# Funcții SQL - CheatSheet

## 1. SELECT

***SELECT*** { [ {***DISTINCT*** | ***UNIQUE***} | ***ALL***] *lista_campuri* | *} 
***FROM*** [*nume_schemă.*]*nume_obiect* ] 
[, [*nume_schemă.*]*nume_obiect ...*] 
[***WHERE*** *condiţie_clauza_where*] 
[***START WITH*** *condiţie_clauza_start_with* 
***CONNECT BY*** *condiţie_clauza_connect_by*] 
[***GROUP BY*** *expresie* [*, expresie ...*] 
[***HAVING*** *condiţie_clauza_having*] ] 
[***ORDER BY*** {*expresie* | *poziţie*} [, {*expresie* | *poziţie*} ...] ] 
[***FOR UPDATE*** 
[***OF*** [ [*nume_schemă.*]*nume_obiect.*]*nume_coloană* 
[, [ [*nume_schemă.*]*nume_obiect.*]*nume_coloană*] ...] 
[***NOWAIT*** | ***WAIT*** *număr_întreg*] ];

## 2. Format dată calendaristică

Datele calendaristice pot fi formatate cu ajutorul funcţiei TO_CHAR(data, format), unde formatul poate fi alcătuit dintr-o combinaţie a următoarelor elemente

| **Element** | **Semnificaţie** |
|---|---|
| D | Numărul zilei din săptămâna (duminica=1; luni=2; …sâmbătă=6) |
| DD | Numărul zilei din lună. |
| DDD | Numărul zilei din an. |
| DY | Numele zilei din săptămână, printr-o abreviere de 3 litere (MON, THU etc.) |
| DAY | Numele zilei din săptămână, scris în întregime. |
| MM | Numărul lunii din an. |
| MON | Numele lunii din an, printr-o abreviere de 3 litere (JAN, FEB etc.) |
| MONTH | Numele lunii din an, scris în întregime. |
| Y | Ultima cifră din an |
| YY, YYY, YYYY | Ultimele 2, 3, respectiv 4 cifre din an. |
| YEAR | Anul, scris în litere (ex: two thousand four). |
| HH12, HH24 | Orele din zi, între 0-12, respectiv 0-24. |

| | |
|---|---|
| MI | Minutele din oră. |
| SS | Secundele din minut. |
| SSSSS | Secundele trecute de la miezul nopţii. |

# 3. Funcţiile single row

## 3.1 Funcţii de conversie

| **Funcţie** | **Descriere** | **Exemplu conversie** |
|---|---|---|
| *TO_CHAR* | converteşte (sau formatează) un număr sau o dată calendaristică în şir de caractere | *TO_CHAR(7) = ' 7'* *TO_CHAR(-7) = '-7' TO_CHAR (SYSDATE, 'DD/MM/YYYY') = ' 18/04/2007'* |
| *TO_DATE* | converteşte (sau formatează) un număr sau un şir de caractere în dată calendaristică | *TO_DATE('18-APR-2007','ddmon-yyyy')* |
| *TO_NUMBER* | converteşte (sau formatează) un şir de caractere în număr | *TO_NUMBER ('-25789', 'S99,999') = -25,789* |

## 3.2 Funcţii pentru prelucrarea caracterelor

| **Funcţie** | **Descriere** | **Exemplu** |
|---|---|---|
| *LENGTH(string)* | întoarce lungimea şirului de caractere *string* | *LENGTH('Informatica')=11* |
| *SUBSTR(string, start [,n])* | întoarce subşirul lui *string* care începe pe poziţia *start* şi are lungimea *n*; dacă *n* nu este specificat, subşirul se termină la sfârşitul lui *string*; | *SUBSTR('Informatica', 1, 4) = 'Info'* *SUBSTR('Informatica', 6) = 'matica'* *SUBSTR('Informatica', -5) = 'matica'* (ultimele 5 caractere) |
| *LTRIM(string [,'chars'])* | şterge din stânga şirului *string* orice caracter care apare în *chars*, până la găsirea primului caracter care nu este în *chars*; în cazul în care *chars* nu este specificat, se şterg spaţiile libere din stânga lui *string*; | *LTRIM ('  info') = 'info'* |
| *RTRIM(string [,'chars'])* | este similar funcţiei *LTRIM*, cu excepţia faptului că ştergerea se face la dreapta şirului de caractere; | *RTRIM ('infoXXXX', 'X') = 'info'* |

| | | |
|---|---|---|
| *TRIM (LEADING \| TRAILING \| BOTH chars FROM expresie)* | elimină caracterele specificate (*chars*) de la începutul (*leading*) , sfârşitul (*trailing*) sau din ambele părţi, dintr-o expresie caracter dată. | *TRIM (LEADING 'X' FROM 'XXXInfoXXX') = 'InfoXXX'*<br>*TRIM (TRAILING 'X' FROM 'XXXInfoXXX') = 'XXXInfo'*<br>*TRIM ( BOTH 'X' FROM 'XXXInfoXXX') = 'Info'*<br>*TRIM ( BOTH FROM ' Info ') = 'Info'* |
| *LPAD(string, length [,'chars'])* | adaugă *chars* la stânga şirului de caractere *string* până când lungimea noului şir devine *length*; în cazul în care *chars* nu este specificat, atunci se adaugă spaţii libere la stânga lui *string*; | *LPAD (LOWER('iNfO'),6) = ' info'* |
| *RPAD(string, length [,'chars'])* | este similar funcţiei *LPAD*, dar adăugarea de caractere se face la dreapta şirului; | *RPAD (LOWER('InfO'), 6, 'X') = 'infoXX'* |
| *REPLACE(string1, string2 [,string3])* | întoarce *string1* cu toate apariţiile lui *string2* înlocuite prin *string3*; dacă *string3* nu este specificat, atunci toate apariţiile lui *string2* sunt şterse; | *REPLACE ('$b$bb','$','a') = 'ababb'*<br>*REPLACE ('$b$bb','$b','ad') = 'adadb'*<br>*REPLACE ('$a$aa','$') = 'aaa'* |
| *UPPER(string), LOWER(string)* | transformă toate literele şirului de caractere *string* în majuscule, respectiv minuscule; | *LOWER ('InFo') = 'info' UPPER ('iNfO') = 'INFO'* |
| *INITCAP(string)* | transformă primul caracter al şirului în majusculă, restul caracterelor fiind transformate în minuscule | *INITCAP ('iNfO') = 'Info'* |
| *INSTR(string, 'chars' [,start [,n]])* | caută în *string*, începând de de la poziţia *start*, a *n*-a apariţie a secvenţei *chars* şi întoarce poziţia respectivă; dacă *start* nu este specificat, căutarea se face de la începutul şirului; dacă *n* nu este specificat, se caută prima apariţie a secvenţei *chars*; | *INSTR (LOWER('AbC aBcDe'), 'ab', 5, 2) = 0*<br>*INSTR (LOWER('AbCdE aBcDe'), 'ab', 5) = 7* |
| *ASCII(char)* | furnizează codul *ASCII* al primului caracter al unui şir | *ASCII ('alfa') = ASCII ('a') = 97* |
| *CHR(num)* | întoarce caracterul corespunzător codului *ASCII* specificat | *CHR(97)= 'a'* |
| *CONCAT(string1, string2)* | realizează concatenarea a două şiruri de caractere | *CONCAT ('In', 'fo') = 'Info'* |
| *TRANSLATE(string, source, destination)* | fiecare caracter care apare în şirurile de caractere *string* şi *source* este transformat în caracterul corespunzător (aflat pe aceeaşi poziţie ca şi în *source*) din şirul de caractere *destination* | *TRANSLATE('$a$aa','$','b') = 'babaa'*<br>*TRANSLATE('$a$aaa','$a','bc') = 'bcbccc'* |

