Departamentul Automatică şi Informatică Industrială
Facultatea Automatică şi Calculatoare
Universitatea POLITEHNICA din Bucureşti

Stocarea si procesarea informatiei

Bibliografie admitere master, an universitar 2020-2021, pentru programele de master coordonate de Departamentul AII:

Automatica si Informatica Industriala (AII)
Managementul si Protectia Informatiei (MPI)
Robotics and Automation (RA)
Prelucrari Complexe de Semnal in Aplicatii Multimedia (PCSAM)
Sisteme Informatice in Medicina (SIM)
Service Engineering and Management (SEM)

I. SQL - UN LIMBAJ PENTRU BAZE DE DATE RELATIONALE

In capitolul anterior s-au prezentat principalele operatii ale algebrei relationale. In acest capitol se urmareste prezentarea unui limbaj de implementare a acestor operatii intr-o baza de date relationala. In general aceste operatii sunt implementate intr-un limbaj de nivel inalt. Cu toate ca sunt mai multe metode de implementare a limbajelor de cereri cel mai cunoscut dintre limbajele implementate partial sau complet in diferite DBMS este SQL (Structured Query Language). Alte implementari, ce nu au confirmat din punctul de vedere al gradului de raspandire sunt cele cunoscute sub numele de QBE (Query By Exemple – limbaj utilizat in perioada 1990-2000 pentru familiarizarea utilizatorilor la formularea cererilor) si 4GL (4 Generation Language – limbaj de generatia patra ce a fost implementat cu succes la mediul Progress).

Cu toate ca initial SQL continea numai acele componente care permiteau implementarea limbajului de cereri, ulterior in SQL au fost integrate mai multe componente care permit atat definirea structurilor de date, manipularea acestora si gestionarea drepturilor de acces la date. Din acest motiv consideram SQL impartit in trei mari componente: SQL-DDL (Data Definition Language), specializat in definirea structurilor de date si a constrangerilor asociate, SQL-DML (Data Manipulation Language), specializat in implementarea limbajului de cereri (interogari) si SQL-DCL (Data Control Language), specializat in gestionarea drepturilor de acces la date. Spunem, in general, ca SQL permite specificarea a ceea ce se doreste obtinut in urma unei cereri si nu a modului in care acest lucru se produce, adica specificam ceeea ce dorim sa obtinem si nu cum obtinem acest lucru. Din acest motiv se mai spune ca SQL este un limbaj delarativ, nu procedural. Initial SQL a fost numit si SEQUEL (Structured QUEry Language) construit si implementat de IBM si a fost destinat pentru implementarea interfetei pentru baza de date relationala numita si SYSTEM R. SQL este noul limbaj pentru bazele de date relationale ale IBM DB2 si SQL/DS. Si alte medii de baze de date utilizeaza acelasi limbaj.

Cu toate ca SQL inglobeaza atat DDL, DML si DCL el are facilitati privind definirea si gestionarea si a altor obiecte ale bazei de date cum sunt: vederi, indecsi. Se descrie in continuare topica SQL implementata ca interfata pentru diferite baze de date relationale incluzand pe langa IBM DB2 SQL/DS, ORACLE, INGRES, UNIFY.

Definirea datelor in SQL (SQL-DDL)

SQL utilizeaza termenii *tabela, rand (integistrare) si coloana* respectiv pentru relatie, n-uplu si atribut. In continuare se vor utiliza acesti termeni cu toate ca este posibila continuarea utilizarii termenilor relatie, n-uplu si atribut. Pentru definirea structurii tabelelor bazei de date sunt utilizate comenzi SQL-DDL grupate in urmatoarele categorii:

1. CREATE TABLE ... permite crearea unei tabele, impreuna cu specificarea constrangerilor asociate acesteia;

2. ALTER TABLE ... permite modificarea structurii unei tabele definite anterioar cu efect adaugarea sau stergerea campurilor, respectiv modificarea constrangerilor impuse acestora;
3. DROP TABLE .... pentru stergerea tabelei specificate din baza de date. Stergerea este efectuata indiferent daca tabela contine sau nu date.

CREATE TABLE

CREATE TABLE este utilizata pentru definirea unei noi tabele, tabela ce are atribuit un nume si la care se defineste o structura. Prin structura tabelei intelegem atribuirea de nume fiecarui camp al tabelei, a tipului de data asociat in concordanta cu tipurile de date valide pentru mediul de baze de date in care se defineste tabela, a formatului de reprezentare a datelor, eventual a domeniului de valori permise si a constrangerilor asociate. Fara sa particularizam, tipurile de date uzual acceptate de majoritatea mediilor de baze de date sunt: sir de caractere, numeric, boolean (logical), data calendaristica, formate binare. Fiecare dintre tipurile de date mentionate pot fi reprezentate in diverse formate, astfel ca pentru datele de tip numeric avem formate de tipul: INTEGER (n), unde n este numarul maxim de digiti, REAL (FLOAT), NUMBER (m,n) cu m numarul maxim de digiti inclusiv punctul zecimal si n numarul de digiti dupa punctul zecimal. Un sir de caractere poate fi de lungime fixa (CHAR(n), unde n este numarul maxim de caractere) sau de lungime variabila (VARCHAR(n), unde n reprezinta tot numarul maxim de caractere, insa in tabela se rezerva pentru o inregistrare doar spatiul necesat pentru stocareasirului precizat la fiecare inregistrare). Se remarca faptul ca in tabelele la care avem campuri de tip CHAR inregistrarile sunt de lungime fixa, pe cand la tabelele in care sunt incluse campuri VARCHAR inregistrarile sunt de lungime variabila. Campurilor de tip BOOLEAN sau LOGICAL li se rezerva un octet chiar daca o valoare booleana poate fi reprezentata pe un singur bit. Campurile de tip data calendaristica au format impus la fiecare mediu de baze de date, ca reprezentare interna, cu toate ca exista o multitudine de formate de afisare ce pot fi setate de utilizator. Datele binare sunt pastrate in campuri de tip BLOB, CBLOB, LBLOB ce au volum implicit dependent de mediul de baze de date.

Ilustram in continuare crearea bazei de date COMPANIE fara sa impunem constrangeri. Asa cum am vazut in capitolele anterioare baza de date este constituita din tabelele: Angajat, Departament, Proiect, Lucreaza_la, Intretinut. In structurile comenzilor ilustrate vom folosi litere mari pentru cuvintele cheie din limbaj si litere mici pentru restul informatiilor.

```
CREATE TABLE Angajat (Nume          VARCHAR(20) ,
                      Ini           CHAR(1),
                      Prenume        VARCHAR(20) ,
                      Ssn           CHAR(13),
                      Data_n        DATE,
                      Adresa         VARCHAR(30),
                      Sex           CHAR(1),
                      Salariu       INTEGER,
                      SSSN          CHAR(13),
                      D_nr          INTEGER);
```

```
CREATE TABLE Departament (D_nume   VARCHAR(20) ,
                Dep_nr          INTEGER,
                D_loc VARCHAR(30)
                Manager         CHAR(13),
                Data_i          DATE);

CREATE TABLE Proiect     (P_nume     VARCHAR(15),
                P_nr            INTEGER,
                P_loc           VARCHAR(30),
                Dep_nr          INTEGER);

CREATE TABLE lucreaza_la (Ssn      CHAR(13),
                P_nr            INTEGER,
                Ore             NUMBER(4,1));

CREATE TABLE Intretinut  (A_ssn      CHAR(13),
                Nume_i          VARCHAR(20),
                Sex             CHAR(1),
                Data_n          DATE,
               Ssn      CHAR(13),
                Relatie         VARCHAR(10));
```

Fig. 7.1 Definirea tabelelor bazei de date Companie

In urma comenzii **CREATE TABLE** se creaza ceea ce se numeste *tabela de baza* in terminologia SQL. In urma acestei comenzi se creaza tabela cu structura specificata urmand ca ulterior tabela sa fie populate cu inregistrari. Astfel, in baza de date se creaza obiectul cu numele tabela, obiect ce este memorat in aceasta forma. Nu trebuie confundata relatia de baza cu o vedere (view) ce este creata utilizand comanda CREATE VIEW si va fi pastrata doar ca definitie.

Obs:

- Campurile in tabela sunt ordonate fizic in ordinea in care au fost precizate in comanda;
- In definirea numelui tabelelor trebuiesc respectate restrictiile impuse de DBMS in care se creaza tabela. Astfel, in Oracle numele tabelei nu poate depasi 30 caractere, nu poate incepe cu caracterul "." (punct) si nu poate contine blank.
- Numele campurilor tabelelor trebuie sa respecte restrictiile impuse de DBMS in care tabela este creata. In Oracle numele unui camp nu poate depasi 30 caractere, dar poate contine blank numai daca se utilizeaza o anumita notatie.

Definirea constrangerilor

Mediile de baze de date permit la definirea tabelelor specificarea constrangerilor impuse intre campurile tabelelor ce formeaza baza de date, modalitate prin care se precizeaza atat relatiile dintre

tabele cat si restrictiile privind valorile posibile admise acestor campuri. Modalitatea de realizare este bazata pe specificarea constrangerilor, constrangerile uzuale ce pot fi declarate fiind:

1. ***PRIMARY KEY***, constrangere ce poate fi impusa unui camp sau unei asociatii de campuri ce nu permite valori nule (pentru toate) sau valori identice la oricare doua inregistrari;
2. ***UNIQUE***, constrangere impusa unui camp sau unei asociatii de campuri ce nu permite valori identice diferite de valoarea null in oricare doua inregistrari;
3. ***FOREIGN KEY***, constrangerea numita si constrangere referentiala prin care valorile unui camp sau asociatii de campuri se regasesc printre valorile unui camp din alta tabele numita si tabela de referinta;
4. ***NOT NULL***, constrangere ce nu permite ca un camp sau o asociatie de campuri sa aiba valoarea NULL, respectiv toate valorile NULL;
5. ***CHECK***, constrangere prin care valorile permise unui camp sau unei asociatii de campuri se gasesc intr-o gama de valori predefinite.

O constrangere poate fi definita la nivel de camp (numita si in linie) sau la nivel de tabela. Cu toate ca, de foarte multe ori, o constrangere poate fi precizata fie la nivel de camp fie la nivel de tabela, sunt totusi situatii in care o constrangere nu poate fi precizata la nivel de camp, fiind obligatorie precizarea sa la nivel de tabela. Astfel, constrangerile impuse unei asociatii de campuri pot fi definite numai la nivel de tabela.

Constragerea PRIMARY KEY

O astfel de constrangere precizeaza faptul ca un camp sau o asociatie de campuri reprezinta o cheie primara a tabelei. O tabela poate avea mai multe chei candidate, adica mai multe campuri ce identifica unic o inregistrare. Dintre acestea se alege una ce va fi definita ca fiind cheie primara. O tabela poate avea o singura cheie primara, adica o astfel de constrangere poate aparea o singura data in definita unei tabele. Constrangerea poate avea nume asociat sau nu. Pentru definirea constrangerii de tip cheie primara la nivel de camp, si respectiv la nivel de tabela, in momentul definirii tabelai, se utilizeaza sintaxa:

```
CREATE TABLE Nume_tabela (…..,
            Nume_camp tip_data [CONSTRAINTS nume_cons] PRIMARY KEY,

…);
CREATE TABLE Nume_tabela (…..,
             Nume_ultim_camp tip_data,
             [CONSTRAINTS nume_cons] PRIMARY KEY (lista campuri));
```

Prima definire corespunde constrangerilor declarate la nivel de linie, cea de a doua la nivel de tabela. Se observa in definirile date faptul ca atribuirea de nume prin [CONSTRAINTS nume_cons] este optionala. Acesta secventa apare daca i se atribuie un nume constrangerii, in cazul in care lipseste constrangerea nu are un nume asociat. La definirea unei constrangeri de tip cheie primara DBMS creaza automat o structura de acces rapida la date (index). Se dau mai jos exemple.

- Definire la nivel de camp in tabela Angajat. Campul Ssn este declarat cheie primara, iar numele constrangerii este ang_PK.

```
CREATE TABLE Angajat (Nume          VARCHAR(20) ,
                      Ini           CHAR(1),
                      Prenume       VARCHAR(20) ,
                      Ssn           CHAR(13) CONSTRAINTS ang_PK PRIMARY KEY,
                      Data_n        DATE,
                      Adresa        VARCHAR(30),
                      Sex           CHAR(1),
                      Salariu       INTEGER,
                      Sssn          CHAR(13),
                      D_nr          INTEGER);
```

- Definire la nivel de tabela in tabela Lucreaza_la. In acest caz nu este posibila definirea la nivel de camp, intrucat cheia primara este o asociatie de campuri caz in care constrangerea de defineste la nivel de tabela. In exemplu, cheia primara este formata din asociatia campurilor Ssn si P_nr, constrangerea are asociat numele lucr_PK (putea sa lipseasca numele). In al doilea exemplu, daca consideram ca in tabela Angajat avem si campurile Serie_bi si Nr_bi cu semnificatia serie si respectiv numar buletin, asociatia lor poate fi o cheie primara si va fi definita la nivel de tabela. Nu am asociat nume la cheia primara formate de campurile (Serie_bi, Nr_bi).

```
CREATE TABLE lucreaza _la       (Ssn      CHAR(13),
                      P_nr          INTEGER,
                      Ore           NUMBER (4,1),
                     CONSTRAINT lucr_PK PRIMARY KEY(Ssn, P_nr));

CREATE TABLE Angajat (Nume          VARCHAR(20) ,
                      Ini           CHAR(1),
                      Prenume       VARCHAR(20) ,
                      Ssn           CHAR(13),
                     Serie_bi       CHAR(2),
                     Nr_bi          CHAR(6),
                      Data_n        DATE,
                      Adresa        VARCHAR(30),
                      Sex           CHAR(1),
                      Salariu       INTEGER,
                      Sssn          CHAR(13),
                      D_nr          INTEGER),     PRIMARY KEY(Serie_bi, Nr_bi));
```

Observatii:

- Daca cheia primara este formata dintr-un singur camp constrangerea se poate formula fie la nivel de camp fie la nivel de tabela, pe cand in situatia in care cheia primara este o asociatie de campuri constramgerea se precizeaza numai la nivel de tabela;

- O tabela are o singura cheie primara;
- Cheia primara este una dintre cheile candidate aleasa de proiectant.

Constrangerea FOREIGN KEY

Constrangerea precizeaza faptul ca un camp dintr-o tabela este o cheie straina, adica valorile pe care acest camp le poate lua se gasesc printre valorile unui camp din alta tabela in care campul este fie cheie primara fie un camp unic. Tabela la care se refera mai poarta denumirea si de tabela referinta sau tabela parinte, iar tabela in care se defineste cheia straina de mai numeste si tabela copil. Pentru definirea constrangerii Foreign key la nivel de camp, respectiv tabela se va utiliza:

```
CREATE TABLE Nume_tabela_copil (....
            Nume_camp tip_data [CONSTRAINTS nume_cons] REFERENCES
            Tabela_referinta (Camp_referinta) FOREIGN KEY
            [no action|on delete cascade|on delete set NULL|on delete set DEFAULT],
             .....);

CREATE TABLE Nume_tabela_copil (....
             Nume_ultim camp tip_data,
             [CONSTRAINTS nume_cons] REFERENCES
             Tabela_referinta (Camp_referinta)
             FOREIGN KEY (nume_camp_tabela_copil)
            [no action|on delete cascade|on delete set NULL|on delete set DEFAULT]);
```

Similar cu cheia primara [CONSTRAINTS nume_cons] poate lipsi daca nu se doreste asociarea unui nume constrangerii definite, REFERENCES specifica tabela parinte cu precizarea campului/campurilor referite, FOREIGN KEY specifica tipul constrangerii si eventual lista camp/campuri daca definirea este facuta la nivel de tabela. Unul dintre atributele optionale [no action|on delete cascade|on delete set NULL|on delete set DEFAULT] precizeaza regulile de validare a datelor cu restrictiile referentiale:

- Uzual, trebuie sa existe totdeauna o inregistrare in tabelul parinte cu valoare identica cu cea a cheii straine la inserarea in tabelul copil.
- Atunci cand o inregistrare din tabelul parinte este stearsa si in tabelul copil exista inregistrari cu acceasi valoare in coloana/coloanele cu care este in relatie prin atributele specificate foreign key, sunt setate reguli referitoare la actiunile intreprinse. Actiunile depind de regulile definte odata cu definirea cheii straine. In majoritatea mediilor de baze de date sunt posibile patru tipuri de reguli : NO ACTION, CASCADE, SET NULL si SET DEFAULT. Daca nu se defineste nici o regula este activa regula implicita.

O regula de tip NO ACTION, nu va fi permisa stergerea randului din tabelul parinte daca exista inregistrari in tabelul copil cu aceste valori. Ca urmare, toate randurile din tabelul copil cu aceeasi valoare a campului vor trebui sterse mai intai dupa care se poate sterge si randul tabelei parinte. Pentru ca un rand al tabelei parinte sa poata fi sters, in tabelul copil, nu trebuie sa existe nici o inregistrare cu acea valoare a cheii straine. Daca este specificat CASCADE, stergerea unui rand in

tabelul parinte va produce automat si stergerea randurilor corespunzatoare din tabela copil, o stergere in cascada. Daca este specificat SET NULL, atunci cand un rand din tabelul parinte este sters in tabelul copil, valoarea cheii straine corespondente va fi setata ca fiind NULL. Aceasta regula este posibila numai daca campul/campurile cheie straina nu are definita si o restrictie NOT NULL. La specificare SET DEFAULT, atunci cand un rand din tabelul parinte este sters in tabelul copil, valoarea cheii straine corespondente va fi setata la valoarea DEFAULT, daca o astfel de valoare este definita pentru respectivul camp. O verificare este facuta si la operatiile de modificare a inregistrarilor, astfel la modificarea unor valori, NO ACTION determina rejectarea operatiei de modificare in tabelul parinte daca la sfasitul operatiei de modificare exista randuri dependente in tabelul copil ce nu au o cheie corespondenta in tabelul parinte.

In tabela Departament campul Dep_nr este definit ca cheie primara putand identifica in mod unic fiecare inregistrare a tabelei, in tabela Angajat campul D_nr identifica departamentul din care face parte angajatul si ca urmare valorile sale trebuie sa se regaseasa printre valorile campului Dep_nr din tabela Departament. In exemplul urmator se defineste la tabela angajat o cheie straina pentru campul D_nr, constrangerea are numele ang_FK, refera din tabela Departament campul Dep_nr si actiunea la stergere este set NULL.

```
CREATE TABLE Angajat (Nume          VARCHAR(20) ,
                      Ini           CHAR(1),
                      Prenume       VARCHAR(20) ,
                      Ssn           CHAR(13) CONSTRAINTS ang_PK PRIMARY KEY,
                      Data_n        DATE,
                      Adresa        VARCHAR(30),
                      Sex           CHAR(1),
                      Salariu       INTEGER,
                      Sssn          CHAR(13),
                      D_nr          INTEGER CONSTRAINTS ang_FK REFERENCES
                     Departament(Dep_nr) FOREIGN KEY on delete set NULL);
```

Obs:

- O cheie straina face referire la o tabela in care campul referit este o cheie primara sau este declarat unic;
- Modul de tratare a integritatii este definit prin specificarea actiunilor intreprinse: *no action* sau *on delete cascade* sau *on delete set NULL* sau *on delete set DEFAULT* la stergerea sau modificarea datelor.

