



Universitatea
Politehnica
din Bucureşti



Facultatea de
Automatică și
Calculatoare



Departamentul
Calculatoare

BAZE DE DATE -1

Conf. dr. ing. Alexandru Boicea
alexandru.boicea@upb.ro

PONDERI EVALUARE FINALA

➤ EVALUARE EXAMEN

- Evaluare in cursul semestrului – **60%**
 - Prezenta si activitate la curs – 10%
 - Evaluare curs(testare la mijlocul semestrului, fara degrevare) – 20%
 - Evaluare laborator – 30%
- Evaluare finala (examen) – **40%**

➤ EVALUARE LABORATOR

- Prezenta laborator – 10%
- Teste laborator – 40%
- Colocviu laborator – 50%

➤ CONDITII PROMOVARE EXAMEN

- Minim 50% din punctajul de laborator
- Minim 50% din punctajul de examen



Capitolul 1

Concepte de baze de date



Definitii

- **Baza de date** reprezinta o modalitate de stocare a datelor pe un suport extern de memorie, intr-o anumita structura, in vederea interogarilor concurente si prelucrarilor ulterioare pentru extragerea informatiilor.
- **Sistem de gestiune a bazei de date(SGBD)** este un ansamblu de programe software care permit utilizatorilor definirea obiectelor, administrarea bazei de date si accesul concurrent la baza de date. Cele mai cunoscute si utilize sisteme de gestiune sunt Oracle, Microsoft SQL Server, IBM DB2, IBM Informix, MySQL, PostgreSQL, Sybase, etc.



Definitii

- **Dictionarul bazei de date** este acea componenta a unui SGBD care contine informatii despre obiectele din baza de date si parametrii de sistem, cum ar fi:
 - Structurile tabelelor, indecsii si constrangerile de integritate definite pe ele;
 - Spatiul fizic si organizarea logica a fisierelor;
 - Userii creati pe baza de date si drepturi de acces.
- **Organizarea datelor** reprezinta procesul de definire, structurare si relationare a datelor in colectii de date.
- **Colectie de date** este un ansamblu de date organizate pe anumite criterii de functionalitate, denumita tabela (pentru bazele de date relaționale) și obiect (pentru bazele de date orientate obiect).



Definitii

- **Structura bazei de date** reprezinta o colectie de descrieri statice ale tipurilor de entitati impreuna cu relatiile logice stabilite intre ele.
- **Entitate** reprezinta un obiect al bazei de date care are o reprezentare unica(de ex. tabelele *Studenti*, *Grupe*, *Catalog*, etc.).
- **Atribut** este o proprietate ce descrie o anumita caracteristica a unei entitati(de ex. pentru tabela *Studenti* atrbute pot fi *nr_matricol*, *nume*, *grupa*, *specializare*, etc.).



Definitii

- **Relatiile logice** reprezinta asocierile dintre mai multe entitati care respecta anumite restrictii de functionalitate (un ex. de asociere intre tabelele *Studenti si Grupe* ar fi *nr_matricol, nume, cod_grupa*)
- **Fișier de date** reprezinta o colectie de date aflate in asociere, definit printr-o structura logica si una fizica.



Modele de baze de date

Regulile și conceptele care permit descrierea structurii unei baze de date formează **modelul datelor**:

- **Modelul ierarhic** - datele sunt organizate sub forma unui arbore, nodurile constând din înregistrări(date) iar arcele fiind referințe(pointeri) către alte noduri(datele sunt organizate într-o structură arborescentă de tip tata/fiu).
- Datele sunt grupate în înregistrări ce descriu o ierarhie de tip 1-1 sau 1-N (un parinte poate avea unul sau mai mulți fii, în timp ce un fiu poate avea doar un parinte).



Modele de baze de date

- Datele sunt organizate în entități și fiecare entitate poate avea mai multe atribută.
- Modelul ierarhic este rar folosit în bazele de date moderne, cel mai cunoscut fiind sistemul de fisiere din Windows.



Modele de baze de date

- **Modelul retea** - datele sunt organizate sub forma unui graf orientat.
 - Nodurile si arcele au aceeasi semnificatie ca la modelul ierarhic.
 - Modelul retea completeaza modelul ierarhic iar datele sunt grupate in inregistrari ce descriu o ierarhie de tip 1-1, 1-N, K-N (un parinte poate avea unul sau mai multi fii, iar un fiu poate avea mai multi parinti).



Modele de baze de date

- Modelul retea a fost conceput de Charles Bachman si datele sunt structurate sub forma unui graf orientat, fiecare nod putand avea mai multe inregistrari parinte si mai multi fii.
- Este conceput ca o metoda flexibila de reprezentare a obiectelor si relatiilor dintre ele.
- In modelul retea fiecare inregistrare poate avea parinti si fii mulți, formând o latice care permite o mai bună modelare a relatiilor dintre entități.



Modele de baze de date

- **Modelul Entitate-Asociere** - este folosit in proiectarea conceptuala de nivel inalt a bazelor de date.
 - Modelul consta intr-o abordare grafica a proiectarii bazelor de date.
 - Datorita simplitatii si expresivitatii sale a fost adoptat de comunitatea stiintifica precum si de producatorii de software si va fi tratat in detaliu intr-un capitol separat.



Modele de baze de date

- **Modelul Relational** - datele sunt organizate sub forma de tabele ce pot fi relateionate intr-o ierarhie de tip 1-1, 1-N, K-N.
 - Modelul este sustinut si de un limbaj standard de manipulare a datelor.
 - Echivalentul unei entitati intr-o baza de date relationala este tabela, iar a unui atribut este coloana.
 - Este cel mai utilizat model de baze de date si de aceea va fi tratat mai in detaliu intr-un capitol ulterior.



