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.

Comanda *SELECT*

Una dintre cele mai importante comenzi ale limbajului de prelucrare a datelor este *SELECT*. Cu ajutorul ei pot fi extrase submulţimi 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*}
FROM	{ [*schema.*]{*tabel* [*PARTITION* (*partition_name*)] |
	[*THE*] (*subquery*)} [*alias_tabel*] }
[*WHERE*	*condiţie*]
[*START WITH*	*condiţie*]
[*CONNECT BY*	*condiţie*]
[*GROUP BY*	*listă de expresii*
[*HAVING*	*condiţie*]]
[*ORDER BY*	{*expresie* | *poziţie* | *c_alias*} [*ASC* | *DESC*]]
[*FOR UPDATE*	[*OF* [*schema.*]{*table* | *view*}.*coloană*] [*NOWAIT*]

Prezenţa clauzelor *SELECT* şi *FROM* este obligatorie deoarece acestea specifică coloanele selectate, respectiv tabelele din care se vor extrage datele. Tabelele specificate în clauza *FROM* pot fi urmate de un *alias*, care va reprezenta numele folosit pentru referirea tabelului respectiv în cadrul instrucţiunii.

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 conţine *alias*-uri, care vor reprezenta numele câmpurilor respective în cadrul tabelului furnizat ca rezultat de instrucţiunea *SELECT*.

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

Clauza *GROUP BY* grupează înregistrările după anumite câmpuri; în cazul prezenţei acestei clauze, clauza *HAVING* poate impune restricţii 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 secvenţă crescătoare valorile *null* sunt afişate ultimele. Dacă nu se face nici o specificaţie, atunci ordinea de returnare este la latitudinea *server*-ului.

Clauzele *START WITH* şi *CONNECT BY* sunt utile pentru a construi cereri ierarhizate. Pentru a specifica înregistrarea rădăcină a arborelui se va folosi clauza *START WITH*. Dacă această clauză este omisă, fiecare înregistrare din tabel poate fi considerată ca înregistrare de start. Cu ajutorul clauzei *CONNECT BY* se pot specifica coloanele (*părinte* şi *copil*) care participă la relaţie. Prin ordinea apariţiilor acestor coloane (în condiţie) se poate determina ordinea de parcurgere a structurii arborescente (*top-down* sau *bottom-up*). Prin folosirea operatorului *PRIOR* se poate face referinţă la înregistrarea *părinte*.

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 excepţie şi nu se va mai aştepta până la ridicarea blocajelor de pe înregistrări.

Operatorii utilizaţi (în ordinea priorităţii de execuţie) sunt:

- operatori aritmetici (unari sau binari),
- operatorul de concatenare (||),
- operatorii de comparare (=, !=, ^=, < >, >, >=, <, <=, *IN* (echivalent cu =ANY, adică egal cu cel puţin una din valorile listei), *NOT IN* (echivalent cu *!=ALL*, adică diferit de toate elementele listei), *ALL*, [*NOT*] *BETWEEN x AND y*, [*NOT*] *EXISTS*, [*NOT*] *LIKE*, *IS* [*NOT*] *NULL*,
- operatori logici (*NOT*, *AND*, *OR*).

Limbajul permite prezenţa unor instrucţiuni *SELECT* **imbricate** în oricare din clauzele *SELECT*, *WHERE*, *HAVING* sau *FROM* (instrucţiunile *SELECT* care apar în clauzele respective se numesc **subcereri**).

În cazul folosirii subcererilor, pot fi utilizaţi operatorii *ALL*, *ANY*, *IN* (=*ANY*), *EXIST*, *NOT IN* (!=*ANY*), care sunt specifici cererilor ce returnează mai multe linii (*multiple-row subquery*) sau operatorii de comparaţie =, <, >, >=, <=, <>, specifici cererilor care returnează o singură linie (*single-row subquery*).

Executarea subcererilor se poate face:

- fie **cu sincronizare** (corelat → evaluarea subcererii face referinţă la o coloană a cererii principale şi cererea interioară se execută pentru fiecare linie a cererii principale care o conţine);
- fie **fără sincronizare** (încuibărit → se execută mai întâi cererea interioară, iar rezultatul ei este transmis cererii de nivel imediat superior).

Cereri mono – relaţie

Exemplu:

Dacă în interiorul *alias*-ului apare un spaţiu liber sau caractere speciale, atunci *alias*-ul trebuie scris între ghilimele.

```
SELECT   dateres-dataim  "numar zile"
FROM     imprumuta;
```

Exemplu:

Valorile de tip caracter şi de tip dată calendaristică trebuie să fie incluse între apostrofuri.

```
SELECT   codel
FROM     imprumuta
WHERE    datares >= '01-JAN-03';
```

Exemplu:

Să se obţină titlurile şi numărul de exemplare ale cărţilor scrise de autorii al căror nume începe cu litera S.

```
SELECT   titlu, nrex
FROM     carte
WHERE    autor LIKE 'S%';
```

***Exemplu*:**

Să se afişeze data şi ora curentă.

```
SELECT   TO_CHAR(SYSDATE,'DD/MM/YY HH24:MI:SS')
FROM     DUAL;
```

***Exemplu*:**

Utilizând ideea că directorul este salariatul care nu are şef, să se tipărească numele directorului.

```
SELECT   ename,NVL(TO_CHAR(mgr),'Nu are sef')
FROM     emp
WHERE    mgr IS NULL;
```

Intrebari

`NVL(x, y)` x si y trebuie sa fie de acelasi tip!
➔ `NVL(comm, 'nu are')` este corect?

```
SELECT  ename, job
FROM    emp
WHERE   mgr IS NULL;
```
Daca utilizati `mgr = NULL` este corect?

Se pot folosi alias-uri in clauza WHERE?
```
SELECT  titlu, pret*nrex pret_total
FROM    carte
WHERE   pret_total>1000;

SELECT  titlu.pret_total
FROM   (SELECT  titlu, pret*nrex pret_total
        FROM carte)
WHERE   pret_total>1000;
```

Exemplele anterioare sunt corecte? Comentati!

<nume angajat> castiga <salariu> lunar, dar doreste <salariu de 3 ori mai mare>
```
SELECT  ename||'castiga'||sal||'lunar, dar doreste'
        ||sal*3 "salariul ideal"
FROM    emp;
```

De ce este incorect?
```
SELECT titlu, MIN(pret)
FROM   carte;
```

Se pot folosi functii grup in clauza WHERE? NU!
```
SELECT  titlu
FROM    carte
WHERE   pret = MAX(pret);
SELECT  titlu
FROM    carte
WHERE   pret= (SELECT  MAX(pret)
               FROM    carte);
```

***Exemplu*:**

Să se afişeze codurile cititorilor care nu au împrumutat cărţi într-un interval precizat.

```
SELECT   DISTINCT codec
FROM     imprumuta
WHERE    dataim NOT BETWEEN '&d1' AND '&d2';
```

SURSA	**REZULTAT**
'''x'''	'x'
''''	'
''','''	','
''');'	');
''''||nume_dep||''''	'informatica'

Se poate folosi alias in clauza WHERE? Pentru coloane NU! Pentru tabele DA!

Clauza *GROUP BY*

Exemplele care urmează arată modul general de constituire a subansamblelor **virtuale** folosind clauza *GROUP BY*. Fiecare expresie care apare în *SELECT* trebuie să aibă aceeaşi valoare pentru toate liniile care aparţin aceleiaşi partiţii. Numele coloanelor din *GROUP BY* nu trebuie să figureze obligatoriu în lista de la *SELECT*.

Clauza *WHERE* are prioritate fata de *GROUP BY*. Nu se poate utiliza alias de coloana in clauza *GROUP BY*.

Pentru a returna informatie corespunxatoare fiecarui grup, pot fi utilizate functiile agregat. Acestea pot aparea in clauzele *SELECT*, *ORDER BY* si *HAVING*. Se poate utiliza functie grup in clauza *WHERE*? Este corect …*WHERE AVG*(*sal*) > *200*? NU!

Cand se utilizeaza *GROUP BY*, server-ul sorteaza implicit multimea rezultata in ordinea crescatoare a valorilor coloanelor dupa care se realizeaza gruparea.

Grupurile sunt formate si functiile grup sunt calculate, inainte ca clauza *HAVING* sa fie aplicata grupurilor.

***Exemplu*:**

Să se obţină numărul de câte ori a fost împrumutată fiecare carte.

```
SELECT   codel, COUNT(*)
FROM     imprumuta
GROUP BY codel;
```

Exemplu:

Pentru fiecare domeniu de carte să se obţină numărul cărţilor din domeniu, media preţurilor şi numărul total de exemplare.

```
SELECT   coded,COUNT(*),AVG(pret),SUM(nrex)
FROM     carte
GROUP BY coded;
```

Dacă în comanda *SELECT* apar atribute coloană (nu funcţii grup) şi se utilizează clauza *GROUP BY* atunci aceste coloane trebuie obligatoriu să apară în clauza *GROUP BY*.

***Exemplu*:**

Să se obţină pentru fiecare autor, media preţurilor cărţilor din bibliotecă.

```
SELECT       autor, AVG(pret)
FROM         carte
GROUP BY     autor;
```

***Exemplu*:**

Pentru departamentele în care salariul maxim depăşeşte 5000$ să se obţină codul acestor departamente şi salariul maxim pe departament.

```
SELECT       deptno, MAX(sal)
FROM         emp
GROUP BY     deptno
HAVING       MAX(sal)>5000;
```

***Exemplu*:**

```
SELECT       MAX(AVG(pret))
FROM         carte
GROUP BY     autor;
```

***Exemplu*:**

Să se afişeze numele şi salariul celor mai prost plătiţi angajaţi din fiecare departament.

```
SELECT   ename, sal
FROM     emp
WHERE    (deptno, sal) IN
              (SELECT    deptno, MIN(sal)
               FROM      emp
               GROUP BY  deptno);
```

Exemplu:

Să se obţină pentru fiecare carte, codul său şi numărul de exemplare care nu au fost încă restituite.

```
SELECT   codel, COUNT(*)
FROM     imprumuta
WHERE    dataef IS NULL
GROUP BY codel;
```

***Exemplu*:**

Să se obţină numărul cărţilor împrumutate cel puţin o dată.

```
SELECT   COUNT(DISTINCT codel)
FROM     imprumuta;
```

***Exemplu*:**

Să se afişeze numărul cărţilor împrumutate cel puţin de două ori (pentru fiecare carte împrumutată mai mult decât o dată să se obţină numărul de câte ori a fost împrumutată).

```
SELECT        COUNT(COUNT(codel))
FROM          imprumuta
GROUP BY      codel
HAVING        COUNT(*)>1;
```

În cererea anterioară *COUNT*(*codel*), reprezintă numărul care arată de câte ori a fost împrumutată fiecare carte, iar *COUNT*(*COUNT*(*codel*)), reprezintă numărul total al cărţilor împrumutate.

***Exemplu*:**

Sa se afiseze numărul de cărţi imprumutate din fiecare domeniu.

```
SELECT d.intdom, COUNT(*)
FROM   domeniu d, carte c, imprumuta I
WHERE  c.codel = i. codel
AND    c.coded = d.coded
GROUP  BY intdom;
```

***Exemplu*:**

Lista codurilor cititorilor care au mai mult de 3 cărţi nerestituite la termen.

```
SELECT   codec
FROM     imprumuta
WHERE    dataef IS NULL AND datares < SYSDATE
GROUP BY codec
HAVING   COUNT(*) > 2;
```

Exemplu:

Pentru fiecare domeniu de carte care conţine cel puţin o carte şi unde preţul oricărei cărţi nu depăşeşte o valoare dată, să se obţină: codul domeniului, numărul cărţilor din domeniu şi numărul mediu de exemplare.

```
SELECT   coded, COUNT(*), AVG(nrex)
FROM     carte
GROUP BY coded
HAVING   COUNT(*) >= 1
AND      MAX(pret) < &pret_dat;
```

Exemplu:

Codurile domeniilor care nu contin carti.

```
SELECT    coded
FROM      carte
GROUP BY  coded
HAVING    COUNT(*) = 0;
```

Nu este corect, deoarece se iau in considerare NUMAI codurile domeniilor care apar in tabelul CARTE.

```
SELECT    intdom
FROM      domeniu d
WHERE 0 = (SELECT  COUNT(*)
           FROM    carte
           WHERE   coded = d.coded);
```

Urmatoarea cerere este corecta?

```
SELECT intdom
FROM   domeniu d,(SELECT  coded, COUNT(*) a
                  FROM    carte
                  GROUP BY coded) b
WHERE   b.coded = d.coded)
AND     b.a = o;
```

***Exemplu*:**

În ce interogări este necesară utilizarea cuvântului cheie HAVING?