Constrangerea UNIQUE

O astfel de constrangere se va impune oricarui camp la care nu se accepta valori identice, altele decat valorile NULL. Trebuie remarcat ca in orice mediu de baze de date valorile NULL sunt considerate diferite, sunt medii de baze de date in care NULL este cea mai mare valoare si medii in care NULL reprezinta cea mai mica valoare. Constrangerea este similara cu cea de tip cheie primara cu diferenta ca se accepta si valori nule. Constrangerea poate fi definita atat la nivel de

camp cat si la nivel de tabela, similar cu cele descrise anterior. Se dau mai jos formele generale de definire la nivel de camp, respectiv tabela.

```
CREATE TABLE Nume_tabela (…..,
            Nume_camp tip_data [CONSTRAINTS nume_cons] UNIQUE,

…);
CREATE TABLE Nume_tabela (…..,
            Nume_ultim_camp tip_data,
            [CONSTRAINTS nume_cons] UNIQUE (lista campuri));
```

Ca exemplu in tabela Angajat in care avem campul Ssn cheie primara si campul D_nr cheie straina, vom defini asociatia de campuri Serie_bi si Nr_bi ca fiind unice. O astfel de definire trebuie facuta la nivel de tabela intrucat asociatia celor doua campuri trebuie sa fie unica.

```
CREATE TABLE Angajat (Nume          VARCHAR(20) ,
            Ini             CHAR(1),
            Prenume         VARCHAR(20) ,
            Ssn             CHAR(13) CONSTRAINTS ang_PK PRIMARY KEY,
            Data_n          DATE,
            Adresa          VARCHAR(30),
            Sex             CHAR(1),
            Salariu         INTEGER,
            Sssn            CHAR(13),
            D_nr            INTEGER CONSTRAINTS ang_FK REFERENCES
           Departament(Dep_nr) FOREIGN KEY on delete set NULL,
            Serie_bi        CHAR(2),
            Nr_bi           CHAR(6),
           CONSTRAINT bi_unic UNIQUE(Serie_bi, Nr_bi));
```

Daca constrangerea implica un singur camp aceasta poate fi definita la nivel de camp. Daca o tabela are o singura constrangere de tip PRIMARY KEY, ea poate avea oricate constrangeri de tip UNIQUE. Vom ilustra modul in care se defineste o constrangere de tip UNIQUE la nivel de camp cat si situatia in care o tabela contine doua campuri cu constrangerea de tip UNIQUE. In tabela Departament impunem ca D_nume si D_loc sa fie unice independent, nu ca asociatie, fara sa dam un nume constrangerii, campul Dep_nr cheie primara, campul Manager cheie straina cu referire la cheia primara a tabelei Angajat cu actiunea la stergere cascade.

```
CREATE TABLE Departament (D_nume   VARCHAR(20) UNIQUE,
            Dep_nr          INTEGER CONSTRAINT dep_PK PRIMARY KEY,
            D_loc VARCHAR(30) UNIQUE,
            Manager         CHAR(13) CONSTRAINT dep_FK
           REFERENCES angajat(SSN) FOREIGN KEY on delete cascade,
```

Data_i DATE);

Constrangerea NOT NULL

Este una dintre cele mai simple tipuri de constrangeri prin care nu se permit valori nule pentru un camp sau asociatie de campuri. Forma generala pentru definirea constrangerii este una din cele de mai jos functie de locul de definire: linie sau tabela.

```
CREATE TABLE Nume_tabela (…..,
        Nume_camp tip_data [CONSTRAINTS nume_cons] NOT NULL,

…);
CREATE TABLE Nume_tabela (…..,
        Nume_ultim_camp tip_data,
        [CONSTRAINTS nume_cons] NOT NULL (lista campuri));
```

De regula la constrangerile NOT NULL nu se atribuie nume cu toate ca acest lucru este permis. La o constrangere NOT NULL pentru o asociatie de campuri nu este permisa valoarea NULL pentru toate campurile, daca la o inregistrare, un singur camp are valoare diferita de NULL inregistrarea este valida. In exemplul urmator vom impune pentru campurile Nume, Prenume, si Data_n valori diferite de NULL, vazute ca si campuri independente.

```
CREATE TABLE Angajat (Nume      VARCHAR(20)  NOT NULL,
        Ini         CHAR(1),
        Prenume     VARCHAR(20) CONSTRAINT p_nenul NOT NULL,
        Ssn         CHAR(13) CONSTRAINTS ang_PK PRIMARY KEY,
        Data_n      DATE NOT NULL,
        Adresa      VARCHAR(30),
        Sex         CHAR(1),
        Salariu     INTEGER,
        Sssn        CHAR(13),
        D_nr        INTEGER CONSTRAINTS ang_FK REFERENCES
       Departament(Dep_nr) FOREIGN KEY on delete set NULL,
        Serie_bi    CHAR(2),
        Nr_bi       CHAR(6),
       CONSTRAINT bi_unic UNIQUE(Serie_bi, Nr_bi));
```

Obs:

- Daca unui camp avand constrangerea UNIQUE i se atribuie si constrangerea NOT NULL se comporta identic cu o constrangere de tip PRIMARY KEY;
- Daca un camp avand constrangerea FOREIGN KEY are asociata si constrangerea NOT NULL, definirea constrangerii FOREIGN KEY cu atributul on delete SET NULL este fara sens, intrucat nu poate lua valori nule. Definirea este acceptata dar nu are nici un efect.

Constrangerea CHECK

Prin constrangerea de tip CHECK se specifica restrictiile privind valorile posibile ale unor campuri si poate fi definita la nivel de camp sau ca expresie la nivel de tabela. Modalitatea de definire a unei constrangeri CHECK este data in continuare:

```
CREATE TABLE Nume_tabela (…..,
                Nume_camp tip_data [CONSTRAINTS nume_cons]
                CHECK (definire_expresie),

…);
CREATE TABLE Nume_tabela (…..,
                Nume_ultim_camp tip_data,
                [CONSTRAINTS nume_cons] CHECK (definire_expresie);
```

In prima forma, expresia de definire invoca numai campul (linia) in care este definita constrangerea, pe cand in a doua forma pot fi invocate in expresie oricare dintre campurile tabelei. Ca exemplu, vom considera o tabela Student intr-o universitate la care pentru campurile “an studiu” si bursa se impun restrictii de valori intre 1 si 4, respectiv trei valori distincte posibile pentru bursa 120, 160
210. In acest exemplu se indica si modul in care este tratat numele unui camp ce contine blank, acesta fiind pus intre ghilimele (cerinta Oracle, multe alte medii nu suporta blank in numele campurilor).

```
CREATE TABLE Student (nume VARCHAR(10) NOT NULL,…
                “an studiu”    INTEGER CHECK(“an studiu” BETWEEN 1 and 5),
        bursa   INTEGER CHECK(bursa = 120 or bursa = 160 or
                bursa = 210)…….)
```

respectiv

```
CREATE TABLE Student (nume VARCHAR(10) NOT NULL,…
                “an studiu”    INTEGER,
bursa   INTEGER,
                CONSTRAINT bursa_an CHECK(“an studiu” >0 and “an studiu<6 and
                (bursa = 120 or bursa = 160 or bursa = 210))…….)
```

Obs:

- In primul exemplu au fost puse restrictii pe campuri pe cand in al doilea restrictiile sunt puse global pe tabela;
- Campul an studiu avand avand denumirea separata prin blank a fost inclus intre semne de grupare “ “;

- S-a utilizat atat o conditie logica cat si operatorul between care verifica daca valorile unui camp sunt cuprinse intre o valoare minima si una maxima;
- A fost specificata si alta constrangere in structura tabelei, cum este constrangerea NOT NULL.

Definirea unei valori implicite (DEFAULT)

Specificarea unei valori implicite nu este o constrangere, ci valoarea unui camp atunci cand se insereaza o noua inregistrare si nu se precizeaza valori pentru campul la care a fost definita o astfel de valoare. Ca exemplu, in tabela Student, definita anterior, campul "an studiu" are de regula valoarea 1 atunci cand se introduce o inregistrare noua, intrucat noii studenti sunt matriculati in anul 1. Daca un student este transferat intr-un an mai mare de la alta universitate la introducerea sa in baza de date se va preciza valoarea campului "an studiu".

```
CREATE TABLE Student (nume VARCHAR(10) NOT NULL,…
                "an studiu"     INTEGER DEFAULT 1,
                bursa           INTEGER,
               CONSTRAINTS CHECK("an studiu" >0 and "an studiu<6 and
               (bursa = 120 or bursa = 160 or bursa = 210))…….)
```

Cu aceste precizari vom defini baza de date Companie introducand si constrangerile aferente fara ca la toate constrangerile sa asociem si nume:

```
CREATE TABLE Angajat (Nume          VARCHAR(20)  NOT NULL,
                Ini             CHAR(1),
                Prenume          VARCHAR(20) NOT NULL,
                Ssn             CHAR(13) CONSTRAINTS ang_PK PRIMARY KEY,
                Data_n          DATE CONSTRAINT dat_nenul NOT NULL,
                Adresa           VARCHAR(30) CONSTRAINT adr_nenul NOT NULL,
                Sex             CHAR(1),
                Salariu         INTEGER DEFAULT 1200,
                Sssn            CHAR(13) CONSTRAINTS sup_FK REFERENCES
               Angajat (Ssn) FOREIGN KEY on delete set NULL,
                D_nr            INTEGER CONSTRAINTS ang_FK REFERENCES
               Departament(Dep_nr) FOREIGN KEY on delete CASCADE,
                Serie_bi        CHAR(2),
                Nr_bi           CHAR(6),
               CONSTRAINT bi_unic UNIQUE(Serie_bi, Nr_bi));

CREATE TABLE Departament (D_nume   VARCHAR(20) NOT NULL,
                Dep_nr          INTEGER PRIMARY KEY,
                D_loc VARCHAR(30) NOT NULL,
                Manager         CHAR(13) CONSTRAINTS dep_FK REFERENCES
```

```
Angajat(SSn) FOREIGN KEY on delete set NULL,
 Data_i          DATE NOT NULL);


CREATE TABLE Proiect     (P_nume     VARCHAR(15) NOT NULL,
 P_nr            INTEGER CONSTRAINTS pr_PK PRIMARY KEY,
 P_loc           VARCHAR(30) NOT NULL,
 Dep_nr          INTEGER CONSTRAINTS pr_FK REFERENCES
Departament(Dep_nr) FOREIGN KEY on delete set NULL);

CREATE TABLE Lucreaza_la (Ssn     CHAR(13) CONSTRAINTS lu_FK REFERENCES
Angajat(SSn) FOREIGN KEY on delete CASCADE,
 P_nr            INTEGER CONSTRAINTS luc_FK REFERENCES
Proiect(P_nr) FOREIGN KEY on delete set NULL,
 Ore             NUMBER(4,1) NOT NULL,
CONSTRAINT lu_PK PRIMARY KEY(Ssn, P_nr));

CREATE TABLE Intretinut  (A_ssn      CHAR(13) CONSTRAINTS int_FK
REFERENCES Angajat(SSn) FOREIGN KEY on delete CASCADE,
 Nume_i          VARCHAR(20) NOT NULL,
 Sex             CHAR(1),
 Data_n          DATE NOT NULL,
Ssn             CHAR(13) PRMARY KEY,
 Relatie         VARCHAR(10) NOT NULL,
CONSTRAINT n_unic UNIQUE(A_ssn,Nume_i));
```

Obs: Toate exemplele date mai sus cu scopul de a crea obiecte de tip tabela ale bazei de date pot fi executate cu succes numai daca utilizatorul care invoca astfel de instructiuni are drepturi corespunzatoare.

Comanda DROP TABLE

Un obiect al bazei de date de tip tabela poate fi sters daca utilizatorul are drepturi corespunzatoare. Stergerea unei tabele poate fi efectuata indiferent daca tabela contine sau nu inregistrari. Comanda specifica pentru stergerea tabelelor este:

```
DROP TABLE Nume_tabela
```

Ca exemplu, daca se doreste stergerea tabelei Intretinut ca obiect al bazei de date COMPANIE, se executa:

```
DROP TABLE Intretinut;
```

Odata cu stergerea unei tabele sunt sterse toate constrangerile in care tabela era implicata, ca urmare si constrangerile de tip FOREIGN KEY in alte tabele care la REFERENCES precizau numele tabelei sterse.

Comanda ALTER TABLE

De foarte multe ori dupa ce o tabela a fost creata este necesara modificarea sa (din punct de vedere sructura). Prin modificarea structurii unei tabele se intelege adaugarea de noi campuri, stergerea unor campuri existente, modificarea definirii unui camp, adaugarea sau stergerea de constrangeri, modificarea acestora, activarea sau dezactivarea constrangerilor. Vom trata mai intai modificarile aferente campurilor tabelei, apoi cele care se refera la constrangeri. Forma generala a unei comenzi de modificare a definitiei unei tabele este:

ALTER TABLE Nume_tabela TIP_MODIF (definitie)

in care prin TIP_MODIF se precizeaza, prin cuvinte cheie, tipul de modificare dorit (adaugare de campuri, stergere de campuri, modificarea definirii unor campuri), iar prin definitie se specifica noile caracteristici ale campului. Definitia permite si specificarea noilor constrangeri pentru campurile modificate. Daca o modificare se refera numai la constrangeri ea va fi realizata conform sectiunii de modificare constrangeri.

Adaugarea unui nou camp

Pentru a adauga un nou camp sau mai multe campuri la o tabela se utilizeaza comanda

```
ALTER TABLE Nume_tabela ADD (
        Nume_camp1 tip_data [DEFAULT expresie] [constrangere],
        Nume_camp2 tip_data  [DEFAULT expresie] [constrangere])
```

Se observa ca sunt optionale precizarea valorii implicite pentru camp si precizarea de constrangeri. Desigur ca aceste constrangeri se refera la campurile noi introduse, nu la campuri existente, campuri ce pot avea sau nu constrangeri. Ca exemplu, sa consideram ca se doreste adaugarea unui camp Studii la tabela Angajat care are o valoare implicita si nu poate avea valori NULL.

```
ALTER TABLE Angajat ADD (studii VARCHAR(15) DEFAULT “medii”
        CONSTRAINT ang_stud NOT NULL)
```

In cazul de fata daca tabela Angajat contine cel putin o inregistrare modificarea structurii tabelei nu este posibila intrucat la adaugarea noului camp in inregistrarile existente valorile sunt NULL si pentru campul Studii s-a introdus constrangerea NOT NULL

.

Obs:

- Daca tabela contine cel putin o inregistrare noua coloana va avea valori nule;

- Impunerea unei constrangeri NOT NULL este permisa numai daca tabela nu contine inregistrari;
- Coloana adaugata este ultima coloana in ordine fizica, celelalte coloane respectand ordinea de definire.

Stergerea unui camp existent

Cu anumite restrictii structura unei tabele poate fi modificata prin stergerea campurilor specificate. Operatia de stergere a unuia sau mai multor campuri este o operatie consumatoare de timp si nu se recomanda a fi executata cu baza de date in productie. Comanda care are ca efect stergerea de coloane are structura:

ALTER TABLE Nume_tabela DROP COLUMN nume_coloana
[CASCADE CONSTRAINTS]
sau

ALTER TABLE Nume_tabela DROP (lisata coloane) [CASCADE CONSTRAINTS]

in care se specifica stergerea unei singure coloane sau stergerea mai multor coloane si optional modul in care sunt tratate constrangerile. In exemplul de mai jos se ilustreaza doua modalitati de stergere din tabela Angajat a campului Sssn. In ambele cazuri nu s-a utilizat CASCADE CONSTRAINTS pentru ca nu are nici un efect. Fiind vorba de un singur camp care este o cheie straina chiar daca referinta este aceeasi tabela nu are efect cascadarea constrangerilor. Cascadarea este importanta daca campul sters este referit in declararea unei constrangeri de tip FOREIGN KEY sau face parte dintr-o asociatie de campuri ce definesc o constrangere.

ALTER TABLE angajat DROP (SSSN)
ALTER TABLE angajat DROP COLUMN SSSN

Obs:

- Cererea poate fi executata atat la tabele cu inregistrari cat si la tabele fara inregistrari;
- Daca tabela contine o singura coloana aceasta nu poate fi stearsa;
- Optiunea CASCADE CONSTRAINTS sterge suplimentar toate constrangerile in care sunt implicate coloanele sterse inclusiv cele de tip FOREIGN KEY
- Stergerea este o operatie consumatoare de timp.

In Oracle a fost gasita o solutie de renuntare la campuri fara sa fie sterse efectiv din tabela, dar baza sa se comporte ca si cum aceste campuri au fost sterse. Solutia oferita de Oracle se bazeaza pe marcare campurilor ca fiind neutilizate prin SET UNUSED. Pentru a declara campuri neutilizabile vom folosi:

ALTER TABLE nume_tabela SET UNUSED (lista coloane)

sau

ALTER TABLE nume_tabela SET UNUSED COLUMN nume_coloana

Dupa ce campurile au fost marcate ca neutilizabile, se produc urmatoarele efecte:

- Campurile marcate nu mai apar in structura afisata la DESCRIBE;
- Campurile marcate nu mai pot fi folosite in interogari SQL;
- Pot fi adaugate coloane cu acelasi nume cu cel al coloanelor marcate ca neutilizabile.
- In afara de spatiul de stocare ocupat, marcarea ca neutilizabile a unor campuri are acelasi efect cu stergerea lor.