Modele de baze de date

- **Modelul obiect** - datele sunt reprezentate sub forma de obiecte si sunt folosite in programarea orientata pe obiecte.
- Un obiect este o unitate de program care este folosita in constructia blocurilor de program.
- Fiecare obiect este capabil sa receptioneze parametri, sa prelucreze date si sa transmita rezultate altor obiecte.



Modele de baze de date

- **Modelul obiect-relational** - se bazeaza pe tehnici de programare *Object-Relational Mapping* (O/RM, ORM, si O/R mapping) pentru conversia datelor din diferite sisteme de baze de date si limbaje de programare orientate pe obiecte.
- Ele au ca efect crearea unei “baze de date virtuale” care poate fi accesata in limbaje de programare specifice.
- Pentru aceasta sunt disponibile pe piata diferite pachete de programe dar se pot crea si propriile tools-uri ORM.



Modele de baze de date

- Cel mai raspandit model de baze de date este cel relational, in care datele sunt stocate in tabele. Popularitatea acestui model se datoreaza simplitatii sale (din punct de vedere al utilizatorului) si a posibilitatii de definire a unor limbaje neprocedurale de descriere si manipulare a datelor.
- Termenul de relatie (care da denumirea modelului) provine din matematica, iar reprezentarea intuitiva a unei relatii este o tabela de asociere intre doua coloane a doua tabele diferite.
- In cazul modelului relational descrierea structurii unei baze de date consta in principal din descrierea tabelelor componente: denumire, lista de coloane si tipul datelor stocate in acestea.



Sistemul de gestiune a bazei de date (SGBD)

- Sistemele de gestiune a bazelor de date structurate au avut, aproximativ, urmatoarea evolutie de-a lungul timpului:
 - Sisteme bazate pe fisiere de date (1950)
 - SGBD bazate pe modelul de date ierarhic(1960)
 - SGBD bazate pe modelul de date retea (1970)
 - SGBD entitate-asociere (1975)
 - SGBD relationale (1980, varianta comerciala)
 - SGBD obiect-relationale(1990)
 - SGBD orientate spre aplicatii (web, baze de date spatiale, temporale, multimedia, etc.) (2000)
 - SGBD de depozitare a datelor (data warehousing, data mining, etc.) (2010)



Sistemul de gestiune a bazei de date (SGBD)

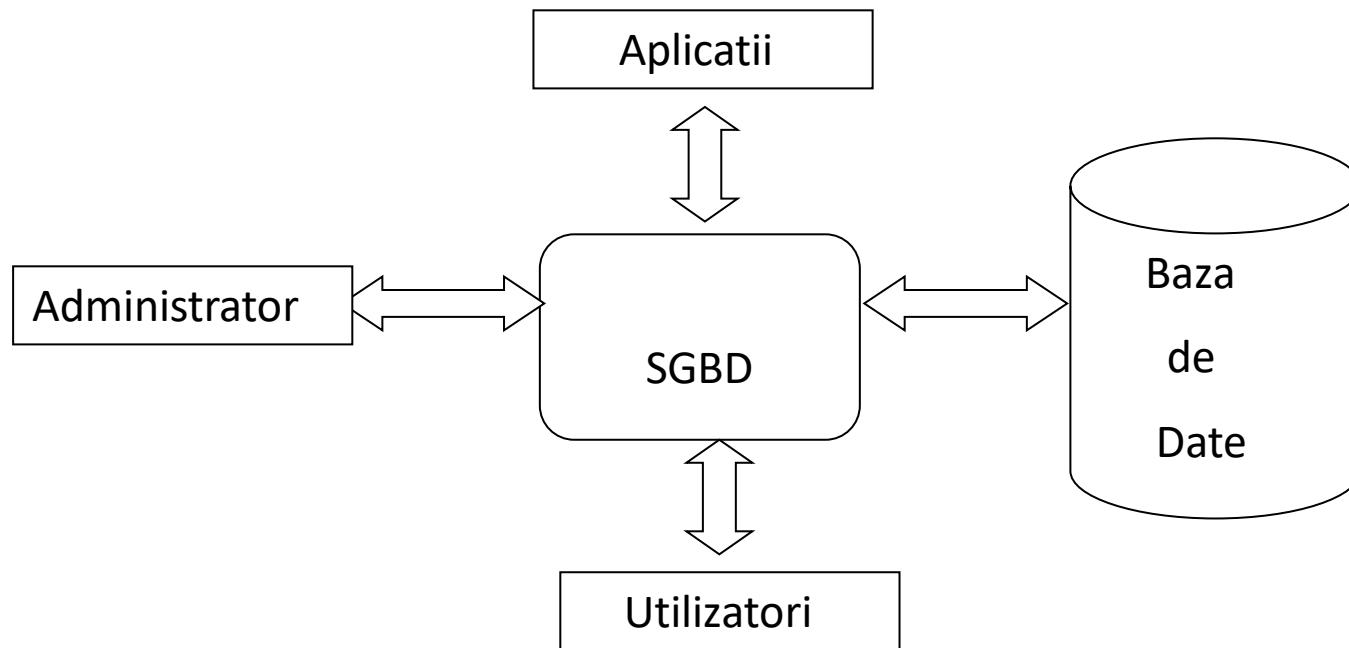


Figura 1. Schema bloc de functionare a unui SGBD



Sistemul de gestiune a bazei de date (SGBD)

- Functiile principale ale unui SGBD sunt:
 - Definirea structurilor tabelare;
 - Definirea relatiilor dintre tabele;
 - Definirea constrangerilor de integritate a datelor;
 - Manipularea datelor cu ajutorul unui limbaj SQL;
 - Controlul drepturilor de acces la baza de date;
 - Controlul accesului concurrent al userilor la date;
 - Administrarea automata a catalogului bazei de date;
 - Administrarea spatiului fizic si logic de stocare;
 - Controlul backup-urilor (copii de siguranta).