A. când este necesar să eliminăm linii duble din rezultat;

B. când este necesar să ordonăm mulţimea rezultat;

C. când este necesar să efectuăm un calcul pe grup;

D. când este necesar să restricţionăm grupurile de linii returnate.

Relaţii ierarhice

SQL permite afişarea rândurilor unui tabel ţinând cont de **relaţiile ierarhice** care apar între rândurile tabelului. O bază de date relaţională nu stochează înregistrările în mod ierarhic. Dacă există o relaţie ierarhică între liniile unui tabel, un proces de parcurgere a unui arbore (*tree walking*) permite construirea ierarhiei. O cerere ierarhică este o metodă de raportare, în ordine, a ramurilor arborelui.

Parcurgerea în mod ierarhic a informaţiilor se poate face doar la nivelul unui singur tabel. Operaţia se realizează cu ajutorul clauzelor *START WITH* şi *CONNECT BY*.

În comanda *SELECT* pot să apară clauzele:

CONNECT BY {*expresie = PRIOR expresie* | *PRIOR expresie = expresie*}
[*START WITH conditie*]

Clauza *START WITH* specifică nodul (înregistrarea de început) arborelui (punctul de start al ierarhiei). De exemplu,

```
START WITH last_nume = 'Ionescu'
```

Dacă lipseşte, orice nod poate fi rădăcină.

Clauza *CONNECT BY* specifică coloanele prin care se realizează relaţia ierarhică. De fapt, relaţia „părinte-copil“ a unei structuri arborescente permite controlul direcţiei în care este parcursă ierarhia şi stabilirea rădăcinii ierarhiei.

Operatorul *PRIOR* face referinţă la linia „părinte“. Plasarea acestui operator determină direcţia interogării, dinspre „părinte“ spre „copil“ (*top-down*) sau invers (*bottom-up*).

Liniile „părinte“ ale interogării sunt identificate prin clauza *START WITH*. Pentru a găsi liniile „copil“, *server*-ul evaluează expresia din dreptul operatorului *PRIOR* pentru linia „părinte“, şi cealaltă expresie pentru fiecare linie a tabelului. Înregistrările pentru care condiţia este adevărată vor fi liniile „copil“. Spre deosebire de *START WITH*, în clauza *CONNECT BY* nu pot fi utilizate subcereri.

Clauza *SELECT* poate conţine pseudo-coloana *LEVEL*, care indică nivelul înregistrării în arbore (cât de departe este de nodul rădăcină). Nodul rădăcină are nivelul 1, fii acestuia au nivelul 2 ş.a.m.d.

Pentru eliminarea unor portiuni din arbore, pot fi utilizate clauzele *WHERE* si *CONNECT BY*. De exemplu, putem elimina un nod cu clauza *WHERE* si putem elimina o ramura utilizand clauza *CONNECT BY*.

***Exemplu*:**

Se presupune că fiecare salariat are un singur superior (ierarhie). Să se afişeze superiorii ierarhic lui Ion.

```
SELECT  LEVEL, nume
FROM    salariat
CONNECT BY  nume = PRIOR nume_sef
START WITH  nume = (SELECT  nume_sef
                    FROM    salariat
                    WHERE   nume = 'Ion');
```

***Exemplu* (bottom up):**

Lista sefilor incepand cu salariatul avand *ID*-ul 101.

```
SELECT     employee_id, last_name, job_id, salary
FROM       employees
START WITH employee_id = 101
CONNECT BY PRIOR manager_id = employee_id;
```

In exemplul care urmeaza, *employee_id* este evaluat pentru linia parinte, iar *manager_id* si *salary* sunt evaluate pentru liniile copil. Operatorul *PRIOR* se aplica la valoarea *employee_id*.

```
... CONNECT BY PRIOR employee_id = manager_id
                     AND salary > 15000;
```

Prin urmare o linie copil trebuie sa aiba valoarea lui *manager_id* egala cu valoarea *employee_id* a liniei parinte si trebuie sa aiba salariul mai mare ca 15000.

***Exemplu* (top down):**

Lista cu numele salariatilor si a sefilor acestora.

```
SELECT     last_name||'are sef pe'||PRIOR last_name
FROM       employees
START WITH last_name = 'KING'
CONNECT BY PRIOR employee_id = manager_id;
```

Rezultatul va fi o singura coloana.

***Exemplu*:**

Se pleaca din radacina ierarhiei, parcurgere *top down* si se elimina salariatul Ionescu, dar se proceseaza liniile copil ale acestuia.

```
SELECT     employee_id, last_name, job_id, salary
FROM       employees
WHERE      last_name != 'Ionescu'
START WITH manager_id IS NULL
CONNECT BY PRIOR employee_id = manager_id;
```

Se pleaca din radacina ierarhiei, parcurgere *top down* si se elimina salariatul Ionescu si toate liniile sale copil.

```
SELECT     employee_id, last_name, job_id, salary
FROM       employees
START WITH manager_id IS NULL
CONNECT BY PRIOR employee_id = manager_id
AND        last_name != 'Ionescu';
```

***Exemplu*:**

Să se afişeze codul, titlul, data creării şi data achiziţiei operelor, astfel încât fiecare operă să fie urmată de cele achiziţionate în anul creării sale. Prima linie afişată va fi cea corespunzătoare operei având codul 110.

```
SELECT     cod_opera, titlu, data_crearii, data_achizitiei
FROM       opera
START WITH cod_opera = 110
CONNECT BY PRIOR TO_CHAR(data_crearii, 'yyyy') =
                 TO_CHAR(data_achizitiei, 'yyyy');
```

***Exemplu*:**

Să se afişeze codul, titlul, data creării şi data achiziţiei operelor, astfel încât fiecare operă să fie urmată de cele achiziţionate în anul creării sale. Se vor considera doar operele a căror valoare este mai mare decât 7000. Prima linie afişată va fi cea corespunzătoare operei create cel mai recent.

```
SELECT     cod_opera, titlu, data_crearii, data_achizitiei,
           valoare
FROM       opera
START WITH data_crearii = (SELECT   MAX(data_crearii)
                           FROM     opera)
CONNECT BY PRIOR TO_CHAR(data_crearii, 'yyyy') =
                 TO_CHAR(data_achizitiei, 'yyyy')
                 AND valoare > 7000;
```

În clauza *CONNECT BY*, coloana *data_crearii* este evaluată pentru linia „părinte“, iar coloanele *data_achizitiei* şi *valoare* sunt evaluate pentru linia „copil“.

Cereri multi – relaţie

Comanda *SELECT* oferă posibilitatea de a consulta informaţii care provin din mai multe tabele. Operatorii care intervin în astfel de cereri pot fi:

- operatori pe mulţimi (*UNION*, *UNION ALL*, *INTERSECT*, *MINUS*);
- operatori compunere care implementează diferite tipuri de *join*.

Există două moduri de realizare a cererilor multi-relaţie:

- forma procedurală, în care trebuie indicat drumul de acces la informaţie prin imbricarea de comenzi *SELECT*;
- forma relaţională, în care drumul de acces la informaţie este în sarcina sistemului.

***Exemplu*:**

Să se obţină, utilizând aceste două forme, codurile şi titlurile cărţilor împrumutate.

a) Forma procedurală (imbricare de comenzi *SELECT*):

```
SELECT   codel, titlu
FROM     carte
WHERE    codel IN (SELECT      DISTINCT codel
                   FROM        imprumuta);
```

b) Forma relaţională:

```
SELECT   carte.codel, titlu
FROM     carte, imprumuta
WHERE    carte.codel = imprumuta.codel;
```

Operatori pe mulţimi (*UNION*, *UNION ALL*, *INTERSECT*, *MINUS*)

Comenzile *SELECT*, care intervin în cereri ce conţin **operatori pe mulţimi**, trebuie să satisfacă anumite condiţii:

- toate comenzile *SELECT* trebuie să aibă acelaşi număr de coloane;
- opţiunea *DISTINCT* este implicită (excepţie *UNION ALL*);
- numele coloanelor sunt cele din prima comandă *SELECT*;
- dimensiunea coloanei implicit este cea mai mare dintre cele două coloane;
- sunt admise combinaţii de forma:

1. `SELECT1 UNION SELECT2 INTERSECT SELECT3` şi ordinea de execuţie este de la stânga la dreapta;
2. `SELECT1 UNION (SELECT2 INTERSECT SELECT3)` şi ordinea este dată de paranteze.

***Exemplu*:**

Să se obţină, utilizând operatorul *INTERSECT*, codurile cărţilor din care sunt mai puţin de 15 exemplare şi care au fost împrumutate de cel puţin trei ori.

```
SELECT   codel
FROM     carte
WHERE    nrex < 15
INTERSECT
SELECT   codel
FROM     imprumuta
GROUP BY codel
HAVING   COUNT(*) > 3;
```

***Exemplu*:**

Să se afişeze codurile cititorilor care nu au împrumutat cărţi.

```
SELECT   codec
FROM     cititor
MINUS
SELECT   DISTINCT codec
FROM     imprumuta;
```

***Exemplu*:**

Să se listeze codul operelor, codul artiştilor şi numele acestora, utilizând operatorul UNION

```
SELECT cod_opera, cod_artist, TO_CHAR(null) nume
FROM   opera
UNION
SELECT TO_NUMBER(null), cod_artist, nume
FROM   artist
```

Operaţii de compunere

Un **join simplu** (*natural join*) este o instrucţiune *SELECT* care returnează linii din două sau mai multe tabele. Este preferabil ca tabelul care are linii mai puţine să fie al doilea în operaţia de compunere. Comanda durează mai puţin, dacă tabela este indexată după coloana, relativ la care se face compunerea. Compunerea a *n* tabele cere minim (*n*-1) condiţii de *join*.

***Exemplu*:**

Să se obţină codurile şi titlurile cărţilor împrumutate.

```
SELECT   carte.codel, titlu
FROM     carte, imprumuta
WHERE    carte.codel = imprumuta.codel;
```

S-ar putea ca tabelele legate prin operaţia de compunere să nu aibă coloane comune (*non-equijoin*). În acest caz în clauza *WHERE* nu apare operatorul egalitate şi sunt folosiţi operatorii: <=, >=, *BETWEEN*.

Pentru a simplifica scrierea şi pentru a elimina ambiguităţile care pot să apară este necesară folosirea *alias*-ului pentru tabele. *Alias*-ul este valid doar pentru instrucţiunea SELECT curentă.

***Exemplu*:**

Să se obţină pentru fiecare salariat numele, salariul şi grila de salarizare (*Θ join*).

```
SELECT   e.ename, e.sal, s.grade
FROM     emp e, salgrade s
WHERE    e.sal BETWEEN s.lasal AND s.hisal;
```

***Exemplu*:**

Să se obţină titlurile şi preţurile cărţilor mai scumpe decât cartea având titlul “Baze de date”, al cărui autor este Oszu (*self join*).

```
SELECT   x.titlu, x.pret
FROM     carte x, carte y
WHERE    x.pret > y.pret
AND      y.titlu = 'Baze de date'
AND      y.autor = 'Oszu';
```

O altă variantă de rezolvare a problemei, ca o cerere cu sincronizare:

```
SELECT   titlu, pret
FROM     carte x
WHERE    EXISTS
```

```
(SELECT  *
 FROM    carte
 WHERE   carte.titlu='Baze de date'
 AND     carte.autor='Oszu'
 AND     x.pret > pret);
```

Exemplu:

Să se obţină informaţii despre cititorii al căror cod este mai mare decât codul unui cititor având un nume dat.

a) Forma procedurală de rezolvare a cererii este următoarea:

```
SELECT   *
FROM     cititor
WHERE    codec > (SELECT   codec
                  FROM     cititor
                  WHERE         nume='&nume1');
```

b) Forma relaţională pentru a rezolva cererea este următoarea:

```
SELECT   c2.*
FROM     cititor c1, cititor c2
WHERE    c1.nume = '&nume1'
AND      c2.codec > c1.codec;
```

Dacă o linie nu satisface condiţia de *join*, atunci linia respectivă nu va apare în rezultatul cererii. Pentru a evita această pierdere, în algebra relaţională a fost introdus operatorul ***outer-join***.

Un *outer-join* (join extern) este reprezentat prin operatorul (+) care este plasat în clauza *WHERE* după numele tabelului ale cărui linii trebuie să nu se piardă din rezultatul cererii. Semnul (+) poate fi plasat în oricare parte a condiţiei din clauza *WHERE*, însă nu în ambele părţi. Efectul operatorului (+) este că se generează valori *null* pentru coloanele tabelului lângă care apare scris, ori de câte ori tabelul nu are nici o linie care să poată fi reunită cu o linie din celălalt tabel.

