza *ALL* determină listarea valorilor corespunzătoare tuturor opțiunilor comenzii *SHOW*, cu excepția lui *SGA* şi *ERRORS*.