Campurile marcate ca neutilizabile pot fi sterse efectiv atunci cand nu afecteaza performatele bazei de date pentru a elibera spatiul de stocare, prin:

ALTER TABLE nume_tabela DROP UNUSED COLUMNS

Modificarea definirii unei coloane

Structura generala pentru modificarea definirii unor coloane existente este:

ALTER TABLE Nume_tabela MODIFY
(nume_coloana [tip_data] [DEFAULT expresie] [constrangere]

in care se invoca cuvantul cheie MODIFY pentru tip de operatie, nume_coloana este numele unei coloane existente in tabela celelalte elemente descriu noua definire a campului (coloanei) modificate. Ca exemplu aratam redefinirea coloanei D_nr aferenta tabelei Angajat:

ALTER TABLE Angajat MODIFY (D_nr INTEGER CONSTRAINT ang_FK REFERENCES departament(Dep_nr) on delete set NULL)

La redefinirea campurilor pot fi produse urmatoarele efecte:

- Schimbarea tipul de date al coloanei daca tabela nu contine inregistrari sau daca redefinirea duce la tipuri de date compatibile (ex: cresterea numarului de caractere pentru un camp cu tip data sir de caractere);
- Poate fi asociata o noua valoare implicita;
- Poate fi adaugata o constrangere;
- Toate aceste operatii pot fi realizate la aceeasi utilizare ALTER TABLE

Adaugarea unei noi constrangeri

O alta categorie de modificare a structurii unei tabele produce efecte numai asupra constrangerilor asociate campurilor. Astfel de modificari nu au efect asupra tipului de data asociat campurilor ci doar asupra constrangerilor impuse. Structura generala a comenzii este:

ALTER TABLE nume_tabela ADD [CONSTRAINT nume_constr] tip_constr(coloana)

Daca in tabela Departament dorim ca valorile campului D_nume sa fie diferite de NULL vom adauga o constrangere care impune valori diferite de NULL.

```
ALTER TABLE Departament ADD CONSTRAINT nume_nenul
        CHECK (D_nume IS NOT NULL)
```

Adaugarea constrangerii putea fi realizata si prin NOT NULL, aici am introdus o constrangere de tip CHECK in care este obligatorie utilizarea pentru comparatie a cuvantului cheie IS (comparatii cu valori NULL). Daca la acelasi tabel dorin pentru campul D_nume valori unice vom adauga constrangerea:

```
ALTER TABLE Departament ADD CONSTRAINT nume_unic UNIQUE(D_nume)
```

iar daca numele departamentului trebuie sa aiba minim 7 caractere vom adauga constrangerea :

```
ALTER TABLE departament ADD CONSTRAINT nume_car_min
        CHECK (LENGTH(d_nume)>6)
```

Stergerea unei constrangeri existente

In anumite situatii se doreste eliminarea unei constrangeri definite anterior. Orice constrangere definita la o tabela poate fi stearsa utilizand o comanda corespunzatoare. La stergerea constrangerii fie ne referim la definirea constrangerii fie constrangerea este specificata prin numele sau. In primele doua exemple se sterg constrangeri prin referirea la definitie (PRIMARY KEY si UNIQUE), pe cand in al treilea se specifica numele constrangerii:

```
ALTER TABLE Nume_tabela DROP PRIMARY KEY [CASCADE]
ALTER TABLE Nume_tabela DROP UNIQUE (lisata_coloane) [CASCADE]
ALTER TABLE Nume_tabela DROP CONSTRAINT nume [CASCADE]
```

Efectul comenzilor:

- DROP PRIMARY KEY si DROP UNIQUE sterge constrangerea de tip cheie primara/valoare unica chiar daca constrangerea nu are un nume asociat;
- DROP CONSTRAINT specifica stergerea unei constrangeri cu nume asociat;
- Optiunea CASCADE are ca efect stergerea constrangerilor dependente daca astfel de constrangeri exista.

Activarea si dezactivarea unei constrangeri

Atunci cand se doreste inhibarea temporara a efectului unor constrangeri nu este practica stergerea acestora, deoarece trebuie redefinita pentru a produce efecte. O metoda mult mai utila este cea de dezactivare a constrangerii cu posibilitatea de activare ulterioara. In acest mod constrangerea este definita o singura data, efectul ei putand fi inhibat intre momentul de dezactivare si cel de activare.

Pentru dezactivarea unei constrangeri se utilizeaza cuvantul cheie DISABLE urmat fie de definitia constrangerii fie de numele acesteia.

ALTER TABLE Nume_tabela DISABLE PRIMARY KEY [CASCADE]
ALTER TABLE Nume_tabela DISABLE UNIQUE (lisata_coloane) [CASCADE]
ALTER TABLE Nume_tabela DISABLE CONSTRAINT nume [CASCADE]

In cele trei exemple sunt dezactivate doua constrangeri prin invocarea definitiei si una prin precizarea numelui asociat. Optiunea CASCADE are acelasi efect cu cea de la stergerea unei constrangeri, adica dezactivarea suplimentara a tuturor constrangerilor dependente.

Pentru activarea unei constrangeri este utilizata o sintaxa similara cu cea de la dezactivare in care se inlocuieste cuvantul cheie DISABLE cu ENABLE si nu poate fi invocata optiunea CASCADE.

ALTER TABLE Nume_tabela ENABLE PRIMARY KEY
ALTER TABLE Nume_tabela ENABLE UNIQUE (lisata_coloane)
ALTER TABLE Nume_tabela ENABLE CONSTRAINT nume

Obs: La activarea unei constrangeri se verifica consistenta datelor si daca constrangerile nu sunt conforme cu datele din tabela constrangerea nu va fi activata producand mesaj de eroare.

Operatii aditionale

O tabela poate fi golita de toate inregistrarile printr-o comanda DDL. Acest lucru este diferit fata de stergerea tuturor inregistrarilor ca urmare a unei instructiuni DML. Marea diferenta consta in faptul ca golirea tabelei nu poate fi revocata fiind o operatie cu AUTOCOMMIT. In plus, la golirea unei tabele poate fi specificat modul in care este gestionat spatiul de stocarea pentru inregistrarile tabelei. Astfel

TRUCATE TABLE Nume_tabela [REUSE STORAGE]

are ca efect stergerea inregistrarilor tabelei cu numele specificat, iar daca este prezenta si optiunea REUSE STORAGE se indica faptul ca spatiul ocupat de liniile sterse ramane alocat tabelei si poate fi utilizat pentru inserari ulterioare, altfel spatiul ocupat de inregistrari este eliberat. In exemplu se arata golirea tabelei Intretinut cu pastrarea spatiului de stocare aferent.

TRUCATE TABLE Intretinut [REUSE STORAGE]

Numele tabelelor poate fi modificat prin redenumirea acestuia utilizand comanda RENAME.

RENAME Nume_vechi TO Nume_nou

Daca dorim redenumirea tabelei Departament cu numele Dep vom scrie:

RENAME Departament TO Dep

Atat tabelelor cat si campurilor li se poate asocia comentariu cu instructiunea COMMENT ON. Pentru comentarii asociate tabelelor avem:

COMMENT ON TABLE Nume_tabela IS ‘text asociat’

La comentarii asociate campurilor

COMMENT ON COLUMN Nume_tabela.Nume_coloana IS ‘text’

Aceste informatii sunt pastrate la mediul Oracle in USER_TA_COMENTS, cele pentru tabele, respectiv in USER_COL_COMMENTS, pentru coloane si in ALL_COL_COMMENTS, pentru toate comentariile.

Limbajul cererilor in SQL (SQL-DML)

SQL-DML implementeaza operatiile algebrei relationale pentru realizarea limbajului de cereri. Astfel, SQL foloseste o declaratie de baza pentru cautarea informatiilor intr-o baza de date, declaratia *SELECT*. Se mentioneaza ca declaratia SELECT nu este relativa la operatia SELECT din algebra relationala descrisa in capitolul 6. Este momentul punctarii unor deosebiri importante intre SQL si modelul relational prezentat in capitolul 6. SQL permite existenta intr-o tabela a doua sau mai multe n-upluri identice din punctul de vedere al valorilor atributelor. Deci, in general rezultatul unei operatii SQL si chiar tabelele nu sunt vazute ca un set de n-upluri cu membrii diferiti, ci un multiset de n-upluri. Includerea unei constrangeri de tip PRIMARY KEY in definirea unei tabele asigura unicitatea fiecarei inregistrari. Rezultatul operatiilor SQL pot sa nu inclusa campuri cheie fapt pentru care in rezultat se regasesc duplicate. Desigur ca pot fi eliminate duplicatele daca utilizatorul doreste. O fraza SQL este compusa din mai multe clauze si este vazuta ca o singura instructiune. Inainte de executie componenta SQL a mediului de baze de date analizeaza cererea si decide care este cea mai adecvata executie a sa. Se spune ca cererea este optimizata pentru executie de componenta numita si optimizator SQL. Calitatea acestei componente diferentiaza foarte mult mediile de baze de date in ceea ce priveste timpul necesar pentru executia interogarilor. O fraza SQL are urmatoarea structura generala ingloband mai multe clauze:

```
SELECT lista_campuri si functii
FROM lista_tabele sau alte obiecte
[WHERE conditii de tip select, join]
[GROUP BY lista campuri grupare]
[HAVING conditii pe functii]
[ORDER BY lista campuri grupare]
```

In aceasta structura numai clauzele SELECT, FROM sunt obligatorii in sensul ca trebuie sa existe in orice cerere valida, existentra celorlalte clauze fiind dependenta de natura cererii. In plus, ordinea clauzelor ce urmeaza dupa SELECT si FROM poate fi modificata. Asa cum se vede in structura frazei SQL cererile se aplica asupra obiectelor din baza de date de tip tabela sau a celor generate cu o structura similara cu tabela. Ca urmare, rezultatul executiei unei cereri este tot o structura de tip tabela. O cerere SQL este formata din mai multe clauze, fiecare clauza incepe cu un cavant cheie, prima clauza determina si tipul cererii. Vom examina pe rand aplicarea principalelor cereri SQL incepand cu cererile simple care se aplica asupra unei singure tabele, cereri pentru mai multe tabele, subcereri.

Cereri SQL la o singura tabela

O forma simpla a unei cereri SQL aplicate pentru o singura tabela poate sa contina patru clauze. Nu tratam inca utilizarea functiilor statistice ce pot aduce in structura cererii alte 2 clauze.

```
SELECT [DISTINCT] lista de expresii
FROM nume_tabela
[WHERE conditie_coloane]
[ORDER BY criterii_sortare_rezultat]
```

In cerere este specificata o singura tabela la clauza FROM, ca urmare operatiile se efectueaza pe aceasta tabela si la executie:

- Se parcurg toate liniile tabelei specificate la clauza FORM;
- Se evalueaza logic conditia/conditiile din clauza WHERE. Daca rezultatul evaluarii este TRUE se adauga liniile la rezultat, iar daca rezultatul este FALSE nu sunt adaugate in rezultat. Daca WHERE lipseste evaluarea este totdeauna TRUE si in rezultat sunt adaugate toate liniile;
- Rezultatul are structura de tabela, numarul coloanelor si tipul de data asociat acestora este dat de lista de expresii precizata in clauza SELECT;
- Cuvantul cheie optional DISTINCT are ca efect eliminarea duplicatelor din rezultat, daca exista. Daca DISTINCT lipseste in rezultat se vor pastra duplicatele;
- Daca este prezenta si clauza ORDER BY, rezultatul este sortat functie de criteriile specificate in aceasta clauza. Daca clauza lipseste, ordinea este de regula data de ordinea logica a inregistrarilor din tabela initiala, dar nu este garantata.

Caracteristicile rezultatului:

- Numarul coloanelor este egal cu numarul expresiilor din clauza SELECT, ordinea acestora este ordinea epresiilor clauzei si tipul de date corespunzator mostenit;
- Numarul liniilor rezultatului este egal cu numarul liniilor din tabela specificata la clauza FROM ce indeplinesc conditia specificata la clauza WHERE;
- Parcurgerea liniilor tabelei specificate este facuta de catre serverul de baze de date si nu se garanteaza ordinea liniilor in rezultat daca nu se utilizeaza si clauza ORDER BY.

Un prim exemplu, cererea de ontinere a Numelui si Prenumelui tuturor angajatilor companiei, se scrie:

```
SELECT Nume, Prenume
FROM Angajat
```

Cererea returneaza o tabela cu doua campuri (Nume, Prenume) si un numar de linii identic cu numarul de linii din tabela Angajat. Daca se doreste afisare tuturor coloanelor tabelei specificate la clauza FROM se va utiliza caracterul *, cu semnificatia, toate. Sunt medii de baze de date care accepta fie * fie ALL. Ordinea fizica a coloanelor in rezultat este data de ordinea fizica a coloanelor tabelei.

```
SELECT *
FROM Angajat
```

Operatiile efectuate pana la acest moment sunt echivalente cu operatii ale algebrei relationala Project.

Un alt exemplu simplu, in care se doreste obtinerea datei de nastere si a adresei tuturor angajatilor cu Salariul mai mare decat 800, necesita introducerea si a clauzei WHERE.

```
SELECT  Data_n,Adresa
FROM    Angajat
WHERE   Salariu > 800
```

Operatia reprezinta a combinatie Select, Project a algebrei relationale, echivalenta cu scrierea.

$P_{Data\_n,Adrresa}(S_{Salariu > 800}$ (ANGAJAT))

Lista de expresii a clauzei SELECT specifica atributele operatiei Project, conditiile din clauza WHERE specifica operatia Select a algebrei relationale. Cu toate ca din punct de vedere al modului de operare cele doua expresii de mai sus sunt echivalente rezultatele pot fi diferite intrucit in rezultatul unei operatii SQL pot exista duplicate si nu a fost utilizat cuvantul cheie DISTINCT.

Invocarea constantelor

In cererile SQL de manipularea datelor pot fi invocate constante numerice, sir de caractere sau valori NULL. Daca de xemplu se executa cererea de mai jos:

```
SELECT ‘Salariul persoanei ’, Nume, Prenume, ‘ este ‘, Salariu
FROM Angajat
```

se va genera ca rezultat o structura de tabela cu 5 coloane in care prima coloana are numele specificat de constanta si continutul insasi constanta ‘Salariul persoanei’, coloana 4 cu numele constantei ‘este’ si valoarea constanta ‘este’, iar coloanele 2, 3 si 5 aduc informatii din tabela Angajat in urma parcurgerii acesteia linie cu linie. Trebuie remarcat faptul ca o constanta de tip sir de caractere este cuprinsa intre ghilimele.

Expresii aritmetice

Elementele ce apar ca argumente in clauza SELECT pot fi expresii complexe continand functii valide in mediul de baza de date in care se executa cererea cat si operatori. Operatori uzuali sunt cei care specifica operatii simple: *, /, +, -. Daca se doreste sa se vizualizeze efectul unei mariri de salariu cu valoarea 100 si rezultatul marit cu 10% se poate apela la cererea:

```
SELECT Nume, Prenume, Salariu, (Salariu + 100)*1.1
FROM Angajat
```

Ca rezultat se obtine o tabela cu 4 coloane, trei dintre ele cu informatii culese din liniile tabelei Angajat, iar coloana 4 o coloana calculata dupa expresia precizata.

Expresii concatenate

Un operator uzual este si operatorul de concatenare (||) care permite crearea de coloane ce culeg informatii dintr-o expresie ce contine constante si rezultate ale operatiilor efectuate asupra coloanelor. Sa consideram un exemplu similar cu cel de mai sus in care dorim sa concatenam constante cu valori din cadrul liniilor inregistrarilor tabelei Angajat.

```
SELECT ‘Salariul persoanei ’ || Nume || ‘ ‘ || Prenume || ‘ este ‘ || Salariu || ‘ lei’
FROM Angajat
```

Cererea genereaza o tabela cu un singur camp al carui nume este dat de expresia utilizata in clauza SELECT. Cum expresia contine un numar mare de caractere este posibil ca aceasta sa exceada numarul de caractere permise la definirea unui camp si ca urmare este necesara redefinirea numelui sau. Acest lucru este posibil prin introducerea de ALIAS la coloana, ca in exemplul urmator.

```
SELECT ‘Salariul persoanei ’ || Nume || ‘ ‘ || Prenume || ‘ este ‘ || Salariu || ‘ lei’ AS
Descriere
FROM Angajat
```

Cuvantul cheie AS poate fi omis fara a vea efect asupra rezultatului, adica SELECT ……. ‘ lei’ Descriere

Eliminarea duplicatelor

In lista de campuri sau expresii asociata clauzei SELECT nu totdeauna sunt utilizate campuri cheie si ca urmare pot fi obtinute in rezultat inregistrari identice. Implementarile SQL pastreaza duplicatele obtinute in urma executiei. Ca exemplu cererea:

SELECT Slalariu FROM Angajat

va genera o multime de inregistrari identice daca in companie sunt angajati cu acelasi salariu, informatia obtinuta nefiind consistenta. Utilizarea cuvantului cheie DISTICT face ca din rezultat sa fie eliminate duplicatele.

SELECT DISTINCT Slalariu FROM Angajat

Utilizarea clauzei WHERE

Toate cererile specificate pana la acest moment se executa pentru toate inregistrarile din tabela. Specificarea unei conditii de selectie poate fi realizata prin introducerea clauzei WHERE urmata de o expresie_logica, expresie ce este evaluata pentru fiecare inregistrare a tabelei, aducand in rezultat numai acele inregistrari pentru care rezultatul evaluarii expresiei este TRUE. Cererea:

SELECT Nume, Prenume FROM Angajat WHERE D_nr=4

aduce in rezultat numai numele si prenumele angajatilor pentru care valoarea campului D_nr este egala cu 4. In clauza WHERE pot fi utilizati o serie de operatori relationali, cei uzuali fiind:

- Egal =
- Mai mare >
- Mai mare sau egal >=
- Mai mic <
- Mai mic sau egal <=
- Diferit <> != ^=

Pot fi formulate conditii complexe legate prin operatorii logici AND, OR, NOT, cat si paranteze de grupare care sa asigure precedenta dorita a operatiilor. Conditia:

((D_nr = 3 or D_nr=4)and Salariu >400) or (D_nr = 5 and Salariu*1.1<=3000)

va extrage date din liniile in care D-nr este 3 sau 5 si valoarea campului Salariu este mai mare decat 400 sau liniile pentru care D_nr=5 si salariul inmultit cu 1,1 este mai mic sau egal cu 3000

Operatorul BERWEEN

In clauza WHERE se poate utiliza si operatorul BETWEEN prin care se verifica apartenenta unei valori la un interval precizat. Conditia se specifica:

expresie BETWEEN valoare_minima AND valoare_maxima

Exemplu:

```
SELECT Nume, Prenume, Salariu
FROM Angajat
WHERE Salariu >300 and D_nr BETWEEN 2 AND 5
```

va obtine numele si prenumele angajatilor cu Salariu mai mare decat 300 si care fac parte din departamentele avand D_nr intre 2 si 5, interval inchis.

Operatorul IN

Un alt operator utilizabil in clauza WSHERE este operatorul IN, operator care testeaza apartenenta unei valori la o multime secificata. Conditia

expresie IN (val_1, val_1,…,val_N)

intoarce valoarea TRUE daca expresia din membrul stang este una dintre valorile specificate val_1, val_1,…,val_N. De exemplu

```
SELECT Nume, Prenume, Salariu
FROM Angajat
WHERE Salariu >300 and D_nr IN (2, 4, 6)
```

aduce numele, prenumele si salariul angajatilor cu Salariu mai mare de 300 si care fac parte din departamentele cu numerele 2, 4 sau 6.

In lista de valori a operatorului IN pot fi incluse si valorile speciale NULL, NOT NULL. Evaluarea conditiei fiind facuta logic, operatorul IN de apartenenta la lista poate fi negat prin prefixarea conditiei cu particula NOT, rezultand forma: NOT IN (lista_valori)

Operatorul LIKE

Operatorul este utilizat pentru a verifica daca valoarea unei expresii de tip sir de caractere respecta un anumit sablon, prin formularea:

expresie LIKE ‘SABLON’ [ESCAPE ‘caracter’]

Sablonul este marginit de apostrofi si poate contine caracterele speciale _ si %:

Caracterul _ are ca efect inlocuirea unui singur caracter din pozitia curenta;

 Caracterul special % inlocuieste orice sir de caractere, inclusiv un sir vid. Exemplu:

```
SELECT Nume, Prenume FROM Angajat
WHERE UPPER(Prenume) LIKE 'A%A'
```

Va verifica daca dupa conversia la litere mari a campului Prenume valorile acestuia incep si se sfarsesc cu litera A.