***Exemplu*:**

Să se obţină titlurile cărţilor şi numele domeniului căruia îi aparţin, remarcând si situaţiile în care domeniul nu ar avea cărţi (dacă domeniul este fără cărţi atunci apare *null* la titlul cărţii).

```
SELECT   titlu, intdom
FROM     carte, domeniu
WHERE    carte.coded(+) = domeniu.coded;
```

***Exemplu*:**

Considerăm că tabelele *dept* şi *emp* au următorul conţinut:

```
dept                    emp

deptno   dname         empno    deptno
1        algebra       101      null
2        analiza       102      null
                       103      null
                       105      1
                       106      1
```

Interogarea următoare furnizează lista tuturor salariaţilor si informatii despre departamentele in care lucreaza, inclusiv a celor care nu sunt asignaţi nici unui departament (*right outher join*).

```
SELECT   a.deptno, a.dname, b.empno, b.deptno
FROM     dept a, emp b
WHERE    a.deptno(+) = b.deptno;
```

Rezultatul cererii anterioare va fi:

```
a.deptno a.dname  b.empno  b.deptno
                  101
                  102
                  103
1        algebra  105      1
1        algebra  106      1
```

Interogarea următoare afişează lista departamentelor, inclusiv a celor care nu au salariaţi **(***left outer join***)**.

```
SELECT   a deptno, a.dname, b.empno, b.deptno
FROM     dept a, emp b
WHERE    a.deptno = b.deptno(+);
```

Rezultatul cererii anterioare va fi:

```
a.deptno a.dname  b.empno  b.deptno
1        algebra  105      1
1        algebra  106      1
2        analiza  null     null
```

Interogarea următoare produce ca rezultat departamentele, chiar şi cele fără funcţionari, şi funcţionarii, chiar şi cei care nu sunt asignaţi nici unui departament (*full outer join*).

```
SELECT   NVL(TO_CHAR(b.empno),'***') id,
         NVL(a.dname,'***') nume_dep
FROM     dept a, emp b
WHERE    a.deptno = b.deptno(+)
UNION
SELECT   NVL(TO_CHAR(b.empno),'***') id,
         NVL(a.dname,'***') nume_dep
FROM     dept a, emp b
WHERE    a.deptno(+) = b.deptno;
```

Rezultatul cererii va fi:

```
id        nume_dep
***       analiza
101       ***
102       ***
103       ***
105       algebra
106       algebra
```

Ce aduce nou *Oracle9i*?

O sintaxa noua, dar fara performante!

```
SELECT tabel1.coloana, tabel2.coloana
FROM tabel1
[NATURAL JOIN tabel2] |
[JOIN tabel2 USING (nume_coloana)] |
[JOIN tabel2
   ON (tabel1.nume_coloana = tabel2.nume_coloana)]|
[LEFT|RIGHT|FULL OUTER JOIN tabel2
   ON (tabel1.nume_coloana = tabel2.nume_coloana)]|
[CROSS JOIN tabel2];
```

Observatii

- Clauza *NATURAL JOIN* se bazeaza pe toate coloanele care au acelasi nume in cele doua tabele. Daca coloanele care au acelasi nume au tipuri diferite, atunci eroare.

- Clauza *USING* permite specificarea numai anumitor coloane care vor apare in conditia de equijoin. Coloanele care apar in clauza *USING* nu pot fi precedate de alias sau nume tabel, oriunde ar apare in cadrul comenzii *SELECT*.
- Pot sa apara join-uri pe multiple tabele, ordinea de executie fiind de la stanga la dreapta.
- O conditie join care contine un operator diferit de operatorul de egalitate defineste un non-equijoin.
- Clauza *CROSS JOIN* implementeaza produsul cartezian.

***Exemplu*:**

```
SELECT nume, dataim, titlu
FROM   cititor
JOIN   imprumuta USING (codec)
JOIN   carte USING (codel);
```

***Exemplu*:**

```
SELECT a.nume, a.sal, b.nivel
FROM   salariat a JOIN salgrade b
       ON a.sal BETWEEN b.lower AND b.higher;
```

***Exemplu*:**

```
SELECT a.nume, b.cod_dep, b.nume
FROM   salariat a FULL OUTER JOIN departament b
       ON (a.cod_dep = b.cod_dep);
```

Subcereri

De cele mai multe ori, pentru a implementa anumite interogări, nu este suficientă o singură cerere *SELECT* ci sunt necesare **subcereri**. Subcererile sunt comenzi *SELECT* încapsulate în oricare din clauzele *SELECT*, *WHERE*, *HAVING*, *FROM*.

Dacă subcererea urmează clauzei *WHERE* sau *HAVING*, ea poate conţine unul dintre operatorii *ALL*, *ANY*, *IN* (*=ANY*), *EXIST*, *NOT IN* (*!=ALL*) care sunt specifici cererilor care întorc mai multe linii (*multiple-row subquery*) sau unul dintre operatorii de comparare (=, <, >, >=, <=, <>) care sunt specifici cererilor care întorc o singură linie (*single-row subquery*).

Subcererile trebuie incluse între paranteze şi trebuie plasate în partea dreaptă a operatorului de comparare. Subcererea nu poate conţine clauza *ORDER BY*.

***Exemplu*:**

Să se obţină numele şi salariul angajaţilor, având salariul minim.

```
SELECT   ename, sal
FROM     emp
WHERE    sal=(SELECT  MIN(sal)
              FROM    emp);
```

***Exemplu*:**

Să se obţină *job*-ul pentru care salariul mediu este minim. Sa se afiseze si salariul mediu.

```
SELECT   job, AVG(sal)
FROM     emp
GROUP BY job
HAVING   AVG(sal)=(SELECT      MIN(AVG(sal))
                   FROM        emp
                   GROUP BY    job);
```

Operatorul ***ANY*** presupune că este adevărată condiţia dacă comparaţia este adevărată pentru cel puţin una din valorile returnate. Sunt evidente relaţiile:

< ANY ⇔ mai mic ca maximul;

> ANY ⇔ mai mare ca minimul;

= ANY ⇔ IN.

Pentru operatorul *ALL* se presupune că este adevărată condiţia, dacă comparaţia este adevărată pentru toate elementele listei returnate. Pentru operatorul *ALL* sunt evidente relaţiile:

< ALL ⇔ mai mic ca minimul;

> ALL ⇔ mai mare ca maximul;

! = ALL ⇔ NOT IN.

***Exemplu*:**

WHERE codec > ALL (‘C1’, ‘C2’) ⇔ este superior tuturor elementelor din listă;

WHERE codec > ANY (‘C1’, ‘C2’) ⇔ este superior cel puţin unui element din listă.

***Exemplu*:**

Să se obţină salariaţii al căror salariu este mai mare ca salariile medii din toate departamentele.

```
SELECT   ename, job
FROM     emp
WHERE    sal > ALL(SELECT      AVG(sal)
                   FROM        emp
                   GROUP BY    deptno);
```

Există subcereri care au ca rezultat mai multe coloane (*multiple-column subquery*). Aceste interogări au următoarea sintaxă generală:

```
SELECT   col,col,…
FROM     tabel
WHERE    (col,col,…) IN (SELECT    col,col,…
                         FROM      tabel
                         WHERE     condiţie);
```

***Exemplu*:**

Să se obţină numele, numărul departamentului, salariul şi comisionul tuturor funcţionarilor ale căror salarii şi comisioane coincid cu salariile şi comisioanele unor salariaţi din departamentul 7.

```
SELECT   ename, deptno, sal, com
FROM     emp
WHERE    (sal,NVL(com,-1)) IN
             (SELECT      sal,NVL(com,-1)
              FROM        emp
              WHERE       deptno = 7);
```

Rezultatul acestei interogări este diferit de rezultatul următoarei interogări:

```
SELECT   ename, deptno, sal, com
FROM     emp
WHERE    sal IN (SELECT   sal
                 FROM     emp
                 WHERE    deptno=7)
AND       NVL(com,-1) IN (SELECT   NVL(com,-1)
                          FROM     emp
                          WHERE    deptno=7);
```

Dacă una din valorile returnate de subcerere este valoarea *null* atunci cererea nu întoarce nici o linie. Prin urmare, dacă valoarea *null* poate să facă parte din rezultatul subcererii nu trebuie utilizat operatorul NOT IN. Problema nu mai apare dacă se utilizează operatorul IN.

***Exemplu*:**

Să se obţină salariaţii care nu au subordonaţi.

```
SELECT   e.ename
FROM     emp e
WHERE    e.empno NOT IN (SELECT  m.mgr
                         FROM    emp m);
```

În acest caz, instrucţiunea *SQL* nu întoarce nici o linie deoarece una din valorile furnizate de subcerere este valoarea *null*.

***Exemplu*:**

Să se obţină numele salariaţilor, salariile, codul departamentului în care lucrează şi salariul mediu pe departament pentru toţi angajaţii care au salariul mai mare ca media salariilor din departamentul în care lucrează (folosirea subcererii în clauza *FROM*).

```
SELECT  a.ename,a.sal,a.deptno,b.salavg
FROM    emp a,(SELECT   deptno,avg(sal) salavg
               FROM     emp
               GROUP BY deptno) b
WHERE    a.deptno=b.deptno
AND      a.sal>b.salavg
```

Exemplu:

Să se obţină lista celor mai scumpe cărţi.

```
SELECT   titlu
FROM     carte
WHERE    pret = (SELECT   MAX(pret)
                 FROM     carte);
```

Exemplu:

Să se obţină lista scriitorilor care au în bibliotecă un număr de exemplare mai mare decât numărul mediu al cărţilor din bibliotecă.

```
SELECT   DISTINCT autor
FROM     carte
WHERE    nrex > (SELECT   AVG(nrex)
                 FROM     carte);
```

Exemplu:

Să se obţină informaţii despre cărţile al căror preţ depăşeşte media preţurilor cărţilor ce aparţin aceluiaşi domeniu

```
SELECT   *
FROM     carte c
WHERE    pret > (SELECT   AVG(pret)
                 FROM     carte
                 WHERE    coded = c.coded);
```

***Exemplu*:**

Să se obţină lista cititorilor care au împrumutat cel puţin o carte.

```
SELECT   nume
FROM     cititor
WHERE    codec IN (SELECT  DISTINCT codec
                   FROM    imprumuta);
```

***Exemplu*:**

Să se obţină codurile cititorilor care nu au împrumutat niciodată cărţi.

```
SELECT   codec
FROM     cititor
WHERE    codec NOT IN
                   (SELECT DISTINCT codec
                    FROM   imprumuta);
```

***Exemplu*:**

Să se obţină lista cititorilor care sunt în întârziere cu predarea cărţilor.

```
SELECT   nume
FROM     cititor
WHERE    codec IN (SELECT DISTINCT codec
                   FROM    imprumuta
                   WHERE   dataef IS NULL
                   AND     dares<SYSDATE);
```

***Exemplu*:**

Să se obţină numele cititorilor care au împrumutat cel puţin o carte scrisă de ZOLA.

```
SELECT   nume
FROM     cititor
WHERE    codec IN
              (SELECT  DISTINCT codec
               FROM    imprumuta
               WHERE   codel IN
                   (SELECT  codel
                    FROM    carte
                    WHERE   autor='ZOLA'));
```

***Exemplu*:**

Să se obţină numele cititorilor care nu au împrumutat nici o carte scrisă de ZOLA.

```
SELECT   nume
FROM     cititor
WHERE    codec NOT IN
              (SELECT  DISTINCT codec
               FROM    imprumuta
               WHERE   codel IN
                      (SELECT  codel
                       FROM    carte
                       WHERE   autor='ZOLA'));
```

Operatorul *IN* poate fi înlocuit cu = *ANY* (un element este în listă dacă şi numai dacă este egal cu un element al listei), iar operatorul *NOT IN* poate fi înlocuit prin *!=ALL*.

***Exemplu*:**

Să se obţină codurile cititorilor care au împrumutat o carte de algebră.

```
SELECT   DISTINCT codec
FROM     imprumuta
WHERE    codel IN
             (SELECT   codel
              FROM     carte
              WHERE    coded=
                    (SELECT  coded
                     FROM    domeniu
                     WHERE   intdom='ALGEBRA'));
```

***Exemplu*:**

Să se obţină cititorii care au împrumutat numai cărţi scrise de 'ZOLA'.

```
SELECT   nume
FROM     cititor
WHERE    codec NOT IN
         (SELECT  DISTINCT codec
          FROM    imprumuta
          WHERE   codel NOT IN
                   (SELECT  codel
                    FROM    carte
                    WHERE   autor='ZOLA'));
```

***Exemplu*:**

Să se obţină numele cititorilor care au împrumutat cel puţin o carte de informatică (procedural).

```
SELECT   nume
FROM     cititor
WHERE    codec IN
(SELECT  DISTINCT codec
 FROM    imprumuta
 WHERE   codel IN
     (SELECT  codel
      FROM    carte
      WHERE   coded=
             (SELECT  coded
              FROM    domeniu
              WHERE   intdom= 'INFORMATICA')));
```

***Exemplu*:**

Să se obţină numele cititorilor şi titlurile cărţilor de informatică împrumutate de aceşti cititori (relational).

```
SELECT   nume, titlu
FROM     cititor, carte, imprumuta, domeniu
WHERE    imprumuta.codel = carte.codel
AND      carte.coded = domeniu.coded
AND      imprumuta.codec = cititor.codec
AND      intdom = 'INFORMATICA';
```

Subcererile pot fi executate corelat (**cu sincronizare**) sau încuibărit (**fără sincronizare**).