## 3.3 Fucţii pentru prelucrarea datelor calendaristice

| Funcţie | Descriere | Exemplu |
|---|---|---|
| *SYSDATE* | întoarce data şi timpul curent | *SELECT SYSDATE FROM dual;* (de revăzut utilizarea acestei funcţii împreună cu *TO_CHAR* în cadrul laboratorului 1) |
| *ADD_MONTHS ( expr_date, nr_luni)* | întoarce data care este după *nr_luni* luni de la data *expr_date*; | *ADD_MONTHS('02-APR-2007', 3)* = '02-JUL-2007'. |
| *NEXT_DAY(expr_date, day)* | întoarce următoarea dată după data *expr_date*, a cărei zi a săptămânii este cea specificată prin şirul de caractere *day* | *NEXT_DAY('18-APR-2007', 'Monday') = '23-APR-2007'* |
| *LAST_DAY(expr_date)* | întoarce data corespunzătoare ultimei zile a lunii din care data *expr_date* face parte | *LAST_DAY('02-DEC-2007') = '31-DEC-2007'* |
| *MONTHS_BETWEEN (expr_date2, expr_date1)* | întoarce numărul de luni dintre cele două date calendaristice specificate. Data cea mai recentă trebuie specificată în primul argument, altfel rezultatul este negativ. | *MONTHS_BETWEEN('02-DEC-2005', '10-OCT-2002') = 37.7419355*<br>*MONTHS_BETWEEN('10-OCT-2002', '02-DEC-2005') = -37.7419355* |
| *TRUNC(expr_date)* | întoarce data *expr_date*, dar cu timpul setat la ora 12:00 AM (miezul nopţii) | *TO_CHAR(TRUNC (SYSDATE), 'dd/mm/yy HH24:MI') = '02/12/05 00:00'* |
| *ROUND(expr_date)* | dacă data *expr_date* este înainte de miezul zilei, întoarce data *d* cu timpul setat la ora 12:00 AM; altfel, este returnată data corespunzătoare zilei următoare, cu timpul setat la ora 12:00 AM | TO_CHAR(ROUND (SYSDATE*), 'dd/mm/yy hh24:mi am') = '03/12/05 00:00 AM'* |
| *LEAST(d1, d2, …, dn), GREATEST(d1, d2, …, dn)* | dintr-o listă de date calendaristice, funcţiile întorc prima, respectiv ultima dată în ordine cronologică | *LEAST(SYSDATE, SYSDATE + 3, SYSDATE – 5) = SYSDATE-5*<br>*GREATEST(SYSDATE, SYSDATE + 3, SYSDATE – 5) = SYSDATE + 3* |

## 3.4 Operații asupra datelor calendaristice

| **Operaţie** | **Tipul de date al rezultatului** | **Descriere** |
|---|---|---|
| *expr_date -/+ expr_number* | *Date* | Scade/adună un număr de zile dintr-o / la o dată. Numărul de zile poate sa nu fie întreg (putem adăuga, de exemplu, un număr de minute sau de ore). |
| *expr_date1 – expr_date2* | *Number* | Intoarce numărul de zile dintre două date calendaristice. Data *expr_date1* trebuie să fie mai recentă decât *expr_date2*, altfel rezultatul este negativ. |

## 3.5 Funcții diverse

| **Funcţie** | **Descriere** | **Exemplu** |
|---|---|---|
| *DECODE(value, if1, then1, if2, then2, … , ifN, thenN, else)* | returnează *then1* dacă *value* este egală cu *if1*, *then2* dacă value este egală cu *if2* etc.; dacă *value* nu este egală cu nici una din valorile *if*, atunci funcţia întoarce valoarea *else*; | *DECODE (‘a’, ‘a’, ‘b’, ‘c’) = ‘b’*<br>*DECODE (‘b’, ‘a’, ‘b’, ‘c’) = ‘c’*<br>*DECODE (‘c’, ‘a’, ‘b’, ‘c’) = ‘c’* |
| *NVL(expr_1, expr_2)* | dacă *expr_1* este *NULL*, întoarce *expr_2*; altfel, întoarce *expr_1*.<br>Tipurile celor două expresii trebuie să fie compatibile sau *expr_2* să poată fi convertit implicit la *expr_1* | *NVL(NULL, 1) = 1*<br>*NVL(2, 1) = 2*<br>*NVL('a', 1) = 'a' -- conversie implicită*<br>*NVL(1, 'a') -- eroare --nu are loc conversia implicită* |
| *NVL2(expr_1, expr_2, expr_3)* | dacă *expr_1* este *NOT NULL*, întoarce *expr_2*, altfel întoarce *expr_3* | *NVL2(1, 2, 3) = 2*<br>*NVL2 (NULL, 1, 2) = 2* |
| *NULLIF (expr_1, expr_2)* | Daca *expr_1 = expr_2* atunci funcţia returnează *NULL*, altfel returnează expresia *expr_1*.<br>Echivalent cu<br>*CASE WHEN expr1 = expr2 THEN NULL ELSE expr1 END* | *NULLIF (1, 2) = 1*<br>*NULLIF (1,1) = NULL* |
| *COALESCE (expr_1, expr_2, ... , expr_n)* | Returnează prima expresie NOT NULL din lista de argumente. | *COALESCE (NULL, NULL, 1, 2, NULL) = 1* |
| *UID, USER* | întorc *ID*-ul, respectiv *username*-ul utilizatorului *ORACLE* curent | *SELECT USER FROM dual;* |
| *VSIZE(expr)* | întoarce numărul de octeți ai unei expresii de tip *DATE, NUMBER* sau *VARCHAR2* | *SELECT VSIZE(salary) FROM employees WHERE employee_id=200;* |

### 3.6 Funcţii aritmetice

Unei singure valori, şi aceste funcţii sunt: *ABS* (valoarea absolută), *CEIL* (partea întreagă superioară), *FLOOR* (partea întreagă inferioară), *ROUND* (rotunjire cu un număr specificat de zecimale), *TRUNC* (trunchiere cu un număr specificat de zecimale), *EXP* (ridicarea la putere a lui *e*), *LN* (logaritm natural), *LOG* (logaritm într-o bază specificată), *MOD* (restul împărţirii a două numere specificate), *POWER* (ridicarea la putere), *SIGN* (semnul unui număr), *COS* (cosinus), *COSH* (cosinus hiperbolic), *SIN* (sinus), *SINH* (sinus hiperbolic), *SQRT* (rădăcina pătrată), *TAN* (tangent), *TANH* (tangent hiperbolic);

Unei liste de valori, iar acestea sunt funcţiile *LEAST* şi *GREATEST*, care întorc cea mai mică, respectiv cea mai mare valoare a unei liste de expresii.