Operatorul IS NULL

Valorile NULL au proprietati specifice si nu pot fi utilizate in expresii cu operatori de egalitate sau diferit. Pentru a compara daca valoarea unui camp este nula se va utiliza IS NULL, respectiv pentru diferit de null IS NOT NULL. Pentru a determina numele si prenumele angajatilor cu nu au supervizor este necesar sa se verifice daca valoarea campului Sssn este nula. Conditia se scrie:

```
SELECT Nume, Prenume FROM Angajat
WHERE Sssn IS NULL
```

Clauza ORDER BY

Daca intr-o cerere nu se specifica o clauza ORDER BY, de regula, nu se garanteaza ordinea inregistrarilor in rezultat. Pentru a specifica o anumita ordine pentru inregistrarile in rezultatul produs de o cerere se va utiliza clauza ORDER BY. In aceasta clauza pot fi invocate atat nume de coloane sau extresii ce apar in rezultat cat si nume de coloane/expresii ce nu apar in rezultat. Coloanele de ordonare sunt tratate in ordinea specificata la clauza ORDER BY, criteriul de ordonare implicit fiind cel ascendent (ASC), iar particula DESC specifica ordine descrescatoare. Se face ordonarea dupa prima expresie specificata, daca sunt inregistrari cu aceeasi valoare a acestei expresii se trece la urmatoarea expresie specificata si asa mai departe pana la ultima expresie. Daca prima expresie specificata la ORDER BY este construita pe baza unui camp cheie primara, atunci celelate expresii nu au nici un efect intrucat nu pot exista valori identice pentru cheia primara.

```
SELECT Nume, Prenume, D_nr, Salariu
From Angajat
WHERE D_nr IN (1,3,5) and Salariu >200 and Sssn IS NULL
ORDER BY D_nr ASC, Salariu DESC, Nume
```

Cererea de mai sus va ordona rezultatul crescator dupa valorile campului D_nr, in cadrul aceluiasi departament descrescator dupa valorile campului Salariu si daca si salariul este egal, crescator dupa nume. Atentie: la ordonarea sirurilor de caractere, literele mari au coa ASCII mai mic decat literele mici. O particularite la clauza ORDER BY consta in faptul ca se poate utiliza in loc de expresia care defineste campul rezultatului si numarul coloanei/expresiei in ordinea in care aceasta apare la clauza SELECT. Cererea anterioara este rescrisa utilizand numarul coloanelor rezultatului astfel.

```
SELECT Nume, Prenume, D_nr, Salariu
```

```
From Angajat
WHERE D_nr IN (1,3,5) and Salariu >200 and Sssn IS NULL
ORDER BY 3 ASC, 4 DESC, 1
```

Cereri SQL la mai multe tabele

Pe scurt aceste cereri au urmatoarele particularitati:

- Implementeaza operatiile algebrei relationale produs cartezian sau JOIN. Sunt medii de baze de date ce nu suporta operatia de produs cartezian datorita faptului ca este extrem de consumatoare de resurse;
- In clauza FROM vom avea o lista de tabele;
- Clauza WHERE va contine pe langa conditii de tip select ale algebrei relationale si conditii de corelare a inregistrarilor din diverse tabele numite si conditii de tip JOIN;
- Daca o conditie specifica o egalitate intre valorile coloanelor ce definesc conditia JOIN, ea se mai numeste si equjoin;
- O conditie de tip JOIN ce nu specifica egalitate se spune ca este de tip non-equjoin;
- Daca coloanele ce participa la o operatie equjoin au acelasi nume conditia se va numi natural join;
- Trebuiesc luate in consideratie si situatiile in care se adauga in rezultat si inregistrari ce nu indeplinesc conditia JOIN numita si join extern (outer join).

Operatia este produs cartezian atunci cand in clauza WHERE nu se specifica nici o conditie intre campurile tabelelor, ca in exemplul"

```
SELECT *
FROM Angajat, Departament
WHERE D_nr = 4
```

Vom incepe cu cereri simple care specifica conditii de tip equjoin. Sa presupunem ca dorim combinarea inregistrarilor tabelei Angajat cu cele ale tabelei departament dar numai pentru departamentul din care face parte angajatul, fiind adus in rezultat numele si prenumele angajatului, numele si locatia departamentului din care acesta face parte..

```
SELECT Nume, Prenume, D_nume, D_location
FROM Angajat, Departament
WHERE D_nr = Dep_nr
```

In cerere nu s-a dorit obtinerea tuturor campurilor tabelelor ci doar a celor specificate. Daca cele doua campuri ar avea acelasi nume in tabelele Angajat si Departament operatia este natural JOIN, dar din punctul de vedere al rezultatului nu este nici o diferenta. In cazul de fata fiind utilizate in cerere doar doua tabele si campurile avand nume diferit conditia de join este corect interpretata nefiind nici o ambiguitate. Sunt situatii, atunci cand se fac operatii cu mai multe tabele, ca o conditie sa devina ambigua datorita existentei campurilor cu acelasi nume in mai multe tabele. Tehnica uzuala de eliminarea ambiguitatii este cea de utilizare a unui ALIAS pentru tabela. Exista doua modalitati de utilizare ALIAS.

- Alias implicit – caz in care acesta este identic cu numele tabelei. Acest alias va prefixa numele campurilor specificand in mod univoc care este sursa de date.

```
SELECT Angajat.Nume, Angajat.Prenume, Departament.D_nume,
       Departament.D_location
FROM Angajat, Departament
WHERE Angajat.D_nr = Departament.Dep_nr
```

In acest caz nu este nevoie sa se prefixeze fiecare camp cu alias implicit.

- Alias explicit – caz in care se atribuie un alt nume logic tabelei, nume ce are efect local si nu este vazut in afara cererii.

```
SELECT A.Nume, A.Prenume, D.D_nume, D.D_location
FROM Angajat A, Departament D
WHERE A.D_nr = D.Dep_nr
```

In exemplu a fost atribuit aliasul A tabelei Angajat si aliasul D tabelei Departament. La utilizarea unui astfel de alias este obligatorie prefixarea fiecarui camp din cerere au aliasul definit, altfel se produce o eroare. Tabela este un obiect al bazei de date ce poate fi invocat la diverse operatii de tip join, chiar join intre o tabela si ea insasi (self_join). In acest caz un alias explicit este foarte util pentru a diferentia obiectele invocate in cerere si aceeasi tabela este vazuta ca doua instante diferite la executia cererii. De exemplu, obtinerea mumelui si prenumelui fiecarui angajat si al supervizorului sau este o operatie care necesita join intre tabela Angajat si ea insasi.

```
SELECT   A.Nume, A.Prenume, S.Nume Nume_s, S.Prenume
         Prenume_s
FROM     Angajat A, Angajat S
WHERE    A.Sssn=S.Ssn
```

Vom discuta exemple in care mai mult de doua tabele sunt precizate in cadrul unei cereri. De exemplu, cererea de listare pentru fiecare proiect localizat in 'Brazi' a numarului proiectului, numarului departamentului ce il coordoneaza, numelui, adresei si datei de nastere a managerului departamentului.

```
SELECT   P_nr, Dep_nume, Nume, Ini, Pren, Adr, Dat_na
FROM     Proiect, Departament, Angajat
WHERE    Proiect.Dep_nr=Departament.Dep_nr AND Dep_man=Ssn AND
P_loc='Brazi‘
```

Join extern (outher join)

Sunt situatii in care se doreste ca pe langa inregistrarile ce indeplinesc conditia de join intre doua sau mai multe tabele sa fie cuprinse si inregistrari ale uneia dintre tabele (sau mai multor) ce nu

indeplinesc conditia de join, inregistrari ce sunt combinate cu inregistrari cu valori nule din tabelele corespondente. Ca exemplu, cererea de listare a numelor si prenumelor angajatilor impreuna cu cele ale supervizorilor lor, iar pentru angajatii ce nu au supervizor se va lista doar numele si prenumele angajatilor, iar numele si prenumele supervizorilor va fi NULL.

```
SELECT    A.Nume, A.Prenume, S.Nume Nume_s, S.Prenume
          Prenume_s
FROM      Angajat A, Angajat S
WHERE     A.Sssn (+)=S.Ssn
```

Marcarea cu semnul (+) la conditia de join indica operatia join aditional, marcarea specifica tabela din care se aduc valorile NULL. O marcare la ambii termeni ai conditiei de join este echivalenta cu operatia join aditional complet. Join extern poate fi invocat si cu alte conditii decat egalitate, BETWEEN, LIKE

Utilizarea functiilor in SQL

In orice sistem de baze de date in cererile SQL se pot utiliza o serie de functii oferite de sistem. Acestea se pot utiliza fie pentru a forma expresii corecte, specifice clauzei in care sunt apelate, fie fac parte intrinseca din structura cererii SQL. In functie de numarul de linii pentru care se calculeaza rezultatul, functiile se impart in doua mari categorii:

- Functii pentru ***linia curenta***, functii care se aplica liniei curente din tabela. Tipul de data al argumentelor functiei, precum si tipul de data al rezultatului impart functiile in urmatoarele categorii:
    - Functii cu argamente si rezultat numerice;
    - Functii cu argumente siruri. Au ca argumente siruri de caractere, rezultatul poate fi sir de caractere sau numeric;
    - Functii cu argumente date calendaristice. Au ca argumente data calendaristica, rezultatul este de tip data calendaristica sau de tip numeric;
    - Functii cu efect de conversie a tipului de data. Sunt caracterizate de faptul ca tipul de data al argumentelor este diferit de tipul de data al rezultatului;
    - Functii avand ca argumente diverse tipuri de data.
- **Functii de grup**, cunoscute si sub numele de functii statistice. Aceste functii se pot aplica pe integ continutul unei tabele sau numai pe anumite inregistrari ale tabelei, inregistrari ce indeplinesc o anumita conditie specificata. Principalele functii din aceasta categorie sunt:
- MIN (expresie) – intoarce valoarea minima a valorilor expresiei specificate;
- MAX (expresie) – intoarce valoarea maxima a valorilor expresiei specificate;
- AVG (expresie) – intoarce valoarea medie a valorilor expresiei specificate;
- SUM (expresie) – intoarce suma valorilor expresiei specificate;
- COUNT (expresie) – intoarce numarul de valori ale expresiei specificate;
- STDDEV (expresie) – calculeaza deviatia standard a valorilor expresiei specificate;
- VARIANCE (expresie) – calculeaza varianta valorilor expresiei specificate;

Functii aplicabile liniei curente

Aceste functii au cateva caracteristici comune ce sunt reprezentate prin:

- Numarul de argumente este fixat prin definitia functiei, acest numar difera de la functie la functie;
- Un argument al functiei poate fi un nume de coloana, o constanta sau o pseudocoloana; Un argument poate fi o expresie ce contine un nume de coloana, constanta sau o alta functie.

Functii cu argumente numerice

Aceaste functii accepta argumente de tip numeric si produc rezultate de tip numeric. In general sunt functii cu un singur argument, dar pot avea si mai multe argumente, dependent de definitie. Vom exemplifica cateva dintre cele mai importante functii din aceasta categorie.

- TRUNC(argument_n [,n]) are ca rezultat un intreg daca n lipseste sau este 0, respectiv un numar cu n digiti dupa punctul zecimal daca argumentul optional n este prezent. Daca argumentul optional n este negativ truncherea se face pastrand partea intreaga;
- ROUND(argument_n [,n]) rotunjire la intreg daca n lipseste sau este 0, respectiv la un numar de digiti precizat prin n dupa punctul zecimal. Rotunjirea se face la cel mai apropiat numar cu numarul de zecimale specificat;
- MOD(arg_n1, arg_n2) determina restul impartirii arg_n1 la arg_n2.

Urmatorul exemplu ilustreaza utilizarea acestor functii intr-o interogare SQL:

```
SELECT Nume, Prenume, TRUNC(Salariu*1.1111, 2) Trunchiat,
        ROUND(Salariu*1.1111, 2) Rotunjit, MOD(Salariu, 5) Rest
FROM Angajat
```

Cererea produce o structura cu 5 campuri (Nume, Prenume, Trunchiat, Rotunjit, Rest) in care Numele si Prenumele sunt preluate din tabela Angajat, Trunchiat, Rotunjit si Rest se obtin prin calcul pornind de la valorile campului Salariu, primele doua fiind rezultatul truncherii la 2 zecimale, respectiv rotunjirea la 2 zecimale, al treilea fiind restul impartirii la 5 a campului Salariu.

- CEIL(arg_n) si FLOOR(arg_n) returneaza cel mai mic intreg mai mare decat argumentul, respectiv cel mai mare intreg mai mic decat argumentul. Astfel, CEIL(1.8) = 2 si FLOOR(1.8) = 1
- Alte functii uzuale sunt cele trigonometrice, logaritm, exponentiala, sign, power, modul. Functia ABS(arg_n) returneaza valoarea absoluta a argumentului, SIGN(arg_n)-semnul, EXP(arg_n) – valoarea lui e la puterea arg_n, SQRT(arg_n) – valoarea radacinii patrate din arg_n, SIN(arg_radiani), COS(arg_radiani), TAN(arg_radiani) – functii trigonomertrice cu argument exprimat in radiani, SINH(arg_n), COSH(arg_n), TANH(arg_n) – functii hiperbolice aplicate arg_n, POWER(arg_n1,arg_n2) – arg_n1 la puterea arg_n2, LOG(arg_baza, arg_n) – logaritm din arg_n in baza arg_baza. Se observa ca dintre functiile specificate, unele au un singur argument altele doua argumente.

Pentru ca o operatie ce apeleaza o anumita functie sa fie valida este necesar ca volorile argumentelor sa respecte domeniile de definitie ale functiilor.

Functii cu argumente siruri de caractere

La aceste functii argumentele pot fi constante de tip sir de caractere, nume de coloane cu tip de data sir de caractere, expresii ce returneaza sir de caractere. Daca un argument nu respecta aceasta conditie el va fi convertit automat, la sir de caractere, daca este posibil. Si aici vom enumera cateva dintre functiile uzuale:

- LOWER(arg_c), UPPER(arg_c), INITCAP(arg_c) sunt functii care au ca efect conversia arg_C in litere mici, litere mari, respectiv litera de inceput litera mare;
- CONCAT(arg_c1, arg_c2) returneaza sirul de caractere rezultat prin concatenarea arg_c1 cu arg_c2, functie ce este echivalenta cu operatorul || utilizat in cererile anterioare;
- LENGTH(arg_c) returneaza numarul de caractere continut in arg_c, rezultat de tip numeric;
- SUBSTR(arg_c, start [,nr_car]), functia extrage un subsir din sirul arg_c, incepand de la pozitia specificata prin start si avand lungimea nr_car. Primul argument este de tip sir caractere pe cand celelalte sunt de tip numeric. Daca al treile argument, optional (nr_car) lipseste se extrage subsirul de la pozitia specificata pana la sfarsitul sirului;
- INSTR(arg_c, subsir [,start]) cu rezultat de tip numeric returneaza pozitia subsirului specificat ca la doilea argument in sirul specificat prin arg_c, pozitie numarata de la inceputul sirului daca argumentul optional start lipseste sau de la pozitia marcata prin argumentul numeric start;
- LPAD (arg_c, dimensiune, [,subsir]) si RPAD(sir, dimensiune, [,subsir]) realizeaza umplerea la stanga sau la dreapta a sirului specificat prin arg_c cu subsirul specificat in al treilea argument sau blank, daca argumentul lipseste, pana la atingerea dimensiunii specificate;
- TRIM ([LEADING|TRAILING|BOTH] [caracter] FROM arg_c), o functie cu o sintaxa aparte cu scopul de eliminare la stanga, la dreapta sau din ambele parti a caracterului specificat pana la intalnirea unui caracter diferit. Daca nu se specifica caracterul ce trebuie eliminat sunt eliminate caracterele blank;
- REPLACE (arg_c, subsir [,subsir_nou]), functia care trebuie sa returneze un sir rezultat din sirul specificat prin arg_c in care s-a inlocuit subsirul specificat prin al doilea argument cu subsirul nou specificat prin al treilea argument, subsir_nou; Lungimea celor doua subsiruri este in general diferita. Daca subsir_nou nu este specificat efectul este eliminarea subsirului precizat de al doilea argument din sirul specificat prin arg_c;
- SOUNDEX (arg_c) intoarce un cod alfanumeric corespunzator sonoritatii valorii argumentului in limba engleza.

Exemplu: Presupunem ca politica formarii adreselor de mail pentru angajatii companiei este de a concatena numele angajatului cu sirul ‘@adr_domeniu.ro’. Pentru a obtine lista adreselor de mail ale angajatilor din departamentul cu numarul 5, impreuna cu numele si prenumele angajatului vom formula cererea:

```
SELECT Nume, Prenume, CONCAT(Nume, '@adr_domeniu.ro') Adr_mail
 FROM Angajat
 WHERE D-nr=5
```

Functii cu argumente date calendaristice

La stocarea datelor calendaristice cele mai multe medii de baze de date pastreaza informatia despre data complet: secol, an, luna, zi, ora, minut, secunda. Cu informatii de tip data calendaristica pot fi efectuate o serie de operatii, cum sunt:

- Diferenta a doua date calendaristice;
- Adaugarea unui numar la o data calendaristica;
- Scaderea unui numar dintr-o data calendaristica;
- Adunare si scadere de ore din date calendaristice utilizand o constructie de impartire la "/24", pentru a genera date corecte.

Cateva functii cu argumente de acest tip:

- ROUND(arg_d [, 'MONTH' | 'YEAR']), functie ce rotunjeste o data calendaristica precizata de arg_d la precizia de luna sau an asa cum se precizeaza in cel de al doilea argument;
- TRUNC(arg_d [, 'MONTH' | 'YEAR']), similar cu round insa nu se face rotunjire ci trunchere;
- NEXT_DAY(arg_d [, numar zi | 'nume_zi']), obtinerea datei primei zile cu numarul specificat sau numele specificat dupa data specificata de arg_d;
- ADD_MONTH(arg_d, numar_luni) returneaza o data calendaristica obtinuta prin adaugarea numarului de luni specificat la data calendaristica specificata de arg_d.

Functii de conversie

Exista o multitudine de functii de conversie de la un tip de data la alt tip de data. Obisnuit, o serie de conversii sunt efectuate automat atunci cand o variabila sau o coloana necesita o astfel de conversie pentru operatiile in care este implicata, daca acest lucru este posibil. Sunt posibile si conversii fortate prin utilizarea functiilor specifice:

- TO_CHAR(arg_n[,'sablon']), converteste valoarea specificata de argumentul numeric (arg_n) in sir de caractere conform sablonului. Aceasta conversie este posibila indiferent de valoarea argumentului numeric;
- TO_NUMBER (arg_c[,'sablon']), converteste valoarea specificata de argumentul sir de caractere (arg_c) in valoarea numerica. Aceasta conversie nu este posibila pentru orice valoare a argumentului numeric;
- TO_DATE(arg_c[,'sablon']), pentru conversia un sir compatibil cu o data calendaristica in data calendaristica.