Subcererile fără sincronizare sunt caracterizate de faptul că se execută cererea cea mai interioară care întoarce un rezultat ce este transmis cererii de nivel superior, care întoarce un rezultat s.a.m.d.

Subcererile cu sincronizare sunt caracterizate de faptul că evaluarea subcererii face referinţă la o coloană a cererii principale, iar evaluarea cererii interioare se face pentru fiecare linie a cererii (principale) care o conţine.

***Exemplu*:**

Să se obţină, utilizând sincronizarea subcererii cu cererea principală, titlurile cărţilor care au toate exemplarele împrumutate (se selectează un titlu din carte şi pentru acest titlu se numără câte exemplare sunt împrumutate).

```
SELECT   titlu
FROM     carte
WHERE    nrex=(SELECT COUNT(*)
               FROM    imprumuta
               WHERE   codel = carte.codel
               AND     dataef IS NULL);
```

***Exemplu*:**

Să se obţină codurile cititorilor şi codul ultimei cărţi împrumutate.

```
SELECT   codec, codel
FROM     imprumuta i
WHERE    dataim>=ALL (SELECT   dataim
                      FROM     imprumuta
                      WHERE    codec=i.codec);
```

Pentru această interogare, clauza *WHERE* putea fi scrisă şi sub forma:

```
WHERE    dataim=(SELECT    MAX(dataim)
                 FROM      imprumuta
                 WHERE     codec=i.codec);
```

***Exemplu*:**

Să se obţină lista codurilor cărţilor împrumutate şi codul primului cititor care a împrumutat aceste cărti.

```
SELECT   codel,codec
FROM     imprumuta i
WHERE    dataim<=ALL (SELECT  dataim
                      FROM    imprumuta
                      WHERE   i.codel=codel);
```

***Exemplu*:**

Să se obţină codurile cărţilor din care cel puţin un exemplar este împrumutat.

```
SELECT   codel
FROM     carte
WHERE    EXISTS
         (SELECT  codel
          FROM    imprumuta
          WHERE   codel = carte.codel
          AND     dataef IS NULL);
```

Operatorul *WHERE EXISTS* (*subcerere*) presupune că predicatul este adevărat dacă subcererea întoarce cel puţin un tuplu, iar *WHERE NOT EXISTS* (*subcerere*) presupune că predicatul este adevărat dacă subcererea nu întoarce nici un tuplu. ➔ *EXISTS* şi *NOT EXISTS* cer sincronizarea subcererii.

***Exemplu*:**

Să se obţină titlurile cărţilor care sunt momentan împrumutate.

Soluţia 1 (cu sincronizare):

```
SELECT   titlu
FROM     carte
WHERE    EXISTS
           (SELECT    *
            FROM      imprumuta
            WHERE     codel = carte.codel
            AND       dataef IS NULL);
```

Soluţia 2 (fără sincronizare):

```
SELECT   titlu
FROM     carte
WHERE    codel IN
           (SELECT    DISTINCT codel
            FROM      imprumuta
            WHERE     dataef IS NULL);
```

***Exemplu*:**

Să se obţină codurile cărţilor care nu au fost împrumutate niciodată.

Soluţia 1 (cu sincronizare)

```
SELECT   codel
FROM     carte
WHERE    NOT EXISTS
              (SELECT  codel
               FROM    imprumuta
               WHERE   codel = carte.codel);
```

Soluţia 2 (fără sincronizare)

```
SELECT   codel
FROM     carte
WHERE    codel NOT IN
              (SELECT  DISTINCT codel
               FROM    imprumuta);
```

***Exemplu*:**

Să se obţină lista salariaţilor având salariul minim în departamentul în care lucrează.

```
SELECT   ename,sal
FROM     emp e
WHERE    sal=(SELECT  MIN(sal)
              FROM    emp
              WHERE   deptno=e.deptno);
```

***Exemplu*:**

Să se obţină numele primilor trei salariaţi având retribuţia maximă (ideea rezolvării este de a verifica dacă numărul salariaţilor care au leafa mai mare decât leafa salariatului considerat, este mai mic decât 3).

```
SELECT   ename
FROM     emp a
WHERE    3>(SELECT    COUNT(*)
            FROM      emp
            WHERE     sal > a.sal);
```

***Exemplu*:**

Să se obţină numele cititorilor care au împrumutat **cel puţin** aceleaşi cărţi ca şi cititorul având codul C19 (ideea problemei este de a selecta cititorii pentru care este vidă lista cărţilor împrumutatede C19 mai puţin lista cărţilor împrumutate de acei cititori).

```
SELECT   nume
FROM     cititor
WHERE    NOT EXISTS
              (SELECT  codel
               FROM    imprumuta
               WHERE   codec='C19'
                 MINUS
                SELECT  codel
               FROM    imprumuta
               WHERE   codec= cititor.codec);
```

Dacă problema era modificată în sensul că „cel puţin”este înlocuit prin „cel mult” atunci trebuiau inversate interogările legate prin *MINUS*.

***Exemplu*:**

Să se obţină codurile cititorilor care au împrumutat aceleaşi cărţi ca şi cititorul având un cod specificat.

Rezolvarea problemei se bazează pe ideea: A = B ⇔ A ⊂ B şi B ⊂ A ⇔ (A-B) = ∅ şi (B-A) = ∅ ⇔ A-B şi B-A nu furnizează nici un tuplu rezultat.

```
SELECT   codec
FROM     imprumuta i
WHERE    NOT EXISTS
              (SELECT  codel
               FROM    imprumuta
               WHERE   codec=i.codec
                  MINUS
               SELECT  codel
               FROM    imprumuta
               WHERE   codec='&ccc')
AND      NOT EXISTS
              (SELECT  codel
               FROM    imprumuta
               WHERE   codec='&ccc'
                  MINUS
               SELECT  codel
               FROM    imprumuta
               WHERE   codec=i.codec)
AND      codec!='&ccc');
```

Ultimul operator (AND), asigură să nu apară în rezultat cititorul specificat.

În cazul **formei relaţionale** de rezolvare a cererii, drumul de acces la informaţie este în sarcina SGBD-lui şi prin urmare nu mai apar cereri imbricate.

***Exemplu*:**

Să se obţină numele cititorilor care au împrumutat cel puţin o carte.

Soluţia 1 (forma relaţională):

```
SELECT   DISTINCT nume
FROM     cititor,imprumuta
WHERE    cititor.codec=imprumuta.codec;
```

Soluţia 2 (forma procedurală):

```
SELECT   nume
FROM     cititor
WHERE    codec IN
           (SELECT    DISTINCT codec
            FROM      imprumuta);
```

***Exemplu*:**

Să se obţină numele cititorilor care au împrumutat cel puţin două cărţi.

Soluţia 1 (forma relaţională):

```
SELECT   nume
FROM     cititor, imprumuta
WHERE    cititor.codec=imprumuta.codec
GROUP BY nume
HAVING   COUNT(*)>1;
```

Soluţia 2 (forma procedurală):

```
SELECT   nume
FROM     cititor
WHERE    codec IN
           (SELECT    codec
            FROM      imprumuta
            GROUP BY  codec
            HAVING    COUNT(*)>1);
```

Exemplu:

Să se afişeze numele, prenumele, salariul lucrătorilor, codurile publicaţiilor la care lucrează şi salariul mediu pe publicaţie pentru toţi angajaţii care au salariul mai mare decât media salariului pe publicaţia respectivă.

```
SELECT   s.nume, s.prenume, s.salariu,
         p.nr_publicatie, a.salariu_mediu
FROM     salariat s, publicatie p,
     (SELECT   p1.nr_publicatie,AVG(salariu) salariu_mediu
      FROM     publicatie p1, salariat s1
      WHERE    p1.cod_salariat = s1.cod_salariat
      GROUP BY  p1.nr_publicatie) a
WHERE    p.nr_publicatie = a.nr_publicatie
AND      s.cod_salariat = p.cod_salariat
AND      s.salariu > a.salariu_mediu;
```

Exemplu:

Să se obţină numele salariaţilor care nu cunosc nici o limbă străină.

```
SELECT   nume, prenume
FROM     salariat
WHERE    NOT EXISTS
     (SELECT *
     FROM   limba
     WHERE  limba.cod_salariat = salariat.cod_salariat
     AND    limba_cun IS NOT NULL);
```

Exemplu:

Să se afişeze graficienii care au întârziat să predea *frame*-urile.

a) cu sincronizare:

```
SELECT   nume, prenume
FROM     salariat
WHERE    EXISTS
     (SELECT  *
     FROM     realizeaza r
     WHERE    salariat.cod_salariat=r.cod_salariat
     AND      data_lim < SYSDATE);
```

b) fără sincronizare:

```
SELECT   nume, prenume
FROM     salariat
WHERE    cod_salariat IN
     (SELECT  DISTINCT cod_salariat
      FROM    realizeaza
      WHERE   data_lim < SYSDATE);
```

Exemplu:

Să se determine revistele coordonate de redactori şefi care nu cunosc limba în care sunt scrise. Se ştie că în urma inspectării vizuale a rezultatului interogării se poate decide schimbarea redactorilor şefi ai revistelor respective, de aceea se doreşte blocarea înregistrărilor găsite.

```
SELECT   p.nr_publicatie
FROM     salariat s, publicatie p
WHERE    s.cod_salariat = p.cod_salariat
AND      p.limba NOT IN
     (SELECT  limba_cun
      FROM    limba
      WHERE   limba.cod_salariat = s.cod_salariat)
FOR UPDATE OF p.cod_salariat;
```

Clauza *WITH*

Cu ajutorul clauzei *WITH* se poate defini un bloc de cerere înainte ca acesta să fie utilizat într-o interogare. Clauza permite reutilizarea aceluiaşi bloc de cerere într-o instrucţiune *SELECT* complexă.

Utilizând clauza *WITH*, să se scrie o cerere care afişează numele artiştilor şi valoarea totală a operelor acestora. Se vor considera artiştii a căror valoare totală a operelor este mai mare decât media valorilor operelor tuturor artiştilor.

```
WITH
val_artist AS (SELECT   nume, SUM(valoare) AS total
               FROM     opera o, artist a
               WHERE    o.cod_artist = a.cod_artist
               GROUP BY nume),
val_medie  AS (SELECT   SUM(total)/COUNT(*) AS medie
               FROM     val_artist)
SELECT   *
FROM     val_artist
WHERE    total > (SELECT   medie
                  FROM     val_medie)
ORDER BY nume;
```

Subcereri scalare

Subcererile scalare în *SQL* returnează valoarea unei singure coloane corespunzătoare unei linii. Dacă subcererea returnează 0 linii, valoarea subcererii scalare este *null*. Dacă subcererea returnează mai mult de o linie, *server*-ul generează o eroare.

Subcererile scalare erau acceptate în *Oracle8i* doar în anumite cazuri, cum ar fi clauzele *FROM* şi *WHERE* ale instrucţiunii *SELECT* sau clauza *VALUES* a instrucţiunii *INSERT*. Utilitatea subcererilor scalare a fost extinsă în *Oracle9i*. Astfel, ele pot apărea în:

- condiţiile şi expresiile care fac parte din *DECODE* sau *CASE*;
- toate clauzele instrucţiunii *SELECT*, cu excepţia lui *GROUP BY*;
- în partea stângă a operatorului, în clauzele *SET* şi *WHERE* ale instrucţiunii *UPDATE*.

Exemplu:

Să se afişeze codul, titlul operelor şi numele artistului doar dacă acesta este Brâncuşi. În caz contrar, se va afişa şirul „alt artist“.

```
SELECT   cod_opera, titlu,
        (CASE WHEN cod_artist =
                    (SELECT   cod_artist
                     FROM     artist
                     WHERE    nume = 'Brancusi')
         THEN 'Brancusi'
         ELSE 'Alt artist' END) artist
FROM     opera;
```

Operatorul *ROLLUP*

Operatorul *ROLLUP* produce o mulţime care conţine liniile obţinute 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 direcţie, de la dreapta la stânga, de-a lungul listei de coloane specificate în clauza *GROUP BY*. Apoi, se aplică funcţia 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 conţine 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);
```

Instrucţiunea 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 combinaţiilor posibile ale expresiilor specificate şi returnează câte o linie totalizatoare pentru fiecare grup. El produce subtotaluri pentru toate combinaţiile 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

Funcţia *GROUPING*

Aceasta funcţie este utilă pentru:

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

Clauza *GROUPING SETS*

GROUPING SETS reprezintă o extensie a clauzei *GROUP BY* care permite specificarea unor grupări multiple de date.

Această extensie, apărută în sistemul *Oracle9i*, permite scrierea unei singure instrucţiuni *SELECT* pentru a specifica grupări diferite (care pot conţine operatorii *ROLLUP* şi *CUBE*), în loc de mai multe instrucţiuni *SELECT* combinate prin operatorul *UNION ALL*. De altfel, reuniunea rezultatelor mai multor cereri este ineficientă întrucât necesită mai multe parcurgeri ale aceloraşi date.

Operatorii *ROLLUP* şi *CUBE* pot fi consideraţi cazuri particulare de mulţimi de grupări. Au loc următoarele echivalenţe:

CUBE(*a*, *b*, *c*)	***GROUPING SETS*** ((*a*, *b*, *c*), (*a*, *b*), (*a*, *c*), (*b*, *c*), (*a*), (*b*), (*c*), ())
ROLLUP(*a*, *b*, *c*)	***GROUPING SETS*** ((*a*, *b*, *c*), (*a*, *b*), (*a*), ())

***Exemplu*:**

Considerând galeriile al căror cod este mai mic decât 50, să se calculeze media valorilor operelor:

- pentru fiecare galerie şi, în cadrul acesteia, pentru fiecare artist;
- pentru fiecare artist şi, în cadrul acestuia, pentru anii de achiziţie corespunzători.