Definirea structurilor tabelare

- Un SGBD trebuie sa includa posibilitatea definirii structurii obiectelor care formeaza baza de date.
- In cazul bazelor de date relationale aceasta consta in principal in posibilitatea crearii si modificarii structurii tabelelor si constrangerilor de integritate asociate acestora.
- Limbajul prin care se realizeaza aceste operatii se numeste DDL (*Data Definition Language*)
- In sistemele relationale bazate pe SQL aceste operatii au fost incluse in limbaj prin intermediul comenzilor de tip CREATE (pentru creare), ALTER (modificare) sau DROP(stergere din dictionar).



Definirea structurilor tabelare

- Structura unei tabele este data de urmatoarele specificatii de definire:
 - Definirea coloanelor si tipurile acestora;
 - Definirea constrangerilor de integritate;
 - Definirea tablespace-lui unde se creeaza tabela;
 - Definirea parametrilor logici si fizici.
- Definirea unei tabele trebuie sa respecte anumite reguli:
 - Sa nu existe duplicare de randuri si coloane ;
 - Nu trebuie respectata o anumita ordine a coloanelor;
 - Tipurile de coloane trebuie definite in concordanta cu tipurile de date ;
 - Relatiile intre tabele se fac pe coloane de acelasi tip .



Integritatea datelor

- Constrainterile de integritate reprezinta anumite reguli pe care datele trebuie sa le respecte la nivel de tabela sau in relatiile cu alte tabele.
- Aceste reguli sunt verificate automat in cazul operatiilor de inserare, stergere si modificare, iar in cazul in care nu se valideaza se genereaza o eroare si tranzactia nu se efectueaza.
- In felul acesta este asigurata o mai mare siguranta in ceea ce priveste corectitudinea datelor si reducerea erorii umane.



Integritatea datelor

- Constrangerile de integritate pot fi :
 - **Valori nenule** – inregistrarile nu pot contine valori nule;
 - **Cheie unica** – defineste o cheie unica pe una sau mai multe coloane (nu pot fi mai multe inregistrari cu aceleasi valori pe coloanele respective, dar se accepta valori nule);
 - **Cheie primara** – defineste o cheie primara la nivel de coloana sau tabela (nu pot fi mai multe inregistri cu aceeasi cheie primara, care nu accepta valori nule);
 - **Cheie externa** – defineste o cheie externa (tabela se relateaza cu alta tabela pe o cheie unica sau cheie primara);
 - **Verificare** – se forteaza o verificare de indeplinire a unor conditii pe o coloana.



Manipularea datelor

- Manipularea datelor se refera la operatiile de lucru cu datele inregistrate intr-o baza de date si se realizeaza cu ajutorul unui limbaj DML (*Data Manipulation Language*) care este inclus in limbajul SQL;
- Operatii principale de manipulare a datelor sunt urmatoarele:
 - **Inserarea de date**(adaugarea de linii noi in tabele);
 - **Stergerea de date** (stergerea de linii din tabele);
 - **Modificarea datelor** (modificarea continutului unor linii existente in tabele);
 - **Interogarea datelor** (selectarea liniilor dupa anumite criterii de interogare).



Categorii de utilizatori

- **Utilizatori privilegiati** - Acestia sunt utilizatori care au dreptul de a afectua toate tipurile de operatii puse la dispozitie de catre sistem (**administratorul bazei de date**);
- **Utilizatori neprivilegiati** - Acestia sunt utilizatorii obisnuiti ai SGBD-ului si dispun de drepturile de acces care le-au fost alocate de catre administratorul bazei de date;
- **Dezvoltatori de aplicatii** - In aceasta categorie intra toti cei implicați in crearea unei aplicatii si in activitatea de mentenanta;
- **Utilizatori de aplicatii** - Sunt cei care au acces la baza de date prin intermediul interfetelor pentru care sunt autorizati.



Niveluri de reprezentare a datelor

- Exista mai multe niveluri de reprezentare a datelor, fiecare avand caracteristici specifice in functie de perspectiva sub care este vazuta, asa cum se vede in Figura 2:
 - **Nivelul intern** – descrie organizarea interna a datelor;
 - **Nivelul conceptual** – descriere modelul de date(de exemplu modelul relational);
 - **Nivelul extern** - nivelul utilizatorilor care acceseaza baza de date.



