# *SQL*

***S**tructured **Q**uery **L**anguage* (*SQL*) este un limbaj universal care poate fi utilizat pentru a defini, interoga, reactualiza şi gestiona baze de date relaţionale. *SQL* este accesibil utilizatorilor începători, dar în acelaşi timp poate oferi programatorilor experimentaţi facilităţi deosebite. *SQL* este un limbaj non-procedural, adică se specifică *ce* informaţie este solicitată, dar nu modul *cum* se obţine această informaţie. *SQL* poate fi utilizat autonom sau prin inserarea comenzilor sale într-un limbaj de programare. *SQL* a sistemului *Oracle* este o extensie a normei *SQL*89 şi o implementare a normei *SQL*92.

În *SQL* se disting trei familii de comenzi:

- Comenzi pentru definirea datelor, care permit descrierea (definirea) structurii obiectelor ce modelează sistemul studiat. Aceste comenzi definesc limbajul de definire a datelor (*LDD*).
- Comenzi pentru prelucrarea datelor, ce permit consultarea, reactualizarea, suprimarea sau inserarea datelor. Aceste comenzi definesc limbajul de prelucrare a datelor (*LMD*).
- Comenzi pentru controlul datelor, care permit asigurarea confidenţialităţii şi integrităţii datelor, salvarea informaţiei, realizarea fizică a modificărilor în baza de date, rezolvarea unor probleme de concurenţă. Aceste comenzi definesc limbajul de control al datelor (*LCD*).

Sistemul impune anumite **restricţii asupra identificatorilor**.

- Numele unui obiect nu poate depăşi 30 de caractere, cu excepţia numelui bazei de date care este limitat la 8 caractere şi a numelui legăturii unei baze care poate ajunge la 128 caractere.
- Nu se face distincţie între litere mici şi litere mari.
- Numele trebuie să înceapă printr-un caracter alfabetic şi nu poate fi un cuvânt cheie rezervat; poate să conţină literele mari şi mici ale alfabetului englez, cifrele 0 - 9 şi caracterele $, _, #.
- Un utilizator nu trebuie să definească două obiecte cu acelaşi nume.
- În general este bine ca numele unui obiect să fie descriptiv şi fără prescurtări excesive.

# Limbajul de definire a datelor

La nivel logic, o bază de date *Oracle* este alcătuită din scheme. O schemă este o mulţime de structuri logice de date, numite obiecte. Ea aparţine unui utilizator al bazei de date şi poartă numele său.

Specificarea bazelor de date şi a obiectelor care le compun se realizează prin intermediul limbajului de definire a datelor (*LDD*). Definirea unui obiect presupune crearea, modificarea şi suprimarea sa. Limbajul de definire a datelor cuprinde instrucţiunile *SQL* care permit realizarea acestor operaţii (*CREATE, ALTER, DROP*). Instrucţiunile *LDD* au efect imediat asupra bazei de date şi înregistrează informaţia în dicţionarul datelor. De asemenea, *LDD* contine instructiunile *RENAME, TRUNCATE* si *COMMENT*.

În cadrul unei scheme se pot defini obiecte de tip: tabel (*table*), vizualizare (*view*), vizualizare materializată (*materialized view*), secvenţă (*sequence*), index (*index*), sinonim (*synonym*), grupare (*cluster*), procedură (*procedure*) şi funcţie (*function*) stocată, declanşator (*trigger*), pachet stocat (*package*), legătură a bazei de date (*database link*), dimensiune (*dimension*) etc.

## Tipuri de date

Pentru memorarea **datelor numerice**, tipurile cele mai frecvent folosite sunt: *NUMBER, INTEGER, FLOAT, DECIMAL*.

Pentru memorarea **şirurilor de caractere, c**ele mai frecvent tipuri de date utilizate sunt: *CHAR, VARCHAR2* şi *LONG*.

Există restricţii referitoare la folosirea tipului de date *LONG*.

- Într-un tabel poate să fie o singură coloană de tip *LONG*.
- Nu pot fi comparate două şiruri de caractere de tip *LONG*.
- O coloană de tip *LONG* nu poate fi parametru într-o procedură.
- O funcţie nu poate întoarce ca rezultat o valoare de tip *LONG*.
- O coloană de tip *LONG* nu poate fi folosită în clauzele *WHERE, ORDER BY, GROUP BY, CONNECT*.
- Operatorii sau funcţiile *Oracle* nu pot fi folosiţi în *SQL* pentru a modifica coloane de tip *LONG*.
- O coloană de tip *LONG* nu poate fi indexată.

Alte tipuri de date scalare furnizate de *SQL* sunt *NCHAR* şi *NVARCHAR2*, folosite pentru reprezentarea caracterelor limbilor naţionale.

Informaţii relative la **timp sau dată calendaristică** se obţin utilizând tipul *DATE*. Pentru fiecare dată de tip *DATE* sunt depuse: secolul, anul, luna, ziua, ora, minutul, secunda. Pentru o coloană de tip *DATE* sistemul rezervă 7 *bytes*, indiferent dacă se memorează doar timpul, sau doar data calendaristică.

Formatul implicit al datei se defineşte cu ajutorul parametrului de iniţializare *NLS_DATE_FORMAT*. În general, acest parametru este setat la forma *DD-MON-YY*. Dacă nu este specificat timpul, timpul implicit este 12:00:00.

În *Oracle8*, alături de aceste tipuri scalare, au fost introduse tipurile de date *LOB* (*Large Objects*), care specifică locaţia unor obiecte de dimensiuni mari.

*Oracle9i* introduce noi tipuri de date pentru timp:

- `TIMESTAMP` (`precizie_fracţiuni_secundă`) cuprinde valori pentru anul, luna şi ziua unei date calendaristice, dar şi valori pentru oră, minut, secundă
- `INTERVAL YEAR` (`precizie_an`) `TO MONTH` stochează o perioadă de timp specificată în ani şi luni, unde *precizie_an* reprezintă numărul de cifre din câmpul *YEAR*.
- `INTERVAL DAY` (*precizie_zi*) `TO SECOND` (`prec_fracţiuni_sec`) stochează o perioadă de timp reprezentată în zile, ore, minute şi secunde.

***Exemplu*:**

Să se creeze un tabel cu trei coloane, *inceput*, *durata_1*, *durata_2*. Coloana *inceput* va cuprinde valori ce reprezintă momente de timp, inclusiv fracţiunile de secundă corespunzătoare. Valorile coloanei *durata_1* vor fi intervale de timp specificate în număr de zile, ore, minute şi secunde. Coloana *durata_3* va conţine intervale de timp precizate în număr de ani şi luni. Să se insereze o înregistrare în acest tabel.

```
CREATE TABLE timp(
      inceput   TIMESTAMP,
      durata_1  INTERVAL DAY(2) TO SECOND(3),
      durata_2  INTERVAL YEAR TO MONTH);
INSERT INTO timp
VALUES (TIMESTAMP '1997-01-31 09:26:50.124',
        INTERVAL '23 7:44:22' DAY TO SECOND,
        INTERVAL '19-02' YEAR TO MONTH);
```

*INTERVAL YEAR TO MONTH*

| | |
|---|---|
| INTERVAL '123' YEAR(3) | un interval de 123 ani |
| INTERVAL '30' MONTH(2) | un interval de 30 luni |
| INTERVAL '123' YEAR | eroare, deoarece implicit precizia este 2, iar 123 are 3 digiti |

*INTERVAL DAY TO SECOND*

| | |
|---|---|
| INTERVAL '180' DAY(3) | un interval de 180 zile |
| INTERVAL '4 5:12' DAY TO MINUTE | un interval de 4 zile, 5 ore si 12 minute |
| INTERVAL '400 5' DAY(3) TO HOUR | un interval de 400 zile si 5 ore |

Exemplu

```
CREATE TABLE  exemplu
     (durata   INTERVAL YEAR(3) TO MONTH);

INSERT  INTO   exemplu
   VALUES (INTERVAL '120' MONTH(3));

SELECT  TO_CHAR(SYSDATE+durata, 'DD-mon-YYYY')
FROM    exemplu;
```

Exemplu

```
CREATE TABLE noi_carti
     (codel  NUMBER,
      …
      start_data TIMESTAMP);

SELECT  start_data
FROM    noi_carti;
```

Pentru informatii de tip *TIMESTAMP*, numarul maxim de digiti pentru fractiuni de secunda este 9, implicit este 6.
Rezultatele cererii:

```
15-JUN-03  12.00.00.000000 AM
23-SEP-03  12.00.00.000000 AM
…
```

Pentru informatii de tip *DATE*, formatul implicit ar fi fost DD-MON-RR

| Câmp | Valori valide pentru date calendaristice | Valori valide pentru intervale |
|---|---|---|
| ***YEAR*** | De la -4712 la 9999 (cu excepţia anului 0). | Orice valoare întreagă. |
| *MONTH* | De la 01 la 12. | De la 0 la 11. |
| DAY | De la 01 la 31 (limitat de valorile câmpurilor *MONTH* şi *YEAR*, corespunzător regulilor calendarului curent). | Orice valoare întreagă. |
| *HOUR* | De la 00 la 23. | De la 0 la 23. |
| *MINUTE* | De la 00 la 59. | De la 0 la 59. |
| *SECOND* | De la 00 la 59.9(*n*), unde „9(*n*)“ este precizia fracţiunilor de secundă. | De la 0 la 59.9(*n*), unde „9(*n*)“ este precizia fracţiunilor de secundă. |
| *TIMEZONE_HOUR* | De la –12 la 13 (prevede schimbările datorate trecerilor la ora de vară sau iarnă). | Nu se aplică. |
| *TIMEZONE_MINUTE* | De la 00 la 59. | Nu se aplică. |

Sistemul *Oracle* permite construcţia de expresii folosind valori de tip dată calendaristică şi interval. Operaţiile care pot fi utilizate în aceste expresii şi tipul rezultatelor obţinute sunt următoarele:

- *Data + Interval*, *Data – Interval*, *Interval + Data*, iar rezultatul este de tip dată calendaristică;
- *Data – Data*, *Interval + Interval*, *Interval – Interval*, *Interval * Number*, *Number * Interval*, *Interval / Number*, iar rezultatul este de tip interval.

## Modele de format

Un **model de format** este un literal caracter care descrie formatul valorilor de tip *DATE* sau *NUMBER* stocate într-un şir de caractere. Atunci când se converteşte un şir de caractere într-o dată calendaristică sau într-un număr, modelul de format indică sistemului cum să interpreteze şirul respectiv. În instrucţiunile *SQL* se poate folosi un model de format ca argument al funcţiilor *TO_CHAR* şi *TO_DATE*. În felul acesta se poate specifica formatul folosit de sistemul *Oracle* pentru a returna sau a stoca o valoare în/din baza de date. Un model de format nu schimbă reprezentarea internă a valorii în baza de date.

O parte dintre elementele cel mai frecvent întâlnite ale unui format de tip numeric sunt sintetizate în tabelul următor.

| Element | Exemplu | Descriere |
|---|---|---|
| , (*virgulă*) | `9,999` | Plasează o virgulă în poziţia specificată. Într-un model de format numeric pot fi precizate mai multe virgule, dar o virgulă nu poate apărea în partea dreaptă a punctului zecimal. |
| . (*punct*) | `99.99` | Plasează un punct zecimal în poziţia specificată. Într-un format numeric, se poate specifica cel mult un punct zecimal. |
| *$* | `$9999` | Include semnul „$“ în faţa unei valori. |
| *0* | `0999`; `9990` | Plasează zerouri în faţa sau la sfârşitul valorii. |
| *9* | `9999` | Întoarce valoarea cu numărul specificat de cifre. Valoarea va avea un spaţiu, respectiv un minus în faţă dacă este pozitivă, respectiv negativă. |
| *C* | `C999` | Plasează în poziţia specificată simbolul *ISO* pentru monede (valoarea curentă a parametrului *NLS_ISO_CURRENCY*). |
| *D* | `99D99` | Plasează în poziţia specificată caracterul zecimal, care este valoarea curentă a parametrului *NLS_NUMERIC_CHARACTER*. Valoarea implicită este punctul. Se poate specifica cel mult un caracter zecimal într-un model de format numeric. |
| *EEEE* | `9.9EEEE` | Returnează o valoare folosind notaţia ştiinţifică. |
| *L* | `L999` | Întoarce în poziţia specificată simbolul monedei locale (valoarea curentă a parametrului *NLS_CURRENCY*). |
| *MI* | `9999MI` | Plasează semnul minus la sfârşitul valorilor negative şi un spaţiu la sfârşitul celor pozitive. Acest element poate fi specificat numai pe ultima poziţie a modelului de format numeric. |
| *S* | `S9999`; `9999S` | Plasează semnele plus sau minus la începutul sau la sfârşitul valorii. Acest element poate apărea doar pe prima sau ultima poziţie a modelului de format numeric. |

Modelele de format pentru date calendaristice pot fi utilizate în cadrul următoarelor funcţii:

- *TO_DATE*, pentru a converti o valoare de tip caracter, care este într-un alt format decât cel implicit, într-o valoare de tip *DATE*;
- *TO_CHAR*, pentru a converti o valoare de tip *DATE*, care este într-un alt format decât cel implicit, într-un şir de caractere.

Un model de format pentru date calendaristice este alcătuit dintr-unul sau mai multe elemente specifice. Scrierea cu litere mari sau mici a cuvintelor, abrevierilor sau a numeralelor romane este respectată în elementul de format corespunzător. De exemplu, modelul de format „*DAY*“ produce cuvinte cu majuscule, cum ar fi „*FRIDAY*“, iar „*Day*“ şi „*day*“ au ca rezultat „*Friday*“, respectiv „*friday*“.

Într-un model de format pentru date calendaristice se pot preciza semne de punctuaţie şi literale caracter, incluse între ghilimele. Toate aceste caractere apar în valoarea returnată pe locul specificat în modelul de format.

| Element | Explicaţie |
|---|---|
| *AD* sau *A.D.* | Indicatorul *AD* (*Anno Domini*) cu sau fără puncte. |
| *BC* sau B.*C.* | Indicatorul *BC* (*Before Christ*) cu sau fără puncte. |
| *D* | Numărul zilei din săptămână (1-7). Duminica este considerată prima zi a săptămânii. |
| *DAY* | Numele zilei completat cu spaţii, până la lungimea de 9 caractere. |
| *DD* | Numărul zilei din lună (1-31). |
| *DDD* | Numărul zilei din an (1-366). |
| *DY* | Numele zilei din săptămână, printr-o abreviere de 3 litere. |
| *FF* | Fracţiunile de secundă. |
| *HH* sau *HH12* | Ora din zi (1-12). |
| *HH24* | Ora din zi (0-23). |
| *MI* | Minutele din oră (0-59). |
| *MM* | Luna din an (01-12). |
| *MON* | Numele lunii, printr-o abreviere de 3 litere. |
| *MONTH* | Numele lunii completat cu spaţii, până la lungimea de 9 litere. |
| *RM* | Luna în cifre romane (I-XII). |
| *RR* | Anul cel mai apropiat de data curentă. |
| *RRRR* | Acceptă intrarea atât cu 2, cât şi cu 4 cifre. Dacă anul de intrare se dă cu 2 cifre, furnizează acelaşi lucru ca şi formatul *RR*. |
| *SS* | Secundele din minut (0-59). |
| *SSSSS* | Secundele trecute de la miezul noptii (0-86399). |
| *TZH* | Ora regiunii. |
| *TZM* | Minutul regiunii. |
| *Y,YYY* | Anul scris cu virgulă după prima cifră. |
| *YEAR* sau *SYEAR* | Anul în litere („S“ prefixează anii i.Hr. cu semnul minus). |
| *YYYY* sau *SYYYY* | Anul cu 4 cifre. |
| *YYY*, *YY* sau *Y* | Ultimele cifre ale anului. |

**Modificatorii *FM* şi *FX*** pot fi utilizaţi în modelele de format din cadrul funcţiei *TO_CHAR*, controlând completarea cu spaţii şi verificarea exactă a formatelor. Un modificator poate să apară într-un model de format de mai multe ori. În acest caz, efectele sale sunt active pentru porţiunea din model care începe la prima apariţie şi apoi dezactivate pentru porţiunea din model care urmează celei de-a doua apariţii ş.a.m.d.