```
SELECT   cod_galerie, cod_artist,
         TO_CHAR(data_achizitiei, 'yyyy') "an achizitie",
         AVG(valoare) "Valoare medie"
FROM     opera WHERE    cod_galerie < 50
GROUP BY GROUPING SETS
         ((cod_galerie, cod_artist),
          (cod_artist, TO_CHAR(data_achizitiei, 'yyyy')));
```

Mulţimea rezultat este constituită din valorile medii pentru fiecare dintre cele două grupuri ((cod_galerie, cod_artist) si (cod_artist, an_achizitie)) şi are forma următoare:

COD_GALERIE	COD_ARTIST	An achizitie	Valoare medie
10	50		3500
10	60		2500
40	50		2020
	50	2000	2380
	50	2002	2300
	60	2001	2000
	60	2003	3000

Exemplul precedent poate fi rezolvat şi prin următoarea instrucţiune compusă:

```
SELECT   cod_galerie, cod_artist, NULL "An achizitie",
         AVG(valoare) "Valoare medie"
FROM     opera
GROUP BY cod_galerie, cod_artist
UNION ALL
SELECT   NULL, cod_artist,
         TO_CHAR(data_achizitiei, 'yyyy'), AVG(valoare)
FROM     opera
GROUP BY cod_artist, TO_CHAR(data_achizitiei, 'yyyy');
```

În absenţa unui optimizor care analizează blocurile de cerere şi generează planul de execuţie, cererea precedentă va parcurge de două ori tabelul de bază (*opera*), ceea ce poate fi ineficient. Din acest motiv, este recomandată utilizarea extensiei *GROUPING SETS*.

Coloane compuse

O coloană compusă este o colecţie de coloane care sunt tratate unitar în timpul calculelor asupra grupurilor. Pentru a specifica o coloană compusă, aceasta se include între paranteze. În operaţia *ROLLUP*(*a*, (*b*, *c*), *d*), coloanele *b* şi *c* formează o coloană compusă şi sunt tratate unitar.

În general, coloanele compuse sunt utile pentru operaţiile *ROLLUP*, *CUBE* şi *GROUPING SETS*. De exemplu, în *CUBE* sau *ROLLUP* coloanele compuse pot determina eliminarea agregării de pe anumite niveluri.

Clauza *GROUP BY ROLLUP*(*a*, (*b*, *c*)) este echivalentă cu următoarea instrucţiune compusă (în care se precizează doar forma clauzelor *GROUP BY*):

GROUP BY a, b, c UNION ALL
GROUP BY a UNION ALL
GROUP BY ( )

Astfel, (*b*, *c*) sunt tratate unitar şi operaţia *ROLLUP* nu va fi efectuată asupra grupurilor în care coloanele *b* şi *c* nu apar simultan. Acest lucru este similar situaţiei în care este definit un *alias x* pentru (*b*, *c*), iar specificaţia clauzei *GROUP BY* este *GROUP BY ROLLUP*(*a*, *x*).

În instrucţiunea precedentă, *GROUP BY* () reprezintă instrucţiunea *SELECT* cu valori *null* pentru coloanele *a* şi *x*. Această clauză este folosită pentru generarea totalurilor generale:

```
SELECT          null, null, coloană_agregat
FROM            nume_tabel
GROUP BY ();
```

Următorul tabel prezintă câteva specificaţii care utilizează operatorii *ROLLUP*, *CUBE*, *GROUPING SETS*, împreună cu instrucţiunile compuse echivalente acestora:

GROUP BY ROLLUP(*a*, *b*, *c*)	*GROUP BY a, b, c UNION ALL* *GROUP BY a, b UNION ALL* *GROUP BY a*
GROUP BY CUBE((*a*, *b*), *c*)	*GROUP BY a, b, c UNION ALL* *GROUP BY a, b UNION ALL* *GROUP BY c UNION ALL* *GROUP BY*()
GROUP BY GROUPING SETS(*a*, *b*, *c*)	*GROUP BY a UNION ALL* *GROUP BY b UNION ALL* *GROUP BY c*
GROUP BY GROUPING SETS (*a*, *b*, (*b*, *c*))	*GROUP BY a UNION ALL* *GROUP BY b UNION ALL* *GROUP BY b, c*
GROUP BY GROUPING SETS((*a*, *b*, *c*))	*GROUP BY a, b, c*
GROUP BY GROUPING SETS(*a*, (*b*), ())	*GROUP BY a UNION ALL* *GROUP BY b UNION ALL* *GROUP BY* ()
GROUP BY GROUPING SETS (*a*, *ROLLUP*(*b*, *c*))	*GROUP BY a UNION ALL* *GROUP BY ROLLUP*(*b*, *c*)

***Exemplu*:**

Să se afişeze următoarele informaţii:

- valoarea medie a operelor de artă din fiecare galerie;
- valoarea medie a operelor de artă pentru fiecare galerie, iar în cadrul acesteia pentru fiecare artist şi fiecare an de achiziţie;
- media generală a tuturor valorilor operelor de artă.

```
SELECT   cod_galerie, cod_artist,
         TO_CHAR(data_achizitiei,'yyyy')"an achizitie",
         AVG(valoare) "Valoare medie"
FROM     opera
GROUP BY ROLLUP
   (cod_galerie,
    (cod_artist, TO_CHAR(data_achizitiei, 'yyyy')));
```

Exemplul precedent poate fi rezolvat utilizând cererea compusă prezentată mai jos. Folosirea coloanelor compuse este recomandată pentru asigurarea unei execuţii eficiente.

```
SELECT   cod_galerie, cod_artist,
         TO_CHAR(data_achizitiei, 'yyyy'),
         AVG(valoare) "Valoare medie"
FROM     opera
GROUP BY cod_galerie, cod_artist,
         TO_CHAR(data_achizitiei, 'yyyy')
UNION ALL
SELECT   cod_galerie, TO_NUMBER(null), TO_CHAR(null),
         AVG(valoare) "Valoare medie"
FROM     opera
GROUP BY cod_galerie
UNION ALL
SELECT TO_NUMBER(null), TO_NUMBER(null), TO_CHAR(null),
         AVG(valoare) "Valoare medie"
FROM     opera
GROUP BY ();
```

Concatenarea grupărilor

Concatenarea grupărilor reprezintă o modalitate concisă de a genera combinaţii de grupări. Acestea se specifică prin enumerarea mulţimilor de grupări (*grouping sets*) şi a operaţiilor *ROLLUP*, *CUBE* separate prin virgulă.

De exemplu, expresia *GROUP BY GROUPING SETS*(*a*, *b*), *GROUPING SETS*(*c*, *d*) defineşte grupările (*a*, *c*), (*a*, *d*), (*b*, *c*), (*b*, *d*).

Concatenarea mulţimilor de grupări este utilă atât pentru uşurinţa dezvoltării cererilor, cât şi pentru aplicaţii. Codul *SQL* generat de aplicaţiile *OLAP* implică deseori concatenarea mulţimilor de grupări, în care fiecare astfel de mulţime defineşte grupările necesare pentru o dimensiune.

***Exemplu*:**

Să se determine media valorilor operelor luând în considerare următoarele grupări: (*cod_galerie*, *cod_artist*, *an_achizitie*), (*cod_galerie*, *cod_artist*), (*cod_galerie*, *an_achizitie*), (*cod_galerie*).

```
SELECT   cod_galerie, cod_artist,
         TO_CHAR(data_achizitiei, 'yyyy') an_achizitie,
         AVG(valoare)
FROM     opera
GROUP BY cod_galerie, ROLLUP(cod_artist),
         CUBE(TO_CHAR(data_achizitiei, 'yyyy'));
```

Funcţii analitice

Funcţiile analitice calculează o valoare agregat pe baza unui grup de înregistrări. Ele diferă de funcţiile agregat prin faptul că, pentru fiecare grup, pot fi returnate mai multe linii rezultat.

Aceste funcţii reprezintă ultimul set de operaţii efectuat la procesarea unei interogări, înaintea clauzei *ORDER BY*. Din acest motiv, o funcţie analitică poate apărea numai în lista *SELECT* sau în clauza *ORDER BY*.

***Exemplu*:**

Pentru fiecare operă de artă, să se afle numărul de creaţii ale căror valori sunt cu cel mult 1000 mai mici şi cu cel mult 2000 mai mari decât valoarea operei respective.

```
SELECT titlu, valoare,
       COUNT(*) OVER (ORDER BY valoare
                      RANGE BETWEEN 1000 PRECEDING
                            AND     2000 FOLLOWING) AS nr_cr
FROM   opera;
```

Cuvântul cheie *OVER* indică faptul că funcţia operează pe mulţimea de rezultate a cererii, adică după evaluarea celorlalte clauze.

Opţiunea *RANGE* defineşte, pentru fiecare linie, o „fereastră“ (o mulţime de linii). Funcţia analitică va fi aplicată tuturor liniilor din această mulţime.

Funcţii 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 sa fie:

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

Funcţii 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

- funcţia *TO_CHAR* converteşte data calendaristică sau informaţia numerică în şir de caractere conform unui format;
- funcţia *TO_NUMBER* converteşte un şir de caractere în număr;

- funcţia *TO_DATE* converteşte un şir de caractere în dată calendaristică conform unui format.

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

Alte funcţii de conversie sunt: *CHARTOROWID*, *CONVERT*, *HEXTORAW*, *RAWTOHEX*, *ROWIDTOCHAR* etc., iar denumirea semnificativă arată rolul fiecăreia.

***Exemplu*:**

```
SELECT    TO_DATE('Feb 22,1981','Mon dd,YYYY')
FROM      DUAL;
```

Funcţii pentru prelucrarea caracterelor

- *LENGTH*(*string*) – returnează lungimea şirului de caractere *string*;
- *LENGTHB*(*string*) – îndeplineşte aceaşi funcţie ca şi *LENGTH*, cu deosebirea că returnează numărul de octeţi ocupaţi;
- *SUBSTR*(*string*, *start* [,*n*]) – returnează subşirul lui *string* care începe pe poziţia *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 spaţiile libere din stânga lui *string*;
- *RTRIM*(*string* [,*'chars'*]) – este similar funcţiei *LTRIM*, cu excepţia 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 spaţii libere la stânga lui *string*;
- *RPAD*(*string*, *length* [,*'chars'*]) – este similar funcţiei *LPAD*, dar adăugarea de caractere se face la dreapta şirului;
- *REPLACE*(*string1*, *string2* [,*string3*]) – returnează *string1* cu toate apariţiile lui *string2* înlocuite prin *string3*; dacă *string3* nu este specificat, atunci toate apariţiile 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 poziţia *start*, a *n*-a apariţie a secvenţei *chars* şi întoarce poziţia respectivă; dacă *start* nu este specificat, căutarea se face de la începutul şirului; dacă *n* nu este specificat, se caută prima apariţie a secvenţei *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;
- *SOUNDEX*(*string*) – returnează reprezentarea fonetică a şirului de caractere specificat;
- *TRANSLATE*(*string*, *from*, *to*) – fiecare caracter care apare în şirurile de caractere *string* şi *from* este transformat în caracterul corespunzător (aflat pe aceeaşi poziţie ca şi în *from*) din şirul de caractere *to*;

Funcţii aritmetice

Cele mai importante funcţii 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 împărţirii a două numere specificate), *POWER* (ridicarea la putere), *SIGN* (semnul unui număr), *COS* (cosinus), *COSH* (cosinus hiperbolic), *SIN*(sinus), *SQRT*(rădăcina pătrată), *TAN*(tangent), funcţiile *LEAST* şi *GREATEST*, care returnează cea mai mică, respectiv cea mai mare valoare a unei liste de expresii etc.

Funcţii 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 nopţii);
- *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.

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`.

`DD "of" MONTH` va avea ca efect `12 of OCTOBER`

Operaţii cu date calendaristice

Operaţie	**Rezultat**	**Descriere**
Data + număr	Data	Adaugă un număr de zile la o dată
Data - număr	Data	Scade un număr de zile dintr-o dată
Data - data	Număr zile	Scade două date calendaristice
Data + numar/24	Data	Adună un număr de ore la o dată

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	Ziua 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

Formatul RR este comentat pe urmatorul exemplu:

Anul curent	Data specificata	Format RR	Format YY
1995	27-OCT-95	1995	1995
1995	27-OCT-17	2017	1917
2001	27-OCT-17	2017	2017
2001	27-OCT-95	1995	2095

***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;
```

Funcţii 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 funcţia întoarce valoarea *else* (selecţie 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) calificare
FROM   salariat;
```

***Exemplu*:**

Pentru înregistrările tabelului *opera*, să se afişeze titlul, data achiziţiei, valoarea şi o coloană reprezentând valoarea operei după ce se aplică o mărire, astfel: pentru operele achiziţionate în 1998 creşterea este de 20%, pentru cele cumpărate în 1999 creşterea este de 15%, iar valoarea celor achiziţionate în anul 2000 creşte cu 10%. Pentru operele cumpărate în alţi ani valoarea nu se modifică.