## 4. JOIN

Pentru *join*, sistemul *Oracle* oferă şi o sintaxă specifică, introdusă de către standardul *SQL3* (*SQL:1999*). Această sintaxă nu aduce beneficii în privinţa performanţei faţă de *join*-urile care se specifică în clauza *WHERE*. Tipurile de *join* conforme cu *SQL3* sunt definite prin cuvintele cheie *CROSS JOIN* (pentru produs cartezian), *NATURAL JOIN*, *FULL OUTER JOIN*, clauzele *USING* şi *ON*.

Sintaxa corespunzătoare standardului *SQL3* este următoarea:

***SELECT*** *tabel_1.nume_coloană*, *tabel_2.nume_coloană*
***FROM*** *tabel_1*
[***CROSS JOIN*** *tabel_2*]
| [***NATURAL JOIN*** *tabel_2*]
| [***JOIN*** *tabel_2* ***USING*** (*nume_coloană*) ]
| [***JOIN*** *tabel_2* ***ON*** (*tabel_1.nume_coloană = tabel_2.nume_coloană*) ]
| [***LEFT*** | ***RIGHT*** | ***FULL OUTER JOIN*** *tabel_2*
***ON*** (*tabel_1.nume_coloană = tabel_2.nume_coloană*) ];

***NATURAL JOIN*** presupune existenţa unor coloane având acelaşi nume în ambele tabele. Clauza determină selectarea liniilor din cele două tabele, care au valori egale în aceste coloane. Dacă tipurile de date ale coloanelor cu nume identice sunt diferite, va fi returnată o eroare.

Coloanele având acelaşi nume în cele două tabele trebuie să nu fie precedate de numele sau *alias*-ul tabelului corespunzător.

***JOIN tabel_2 USING nume_coloană*** efectuează un *equijoin* pe baza coloanei cu numele specificat în sintaxă. Această clauză este utilă dacă există coloane având acelaşi nume, dar tipuri de date diferite. Coloanele referite în clauza *USING* ***trebuie să nu conţină calificatori*** (să nu fie precedate de nume de tabele sau *alias*-uri) în nici o apariţie a lor în instrucţiunea *SQL*. Clauzele *NATURAL JOIN* şi *USING* nu pot coexista în aceeaşi instrucţiune *SQL*.

***JOIN tabel_2 ON tabel_1.nume_coloană = tabel_2.nume_coloană*** efectuează un *equijoin* pe baza condiţiei exprimate în clauza *ON*. Această clauză permite specificarea separată a condiţiilor de *join*, respectiv a celor de căutare sau filtrare (din clauza *WHERE*).

***LEFT, RIGHT*** şi ***FULL OUTER JOIN tabel_2 ON (tabel_1.nume_coloană = tabel_2.nume_coloană)*** efectuează *outer join* la stânga, dreapta, respectiv în ambele părţi pe baza condiţiei exprimate în clauza *ON*.

Un *join* care returnează rezultatele unui *inner join*, dar şi cele ale *outer join*-urilor la stânga şi la dreapta se numeşte *full outer join*.

# 5. Operatorii pe mulțimi

Operatorii pe mulţimi combină rezultatele obţinute din două sau mai multe interogări. Cererile care conţin operatori pe mulţimi se numesc ***cereri compuse***. Există patru operatori pe mulţimi: *UNION*, *UNION ALL*, *INTERSECT* şi *MINUS*.

Toţi operatorii pe mulţimi au aceeaşi precedenţă. Dacă o instrucţiune *SQL* conţine mai mulţi operatori pe mulţimi, *server*-ul *Oracle* evaluează cererea de la stânga la dreapta (sau de sus în jos). Pentru a schimba această ordine de evaluare, se pot utiliza paranteze.

- ***Operatorul UNION*** returnează toate liniile selectate de două cereri, eliminând duplicatele. Acest operator nu ignoră valorile *null* şi are precedenţă mai mică decât operatorul *IN*.
- Operatorul ***UNION ALL*** returnează toate liniile selectate de două cereri, fără a elimina duplicatele. Precizările făcute asupra operatorului *UNION* sunt valabile şi în cazul operatorului *UNION ALL*. În cererile asupra cărora se aplică *UNION ALL* nu poate fi utilizat cuvântul cheie *DISTINCT*.
- Operatorul ***INTERSECT*** returnează toate liniile comune cererilor asupra cărora se aplică. Acest operator nu ignoră valorile *null*.
- Operatorul ***MINUS*** determină liniile returnate de prima cerere care nu apar în rezultatul celei de-a doua cereri. Pentru ca operatorul *MINUS* să funcţioneze, este necesar ca toate coloanele din clauza *WHERE* să se afle şi în clauza *SELECT*.

# 6. Subcereri

Prin intermediul subcererilor se pot construi interogări complexe pe baza unor instrucţiuni simple.

O subcerere (subinterogare) este o comandă *SELECT* integrată într-o clauză a altei instrucţiuni *SQL*, numită instrucţiune „părinte“ sau instrucţiune exterioară. Subcererile mai sunt numite instrucţiuni *SELECT* imbricate sau interioare.

Rezultatele subcererii sunt utilizate în cadrul cererii exterioare, pentru a determina rezultatul final. În funcţie de modul de evaluare a subcererii în raport cu cererea exterioară, subcererile pot fi:

- nesincronizate (necorelate) sau
- sincronizate (corelate).

Prima clasă de subcereri este evaluată dinspre interior către exterior, adică interogarea externă acţionează pe baza rezultatului cererii interne. Al doilea tip de subcerere este evaluat invers, adică interogarea externă furnizează valori cererii interne, iar rezultatele subcererii sunt transferate cererii externe.

Subcererile nesincronizate care apar în clauza *WHERE* a unei interogări sunt de forma următoare:

***SELECT*** *expresie1, expresie2, ...* ***FROM*** *nume_tabel1*

***WHERE*** *expresie_condiţie operator* (***SELECT*** *expresie*

***FROM*** *nume_tabel2*);

- cererea internă este executată prima şi determină o valoare (sau o mulţime de valori);
- cererea externă se execută o singură dată, utilizând valorile returnate de cererea internă.

Subcererile sincronizate care apar în clauza *WHERE* a unei interogări au următoarea formă generală:

***SELECT*** *expresie_ext_1*[, *expresie_ext_2 ...*] ***FROM*** *nume_tabel_1 extern*

***WHERE*** *expresie_condiţie operator*

(***SELECT*** *expresie*
***FROM*** *nume_tabel_2*

***WHERE*** *expresie = extern.expresie_ext*);

- cererea externă determină o linie candidat;
- cererea internă este executată utilizând valoarea liniei candidat;
- valorile rezultate din cererea internă sunt utilizate pentru calificarea sau descalificarea liniei candidat;
- paşii precedenţi se repetă până când nu mai există linii candidat.

***Obs:*** *operator* poate fi:

- *single-row operator* (>, =, >=, <, <>, <=), care poate fi utilizat dacă subcererea returnează o singură linie;
- *multiple-row operator* (*IN*, *ANY*, *ALL*), care poate fi folosit dacă subcererea returnează mai mult de o linie.