Niveluri de reprezentare a datelor

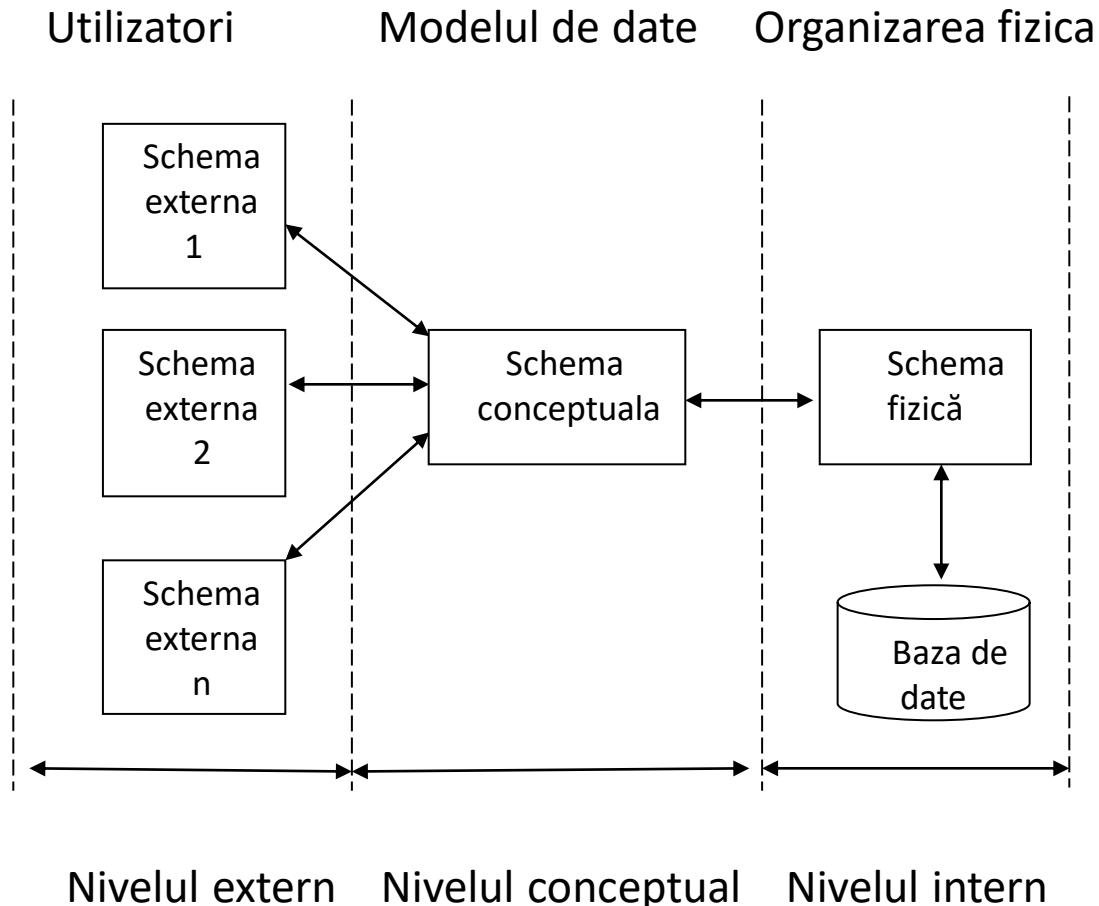


Figura 2. Niveluri de reprezentare a datelor



Nivelul intern

- Descrierea bazei de date la **nivel intern** poarta numele de **schema fizica** si sistemul de gestiune a bazelor de date pune la dispozitie facilitati pentru inregistrarea si modificarea acestora.
- La acest nivel baza de date este descrisa din perspectiva stocarii sale pe dispozitivele fizice: identificarea discurilor si a cailor unde este stocata, numele fisierelor care formeaza baza de date, structura fizica a acestora, etc.



Nivelul conceptual

- Descrierea bazei de date la **nivel conceptual** poarta numele de **schema conceptuala** (numita uneori si **schema logica**) a bazei de date.
- In modelul relational schema consta in:
 - Tabelele care formeaza baza de date;
 - Structura (coloanele) fiecarei tabele;
 - Tipul de date asociat coloanelor;
 - Elementele pe baza carora se realizeaza interconectarea tabelelor (coloane comune);
 - Constrainteri de integritate;
 - Operatii declansate automat la modificarea unor elemente ale bazei de date;



Nivelul extern

- Diferite categorii de utilizatori ai unei baze de date au nevoie în activitatea lor numai de anumite parti specifice ale schemei conceptuale.
- **Nivelul extern** se ocupă de descrierea acestor parti și poartă numele de **scheme externe** (echivalentul userilor creati pe baza de date).
- O baza de date are asociată o singură schema fizică și o singură schema conceptuală, dar mai multe scheme externe.
- Pentru administratorului bazei de date schema externă coincide cu schema conceptuală, deoarece are toate drepturile de acces.
- Celelalte categorii de utilizatori accesează baza de date doar prin intermediul schemelor externe specifice acestora, pe baza drepturilor de acces alocate.



Independenta datelor

- Existenta celor trei niveluri de descriere permite definirea conceptului de independenta intre datele stocate in baza de date si aplicatiile care utilizeaza aceste date.
- Conceptul de independenta a datelor a aparut odata cu dezvoltarea sistemelor complexe de aplicatii, pentru care cablarea informatiilor structurate in cadrul programului constituie o bariera in calea dezvoltarii si modificarii acestora.



Independenta datelor

- ***Independenta logica*** reprezinta posibilitatea de schimbare a schemei conceptuale a bazei de date fara modificarea schemelor externe. Conditia este ca modificarea sa nu elimine niciunul dintre elementele necesare translatiei de la schema externa la schema conceptuala.
- Unele operatii implica insa si modificarea definirii schemelor externe.
- Exemple de modificare a schemei conceptuale ar fi:
 - Adaugarea de noi tabele in baza de date;
 - Adaugarea de noi coloane in tabelele existente;
 - Adaugarea de noi constrangeri de integritate;
 - Modificarea numelor tabelelor si coloanelor existente;
 - Modificarea in anumite limite a tipurilor de date;
 - Restructurarea bazei de date.



Independenta datelor

Exemple:

- Sa consideram o baza de date continand la nivel conceptual o tabela cu date despre studenti cu urmatoarea structura:
Studenti (*nr_matricol*, *nume*, *cod_facultate*, *medie*)
si n scheme externe continand tabelele virtuale
Studenti-1, Studenti-2, ..., Studenti-n definite astfel:
Studenti-n (*nr_matricol*, *nume*, *cod_facultate*, *medie*)
care contine liniile din tabela **Studenti** unde *cod_facultate* = n .
- In cazul modificarii bazei de date prin adaugarea unei noi tabele cuprinzand lista specializarilor din cadrul facultatii si a unei noi coloane in tabela **Studenti** , pentru a specifica la ce specializare este inscris fiecare student, tabelele cu datele studentilor din schemele externe vor ramane aceleasi, daca schimbam structura astfel:



Independenta datelor

- Baza de date conceptuala:

Studenti (*nr_matricol, nume, cod_facultate, medie, cod_specializare*)

Specializari (*cod_specializare, den_specializare*)

- Schemele externe:

Studenti-n (*nr_matricol, nume, cod_facultate, medie*) va contine in continuare valorile de pe coloanele *nr_matricol, nume, cod_facultate* si *medie* din liniile din tabela **Studenti** unde *cod_facultate = n*.