```
SELECT   titlu, data_achizitiei, valoare,
       CASE TO_CHAR(data_achizitiei, 'yyyy')
          WHEN '1998' THEN valoare * 1.20
          WHEN '1999' THEN valoare * 1.15
          WHEN '2000' THEN valoare * 1.10
          ELSE valoare
       END "Valoare marita"
FROM     opera;
```

Instrucţiunea din acest exemplu poate fi rescrisă utilizând funcţia *DECODE*:

```
SELECT   titlu, data_achizitiei, valoare,
     DECODE (TO_CHAR(data_achizitiei, 'yyyy'),
             '1998', valoare * 1.20,
             '1999', valoare * 1.15,
             '2000', valoare * 1.10,
              valoare) "Valoare marita"
FROM     opera;
```

Funcţii 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* (deviaţia standard),
- *VARIANCE* (dispersia).

Observaţii:

- Funcţiile grup operează pe un grup de linii şi nu cer folosirea clauzei *GROUP BY*.
- Funcţiile grup ignoră valorile *null*.
- Orice funcţie grup întoarce o singură valoare.
- Ele întorc valoarea *null* când sunt aplicate unei mulţimi vide, cu excepţia operatorului *COUNT* care întoarce valoarea zero.
- Spre deosebire de funcţiile *COUNT*, *MIN* şi *MAX* care pot fi aplicate unor câmpuri numerice sau nenumerice, restul funcţiilor grup se aplică doar câmpurilor numerice.
- Funcţiile grup pot să apară în lista de la *SELECT* sau în clauza *HAVING*.

***Exemplu*:**

Să se afişeze numărul cărţilor distincte împrumutate.

```
SELECT   COUNT(DISTINCT codel)
FROM     imprumuta;
```

***Exemplu*:**

Comanda care urmează este greşită! De ce?

```
SELECT   titlu, COUNT(*)
FROM     carte;
```

***Exemplu*:**

Să se calculeze media preţurilor cărţilor din bibliotecă.

```
SELECT   AVG(pret)
FROM     carte;
```

***Exemplu*:**

```
SELECT  MAX(pret) - MIN(pret) diferenta
FROM carte;
```

***Exemplu*:**

Să se obţină suma, media valorilor, valoarea minimă şi cea maximă pentru operele de artă expuse în galeria având codul 30. De asemenea, se va afişa numărul de opere şi numărul de artişti care au creaţii expuse în această galerie.

```
SELECT   SUM(valoare) Suma, AVG(valoare) Media,
         MIN(valoare) Minim, MAX(valoare) Maxim,
         COUNT(*) Numar,
         COUNT(DISTINCT cod_artist) "Numar artisti"
FROM     opera
WHERE    cod_galerie = 30;
```

Întrucât funcţiile grup ignoră valorile *null*, această instrucţiune va returna media valorilor pe baza liniilor din tabel pentru care există o valoare validă stocată în coloana *valoare*. Aceasta înseamnă că suma valorilor se împarte la numărul de valori diferite de *null*. Pentru a calcula media pe baza tuturor liniilor din tabel, se utilizează:

```
SELECT   AVG(NVL(valoare, 0))
FROM     opera;
```

***Exemplu*:**

Să se afişeze media valorilor operelor de artă pentru fiecare galerie şi, în cadrul acesteia, pentru fiecare artist.

```
SELECT   cod_galerie, cod_artist, AVG(valoare)
FROM     opera
GROUP BY cod_galerie, cod_artist;
```

Comanda *INSERT*

INSERT INTO *nume_tabel / nume_view* [(*col1*[, *col2*[,…]])]
VALUES (*expresia1*[, *expresia2*[,…]]) */ subcerere*;

- *expresia1*, *expresia2*, reprezintă expresii a căror evaluare este atribuită coloanelor precizate (se inserează o linie);
- *subcerere*, reprezintă o interogare (se inserează una sau mai multe linii).

Observaţii:

- Dacă lipseşte specificaţia coloanelor se consideră că sunt completate toate câmpurile tabelului sau vizualizării.
- Dacă nu a fost specificată lista coloanelor şi dacă există câmpuri care nu au valori efective, atunci valoarea *null* va fi atribuită acestor câmpuri.
- Dacă se introduc date doar în anumite coloane, atunci aceste coloane trebuie specificate. În restul coloanelor se introduce automat *null* (daca nu exista *DEFAULT*).
- Specificarea cererii din comanda *INSERT* determină copierea unor date dintr-un tabel în altul pe atâtea linii câte au rezultat din cerere.
- Dacă se introduc numai anumite câmpuri într-o înregistrare, atunci printre acestea trebuie să se găsească câmpurile cheii primare.
- Pentru a putea executa comanda *INSERT* este necesar ca utilizatorul care execută această instrucţiune să aibă privilegiul de a insera înregistrări în tabel sau în vizualizare.

***Exemplu*:**

Să se insereze în tabelul *carte* toate cărţile din tabelul *carte_info*, presupunând că tabelul *carte_info* a fost deja creat. De asemenea, să se introducă o nouă carte căreia i se cunoaşte codul (c34), titlul (algebra) şi preţul (500).

```
INSERT INTO  carte
SELECT       *
FROM         carte_info;
INSERT INTO  carte(codel,titlu,autor,nrex,pret,coded)
VALUES       ('c34','algebra',null,null,500,null);
```

***Exemplu*:**

```
INSERT INTO carte(codel, nrex)
VALUES ('c25', 25);
INSERT INTO domeniu
VALUES ('&cod','&intdom');inserare prin parametrizare
```

***** Exemplu*:**

```
INSERT INTO
   (SELECT cod_opera, titlu, data
    FROM   opera
    WHERE  cod_galerie = 40)
VALUES (…);
```

***** Exemplu*:**

```
INSERT INTO opera(cod_opera,…)
VALUES (123,…)
RETURNING valoare*10, cod_opera INTO :x, :y;
```

***Exemplu*:**

Presupunând că tabelul *salariat* a fost completat cu datele tuturor salariaţilor editurii, să se completeze tabelele *grafician*, *tehnoredactor* şi *redactor_sef*, în concordanţă cu datele conţinute în tabelul *salariat* (nu pot exista graficieni, tehnoredactori sau redactori şefi care să nu fie salariaţi!).

```
INSERT   INTO grafician (cod_salariat)
SELECT   cod_salariat
FROM     salariat
WHERE    job = 'grafician';
INSERT   INTO tehnoredactor (cod_salariat)
SELECT   cod_salariat
FROM     salariat
WHERE    job = 'tehnoredactor';
INSERT   INTO redactor_sef (cod_salariat)
SELECT   cod_salariat
FROM     salariat
WHERE    job = 'redactor_sef';
```

***Exemplu*:**

Se doreşte ca toţi graficienii având salariile mai mari decât media salariilor să colaboreze la realizarea tuturor *frame*-urilor din publicaţii coordonate de redactori şefi având vechimea maximă. Să se completeze tabelul *realizeaza* cu înregistrările corespunzătoare.

```
INSERT   INTO realizeaza (cod_salariat, nr_publicatie,
         nr_capitol, nr_frame)
SELECT   s.cod_salariat,f.nr_publicatie, f.nr_capitol,
         f.nr_frame
FROM     salariat s, frame f
WHERE    s.salariu > (SELECT   AVG(s1.salariu)
                      FROM     salariat s1)
AND      job = 'grafician'
AND      f.nr_publicatie IN
         (SELECT  p.nr_publicatie
          FROM    salariat s2, publicatie p
          WHERE   s2.cod_salariat = p.cod_salariat
          AND     s2.vechime = (SELECT  MAX(s3.vechime)
                                FROM    salariat s3));
```

** Inserare în multiple tabele

Începând cu *Oracle9i*, comanda *INSERT* permite inserarea de date in multiple tabele. Ea este utila in mediul *warehouse*. Inserarea se poate realiza neconditionat sau conditionat (utilizand clauza *WHEN*). O comanda *INSERT* multitabel poate contine maximum 127 clauze *WHEN*.

Inserările multiple sunt permise numai pentru tabele (nu pentru vizualizari sau vizualizari materializate). Subcererea nu poate utiliza o secventa.

1. Inserare necondiţionată utilizând clauza *ALL*

***Exemplu*:**

```
INSERT ALL
    INTO sal_history VALUES(empid, hiredate, sal)
    INTO mgr_history VALUES(empid, mgr, sal)
    SELECT  employee_id empid, hire_date hiredate,
            salary sal, manager_id mgr
    FROM    employees
    WHERE   employee_id > 177;
```

2. Inserare condiţionată utilizând clauzele *WHEN* şi *ALL*

***Exemplu*:**

```
INSERT ALL
    WHEN sal > 1000 THEN
      INTO sal_history VALUES(empid, hiredate, sal)
    WHEN mgr >177   THEN
      INTO mgr_history VALUES(empid, mgr, sal)
      SELECT  employee_id empid, hire_date hiredate,
              salary sal, manager_id mgr
      FROM    employees
      WHERE   employee_id > 177;
```

3. Inserare condiţionată utilizând clauza *FIRST*

În acest caz, *server*-ul *Oracle* evalueaza fiecare clauză *WHEN* în ordinea apariţiei în comanda *INSERT*. Opţiunea *FIRST* determină inserarea corespunzătoare primei clauze *WHEN* a cărei condiţie este evaluată *true*. Toate celelalte clauze *WHEN* sunt ignorate pentru linia respectivă. Pentru liniile care nu satisfac prima conditie *WHEN*, restul conditiilor sunt evaluate in aceeasi maniera ca pentru *INSERT* conditional. Dacă nici o condiţie din clauzele *WHEN* nu este adevărată, atunci sistemul execută clauza *INTO* corespunzătoare opţiunii *ELSE*, iar dacă aceasta nu există, nu efectuează nici o acţiune.

***Exemplu*:**

```
INSERT FIRST
    WHEN   sal > 20000         THEN
      INTO special_sal  VALUES(deptid, sal)
    WHEN hiredate LIKE('%00%') THEN
      INTO hiredate_history_oo VALUES(deptid, hiredate)
    WHEN hiredate LIKE('%99%') THEN
      INTO hiredate_history_99 VALUES(deptid, hiredate)
    ELSE
    INTO hiredate_history VALUES(deptid, hiredate)
    SELECT   department_id deptid, SUM(salary) sal,
             MAX(hire_date) hiredate
    FROM     employees
    GROUP BY department_id;
```

4. Inserare din tabele nerelaţionale (pivotare nerelaţional → relaţional)

***Exemplu*:**

Tabelul *alfa* (*emp_id*, *week_id*, *sale_lu*, *sale_ma*, *sale_mi*, *sale_jo*, *sale_vi*) provine dintr-o bază nerelaţională. Să se depună aceste date, în format relaţional, în tabelul *sales_info* (*emp_id*, *week*, *sales*).

Practic, in tabelul *sales_info* se vor insera 5 inregistrari.

```
INSERT ALL
    INTO sales_info VALUES (emp_id, week_id, sale_lu)
    INTO sales_info VALUES (emp_id, week_id, sale_ma)
    INTO sales_info VALUES (emp_id, week_id, sale_mi)
    INTO sales_info VALUES (emp_id, week_id, sale_jo)
    INTO sales_info VALUES (emp_id, week_id, sale_vi)
    SELECT  emp_id, week_id, sale_lu, sale_ma,
            sale_mi, sale_jo, sale_vi
    FROM    alfa;
```

** Utilizarea subcererilor in comenzi *LMD*

O subcerere poate fi folosită pentru a identifica tabelul şi coloanele referite de o comandă *LMD*. De exemplu, subcererile pot fi folosite in clauza *INTO* a comenzii *INSERT*.

***Exemplu*:**

```
INSERT INTO (SELECT  cod_opera, titlu, valoare
             FROM    opera
             WHERE   cod_galerie = 17)
VALUES (234, 'Flori', 1234567);
```

Comanda *DELETE*

Ştergerea unei linii dintr-un tabel (simplu, partiţionat sau tabel de bază a unei vizualizări) se realizează prin comanda *DELETE*.

DELETE
[*FROM*] *tablename / viewname* [*AS alias*]
[*WHERE condiţie*] [*clauza_returning*]

***Observaţii*:**

- Comanda *DELETE* nu şterge structura tabelului.

- Pentru a se putea executa instrucţiunea *DELETE*, utilizatorul care o lansează în execuţie trebuie să aibă acest privilegiu.
- În clauza *WHERE* pot fi folosite şi subcereri.
- Comanda nu poate fi folosită pentru ştergerea valorilor unui câmp individual. Acest lucru se poate realiza cu ajutorul comenzii *UPDATE*.
- Atenţie la ştergere, pentru a nu afecta integritatea referenţială!

***Exemplu*:**
Să se elimine cititorii care au numele 'Popa'şi cei care au restituit astăzi cel puţin o carte.