Operatorul *NOT* poate fi utilizat în combinaţie cu *IN*, *ANY* şi *ALL*.

Cuvintele cheie *ANY* şi *ALL* pot fi utilizate cu subcererile care produc o singură coloană de valori. Dacă subcererea este precedată de către cuvântul cheie *ALL*, atunci condiţia va fi adevărată numai dacă este satisfăcută de către toate valorile produse de subcerere. Astfel, <*ALL* are semnificaţia „mai mic decât minimul“, iar >*ALL* este echivalent cu „mai mare decât maximul“. Dacă subcererea este precedată de către cuvântul cheie *ANY*, condiţia va fi adevărată dacă este satisfăcută de către oricare (una sau mai multe) dintre valorile produse de subcerere. În comparaţii, <*ANY* are semnificaţia „mai mic decât maximul“; >*ANY* înseamnă „mai mare decât minimul“; =*ANY* este echivalent cu operatorul *IN*.

Dacă subcererea returnează mulţimea vidă, atunci condiţia *ALL* va returna valoarea *true*, iar condiţia *ANY* va returna valoarea *false*. Standardul *ISO* permite utilizarea cuvântului cheie *SOME*, în locul lui *ANY*.

# 7. Funcţii grup 1

## 7.1 GROUP BY

Clauza *GROUP BY* este utilizată pentru a diviza liniile unui tabel în grupuri. Pentru a returna informaţia corespunzătoare fiecărui astfel de grup, pot fi utilizate funcţiile agregat. Ele pot apărea în clauzele: o *SELECT* o *ORDER BY* o *HAVING*.
*Server*-ul *Oracle* aplică aceste funcţii fiecărui grup de linii şi returnează un singur rezultat pentru fiecare mulţime.

Dintre funcţiile grup definite în sistemul *Oracle,* se pot enumera: *AVG, SUM, MAX, MIN, COUNT, STDDEV, VARIANCE* etc. Tipurile de date ale argumentelor funcţiilor grup pot fi *CHAR, VARCHAR2, NUMBER* sau *DATE*.

- Funcţiile *AVG, SUM, STDDEV* şi *VARIANCE* operează numai asupra valorilor numerice.
- Funcţiile *MAX* şi *MIN* pot opera asupra valorilor numerice, caracter sau dată calendaristică.

Absenţa clauzei GROUP BY conduce la aplicarea funcţiei grup pe mulţimea tuturor liniilor tabelului.

Toate funcţiile grup, cu excepţia lui *COUNT*(*), ignoră valorile *null*. *COUNT*(*expresie*) returnează numărul de linii pentru care expresia dată nu are valoarea *null*. Funcţia *COUNT* returnează un număr mai mare sau egal cu zero şi nu întoarce niciodată valoarea *null*.

Când este utilizată clauza *GROUP BY, server*-ul sortează implicit mulţimea rezultată în ordinea crescătoare a valorilor coloanelor după care se realizează gruparea.

În clauza *GROUP BY* a unei cereri se pot utiliza operatorii *ROLLUP* şi *CUBE*. Aceşti operatori sunt disponibili începând cu versiunea *Oracle8i*.

Expresiile din clauza *SELECT* a unei cereri care conţine opţiunea *GROUP BY* trebuie să reprezinte o proprietate unică de grup, adică fie un atribut de grupare, fie o funcţie de agregare aplicată tuplurilor unui grup, fie o expresie formată pe baza primelor două. Toate expresiile din clauza *SELECT*, cu excepţia funcţiilor de agregare, se trec în clauza *GROUP BY* (unde pot apărea cel mult 255 expresii).

## 7.2 HAVING

Opţiunea *HAVING* permite restricţionarea grupurilor de linii returnate, la cele care îndeplinesc o anumită condiţie.
Dacă această clauză este folosită în absenţa unei clauze *GROUP BY*, aceasta presupune că gruparea se aplică întregului tabel, deci este returnată o singură linie, care este reţinută în rezultat doar dacă este îndeplinită condiţia din clauza *HAVING*.

# 8. Operatori

## 8.1 Operatorul ROLLUP

Acest operator furnizează valori agregat şi superagregat corespunzătoare expresiilor din clauza *GROUP BY*. Operatorul *ROLLUP* poate fi folosit pentru extragerea de statistici şi informaţii totalizatoare din mulţimile rezultate. Acest operator poate fi util la generarea de rapoarte, diagrame şi grafice.

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

Dacă în clauza *GROUP BY* sunt specificate *n* coloane, pentru a produce subtotaluri fără operatorul *ROLLUP* ar fi necesare *n* + 1 instrucţiuni *SELECT* conectate prin *UNION ALL*. Aceasta ar face execuţia cererii ineficientă pentru că fiecare instrucţiune *SELECT* determină accesarea tabelului. Operatorul *ROLLUP* determină rezultatele efectuând un singur acces la tabel şi este util atunci când sunt implicate multe coloane în producerea subtotalurilor. Ilustrăm aplicarea acestui operator prin urmatorul exemplu.

## 8.2 Operatorul CUBE

Operatorul *CUBE* grupează liniile selectate pe baza valorilor tuturor combinaţiilor posibile ale expresiilor specificate şi returnează câte o linie totalizatoare pentru fiecare grup. Acest operator este folosit pentru a produce mulţimi de rezultate care sunt utilizate în rapoarte. În vreme ce *ROLLUP* produce subtotalurile doar pentru o parte dintre combinaţiile posibile, operatorul *CUBE* produce subtotaluri pentru toate combinaţiile posibile de grupări specificate în clauza *GROUP BY*, precum şi un total general.

Dacă există *n* coloane sau expresii în clauza *GROUP BY*, vor exista $2^n$ combinaţii posibile superagregat. Din punct de vedere matematic, aceste combinaţii formează un cub *n*-dimensional, de aici provenind numele operatorului. Pentru producerea de subtotaluri fără ajutorul operatorului *CUBE* ar fi necesare $2^n$ instrucţiuni *SELECT* conectate prin *UNION ALL*.

# 9. Funcţii grup 2

## 9.1 GROUP BY ROLLUP

***GROUP BY ROLLUP*** (expr_1, expr_2, …, expr_n) generează ***n+1 tipuri de linii***, corespunzătoare următoarelor grupări:

- GROUP BY (expr_1, expr_2, …, expr_n-1, expr_n)
- GROUP BY (expr_1, expr_2, …, expr_n-1)
- …
- GROUP BY (expr_1, expr_2) o GROUP BY (expr_1)
- GROUP BY () – corespunzător absenţei clauzei GROUP BY şi deci, calculului funcţiilor grup din cerere pentru întreg tabelul.

***Obs:***

- Lista de expresii care urmează operatorului ROLLUP este parcursă de la dreapta la stânga, suprimându-se câte o expresie .
- O cerere în care apare un astfel de operator este echivalentă cu reuniunea (UNION ALL) a n+1 cereri.

## 9.2 GROUP BY CUBE

***GROUP BY CUBE*** (expr_1, expr_2, …, expr_n) generează $2^n$ tipuri de linii, corespunzătoare tuturor combinaţiilor posibile de expresii din lista.