Functii cu argumente de tipuri diferite

O caracteristica specifica acestor functii consta in faptul ca tipul si numarul parametrilor poate fi diferit de la apel la apel. Unele dintre functii nu au numar de parametrii fixat, acesta fiind determinat de apel. Cateva dintre aceste functii sunt descrise mai jos:

- NVL(expr1, expr2) testeaza daca expr1 este nula si returneaza expr1 daca este nenula si expr2 daca expr1 este nula. Cele doua expresii pot avea orice tip de data;
- NVL2(expr1, expr2, expr3) este o versiune mai generala a celei anterioare, testeaza expr1 si returneaza expr2 daca expr1 este nenula, respectiv expr3 altfel;
- COLEASCE(lista expresii), functie cu o multime de argumente neprecizata care returneaza prima valoare nenula din lista de expresii;
- GRATEST(lista expresii) returneaza cea mai mare valoare din lista de expresii. Expresiile nu pot avea orice tip de date, ci trebuie avut in vedere ca tipul datelor pentru expresii sa poata fi ordonate;
- LEAST(lista expresii) similara cu GRATEST, dar reurneaza cea mai mica valoare din lista de expresii;
- DECODE(expr, test1, rez1, test2, rez2,…., testn, rezn) este o functie cu oricate argumente prin care argumentul aflat pe prima pozitie este supus testelor notate test1, test2, .., testn si returneza expresia corespunzatoare rezultatului asociat testului care este evaluat TRUE. Testarea este facuta in ordine secventiala si la primul test valid se asigneaza valoarea fara a efectua toate testele pana la capat;
- CASE expresie WHEN test1 THEN rez1 [WHEN test2 THEN rez2.. ….. [ELSE rez_implicit]] END, este o functie similara cu DECODE asignand rezultatul dupa validitatea testului. Se observa ca exista si optiunea ELSE pentru cazul in care nici un test nu este evaluat la TRUE.

Functii statistice

Functiile statistice se aplica numai pe multimi de valori. La calculul rezultatului pot fi luate un consideratie sau nu valorile duplicate sau nule. Ca regula generala:

- Toate functiile cu exceptia functiei COUNT(*) ignora valorile nule;
- Fara a preciza altfel (implicit), se iau in consideratie toate valorile duplicat, pentru a lua in consideratie numai valorile distincte expresia va trebui prefixata de cuvantul cheie DISTINCT.

Sintaxa pentru functiile MIN si MAX este cea de mai jos:

MIN([ALL|DISTINCT] expresie)
MAX([ALL|DISTINCT] expresie)

in care prin expresie se precizeaza domeniul asupra caruia se aplica functia. In mod uzual acesta este o expresie ce include numele unui camp al tabelei. Valorile nule sunt ignorate, astfel ca functiile se aplica numai valorilor diferite de nul. Daca nu este precizata nici o optiune din categoria ALL|DISTINCT sunt luate in consideratie toate valorile duplicat fara a avea nici un efect asupra rezultatului. Cele doua optiuni sunt pastrate numai pentru uniformitate cu celelalte functii fara sa

aiba efecte asupra rezultatelor in cazul de fata. Functiile pot fi aplicate pentru expresii cu orice tip de data pentru care exista criterii de ordonare. Cererea ce obtine salariul minim si salariul maxim al angajatilor companiei este data ca exemplu:

```
SELECT MIN(Salariu) minim, MAX(salariu) maxim FROM Angajat
```

Pentru functiile SUM si AVG sintaxa este similara cu cea de la functiile anterioare insa pot fi aplicate numai la expresii cu tip de data numeric.

```
SUM ([ALL|DISTINCT] expresie_n)
AVG ([ALL|DISTINCT] expresie_n)
```

Utilizarea parametrilor optionali ALL, respectiv DISTINCT are efecte asupra rezultatului in sensul ca utilizarea cuvantului cheie DISTINCT face ca suma si media sa fie calculata numai pentru valorile distincte ale expresiei, in schimb prefixarea cu ALL sau lipsa acesteia are ca efect luarea in consideratie a tuturor duplicatelor. In nici una dintre forme nu sunt luate in consideratie valorile nule. Pentru a ilustra diferenta dintre utilizare DISTINCT si fara utilizarea sa in cererea de mai jos se determina suma tuturor salariilor in suma1, suma salariilor distincte in suma2, media salariilor in media1 si media salariilor distincte in media2, operatii efectuate doar pentru angajatii din departamentele 2 sau 4.

```
SELECT SUM(Salariu) suma1, SUM(DISTINCT Salariu) suma2, AVG(Salariu) media1,
AVG(DISTINCT Salariu) media2
    FROM Angajat
    WHERE D_nr=2 or D_nr=4
```

***Obs*:** Nu totdeauna rezultatul functiei SUM este valid, intrucat la o tabela cu numar mare de inregistrari si valori mari ale expresiei este posibil ca rezultatul sa depasasca domeniul maxim de reprezentare permis de mediul de baze de date.

Functia COUNT are interpretare diferita in sensul ca rezultatul poate fi afectat si de valorile nule. Astfel daca se apeleaza functia COUNT (*), rezultatul reprezinta numarul de linii ale tabelei, pe cand COUNT (expresie), echivalenta cu COUNT(ALL expresie) numai acele linii pentru care expresia este diferita de nul. Prefixarea expresiei cu atributul DISTINCT face ca numararea se se realizeze numai pentru liniile cu valori diferite ale expresiei. Functia COUNT poate fi aplicata pentru orice tip de data. In rezumat pentru COUNT exista urmatoarele forme:

- COUNT(*) intoarce numarul de linii;
- COUNT(expresie) intoarce numarul de valori nenule ale expresiei;
- COUNT(DISTINT expresie) returneaza numarul de valori nenule si distincte.

Functiile STDDEV([ALL|DISTINCT] expresie) si VARIANCE([ALL|DISTINCT] expresie) sunt aplicabile pentru expresii numerice si returneaza deviatia standard, respectiv varianta valorilor expresiei.

Daca o cerere SQL utilizeaza functii din aceasta categorie rezultatul este format dintr-o singura linie, operatia de calcul fiind efectuata pentru toate liniile tabelei la care se aplica, linii ce indeplinesc conditia specificata in clauza WHERE. Daca nu se precizeaza o clauza WHERE functiile se calculeaza pentru toate liniile tabelei. In acest caz spunem ca functiile aplicate fac parte din categoria functii agregat.

Utilizarea acelorasi functii, dar pe grupuri de inregistrari poate fi realizata in conjunctie cu clauza GROUP BY. Prin clauza GROUP BY se precizeaza una sau mai multe coloane de grupare, cu efect de calcul a functiilor pentru liniile cu aceleasi valori ale coloanelor de grupare. In conjunctie cu GROUP BY numarul de linii este egal cu numarul valorilor distincte ale coloanelor de grupare. Sa consideram ca dorim sa obtinem pentru fiecare departament salariul minim, salariul maxin si media salariilor, ordonate dupa numarul departamentului. Crerea SQL care produce acest lucru este:

```
SELECT D_nr, MAX(Salariu), MIN(Salariu), AVG(Salariu)
 FROM Angajat
 GROUP BY D_nr
 ORDER BY D_nr
```

Daca in rezultat se doreste si numele departamentului, atunci cererea va trebui sa apeleze la o operatie de tip join intre tabelele Angajat si Departament.

```
SELECT D_nr, D_nume, MAX(Salariu), MIN(Salariu), AVG(Salariu)
 FROM Angajat, Departament
 WHERE D_nr=Dep_nr
 GROUP BY D_nr
 ORDER BY D_nr
```

Rezultatul are atatea linii cate valori distincte are campul D_nr. In exemplu functiile iau in consideratie duplicatele.

In cererile SQL pot fi necesare conditii de filtrare pentru rezultatele functiilor calculate. Astfel de conditii nu pot fi specificate in clauza WHERE, clauza care este evaluata inainte de executie, motiv pentru care a fost introdusa clauza HAVING, clauza ce este evaluata dupa executie. O clauza de tip HAVING nu are sens la functii agregat, intrucat produc o singura linie. Clauza HAVING este utilizata in conjunctie cu GROUP BY avand ca efect filtrarea rezultatului dupa valorile produse de functii. Ca exemplu, la cererea de obtinere a salariului minim, salariului maxim si a mediei salariilor pe departamente se doreste filtrarea rezultatului si obtinerea numai a acelor linii pentru care media salariilor este mai mare decat 1000.

```
SELECT D_nr, MAX(Salariu), MIN(Salariu), AVG(Salariu)
 FROM Angajat
 GROUP BY D_nr
 HAVING AVG(Salariu) >1000
```

```
ORDER BY MIN(Salariu)
```

Functiile agregat sau de grup pot fi utilizate in cereri complexe combinate cu operatii de tip join. Sa consideram ca exemplu cererea de listare a numelui si prenumelui angajatilor care lucreaza mai putin de 40 ore la proiectele coordonate de departamentul din care angajatul face parte:

```
SELECT Nume, Prenume, SUM(Ore)
FROM Angajat, Lucreaza, Proiect
WHERE Angajat.Ssn = Lucreaza.Ssn AND Angajat.D_nr = Proiect.D_nr AND
Proiect.P_nr = Lucreaza.P_nr
GROUP BY Lucreaza.Ssn
HAVING SUM(Ore) < 40
```

Pentru realizarea cererii este necesar join intre Angajat si Lucreaza dupa ssn, join intre Angajat si Proiect pentru ca proiectul sa fie coordonat de departamentul din care face parte angajatul, respectiv join intre Proiect si Lucreaza ca operatia de insumare sa fie efectuata numai la proiectele coordonate de departamentul din care face parte angajatul.

Subcereri

SQL suporta o varietate de structuri de interogare, structuri care pot utiliza in interiorul unei cereri SQL o alta cerere SQL. O subcerere este o cerere SELECT inclusa intr-o alta cerere SQL. Aceste constructii se folosesc atunci cand rezultatul dorit nu se poate obtine cu o singura parcurgere a datelor. De exemplu daca se doreste obtinerea angajatilor care lucreaza la proiecte un numar de ore mai mare decat media orelor prestate de angajati la proiecte este necesara obtinerea mediei orelor dupa care se obtin datele despre angajati. Intrucat subcererile pot sa apara in toate clauzele cererii SQL principale, evaluarea lor este diferita functie de clauza in care subcererea apare: WHERE si HAVING fac evaluarea subcererii ca expresii logice;

- ORDER BY caz in care ordonarea se face dupa rezultatul unei subcereri;
- SELECT valoare returnata de subcerere este prezenta in rezultatul final;
- FROM rezultatul este asimilat cu o tabela temporara ce participa la cererea principala ca o tabela a bazei de date.

Intrucat o subcerere este tratata ca o cerere SQL, functie de rezultatul returnat de subcerere acestea se impart in:

- Subcereri ce returneaza o singura valoare (o coloana si o singura inregistrare);
- Subcereri ce returneaza o coloana (o coloana cu zero sau mai multe inregistrari);
- Subcereri ce returneaza o tabela (cel putin doua coloane cu zero sau mai multe inregiatrari)

Subcereri ce returneaza o singura valoare

Aceste subcereri intorc rezultate ce pot fi utilizate in cererea principala similar cu utilizarea de constante. Rezultatul subcererii este vazut ca o valoare ce poate fi comparata cu valorile altei expresii cu acelasi tip de data. Sa consideram cererea care doreste sa obtina angajatul sau angajatii companiei care lucreaza la un proiect un numar de ore egal cu numarul maxim de ore desfasurate

la proiecte. Cererea nu poate fi scrisa ca o singura interogare intrucat mai intai trebuie determinat numarul maxim de ore efectuate de un angajat la un proiect.

```
SELECT Nume, Prenume
FROM Angajat, Lucreaza
WHERE Angajat.ssn=Lucreaza.ssn and Ore = (SELECT Max(Ore)
                                          FROM Lucreaza)
```

In operatiile de comparatie in care este implicata subcererea aceasta trebuie sa se gaseasca in membrul drept si este cuprinsa intre paranteze. Intr-o cerere pot fi utilizate mai multe subcereri impreuna cu oricare dintre operatorii permisi cererilor principale. Ilustram un exemplu in care se doreste obtinerea numelui si prenumelui angajatilor care lucreaza la cel putin un proiect un numar de ore cuprins intre 70% si 120% din media orelor efectuate la proiecte de catre angajatii companiei. In acest caz se va utiliza operatorul BETWEEN si doua subcereri identice.

```
SELECT Nume, Prenume
FROM Angajat, Lucreaza
WHERE Angajat.ssn=Lucreaza.ssn and Ore BETWEEN ((SELECT Max(Ore)*0.7
        FROM Lucreaza) and (SELECT Max(Ore)*1.3 FROM Lucreaza))
```

Subcereri care returneaza o coloana

Datele din coloana returnata de subcerere sunt asimilate cu o multime. Un element poate fi sau nu membru al acestei multimi. O astfel de subcerere poate fi implicata intr-o operatie de verificarea apartenentei valorilor unei coloane la rezultatul returnat de subcerere. Operatorul ce se utilizeaza in acest caz este fie IN, fie negatul sau NOT IN.

```
SELECT Nume, Prenume
FROM Angajat, Lucreaza
WHERE Angajat.ssn=Lucreaza.ssn and
P_nr IN (SELECT P_nr FROM  Proiect WHERE D_nr =3)
```

Conditia in care este implicata subcererea se interpreteaza ca fiind TRUE daca valoarea campului P_nr de la linia curenta apartine listei da valori ale coloanei returnate de subcerere. Este evident ca tipul de data al coloanei din membrul stang trebuie sa fie identic cu tipul de data al coloanei. Cererea anterioara obtine numele si prenumele angajatilor care lucreaza la proiecte coordonate de departamentul cu numarul 3. Subcererile ce returneaza o coloana pot fi utilizate si in conjunctie cu operatorii de prefixare SOME, ANY, ALL. In aceasta situatie intre valoarea expresiei ce constituie membrul stang se specifica o operatie de comparatie, testul fiind realizat pentru unele sau toate valorile liniilor aferente coloanei din membrul drept. Interpretarea operatorilor:

- SOME si ANY conditie adevarata daca macar o valoare din cele returnate de subcerere indeplineste conditia de comparatie;
- ALL conditia este adevarata daca toate valorile returnate de subcerere indeplineste conditia de comparatie.

Cererea

```
SELECT Nume, Prenume
FROM Angajat, Lucreaza
WHERE Angajat.ssn=Lucreaza.ssn and
Ore > ALL (SELECT Ore
                FROM  Proiect, Lucreaza
                WHERE Proiect.D_nr=4 and Lucreaza.P_nr = Proiect.P_nr)
```

Returneaza numele si prenumele angajatilor cre lucreaza la proiecte un numar de ore mai mare decat oricare dintre numarul de ore prestate la proiectele coordonate de departamentul cu numarul 4 de oricare dintre angajatii companiei. Acelasi rezultat s-ar fi obtinut cu cererea:

```
SELECT Nume, Prenume
FROM Angajat, Lucreaza
WHERE Angajat.ssn=Lucreaza.ssn and
Ore > (SELECT MIN(Ore)
                FROM  Proiect, Lucreaza
                WHERE Proiect.D_nr=4 and Lucreaza.P_nr = Proiect.P_nr)
```

Daca se doreste lista angajatilor care lucreaza la proiecte un numar de ore mai mare decat macar una dintre numarul de ore efectuate la proiectele coordonate de departamentul cu numarul 3, cererea se va scrie:

```
SELECT Nume, Prenume
FROM Angajat, Lucreaza
WHERE Angajat.ssn=Lucreaza.ssn and
Ore > ANY (SELECT Ore
                FROM  Proiect, Lucreaza
                WHERE Proiect.D_nr=3 and Lucreaza.P_nr = Proiect.P_nr)
```

***Obs**:* O eroare frecventa este cea de prefixarea cu operatorii ANY, SOME, ALL a subcererilor in cazul in care operatie este de tip egalitate sau chiar apartenenta.

Subcereri care returneaza o tabela

Am numit subcereri care returneaza o tabela acele subcereri carea au in rezultat cel putin doua coloane si zero sau mai multe linii. Similar cu subcererile care returneaza o coloana acestea pot fi utilizate in clauza WHERE in conjunctie cu operatorul IN. Forma uzuala pentru utuilizare este: WHERE (lista_expresii) IN (subcerere). Pentru corectitudine aceasta forma trebuie sa indeplineasca urmatoarele conditii:

- Lista de expresii si subcererea trebuiesc incadrate intre paranteze rotunde;

- Numarul de coloane din subcerere trebuie sa fie egal cu numarul de expresii din lista de expresii;
- Intre tipurile de date ale expresiilor si cea a coloanelor returnate de subcerere trebuie sa fie corespondenta pozitionala, eventual conversie automata daca tipurile de date sunt compatibile pentru conversie;
- Verificarea apartenentei la multimea intoarsa de subcerere este facuta pozitional intre elementele expresiei si coloanele subcererii;
- Daca rezultatul subcererii nu contine inregistrari conditia este evaluata ca fiind falsa.

Ca exemplu, pentru a obtine lista angajatilor care lucreaza la un singur proiect, suma tuturor orelor efectuate de un angajat la proiecte este egala cu orele efectuate la singurul proiect la care lucreaza. In cererea de mai jos se verifica daca perechea (Ssn, Ore) se gaseste printre liniile intoarse de subcerere, linii ce contin informatiile Ssn si SUM(Ore), suma fiind facuta cu clauza GROUP BY dupa SSN.

```
SELECT Nume, Prenume, Ore
FROM Angajat, Lucreaza
WHERE Angajat.Ssn=Lucreaza.Ssn and
        (Ssn, Ore) IN (SELECT Ssn, SUM(Ore)
                       FROM Lucreaza
                       GROUP BY Ssn)
```

Daca o linie contine o valoare null sau o expresie din lista contine valoarea null rezultatul evaluarii este fals. Vom reveni la astfel de subcereri in conjunctie cu operatorul EXISTS.