```
DELETE FROM  cititor
WHERE        nume='Popa'
OR           codec IN (SELECT  codec
                       FROM    imprumuta
                       WHERE   data_ef=SYSDATE);
```

***Exemplu*:**
Să se şteargă tehnoredactorii care colaborează la mai puţin de trei publicaţii.

```
DELETE  FROM  salariat
WHERE   job = 'tehnoredactor'
        AND COUNT(SELECT  DISTINCT c.nr_publicatie
            FROM    capitol c
            WHERE   c.cod_salariat = cod_salariat)< 3;
```

***** Exemplu*:**
Să se elimine redactorii şefi care nu au coordonat nici o publicaţie.

```
DELETE FROM redactor_sef
WHERE       cod_salariat NOT IN
               (SELECT DISTINCT  cod_salariat
                FROM             publicatie);
```

***** Exemplu*:**
Să se şteargă salariul angajatului având codul 1279.

```
UPDATE salariat
SET    salariu=null
WHERE  cod_salariat = 1279;
```

***** Exemplu*:**
Urmatoarele doua comenzi sunt echivalente.

```
DELETE FROM opera
WHERE       cod_opera = 777;

DELETE FROM (SELECT * FROM opera)
WHERE       cod_opera = 777;
```

***** Exemplu*:**

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

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

***** Exemplu*:**

Pentru fiecare autor care are mai mult de 10 creaţii expuse în muzeu, să se şteargă ultima operă creată de acesta.

```
DELETE FROM   opera o1
WHERE         cod_artist =
                 (SELECT cod_artist
                  FROM   opera o2
                  WHERE  cod_artist = o1.cod_artist
                  AND    data_crearii =
                      (SELECT   MAX(data_crearii)
                       FROM     opera
                       WHERE    cod_artist = o2.cod_artist)
                  AND    10 <
                      (SELECT   COUNT(*)
                       FROM     opera
                     WHERE    cod_artist = o2.cod_artist));
```

Comanda *UPDATE*

Pentru modificarea valorilor existente intr-un tabel sau intr-un tabel de baza a unei vizualizari se utilizeaza comanda *UPDATE*. Valorile câmpurilor care trebuie modificate pot fi furnizate explicit sau pot fi obţinute în urma unei cereri *SQL*.

UPDATE *tablename / viewname*
SET (*column1*[,*column2*[,*...*]]) = (*subquery*) / *column = expr* / (*query*)
[*WHERE* *condition*]

***Observaţii*:**

- Pentru a se putea executa instrucţiunea *UPDATE*, utilizatorul care o lansează în execuţie trebuie să aibă acest privilegiu.
- Dacă nu este specificată clauza *WHERE* se vor modifica toate liniile.
- Cererea trebuie să furnizeze un număr de valori corespunzător numărului de coloane din paranteza care precede caracterul de egalitate.

***Exemplu*:**

Preţul cărţilor scrise de Lucian Blaga să fie modificat, astfel încât să fie egal cu preţul celei mai scumpe cărţi de informatică din bibliotecă.

```
UPDATE  carte
SET     pret = (SELECT  MAX(pret)
                FROM    carte
                WHERE   coded = 'I')
WHERE   autor = 'Lucian Blaga';
```

***Exemplu*:**

Să se modifice preţul cărţilor din bibliotecă, care se găsesc într-un număr de exemplare mai mic decât media numărului de exemplare pe bibliotecă. Noua valoare a preţului să fie egală cu suma preţurilor cărţilor scrise de Zola.

```
UPDATE  carte
SET     pret = (SELECT  SUM(pret)
                FROM    carte
                WHERE   autor = 'Zola')
WHERE   nrex < (SELECT  AVG(nrex)
                FROM    carte);
```

***Exemplu*:**

Să se reducă cu 10% salariile redactorilor şefi care nu sunt asociaţi nici unei publicaţii.

```
UPDATE  salariat
SET     salariu = 0,9*salariu
WHERE   cod_salariat IN
          (SELECT  cod_salariat
           FROM    redactor_sef
           WHERE   cod_salariat NOT IN
                  (SELECT  cod_salariat
                   FROM    publicatie));
```

***Exemplu*:**

Să se mărească cu 5% salariile redactorilor şefi ce coordoneaza publicaţiile care au cel mai mare număr de *frame*-uri.

```
UPDATE  salariat
SET     salariu = 1,05*salariu
WHERE  cod_salariat IN
     (SELECT  cod_salariat
      FROM    publicatie
      WHERE   nr_publicatie IN
           (SELECT    nr_publicatie
            FROM      frame
            GROUP BY  nr_publicatie
            HAVING    COUNT(*) > ALL
                      (SELECT    COUNT(*)
                       FROM      frame
                       GROUP BY  nr_publicatie)));
```

** *Oracle9i* permite utilizarea valorii implicite *DEFAULT* in comenzile *INSERT* si *UPDATE*. Unei coloane i se atribuie valoarea implicită definită la crearea sau modificarea structurii tabelului dacă nu se precizează nici o valoare sau dacă se precizează cuvântul cheie DEFAULT în comenzile INSERT sau UPDATE. Dacă nu este definită nici o valoare implicită pentru coloana respectivă, sistemul îi atribuie valoarea *null*.

***Exemplu*:**

```
UPDATE   carte
SET      pret = DEFAULT
WHERE    codel = 77;
```

Comanda *MERGE*

Comanda *MERGE* (aparută în versiunea *Oracle9i*) permite inserarea sau actualizarea condiţionată a datelor dintr-un tabel al bazei. Comanda este utilizată frecvent în aplicaţii *data warehouse*.

În clauza *USING* este specificată sursa datelor (tabel, vizualizare, subcerere) care vor fi inserate sau reactualizate. În clauza *INTO* este specificat tabelul destinatie (eventual *alias*) in care se insereaza sau actualizeaza inregistrari. In clauza *ON* este data condiţia de *join* după care comanda *MERGE* realizează fie operaţia de inserare, fie actualizare.

Instructiunea *MERGE* realizează *UPDATE* dacă inregistrarea (linia) este deja în tabel sau realizeaza *INSERT* în caz contrar. In acest fel, se pot evita comenzi *UPDATE* multiple. Nu se poate reactualiza aceeaşi linie de mai multe ori, în aceeaşi comandă *MERGE*.

***Exemplu*:**

```
MERGE INTO copie_carte cc
    USING carte i
    ON (cc.codel = i.codel)
    WHEN MATCHED THEN
        UPDATE SET
        cc.pret = i.pret,
        cc.coded = i.coded
    WHEN NOT MATCHED THEN
        INSERT(cc.codel, cc.autor, cc.nrex)
        VALUES(i.codel, i.autor, i.nrex);
```

Comanda *EXPLAIN PLAN*

Cand se lanseaza o cerere *SQL*, sistemul verifica daca aceasta se afla deja stocata in zona de memorie partajata *SQL*. In caz contrar, sistemul verifica instructiunea sintactic si semantic, controleaza privilegiile, genereaza un plan de executie optim, îi alocă o zona partajata *SQL* in *library cache* si executa cererea. Secventa de pasi parcursa de sistem pentru a executa o instructiune constituie planul de executie al acesteia.

Comanda *EXPLAIN PLAN* afiseaza calea utilizata de optimizor la executarea unei comenzi *LMD*. Mai exact, va plasa intr-un tabel, numit *PLAN_TABLE*, cate o linie pentru fiecare etapa din planul de executie al comenzii.

Sintaxa simplificata a comenzii:

EXPLAIN PLAN [*SET STATEMENT_ID = 'text'*]
FOR instructiune;

Clauza *SET STATEMENT_ID* permite atribuirea unui identificator instructiunii al carei plan de executie este generat.

Tabelul *PLAN_TABLE* contine informatii referitoare la: ordonarea tabelelor referite in instructiune, metoda de *join* pentru tabele (daca este cazul), costul si cardinalitatea fiecarei operatii, operatiile asupra datelor (filtrari, sortari, agregari).

Dintre cele mai importante coloane ale tabelului *PLAN_TABLE* amintim:

Coloana	Explicatie
STATEMENT_ID	valoarea parametrului specificat in comanda *EXPLAIN PLAN*
TIMESTAMP	data si ora la care a fost lansata comanda *EXPLAIN PLAN*
OPERATION	numele operatiei efectuate la acest pas
OPTIONS	optiuni asupra operatiilor descrise in coloana *OPERATION*
OBJECT_OWNER	numele utilizatorului care detine schema din care face parte tabelul sau indexul
OBJECT_NAME	numele tabelului sau indexului
ID	numarul atribuit fiecarui pas din planul de executie
PARENT_ID	identificatorul urmatorului pas de executie care opereaza asupra rezultatului pasului curent
COST	costul operatiei estimat de *CBO* (cost based optimizer)
CARDINALITY	numarul de linii accesate de operatie

***Exemplu*:**

Să se determine planul de executie al instructiunii de dublare (actualizare) a valorii cartilor scrise de Cioran.

```
EXPLAIN PLAN SET STATEMENT_ID = 'actualizare 2007'
FOR UPDATE carte
    SET    valoare = valoare*2
    WHERE  autor = 'Cioran';
```

Interogarea (partiala) planului de executie:

```
SELECT OPERATION, OBJECT_NAME
FROM   PLAN_TABLE
START  WITH ID = 0 AND STATEMENT_ID = 'actualizare2007'
CONNECT BY PRIOR ID = PARENT_ID
          AND STATEMENT_ID = 'actualizare 2007';
```

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 dispoziţia unui număr mare de utilizatori o mulţime coerentă de date;
- să garanteze coerenţa datelor în cazul manipulării simultane de către diferiţi utilizatori.

Coerenţa este asigurată cu ajutorul conceptului de **tranzacţie**. Tranzacţia este unitatea logică de lucru constând din una sau mai multe instrucţiuni *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 operaţiile ce constituie tranzacţia sunt executate şi devin efective, spunem că tranzacţia este **validată**, iar modificările aduse de tranzacţie devin definitive.

Dacă dintr-un motiv sau altul (neverificarea condiţiilor, accesul imposibil) o operaţie a tranzacţiei nu a fost executată spunem că tranzacţia a fost **anulată**. Modificările aduse de toate operaţiile tranzacţiei anulate sunt şi ele anulate şi se revine la starea bazei de date de dinaintea tranzacţiei anulate.

Este posibil ca o tranzacţie să fie descompusă în **subtranzacţii**, astfel încât dacă este necesar să se anuleze doar parţial unele operaţii.

➔ Fiecare tranzacţie se poate termina:

- “normal” (*commit*);
- “anormal” (*rollback*).

Controlul tranzacţiilor constă în:

- definirea începutului şi sfârşitului unei tranzacţii,
- validarea sau anularea acesteia,
- eventuală descompunere în subtranzacţii.

Limbajul pentru controlul datelor (*LCD*) permite salvarea informaţiei, realizarea fizică a modificărilor în baza de date, rezolvarea unor probleme de concurenţă.

Limbajul conţine următoarele instrucţiuni:

- *COMMIT* - folosită pentru permanentizarea modificărilor executate asupra BD (modificările sunt înregistrate şi sunt vizibile tuturor utilizatorilor);
- *ROLLBACK* - folosită pentru refacerea stării anterioare a BD (sunt anulate toate reactualizările efectuate de la începutul tranzacţiei);
- *SAVEPOINT* - folosită în conjuncţie cu instrucţiunea *ROLLBACK*, pentru definirea unor puncte de salvare în fluxul programului.

O tranzacţie constă:

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

Tranzacţia **începe:**

- după o comandă *COMMIT*,
- după o comandă *ROLLBACK*,
- după conectarea iniţială la Oracle,
- când este executată prima instrucţiune *SQL*.

Tranzacţia se **termină:**

- dacă sistemul cade;
- dacă utilizatorul se deconectează;
- dacă se dau comenzile *COMMIT* sau *ROLLBACK* ;
- dacă se execută o comandă *LDD*.

După ce se termină o tranzacţie, prima instrucţiune *SQL* executabilă va genera automat începutul unei noi tranzacţii.

➔ Un *commit* apare automat:

- când este executată o comandă *LDD*;
- când este executată o comandă *LCD*;
- după o ieşire normală din *SQL*Plus* fără specificarea explicită a comenzilor *COMMIT* sau *ROLLBACK*.

Un *rollback* apare automat după o ieşire "anormală" din *SQL*Plus* sau o cădere sistem.

Din momentul în care s-a executat instrucţiunea ***COMMIT***, BD s-a modificat (permanent) în conformitate cu instrucţiunile *SQL* executate în cadrul tranzacţiei care tocmai s-a terminat. Din acest punct începe o nouă tranzacţie.

Dacă se foloseşte utilitarul *SQL*Plus*, există posibilitatea ca după fiecare comandă *LMD* să aibă loc o permanentizare automată a datelor (un *COMMIT* implicit). Acest lucru se poate realiza folosind comanda:

SET AUTO[*COMMIT*] {*ON* | *OFF*}

Comanda *ROLLBACK* permite restaurarea unei stări anterioare a BD.

ROLLBACK [*TO* [*SAVEPOINT*] *savepoint*];

Dacă nu se specifică nici un *savepoint*, toate modificările făcute în tranzacţia curentă sunt anulate, iar dacă se specifică un anumit *savepoint*, atunci doar modificările de la acel *savepoint* până în momentul respectiv sunt anulate. Executarea unei instrucţiuni *ROLLBACK* presupune terminarea tranzacţiei curente şi începerea unei noi tranzacţii.

Punctele de salvare pot fi considerate ca nişte etichete care referă o submulţime a schimbărilor dintr-o tranzacţie, marcând efectiv un punct de salvare pentru tranzacţia curentă. Punctele de salvare NU sunt obiecte ale schemei. Prin urmare, nu sunt referite in DD.

Server-ul *Oracle* implementează un punct de salvare implicit pe care îl mută automat după ultima comandă *LMD* executată. Dacă este creat un punct de salvare având acelaşi nume cu unul creat anterior, cel definit anterior este şters automat.

SAVEPOINT savepoint;

***Exemplu*:**

Comanda *ROLLBACK* nu va genera terminarea tranzacţiei.