## 9.3 GROUPING

Pentru determinarea modului în care a fost obţinută o valoare totalizatoare cu ROLLUP sau CUBE, se utilizează funcţia:

***GROUPING***(expresie)

Aceasta întoarce:

- valoarea 0, dacă expresia a fost utilizată pentru calculul valorii agregat
- valoarea 1, dacă expresia nu a fost utilizată.

## 9.4 GROUPING SETS

Dacă se doreşte obţinerea numai a anumitor grupări superagregat, acestea pot fi precizate prin intermediul clauzei :

***GROUPING SETS*** ((expr_11, expr_12, …, expr_1n), (expr_21,xpr_22, …expr_2m), …)

# 10. Subcereri corelate

O subcerere (cerere imbricată sau încuibărită) corelată poate avea forma următoare:

***SELECT*** *nume_coloană_1*[, *nume_coloană_2 ...*]
***FROM*** *nume_tabel_1 extern*
***WHERE*** *expresie operator*
(***SELECT*** *nume_coloană_1* [, *nume_coloană_2 ...*]
***FROM*** *nume_tabel_2*
***WHERE*** *expresie_1 = extern.expresie_2*);

Modul de execuţie este următorul :

- cererea externă determină o linie candidat;
- cererea internă este executată utilizând valoarea liniei candidat;
- valorile rezultate din cererea internă sunt utilizate pentru calificarea sau descalificarea liniei candidat;
- paşii precedenţi se repetă până când nu mai există linii candidat. ***Obs:*** *operator* poate fi:
- *single-row operator* (>, =, >=, <, <>, <=), care poate fi utilizat dacă subcererea returnează o singură linie;
- *multiple-row operator* (*IN*, *ANY*, *ALL*), care poate fi folosit dacă subcererea returnează mai mult de o linie.

***Obs:*** O subcerere (corelată sau necorelată) poate apărea în clauzele:

- SELECT
- FROM (vezi laboratorul 4)
- WHERE
- HAVING (vezi laboratorul 4)
- START WITH (vezi mai jos – la cereri ierarhice)

## 10.1 Operatorul EXISTS

În instrucţiunile *SELECT* imbricate, este permisă utilizarea oricărui operator logic.

Pentru a testa dacă valoarea recuperată de cererea externă există în mulţimea valorilor regăsite de cererea internă corelată, se poate utiliza operatorul *EXISTS*.

Dacă subcererea returnează cel puţin o linie, operatorul returnează valoarea *TRUE*. În caz contrar, va fi returnată valoarea *FALSE*.

Operatorul *EXISTS* asigură că nu mai este continuată căutarea în cererea internă după ce aceasta regăseşte o linie.

# 11. Subcereri ierarhice

## 11.1 START WITH, CONNECT BY

Clauzele ***START WITH*** şi ***CONNECT BY*** se utilizează în formularea cererilor ierarhice. *START WITH* specifică o condiţie care identifică liniile ce urmează să fie considerate ca rădăcini ale cererii ierarhice respective. Dacă se omite această clauză, sistemul *Oracle* utilizează toate liniile din tabel drept linii rădăcină.

*CONNECT BY* specifică o condiţie care identifică relaţia dintre liniile „părinte“ şi „copil“ ale ierarhiei. Condiţia trebuie să conţină operatorul *PRIOR* pentru a face referinţă la linia „părinte“.

Operatorul *PRIOR* face referinţă la linia „părinte“. Plasarea acestui operator determină direcţia interogării, dinspre „părinte“ spre „copil“ (*top-down*) sau invers (*bottom-up*). Traversarea *top-down*, respectiv *bottom-up* a arborelui se realizează prin specificări de forma următoare:

*Top-down:* ***CONNECT BY PRIOR*** *cheie_parinte = cheie_copil*;
*Bottom-up:* ***CONNECT BY PRIOR*** *cheie_copil = cheie_parinte*;

***Obs:*** Operatorul *PRIOR* poate fi plasat în faţa oricărui membru al condiţiei specificate în clauza *CONNECT BY*.

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

## 11.2 LEVEL

Pseudocoloana ***LEVEL*** poate fi utilă într-o cerere ierarhică. Aceasta determină lungimea drumului de la rădăcină la un nod.

# 12. Clauza WITH

Cu ajutorul clauzei *WITH* se poate defini un bloc de cerere înainte ca acesta să fie utilizat într-o interogare.

Clauza permite reutilizarea aceluiaşi bloc de cerere într-o instrucţiune *SELECT* complexă. Acest lucru este util atunci când o cerere face referinţă de mai multe ori la acelaşi bloc de cerere, care conţine operaţii *join* şi funcţii agregat.

# 13. Analiza Top – n

Pentru aflarea primelor n rezultate ale unei cereri, este utilă pseudocoloana *ROWNUM*. Aceasta returnează numărul de ordine al unei linii în rezultat.

# 14. Operatorul DIVISION

Diviziunea este o operație binară care definește o relație ce conține valorile atributelor dintr-o relație care apar **în toate** valorile atributelor din cealaltă relație.

Operatorul *DIVISION* este legat de cuantificatorul universal (∀) care nu există în *SQL*. Cuantificatorul universal poate fi însă simulat cu ajutorul cuantificatorului existențial (∃) utilizând relația:

$$\forall x\ P(x) \equiv \neg\ \exists\ x\ \neg\ P(x).$$

Prin urmare, operatorul *DIVISION* poate fi exprimat în *SQL* prin succesiunea a doi operatori *NOT EXISTS*. Alte modalități de implementare a acestui operator vor fi prezentate în exemplul de mai jos.

Extindem diagrama *HR* cu o nouă entitate, *PROJECT*, şi o nouă asociere: „angajat lucreaza în cadrul unui proiect”, între entitățile *EMPLOYEES* şi *PROJECT*. Aceasta este o relație *many-to-many*, care va conduce la apariția unui tabel asociativ, numit *WORKS_ON*.

O altă asociere între entitățile *EMPLOYEES* şi *PROJECT* este „angajat conduce proiect”. Aceasta este o relație *one-to-many*.

Noile tabele au următoarele scheme relaționale:

- *PROJECT*(project_id#, project_name, budget, start_date, deadline, delivery_date, project_manager) - project_id reprezintă codul proiectului şi este cheia primară a relației *PROJECT*
  - project_name reprezintă numele proiectului
  - budget este bugetul alocat proiectului
  - start_date este data demarării proiectului

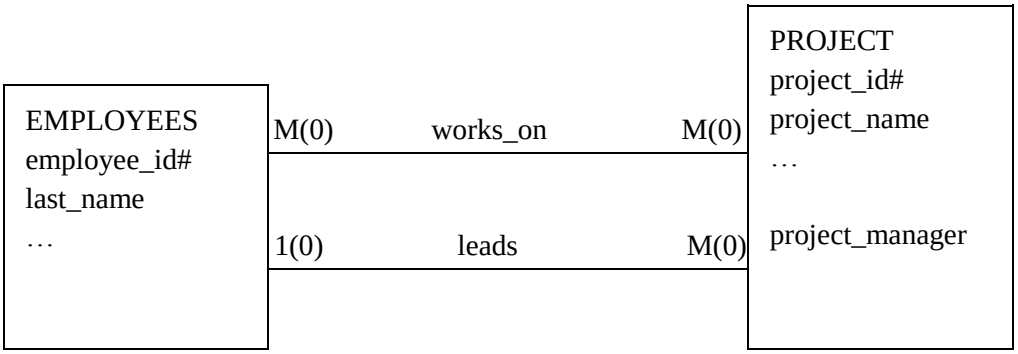
- deadline reprezintă data la care proiectul trebuie să fie finalizat
- delivery_date este data la care proiectul este livrat efectiv
- project_manager reprezintă codul managerului de proiect şi este cheie externă. Pe cine referă această coloană ? Ce relație implementează această cheie externă?
-