Modificatorul *FM* (*Fill Mode*) suprimă completarea cu spaţii în valoarea returnată de funcţia *TO_CHAR*, iar *FX* (*Format eXact*) impune corespondenţa exactă dintre argumentul de tip caracter şi modelul de format precizat pentru data calendaristică respectivă.

**Valoarea *null***, reprezentând lipsa datelor, nu este egală sau diferită de nici o altă valoare, inclusiv *null*.

Totuşi, există două situaţii în care sistemul *Oracle* consideră două valori *null* ca fiind egale: la evaluarea funcţiei *DECODE* şi dacă valorile *null* apar în chei compuse. Două chei compuse care conţin valori *null* sunt considerate identice dacă toate componentele diferite de *null* sunt egale.

## Pseudocoloane

O **pseudocoloană** se comportă ca şi o coloană a unui tabel, dar nu este stocată efectiv într-un tabel. Se pot face interogări asupra pseudocoloanelor, dar nu se pot insera, actualiza sau şterge valorile acestora.

- *LEVEL* returnează nivelul liniilor rezultat ale unei cereri ierarhice.
- *CURRVAL* şi *NEXTVAL* sunt pseudocoloane utile în lucrul cu secvenţe şi sunt tratate în secţiunea corespunzătoare acestora.
- *ROWID* returnează adresa unei linii din baza de date, furnizând modul cel mai rapid de a accesa linia respectivă. În sistemul *Oracle,* valorile acestei pseudocoloane conţin următoarele informaţii necesare pentru a localiza o linie: numărul obiectului, blocul de date, fişierul de date, linia în cadrul blocului de date. Valorile pseudocoloanei *ROWID* sunt de tipul *ROWID* sau *UROWID*.
- *ROWNUM* returnează numărul de ordine al liniilor rezultate în urma execuţiei unei cereri. Pseudocoloana poate fi utilizată pentru a limita numărul de linii returnate. Dacă este folosită clauza *ORDER BY* într-o subcerere, iar condiţia în care apare *ROWNUM* este plasată în cererea de nivel superior, atunci condiţia va fi aplicată după ordonarea liniilor.

***Exemplu*:**

Să se afişeze informaţii despre operele de artă având cele mai mici 10 coduri.

```
SELECT *
FROM   (SELECT * FROM opera ORDER BY cod_opera)
WHERE  ROWNUM < 11;
```

# Tabele

## Crearea unui tabel

Crearea unui tabel constă din generarea structurii sale, adică atribuirea unui nume tabelului şi definirea caracteristicelor sale (se definesc coloanele, se definesc constrângerile de integritate, se specifică parametrii de stocare etc.).

În *Oracle9i* tabelele pot fi create în orice moment, chiar şi în timpul utilizării bazei. Structura unui tabel poate fi modificată *online*. Nu este necesar să se specifice dimensiunea acestuia. Totuşi, din considerente administrative, este important să se cunoască estimativ cât spaţiu va utiliza tabelul.

Comanda *CREATE TABLE* permite crearea unui tabel relaţional sau a unui tabel obiect. Tabelul relaţional reprezintă structura fundamentală pentru stocarea datelor utilizatorului. Un tabel obiect utilizează un tip obiect pentru definiţia unei singure coloane şi este folosit pentru a stoca instanţele unui obiect particular.

Pentru a crea un tabel, utilizatorul trebuie să aibă acest privilegiu şi să dispună de spaţiul de memorie în care să creeze obiectul. La nivelul schemei sale, un utilizator are toate privilegiile.

*CREATE TABLE* [<*nume_schema*>.] <*nume_tabel*> (
<*nume_coloana_1*> <*tip_date*> [*DEFAULT* <*expresie*>],
…
<*nume_coloana_n*> <*tip_date*> [*DEFAULT* <*expresie*>])
[*CLUSTER* <*nume_cluster*> (<*coloana_1*>,..., <*coloana_m*>)]
[*ENABLE* | *DISABLE* <*clause*>];

Comanda poate conţine opţional clauza *TABLESPACE*, care specifică spaţiul tabel în care va fi stocat tabelul. De asemenea, poate conţine opţional clauza *STORAGE* care este folosită pentru setarea parametrilor de stocare prin intermediul cărora se specifică mărimea şi modul de alocare a extinderilor segmentului tabel. La crearea unui tabel nu este nevoie să se specifice dimensiunea maximă a acestuia, ea fiind determinată până la urmă de marimea spaţiului alocat spaţiului tabel în care este creat tabelul.

Structura unui tabel poate fi creată în următoarele patru moduri:

- fără a indica cheile;
- indicând cheile la nivel de coloană;
- indicând cheile la nivel de tabel;
- prin copiere din alt tabel.

**1.** Crearea structurii unui tabel fără a indica cheile:

```
CREATE TABLE     carte
   (codel        CHAR(5),
    titlu        VARCHAR2(30),
    autor        VARCHAR2(30),
    pret         NUMBER(8,2),
    nrex         NUMBER(3),
    coded        CHAR(5));
```

**2.** Crearea structurii unui tabel indicând cheile la nivel coloană:

```
CREATE TABLE     carte
   (codel        CHAR(5) PRIMARY KEY,
    titlu        VARCHAR2(30),
    autor        VARCHAR2(30),
    pret         NUMBER(8,2),
    nrex         NUMBER(3),
    coded        CHAR(5) NOT NULL
              REFERENCES domeniu(coded));
```

Constrângerea de **cheie primară** sau **externă** ce presupune?

```
CREATE TABLE carte
   (codel    CHAR(5) PRIMARY KEY,
    titlu    VARCHAR2(30),
    autor    VARCHAR2(30),
    pret     NUMBER(8,2),
    nrex     NUMBER(3),
    coded    CHAR(5) NOT NULL
             REFERENCES domeniu(coded)
             ON DELETE CASCADE);
```

Opţiunea *ON DELETE CASCADE* specifică că suprimarea oricărui domeniu de carte din tabelul *domeniu* este autorizată şi implică suprimarea automată a tuturor cărţilor din domeniul respectiv care se găsesc în tabelul *carte*.

**3.** Crearea structurii unui tabel indicând cheile la nivel de tabel:

```
CREATE TABLE carte
   (codel    CHAR(5),
    titlu    VARCHAR2(30),
    autor    VARCHAR2(30),
    pret     NUMBER(8,2),
    nrex     NUMBER(3),
    coded    CHAR(5) NOT NULL,
    PRIMARY KEY (codel),
     FOREIGN KEY (coded)
                  REFERENCES domeniu (coded));
```

Dacă cheia primară are mai mult de o coloană atunci cheile trebuie indicate la nivel de tabel.

```
CREATE TABLE imprumuta
   (codel    CHAR(5),
    codec    CHAR(5),
    dataim   DATE DEFAULT SYSDATE,
    datares  DATE,
    dataef   DATE,
    PRIMARY KEY (codel, codec, dataim),
    FOREIGN KEY (codel)
                 REFERENCES carte(codel),
    FOREIGN KEY (codec)
                 REFERENCES cititor(codec));
```

**4.** Crearea structurii unui tabel prin copiere din alt tabel:

```
CREATE TABLE carte_info
   AS SELECT codel, titlu, autor
   FROM      carte
   WHERE     coded = 'I';
```

Constrângerile din primul tabel nu se păstrează şi pentru al doilea tabel. Comanda creează un tabel, dar şi inserează date în tabel.

## Constrângeri

Constrângerea este un mecanism care asigură că valorile unei coloane sau a unei mulţimi de coloane satisfac o condiţie declarată. Unei constrâgeri i se poate da un nume unic. Dacă nu se specifică un nume explicit atunci sistemul automat îi atribuie un nume de forma *SYS_Cn*, unde *n* reprezintă numărul constrângerii. Constrângerile pot fi şterse, pot fi adăugate, pot fi activate sau dezactivate, dar nu pot fi modificate.

***Exemplu*:**

Să se definească o constrângere la **nivel de coloană** prin care să se specifice cheia primară şi cheia externă.

```
CREATE TABLE carte
         (codel       CHAR(5)
             CONSTRAINT cp_carte PRIMARY KEY,
          titlu       VARCHAR2(30),…
          coded       CHAR(5)
             CONSTRAINT nn_coded NOT NULL
             CONSTRAINT ce_coded
                    REFERENCES domeniu(coded));
```

***Exemplu*:**

Să se definească o constrângere la **nivel de tabel** prin care să se specifice cheia primară şi cheia externă.

```
CREATE TABLE carte
         (codel       CHAR(5),
          titlu       VARCHAR2(30),…
          coded       CHAR(5) NOT NULL,
          CONSTRAINT cp_carte PRIMARY KEY (codel),
          CONSTRAINT ce_coded
          FOREIGN KEY (coded)
                 REFERENCES domeniu(coded));
```

***Observaţii***

- Liniile ce nu respectă constângerea sunt depuse automat într-un tabel special.
- Constrângerile previn ştergerea unui tabel dacă există dependenţe.
- Constrângerile pot fi create o dată cu tabelul sau după ce acesta a fost creat.
- Constrângerile pot fi activate sau dezactivate în funcţie de necesităţi.

**Constrângeri declarative:** constrângeri de domeniu, constrângerea de integritate a entităţii, constrângerea de integritate referenţială.

**Constrângerile de domeniu** definesc valori luate de un atribut (*DEFAULT*, *CHECK*, *UNIQUE*, *NOT NULL*).

- constrângerea (coloană) ***DEFAULT*** ;
- constrângerea (coloană sau tabel) ***CHECK*** ; constrângerea *CHECK* la nivel de tabel poate compara coloane între ele, poate face referinţă la una sau mai multe coloane, dar nu poate conţine subcereri. Constrângerea la nivel de coloană nu poate referi alte coloane ale aceluiaşi tabel.

```
CREATE TABLE carte
   (codel    CHAR(5),…
    pret     NUMBER(8,2)
             CONSTRAINT alfa
             CHECK (pret < nrex),…);
```

La execuţia acestei comenzi apare mesajul: *ORA – 02438*: *Column check constraint cannot reference other columns.*

Dacă dupa *NUMBER*(*8, 2*) se adaugă o virgulă, atunci constrângerea va fi la nivel de tabel, iar în aceste caz este permisă referirea altei coloane.

- constrângerea (coloană sau tabel) ***UNIQUE*** ;
- constrângerea declarativă ***NOT NULL*** poate fi doar la nivel coloană.

**Constrângerea de integritate a entităţii** precizează cheia primară a unui tabel. Când se creează cheia primară se generează automat un index unic. Valorile cheii primare sunt distincte şi diferite de valoarea *null*.

**Constrângerea de integritate referenţială** asigură coerenţa între cheile primare şi cheile externe corespunzătoare. Când este definită o cheie externă sistemul *Oracle* verifică:

- dacă a fost definită o cheie primară pentru tabelul referit de cheia externă;
- dacă numărul coloanelor ce compun cheia externă corespunde numărului de coloane a cheii primare;
- dacă tipul şi lungimea fiecărei coloane a cheii externe corespunde cu tipul şi lungimea fiecărei coloane a cheii primare.

Această constrângere poate fi definită la nivel de coloană sau tabel şi asigură coerenţa între cheile primare şi cheile externe corespunzătoare. Constrângerea *FOREIGN KEY* desemnează o coloană sau o combinaţie de coloane drept cheie externă şi stabileşte relaţia cu o cheie primară sau o cheie unică din acelaşi tabel sau din altul. O cheie externă compusă poate fi definită doar la nivel de tabel.

Cheia externă se defineşte în tabelul „copil“, iar tabelul care conţine coloana referită reprezintă tabelul „părinte“. Valoarea unei chei externe trebuie să fie egală cu o valoare existentă în tabelul „părinte“ sau să fie *null*. Cheile externe se bazează pe valorile datelor şi sunt *pointer*-i logici.

Cheile externe se definesc folosind următoarele cuvinte cheie:

- *FOREIGN KEY* este utilizat într-o constrângere la nivel de tabel pentru a defini coloana din tabelul „copil“;
- *REFERENCES* identifică tabelul „părinte“ şi coloana corespunzătoare din acest tabel;
- *ON DELETE CASCADE* determină ca, odată cu ştergerea unei linii din tabelul „părinte“, să fie şterse şi liniile dependente din tabelul „copil“;
- *ON DELETE SET NULL* determină modificarea automată a valorilor cheii externe la valoarea *null*, atunci când se şterge valoarea „părinte“.

Definiţiile şi numele constrângerilor definite se pot fi consulta prin interogarea vizualizărilor *USER_CONSTRAINTS* şi *ALL_CONSTRAINTS* din dicţionarul datelor.

În versiunea *Oracle8* există posibilitatea ca o constrângere să fie **amânată** (*DEFERRABLE*). În acest caz, mai multe comenzi *SQL* pot fi executate fără a se verifica restricţia, aceasta fiind verificată numai la sfârşitul tranzacţiei, atunci când este executată comanda *COMMIT*. Dacă vreuna din comenzile tranzacţiei încalcă restricţia, atunci întreaga tranzacţie este derulată înapoi şi este returnată o eroare. Opţiunea implicită este *NOT DEFERRABLE*.

O noutate introdusă în *Oracle8* este posibilitatea de a **partiţiona tabele**, adică de a împărţi tabelul în mai multe părţi independente, fiecare cu parametri de stocare diferiţi şi cu posibilitatea ca părţi diferite ale tabelului să se găsească pe spaţii tabel diferite. Fiecare partiţie a tabelului va conţine înregistrări care au valoarea cheii într-un interval specificat. Partiţionarea este transparentă pentru utilizatori şi aplicaţii. Dacă o parte a tabelului este inaccesibilă, celelalte părţi pot fi disponibile pentru reactualizare. De asemenea, se poate bloca accesul la o parte a tabelului în timp ce restul înregistrărilor sunt disponibile.