- Independenta logica implica folosirea de catre SGBD a informatiilor de definire a schemelor externe, stocate in catalogul sistemului, pentru conversia oricarei operatii din structura schemei externe a aplicatiei sau utilizatorului care a lansat-o, in structura schemei conceptuale a bazei de date.



Independenta datelor

- ***Independenta fizica*** reprezinta posibilitatea de schimbare a schemei fizice a bazei de date fara modificarea schemei conceptuale si implicit a schemelor externe. Aceasta da posibilitatea reorganizarii fizice a bazei de date fara afectarea aplicatiilor care o folosesc.
- Alte operatii pot fi suportate numai prin modificarea catalogului sau a fisierelor de configurare pe care SGBD-ul le foloseste pentru a face translatia de la schema conceptuala la schema fizica, cum ar fi:
 - Schimbarea dispozitivelor fizice pe care este stocata baza de date;
 - Schimbarea numelor fisierelor fizice in care este stocata baza de date sau a directoarelor unde acestea sunt plasate;



Independenta datelor

- Adaugarea de noi structuri de cautare rapida (indexe) pentru cresterea vitezei de executie a unei operatii;
- Schimbarea in anumite conditii a structurii fizice a fisierelor bazei de date;
- Schimbarea unor parametri ai sistemului de gestiune care afecteaza modul in care datele sunt stocate la nivel fizic, de exemplu dimensiunea blocului de date.
- Deoarece nu toate sistemele de gestiune a bazelor de date implementeaza total cele trei niveluri de descriere, posibilitatea de a asigura cele doua tipuri de independenta a datelor este conditionata de facilitatile oferite de catre sistemul de gestiune.



Bibliografie

- Ramez Elmasri, Shamkant B. Navathe, *Fundamentals of Database Systems*, Pearson Education, Boston 2011
- Hector Garcia-Molina, Jeffrey D. Ullman, Jennifer D. Widom, *Database Systems: The Complete Book*, Prentice-Hall, Englewood Cliffs, NJ, 2002.
- J.D. Ullman, J. Widom: *A First Course in Database Systems*, Prentice Hall, first edition 1997, second edition 2002.
- Florin Radulescu, Alexandru Boicea, *Baze de date online*, Editura Oamenilor de Stiinta din Romania, 2011
- Alexandru Boicea, *Baze de date – Note de curs*, Universitatea Politehnica din Bucuresti
- Alexandru Boicea, *Oracle SQL, SQL*Plus*, Editura Printech, 2007



Capitolul 2

Introducere în algebra relațională



Elemente de algebra relationala

- Algebra relationala este unul dintre cele doua limbaje formale de interogare ale modelului relational si oferă mijloace puternice de a construi relații noi din alte relații date. Atunci cand relatiile date sunt reprezentate de date stocate, relațiile construite cu mijloacel algebrei pot fi raspunsuri la fraze de interogare asupra acestor informatii.
- Orice algebra permite construirea de expresii prin aplicarea unor operatori asupra unor operanzi atomici sau asupra altor expresii algebrice.
- In algebra relationala, **operanzii** sunt:
 - **variabile**, care reprezinta relații;
 - **constante**, care sunt relații finite.



Elemente de algebra relationala

- În algebra relatională “clasică” toti operanții și toate rezultatele expresiilor sunt multimi. Vom grupa **operatiile din algebra relatională** în patru clase:
 - **operatii specifice teoriei multimilor** (reuniune, intersecție, diferență), dar aplicate asupra relațiilor;
 - **operatii care îndepărtează parti ale unei relații** (selectie, proiecție);
 - **operatii care asociază tuplurile a două relații** (produs cartezian, jonctiune);
 - **operatia prin care sunt atribuite nume noi atributelor relației și/sau relației.**



Elemente de algebra relationala

- O proprietate fundamentală în algebra relatională constă în faptul că fiecare operator acceptă instantele unei relații (sau a două) în calitate de argumente și întoarce ca rezultat o alta instanță de relație. Aceasta proprietate permite folosirea compusă a operatorilor (operatia de compunere) pentru a forma fraze de interogare complexe.
- O astfel de fraza de interogare corespunde unei expresii algebrice relationale, care se definește recursiv ca fiind o relație și un operator algebric unar aplicat unei singure expresii (sau ca un operator algebric binar aplicat la două expresii).



Operatii pe multimi aplicate relatiilor

■ Reuniunea

Fie R, S relatii.

Reuniunea celor doua relatii este

$T = R \cup S$, unde

$T = \{ \text{tupluri } t_i, \text{ a. i. } t_i \in r \vee t_i \in s, \forall t_i \in t \}.$

Conditii: R, S au multimi identice de atribute, cu aceleasi domenii de valori.

Observatie: T nu contine duplicate.



Operatii pe multimi aplicate relatiilor

Exemplu:

R

Nr_matricol	Nume_student	Cod_grupa	Media
1011	Ionescu Silvia	331CC	9,50
1022	Popescu Daniel	332CC	9,15

S

Nr_matricol	Nume_student	Cod_grupa	Media
1011	Ionescu Silvia	331CC	9,50
1033	Popa Cornel	332CC	9,05

T = R ∪ S

Nr_matricol	Nume_student	Cod_grupa	Media
1011	Ionescu Silvia	331CC	9,50
1022	Popescu Daniel	332CC	9,15
1033	Popa Cornel	332CC	9,05



Operatii pe multimi aplicate relatiilor

■ Intersectia

Fie R, S relatii.