2) *WORKS_ON*(project_id#, employee_id#, start_date, end_date)

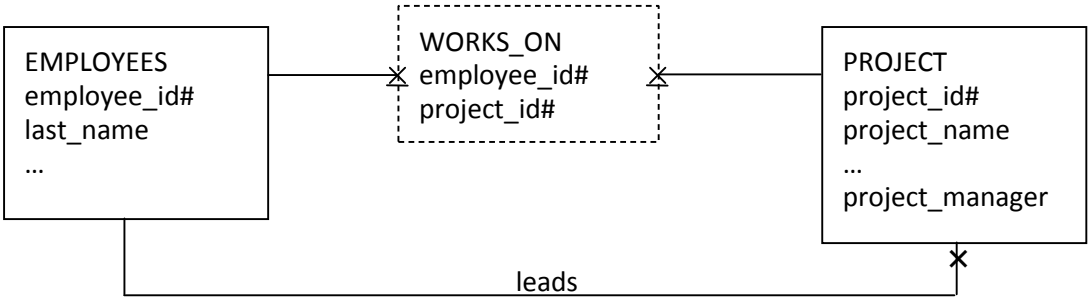
- cheia primară a relației este compusă din atributele employee_id şi project_id.

Scriptul pentru crearea noilor tabele şi inserarea de date în acestea este *hr_project.sql*.

Diagrama entitate-relație corespunzătoare modelului *HR* va fi extinsă, pornind de la entitatea *EMPLOYEES*, astfel:




Partea din diagrama conceptuală corespunzătoare acestei extinderi a modelului este următoarea:




# 15. Variabile de substituție

Variabilele de substitutie sunt utile in crearea de comenzi/script-uri dinamice (care depind de niste valori pe care utilizatorul le furnizeaza la momentul rularii).

Variabilele de substitutie se pot folosi pentru stocarea temporara de valori, transmiterea de valori intre comenzi *SQL* etc. Ele pot fi create prin: comanda *DEFINE*.( *DEFINE* variabila = valoare )

Prefixarea cu & (indica existenta unei variabile intr-o comanda *SQL*, daca variabila nu exista, atunci *SQL*Plus* o creeaza).

Prefixarea cu && (indica existenta unei variabile intr-o comanda *SQL*, daca variabila nu exista, atunci *SQL*Plus* o creaza).

Deosebirea fata de & este ca, daca se foloseste &&, atunci referirea ulterioara cu & sau && nu mai cere ca utilizatorul sa introduca de fiecare data valoarea variabilei. Este folosita valoarea data la prima referire. Variabilele de substitutie pot fi eliminate cu ajutorul comenzii *UNDEF*[*INE*]

## 15.1 Comanda DEFINE

| Forma comenzii | Descriere |
|---|---|
| DEFINE variabila = valoare | Creaza o variabila utilizator cu valoarea de tip sir de caracter precizata. |
| DEFINE variabila | Afiseaza variabila, valoarea ei si tipul de data al acesteia. |
| DEFINE | Afiseaza toate variabilele existente in sesiunea curenta, impreuna cu valorile si tipurile lor de date. |

***Observatii:***

Variabilele de tip *DATE* sau *CHAR* trebuie sa fie incluse intre apostrofuri in comanda *SELECT*.

Dupa cum le spune si numele, variabilele de sustitutie inlocuiesc/substituie in cadrul comenzii *SQL* variabila respectiva cu sirul de caractere introdus de utilizator. Variabilele de sustitutie pot fi utilizate pentru a inlocui la momentul rularii:

- conditii *WHERE*;
- clauza *ORDER BY*;
- expresii din lista *SELECT*;
- nume de tabel;
- intreaga comanda *SQL*;

Odata definita, o variabila ramane pana la eliminarea ei cu o comanda *UNDEF* sau pana la terminarea sesiunii *SQL*Plus* respective.

Comanda *SET VERIFY ON | OFF* permite afisarea sau nu a comenzii inainte si dupa inlocuirea variabilei de substitutie.

## 15.2 Comenzi interactive în SQL Plus

| Comanda | Descriere |
|---|---|
| *ACC*[*EPT*] *variabila* [*tip*] [*PROMPT text*] | Citeşte o linie de intrare şi o stochează într-o variabilă utilizator. |
| *PAU*[*SE*] [*text* ] Afişează o linie vidă, | urmată de o linie conținând text, apoi aşteaptă ca utilizatorul să apese tasta *return*. De asemenea, această comandă poate lista două linii vide, urmate de aşteptarea răspunsului din partea utilizatorului. |
| *PROMPT* [*text*] | Afişează mesajul specificat sau o linie vidă pe ecranul utilizatorului. |

## 15.3 Creare fişiere script

De obicei, un fişier script constă în comenzi *SQL*Plus* şi cel puțin o instrucțiune *SELECT*. Crearea unui fişier script simplu se poate realiza urmând etapele expuse în continuare.

- Se redactează instrucțiunea *SELECT* la *prompt*-ul *SQL* sau în regiunea de editare din *iSQL*Plus*.
- Se salvează instrucțiunea *SELECT* într-un fişier *script*.
- Se editează fişierul *script*, adăugându-se comenzile *SQL*Plus* corespunzătoare.
- Se verifică dacă instrucțiunea *SELECT* este urmată de un caracter pentru execuție („;“ sau „/“).
- Se salvează fişierul *script*.

Se execută fişierul *script* (prin comenzile @ sau *START*). În *SQL Developer*, se încarcă fişierul şi se acționează butonul *Run Script*.

# 16. LMD & LCD

Comenzile SQL care alcătuiesc **LMD** permit:

- regăsirea datelor (*SELECT*);
- adăugarea de noi înregistrări (*INSERT*);
- modificarea valorilor coloanelor din înregistrările existente (*UPDATE*);
- adăugarea sau modificarea condiţionată de înregistrări (*MERGE*); ¾ suprimarea de înregistrări (*DELETE*).

**Tranzacţia** este o unitate logică de lucru, constituită dintr-o secvenţă de comenzi care trebuie să se execute atomic (ca un întreg) pentru a menţine consistenţa bazei de date.

*Server*-ul *Oracle* asigură consistenţa datelor pe baza tranzacţiilor, inclusiv în eventualitatea unei anomalii a unui proces sau a sistemului. Tranzacţiile oferă mai multă flexibilitate şi control în modificarea datelor.