***Exemplu*:**

```
CREATE TABLE carte (   )
   PARTITIONED BY RANGE (nrex)
    ((PARTITION mic VALUES LESS THAN(2)
       TABLESPACE…
       STORAGE …),
      PARTITION mediu VALUES LESS THAN (10)
       TABLESPACE…
       STORAGE …),
      PARTITION mare VALUES LESS THAN (MAXVALUE)
       TABLESPACE…
       STORAGE …));
```

***Exemplu*:**

Să se definească tabelul *opera_part_dom*, creând partiţii corespunzătoare operelor de artă achiziţionate înainte de 1980, între 1980 şi 1989, între 1990 şi 1999, respectiv după anul 2000.

```
CREATE TABLE opera_part_dom (
   cod_opera       NUMBER,
   tip             VARCHAR2(15),
```

```
   titlu            VARCHAR2(100),
   cod_artist       NUMBER,
   data_crearii     DATE,
   data_achizitiei DATE,
   valoare          NUMBER,
   cod_galerie      NUMBER,
   cod_sala         NUMBER,
   material         VARCHAR2(2000),
   stil             VARCHAR2(100),
   dim1             NUMBER,
   dim2             NUMBER,
   stare            VARCHAR2(50))
PARTITION BY RANGE (data_achizitiei)
  (PARTITION opera_inainte_1980 VALUES LESS THAN
                    (TO_DATE('01-JAN-1980','DD-MON-YYYY')),
   PARTITION opera_inainte_1990 VALUES LESS THAN
                    (TO_DATE('01-JAN-1990','DD-MON-YYYY')),
   PARTITION opera_inainte_2000 VALUES LESS THAN
                    (TO_DATE('01-JAN-2000','DD-MON-YYYY')),
   PARTITION opera_dupa_2000    VALUES LESS THAN (MAXVALUE));
```

## Modificarea structurii unui tabel

Comanda care realizează modificarea structurii tabelului (la nivel de coloană sau la nivel de tabel), dar nu modificarea conţinutului acestuia, este *ALTER TABLE*.

*ALTER TABLE* realizează modificarea structurii unui tabel nepartiţionat sau partiţionat, a unei partiţii sau subpartiţii dintr-un tabel. Pentru tabele obiect sau tabele relaţionale conţinând coloane obiect, instrucţiunea poate fi utilizată pentru a converti tabelul la ultima definiţie a tipului referit, după ce acesta a fost modificat. Comanda nu schimbă conţinutul tabelului.

Comanda *ALTER TABLE* permite:

- adăugarea (*ADD*) de coloane, chei (primare sau externe), constrângeri într-un tabel existent;
- modificarea (*MODIFY*) coloanelor unui tabel;
- specificarea unei valori implicite pentru o coloană existentă;
- activarea şi dezactivarea (*ENABLE*, *DISABLE*) unor constrângeri;
- suprimarea unei coloane;
- suprimarea (*DROP*) cheii primare, a cheii externe sau a unor constrângeri.

Comanda *ALTER TABLE* are următoarea sintaxă simplificată:

*ALTER TABLE* [<*nume_schema*>.] <*nume_tabel*>
[*ADD* (<*nume_coloana*> <*tip_date*>, <*constrângere*>) |
*MODIFY* (<*nume_coloana_1*>,…, <*nume_coloana_n*>) |
*DROP* <*clauza_drop*>,]
[*ENABLE* | *DISABLE* <*clause*>];

**1.** Pentru a adăuga o coloană, o cheie primară, o cheie externă sau o constrângere unui tabel este folosită următoarea formă:

```
ALTER TABLE    nume_tabel
ADD            (nume_coloana    constrangere,…
                nume_coloana    constrangere);
```

**2.** Pentru a modifica una sau mai multe coloane existente:

```
ALTER TABLE    nume_tabel
MODIFY         (nume_coloana    constrangere,…
                nume_coloana    constrangere);
```

**3.** Pentru a suprima cheia primară sau alte constrângeri sunt utilizate formele:

```
ALTER TABLE    nume_tabel
DROP   PRIMARY KEY;
ALTER TABLE    nume_tabel
DROP   CONSTRAINT nume_constrangere;
```

**4.** Pentru a activa (*ENABLE*) sau dezactiva (*DISABLE*) constrângeri:

```
ALTER TABLE    nume_tabel
ENABLE     nume_constrangere;
```

**Observaţii**

- Nu se poate specifica poziţia unei coloane noi în structura tabelului. O coloană nouă devine automat ultima în cadrul structurii tabelului.
- Modificarea unei coloane presupune schimbarea tipului de date, a dimensiunii sau a valorii implicite a acesteia. O schimbare a valorii implicite afectează numai inserările care succed modificării.
- Dimensiunea unei coloane numerice sau de tip caracter poate fi mărită, dar nu poate fi micşorată decât dacă acea coloană conţine numai valori *null* sau dacă tabelul nu conţine nici o linie.
- Tipul de date al unei coloane poate fi modificat doar dacă valorile coloanei respective sunt *null*.

- Definirea cheii primare sau a cheii externe după crearea tabelului.

```
CREATE TABLE   carte
  (CODEL   char(5),
   …);
ALTER TABLE    carte
ADD CONSTRAINT cheie_prim PRIMARY KEY (codel);
```

- Suprimarea cheii primare.

```
ALTER TABLE  carte
DROP PRIMARY KEY;
```

- Dacă există o CE care referă o CP şi dacă se încearcă ştergerea cheii primare, această ştergere nu se poate realiza (tabelele sunt legate prin declaraţia de cheie externă). Ştergerea este totuşi permisă dacă în comanda *ALTER* apare opţiunea *CASCADE*, care determină şi ştergerea cheilor externe ce referă cheia primară.

```
ALTER TABLE  domeniu
DROP PRIMARY KEY CASCADE;
```

- Suprimarea cheii externe.

```
ALTER TABLE        carte
ADD CONSTRAINT beta
      FOREIGN KEY (coded) REFERENCES domeniu;
ALTER TABLE        carte
DROP CONSTRAINT    beta;
```

- Schimbarea cheii primare. Este destul de complicat procesul schimbării cheii primare fără a afecta modul de proiectare a bazei de date. Schimbarea se face în două etape: se şterge cheia primară şi apoi se recreează.

```
ALTER TABLE    carte
ADD   (PRIMARY KEY(codel));
ALTER TABLE    carte
DROP PRIMARY KEY;
ALTER TABLE    carte
ADD   PRIMARY KEY(titlu, autor));
```

- Adăugarea unei coloane. Această coloană iniţial va fi *null* (pentru toate liniile). Nu se poate specifica unde să apară coloana, ea devenind ultima coloană a tabelului.

```
ALTER TABLE    carte
ADD            (rezumat LONG);
```

- Pentru suprimarea unei coloane a tabelului este utilizată următoarea formă:

  ***ALTER TABLE*** *nume_tabel*
  ***DROP*** (*nume_coloană*) [***CASCADE CONSTRAINTS***];

Opţiunea *DROP* a comenzii *ALTER TABLE* a fost introdusă în versiunea *Oracle8i*. Coloana care este suprimată poate să conţină date. O comandă *ALTER TABLE* permite ştergerea unei singure coloane. După o astfel de operaţie, în tabel trebuie să rămână cel puţin o coloană. Odată suprimată, o coloană nu poate fi recuperată.

Clauza *CASCADE CONSTRAINTS* permite suprimarea tuturor constrângerilor de integritate referenţială care se referă la cheile primare sau unice definite asupra coloanelor şterse. De asemenea, prin această clauză se suprimă şi constrângerile multicoloană definite pe baza coloanelor şterse.

- Constrângerile pot fi adăugate (*ADD CONSTRAINT*), şterse (*DROP CONSTRAINT*), activate (*ENABLE*) sau dezactivate (*DISABLE*), dar nu pot fi modificate.

```
ALTER TABLE          cititor
ADD CONSTRAINT       cp_cititor
                     PRIMARY KEY (codec)
                     DISABLE;
ALTER TABLE          cititor
ENABLE CONSTRAINT    cp_cititor;
```

Prima comandă adaugă o constrângere, dar nu-i dă viaţă. Constrângerea există, dar *server*-ul nu o verifică. Când se activează o constrângere, sistemul controlează toate liniile tabelului şi inserează într-un tabel special toate liniile care nu verifică constrângerea. Tabelul are următoarea structură:

```
(ROW_ID         ROWID
(OWNER          VARCHAR2(30),
(TABLE_NAME     VARCHAR2(30),
(CONSTRAINT     VARCHAR2(30))
```

- Din punct de vedere fizic, comanda *ALTER TABLE* permite schimbarea parametrilor *PCTFREE* şi *PCTUSED* şi a parametrilor din clauza *STORAGE*.
- Comanda permite alocarea (*ALLOCATE EXTENT*) şi dealocarea (*DEALLOCATE UNUSED*) manuală a spaţiului utilizat de către un tabel. Alocarea se face prin adăugarea de noi extinderi, iar dealocarea reprezintă eliberarea spaţiului nefolosit de tabel (care nu a fost folosit niciodată sau a devenit liber datorită ştergerii unor linii).

- Alte opţiuni ale comenzii *ALTER TABLE*, care au apărut începând cu versiunea *Oracle8i*, sunt *SET UNUSED* şi *DROP UNUSED COLUMNS*:

  ***ALTER TABLE*** *nume_tabel*
  ***SET UNUSED*** [ ( ] *nume_coloană* [ ) ];

  ***ALTER TABLE*** *nume_tabel*
  ***DROP UNUSED COLUMNS***;

Opţiunea *SET UNUSED* permite marcarea uneia sau mai multor coloane ca fiind nefolosite, cu scopul de a fi şterse atunci când necesităţile sistemului impun acest lucru. Coloanele nefolosite sunt tratate ca şi cum ar fi fost suprimate, deşi datele acestora rămân în liniile tabelului. După ce o coloană a fost marcată *UNUSED*, utilizatorul nu mai are acces la aceasta prin intermediul cererilor. În plus, numele şi tipurile de date ale coloanelor nefolosite nu vor fi afişate cu comanda *DESCRIBE* din *SQL*Plus*. Utilizatorul poate să adauge tabelului o nouă coloană având acelaşi nume cu cel al unei coloane nefolosite.

*DROP UNUSED COLUMNS* şterge din tabel toate coloanele marcate ca fiind nefolosite. Acest lucru se poate utiliza pentru eliberarea spaţiului de pe disc corespunzător coloanelor nefolosite din tabel. Dacă tabelul nu conţine nici o coloană nefolosită, instrucţiunea nu întoarce o eroare şi nu are nici un efect.

Atunci când se elimină o coloană dintr-un tabel folosind opţiunea *DROP* a comenzii *ALTER TABLE*, vor fi suprimate şi coloanele din tabel care sunt marcate cu opţiunea *SET UNUSED*.

## Suprimarea unui tabel

Pentru ştergerea unui tabel este utilizată comanda *DROP TABLE*:

***DROP TABLE*** [*nume_schema*.]*nume_tabel*
[***CASCADE CONSTRAINTS***];

Clauza *CASCADE CONSTRAINTS* permite suprimarea tuturor constrângerilor de integritate referenţială corespunzătoare cheilor primare şi unice din tabelul supus ştergerii. Dacă se omite această clauză şi există constrângeri de integritate referenţială, sistemul returnează o eroare şi nu suprimă tabelul.

Suprimarea unui tabel presupune:
- suprimarea definiţiei sale în dicţionarul datelor;
- suprimarea indecşilor asociaţi;
- suprimarea privilegiilor conferite în legătură cu tabelul;
- recuperarea spaţiului ocupat de tabel;

– permanentizarea tranzactiilor in asteptare;

– invalidarea (dar nu suprimarea) funcţiilor, procedurilor, vizualizărilor, secventelor, sinonimelor referitoare la tabel.

Odată executată, instrucţiunea *DROP TABLE* este ireversibilă. Ca şi în cazul celorlalte instrucţiuni ale limbajului de definire a datelor, această comandă nu poate fi anulată (*ROLLBACK*).

*Oracle 10g* introduce o noua maniera pentru suprimarea unui tabel. Cand se sterge un tabel, baza de date nu elibereaza imediat spatiul asociat tabelului. Ea redenumeste tabelul si acesta este plasat intr-un *recycle bin* de unde poate fi eventual recuperat ulterior prin comanda *FLASBACK TABLE*.

***Exemplu*:**

```
DROP TABLE exemplu;
SELECT …
INSERT…
FLASHBACK TABLE exemplu TO BEFORE DROP;
```

In *Oracle 10g* stergerea unui tabel se poate face simultan cu eliberarea spatiului asociat tabelului, daca este utilizata clauza *PURGE* in comanda *DROP TABLE*. Nu este posibil un *rollback* pe o comanda *DROP TABLE* cu clauza *PURGE*.

Pentru ştergerea întregului conţinut al unui tabel şi eliberarea spaţiului de memorie ocupat de acesta, sistemul *Oracle* oferă instrucţiunea:

***TRUNCATE TABLE*** *nume_tabel*;

Fiind o instrucţiune *LDD*, aceasta nu poate fi anulată ulterior (printr-o operaţie *ROLLBACK*). Ea reprezintă o alternativă a comenzii *DELETE* din limbajul de prelucrare a datelor. De remarcat că instrucţiunea *DELETE* nu eliberează spaţiul de memorie. Comanda *TRUNCATE* este mai rapidă deoarece nu generează informaţie *ROLLBACK* şi nu activează declanşatorii asociaţi operaţiei de ştergere. Dacă tabelul este „părintele“ unei constrângeri de integritate referenţială, el nu poate fi trunchiat. Pentru a putea fi aplicată instrucţiunea *TRUNCATE*, constrângerea trebuie să fie mai întâi dezactivată.

In *DD*, informaţiile despre tabele se găsesc în vizualizarea *USER_TABLES*. Dintre cele mai importante coloane ale acesteia, se remarca:

| | |
|---|---|
| *TABLE_NAME* | Numele tabelului |
| *TABLESPACE_NAME* | Spaţiul tabel în care se află tabelul |
| *CLUSTER_NAME* | Numele *cluster*-ului din care face parte tabelul |

| | |
|---|---|
| *PCT_FREE* | Procentul de spaţiu păstrat liber în interiorul fiecărui bloc |
| *PCT_USED* | Procentul de spaţiu ce poate fi utilizat în fiecare bloc |
| *INI_TRANS* | Numărul iniţial de tranzacţii concurente în interiorul unui bloc |
| *NITIAL_EXTENT* | Dimensiunea spaţiului alocat pentru prima extensie |
| *NEXT_EXTENT* | Dimensiunea spaţiului alocat pentru următoarea extensie |
| *MIN_EXTENTS* | Numărul minim de extensii ce se alocă la crearea unui tabel |
| *MAX_EXTENTS* | Numărul maxim de extensii ce se alocă la crearea unui tabel |
| *PCT_INCREASE* | Procentul cu care creşte dimensiunea unei extensii |
| *BACKED_UP* | *Y* sau *N*, după cum tabelului i-a fost făcută o copie de siguranţă de la ultima modificare |
| *NUM_ROWS* | Numărul de înregistrări din tabel |
| *BLOCKS* | Numărul de blocuri utilizate de tabel |
| *EMPTY_BLOCKS* | Numărul de blocuri ce nu conţin date |
| *AVG_SPACE* | Spaţiul mediu liber din tabel |
| *AVG_ROW_LEN* | Lungimea medie, în octeţi, a unei linii |
| *TABLE_LOCK* | *ENABLED* (activat) sau *DISABLED* (dezactivat): este activată sau nu blocarea tabelului |
| *PARTITIONED* | *YES* sau *NO*, indică dacă tabelul este partiţionat (sau nu) |
| *TEMPORARY* | *Y* sau *N*, indică dacă tabelul este temporar (sau nu) |
| *NESTED* | *YES* sau *NO*, indică dacă tabelul este imbricat (sau nu) |

***Exemplu*:**

```
DESCRIBE USER_TABLES
SELECT   TABLE_NAME, NUM_ROWS, NESTED
FROM     USER_TABLES;
```

***Exemplu*:**

```
SELECT 'DROP TABLE' || OBJECT_NAME || ';'
FROM USER_OBJECTS
WHERE OBJECT_TYPE = 'TABLE';
```

** Versiunea *Oracle9i* introduce următoarele funcţionalităţi:

- crearea de tabele externe (ale căror date se află în afara bazei de date);
- crearea de fişiere gestionate de sistemul *Oracle* (*Oracle Managed Files*);
- partiţionarea de tip listă sau domeniu-listă a unui tabel.