Subcereri in HAVING

Expresii logice ce contin subcereri pot fi utilizate si in clauza HAVING in mod similar cu cele utilizate in clauza WHERE. La aceste cereri conditiile vor contine doar elemente ce pot sa apara in clauza HAVING, adica: constante, expresii de grupare, functii statistice. Ca exemplu, cererea de determinarea a numelui, prenumelui, a numarului minin si numarului maxim de ore desfasurate de angajati la proiecte, pentru proiectele la care media numarului de ore este mai mare decat media numarului de ore calculata pentru toate proiectele companiei, se poate scrie:

```
SELECT Nume, Prenume, MIN(Ore), MAX(Ore)
FROM Angajat, Lucreaza
WHERE Angajat.ssn = Lucreaza.ssn
GROUP BY P_nr
HAVING AVG(Ore) > (SELECT AVG(ore) FROM Lucreaza)
```

In exemplu, subcererea returneaza o singura valoare, media tuturor orelor efectuate de angajati la proiecte, valoare care este comparata cu media orelor efectuate de fiecare angajat la proiecte. Un alt exemplu, similar, doreste sa obtina numele si prenumele, precum si media orelor desfasurate de un angajat la proiecte, dar numai daca aceasta medie este mai mare decat cea mai mare medie

a orelor efectuate pe proiect. Pentru aceasta, subcererea returneaza o singura valoare MAX(AVG(ore)) in care AVG(Ore) este calculat cu GROUP BY P_nr.

```
SELECT Nume, Prenume, AVG(Ore)
FROM Angajat, Lucreaza
WHERE Angajat.ssn = Lucreaza.ssn
GROUP BY P_nr
HAVING AVG(Ore) > (SELECT MAX(AVG(ore)) FROM Lucreaza GROUP BY
                   P_nr)
```

Subcereri in clauza FROM

Subcererile din caluza FROM pot produce o valoare, o coloana sau o tabela, in oricare dintre cazuri rezultatul fiind tratat ca o tabela temporara asupra careia se aplica cererea principala. In primul exemplu subcererea returneaza toate campurile tabelelor Angajat, Lucreaza si Proiect cu respectarea conditiilor de join specificate si cu conditiile ca valorile campului ore sa fie cuprins intre 6 si 20 si departamentul ce coordoneaza proiectul are numarul 5. Din acesta se extrage numele si prenumele angajatilor. Pe ansamblu cererea returneaza numele si prenumele angajatilor care lucreaza la proiectele coordonate de departamentul cu numarul 5 intre 6 si 20 ore.

```
SELECT Nume, Prenume
FROM (SELECT * FROM Angajat A, Lucreaza B, Proiect C
      WHERE A.ssn = B.ssn and B.P_nr = C.P_nr and C.Dep_nr = 5 and
       Ore BETWEEN 6 and 20)
```

Aceasi rezultat se obtine prin scrierea unei singure cereri ca mai jos:

```
SELECT A.Nume, A.Prenume FROM Angajat A, Lucreaza B, Proiect C
      WHERE A.ssn = B.ssn and B.P_nr = C.P_nr and C.Dep_nr = 5 and
       Ore BETWEEN 6 and 20)
```

Observatii:

- Daca mai multe tabele sunt invocate in cererea principala unei subcereri trebuie sa i se asocieze un alias pentru ca altfel nu pot fi referite fara ambiguitati campurile returnare de subcerere.
- Rezultatele subcererilor sunt tratate similar cu tabelele permanente, ele pot fi implicate si in operatii SQL-3

Subcereri corelate

In toate exemplele anterioare subcererea se executa o singura data, dupa care rezultatul este utilizat in evaluare. Sunt situatii in care rezultatul unei subcereri este dependent de valorile liniei curente, caz in care sunt numite si subcereri corelate. Astfel, daca se doreste determinarea angajatilor care lucreaza la un proiect un numar de ore mai mare decat media numarului de ore lucrat de angajatii la acelasi proiect este nevoie ca la fiecare proiect sa se calculeze media orelor.

```
SELECT A.Nume, A.Prenume, B.Ore
FROM Angajat A, Lucreaza B
WHERE A.ssn = B.ssn and Ore > (SELECT AVG(ore)
        FROM Lucreaza
        WHERE B.P_nr=P_nr)
```

Se observa ca in subcerere se face o operatie de join intre tabela deschisa in subcerere si o valoare data de inregistrarea curenta a tabelei din cererea principala. Pentru fiecare inregistrare ce indeplineste conditia de join A.Ssn = B.Ssn valoarea P_nr este transmisa subcererii prin B.P_nr solicitand calculul punctajului mediu pe proiect. Subcererea produce o singura valoare ce va fi comparata cu valoarea campului ore al inregistrarii curente. Daca rezultatul evaluari este adevarat se include in rezultat. Amintesc cu aceasta ocazie ca daca intr-o cerere s-a definit un alias pentru o tabela, acesta trebuie utilizat pentru prefixarea aricarei coloane in cerere.

Subcereri pe clauza ORDER BY

In ultimele versiuni SQL clauza ORDER BY poate contine o subcerere corelata ce intoarce o singura valoare. O cerere necorelata nu duce la ordonarea rezultatului, intrucat daca rezultatul este o constanta are aceeasi valoare pentru fiecare linie supusa sortarii. Sa presupunem ca dorim sa obtinem angajatii companiei ordonati descreascator dupa numarul persoanelor aflate in intretinere. Pentru asta vom utiliza o subcerere corelata, ce se executa la fiecare linie a cererii principale in clauza ORDER BY avand criteriu de ordonare descrescator.

```
SELECT Nume, Prenume
FROM Angajat
ORDER BY (SELECT COUNT(*) FROM Intretinut
        WHERE Angajat.ssn=Intretinut.ssn) DESC
```

Subcereri pe clauza SELECT

Ca si la cele din clauza ORDER BY aceste subcereri trebuie sa intoarca o singura valoare, rolul lor fiind de a popula un anumit camp al rezultatului. Ca exemplu, consideram cererea de listare a numelui si prenumelui angajatilor si a numarului persoanelor aflate in intretinere, ordonati crescator dupa numarul persoanelor in intretinere. Subcererea corelata din clauza SELECT determina pentru fiecare angajat din cererea principala numarul persoanelor in intretinere, si in plus aceeasi conditie este utilizata la ordonare.

```
SELECT Nume, Prenume (SELECT COUNT(*) FROM Intretinut
        WHERE Angajat.ssn=Intretinut.ssn)
FROM Angajat
ORDER BY (SELECT COUNT(*) FROM Intretinut
        WHERE Angajat.ssn=Intretinut.ssn)
```

O varianta similara dar mai putin consumatoare de timp este cea in care se specifica numarul coloanei de ordonare, fara a fi executata subcererea si in clauza ORDER BY.

```
SELECT Nume, Prenume (SELECT COUNT(*) FROM Intretinut
                      WHERE Angajat.ssn=Intretinut.ssn)
FROM Angajat
ORDER BY 3
```

Operatorul EXISTS

In anumite situatii este important a detecta daca rezultatul unei subcereri contine sau nu inregistrari. Operatorul EXISTS testeaza daca subcererea primita ca argument intoarce rezultat nevid. Daca se doreste obtinerea angajatilor companiei ce au persoane in intretinere, se poate formula o interogare ce are inclusa o subcerere care face join intre inregistrarea curenta din Angajat cu tabela Intretinut, rezultatul continand inregistrari numai daca conditia de join este adevarata cel putin odata.

```
SELECT Nume, Prenume
FROM Angajat
WHERE EXISTS (SELECT *
                    FROM Intretinut
                    WHERE Angajat.ssn = ssn)
```

Operatii UNION, INTERSECT, MINUS

Aceste operatii sunt implementarile operatiilor specifice multimilor, cum sunt operatiile: REUNIUNE, INTERSECTIE, DIFERENTA. Operatorii utilizati sunt UNION pentru REUNIUNE, INTERSECT pentru INTERSECTIE si MINUS pentru DIFERENTA. Acesti operatori permit combinarea prin operatii de REUNIUNE, INTERSECTIE, DIFERENTA a rezultatelor mai multor cereri, operatii valide numai daca rezultatele cererilor sunt compatibile, adica au acelasi numar de coloane si tipurile de date sunt corespondente pozitional. Cum numele expresiilor clauzei SELECT poate fi diferit, coloanele sunt denumite de argumentele primei cereri SELECT. Clauza ORDER BY nu poate fi utilizata la subcererile componente, fiind permisa utilizarea acestei clauze o singura data la sfarsitul cererii. Cererea de producere a numelui si prenumelui angajatilor care lucreaza in departamentul cu numarul 3, sau in departamentul cu numarul 5 si care au un salariu mai mare de 400 se poate scrie:

```
(SELECT Nume, Prenume FROM Angajat WHERE D_nr=3
      UNION
SELECT Nume, Prenume FROM Angajat WHERE D_nr=5)
      INTERSECT
SELECT Nume, Prenume FROM Angajat WHERE Salariu>400
ORDER BY Nume
```

Obs: Cererea putea fi scrisa simplu si fara utilizarea acesor operatori ca in exemplul:

```
SELECT Nume, Prenume FROM Angajat WHERE (D_nr=5 or D_nr=3)
and Salariu >400
```

Aceste operatii sunt extrem de utile si nu pot fi ocolite daca datele intre care se fac operatiile privin din surse diferite. Pentru a exemplifica sa presupunem ca tabela Intretinut are coloanele Nume_i, Prenume_i si celelalte coloane specificate la definirea sa si se doreste sa se obtina numele si prenumele angajatilor care au in intretinere pe cineva cu acelasi nume si prenume.

```
SELECT Nume, Prenume FROM Angajat
        INTERSECT
SELECT Nume_i, Prenume_i FROM Intretinut, Angajat
WHERE Intretinut.Ssn=Angajat.Ssn
```

Actualizarea datelor in SQL

Toate interogarile DML prezentate pana la acest moment nu produc modificari ale datelor stocate in baza de date, ci doar produc rezultate calculate in urma cererilor formulate. Este momentul sa vedem cum se produc si care sunt mecanismele privind asigurarea consistentei datelor la operatiile ce au ca efect modificarea continutului bazei de date. Dependent de efect, operatiile de modificare pot fi grupate in:

- Adaugarea de linii intr-o tabela, realizata prin operatia generica INSERT;
- Stergerea de linii dintr-o tabela, realizata cu operatia DELETE;
- Modificarea valorilor unor campuri ale anumitor inregistrari dintr-o tabela, prin operatia UPDATE;
- Operatii INSERT sau UPDATE functie de indeplinirea unei anumite conditii, realizate prin MERGE;
- Gestionarea tranzactiilor prin COMMIT, ROLLBACK, SAVEPOINT. Asa cum se va discuta mai departe, operatiile de modificare in SQL sunt operatii tranzactionale in sensul ca ele pot fi revocate si efectul modificariloe este pastrat in memorie pana cand tranzactia este incheiata. De fapt una dintre cele mai importante facilitati ale mediilor de baze de date mutiuser se refera la gestiunea concurentei pentru pastrarea integritatii datelor.

Adaugarea de linii (INSERT)

Pentru adaugarea de noi linii intr-o tabela sunt posibile mai multe abordari: inserarea unei linii complete, sau numai anumite campuri ale liniei sa fie precizate altele fiind NULL. Sintaxa uzuala pentru INSERT este:

```
INSERT INTO nume_tabela [(lista_coloane)]
        VALUES (lista_valori)
```

Se observa ca lista de coloane este optionala, daca aceasta lipseste efectul este urmatorul:

- Se insereaza o linie completa dupa inregistrarea curenta a tabelei;
- Numarul valorilor ce trebuiesc precizate in VALUES este egal cu numarul de coloane al tabelei;
- Ordinea valorilor in VALUES si tipul de data este acelasi cu ordinea fizica a coloanelor definita la crearea tabelei (corespondenta pozitionala);
- Daca se doreste ca valoarea pentru o coloana sa fie nula se va specifica la pozitia corespunzatoare NULL;
- Daca pentru o coloana care are definita o valoare implicita se doreste acea valoare se specifica DEFAULT. Daca valoarea este precizata in tabela se va stoca acea valoare;
- Inserarea mai multor linii cu aceasta sintaxa necesita cate o cererea separata pentru fiecare linie.

Ca exemplu, cererile:

INSERT INTO Lucreaza VALUES (‘1561124402273’, 27, 13)
INSERT INTO Lucreaza VALUES (‘1561124400273’, 27, NULL)
INSERT INTO Lucreaza VALUES (‘1561124400273’, DEFAULT, 15)

adauga cate o linie in tabela Lucreaza, tabela in care coloanele sunt Ssn de tip sir caractere, P_nr de tip numeric si Ore tot de tip numeric. In prima cerere sunt precizate valori pentru toate cele trei coloane, in a doua s-a optat pentru valoarea NULL la cea de a treia coloana, iar in ultima valoarea DEFAULT pentru a doua coloana. Pentru ca o operatie INSERT de acest tip sa fie valida valorile inserate trebuie sa respecte constrangerile impuse la definirea tabelei:

- Daca un camp cheie primara are o valoare egala cu valoarea stocata la alta inregistrare sau NULL, operatia esueaza;
- Daca un camp unic are o valoare egala cu valoarea stocata la alta inregistrare, operatia esueaza;
- Un camp NOT NULL nu poate avea pentru linia inserata valoarea NULL;
- Un camp cheie straina trebuie sa aiba o valoare care se regaseste in tabela de referinta, in corespondenta cu setul de conditii impuse (no action, cascade, set null...);

De cele mai multe ori la adaugarea unei inregistrari se specifica o lista de coloane atat la inserarea unei linii complete cat si la inserarea de linii incomplete. La inserarea de linii complete specificarea listei coloanelor face cererea de inserare independenta de ordinea fizica a coloanelor tabelei. Coloanele neprecizate in aceasta structura vor avea valoarea DEFAULT, daca aceasta exista, sau valoare NULL in caz contrar. Pentru ca operatia sa fie corecta numarul coloanelor din lista trebuie sa fie egal cu numarul de valori si la fel tipul de data in corespondenta pozitionala. Motivele pentru care o operatie INSERT de acest tip poate fi invalida sunt identice cu cele prezentate mai sus. Ca exemplu, cererea de insert in tabela Angajat specificata mai jos, precizeaza valori numai pentru 4 din campurile tabelei.

INSERT INTO Angajat (Nume, Prenume, Ssn, D_nr) VALUES (‘Popescu’, ‘Ion’,

‘1541124400273’, 4)

Obs: In operatia INSERT poate fi utilizat si rezultatul unei cereri necorelate care intoarce o singura valoare in lista de valori, ca in exemplul:

```
INSERT INTO Departament (D-nr, D_nume) VALUES((SELECT MAX(D_nr)
    FROM Departament)+1, “Cercetare”)
```

Inserarea rezultatului unei cereri

De foarte multe ori se doreste ca printr-o singura cerere sa se insereze mai multe linii intr-o tabela. Acest lucru este posibil numai daca valorile ce sunt inserate sunt produse de o interogare (subcerere) aplicata altor tabele. In aceasta situatie sintaxa cererii de adaugare de linii la tabela este:

```
INSERT INTO nume_tabela [(lista_coloane)] cerere_select
```

Similar cazurilor anterioare, daca lista de coloane lipseste se considera implicit toate coloanele ce trebuiesc sa fie in corespundenta pozitionala cu rezultatul returnat de cererea SELECT. Din punctul de vedere al executiei acestei operatii si al conditiilor de validitate, avem urmatoarele observatii:

- Se executa cererea SELECT;
- Liniile rezultatului returnate de SELECT sunt inserate in tabela;
- Daca o linie a rezultatului nu restecta constrangerile asociate, intreaga cerere de INSERT va esua.

Ca exemplu consideram ca am creat o tabela Dep_info in care dorim sa pastram pentru fiecare departament, numarul de angajati si media salariului angajatilor departamentului. Valorile inserate in tabela sunt rezultatul unei cereri aplicate tabelei Angajat, rezultatul functiilor fiind grupat dupa D_nr.

```
INSERT INTO Dep_info (Dep_nr, Nr_ang, Medie_sal)
SELECT D-nr, Count(ssn), AVE(Salariu)
    FROM Angajat
    GROUP BY D_nr
```

Obs: In locul unui nume de tabela poate fi folosita o subcerere, asa cum se va discuta la paragraful WITH CHECK OPTION. Aceste subcereri trebuie sa satisfaca restrictiile privind actualizarea datelor prin intermediul vederilor.

Stergere de linii (DELETE)

Operatia DELETE permite stergerea uneia sau mai multor inregistrari, ce indeplinesc conditia specificata din tabela precizata. Forma generala a cererii DELETE este:

```
DELETE FROM nume_tabela
[WHERE conditie]
```

Clauza WHERE specifica conditii similare cu cele care apar in clauza WHERE intr-o cerere SELECT si chiar subcereri. Daca nu exista clauza WHERE sunt sterse toate liniile tabelei, operatie cu efect similar TRUNCATE. Pentru stergerea angajatilor din departamentul cu numarul 5 cererea se va formula:

```
DELETE FROM Angajat WHERE D_nr=5
```

Daca se doreste stergerea tuturor angajatilor care lucreaza la proiecte un numar total de ore mai mic decat 40 cererea va utiliza in clauza WHERE o subcerere, ca in exemplul:

```
DELETE FROM Angajat
    WHERE Ssn IN (SELECT Ssn FROM Lucreaza
        GROUP BY Ssn
        HAVING SUM(Ore)<40)
```

Avem mai jos un exemplu in care s-a utilizat BETWEEN.

```
DELETE FROM Angajat
    WHERE ssn IN (SELECT ssn FROM Lucreaza
        WHERE Ore BETWEEN 2 and 10)
```

Similar cu operatia INSERT cererea poate produce o eroare sau efecte aditionale ce duc la modificari daca stergerea afecteaza linii cu o cheie referita printr-o constrangere de tip FOREIGN KEY intr-o alta tabela din baza de date, functie de modul in care a fost precizata actiunea la stergere.

Actualizarea datelor (UPDATE)

O operatie uzuala de actualizare a datelor este si cea numita UPDATE. Sintaxa cererii UPDATE este:

```
UPDATE nume_tabela
SET col1=expresie1 [, col2=expresie2, …. ] [WHERE
conditie]
```

in care: la clauza SET numarul perechilor col=expresie este data de numarul coloanelor ce se actualizeaza, conditia WHERE determina care linii sunt afectate de actualizare. Urmatoarele observatii asupra efectului operatiei sunt valide:

- In clauza WHERE pot sa apara aceleazi conditii ca la clauza similara din cererile SELECT;

- Daca clauza WHERE lipseste se actualizeaza toate liniile, altfel numai cele care indeplinesc conditia;
- O coloana nu poate sa apara in SET de mai multe ori.

Pentru actualizarea salariului angajatilor din departamentul cu numarul 4 in urma unei mariri de 10% vom scrie:

```
UPDATE Angajat SET Salariu=Salariu*1.1 WHERE D_nr=4
```

Cererile de mai jos folosesc pentru actualizare valorile DERFAULT si respectiv NULL:

```
UPDATE Angajat SET Adresa =  DEFAULT
UPDATE Angajat SET Functia = NULL
```

Atat in clauza SET cat si in clauza WHERE pot fi utilizate subcereri. In cererea urmatoare se actualizeaza coloana Nr_ang valoarea fiind rezultatul unei subcereri corelate, iar clauza WHERE utilizeaza alta subcerere corelata pentru a determina liniile afectate de modificare:

```
UPDATE Dep_info
SET Nr_ang = (SELECT Count(ssn) FROM Angajat WHERE
D_nr=Dep_info.Dep_nr)
        WHERE Dep_nr IN (SELECT D_nr FROM Angajat
                GROUP BY D_nr HAVING count(*) >3)
```

Similar cu celelalte operatii cu efect modificarea continutului bazei de date, anumite actualizari pot esua datorita nerespectarii constrangerilor definite pentru tabela, cateva dintre aceste cazuri fiind:

- Noua valoare este NULL si este activa constrangerea NOT NULL;
- Noua valoare pentru un camp cheie primara sau valoare unica se regaseste in tabela;
- Noua valoare nu verifica o conditie de tip CHECK;
- Vechea valoare este referita prin constrangeri de tip FOREIGN KEY si actiunea nu permite modificarea sa;
- Noua valoare nu se regaseste in tabela referita prin FOREIGN KEY si actiunea nu permita validarea valorii.

Cereri MERGE

O cerere de tip MERGE efectueaza fie actualizarea fie inserarea unor linii intr-o tabela dupa cum indeplineste sau nu conditia de join specificata. Sintaxa cererii MERGE este mai complexa:

```
MERGE INTO nume_tabela1 [alias tabela1]
        USING nume_tabela2 | nume_vedere | subcerere
        ON (conditie join)
```

```
WHEN MATCHED THEN
        UPDATE SET col1=expr1 [,col2=expr2, ….]
WHEN NOT MATCHED THEN
        INSERT [(lista_coloane)] VALUES (lista_valori)
```

unde:

- INTO nume_tabela1 [alias tabela] specifica tabela in care operatia de insert sau update este efectuata. Este numita si tabela destinatie;
- USING nume_tabela2 | nume_vedere | subcerere specifica un alt obiect al bazei de date utilizat la operatia de join. Acest obiect este fie tabela, fie o vedere fie rezultatul unei subcereri. In toate cazurile obiectul produs este similar unei tabele. Acest obiect este numit si sursa intrucat la modificare el este sursa de date;
- Clauza ON precizeaza o conditie de join ce trebuie indeplinita intre cele doua obiecte ale bazei invocate la clauzele anterioare;
- UPDATE SET specifica coloana/coloanele tabelei destinatie care se modifica si care este noua valoare din tabela destinatie daca conditia de join este indeplinita;
- INSERT VALUES specifica valorile inserate daca nu este indeplinita conditia de join din tabela sursa. Coloanele tabelei destinatie trebuie sa fie in corespondenta pozitionala cu cele ale tabelei sursa.