```
COMMIT
INSERT …
SAVEPOINT a
UPDATE …
INSERT …
SAVEPOINT b
DELETE …
ROLLBACK TO a
```

Starea datelor înainte de *COMMIT* sau *ROLLBACK* este următoarea:

- starea anterioară a datelor poate fi recuperată;
- utilizatorul curent poate vizualiza rezultatele operaţiilor *LMD* prin interogări asupra tabelelor;
- alţi utilizatori nu pot vizualiza rezultatele comenzilor *LMD* făcute de utilizatorul curent (*read consistency*);
- înregistrările (liniile) afectate sunt blocate şi, prin urmare, alţi utilizatori nu pot face schimbări în datele acestor înregistrări.

Execuţia unei comenzi *COMMIT* implică anumite modificări.

- Toate schimbările (*INSERT, DELETE*, *UPDATE*) din baza de date făcute după anterioara comandă *COMMIT* sau *ROLLBACK* sunt definitive. Comanda se referă numai la schimbările făcute de utilizatorul care dă comanda *COMMIT*.
- Toate punctele de salvare vor fi şterse.
- Starea anterioară a datelor este pierdută definitiv.
- Toţi utilizatorii pot vizualiza rezultatele.
- Blocările asupra liniilor afectate sunt eliberate; liniile pot fi folosite de alţi utilizatori pentru a face schimbări în date.

Execuţia unei comenzi *ROLLBACK* implică anumite modificări.

- Anulează tranzacţia în curs şi toate modificările de date făcute după ultima comandă *COMMIT*.
- Sunt eliberate blocările liniilor implicate.
- Nu şterge un tabel creat prin *CREATE TABLE*. Eliminarea tabelului se poate realiza doar prin comanda *DROP TABLE*.

***Exemplu*:**

Ce efect are următoarea secvenţă de instrucţiuni?

```
(a)  SELECT        *
     FROM          salariat;

(b)  SAVEPOINT     a;

(c)  DELETE FROM   salariat;
     INSERT INTO   salariat
```

```
    VALUES (18,'Breaban','Marin',4,5000, 'tehnored');
    INSERT INTO  salariat
    VALUES (23,'Popescu','Emil',7,40000,'grafician');
    SAVEPOINT    b;

(d) INSERT INTO  salariat
    VALUES (29,'','',5,3000000,'tehnoredactor');
    SELECT       AVG(salariu)
    FROM         salariat;

(e) ROLLBACK TO  b;
    SELECT       AVG(salariu)
    FROM         salariat;

(f) ROLLBACK TO  a;
    INSERT INTO  salariat
    VALUES   (18,'Ion','Mihai',5,580,'redr_sef');
    COMMIT;
```

Consistenţa la citire

Într-un sistem *multi-user*, sistemul *Oracle* furnizează ***read consistency*** la nivel de instrucţiune *SQL*, adică o singură comandă *SQL* nu poate da rezultate care sunt contradictorii sau inconsistente. *Read consistency* asigură că fiecare utilizator "vede" datele aşa cum existau la ultimul *commit*, înainte să înceapă o operaţie *LMD*. Prin urmare, modificările efectuate asupra unei baze de date nu sunt vizibile decât după ce operaţia de actualizare a fost validată. Numai utilizatorul care a executat tranzacţia poate vedea modificările făcute de el în cursul acestei tranzacţii.

Modelul multiversiune, furnizat de *Oracle*, asigură **consistenţa la citire:**

- garantează că setul de date văzut de orice instrucţiune *SQL* este consistent şi nu se schimbă în timpul execuţiei unei instrucţiuni (*Oracle* asigură o consistenţă la citire la nivel de instrucţiune);
- operaţiile de citire (*SELECT*) nu trebuie să vadă datele care sunt în proces de schimbare;
- operaţiile de scriere (*INSERT*, *DELETE*, *UPDATE*) nu trebuie să afecteze consistenţa datelor şi să întrerupă sau să intre în conflict cu alte operaţii de scriere concurente.

Cum se implementează modelul multiversiune? Dacă asupra bazei este executată o comandă *LMD*, *server*-ul *Oracle* face o copie a datelor dinainte de modificare şi o depune în segmentul *rollback* (*undo*).

Toţi utilizatorii (cu excepţia celor care modifică datele) vor vedea datele cum sunt înainte de modificare (văd conţinutul segmentului *undo*). Dacă comanda *LMD* este *commit*, atunci schimbările din baza de date devin vizibile oricărui utilizator care foloseşte instrucţiunea *SELECT*. Când se termină tranzacţia, spaţiul ocupat în segmentul *undo* de "vechea" dată este liber pentru reutilizare. *Server*-ul Oracle asigură astfel o vizualizare consistentă a datelor în orice moment.

Blocări

Blocările sunt folosite în *ORACLE* pentru a asigura integritatea datelor, permiţând în acelaşi timp accesul concurent la date de către un număr "infinit" de utilizatori.

Din punct de vedere a resursei blocate, blocările pot fi:

- la nivel de linie (blocarea afectează un rând);
- nivel de tabel (blocarea afectează întreg tabelul).

La **nivel de rând**, blocările se pot face numai în *modul exclusiv* (*X*), adică un utilizator nu poate modifica un rând până ce tranzacţia care l-a blocat nu s-a terminat (prin permanentizare sau prin derulare înapoi).

Blocările **la nivel de tabel** pot fi făcute în mai multe feluri, în funcţie de caracterul mai mult sau mai puţin restrictiv al blocării (*RS – row share*; *RX – row exclusive*; *S – share*; *SRX – share row exclusive*; *X – exclusive*).

- Modul *X* de blocare la nivel de tabel este cel mai restrictiv. Blocarea în mod *X* este obţinută la executarea comenzii *LOCK TABLE* cu opţiunea *EXCLUSIVE*. O astfel de blocare permite altor tranzacţii doar interogarea tabelului. Tabelul nu mai poate fi blocat în acelaşi timp de nici o altă tranzacţie în nici un mod.
- Modul de blocare *RX* arată că tranzacţia care deţine blocarea a făcut modificări asupra tabelului. O blocare *RX* permite acces (*SELECT*, *INSERT*, *UPDATE*, *DELETE*) concurent la tabel şi blocarea concurentă a tabelului de către altă tranzacţie în modurile *RS* şi *RX*.
- Modul de blocare *S* (se obţine prin comanda *LOCK TABLE* cu opţiunea *SHARE*) permite altor tranzacţii doar interogarea tabelului şi blocarea sa în modurile *S* şi *RS*.

- Modul de blocare *SRX* (se obţine prin comanda *LOCK TABLE* cu opţiunea *SHARE ROW EXCLUSIVE*) permite altor tranzacţii doar interogarea tabelului şi blocarea sa în modul *RS*.
- Modul de blocare *RS* permite acces (*SELECT*, *INSERT*, *UPDATE*, *DELETE*) concurent la tabel şi blocarea concurentă a tabelului de către altă tranzacţie în orice mod, în afară de *X*. Modul de blocare *RS*, care este cel mai puţin restrictiv, arată că tranzacţia care a blocat tabelul, a blocat rânduri din tabel şi are intenţia să le modifice.

Din punct de vedere a modului de declanşare a blocării, blocările pot fi:

- implicite (blocarea este făcută automat de sistem în urma unei operaţii *INSERT*, *DELETE* sau *UPDATE* şi nu necesită o acţiune din partea utilizatorului);
- explicite (blocarea este declanşată ca urmare a comenzilor *LOCK TABLE* sau *SELECT* cu clauza *FOR UPDATE*).

Folosirea clauzei *FOR UPDATE* într-o comandă *SELECT* determină blocarea rândurilor selectate în modul *X* şi blocarea întregului tabel (sau tabelelor) pe care se face interogarea în modul *RS*. La actualizarea rândurilor (*UPDATE*) blocarea la nivel de linie se menţine în timp ce blocarea la nivel de tabel devine *RX*.

***Exemplu*:**

```
SELECT  salariu
FROM    salariat
WHERE   cod_salariat = 1234
FOR UPDATE OF salariu;

UPDATE  salariat
SET     salariu = 23456
WHERE   cod_salariat = 1234;

COMMIT;
```

La executarea primei comenzi, rândul cu *cod_salariat = 1234* este blocat în mod *X* în timp ce tabelul *salariat* este blocat în modul *RS*. La executarea celei de a doua comenzi, blocarea la nivel de linie se menţine în timp ce blocarea la nivel de tabel devine *RX*. La executarea comenzii *COMMIT*, tranzacţia este permanentizată şi toate blocările sunt eliberate.

Unul sau mai multe tabele, vizualizari, partitii sau subpartitii ale unor tabele pot fi blocate în oricare din modurile prezentate mai sus folosind comanda *LOCK TABLE*, care are sintaxa simplificata:

LOCK TABLE nume_tabel [, *nume tabel*] ...
IN mod_blocare MODE [*NOWAIT*]

Clauza *NOWAIT* determină sistemul să returneze imediat controlul către utilizatorul care încearcă să realizeze o blocare asupra unui tabel. Dacă acesta este deja blocat, atunci sistemul va returna un mesaj corespunzător. Altfel, sistemul va aştepta până când tabelul devine disponibil, îl va bloca şi apoi va returna controlul utilizatorului.

Blocarile obtinute in urma acestei comenzi sunt prioritare celor impuse automat de catre sistem. Un tabel ramane blocat pana la operatia *COMMIT* sau *ROLLBACK* asupra tranzactiei sau pana la revenirea intr-un punct intermediar (*SAVEPOINT*) definit inainte de blocarea tabelului.

O blocare impusa asupra unui tabel nu impiedica ceilalti utilizatori sa îl consulte. Blocarea unei vizualizari implica blocarea tabelelor sale de bază.

Campul *mod_blocare* poate avea valorile *ROW SHARE, ROW EXCLUSIVE, SHARE, SHARE ROW EXCLUSIVE, EXCLUSIVE*. Dacă se specifică *NOWAIT* şi rândurile selectate sunt deja blocate de altă tranzacţie, atunci utilizatorul este înştiinţat de acest lucru, returnându-i-se controlul.

Datorită accesului concurent la date este posibil ca mai mulţi utilizatori să se blocheze reciproc. Această situaţie este numită **interblocare** (*deadlock*), pentru că fiecare dintre utilizatori aşteaptă ca celălalt să elibereze resursa blocată. În cazul acesta problema nu se poate rezolva prin simpla aşteptare, una din tranzacţii trebuind să fie derulată înapoi. *Oracle* detectează automat interblocările. În acest caz, *Oracle* semnalează o eroare uneia dintre tranzacţiile implicate şi derulează înapoi ultima instrucţiune din această tranzacţie. Acest lucru rezolvă interblocarea, deşi cealaltă tranzacţie poate încă să aştepte până la deblocarea resursei pentru care aşteaptă.

Care din următoarele comenzi încheie o tranzacţie?

SELECT
ROLLBACK
UPDATE
DELETE
CREATE TABLE