Sintaxa comenzii *CREATE TABLE* este următoarea:

***CREATE*** [***GLOBAL TEMPORARY***] ***TABLE*** [*schema.*]*nume_tabel*
{*nume_coloană_1 tip_date* [***DEFAULT*** *expresie*]
[*constr_coloană_ref*] [*constr_coloană* [*constr_coloană*] ...]
| {*constr_tabel_sau_view* | *constr_tabel_ref*} }

[, {*nume_coloană_2…*} ]
[***ON COMMIT*** {***DELETE*** | ***PRESERVE***} ***ROWS***]
[*proprietăţi_fizice*] [*proprietăţi_tabel*];

## Tabele temporare

Opţiunea *GLOBAL TEMPORARY* permite crearea unui tabel temporar, al cărui scop este de a stoca date specifice unei sesiuni. Aceste date sunt stocate în tabel numai pe durata unei tranzacţii sau a întregii sesiuni.

Definiţia unui tabel temporar este accesibilă tuturor sesiunilor, dar informaţiile dintr-un astfel de tabel sunt vizibile numai sesiunii care inserează linii în acesta.

Precizarea opţiunii *ON COMMIT* determină dacă datele din tabelul temporar persistă pe durata unei tranzacţii sau a unei sesiuni.

Clauza *DELETE ROWS* se utilizează pentru definirea unui tabel temporar specific unei tranzacţii, caz în care sistemul trunchiază tabelul, ştergând toate liniile acestuia după fiecare operaţie de permanentizare (*COMMIT*).

Clauza *PRESERVE ROWS* se specifică pentru a defini un tabel temporar specific unei sesiuni, caz în care sistemul trunchiază tabelul la terminarea sesiunii.

***Exemplu*:**

Să se creeze un tabel temporar *achizitii_azi*. Sesiunea fiecărui angajat care se ocupă de achiziţii va permite stocarea în acest tabel a achiziţiilor sale de la data curentă. La sfârşitul sesiunii, aceste date vor fi şterse.

```
CREATE GLOBAL TEMPORARY TABLE achizitii_azi
   ON COMMIT PRESERVE ROWS
   AS SELECT  *
      FROM    opera
      WHERE   data_achizitiei = SYSDATE;
```

## Tabele externe

În cadrul clauzei *proprietăţi_fizice*, cuvântul cheie *EXTERNAL* şi opţiunea *clauza_tabel_extern* permit crearea unui tabel extern. Un astfel de tabel este *read-only*. Metadatele corespunzătoare lui sunt reţinute în dicţionarul datelor, iar datele sale sunt stocate în afara bazei, în sistemul de fişiere al *server*-ului.

Tabelele externe permit interogarea datelor fără încărcarea lor prealabilă în bază. Nici o operatie *LMD* nu este permisa asupra acestora, nici un index nu poate fi creat relativ la aceste tabele. Sunt permise doar interogari utilizand *SQL*, *PL/SQL* sau *Java*.

## Tabele organizate pe bază de index

Opţiunile *INDEX* şi *clauza_tabel_org_index*, prezente în cadrul clauzei *proprietăţi_fizice*, permit crearea unui tabel organizat pe bază de index. Sistemul *Oracle* va menţine liniile acestui tabel (atât valorile coloanelor care compun cheia primară, cât şi valorile celorlalte coloane) într-un index construit pe baza cheii primare.

## Paralelism

Prin clauza *AS* se poate specifica o subcerere care determină crearea şi popularea unui tabel. Coloanele tabelului vor fi cele precizate în lista *SELECT* a subcererii. Liniile returnate de subcerere vor fi inserate în tabel la momentul creării. Crearea unui tabel prin copierea structurii şi conţinutului dintr-un tabel sursă nu păstrează constrângerile tabelului sursă.

***Exemplu*:**

Să se creeze un tabel care conţine codurile, titlurile şi valorile corespunzătoare operelor care au fost expuse în galeria având codul 100.

```
CREATE TABLE opere_100
   AS SELECT cod_opera, titlu, valoare
      FROM   opera
      WHERE  cod_galerie = 100;
```

Prin *clauza_paralelism* se poate paraleliza operaţia de creare a unui tabel. De asemenea, clauza stabileşte gradul implicit de paralelism al interogărilor şi instrucţiunilor *LMD* ce vor fi efectuate asupra tabelului. Gradul de paralelism al unei operaţii este egal cu numărul de fire paralele de execuţie (*thread*) utilizate pentru realizarea acesteia. Fiecare fir de execuţie poate utiliza unul sau două *server*-e cu execuţie paralelă. Sintaxa clauzei este următoarea:

{***NOPARALLEL*** | ***PARALLEL*** [*nr_întreg*] }

Opţiunea *NOPARALLEL* este implicită şi determină execuţia serială a operaţiei de creare, a cererilor şi comenzilor de prelucrare. Pentru indicarea unui anumit grad de paralelism, se precizează un număr întreg în clauza *PARALLEL*. Acest lucru nu este, însă, necesar deoarece sistemul *Oracle* poate determina gradul optim de paralelism.

Utilizarea paralelismului accelerează crearea tabelului, iar operaţiile de interogare şi prelucrare asupra acestuia vor fi mai rapide.

***Exemplu*:**

Să se creeze tabelul *opere_100* utilizând un număr optim de *server*-e cu execuţie paralelă pentru a scana tabelul *opera* şi a introduce linii în tabelul creat.

# Indecşi

Un index este un obiect al schemei unei baze de date care:

- creşte viteza de execuţie a cererilor;
- garantează că o coloană conţine valori unice.

*Server*-ul *Oracle* utilizează identificatorul *ROWID* pentru regăsirea liniilor în structura fizică a bazei de date. Indexul, din punct de vedere logic, este compus dintr-o valoare cheie şi din identificatorul adresă *ROWID*.

Cheia indexului poate fi coloana unui tabel sau concatenarea mai multor coloane (numărul maxim de coloane care pot defini cheia indexului este 32). Coloanele care apar în cheia indexului trebuie declarate *NOT NULL* în tabel.

Indecşii sunt utilizaţi şi întreţinuţi automat de către *server*-ul Oracle. Odată creat indexul, el nu necesită o acţiune directă din partea utilizatorului.

Indecşii pot fi creaţi în două moduri:

- automat, de *server*-ul *Oracle* (*PRIMARY KEY*, *UNIQUE KEY*);
- manual, de către utilizator (*CREATE INDEX*, *CREATE TABLE*).

*Server*-ul *Oracle* creează automat un index unic atunci când se defineşte o constrângere *PRIMARY KEY* sau *UNIQUE* asupra unei coloane sau unui grup de coloane. Numele indexului va fi acelaşi cu numele constrângerii.

Indecşii, fiind obiecte ale schemei bazei, beneficiază de procesul de definire a unui obiect.

**Crearea unui index** (care nu este obligatoriu unic) pe una sau mai multe coloane ale unui tabel se face prin comanda:

*CREATE* [*UNIQUE*] *INDEX* <*nume_index*>
*ON* [<*nume_schema*>.] <*nume_tabel*>
(<*nume_col*> [*ASC* | *DESC*], <*nume_col*> [*ASC* | *DESC*], ...)
| *CLUSTER* <*nume_cluster*>];

Când este creat un index, un segment de date este rezervat automat în spaţiul tabel. Alocarea de memorie este controlată prin clauzele *INITIAL*, *PCTINCREASE*, *PCTFREE*, *NEXT*, care pot să apară în comanda *CREATE INDEX*.

Gestiunea inserţiilor şi a reactualizărilor se face, ca şi la tabele, utilizând parametrii *PCTFREE* şi *PCTUSED*.

***Exemplu*:**

**1)** Să se creeze un index descrescător relativ la coloana *adresa* din tabelul *cititor*.

```
CREATE INDEX  cititor_idx
       ON cititor (adresa DESC);
```

**2)** Să se afişeze informaţiile referitoare la indexul *cititor_idx*.

```
SELECT   TABLE_NAME, UNIQUENESS, MIN_EXTENTS
FROM     USER_INDEXES
WHERE    INDEX_NAME='cititor_idx';
```

**3)** Să se creeze un index explicit, referitor la cheia primara a unui tabel.

```
CREATE TABLE exemplu
(cod_id NUMBER(6)
        PRIMARY KEY USING INDEX
        (CREATE INDEX cod_id_idx ON exemplu (cod_id)),
 nume VARCHAR2(20));
```

Mai mulţi indecşi asupra unui tabel nu implică întotdeuna interogări mai rapide. Fiecare operaţie *LMD* care este permanentizată asupra unui tabel cu indecşi asociaţi presupune actualizarea indecşilor respectivi.

Prin urmare, **este recomandată** crearea de indecşi numai în anumite situaţii:

- coloana conţine o varietate mare de valori;
- coloana conţine un număr mare de valori *null*;
- una sau mai multe coloane sunt utilizate frecvent împreună într-o clauză *WHERE* sau într-o condiţie de *join*;
- tabelul este mare şi este de aşteptat ca majoritatea interogărilor asupra acestuia să regăsească mai puţin de 2-4% din linii.

Crearea unui index **nu este recomandată** în următoarele cazuri:

- tabelul este mic;
- coloanele nu sunt utilizate des în cadrul unei condiţii dintr-o cerere;
- majoritatea interogărilor regăsesc mai mult de 2-4% din liniile tabelului;
- tabelul este actualizat frecvent;
- coloanele indexate sunt referite în expresii.

**In concluzie, ce tabele sau ce coloane trebuie (sau nu) indexate?**

- indexaţi tabelele pentru care interogările selectează un număr redus de rânduri (sub 5%);
- indexaţi tabelele care sunt interogate folosind clauze *SQL* simple;
- nu indexaţi tabelele ce conţin puţine înregistrări (accesul secvenţial este mai simplu);

- nu indexaţi tabelele care sunt frecvent actualizate, deoarece ştergerile, inserările şi modificările sunt îngreunate de indecşi;
- indexaţi coloanele folosite frecvent în clauza *WHERE* sau în clauza *ORDER BY*;
- nu indexaţi coloanele ce conţin date asemănătoare (puţine valori distincte);

***Exemplu*:**

```
CREATE INDEX upper_nume_idx ON artist (UPPER(nume));
```

Indexul creat prin instrucţiunea precedentă facilitează prelucrarea unor interogări precum:

```
SELECT   *  FROM     artist
WHERE    UPPER(nume) = 'GRIGORESCU';
```

Pentru a asigura că *server*-ul *Oracle* utilizează indexul şi nu efectuează o căutare asupra întregului tabel, valoarea funcţiei corespunzătoare expresiei indexate trebuie să nu fie *null* în interogările ulterioare creării indexului. Următoarea instrucţiune garantează utilizarea indexului dar, în absenţa clauzei *WHERE*, *server*-ul *Oracle* ar putea cerceta întreg tabelul.

```
SELECT   *  FROM     artist
WHERE    UPPER(nume) IS NOT NULL
ORDER BY UPPER(nume);
```

***Exemplu*:**

Să se creeze tabelul *artist*, specificându-se indexul asociat cheii primare.

```
CREATE TABLE artist (
   cod_artist    NUMBER
                 PRIMARY KEY USING INDEX
                 (CREATE INDEX artist_cod_idx
                  ON artist(cod_artist)),
   nume          VARCHAR2(30) NOT NULL,
   prenume       VARCHAR2(30),    …);
```

**Ştergerea unui index** se face prin comanda:

*DROP INDEX nume_index* [*ON* [*nume_schema.*] *nume_tabel*]

Pentru a suprima indexul trebuie ca acesta să se găsească în schema personală sau să ai privilegiul de sistem *DROP ANY INDEX*.

Pentru a **reconstrui un index** se pot folosi două metode:

- se şterge indexul (*DROP INDEX*) şi se recreează (*CREATE INDEX*);

- se utilizează comanda *ALTER INDEX* cu opţiunea *REBUILD*.

**Modificarea parametrilor de stocare a indecşilor** (*STORAGE*), alocarea (*ALLOCATE EXTENT*) şi dealocarea (*DEALLOCATE UNUSED*) manuală a spaţiului utilizat de un index se pot realiza cu ajutorul comenzii *ALTER INDEX*. Începând cu *Oracle9i*, este posibilă modificarea structurii indecşilor prin comanda *ALTER INDEX*.

**Validarea unui index**, adică verificarea integrităţii indexului specificat pentru un tabel, se face prin comanda:

*VALIDATE INDEX nume_index* [*ON nume_tabel*] [*WITH LIST*]

*Oracle8* foloseşte următoarele **tipuri de indecşi**:

- index de tip arbore *B** – creat la executarea unei comenzi standard *CREATE INDEX*;
- index partiţionat – folosit în cazul tabelelor mari pentru a stoca valorile coloanei indexate în mai multe segmente;
- index de *cluster* – bazat pe coloanele comune ale unui *cluster*;
- index cu cheie inversă – sunt *B** arbori, dar care stochează datele în mod invers;
- index de tip *bitmap* – nu se stochează valorile efective ale coloanei indexate, ci un *bitmap* format pe baza acestor valori.

Versiunea *Oracle*8 permite **construirea de tabele organizate pe bază de index**. În acest caz, datele sunt stocate în indexul asociat. Un astfel de tabel poate fi manipulat de către aplicaţii la fel ca un tabel obişnuit, folosind comenzi *SQL*. Diferenţa constă în faptul că în cazul tabelului organizat pe bază de index, toate operaţiile sunt efectuate numai asupra indexului. În loc ca fiecare intrare a indexului să conţină valoarea coloanei sau coloanelor indexate şi valoarea *ROWID* (care identifică unic un rând) pentru rândul corespunzător, ea conţine întreg rândul.

Coloana sau coloanele după care se face indexarea sunt cele care constituie cheia primară a tabelului.

Informaţii referitoare la indecşi şi la coloanele care îi definesc pot fi regăsite în vizualizările *USER_INDEXES*, respectiv *USER_IND_COLUMNS* din dicţionarul datelor. Dintre coloanele tabelului *USER_INDEXES* se remarcă:

| | |
|---|---|
| *INDEX_NAME* | Numele indexului |
| *INDEX_TYPE* | Tipul indexului (*NORMAL, LOB, CLUSTER* etc.) |
| *TABLE_OWNER* | Proprietarul tabelului indexat |
| *TABLE_NAME* | Numele tabelului indexat |

| | |
|---|---|
| *TABEL_TYPE* | Tipul tabelului indexat (*TABLE*, *CLUSTER* etc.) |
| *UNIQUENESS* | Starea de unicitate (*UNIQUE*, *NONUNIQUE*) |
| *TABLESPACE_NAME* | Spaţiul tabel în care este stocat indexul |
| *INITIAL_EXTENT* | Spaţiul alocat pentru prima extensie |
| *NEXT_EXTENT* | Spaţiul alocat pentru următoarea extensie |
| *MIN_EXTENTS* | Numărul minim de extensii alocate |
| *MAX_EXTENTS* | Numărul maxim de extensii |
| *PCT_INCREASE* | Procentul cu care cresc extensiile |
| *BLEVEL* | Nivelul din B-arbore. Acesta arată adâncimea indexului de la ramuri la frunze |
| *LEAF_BLOCKS* | Numărul de blocuri frunză din index |
| *DISTINCT_KEYS* | Numărul de chei distincte în index |
| *STATUS* | Starea indexului (*VALID*, *INVALID*, *DIRECT_LOAD*) |
| *NUM_ROWS* | Numărul de linii utilizate. Acesta nu trebuie să includă valorile *NULL* din tabelul de bază |
| *PARTITIONED* | Determină dacă indexul este partiţionat (*YES* sau *NO*) |
| *GENERATED* | Determină dacă sistemul a generat numele indexului (*Y*) sau utilizatorul (*N*) |

***Exemplu***:

Să se obţină informaţii referitoare la indecşii tabelului *carte.*

```
SELECT   a.index_name, a.column_name,
         a.column_position poz, b.uniqueness
FROM     user_indexes b, user_ind_columns a
WHERE    a.index_name = b.index_name
AND      a.table_name = 'carte';
```

# Secvenţe

O secvenţă este un obiect în baza de date care serveşte pentru a genera întregi unici în sistemele multi-utilizator, evitând apariţia conflictelor şi a blocării.