Comenzile SQL care alcătuiesc **LCD** sunt:

- ROLLBACK – pentru a renunţa la modificările aflate în aşteptare se utilizează instrucţiunea *ROLLBACK*. În urma execuţiei acesteia, se încheie tranzacţia, se anulează modificările asupra datelor, se restaurează starea lor precedentă şi se eliberează blocările asupra liniilor.
- COMMIT - determină încheierea tranzacţiei curente şi permanentizarea modificărilor care au intervenit pe parcursul acesteia. Instrucţiunea suprimă toate punctele intermediare definite în tranzacţie şi eliberează blocările tranzacţiei.

***Obs:*** O comandă LDD (*CREATE, ALTER, DROP*) determină un COMMIT implicit.

- SAVEPOINT - Instrucţiunea *SAVEPOINT* marchează un punct intermediar în procesarea tranzacţiei. În acest mod este posibilă împărţirea tranzacţiei în subtranzacţii. Această instrucţiune nu face parte din standardul *ANSI* al limbajului *SQL*.

## 16.1 Comanda INSERT

### 16.1.1 Inserări mono-tabel

Comanda INSERT are următoarea sintaxă simplificată:

***INSERT INTO*** *obiect* [***AS*** *alias*] [ (*nume_coloană* [, *nume_coloană* ...] ) ]

{***VALUES*** ( {*expr* | ***DEFAULT***} [, {*expr* | ***DEFAULT***}
...] ) | *subcerere*}

Subcererea specificată în comanda *INSERT* returnează linii care vor fi adăugate în tabel.

Dacă în tabel se introduc linii prin intermediul unei subcereri, coloanele din lista *SELECT* trebuie să corespundă, ca număr şi tip, celor precizate în clauza *INTO*. În absenţa unei liste de coloane în clauza *INTO*, subcererea trebuie să furnizeze valori pentru fiecare atribut al obiectului destinaţie, respectând ordinea în care acestea au fost definite.

***Observaţii (tipuri de date):***

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

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

- implicit, prin omiterea numelui coloanei din lista de coloane;
- explicit, prin specificarea în lista de valori a cuvântului cheie *null*

În cazul şirurilor de caractere sau al datelor calendaristice se poate preciza şirul vid ('').

***Observaţii (erori):***

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

- nu a fost specificată o valoare pentru o coloană *NOT NULL*;
- există valori duplicat care încalcă o constrângere de unicitate;
- a fost încălcată constrângerea de cheie externă sau o constrângere de tip *CHECK*;
- există o incompatibilitate în privinţa tipurilor de date;
- s-a încercat inserarea unei valori având o dimensiune mai mare decât a coloanei corespunzătoare.

### 16.1.2 Inserări multi-tabel

O inserare multi-tabel presupune introducerea de linii calculate pe baza rezultatelor unei subcereri, într-unul sau mai multe tabele. Acest tip de inserare, introdus de *Oracle9i*, este util în mediul *data warehouse*.

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

Sintaxa comenzii *INSERT* în acest caz poate fi:

- Pentru inserări necondiţionate:

***INSERT ALL INTO***… [***INTO***…]
*subcerere;*

- Pentru inserări condiţionate:

***INSERT*** [***ALL*** | ***FIRST***]
***WHEN*** *condiţie* ***THEN INTO***…
[***WHEN*** *condiţie* ***THEN INTO***…
[***ELSE INTO*** …]]
*subcerere*;

- *ALL* determină evaluarea tuturor condiţiilor din clauzele *WHEN*. Pentru cele a căror valoare este *TRUE*, se inserează înregistrarea specificată în opţiunea *INTO* corespunzătoare.
- *FIRST* determină inserarea corespunzătoare primei clauze *WHEN* a cărei condiţie este evaluată *TRUE*. Toate celelalte clauze *WHEN* sunt ignorate.

## 16.2 Comanda UPDATE

Sintaxa simplificată a comenzii ***UPDATE*** este:

*UPDATE nume_tabel [alias]*

*SET col1 = expr1[, col2=expr2]*

*[WHERE conditie];*

sau

*UPDATE nume_tabel [alias]*

*SET (col1,col2,...) = (subcerere)*

*[WHERE conditie];*

***Observaţii:***

- de obicei pentru identificarea unei linii se foloseşte o condiţie ce implică cheia primară;
- dacă nu apare clauza *WHERE* atunci sunt afectate toate liniile tabelului specificat;
- cazurile în care instrucţiunea *UPDATE* nu poate fi executată sunt similare celor în care eşuează instrucţiunea *INSERT*. Acestea au fost menţionate anterior.

## 16.3 Comanda DELETE

Sintaxa simplificată a comenzii ***DELETE*** este:

*DELETE FROM nume_tabel*

*[WHERE conditie];*

Daca nu se specifica nici o conditie, vor fi şterse toate liniile din tabel.

## 16.4 Comanda MERGE

Instrucţiunea *MERGE* permite inserarea sau actualizarea condiţionată a datelor dintr-un tabel al bazei de date. Sintaxa ei simplificată este următoarea:

*MERGE INTO nume_tabel* [*alias*]

*USING* {*tabel* | *vizualizare* | *subcerere*} [*alias*]
*ON* (*condiţie*)

*WHEN MATCHED THEN*

*UPDATE SET coloana_1* = {*expr_u1* |
*DEFAULT*},…,

*coloana_n* = {*expr_un* | *DEFAULT*}

*WHEN NOT MATCHED THEN*

*INSERT* (*coloana_1*,…, *coloana_n*)

*VALUES* (*expr_i1*,…, *expr_in*);

Instrucţiunea efectuează:

- *UPDATE* dacă înregistrarea există deja în tabel
- *INSERT* dacă înregistrarea este nouă.

În acest fel, se pot evita instrucţiunile *UPDATE* multiple.

# 17. LDD

În general, instrucţiunile *LDD* sunt utilizate pentru definirea structurii corespunzătoare obiectelor unei scheme: tabele, vizualizări, vizualizări materializate, indecşi, sinonime, clustere, proceduri şi funcţii stocate, declanşatori, pachete stocate etc.

Aceste instrucţiuni permit:

- crearea, modificarea şi suprimarea obiectelor unei scheme şi a altor obiecte ale bazei de date, inclusiv baza însăşi şi utilizatorii acesteia (*CREATE*, *ALTER*, *DROP*);
- modificarea numelor obiectelor unei scheme (*RENAME*);
- ştergerea datelor din obiectele unei scheme, fără suprimarea structurii obiectelor respective (*TRUNCATE*).

Implicit, o instrucţiune *LDD* permanentizează (*COMMIT*) efectul tuturor instrucţiunilor precedente şi marchează începutul unei noi tranzacţii.

Instrucţiunile *LDD* au efect imediat asupra bazei de date şi înregistrează informaţia în dicţionarul datelor.

Definirea unui obiect presupune: crearea (CREATE), modificarea (ALTER) şi suprimarea sa (DROP).

**Reguli de numire a obiectelor bazei de date**

- Identificatorii obiectelor trebuie să înceapă cu o literă şi să aibă maximum 30 de caractere, cu excepţia numelui bazei de date care este limitat la 8 caractere şi celui al legăturii unei baze de date, a cărui lungime poate atinge 128 de caractere.
- Numele poate conţine caracterele *A-Z*, *a-z*, 0-9, _, *$* şi #.
- Două obiecte ale aceluiaşi utilizator al *server*-ului *Oracle* nu pot avea acelaşi nume.
- Identificatorii nu pot fi cuvinte rezervate ale *server*-ului *Oracle*.