Exemplul de mai jos ilustreaza operatie de tip MERGE la tabela Dep_info in care se face join cu rezultatul cererii de la clauza USING. Daca conditia de join este indeplinita la fiecare departament se actualizeaza numarul de angajati cu rezultatul intors de subcerere, iar daca conditia de join nu este indeplinita, adica a aparut un departament nou in tabela Dep_info se insereaza o inregistrare cu noul departament si numarul angajatilor acestuia.

```
MERGE INTO Dep_info
USING (SELECT D_nr, COUNT(*) AS Nr FROM Angajat GROUP BY D_nr)
ON (Dep_nr=D_nr)
WHEN MATCHED THEN
        UPDATE SET Dep_info.nr_ang = Nr
WHEN NOT MATCHED THEN
        INSERT VALUES (D_nr, Nr)
```

Controlul concurentei

Sistemele de baze de date actuale trebuie sa gestioneze accesul concurent al mai multor utilizatori la aceleasi tabele. Operatiile cerute de acesti utilizatori se impart in:

- Operatii de citire – sunt operatiile care interogheaza baza de date pentru a obtine diverse rezultate, operatii realizate de regula prin cereri de tip SELECT;
- Operatii de scriere – prin care continutul tabelelor este modificat, realizate prin INSERT, UPDATE, DELETE, MERGE

Daca utilizatorii executa doar operatii de citire din baza de date nu sunt probleme legate de pastrarea consistentei tabelelor, toti utilizatorii au acces simultan la aceeasi informatie. Problemele

devin mult mai complicate atunci cand unul sau mai multi utilizatori cer operatii de scriere in tabele. Pentru pastrarea consistentei sunt necesare mecanisme de gestiune si control a modificarilor. Toate operatiile de modificare intr-o tabela sunt realizate in buffere de memorie si devin permanente sau sunt revocate in urma COMMIT sau ROLLBACK. Mecanismul uzual este cel de blocare implicita la scriere, modificarile nu sunt vizibile de alti utilizatori pana la scrierea definitiva in baza de date prin COMMIT. Toate modificarile efectuate inainte de COMMIT pot fi revocate complet sau partial. Toate blocarile sunt ridicate la revocare sau la COMMIT Prin blocare la scriere intelegem blocarea la scriere numai a inregistrarilor ce sunt afectate de actualizare, celelalte inregistratri nefiind blocate in nici un fel. Daca o inregistrare este blocata la scriere, o alta modificare a aceleiasi inregistrari este in asteptare pana cand prima modificare este incheiata pri revocare sau comitere. Pentru clarificare vom considera urmatorul scenariu privind modificarile concurente efectuate de mai multi utilizatori asupra unei tabele.

Fie urmatoarea succesiune de cereri la aceeasi tabela in ordine temporara:

Moment de timp	Utilizator	Operatie
T1	U1	UPDATE
T2>T1	U1	Vizualizare (vede modificarile facute la momentul T1 prin UPDATE)
T3>T2	U2	Vizualizare (nu vede modificarile facute de T1 prin UPDATE)
T4>T3	U3	UPDATE
T5>T4	U1	Vizualizare (vede modificarile facute la momentul T1 prin
		UPDATE, nu vede modificarile facute la momentul T4 de U3)
T6>T5	U1	Executa COMMIT (datele modificate de U1 sunt scrise permanent si modificarile facute de U3 sunt in asteptare pentru ca U3 nu a incheiat tranzactia.
T7>T6	U2	Vizualizare (vede modificarile facute de U1 si comise, nu le vede pe cele facute de U3)

Se defineste notiunea de tranzactie o seccesiune de operatii efectuate asupra bazei de date ce sunt fie incheiate prin scrierea in baza, fie revocate. Numai operatiile SQL-DML sunt tranzactionale in sensul ca se pot comite sau revoca, operatiile DDL si DCL nu pot fi vazute ca tranzactii si nu pot fi revocate, adica se executa direct. Urmatoarele actiuni pot fi intreprinse asupra tranzactiilor nefinalizate:

- La inchiderea oricarei sesiuni de lucru de catre un utilizator COMMIT este executat automat;
- Inainte de COMMIT o tranzactie poate fi revocata total sau partial;
- La caderea tensiunii de alimentare sistemul executa automat ROLLBACK;

- Se poate face COMMIT automat daca sistemul este setat corespunzator. Se poate invoca o setare SET AUTOCOMMIT ON care face ca toate tranzactiile sa fie comise la executie, valoarea implicita fiind OFF.

In succesiunea operatiilor componente unei tranzactii pot fi marcate puncte aditionale de revenire. Pentru marcarea acestor puncte se apeleaza SAVEPOINT nume_punct. Cu aceste puncte marcate se poate apela la comanda de revocare partiala sau totala ROLLBACK TO [SAVEPOINT] nume_punct. Ca exmplu:

```
        UPDATE .....
SAVEPOINT nume_punct1
INSET .....
        SAVEPOINT
nume_punct2  UPDATE .....
SAVEPOINT nume_punct3
DELETE.....
       ROLLBACK TO SAVEPOINT nume_punct2
        COMMIT
```

Succesiunea de comenzi anterioara comite tranzactiile specificate pana la SAVEPOINT nume_punct2, adica primul UPDATE si operatia INSERT. Intr-o tranzactie pot fi puse oricate puncte de revenire, numele acestora trebuiesc sa fie distincte.

II. NOȚIUNI DE PRELUCRARE A INFORMAȚIILOR DE TIP IMAGINE

Prelucrarea imaginilor este un domeniu multidisciplinar, aflat într-o continuă dezvoltare cu aplicații în diferite domenii de actualitate: apărare, industrie, medicină, agricultură, transporturi, mediu etc. Utilizând sisteme de calcul performante pentru procesarea automată a imaginilor, pentru manipularea și interpretarea informațiilor rezultate, se pot rezolva o mare varietate de probleme stringente și complexe din aceste domenii.

Spre deosebire de analizatorul vizual uman, sistemele de prelucrare a imaginilor bazate pe tehnică de calcul, nu sunt limitate doar la zona vizuală din spectrul electromagnetic (390μm ÷ 760μm), având un interval mult mai larg: de la raze gamma până la unde radio. De asemenea, aceste sisteme pot achiziţiona imagini de la surse mult diferite faţă de cele obişnuite pentru ochiul uman, cum ar fi: sistemele ultrasonice, termale, radiologice, microscoape electronice etc. Se poate considera că prelucrarea imaginii este parte componentă a conceptului mai larg de vedere artificială, considerat un domeniu multidisciplinar care utilizează cunoştinţe de optică, geometrie, electronică, programare, inteligenţă artificială şi construcţie a calculatoarelor.

Vederea artificială conține trei subprocese:

a) obţinerea reprezentării numerice a imaginii;
b) prelucrarea imaginii prin utilizarea tehnicii de calcul;
c) analiza şi interpretarea rezultatelor în diverse scopuri.

Funcţiile complexe ale vederii artificiale şi cantitatea mare de date vehiculate impun dotarea sistemelor de vedere artificială cu echipamente de calcul puternice, de regulă, specializate. Procesul de extragere, caracterizare şi interpretare a informaţiei din imagini poate fi divizat în şase etape: achiziţia imaginii (etapa senzorială), prelucrarea primară, segmentarea, descrierea, recunoaşterea şi interpretarea (etapele de prelucrare). De regulă, operaţiile complexe menţionate trebuie să fie executate în timp scurt (timp real).

Achiziţia imaginii are drept scop obţinerea reprezentării imaginii sub forma unui semnal electric analogic sau numeric.

Prelucrarea primară constă în tehnici de reducere a zgomotului din imagine şi de evidenţiere a detaliilor. Aceste procese de prelucrare a imaginii se caracterizează prin faptul că, atât intrarea cât şi ieşirea sunt imagini de aceeaşi dimensiune (număr de pixeli).

Segmentarea este procesul prin care se separă imaginea în obiecte (zone) de interes.

Descrierea se ocupă cu evidenţierea trăsăturilor sau primitivelor ce concentrează informaţia necesară separării claselor de obiecte.

Recunoaşterea este procesul prin care se identifică sau se clasifică aceste obiecte. Atât descrierea, cât şi recunoaşterea sunt proceduri de prelucrare complexă, rezultatul (ieşirea) nemaifiind o imagine.

În sfârşit, *interpretarea* atribuie o anumită semnificaţie unei mulţimi de obiecte recunoscute, având atribute cognitive sau de analiză.

Etapele de prelucrare a imaginilor se grupează, după gradul de complexitate, astfel:
a) *prelucrare de nivel scăzut* (prelucrarea primară);
b) *prelucrare de nivel mediu* (descrierea, codificarea și condensarea informaţiei prin extragerea trăsăturilor);
c) *prelucrare de nivel înalt* (recunoaşterea formelor – obiectelor, segmentarea și interpretarea).

1. REPREZENTAREA IMAGINILOR

În cele ce urmează vom considera imaginile în spațiul bidimensional. Dacă se consideră că imaginea este monocromă (echivalent, doar cu nuanţe de gri) și staţionară se defineşte *funcţia nivel de gri (sau funcţia de intensitate) G* ca fiind legea ce asociază unui punct (x,y) din domeniul de definiţie D (de regulă interiorul unui dreptunghi D din planul R^2) un număr corespunzător strălucirii (luminanţei) din acel punct (1.1):

$$G : D \to R \qquad (1.1)$$

Valoarea $G(x,y)$ se numeşte *nivelul de gri în punctul* (x,y).

În mod analog, pentru imaginile color se consideră un set de trei funcţii G_R, G_G, G_B, corespunzătoare celor trei culori fundamentale (roşu - R, verde - G şi albastru - B) ce alcătuiesc nuanţa de culoare şi intensitatea luminoasă dintr-un punct al imaginii. Asemănător, se poate considera setul HSV (Hue, Saturation, Value).

Datorită dezvoltării puternice a tehnicilor numerice de calcul, un interes deosebit îl prezintă imaginile numerice. Astfel, semnalul provenit de la senzorul vizual poate fi eşantionat spaţial (în plan) şi cuantizat în nivel. În acest mod, se obţine *imaginea numerică* (monocromă sau alb – negru, cu nuanţe de gri), caracterizată prin funcţia G, (1.2):

$$G : M_1 \times M_2 \to N \qquad (1.2)$$

unde $M_1 \subset N$, $M_2 \subset N$, N fiind mulţimea numerelor naturale. Deoarece în lucrare se vor folosi doar imagini numerice nu există pericol de confuzie între (1.1) și (1.2).

$$M_1 = \{0,1,\ldots,n_1 - 1\}, M_2 = \{0,1,\ldots,n_2 - 1\}$$

Perechea $(n_1, n_2) \overset{not}{=} n_1 \times n_2$ se numeşte ***dimensiunea imaginii***.

Un element de imagine denumit în continuare ***pixel*** este un triplet care contine poziţia pixelului (i,j) si valoarea (nivelul de gri) pixelului $G(i,j)$:

$$E_{i,j,G} = \{i, j, G(i, j)\}, i \in M_1, \; j \in M_2$$

Imaginea numerică se poate reprezenta printr-o matrice [G] de dimensiune $n_1 \times n_2$ (1.3). Imaginile color se pot reprezenta, în domeniul original, spaţial, printr-un set de trei matrice, $[G_R]$, $[G_G]$, $[G_B]$, de aceeaşi dimensiune, corespunzând ponderilor culorilor fundamentale în alcătuirea pixelilor color.

$$[G] = \begin{bmatrix} G(0,0) & G(0,1) & G(0, n_2 - 1) \\ G(1,0) & G(1,1) & G(1, n_2 - 1) \\ \ldots & \ldots & \ldots \\ G(n_1 - 1,0) & \ldots & G(n_1 - 1, n_2 - 1) \end{bmatrix} \qquad (1.3)$$

In reprezentarea prin numere naturale, de regulă, fiecare culoare fundamentală ia valori în gama 0 (minimum) ÷ 255 (maximum). Această soluţie – *Truecolor* – (reprezentarea pe 24 de biţi, câte 8 biţi pentru o culoare fundamentală) s-a ales deoarece este suficientă în majoritatea aplicaţiilor.

2. PRELUCRAREA PRIMARĂ A IMAGINILOR

2.1. Operatori punctuali

După numărul de operanzi implicaţi, operaţiile pot fi *cu un operand* (exemple: translaţia imaginii, modificarea contrastului) sau *cu mai mulţi operanzi*, evident, de aceeaşi dimensiune (exemple: adunarea şi scăderea a două imagini).

Pentru implementarea operațiilor punctuale cu un singur operand se folosesc trei *operatori de bază* şi anume: operatorul identitate, operatorul de inversare şi operatorul cu prag pentru binarizare.

Operatorul identitate se caracterizează prin relaţia (2.5):

$$G'(i,j) = G(i,j), \qquad i \in M_1,\ j \in M_2 \tag{2.5}$$

şi este reprezentat grafic, pentru $N_G = \{0,1,...,9\}$, în Fig. 2.5.

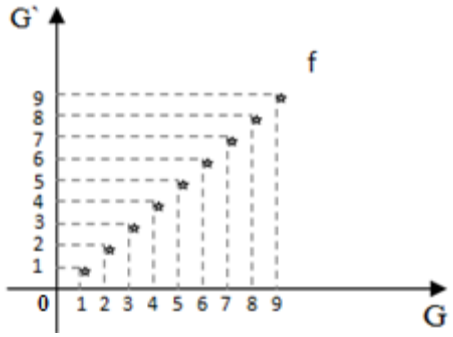


Fig. 2.5. Operatorul identitate.

Operatorul de complementare se caracterizează prin relaţia (2.6):

$$G'(i,j) = G_{\max} - G(i,j),\ i \in M_1,\ j \in M_2 \tag{2.6}$$

$G_{\max}$ fiind valoarea maximă a funcţiei nivel de gri şi este reprezentat grafic în Fig. 2.6.

Se observă că acest operator realizează o complementare a valorilor lui G faţă de $G_{\max}$.

Exemplu ($G_{\max}$=9)

$$\begin{matrix} 1 & 2 & 0 & 1 & 0 \\ 1 & 6 & 6 & 6 & 1 \\ 0 & 6 & 9 & 6 & 2 \\ 1 & 6 & 6 & 6 & 2 \\ 0 & 2 & 1 & 1 & 1 \end{matrix} \xrightarrow{f} \begin{matrix} 8 & 7 & 9 & 8 & 9 \\ 8 & 3 & 3 & 3 & 8 \\ 9 & 3 & 0 & 3 & 7 \\ 8 & 3 & 3 & 3 & 7 \\ 9 & 7 & 8 & 8 & 8 \end{matrix}$$

[G] [G`]

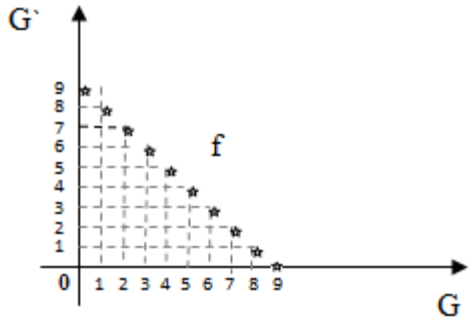


Fig. 2.6. Operatorul de complementare.

Operatorul cu prag pentru binarizare se caracterizează prin relaţia (2.7):

$$G`(i,j)=\begin{cases}0, & G(i,j)<P\\ 1, & G(i,j)\geq P\end{cases}, \quad i\in M_1, \quad j\in M_2, \tag{2.7}$$

P fiind un prag prestabilit şi este reprezentat grafic în Fig. 2.7.

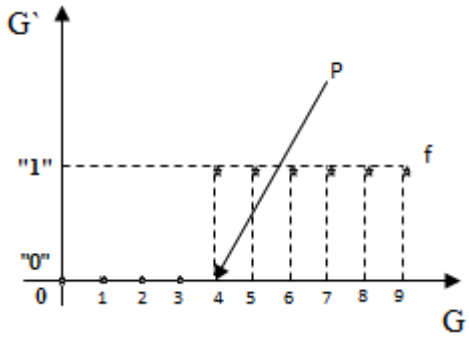


Fig. 2.7. Operator cu prag pentru binarizare.

Exemplu ($G_{max} = 9$, $P = 4$)

$$\begin{array}{ccccc} 1 & 2 & 0 & 1 & 0 \\ 1 & 6 & 6 & 6 & 1 \\ 0 & 6 & 9 & 6 & 2 \\ 1 & 6 & 6 & 6 & 2 \\ 0 & 2 & 1 & 1 & 1 \end{array} \xrightarrow{f} \begin{array}{ccccc} 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{array}$$

$$[G] \qquad\qquad [G`]$$

Combinând operatorii de bază se pot obține o serie de ***operatori derivaţi***.

2.2 Prelucrarea primară a imaginilor prin filtre locale (Operatori locali)

Prelucrarea prin metode locale presupune înlocuirea valorii unui pixel (pixel inițial) cu valoare unei funcții depinzând de valorile pixelilor dintr-o vecinătate a pixelului inițial. Dispozitivele (programele) care realizează aceste operații le vom denumi filtre. Cele mai importante operații de filtrare locală sunt: binarizarea, eliminarea (reducerea) zgomotului și extragerea liniilor de contrast (în practică contururilor).

Binarizarea imaginilor

Binarizarea imaginilor joacă un rol extrem de important în cadrul aplicaţiilor industriale ce includ module de vedere artificială sau de recunoaşterea formelor, întrucât rolul acesteia este de a discrimina între zone de imagine cu diferite semnificaţii (discriminarea obiect – fundal). În urma operației de binarizare, de cele mai multe orise obține o segmentare a imaginii în regiuni de interes.

Algoritmii de binarizare se clasifică după modul de alegere a valorii pragului de binarizare. Astfel, principalele tipuri de algoritmi de binarizare sunt:

- binarizare cu prag fixat;
- binarizare cu prag adaptiv bazate pe:
  - tehnici cu prag global;
  - tehnici de binarizare locală.

Operaţia de binarizare în sine este extrem de simplă, fiecare pixel din imagine primeşte una dintre cele două valori posibile (alb sau negru – 1 sau 0) pe baza comparaţiei valorii intensităţii cu o valoare prag (relaţia 3.1).

$$B(i,j)=\begin{cases}1, \text{ dacă } G(i,j)\geq P(i,j)\\ 0, \text{ dacă } G(i,j)<P(i,j)\end{cases} \qquad (3.1)$$

Binarizare cu prag fixat

Binarizarea cu prag fixat este trivială ca aplicaţie, deoarece decizia alegerii pragului nu este automată. Totuşi descrierea modului de alegere a pragului de binarizare este utilă pentru înţelegerea modului în care binarizarea poate discrimina pe criterii de semnificaţie. Operatorul de binarizare cu prag fixat a fost prezentat în relația (2.7).