Secvenţele sunt memorate şi generate indiferent de tabele ➔ aceeaşi secvenţă poate fi utilizată pentru mai multe tabele. O secvenţă poate fi creată de un utilizator şi poate fi partajată de mai mulţi utilizatori.

**Crearea unei secvenţe** se face cu ajutorul comenzii:

*CREATE SEQUENCE* [<*nume_schema*>.]<*nume_secventa*>
 [*INCREMENT BY n*] [*START WITH m*]
 [{*MAXVALUE n* | *NOMAXVALUE*}] [{*MINVALUE n* | *NOMINVALUE*}]
 [{*CACHE k* | *NOCACHE*}]

*CACHE k | NOCACHE* specifică numărul de valori alocate de *server*-ul *Oracle* pe care le va păstra în memoria *cache* pentru a oferi utilizatorilor un acces rapid (implicit sunt alocate 20 de valori);

O secvenţă este referită într-o comandă *SQL* cu ajutorul pseudo-coloanelor:

- *NEXTVAL* – referă valoarea următoare a secvenţei;
- *CURRVAL* – referă valoarea curentă a secvenţei.

*NEXTVAL* şi *CURRVAL* **pot** fi folosite în:

- clauza *VALUES* a unei comenzi *INSERT*;
- clauza *SET* a unei comenzi *UPDATE*;
- lista *SELECT* a unei subcereri dintr-o comanda *INSERT*;
- lista unei comenzi *SELECT*.

*NEXTVAL* şi *CURRVAL* **nu pot** fi folosite în:

- subinterogare in *SELECT*, *DELETE* sau *UPDATE*;
- interogarea unei vizualizări;
- comandă *SELECT* cu operatorul *DISTINCT*;
- comandă *SELECT* cu clauza *GROUP BY*, *HAVING* sau *ORDER BY*;
- clauza *WHERE* a unei comenzi *SELECT*;
- condiţia unei constrângeri *CHECK*;
- valoarea *DEFAULT* a unei coloane într-o comandă *CREATE TABLE* sau *ALTER TABLE*;
- comandă *SELECT* care este combinată cu altă comandă *SELECT* printr-un operator mulţime (*UNION*, *INTERSECT*, *MINUS*).

Din dicţionarul datelor pot fi obţinute informaţii despre secvenţe folosind vizualizarea *USER_SEQUENCES*.

**1)** Să se creeze o secvenţă *domeniuseq* care să fie utilizată pentru a insera noi domenii în tabelul *domeniu* şi să se insereze un nou domeniu.

**2)** Să se afişeze informaţiile referitoare la secvenţa *domeniuseq*.

```
CREATE SEQUENCE domeniuseq
  START WITH 1
  INCREMENT BY 1;
INSERT INTO domeniu
VALUES (domeniuseq.NEXTVAL,'Informatica');
```

```
SELECT INCREMENT, START, MAXVALUE, MINVALUE,
FROM   USER_SEQUENCES
WHERE  SEQUENCE_NAME = 'domeniuseq';
```

**Modificarea unei secvenţe** se face prin comanda *ALTER SEQUENCE*. Sintaxa comenzii este similară instrucţiunii *CREATE SEQUENCE* , dar:

- noua valoare maximă pentru *MAXVALUE* nu poate fi mai mică decât valoarea curentă;
- opţiunea *START WITH* nu poate fi modificată de comandă.

**Suprimarea unei secvenţe** se face cu ajutorul comenzii:

*DROP SEQUENCE* [<*nume_schema*>.]<*nume_secventa*>;

După ce a fost ştearsă, secvenţa nu mai poate fi referită. Pentru a putea şterge sau modifica secvenţa trebuie fie să fi proprietarul acesteia, fie să ai privilegiul de sistem *DROP ANY SEQUENCE*, respectiv privilegiul *ALTER SEQUENCE*.

# Comentarii

Sistemul *Oracle* oferă posibilitatea de a comenta obiectele create, printr-un text care este inserat în dicţionarul datelor. Comentariul se poate referi la tabele, vizualizări, clişee sau coloane.

```
COMMENT ON {TABLE nume_obiect | COLUMN
    nume_obiect.nume_coloana}    IS 'text comentariu'
```

# Sinonime

*Oracle* oferă posibilitatea de a atribui mai multe nume aceluiaşi obiect. Aceste nume adiţionale sunt numite sinonime (*synonymes*). Ele sunt utile deoarece permit simplificarea formulării cererii şi referirea la obiecte, fără a fi nevoie să se specifice proprietarii obiectelor sau localizarea acestora.

Spre deosebire de *alias* a cărui durată de viaţă este limitată la cererea ce conţine *alias*-ul, sinonimele sunt salvate în dicţionarul datelor şi pot fi reutilizate.

Sistemul *Oracle* permite crearea de sinonime pentru obiecte de tipul: tabel, vizualizare, secvenţă, funcţie, procedură, pachet, clişeu, sinonim.

```
CREATE [PUBLIC] SYNONYM [schema.]nume_sinonim
    FOR [schema.]obiect
```

Administratorul bazei poate produce şi poate suprima sinonime publice sau private, iar utilizatorii pot genera sau suprima doar sinonime private. Pentru suprimarea unui sinonim din baza de date se utilizează comanda:

```
DROP [PUBLIC] SYNONYM [schema.]nume_sinonim
```

# Definirea legăturilor între baze de date

O legătură între două baze de date (*database link*) este un obiect al unei scheme dintr-una din baze, care permite accesarea obiectelor din cealaltă bază de date. A doua bază poate să nu aparţină sistemului *Oracle*.

Odată creată o legătură, aceasta poate fi utilizată în instrucţiunile *SQL* pentru a face referinţă la tabele sau vizualizări dintr-o altă bază de date, prin sufixarea numelor acestora cu şirul de caractere *@nume_legătură_bd*. Prin intermediul unei legături se pot interoga tabele sau vizualizări din cealaltă bază de date. De asemenea, aceste obiecte distante pot constitui subiectul instrucţiunilor *INSERT*, *UPDATE*, *DELETE* sau *LOCK TABLE*.

O legătură între baze de date se creează prin comanda *CREATE DATABASE LINK*, care are următoarea sintaxă simplificată:

***CREATE*** [***SHARED***] [***PUBLIC***] ***DATABASE LINK*** *nume_legătura_bd*
[ {***CONNECT TO*** {***CURRENT_USER*** | *utilizator*
***IDENTIFIED BY*** *parola* [*clauza_autentificare*] }
| *clauza_autentificare*} ] [***USING*** '*şir_conectare*'];

Cuvântul cheie *SHARED* determină utilizarea unei singure conexiuni de reţea pentru a crea o legătură publică între bazele de date. Aceasta va putea fi folosită de mai mulţi utilizatori.

Cuvântul cheie *PUBLIC* determină crearea unei legături publice între bazele de date. O astfel de legătură este disponibilă tuturor utilizatorilor. Dacă se omite această opţiune, legătura este privată şi, în acest fel, disponibilă numai utilizatorului care a creat-o.

Prin *nume_legătura_bd* se poate specifica numele complet sau parţial al legăturii bazei de date. Dacă se precizează numai numele bazei de date, sistemul va adăuga implicit domeniul bazei locale.

O legătură între baze de date nu poate fi creată în schema altui utilizator. De alfel, *nume_legătura_bd* nu poate fi prefixat de numele unei scheme.

Clauza *USING* specifică numele serviciului corespunzător bazei de date distante. Dacă se precizează doar numele bazei, sistemul adaugă implicit numele domeniului. Prin urmare, dacă numele domeniului bazei distante este diferit de numele domeniului bazei curente, trebuie specificat întreg numele serviciului.

***Exemplu*:**

a) Să se definească o legătură de baze de date publică partajată care face referinţă la baza de date specificată prin numele de serviciu *arta*.

```
CREATE SHARED PUBLIC DATABASE LINK distant
  USING 'arta';
```

b) Se presupune că tabelul *opera* se află în baza de date accesata prin legatura *distant*. Să se mărească cu 10% valoarea operelor de artă create de artistul având codul 100.

```
UPDATE  opera@distant
SET     valoare = valoare * 1.1
WHERE   cod_artist = 100;
```

Clauza *CONNECT TO* permite activarea conexiunii la baza distantă. Cu ajutorul opţiunii *CURRENT_USER* se poate crea o legătură a utilizatorului curent. Acesta trebuie să aibă un cont valid în baza de date distantă şi să fie un utilizator global.

Clauza *IDENTIFIED BY* specifică numele utilizatorului şi parola pentru conexiunea la baza de date distantă, creând o legătură a unui utilizator fixat. În absenţa acestei clauze, legătura va folosi numele şi parola fiecărui utilizator care este conectat la bază.

***Exemplu*:**

1. Se presupune că utilizatorul *student*, având parola *oracle*, de pe baza de date *distant* defineşte o legătură proprie, numită *local*, către schema *student* de pe baza de date *local*.

```
CREATE DATABASE LINK local
CONNECT TO student IDENTIFIED BY oracle
USING 'local';
```

2. Să se afişeze conţinutul tabelului *artist* din schema *student* a bazei de date *local*.

```
SELECT *
FROM   artist@local;
```

3. Se presupune că utilizatorul *student* are privilegiul *SELECT* asupra tabelului *profesor.opera*. Să se afişeze, de către *student*, conţinutul acestui tabel.

```
SELECT * FROM profesor.opera@local;
```

Specificarea numelui utilizatorului şi a parolei pentru instanţa destinaţie are loc în *clauza_autentificare*. Această informaţie este folosită pentru validarea utilizatorului pe *server*-ul distant şi este necesară din considerente de securitate. Numele şi parola sunt utilizate exclusiv pentru autentificare, nefiind efectuată nici o altă operaţie în numele acestui utilizator. Clauza este obligatorie dacă se precizează cuvântul cheie *SHARED*.

***Exemplu*:**

1. Să se definească o legătură a utilizatorului curent către baza de date *distant*, utilizând numele întreg al serviciului ca identificator al legăturii.

```
CREATE DATABASE LINK distant.info.univ.ro
  CONNECT TO CURRENT_USER USING 'distant';
```

2. Să se definească un sinonim care să mascheze faptul că tabelul *opera* al schemei *student* se află pe baza de date *distant*.

```
CREATE SYNONYM tab_opera
  FOR    student.opera@distant.info.univ.ro;
```

Odată accesată, o legătură către o bază de date rămâne deschisă până la închiderea sesiunii. O legătură este deschisă în sensul că există un proces activ pe fiecare dintre bazele de date distante accesate prin legătură.

La închiderea unei sesiuni, legăturile care au fost active vor fi închise în mod automat. Dacă se stabileşte o conexiune de reţea care nu este utilizată frecvent în aplicaţie, poate fi utilă închiderea manuală a legăturii active din sesiunea curentă. Instrucţiunea care permite acest lucru este următoarea:

***ALTER SESSION CLOSE DATABASE LINK*** *nume_legătura_bd*;

Suprimarea unei legături către o bază de date se realizeaza prin comanda:

***DROP*** [***PUBLIC***] ***DATABASE LINK*** *nume_legătura_bd*;

În cazul ştergerii unei legături private, aceasta trebuie să aparţină schemei utilizatorului care iniţiază suprimarea. Suprimarea unei legături publice necesită privilegiul corespunzător (*DROP PUBLIC DATABASE LINK*).

# Vizualizări

Vizualizarea (*view*) este un tabel logic (virtual) relativ la date din una sau mai multe tabele sau vizualizări. Vizualizarea este definită plecând de la o cerere a limbajului de interogare a datelor, moştenind caracteristicile obiectelor la care se referă. Vizualizarea, fiind virtuală, nu solicită o alocare de memorie pentru date. Ea este definită în DD cu aceleaşi caracteristici ca şi un tabel.

Textul cererii care defineşte vizualizarea este salvat în DD. Nucleul *Oracle* determină fuzionarea cererii relative la vizualizare cu comanda de definire a vizualizării, analizează rezultatul fuziunii în zona partajată şi execută cererea.

➔ *Oracle* transformă cererea referitoare la o vizualizare într-o cerere relativă la tabelele de bază. Vizualizarea este memorata in *DD* sub forma unui *SELECT*.

Dacă sunt utilizate clauzele *UNION, GROUP BY* şi *CONNECT BY,* atunci *Oracle* nu determină fuzionarea, el va rezolva vizualizarea şi apoi va aplica cererea rezultatului obţinut.

O vizualizare reflectă la orice moment conţinutul exact al tabelelor de bază. Orice modificare efectuată asupra tabelelor se repercutează instantaneu asupra vizualizării. Ştergerea unui tabel implică invalidarea vizualizărilor asociate tabelului şi nu ştergerea acestora.

Vizualizările sunt definite pentru:

- furnizarea unui nivel mai înalt de securizare a bazei;
- simplificarea formulării unei cereri;
- mascarea complexităţii datelor;
- afişarea datelor într-o altă reprezentare decât cea a tabelelor de bază;
- asigurarea independenţei datelor;
- asigurarea confidenţialităţii anumitor informaţii;
- definirea constrângerilor de integritate;
- restricţionarea acesului la date.

Vizualizările pot fi simple şi complexe. O vizualizare **simplă** extrage date dintr-un singur tabel, nu conţine funcţii sau grupări de date şi asupra ei pot fi efectuate operaţii *LMD*.

O vizualizare este considerată **complexă** dacă extrage date din mai multe tabele, conţine funcţii sau grupări de date şi nu permite întotdeauna (prin intermediul său) operaţii *LMD* asupra tabelelor de bază.

Operaţiile *LMD* asupra vizualizărilor complexe sunt restricţionate de următoarele reguli:

- nu se poate insera, actualiza sau şterge o linie dintr-o vizualizare dacă aceasta conţine funcţii grup, clauza *GROUP BY*, cuvântul cheie *DISTINCT* sau pseudocoloana *ROWNUM*;
- nu se poate adăuga sau modifica o linie dintr-o vizualizare, dacă aceasta conţine coloane definite prin expresii;
- nu pot fi adăugate linii printr-o vizualizare, dacă tabelul de bază conţine coloane care au constrângerea *NOT NULL* şi nu apar în lista *SELECT* a vizualizării.