Intersectia relatiilor este $T = R \cap S$, unde

$T = \{ \text{tupluri } t_i, \text{ a. i. } t_i \in r \wedge t_i \in s, \forall t_i \in t \}$.

Observatie: T nu contine duplicate.

Exemplu:

$$T = R \cap S$$

Nr_matricol	Nume_student	Cod_grupa	Media
1011	Ionescu Silvia	331CC	9,50



Operatii pe multimi aplicate relatiilor

■ Diferenta

Diferenta poate fi definita astfel:

$$T = R - S, \text{ unde}$$

$$T = \{ \text{tupluri } t_i, \text{ a. i. } t_i \in r \wedge t_i \notin s, \forall t_i \in t \} \quad \text{sau}$$

$$T = S - R, \text{ unde}$$

$$T = \{ \text{tupluri } t_i, \text{ a. i. } t_i \in s \wedge t_i \notin r, \forall t_i \in t \}.$$

Exemplu:

$$T = R - S$$

Nr_matricol	Nume_student	Cod_grupa	Media
1022	Popescu Daniel	332CC	9,15

$$T = S - R$$

Nr_matricol	Nume_student	Cod_grupa	Media
1033	Popa Cornel	332CC	9,05

Observatie: $R - S \neq S - R$.



Operatii pe multimi aplicate relatiilor

■ Proiectia

Fie relatia R cu attributele A_1, A_2, \dots, A_n .

Fie, de asemenea, attributele A_1, A_2, \dots, A_k , a. î.

$\{A_1, A_2, \dots, A_k\} \subset \{A_1, A_2, \dots, A_n\}$.

Atunci, proiectia relatiei R pe attributele A_1, A_2, \dots, A_k (sau pe valorile acestor attribute), notata cu $\pi_{A_1, A_2, \dots, A_k}(R)$, este relatia obtinuta din R prin extragerea valorilor atributelor A_1, A_2, \dots, A_k (π - pi)



Operatii pe multimi aplicate relatiilor

Exemplu:

$\pi_{\text{ nume_student, media}}(S)$

Nume_student	Media
Ionescu Silvia	9,50
Popa Cornel	9,05



Operatii pe multimi aplicate relatiilor

- **Proiectia** se poate defini formal ca fiind relatie:

$\pi_{i_1, i_2, \dots, i_k} = \{ \text{tupluri } t_i, \text{ a. } \hat{t}. t_i = (a_1, a_2, \dots, a_k),$
iar in R \exists tuplul $t_j = (b_1, b_2, \dots, b_n)$, $a_j = b_j$ pentru
 $j = 1 \dots k \}$

Mai sus a_j , respectiv b_j , sunt valori ale atributelor corespunzatoare din multimile $\{ A_1, A_2, \dots, A_k \}$ si, evident, $\{ A_1, A_2, \dots, A_n \}$. Simbolurile a_1, a_2, \dots reprezinta coloane din R .



Operatii pe multimi aplicate relatiilor

■ Selectia

Prin definitie, operatia de selectie aplicata unei relatii R , notata cu $\sigma_F(R)$, consta in extragerea din R a acelor tupluri care indeplinesc formula (clauza) F . Schema relatiei obtinuta este aceeasi cu schema relatiei R , atributele fiind aranjate, prin conventie, in aceeasi ordine. Operanzii continuti in clauza F sunt constante sau atribut din schema relatiei R . Operatorii sunt fie operatori aritmetici uzuali, fie operatori logici(de comparatie, negatie, etc). *(σ –sigma)*



Operatii pe multimi aplicate relatiilor

Exemplu:

σ media > 9 \wedge media < 9.50 (**T**) , unde **T**= **R** \cup **S** ;

Nr_matricol	Nume_student	Cod_grupa	Media
1022	Popescu Daniel	332CC	9,15
1033	Popa Cornel	332CC	9,05



Operatii pe multimi aplicate relatiilor

■ Produsul cartezian

Fie relatiile **R** si **S** de aritati R_1, R_2 (domenii de valori).

Fie in **R** si **S** tuplurile $(r_{i1}, r_{i2}, \dots, r_{ik})$, respectiv $(s_{j1}, s_{j2}, \dots, s_{jp})$.

Formal, produsul cartezian **T** = **R** × **S** al relatiilor **R** si **S** se defineste prin:

$T = \{ \text{tupluri } t_{ij}, \text{ a. i. } t_i = (r_{i1} r_{i2} \dots r_{ik} s_{j1} s_{j2} \dots s_{jp}), \text{ unde}$
 $(r_{i1} r_{i2} \dots r_{ik}) \in R \text{ si } (s_{j1} s_{j2} \dots s_{jp}) \in S \}$.

- Tuplurile din **T** reprezinta toate asocierile posibile dintre tuplurile din **R** si tuplurile din **S**.



Operatii pe multimi aplicate relatiilor

Exemplu:

R

Nr_matricol	Nume_student	Cod_spec
1011	Ionescu Silvia	1
1022	Popescu Daniel	2
1033	Popa Cornel	1

S

Cod_spec	Den_specializare
1	TI
2	AIS

T = R x S

Nr_matricol	Nume_student	R.Cod_spec	S.Cod_spec	Den_specializare
1011	Ionescu Silvia	1	1	TI
1011	Ionescu Silvia	1	2	AIS
1022	Popescu Daniel	2	1	TI
1022	Popescu Daniel	2	2	AIS
1033	Popa Cornel	1	1	TI
1033	Popa Cornel	1	2	AIS



Operatii pe multimi aplicate relatiilor

- Observatie:

Numarul de atribute ale produsului cartezian

$T = R \times S$ este suma atributelor relatiilor R si S

($3 + 2 = 5$ atribute) iar numarul de tupluri este produsul numerelor de tupluri ale relatiilor R si S

($3 \times 2 = 6$ tupluri).