Identificatorii obiectelor nu sunt *case-sensitive*

## 17.1 Crearea tabelelor – CREATE TABLE

Formele simplificate ale comenzii de creare a tabelelor sunt:

***CREATE TABLE*** *nume_tabel* **(**
*coloana_1 tip_date* ***[DEFAULT*** *valoare****]***
*[constrangere_nivel_coloana [constrangere_nivel_coloana]...]***,**
**. . . . . .**
*coloana_n tip_date* ***[DEFAULT*** *valoare****]***
*[constrangere_nivel_coloana [constrangere_nivel_coloana]...]***,**
***[****constrangeri_nivel_tabel****]***
**);**

sau

***CREATE TABLE*** *nume_tabel [(coloana_1,...,*
*coloana_n)]* ***AS*** *subcerere;*

**Constrângerile definite aupra unui tabel pot fi de următoarele tipuri:**

- NOT NULL - coloana nu poate conţine valoarea *Null*; (*NOT NULL*)
- UNIQUE – pentru coloane sau combinaţii de coloane care trebuie să aibă valori unice în cadrul tabelului; ( *UNIQUE* (*col1, col2, …*) )
- PRIMARY KEY - identifică în mod unic orice înregistrare din tabel. Implică NOT NULL + UNIQUE; (*PRIMARY KEY* (*col1, col2, …* ))
- FOREIGN KEY - stabileşte o relaţie de cheie externă între o coloană a tabelului şi o coloană dintr-un tabel specificat.
  [*FOREIGN KEY nume_col*]
  *REFERENCES nume_tabel*(*nume_coloana*)
  [ *ON DELETE* {*CASCADE*| *SET NULL*}]

  - *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“.
- CHECK- o condiţie care să fie adevărată la nivel de coloană sau linie (*CHECK* (*conditie*)). ***Obs:***
- Constrângerile pot fi create o dată cu tabelul sau adăugate ulterior cu o comandă *ALTER TABLE*.
- Constrângerile de tip CHECK se pot implementa la nivel de coloană doar dacă nu referă o altă coloană a tabelului.
- În cazul în care cheia primară (sau o cheie unică) este compusă, ea nu poate fi definită la nivel de coloane, ci doar la nivel de tabel.
- Constrângerea de tip NOT NULL se poate declara doar la nivel de coloană.

Principalele **tipuri de date** pentru coloanele tabelelor sunt următoarele:

| **Tip de date** | **Descriere** |
|---|---|
| VARCHAR2(n) [BYTE \| CHAR] | Defineşte un şir de caractere de dimensiune variabilă, având lungimea maximă de *n* octeţi sau caractere. Valoarea maximă a lui *n* corespunde la 4000 octeţi, iar cea minimă este de un octet sau un caracter. |
| CHAR(n) [BYTE \| CHAR] | Reprezintă un şir de caractere de lungime fixă având *n* octeţi sau caractere. Valoarea maximă a lui *n* corespunde la 2000 octeţi.<br>Valoarea implicită şi minimă este de un octet. |
| NUMBER(p, s) | Reprezintă un număr având *p* cifre, dintre care *s* cifre formează partea zecimală |
| LONG | Conţine şiruri de caractere având lungime variabilă, care nu pot ocupa mai mult de 2GB. |
| DATE | Reprezintă date calendaristice valide, între 1 ianuarie 4712 i.Hr. şi 31 decembrie 9999 d.Hr. |

## 17.2 Modificarea structurii tabelelor – ALTER TABLE

**Modificarea structurii unui tabel** se face cu ajutorul comenzii ***ALTER TABLE***. Forma comenzii depinde de tipul modificării aduse:

- adăugarea unei noi coloane (nu se poate specifica poziţia unei coloane noi în structura tabelului; o coloană nouă devine automat ultima în cadrul structurii tabelului)

  ***ALTER TABLE*** *nume_tabel*
  ***ADD (****coloana tip_de_date* ***[DEFAULT*** *expr][, ...]****);***

- modificarea unei coloane (schimbarea tipului de date, a dimensiunii sau a valorii implicite a acesteia; schimbarea valorii implicite afectează numai inserările care succed modificării)

  ***ALTER TABLE*** *nume_tabel*
  ***MODIFY*** (*coloana tip_de_date* [***DEFAULT*** *expr*][, ...])***;*** • eliminarea unei coloane din structura tabelului: ***ALTER TABLE*** *nume_tabel*
  ***DROP COLUMN*** *coloana****;***

***Obs:***

- 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*.
- o coloană *CHAR* poate fi convertită la tipul de date *VARCHAR2* sau invers, numai dacă valorile coloanei sunt *null* sau dacă nu se modifică dimensiunea coloanei.

Comanda *ALTER* permite **adăugarea unei constrângeri într-un tabel existent, eliminarea, activarea sau dezactivarea constrângerilor**.

- Pentru adăugare de constrângeri, comanda are forma:

  ***ALTER TABLE*** *nume_tabel*
  ***ADD*** [***CONSTRAINT*** *nume_constr*] *tip_constr* (*coloana*);

- Pentru eliminare de constrângeri:

  ***ALTER TABLE*** *nume_tabel*
  ***DROP PRIMARY KEY | UNIQUE***(*col1, col2, ...*) | ***CONSTRAINT*** *nume_constr;*

- Pentru activare/dezactivare constrângere:

  ***ALTER TABLE*** *nume_tabel*
  ***MODIFY CONSTRAINT*** *nume_constr*
  ***ENABLE|DISABLE****;* sau
  ***ALTER TABLE*** *nume_tabel*
  ***ENABLE| DISABLE CONSTRAINT*** *nume_constr;*

## 17.3 Suprimarea tabelelor – DROP/TRUNCATE TABEL

Ştergerea fizică a unui tabel, inclusiv a înregistrărilor acestuia, se realizează prin comanda:

***DROP TABLE*** *nume_tabel;*

Pentru ştergerea conţinutului unui tabel şi păstrarea structurii acestuia se poate utiliza comanda:

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

## 17.4 Redenumirea tabelelor – RENAME

Comanda ***RENAME*** permite redenumirea unui tabel, vizualizare sau secvenţă.

***RENAME*** *nume1_obiect* ***TO*** *nume2_obiect;*

***Obs:***

- În urma redenumirii sunt transferate automat constrângerile de integritate, indecşii şi privilegiile asupra vechilor obiecte.

Sunt invalidate toate obiectele ce depind de obiectul redenumit, cum ar fi vizualizări, sinonime sau proceduri şi funcţii stocate.

## 17.5 Consultarea dicţionarului datelor

Informaţii despre tabelele create se găsesc în vizualizările:

- USER_TABLES –informaţii complete despre tabelele utilizatorului.
- TAB – informaţii de bază despre tabelele existente în schema utilizatorului.

Informaţii despre constângeri găsim în USER_CONSTRAINTS, iar despre coloanele implicate în constrângeri în USER_CONS_COLUMNS.

# Cuprins