Pentru a obţine informaţii referitoare la vizualizările definite, se pot interoga vizualizările *USER_VIEWS* şi *ALL_VIEWS* din dicţionarul datelor. Textul instrucţiunii *SELECT* care defineşte o vizualizare este stocat într-o coloană de tip *LONG*, numită *TEXT*.

Atunci când datele sunt accesate prin intermediul unei vizualizări, *server*-ul *Oracle* efectuează următoarele operaţii:

- recuperează definiţia acesteia din *USER_VIEWS*;
- verifică privilegiile de acces la tabelele ei de bază;
- converteşte cererea într-o operaţie echivalentă asupra tabelelor de bază.

**Crearea unei vizualizări** se realizează cu ajutorul comenzii:

*CREATE* [*OR REPLACE*][*FORCE* | *NOFORCE*] *VIEW*
[<*nume_schema*>.]<*nume_view*> [(<*alias*>[,<*alias*>]...)]
*AS* <*cerere_SELECT*>
[*WITH* {*CHECK OPTION* [*CONSTRAINT* <*nume_constrangere*>] |
*READ ONLY* }];

- *OR REPLACE* recreează vizualizarea dacă aceasta deja există.
- *FORCE* creează vizualizarea chiar dacă tabelul de bază nu există sau chiar dacă vizualizarea face referinţă la obiecte care încă nu sunt create. Deşi vizualizarea va fi creată, utilizatorul nu poate să o folosească.
- *NO FORCE* este implicită şi se referă la faptul că vizualizarea este creată numai dacă tabelele de bază există.
- Cererea este o comandă *SELECT* care poate să conţină *alias* pentru coloane.
- *WITH CHECK OPTION* specifică faptul că reactualizarea datelor din tabele (inserare sau modificare) se poate face numai asupra datelor selectate de vizualizare (care apar în clauza WHERE).

– *WITH READ ONLY* asigură că nici o operaţie *LMD* nu poate fi executată asupra vizualizării.

***Exemplu*:**

Să se genereze o vizualizare care conţine informaţii referitoare la împrumutul cărţilor şi în care să fie implementată constrîngerea că orice carte, care există într-un singur exemplar, poate fi împrumutată maximum 15 zile.

```
CREATE VIEW imprumutare
    AS SELECT    *
    FROM     imprumuta
    WHERE    codel NOT IN
                 (SELECT  codel
                  FROM    carte
                  WHERE   nrex = 1)
    OR       datares - dataim < 15
    WITH CHECK OPTION;
```

Cererea care defineşte vizualizarea poate fi complexă, incluzând *join*-uri, grupări şi subcereri, însă nu poate conţine clauza *ORDER BY*. Dacă este necesar, această clauză poate fi specificată la interogarea vizualizării. Interogarea unei vizualizări este similară celei unui tabel. Numărul coloanelor specificate în definiţia vizualizării trebuie să fie egal cu cel din lista asociată comenzii *SELECT*.

Asupra cererii care defineşte vizualizarea se impun următoarele restricţii:

- nu pot fi selectate pseudocoloanele *CURRVAL* şi *NEXTVAL* ale unei secvenţe;
- dacă sunt selectate pseudocoloanele *ROWID*, *ROWNUM* sau *LEVEL*, acestora trebuie să li se specifice *alias*-uri;
- dacă cererea selectează toate coloanele unui tabel, utilizând simbolul „*“, iar ulterior se adaugă coloane noi tabelului, vizualizarea nu va conţine acele coloane până la recrearea sa printr-o instrucţiune *CREATE OR REPLACE VIEW*;
- pentru vizualizările obiect, numărul şi tipul elementelor selectate de cerere trebuie să coincidă cu cel al atributelor de pe primul nivel al tipului obiect.

Aportul versiunii *Oracle9i* în ceea ce priveşte instrucţiunea *CREATE VIEW* constă în posibilitatea:

- creării de subvizualizări ale vizualizărilor obiect;
- definirii de constrângeri asupra vizualizărilor.

***Exemplu*:**

a) Să se creeze o vizualizare care conţine numele şi prenumele artistului, numărul operelor sale şi valoarea medie a acestora.

```
CREATE VIEW artist_nr_val(nume, numar_opere, val_medie)
AS SELECT   a.nume || ' ' || a .prenume "Nume si prenume",
            COUNT(o. cod_opera) numar, AVG(o.valoare) medie
   FROM     opera o, artist a
   WHERE    o.cod_artist = a.cod_artist
   GROUP BY o.cod_artist, a.nume, a.prenume;
```

b) Să se creeze vizualizarea *sculptura* ce va conţine codul operei, data achiziţiei, codul artistului şi stilul operelor al căror tip este „sculptura“.

```
CREATE OR REPLACE VIEW sculptura
         (cod_sculptura, informatii, cod_sculptor, stil)
AS SELECT   cod_opera,
            'Sculptura ' || titlu ||
            ' a fost achizitionata la data ' ||
            data_achizitiei, cod_artist, stil
   FROM     opera
   WHERE    tip = 'sculptura';
```

**Modificarea unei vizualizări** presupune modificarea definiţiei acesteia. Pentru a înlocui o vizualizare trebuie avut privilegiul de sistem necesar pentru distrugerea şi crearea acesteia. Înlocuirea se poate face în două moduri.

- Vizualizarea poate fi distrusă (*DROP VIEW*) şi apoi recreată (*CREATE*) cu noua definiţie. Atunci când este distrusă, toate privilegiile sunt retrase. Aceste privilegii trebuie să fie create pentru noua vizualizare.
- Vizualizarea poate fi recreată prin redefinire cu instrucţiunea *CREATE VIEW*, dar cu clauza *OR REPLACE*. Această metodă conservă toate privilegiile curente.

In *Oracle9i* este posibila adaugarea de constrangeri unei vizualizari prin comanda *ALTER VIEW*.

Modificarea unui vizualizări are următoarele efecte:

- definiţia vizualizării din DD este actualizată;
- nici unul din obiectele de bază nu este afectat de înlocuire;
- toate restricţiile care existau în vizualizarea originală sunt distruse;

- toate vizualizările şi programele *PL/SQL* dependente de vizualizarea înlocuită devin invalide.

**Suprimarea unei vizualizări** se realizează prin comanda *DROP VIEW* care şterge definiţia vizualizării din baza de date.

*DROP VIEW <nume_view>* [*CASCADE CONSTRAINT*];

Ştergerea vizualizării nu va afecta tabelele relativ la care a fost definită vizualizarea. Aplicaţiile şi vizualizările care se bazează pe vizualizarea suprimată devin invalide. Pentru a suprima o vizualizare, utilizatorul trebuie să aibă privilegiul *DROP ANY VIEW* sau să fie creatorul vizualizării respective.

Similar opţiunii corespunzătoare din comanda *DROP TABLE*, clauza *CASCADE CONSTRAINTS* permite suprimarea tuturor constrângerilor de integritate referenţială corespunzătoare cheilor primare şi unice din vizualizarea supusă ştergerii. Dacă se omite această clauză şi există astfel de constrângeri, instrucţiunea *DROP VIEW* va eşua.

**Recompilarea unei vizualizări** permite detectarea eventualelor erori referitoare la vizualizare, înaintea executării vizualizării. După fiecare modificare a tabelelor de bază este recomandabil ca vizualizarea să se recompileze:

*ALTER VIEW <nume_view> COMPILE*;

**Reactualizarea tabelelor implică reactualizarea corespunzătoare a vizualizărilor!!!**

## Reactualizarea vizualizărilor implică reactualizarea tabelelor de bază? NU! Există restricţii care trebuie respectate!!!

- Nu pot fi modificate date din vizualizare sau adaugate date prin vizualizare, daca aceasta contine coloane definite prin expresii.
- Nu pot fi înserate, şterse sau actualizate date din vizualizări ce conţin: operatorul *DISTINCT*; clauzele *GROUP BY*, *HAVING*, *START WITH*, *CONNECT BY*; pseudo-coloana *ROWNUM*; funcţii grup; operatori de mulţimi.
- Nu pot fi inserate sau actualizate date care ar încălca constrângerile din tabelele de bază.
- Nu pot fi inserate sau actualizate valorile coloanelor care rezultă prin calcul.
- Nu se pot face operaţii *LMD* asupra coloanelor calculate cu *DECODE*.

Alături de restricţiile prezentate anterior, aplicabile tuturor vizualizărilor, există restricţii specifice, aplicabile vizualizărilor bazate pe mai multe tabele.

**Regula fundamentală** este că orice operaţie *INSERT*, *UPDATE* sau *DELETE* pe o vizualizare bazată pe mai multe tabele poate modifica datele doar din unul din tabelele de bază. In care???

Un tabel de bază al unei vizualizări este **protejat prin cheie** (*key preserved table*) dacă orice cheie selectată a tabelului este de asemenea şi cheie a vizualizării. Deci, un tabel protejat prin cheie este un tabel ale cărui chei se păstrează şi la nivel de vizualizare. Pentru ca un tabel să fie protejat prin cheie **nu** este necesar ca tabelul să aibă toate cheile selectate în vizualizare. Este suficient ca, atunci când cheia tabelului este selectată, aceasta să fie şi cheie a vizualizării.

Asupra unui *join view* pot fi aplicate instrucţiunile *INSERT*, *UPDATE* sau *DELETE*, doar dacă sunt îndeplinite următoarele condiţii:

- instrucţiunea *LMD* afectează numai unul dintre tabelele de bază;
- în cazul instrucţiunii *UPDATE*, toate coloanele care pot fi reactualizate trebuie să corespundă coloanelor dintr-un tabel protejat prin cheie (în caz contrar, *Oracle* nu va putea identifica unic înregistrarea care trebuie reactualizată);
- în cazul instrucţiunii *DELETE*, rândurile unei vizualizări pot fi şterse numai dacă există un tabel în *join* protejat prin cheie şi numai unul (în caz contrar, *Oracle* nu ar şti din care tabel să şteargă);
- în cazul instrucţiunii *INSERT*, toate coloanele în care sunt inserate valori trebuie să provină dintr-un tabel protejat prin cheie.

*ALL_UPDATABLE_COLUMNS*, *DBA_UPDATABLE_COLUMNS* şi *USER_UPDATABLE_COLUMNS* sunt vizualizări din DD ce conţin informaţii referitoare la coloanele vizualizărilor existente, care pot fi reactualizate.

***Exmplu*:**

1. Să se creeze un *view* ce conţine câmpurile *nume, prenume, job* din tabelul *salariat*.
2. Să se insereze, să se actualizeze şi să se şteargă o înregistrare în acest *view*. Ce efect vor avea aceste acţiuni asupra tabelului de bază?

```
CREATE VIEW  vederea2
AS SELECT    nume, prenume, job
FROM         salariat;
```

Nu se pot face inserari deoarece *view-ul* nu conţine cheia primară!

```
INSERT INTO  vederea2
VALUES       ('Popescu','Valentin','grafician');
```

va genera eroarea:

```
ORA-01400: cannot insert NULL into
            ("SCOTT"."SALARIAT"."COD_SALARIAT")
```

Actualizarea *job*-ului salariatului având numele "Popescu":

```
UPDATE   vederea2
SET      job = 'programator'
WHERE    nume = 'Popescu';
SELECT   nume, prenume, job  FROM      salariat;
```

Ştergerea înregistrării referitoare la salariatul având numele "Popescu":

```
DELETE   vederea2
WHERE    nume = 'Popescu';
```

Operaţiile care se realizează asupra *view*-ului se realizează şi în tabelul *salariat*. Pentru un caz mai general, când *view*-ul conţine cheia externă a tabelului de bază, sunt permise modificări ale *view*-ului, dacă acestea nu afectează cheia externă.

***Exemplu*:**

Să se creeze un *view* care conţine câmpurile *nume, prenume, job* din tabelul *salariat.* Să se introducă în *view* doar persoanele care sunt graficieni.

```
CREATE VIEW  vederea21
AS SELECT    nume, prenume, job
FROM         salariat
WHERE        job = 'grafician'
WITH CHECK OPTION;
```

Să se creeze o vizualizare care să conţină *cod_salariat, nume*, *prenume* din tabelul *salariat* şi coloana *tip* din tabelul *grafician*. Apoi să se insereze, să se actualizeze şi să se şteargă o înregistrare din acest *view* (vizualizarea conţine cheia primară *cod_salariat* din tabelele *salariat* şi *grafician*).

```
CREATE VIEW  vederea4
AS SELECT    s.cod_salariat,nume,prenume,tip
FROM         salariat s, grafician g
WHERE        s.cod_salariat=g.cod_salariat;
```

În cazul inserării unei înregistrări pentru care se specifică toate câmpurile:

```
INSERT INTO   vederea4
VALUES   (30,'Popescu','Valentin','artist plastic');
```

va apare următoarea eroare:

```
ORA-01776: cannot modify more than one base TABLE
through a join view
```

Pot fi inserate date doar într-un tabel de bază (în oricare, dar în unul singur) prin intermediul *view*-ului, astfel:

```
INSERT INTO  vederea4 (cod_salariat, nume)
VALUES       (30, 'Popescu');
```

Comanda pentru ştergerea unei înregistrări:

```
DELETE       vederea4
WHERE        cod_salariat = 3;
```

va genera următoarea eroare:

```
ORA-01752: cannot delete from view without exactly one
key-preserved TABLE.
```

Modificarea unei înregistrări se face prin secvenţa care urmează. Toate actualizările care se fac în *view* se fac şi în tabelele de bază.

```
UPDATE       vederea4
SET          tip = 'designer'
WHERE        cod_salariat = 3;
```

***Exemplu*:**

Care dintre coloanele unei vizualizări sunt actualizabile?

```
SELECT   column_name, updatable
FROM     user_updatable_columns
WHERE    table_name = 'vederea4';
```

***Exemplu*:**

1. Să se creeze un *view* (*vederea3*) care să conţină, pentru fiecare categorie de salariat, salariile medii şi numărul de angajaţi din tabelul *salariat.*
2. Să se insereze, să se actualizeze şi să se şteargă o înregistrare în *view.*

```
CREATE VIEW  vederea3 (nr, job, salmed)
AS SELECT    COUNT(*), job, AVG(salariu)
FROM         salariat
GROUP BY     job;
```

Nu se pot face inserări, actualizări sau ştergeri într-un *view* ce conţine funcţii grup. După oricare din aceste operaţii apare acelaşi mesaj:

```
ORA-01732: data manipulation operation not legal on
this view
```

***Exemplu*:**

Să se creeze o vizualizare care să conţină coloanele *cod_contractant*, *adresa*, *telefon* din tabelul *contractant* şi coloanele *nr_contract*, *tip_contract*, *data_incheiere* din tabelul *contract*. Să se insereze o înregistrare în vizualizare.

```
CREATE VIEW  vederea44
AS SELECT    c.cod_contractant, adresa, telefon,
             co.nr_contract, tip_contract,
             data_incheiere
FROM         contractant c, contract co
WHERE        c.cod_contractant=co.cod_contractant;
```

La inserarea unei înregistrări căreia i se specifică valorile tuturor câmpurilor din ambele tabele:

```
INSERT INTO  vederea44(cod_contractant, adresa,
                       nr_contract, data_incheiere)
VALUES       (200, 'Str. Marmurei, 14', '6235',
         TO_DATE('January 03,2002','Month dd,yyyy'));
```

se obţine eroarea:

```
ORA-01779: cannot modify a column which maps to a non
key-preserved TABLE
```

Cele două tabele de bază, *contractant* şi *contract*, se află într-o relaţie "one-to-many", iar *view*-ul creat conţine cheile primare din ambele tabele.