Operatii pe multimi aplicate relatiilor

■ Jonctiunea (*Join*)

Jonctiunea (numita *Join* sau *Theta-join*) este o operatie compusa, care implica efectuarea unui produs cartezian si a unei selectii.

Fie relatiile **R** si **S**, joinul lor (notat $R \bowtie_F S$) se obtine din produsul cartezian al relatiilor R si S urmat de o selectie dupa conditia F (numita si *conditie de join*). Operatorii din conditia F pot fi operatori aritmetici sau operatori logici.

$$R \bowtie_F S = \sigma_F (R \times S)$$



Operatii pe multimi aplicate relatiilor

Exemplu:

- Sa facem o selectie pentru studentii din **R** care au specializarea TI (cod_spec=1), facand apoi un join cu **S** pentru a extrage denumirea specializarii.
- ✓ In pasul intai se calculeaza produsul cartezian:

$$T = R \times S$$

Nr_matricol	Nume_student	R.Cod_spec	S.Cod_spec	Den_specializare
1011	Ionescu Silvia	1	1	TI
1011	Ionescu Silvia	1	2	AIS
1022	Popescu Daniel	2	1	TI
1022	Popescu Daniel	2	2	AIS
1033	Popa Cornel	1	1	TI
1033	Popa Cornel	1	2	AIS



Operatii pe multimi aplicate relatiilor

- ✓ În pasul al doilea se face o selectie pe relatia $T = R \times S$ cu conditia de join

$$F = r.cod_spec=1 \wedge r.cod_spec = s.cod_spec$$

$$R \bowtie_{r.cod_spec=1 \wedge r.cod_spec=s.cod_spec} S$$

Nr_matricol	Nume_student	R.Cod_spec	S.Cod_spec	Den_specializare
1011	Ionescu Silvia	1	1	TI
1033	Popa Cornel	1	1	TI

- În cazul în care condiția de join este una de egalitate, joinul se mai numește *equi-join*. În restul cazurilor se folosește sintagma *non-equi-join*.
- În limbajul SQL condițiile de join se pun în clauza WHERE a unei cereri SELECT sau UPDATE.



Operatii pe multimi aplicate relatiilor

■ Jonctiunea naturală(*Natural Join*)

Join-ul natural pentru doua relatii **R** si **S** (notat $R \bowtie S$)se obtine facand joinul celor doua relatii dupa conditia “coloanele cu acelasi nume si de acelasi tip au valori egale” si eliminand prin proiectie atributele duplicat (coloanele dupa care s-a facut join-ul).

- Observatie: Join-ul natural nu tine cont de semnificatia coloanelor dupa care se face join.
- In limbajul SQL se specifica in clauza WHERE prin sintaxa NATURAL JOIN .



Operatii pe multimi aplicate relatiilor

Exemplu:

Pentru exemplul anterior, daca se face un join natural dupa atributul Cod_spec, care este comun ambelor relatii, obtinem:

$R \bowtie S$

Nr_matricol	Nume_student	Cod_spec	Den_specializare
1011	Ionescu Silvia	1	TI
1022	Popescu Daniel	2	AIS
1033	Popa Cornel	1	TI



Operatii pe multimi aplicate relatiilor

■ Jonctiune externa (*Outer Join*)

Am vazut in exemplele anterioare ca un join simplu sau join natural returneaza acele linii care indeplinesc conditia de join (join simplu) sau conditia de egalitate a atributelor (denumire si valori) in cazul unui join natural. Joinul extern se foloseste atunci cand in rezultat dorim sa apara si liniile care nu indeplinesc o conditie de join (din lipsa de date), de exemplu studentii care nu au specificata specializarea pe coloana Cod_spec sau specializarile care nu au studenti asignati. In acest caz coloanele care nu au valori de corespondenta apar cu valoare nula.



Operatii pe multimi aplicate relatiilor

- Exista trei tipuri de join extern:
 - ✓ **Join extern stanga** (*left outer join*), in care in rezultat apar toate tuplurile relatiei din stanga operatorului.
Notatia este: $R \bowtie_L S$.
 - ✓ **Join extern dreapta** (*right outer join*), in care in rezultat apar toate tuplurile relatiei din dreapta operatorului.
Notatia este: $R \bowtie_R S$
 - ✓ **Join extern complet** (*full outer join*), in care in rezultat apar toate tuplurile relatiilor din stanga si din dreapta operatorului. Notatia este: $R \bowtie S$
- Observatie: In rezultatul joinului extern sunt intotdeauna continute tuplurile (liniile) din rezultatul joinului general dupa aceeasi conditie.



Operatii pe multimi aplicate relatiilor

Exemplu: R

Nr_matricol	Nume_student	Cod_spec
1011	Ionescu Silvia	1
1022	Popescu Daniel	2
1033	Popa Cornel	1
1044	Toma Diana	4

S	Cod_spec	Den_specializare
1	TI	
2	AIS	
3	SI	

$$T = R \times S$$

Nr_matricol	Nume_student	R.Cod_spec	S.Cod_spec	Den_specializare
1011	Ionescu Silvia	1	1	TI
1011	Ionescu Silvia	1	2	AIS
1011	Ionescu Silvia	1	3	SI
1022	Popescu Daniel	2	1	TI
1022	Popescu Daniel	2	2	AIS
1022	Popescu Daniel	2	3	SI
1033	Popa Cornel	1	1	TI
1033	Popa Cornel	1	2	AIS
1033	Popa Cornel	1	3	SI
1044	Toma Diana	4	1	TI
1044	Toma Diana	4	2	AIS
1044	Toma Diana	4	3	SI



Operatii pe multimi aplicate relatiilor

R◁○▷ $L(r.\text{cod_spec} = s.\text{cod_spec})$ **S**

Nr_matricol	Nume_student	R.Cod_spec	S.Cod_spec	Den_specializare
1011	Ionescu Silvia	1	1	TI
1022	Popescu Daniel	2	2	AIS
1033	Popa Cornel	1	1	TI
1044	Toma Diana	4	null	null

- Observatie: Rezultatul contine linii din **R**(studenti) care nu au **Cod_spec** corespondent in **S**(specializari).