Această situaţie apare des în aplicaţii industriale şi permite alegerea ușoară a pragului de binarizare: valoarea corespunzătoare minimului dintre cele două puncte de maxim ale histogramei. Modificările pragului de binarizare poate conduce la rezultate (imagini diferite)

Binarizare cu prag adaptiv

În condiţii de mediu dificile, această metodă dă rezultate mai bune, deoarece ţine seama de intensitatea luminoasă într-o vecinătate a punctului considerat. Pentru stabilirea pragului de binarizare $P(i,j)$, se consideră o vecinătate simetrică $V_{i,j}$, de dimensiune $n\times n$, $n = 3 \div 15$. Tehnicile de binarizare cu prag adaptiv determinat local pentru fiecare punct al imaginii se comportă mai

bine pe imagini în care intensitatea luminoasă diferă de la o zonă la alta a imaginii. Practic pentru fiecare pixel se determină valoarea pragului pe baza valorilor dintr-o vecinătate simetrică.

Pentru stabilirea valorii pragului se poate folosi una din urmatoarele mărimi:

- Media aritmetică dintre valoarea maximă și valoarea minimă a nivelului de gri din vecinătate;
- Media aritmetică a nivelului de gri din vecinătate ;
- Mediana nivelului de gri din vecinătate;
- Punctul de minim dintre cele două vârfuri ale histogramei nivelului de gri, dacă aceasta este bimodală.

Reducerea zgomotului din imagini

Zgomotul de imagine apare ca urmare a unor procese nedorite care se manifestă în timpul achiziţionării semnalului util. În cazul imaginilor este vorba de variaţii ale sensibilităţii senzorilor, variaţii ale mediului, natura discretă a radiaţiei luminoase, erori de transmisie sau de cuantizare etc.

În general zgomotul poate fi grupat în două clase:

- *zgomot independent de imagine*: apare datorită unor factori ce ţin în principal de caracteristicile dispozitivului de achiziţie;
- *zgomot dependent de imagine*: apare în special datorită modului în care anumite suprafeţe reflectă lumina.

Eliminarea sau reducerea zgomotului din imagine este prima operație de prelucrare primară ce trebuie efectuată asupra unei imagini deoarece zgomotul poate fi amplificat de prelucrarile ulterioare cum ar fi îmbunătățirea contrastului. În unele aplicaţii ale vederii artificiale, ca de exemplu interpretarea imaginilor luate de la distanță (satelit, avion sau dronă) sau a imaginilor medicale, problema reducerii zgomotului este destul de dificilă. În alte aplicaţii, cum sunt cele industriale sau de laborator, unde condiţiile de mediu sunt create artificial, problema reducerii zgomotului este mai simplă.

Principalele tipuri de zgomote ce apar în imagini sunt zgomotul gaussian și zgomotul de tip „sare şi piper”. Zgomotul gaussian poate fi redus prin utilizare unui filtru de tip trece jos. Din această cauză, prin netezire nu se reduce doar zgomotul ci şi detalii ale imaginii. De regulă pentru reducerea zgomotului de imagine se folosesc filtre locale (acționând în vecinătăți ale pixelului considerat) deoarece au o comportare mai bună față de liniile de contur decât cele globale.

Filtrul bazat pe valoarea medie

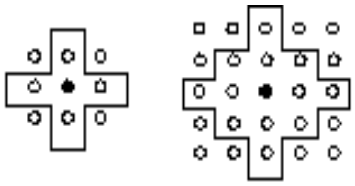
Filtrul bazat pe valoarea medie înlocuieşte valoarea nivelului de gri în punctul *(i,j)* cu valoarea medie (neponderată) pe o vecinătate $(2h+1)\times(2h+1)$ a punctului respectiv:

$$G'(i,j)=\frac{1}{(2h+1)^2}\sum_{m=-h}^{h}\sum_{n=-h}^{h}G(i+m,j+n),$$

$$i=\{h,h+1,...,n_1-1-h\}\ ;\ j=\{h,h+1,...,n_2-1-h\}$$

Filtrul bazat pe valoarea mediană (filtru median)

Filtrul median înlocuieşte valoarea centrală $G(i,j)$ cu mediana valorilor dintr-o vecinătate simetrică, de dimensiune $(2h+1)\times(2h+1)$, centrată în punctul (i,j). Se obţin rezultate asemănătoare cazului monodimensional şi anume că filtrul median bidimensional păstrează liniile de pe contur şi zonele constante mai mari decât fereastra şi elimină zgomotele de tip impuls. Ca vecinătăţi pentru filtrul median se folosesc vecinătăţi simetrice de n puncte (cu n impar), ca în exemplele urmatoare.



Filtrul de netezire logică (Filtrul binar)

Un caz important, care apare la sistemele de vedere artificială ale roboţilor industriali, este acela în care $G(i,j)$ este codificată pe un singur bit (imagine binarizată). Atunci, pentru eliminarea zgomotului de tip *sare şi piper* (1 între *0* -uri sau *0* între *1*-uri) se pot folosi filtre locale, bazate pe funcţii logice .

Pentru a putea folosi filtrul binar (FB) este necesară binarizarea imaginii. Pragul de binarizare se alege în aşa fel încât, prin binarizare să se detaşeze silueta obiectului de fond şi imaginea să aibă un zgomot – pixeli alteraţi – cât mai redus, iar acesta să fie de tip *sare şi piper*. Astfel, se obţine o imagine având *1* logic pentru pixelii aparţinând formei şi *0* logic pentru pixelii aparţinând fondului.

Eliminarea zgomotului din imagini binarizate prin operații morfologice

Eroziunea și dilatarea sunt două operații morfologice utilizate cu precădere în analiza imaginilor binare [Pr06]. Operația de *dilatare* mărește obiectele din imagini, permițând umplerea unor mici goluri și conectarea obiectelor disjuncte, în timp ce *eroziunea* micșorează obiectele prin erodarea marginilor obiectelor.

Pentru un element structural 3×3, operațiile morfologice de mai sus pot fi considerate într-o formă simplificată astfel: Fie $B(i,j)$, $i \in \{0,1,\dots,n_1-1\}$, $j \in \{0,1,\dots,n_2-1\}$ o imagine binarizată și $C(i,j)$ conturul său. Imaginea diferență dintre imaginea binarizată și imaginea contur este *imaginea erodată* a lui $B(i,j)$ și se notează cu E

$$E(i,j) = B(i,j) - C(i,j)$$,

Dacă, baleind imaginea binarizată cu o fereastră 3x3 şi întâlnind 1 în punctul central, se adaugă 1 în punctele vecine, se obţine *imaginea dilatată* $D(i,j)$.

Combinând eroziunea cu dilatarea, se obţine o metodă de eliminare a unor zone perturbate, de dimensiuni maxime de doua linii sau doua coloane alaturate.

Evident, aplicarea metodei conduce la pierderea detaliilor având aceste dimensiuni sau la alterarea celor de dimensiuni apropiate de 2×2. Metoda permite eliminarea unei linii sau a unei coloane atunci când senzorul este defect, sau a unor pete de dimensiuni similare datorate strălucirii pieselor.

Determinarea liniilor de contrast și a contururilor

Informaţia furnizată de imagine este conţinută, mai ales, în liniile de contrast puse în evidenţă de variaţiile de strălucire (contrast) pe imagine. Algoritmii de extragere a conturului sunt, în principiu, algoritmi de creștere a contrastului combinați cu algorimi de binarizare [Pr06].

Din punct de veder teoretic, pentru creșterea contrastului se folosesc algoritmi diferențiali (de gradient) aplicați funcției nivel de gri $G: D \to R$ (presupusă derivabilă). Astfel $G(x,y)$ se înlocuiește cu valoarea absolută a gradientului său, $Grad\ G(x,y)$:

$$A(x,y) = \left|GradG(x,y)\right| = \sqrt{\left(\frac{\partial G}{\partial x}\right)^2 + \left(\frac{\partial G}{\partial y}\right)^2}$$

Deoarece, de regulă, nu interesează nuanţele de gri de pe contur, urmează binarizarea lui $A(x,y)$:

$$C(x,y) = \begin{cases} 1, & dacă \quad A(x,y) \geq P \\ 0, & dacă \quad A(x,y) < P, \end{cases}$$

unde prin $C(x,y)$ s-a notat noua valoare a funcţiei nivel de gri în punctul (x,y), iar prin P, pragul de comparaţie (fix sau variabil) utilizat la binarizare. C: $D \to \{0,1\}$ este imaginea conturului.

În cazul imaginilor numerice, s-au propus diverşi algoritmi care aproximează gradientul într-un punct utilizând măști (matrice) de diverse dimensiuni. Operatorii cei mai cunoscuti sunt: Roberts, Sobel, Mero-Vassy, Prewitt, Otsu.

3. EXTRAGEREA TRĂSĂTURILOR DIN IMAGINI

Trăsăturile unei forme (imagine) sunt funcții numerice ce exprimă caracteristici sau proprietăți ale acelei forme și care au rolul de a condensa informaţia în aşa fel încât forma să poată fi reprezentată univoc într-un spaţiu de dimensiune mică, numit spaţiul trăsăturilor. În acest spațiu forma sau imaginea poate fi clasificată sau interpretată cu un volum de calcul mult mai mic decât dacă s-ar utiliza imaginea inițială. Trăsăturile trebuie să prezinte proprietatea de definire și separare a claselor, în sensul că, pe de o parte trebuie să ia valori cât mai diferite de la o clasă la alta, iar pe de altă parte să aibă valori cât mai apropiate în cadrul aceleiași clase (proprietatea de grupare).

În Tabelul 4.1 sunt prezentate trăsăturile mai des folosite în procesul de recunoaştere a formelor (în general, pentru forme singulare într-un cadru de imagine). Ele sunt extrase din imaginea binarizată (mai precis, imaginea segmentată, considerând valoarea 1 pentru formă și 0 pentru fundal).

Tabelul 4.1. Exemple de trăsături utilizate în procesul de recunoaştere a formelor plane

Tipul/ Trăsătura	**Geome-trice**	**Funcțio-nale**	**Topolo-gice**	**Invariante la rotație și translație**	**Dependente de rotație**	**Dependente de translație**
Aria	**X**	-	-	X	-	-
Perimetrul	X	-	-	X	-	-
Centrul de greutate	X	-	-	-	-	X
Raza minimă	X	-	-	X	-	-
Raza maximă	X	-	-	X	-	-
Raza medie	X	-	-	X	-	-
Numărul lui Euler	-	-	X	X	-	-
Numărul de orificii	-	-	X	X	-	-
Momente	-	X	-	-	X	X

După cum este arătat în Tabelul 4.1, marcat cu X, trăsăturile pentru forme (obiecte) plane singulare se pot clasifica după mai multe criterii. Astfel, după natura lor, trăsăturile pot fi: geometrice, funcţionale, topologice etc. De asemenea, aceste caracteristici pot fi grupate după invarianţa la anumite transformări geometrice: trăsături independente de rotaţie şi translaţie, dependente de rotaţie, dependente de translaţie etc. În afară de aceste trăsături mai există și trăsături pentru forme speciale, cum ar fi fractalii sau texturile. Ele vor fi abordate într-un subcapitol separat.

În vederea creşterii vitezei de recunoaştere a formelor se recomandă determinarea trăsăturilor prin prelucrarea paralelă, fie cu procesoare dedicate fiecărei trăsături, fie cu procesoare dedicate pixelilor (procesoare matriceale).

În general, nu există o metodă teoretică, general valabilă de alegere și utilizare a trăsăturilor în procesul de recunoaștere și clasificare a formelor, această operație fiind dependentă de aplicație și necesitând o serie de etape de testare și validare a rezultatelor.

4. RECUNOAȘTEREA FORMELOR DE TIP IMAGINE

În multe aplicații ale vederii artificiale un interes deosebit îl au detecția, marcarea și evaluarea unor zone cu anumite proprietăți pe care le putem numi forme sau regiuni de interes. Detecția acestora se face prin metode specifice de recunoaștere a formelor, adică o clasificare sau atribuire a lor unor grupări numite *clase*. Operația de marcare sau de segmentare se face prin etichetarea cu

valori specifice a pixelilor regiunii clasificate. De regulă se cere și evaluarea mărimii acestei zone segmentate (ariei).

După cum s-a precizat anterior, recunoaşterea formelor este o etapă de nivel înalt în prelucrarea imaginilor. Rezultatul acestei etape nu mai reprezintă o imagine, ci este decizia de apartenență la o clasă specificată iniţial.

În general, prin recunoaşterea formelor se înţelege clasificarea sau discriminarea unor obiecte, semnale sau procese, numite forme, dintr-o mulţime dată. Această mulţime este partiţionată în submulţimi, numite *clase*, separate prin anumite proprietăţi. Pe de altă parte, aceste proprietăţi trebuie să grupeze formele în interiorul claselor, ele fiind comune membrilor unei clase. Scopul recunoaşterii formelor este acela de a atribui formei analizate, numele (eticheta) clasei din care face parte.

Pentru recunoaşterea automată a formelor, se pot utiliza, fie *metode matematice*, fie *metode sintactice*. Metodele matematice se grupează, după proprietăţile claselor, în *metode deterministe* şi *metode statistice*. În cazul recunoaşterii formelor bidimensionale (de tip imagine), cele mai larg utilizate metode matematice sunt cele cu *decizie teoretică*. Aceste metode se bazează pe utilizarea unor *funcţii de decizie* (funcţii discriminant) a căror valoare trebuie extremizată pentru realizarea clasificării. Rezultatul recunoaşterii formelor prin utilizarea funcţiilor de decizie constă doar în atribuirea etichetei clasei formei recunoscute. Metodele sintactice sau structurale utilizează descrieri simbolice ale formelor, care trebuie să fie propoziţii valide într-un anumit limbaj. Rezultatul recunoaşterii prin utilizarea metodelor sintactice conţine eticheta clasei (clasificarea), cât şi descrierea structurală a formei. De cele mai multe ori, în procesul de recunoaştere, se utilizează o singură metodă, dar sunt cazuri în care se întâlnesc şi combinaţii ale acestor metode.

Schema funcţională a unui sistem general de recunoaştere a formelor este prezentată în Fig. 4.1. Semnificaţia notaţiilor din Fig. 4.1 este următoarea: FI – forma de intrare, BA – bloc de achiziţie a datelor, BR – bloc de reprezentare a formei, BD – bloc de decizie, BAS – bloc de analiză structurală, C – clasificare și C+D – clasificare şi descriere. În paranteză s-a marcat cazul recunoașterii sintactice.

Fig. 4.1. Sistem general de recunoaştere a formelor.

Dintre metodele şi tehnicile de recunoaştere a formelor prezentate, cele mai utilizate în aplicații sunt cele bazate pe funcţii de decizie de tip *distanţă minimă*. Metodele bazate pe funcţii de decizie de tip distanţă minimă în spaţiul trăsăturilor oferă anumite avantaje faţă de celelalte metode: condensarea informaţiei, volum mai mic de calcul, extragerea relativ uşoară a trăsăturilor invariante la rotaţie şi translaţie, sensibilitate mai redusă la zgomot. De asemenea, imaginile conţinând texturi sau fractali se clasifică, de regulă, tot pe baza distanţei minime în spaţiul trăsăturilor.

4.1.Metode de recunoaştere prin comparație directă

Una din primele şi cele mai simple metode de recunoaştere a formelor este potrivirea cu modelul. În această abordare fiecărei clase îi corespunde un model (şablon). Forma de clasificat (forma de intrare în dispozitivul de recunoaştere) este comparată cu acest model. Pe baza criteriului de clasificare stabilit în prealabil (un criteriu de similitudine a formei de intrare cu modelul), forma de intrare se atribuie unei anumite clase. Pentru detecția obiectului din imagine, aceasta este baleiată cu o fereastră (mască) ce reprezintă modelul până când acesta şi obiectul real din imagine se suprapun (ceea ce corespunde unei valori extreme a *criteriului de similitudine*).

În cazul imaginilor binare, cel mai simplu criteriu de similitudine îl constituie numărul maxim de coincidenţe între pixelii şablonului şi pixelii porţiunii de imagine comparate cu şablonul. În cazul imaginilor cuantizate pe mai multe niveluri de gri, se adoptă drept criteriu de similitudine fie distanţa minimă dintre şablon şi fereastra de imagine, fie funcţia de intercorelaţie normalizată maximă dintre şablon şi fereastra de imagine.

4.2. Metode de recunoaştere prin compararea trăsăturilor

Imaginea de intrare [G], conţinând forma de clasificat, trebuie comparată, în spaţiul euclidian al formelor, de dimensiune $n_1 \times n_2$, cu formele de referinţă ce reprezintă clasele C^i din mulţimea: $C = \{C^i, i = 1, 2, \ldots, p\}$.

De obicei, dimensiunea $n_1 \times n_2$ a matricei [G] este foarte mare şi atunci comparaţia nu se mai face în spaţiul formelor, ci într-un spaţiu $\boldsymbol{X}$ de dimensiune s, mult mai mică, numit *spaţiul trăsăturilor* ($s << n_1 n_2$).

După cum s-a menţionat amterior, în linii mari, prin trăsături se înţeleg caracteristici numerice comune tuturor formelor analizate, ce concentrează o cantitate mare de informaţie şi care, pe cât posibil, iau valori cât mai diferite de la o clasă la alta și cât mai apropiate în interiorul aceleiași clase. Aceste trăsături pot fi considerate descriptori de forme bidimensionale.

Extragerea trăsăturilor în scopul recunoaşterii formelor este particulară şi ţine seama de aplicaţia respectivă, neexistând până în prezent o teorie riguroasă privind alegerea lor. Totuşi, criteriile de selecţie trebuie să ţină seama de următoarele elemente: eficienţa, importanţa trăsăturilor în separarea claselor (în spaţiul trăsăturilor, distanţa dintre clase să fie cât mai mare, iar în interiorul claselor distanțele să fie cât mai mici), timpul, costul, redundanţa etc. Uneori, se cere invarianţa trăsăturilor la rotaţia şi translaţia imaginii.

4.3. Metode de recunoaştere sintactice

În abordarea sintactică a recunoaşterii formelor, este esenţială descompunerea formelor în subforme sau *primitive*. Fiecare primitivă este interpretată ca simbol permis într-o anumită gramatică, unde prin gramatică sunt înţelese, în linii mari, un set de reguli de sinteză pentru generarea propoziţiilor din simboluri date. Fiecărei clase îi corespunde un limbaj generat cu o anumită gramatică:

$$C^j \leftrightarrow L(G_j), \quad j \epsilon \{1, 2, \ldots, p\}$$

Fiind dată o propoziţie (un şir de simboluri), ce reprezintă o formă imagine, problema care se pune este de a decide în care limbaj descrierea acestei forme este o propoziţie validă. Forma este atribuită în mod unic clasei C^j dacă ea este o propoziţie din $L(G_j)$ şi nu din alt limbaj.

5. APLICATII ALE PRELUCRARII IMAGINILOR IN DIVERSE DOMENII (LA ALEGEREA STUDENTULUI): SPECIFICAREA DOMENIULUI, METODA DE ABORDARE, EVENTUALE REZULTATE PRECONIZATE SAU OBTINUTE.