Doar tabelul *contract* este protejat prin cheie şi, prin urmare, doar el poate fi modificat prin intermediul *view*-ului. Aceasta, deoarece ar putea exista mai multe înregistrări în *view*, cu aceeaşi valoare corespunzătoare câmpului *cod_contractant* (CP în *contractant*).

Exact aceeaşi eroare se obţine dacă încercăm inserarea unei înregistrări în *vederea44*, specificând fie şi numai un câmp provenind din tabela *contractant* (indiferent dacă el conţine sau nu CP).

Singura operaţie de inserare permisă este aceea efectuată prin specificarea cheilor provenind doar din tabelul *contract*. Astfel, prin executarea comenzii:

```
INSERT INTO  vederea44(nr_contract, tip_contract)
VALUES       ('6234', 0);
```

este creată o înregistrare, dar este modificat şi tabelul *contract*. Dacă la inserţie nu se specifică cheia primară din *contract*:

```
INSERT INTO  vederea44(tip_contract)
VALUES       (1);
ORA-01400: mandatory (NOT NULL) column is missing or
NULL during insert
```

Cererea din definiţia vizualizării poate fi restricţionată prin clauzele *WITH READ ONLY* şi *WITH CHECK OPTION*. Opţiunea *WITH READ ONLY* asigură că nu pot fi efectuate operaţii *LMD* asupra vizualizării. Constrângerea *WITH CHECK OPTION* garantează faptul că va fi permisă, prin intermediul vizualizării, numai inserarea sau actualizarea de linii accesibile acesteia (care sunt selectate de cerere). Prin urmare, această opţiune asigură constrângeri de integritate şi verificări asupra validităţii datelor inserate sau actualizate.

Opţiunea *WITH CHECK OPTION* nu poate funcţiona dacă:

- există o cerere imbricată în cadrul subcererii vizualizării sau în vreuna dintre vizualizările de bază;
- operaţiile de inserare, ştergere şi modificare se fac prin intermediul declanşatorilor *INSTEAD OF*.

Cuvântul cheie *CONSTRAINT* permite numirea constrângerii *WITH CHECK OPTION*. În absenţa acestei clauze, constrângerea va avea un nume implicit de forma *SYS_Cn*, unde *n* este un număr întreg unic.

***Exemplu*:**

Să se creeze o vizualizare ce conţine artiştii de naţionalitate română, care au opere expuse în muzeu. Definiţia vizualizării nu va permite modificarea naţionalităţii unui artist sau inserarea unui artist având altă naţionalitate decât cea română.

```
CREATE VIEW artist_roman
AS SELECT   *  FROM     artist
   WHERE    nationalitate = 'romana'
WITH CHECK OPTION CONSTRAINT artist_roman_ck;
UPDATE   artist_roman
SET      nationalitate = 'engleza'
WHERE    cod_artist = 25;
```

Încercarea de actualizare a unei linii prin instrucţiunea anterioară va genera eroarea „*ORA-01402*: *view WITH CHECK OPTION where-clause violation*“.

***Exemplu*:**

Să se creeze o vizualizare asupra tabelului *galerie* care să nu permită efectuarea nici unei operaţii *LMD*.

```
CREATE VIEW viz_galerie
AS SELECT   cod_galerie, nume_galerie
   FROM     galerie
WITH READ ONLY;
DELETE FROM viz_galerie
WHERE  cod_galerie = 10;
```

Încercarea de ştergere a unei linii din vizualizarea *viz_galerie* determină apariţia erorii „*ORA-01752*: *cannot delete from view without exactly one key-preserved table*“. Dacă se încearcă modificarea sau inserarea unei linii prin intermediul unei vizualizări asupra căreia a fost definită o constrângere *WITH READ ONLY*, *server*-ul *Oracle* generează eroarea „*ORA-01733*: *virtual column not allowed here*“.

***Exemplu*:**

Să se creeze o vizualizare care conţine codul şi titlul operelor de artă, codul şi numele artiştilor care le-au creat, precum şi codul galeriilor unde sunt expuse. Să se afle dacă este posibilă adăugarea unei noi înregistrări prin intermediul acestei vizualizări.

```
CREATE VIEW opera_artist
AS SELECT o.cod_opera, o.titlu, o.cod_galerie,
          a.cod_artist, a.nume
   FROM   opera o, artist a
   WHERE  o.cod_artist = a.cod_artist;
```

Instrucţiunea următoare afişează numele coloanelor şi valorile *YES*/*NO*, după cum aceste coloane sunt, sau nu, modificabile.

```
SELECT COLUMN_NAME, UPDATABLE
FROM   USER_UPDATABLE_COLUMNS
WHERE  TABLE_NAME = 'OPERA_ARTIST';
```

Se va obţine că doar primele trei coloane ale vizualizării sunt modificabile.

Indexul primar al coloanei *cod_artist* din tabelul *artist* nu este unic în vizualizarea *opera_artist*. Prin urmare, tabelul *artist* nu este *key-preserved*, iar coloanele sale nu sunt modificabile.

Instrucţiunea următoare va genera eroarea „*ORA-01776*: *cannot modify more than one base table through a join view*“.

```
INSERT INTO opera_artist
VALUES     (200, 'Poeme de l''ame', 20, 147, 'Janmot');
```

În schimb, instrucţiunea următoare va fi executată cu succes, întrucât adaugă o înregistrare în tabelul de bază *opera*, ale cărui coloane sunt modificabile.

```
INSERT INTO opera_artist (cod_opera, titlu, cod_galerie)
VALUES (200, 'Poeme de l''ame', 20);
```

## Constrângeri asupra vizualizărilor

Începând cu versiunea *Oracle9i* pot fi specificate constrângeri pentru vizualizări. Se pot defini constrângeri la nivel de vizualizare, respectiv la nivel de coloană sau atribut. Constrângerile asupra vizualizărilor constituie o submulţime a constrângerilor specifice tabelelor.

Pot fi specificate explicit numai constrângerile *UNIQUE*, *PRIMARY KEY* şi *FOREIGN KEY*. Constrângerea de tip *CHECK* poate fi realizată prin precizarea clauzei *WITH CHECK OPTION* în comanda care defineşte vizualizarea.

Constrângerile asupra vizualizărilor pot fi definite numai în modul *DISABLE NOVALIDATE*. Aceste cuvinte cheie trebuie specificate la declararea constrângerii, nefiind permisă precizarea altor stări.

***Exemplu*:**

Să se creeze o vizualizare care conţine codurile, numele şi adresele galeriilor. Se va impune unicitatea valorilor coloanei *adresa* şi constrângerea de cheie primară pentru coloana corespunzătoare codului galeriei.

```
CREATE VIEW viz_galerie(
  cod_gal, nume, adresa UNIQUE DISABLE NOVALIDATE,
  CONSTRAINT cp_viz PRIMARY KEY (cod_gal) DISABLE NOVALIDATE)
AS SELECT cod_galerie, nume_galerie, adresa
   FROM galerie;
```

## Definirea vizualizărilor materializate

O vizualizare materializată, cunoscută în versiunile anterioare sub numele de clişeu (*snapshot*), este un obiect al schemei ce stochează rezultatele unei cereri şi care este folosit pentru a rezuma, calcula, replica şi distribui date.

Clauza *FROM* a cererii poate referi tabele, vizualizări sau alte vizualizări materializate. Luate în ansamblu, aceste obiecte sunt referite prin tabele *master* (în temeni de replicare) sau prin tabele detaliu (în termeni de *data warehouse*).

Optimizorul pe bază de costuri poate utiliza vizualizările materializate pentru a îmbunătăţi execuţia cererilor. Acesta recunoaşte automat situaţiile în care o astfel de vizualizare poate şi trebuie să fie utilizată pentru rezolvarea unei cereri. În urma unui asemenea demers, optimizorul rescrie cererea utilizând vizualizarea materializată.

În *data warehouse*, vizualizările materializate sunt utile pentru a calcula şi stoca date agregat, precum totaluri sau medii aritmetice. De asemenea, acest tip de vizualizare este utilizat pentru a efectua cereri în care intervin operaţii de compunere şi în care pot apărea agregări.

În mediile distribuite, vizualizările materializate sunt utilizate pentru replicarea datelor la *site*-uri distribuite şi sincronizarea modificărilor efectuate pe diferite *site*-uri. Astfel, vizualizările materializate permit accesul local la date care, altfel, ar fi trebuit să fie accesate de la locaţii distante.

Din anumite puncte de vedere, vizualizările materializate sunt similare indecşilor:

- consumă spaţiu de stocare;
- trebuie reactualizate dacă datele din tabelele de bază sunt modificate;
- îmbunătăţesc performanţa execuţiei instrucţiunilor *SQL* dacă sunt folosite pentru rescrierea cererilor;
- sunt transparente aplicaţiilor *SQL* şi utilizatorilor.

Spre deosebire de indecşi, vizualizările materializate pot fi accesate utilizând instrucţiuni *SELECT* şi pot fi actualizate prin instrucţiunile *INSERT*, *UPDATE*, *DELETE*.

Asupra unei vizualizări materializate se pot defini unul sau mai mulţi indecşi. O vizualizare materializată poate fi partiţionată. De asemenea, se pot defini vizualizări materializate asupra unui tabel partiţionat.

Similar vizualizărilor obişnuite, asupra celor materializate se pot defini constrângerile *PRIMARY KEY*, *UNIQUE* şi *FOREIGN KEY*. Singura stare validă a unei constrângeri este *DISABLE NOVALIDATE*.

Pentru compatibilitate cu versiunile anterioare, cuvintele cheie *SNAPSHOT* şi *MATERIALIZED VIEW* sunt echivalente.

***CREATE MATERIALIZED VIEW*** [*schema*.]*nume_viz_materializată*
[***OF*** [*schema*.]*tip_obiect*] [ (*constr_ref_domeniu*) ]
[***ORGANIZATION INDEX*** *clauza_tabel_org_index*]
[ {*proprietăţi_vm* | ***ON PREBUILT TABLE***
[{***WITH*** | ***WITHOUT***} ***REDUCED PRECISION***] } ]
[ {***USING INDEX***
[ {*clauza_atribute_fizice* | ***TABLESPACE*** *nume_sp_tabel*}
[ {*clauza_atribute_fizice* | ***TABLESPACE*** *nume_sp_tabel*} ] ...]
| ***USING NO INDEX***} ]
[*refresh_vm*] [***FOR UPDATE***]
[ {***DISABLE*** | ***ENABLE***} ***QUERY REWRITE***] ***AS*** *subcerere*;

Clauza *OF* permite crearea unei vizualizări materializate obiect.
Sintaxa clauzei *constr_ref_domeniu* este următoarea:

***SCOPE FOR*** ( {*ref_coloana* | *ref_atribut*} )
***IS*** [*schema*.]*nume_tabel_scope*
[, ***SCOPE FOR*** ( {*ref_coloana* | *ref_atribut*} )
***IS*** [*schema*.]*nume_tabel_scope*] ...

Clauza poate fi utilizată pentru restricţionarea domeniului referinţelor la tabelul *nume_tabel_scope*. Valorile dintr-o coloană de tip *REF* vor adresa obiecte din tabelul identificat prin *nume_tabel_scope*. În acest tabel sunt stocate instanţe de obiecte care au acelaşi tip ca şi coloana *REF*.

Opţiunea *ON PREBUILT TABLE* permite considerarea unui tabel existent ca fiind o vizualizare materializată predefinită. Tabelul trebuie să aibă acelaşi nume şi să se afle în aceeaşi schemă ca vizualizarea materializată rezultată. La ştergerea acestei vizualizări, tabelul revine la statutul său iniţial. Pentru o vizualizare materializată de acest tip, *alias*-urile de coloană din clauza *subcerere* trebuie să corespundă, ca număr şi tip de date, coloanelor din tabel.

Clauza *WITH REDUCED PRECISION* permite ca precizia coloanelor tabelului sau vizualizării materializate să nu coincidă cu precizia coloanelor returnate de *subcerere*. Pentru a impune respectarea întocmai a preciziei, sintaxa dispune de opţiunea *WITHOUT REDUCED PRECISION*, care este implicită.

Atributele fizice au o semantică asemănătoare celei descrise de *clauza_proprietăţi_fizice* din cadrul comenzii *CREATE TABLE*. Spre deosebire de tabele, pentru o vizualizare materializată nu poate fi specificată opţiunea *ORGANIZATION EXTERNAL*.

Clauza *TABLESPACE* specifică spaţiul tabel în care urmează să fie creată vizualizarea materializată. În absenţa acesteia, vizualizarea va fi creată în spaţiul tabel implicit al schemei care o conţine.

Clauza *USING INDEX* permite stabilirea de valori ale parametrilor *INITRANS*, *MAXTRANS* şi *STORAGE* ai indexului implicit care este utilizat de sistemul *Oracle* pentru a întreţine datele vizualizării materializate. Dacă este omisă clauza, sistemul va utiliza indexul implicit pentru ameliorarea vitezei de reactualizare incrementală a vizualizării materializate.

Clauza *proprietăţi_vm* este utilă pentru descrierea vizualizărilor materializate care nu se bazează pe un tabel existent (nu sunt construite cu opţiunea *ON PREBUILT TABLE*).

Dintre proprietăţile care pot fi specificate în această clauză, se menţionează: *clauza_partiţionare_tabel*, *CACHE* sau *NOCACHE*, *clauza_paralelism*. Pe lângă acestea, poate fi menţionată opţiunea *BUILD IMMEDIATE* | *DEFERRED* care determină introducerea de linii în vizualizarea materializată imediat, respectiv la prima operaţie de reactualizare (*refresh*). În acest ultim caz, până la prima operaţie de reactualizare, vizualizarea nu va putea fi utilizată în rescrierea cererilor. Opţiunea *IMMEDIATE* este implicită.

Prin *refresh_vm* se specifică metodele, modurile şi momentele la care sistemul va reactualiza vizualizarea materializată.

{***REFRESH***
 [ {***FAST*** | ***COMPLETE*** | ***FORCE***} ] [***ON*** {***DEMAND*** | ***COMMIT***} ]
 [***START WITH*** *data*] [***NEXT*** *data*]
 [ ***WITH*** {***PRIMARY KEY*** | ***ROWID***} ]
| ***USING***
 {***DEFAULT*** [ {***MASTER*** | ***LOCAL***} ] ***ROLLBACK SEGMENT***
 | [ {***MASTER*** | ***LOCAL***} ]
 ***ROLLBACK SEGMENT*** *nume_segm_anulare* }
 [ {***DEFAULT*** [ {***MASTER*** | ***LOCAL***} ] ***ROLLBACK SEGMENT***
 | [ {***MASTER*** | ***LOCAL***} ]
 ***ROLLBACK SEGMENT*** *nume_segm_anulare*}...] }
| ***NEVER REFRESH***}

Opţiunea *FAST* indică metoda de reactualizare incrementală, care se efectuează corespunzător modificărilor survenite în tabelele *master*. Modificările sunt stocate într-un fişier *log* asociat tabelului *master*. Clauza *COMPLETE* implică reactualizarea completă, care se realizează prin reexecutarea completă a cererii din definiţia vizualizării materializate. Clauza *FORCE* este implicită şi presupune reactualizarea de tip *FAST*, dacă este posibil. În caz contrar, reactualizarea va fi de tip *COMPLETE*.

Clauza *ON COMMIT* indică declanşarea unei operaţii de reactualizare de tip *FAST* ori de câte ori sistemul permanentizează o tranzacţie care operează asupra unui tabel *master* al vizualizării materializate. Clauza nu este permisă pentru vizualizările materializate ce conţin tipuri obiect.

Clauza *ON DEMAND* este implicită şi indică efectuarea reactualizării vizualizării materializate la cererea utilizatorului, prin intermediul procedurilor specifice din pachetul *DBMS_MVIEW* (*REFRESH*, *REFRESH_ALL_MVIEWS*, *REFRESH_DEPENDENT*).

Opţiunile *START WITH* şi *NEXT* nu pot fi specificate dacă s-au precizat clauzele *ON COMMIT* sau *ON DEMAND*. Expresiile de tip dată calendaristică indicate în cadrul acestor opţiuni specifică momentul primei reactualizări automate şi determină intervalul dintre două reactualizări automate consecutive.

Clauza *WITH PRIMARY KEY* este implicită şi permite ca tabelele *master* să fie reorganizate fără a afecta eligibilitatea vizualizării materializate pentru reactualizarea de tip *FAST*. Tabelul *master* trebuie să conţină o constrângere *PRIMARY KEY*. Opţiunea nu poate fi specificată pentru vizualizări materializate obiect. Opţiunea *WITH ROWID* asigură compatibilitatea cu tabelele *master* din versiunile precedente lui *Oracle8*.

Clauza *USING ROLLBACK SEGMENT* specifică segmentul de anulare distant care urmează să fie utilizat pentru reactualizarea vizualizării materializate. Cuvântul cheie *DEFAULT* determină ca sistemul să aleagă acest segment în mod automat. Opţiunile *MASTER* şi *LOCAL* specifică segmentul de anulare distant care urmează să fie utilizat pe *site*-ul distant pentru vizualizarea materializată individuală, respectiv pentru grupul local de reactualizare care conţine vizualizarea materializată. Opţiunea *LOCAL* este implicită.

Clauza *NEVER REFRESH* previne reactualizarea vizualizării materializate prin mecanisme *Oracle* sau prin proceduri. Pentru a permite reactualizarea, trebuie efectuată o operaţie *ALTER MATERIALIZED VIEW…REFRESH*.

Clauza *FOR UPDATE* permite actualizarea unei vizualizări materializate. *QUERY REWRITE* permite specificarea faptului că vizualizarea materializată este eligibilă pentru operaţia de rescriere a cererilor.

Opţiunea *AS* specifică cererea care defineşte vizualizarea materializată. Dacă în clauza *FROM* a cererii din definiţia vizualizării materializate se face referinţă la o altă vizualizare materializată, atunci aceasta va trebui reactualizată întotdeauna înaintea celei create în instrucţiunea curentă.

***Exemplu*:**

a) Să se creeze şi să se completeze cu înregistrări o vizualizare materializată care va conţine titlul operelor de artă, numele artistului şi suma valorilor poliţelor de asigurare încheiate.

Reactualizările ulterioare ale acestei vizualizări se vor realiza prin reexecutarea cererii din definiţie. Vizualizarea creată va putea fi aleasă pentru rescrierea cererilor.

```
CREATE MATERIALIZED VIEW opera_artist_polite
   BUILD IMMEDIATE
   REFRESH COMPLETE
   ENABLE QUERY REWRITE
   AS SELECT   o.titlu, a.nume, SUM(p.valoare) suma_polite
      FROM     opera o, artist a, polita_asig p
      WHERE    o.cod_artist = a.cod_artist
      AND      o.cod_opera = p.cod_opera
      GROUP BY o.cod_opera, o.titlu, a.nume;
```

b) Să se creeze tabelul *opera_artist_polite*. Acesta va fi utilizat ca tabel sumar preexistent în crearea unei vizualizări materializate ce va permite diferenţe de precizie şi rescrierea cererilor.

```
CREATE TABLE opera_artist_polite(
  titlu VARCHAR2(25),
  nume VARCHAR2(15),
  suma_polite NUMBER);

CREATE MATERIALIZED VIEW opera_artist_polite
   ON PREBUILT TABLE WITH REDUCED PRECISION
   ENABLE QUERY REWRITE
   AS SELECT   o.titlu, a.nume, SUM(p.valoare) suma_polite
      FROM     opera o, artist a, polita_asig p
      WHERE    o.cod_artist = a.cod_artist
      AND      o.cod_opera = p.cod_opera
      GROUP BY o.cod_opera, o.titlu, a.nume;
```

c) Să se creeze o vizualizare materializată care conţine informaţiile din tabelul *artist*, permite reorganizarea acestuia şi este reactualizată la momentul creării, iar apoi la fiecare 5 minute.

```
CREATE MATERIALIZED VIEW artist_vm
   REFRESH FAST START WITH SYSDATE NEXT SYSDATE + 1/288
   WITH PRIMARY KEY
   AS SELECT * FROM artist;
```

Pentru reactualizarea de tip *FAST*, este necesar un fişier *log* în care să fie stocate modificările. Instrucţiunea precedentă generează eroarea „*ORA-23413*: *table* "*artist*" *does not have a materialized view log*". Pentru remedierea acestei situaţii, înainte de crearea vizualizării se va lansa următoarea comandă:

```
CREATE MATERIALIZED VIEW LOG ON artist;
```

Comanda *ALTER MATERIALIZED VIEW* permite intervenţia asupra unei vizualizări materializate, într-unul din următoarele sensuri:

- modificarea caracteristicilor de stocare;
- modificarea metodei, modului sau timpului de reactualizare (*refresh*);
- modificarea structurii, astfel încât să devină un alt tip de vizualizare materializată;
- activarea sau dezactivarea funcţiei de rescriere a cererilor.

***ALTER MATERIALIZED VIEW*** [ *schema.*]*nume_viz_materializată*
[*atribute_fizice*] [***USING INDEX*** *atribute_fizice*]
[ {***REBUILD*** | *alter_vm_refresh*} ]
[ { {***ENABLE*** | ***DISABLE***} ***QUERY REWRITE***
| ***COMPILE*** | ***CONSIDER FRESH***} ];

Opţiunea *USING INDEX* modifică parametrii de stocare asociaţi indexului folosit de sistem pentru a întreţine datele vizualizării materializate.

Clauza *REBUILD* permite regenerarea operaţiilor de reactualizare atunci când se modifică un tip care este referit în vizualizarea materializată. Specificarea acestei opţiuni interzice utilizarea altor clauze în aceeaşi instrucţiune *ALTER MATERIALIZED VIEW*.

Clauza *alter_vm_refresh* permite modificarea metodelor, modurilor şi timpului implicit de reactualizare automată. În cazul modificării conţinutului tabelelor *master* ale vizualizării materializate, datele din vizualizare trebuie reactualizate astfel încât să reflecte datele existente.

Clauza *QUERY REWRITE*, prin opţiunile *ENABLE* şi *DISABLE*, determină ca vizualizarea materializată să fie, sau nu, eligibilă pentru rescrierea cererilor.

Clauza *COMPILE* permite revalidarea explicită a vizualizării materializate. Dacă un obiect de care depinde vizualizarea materializată este suprimat sau modificat, vizualizarea rămâne accesibilă, dar nu este eligibilă pentru rescrierea cererilor. Clauza este utilă pentru revalidarea explicită a vizualizării materializate, astfel încât aceasta să devină eligibilă în operaţia de rescriere a cererilor.

Opţiunea *CONSIDER FRESH* indică sistemului să considere vizualizarea materializată ca fiind reactualizată şi deci eligibilă pentru rescrierea cererilor.

***Exemplu*:**

Să se modifice vizualizarea materializată *opera_artist_polite* creată anterior, astfel încât metoda de reactualizare implicită să fie de tip *FAST*, iar intervalul de timp la care se realizează reactualizarea să fie de 7 zile. Nu va fi permisă utilizarea acestei vizualizări pentru rescrierea cererilor.

```
ALTER MATERIALIZED VIEW opera_artist_polite
   REFRESH FAST NEXT SYSDATE + 7 DISABLE QUERY REWRITE;
```

Pentru că nu se specifică valoarea corespunzătoare opţiunii *START WITH* în clauza *REFRESH*, următoarea reactualizare va avea loc la momentul stabilit prin comanda de creare a vizualizării materializate sau prin ultima comandă de modificare a acesteia. Sistemul va reactualiza vizualizarea evaluând expresia din clauza *NEXT*, iar apoi va executa această operaţie o dată pe săptămână.

***DROP MATERIALIZED VIEW*** [*schema*.]*nume_viz_materializată*;

# Grupări

*Cluster*-ul este o regrupare fizică a două sau mai multe tabele, relativ la una sau mai multe coloane, cu scopul măririi performanţelor. Coloanele comune definesc cheia *cluster*-ului.

Un *cluster* este un obiect al bazei care necesită:

- un nume unic la nivelul schemei,
- specificare a coloanelor care compun cheia cluster-ului,
- specificare a spaţiului de stocare (opţional),
- un index (relativ la cheia *cluster*-ului).

Un *cluster* trebuie să aibă cel puţin un index. Acest index trebuie creat înaintea oricărei comenzi *LMD* care va acţiona asupra tabelelor *cluster*-ului. Un index al *cluster*-ului se deosebeşte de un index al tabelului (de exemplu, absenţa indexului afectează utilizatorul – datele *cluster*-ului nu sunt accesibile).

Coloanele comune definite pentru *cluster*, reprezintă cheia *cluster*-ului şi criteriul de regrupare.

Liniile diferitelor tabele sunt regrupate în interiorul aceluiaşi bloc urmărind cheia *cluster*-ului. Dacă liniile asociate unei aceiaşi valori a cheii *cluster*-ului necesită un spaţiu de mai multe blocuri, atunci blocurile sunt înlănţuite.

Crearea unui *cluster* presupune:

- crearea structurii *cluster*-ului;
- crearea indexului *cluster*-ului;
- crearea tabelelor care vor compune *cluster*-ul.

Crearea unui *cluster* se realizeaza prin comanda:

*CREATE CLUSTER nume_cluster*
(*nume_coloana tip_data* [,*nume_coloana tip_data*] ...) [*SIZE n*]

Există două modalităţi pentru introducerea unui tabel într-un *cluster*.

- O primă variantă presupune că *cluster*-ul este creat pentru un tabel care deja există. De fapt, nu se poate asocia un *cluster* unui tabel care există!
- A doua variantă presupune că introducerea tabelului în *cluster* se face în momentul creării structurii tabelului (comanda *CREATE TABLE*).

***Exerciţiu*:**

Să se obţină un *cluster* referitor la lista cărţilor din fiecare domeniu.

Varianta 1

```
CREATE CLUSTER cdoml(cdom  CHAR(1));
CREATE INDEX indcom ON CLUSTER cdoml;
CREATE TABEL domino
         CLUSTER cdoml(coded)
         AS SELECT * FROM domeniu;
DROP TABEL domeniu;
RENAME domino TO domeniu;
ALTER TABLE  carte
MODIFY       coded NOT NULL;
CREATE TABEL carticica
         CLUSTER cdoml(coded)
         AS SELECT * FROM carte;
DROP TABLE carte;
RENAME carticica TO carte;
```

Varianta 2

```
CREATE CLUSTER cdoml(cdom  CHAR(1));
CREATE INDEX indcom ON CLUSTER cdoml;
-- crearea spatiului
CREATE TABLE domeniu
         (coded   CHAR(1) NOT NULL,
         intdom   CHAR() ... )
         CLUSTER  cdoml(coded);
CREATE TABLE carte
         (codel   CHAR(5) NOT NULL,
          …
            coded CHAR(1) NOT NULL)
         CLUSTER  cdoml(coded);
```

Pentru a scoate un tabel dintr-un *cluster* sunt parcurse următoarele etape: se creează un nou tabel, în afara *cluster*-ului, prin duplicarea celui vechi; se distruge tabelul din *cluster*; se suprimă *cluster*-ul.

```
CREATE TABLE alfa
    AS SELECT * FROM domeniu;
DROP TABLE domeniu;
RENAME alfa TO domeniu;
CREATE TABLE beta
    AS SELECT * FROM carte;
DROP TABLE carte;
RENAME beta TO carte;
DROP CLUSTER cdoml;
```

Un alt tip de cluster oferit de *Oracle* este *cluster*-ul *hash*. În acest caz, pentru a accesa o înregistrare, *cluster*-ul *hash* nu foloseşte un index, ci o funcţie numerică, numită funcţia *hash*. Funcţia are ca parametru cheia *cluster*-ului şi returnează o anumită valoare (valoare *hash*). Această valoare corespunde blocului de date din *cluster* pe care *Oracle* îl va citi sau scrie pe baza comenzii executate.

De exemplu, apelurile telefonice efectuate de un client într-o lună vor fi facturate împreună. Apelurile pot fi depozitate într-un *cluster hash* a cărui cheie este formată din coloanele ce conţin numărul telefonului, anul şi luna în care a avut loc convorbirea.

Suprimarea unui *cluster* din baza de date se face prin comanda:
*DROP CLUSTER nume_cluster*

În urma ştergerii unui *cluster*, tabelele pe care acesta le conţine nu mai sunt grupate. Secvenţa următoare suprimă: *cluster*-ul, toate tabelele definite relativ la acest *cluster* şi constrângerile lor de integritate.

```
DROP CLUSTER nume_cluster
    INCLUDING TABLES
    CASCADE CONSTRAINTS;
```

Modificarea unui *cluster* permite redefinirea condiţiilor, modificarea parametriilor de stocare şi a caracteristicelor de stare (*ALTER CLUSTER*).

# Informaţii despre obiectele bazei de date

Pot fi obţinute consultând DD. Dintre ele se remarcă:

- definiţiile tuturor obiectelor din baza de date;
- spaţiul alocat şi spaţiul utilizat în prezent de obiectele schemei;
- constrângerile de integritate;
- numele utilizatorilor bazei;
- privilegiile şi rolurile acordate fiecărui rol;
- alte informaţii generale despre baza de date.

Tabelul *USER_CATALOG* conţine informaţii despre tabelele şi vizualizările definite de un utilizator particular. Acest tabel poate fi referit şi prin sinonimul său public *CAT*.

Tabelul USER_OBJECTS conţine informaţii despre toate obiectele definite de utilizatorul curent. Tabelul are următoarea schemă relaţională:

*USER_OBJECTS* (*object_name*, *object_id*, *object_type*, *created*, *last_ddl_time*, *timestamp*, *status*)

Vizualizările cele mai importante ale dicţionarului datelor conţin:

- descrierea tabelelor definite de utilizatori (*USER_ALL_TABLES*),
- informaţii despre constrângerile definite de utilizator(*USER_CONSTRAINTS*),
- informaţii despre legăturile bazei de date (*USER_DB_LINKS*),
- erorile curente ale obiectelor depozitate (*USER_ERRORS*),
- informaţii despre indecşii creaţi de utilizator (*USER_INDEXES*),
- informaţii despre tabelele utilizatorului (*USER_TABLES*) etc.

Vizualizările din dicţionarul datelor referitoare la tabele conţin:

- *USER_TAB_COLUMNS*|*COLS* – informaţii despre coloanele tabelelor,
- *USER_CONS_COLUMNS* – informaţii despre constrângeri la nivel coloană,
- *USER_TAB_COMMENTS* – informaţii despre comentarii la nivel tabel,
- *USER_COL_COMMENTS* – informaţii despre comentarii la nivel coloană,
- *USER_TAB_PARTITIONS* – informaţii despre partiţiile tabelelor.