Operatii pe multimi aplicate relatiilor

R $\bowtie_O\triangleright$ R(r.cod_spec=s.cod_spec) S

Nr_matricol	Nume_student	R.Cod_spec	S.Cod_spec	Den_specializare
1011	Ionescu Silvia	1	1	TI
1022	Popescu Daniel	2	2	AIS
1033	Popa Cornel	1	1	TI
null	null	null	3	SI

- Observatie: Rezultatul contine linii din **S**(specializari) care nu au Cod_spec corespondent in **R**(studenti).



Operatii pe multimi aplicate relatiilor

R▷◁O▷◁ S
(r.cod_spec=s.cod_spec)

Nr_matricol	Nume_student	R.Cod_spec	S.Cod_spec	Den_specializare
1011	Ionescu Silvia	1	1	TI
1022	Popescu Daniel	2	2	AIS
1033	Popa Cornel	1	1	TI
1044	Toma Diana	4	null	null
null	null	null	3	SI

- Observatie: Rezultatul contine linii din **R(studenti)** care nu au Cod_spec corespondent in **S(specializari)**, dar si linii din **S(specializari)** care nu au Cod_spec corespondent in **R(studenti)**.



Operatii pe multimi aplicate relatiilor

■ Semi-jonctiunea (*Semi-Join*)

Prin definitie, semi-jonctiunea relatiilor R si S , notata $R \triangleright < S$ este proiectia jonctiunii naturale a celor doua relatii pe atributele din R : $R \triangleright < S = \pi_R R \bowtie S$

O formula echivalenta pentru realizarea acestei operatiuni este $R \triangleright < S = R \bowtie \pi_{R \cap S}(S)$. Deci semi-join-ul lui R in raport cu S este o relatie care contine multimea tuplurilor lui R care participa la joinul natural cu S . In general, un semi-join nu este simetric, deci $R \triangleright < S \neq S \triangleright < R$. Diferenta intre join-ul conventional si semi-join este ca un semi-join retuneaza cel mult o linie din prima tabela atunci cand gaseste mai multe linii corespondente in a doua tabela care respecta conditia de join, adica nu retuneaza dubluri de linii. In limbajul SQL semi-join-ul se foloseste impreuna cu operatorii EXIST si IN.



Operatii pe multimi aplicate relatiilor

Exemplu:

Pentru exemplul anterior, daca se face un semi-join natural dupa atributul Cod_spec, care este comun ambelor relatii, obtinem:

$R \triangleright\!< S$

Nr_matricol	Nume_student	Cod_spec
1011	Ionescu Silvia	1
1022	Popescu Daniel	2
1033	Popa Cornel	1

$S \triangleright\!< R$

Cod_spec	Den_specializare
1	TI
2	AIS



Operatori pe multiset-uri

- **Multiset** este o multime in care se admit aparitii multiple ale unor elemente. Multimile conventionale, inclusiv relatiile, nu contin duplicate. In practica bazelor de date intr-o tabela sau un rezultat al unei cereri de interogare de date pot sa apară linii dupicat. In acest caz nu mai putem vorbi de relatiile (care nu permit tupluri dupicat) ci de multiseturi (*bags*). Primele SGBD care au folosit modelul relational au continut limbaje de interogare bazate pe algebra relationala. Aceste sisteme tratau relatiile ca multiset-uri, nu ca multimi conventionale, din ratiuni de eficienta a timpului de interogare. Cu alte cuvinte, daca utilizatorul nu cerea explicit ca tuplurile dupicat (prezente in realitate) sa fie eliminate, relatiile rezultate puteau contine duplicate.
- Prezentam pe scurt efectul unora dintre operatorii de mai sus aplicati multiset-urilor:



Operatori pe multiset-uri

■ Reuniunea

Rezultatul operatiei este asemanator cu al reuniunii din algebra relationala dar din rezultatul final nu se elimina duplicatele.

Exemplu:

R (contine notele studentilor care au urmat cursul BD1)

Nr_matricol	Nume_student	Nota
1011	Ionescu Silvia	9
1022	Popescu Daniel	7
1033	Popa Cornel	10
1044	Toma Diana	8

S (contine notele studentilor care au urmat cursul BD2)

Nr_matricol	Nume_student	Nota
1011	Ionescu Silvia	8
1022	Popescu Daniel	7
1033	Popa Cornel	10



Operatori pe multiset-uri

$T = R \cup S$ (contine notele studentilor care au urmat cursurile BD1 si BD2)

Nr_matricol	Nume_student	Nota
1011	Ionescu Silvia	9
1022	Popescu Daniel	7
1033	Popa Cornel	10
1044	Toma Diana	8
1011	Ionescu Silvia	8
1022	Popescu Daniel	7
1033	Popa Cornel	10

In urma operatiei de reuninune se obtine multiset-ul T care contine linii dupicat.



Operatori pe multiset-uri

■ Intersectia

Rezultatul operatiei este asemanator cu al intersectiei din algebra relationala dar din rezultatul final nu se elimina duplicatele.

Exemplu:

$T = R \cap S$ (contine notele studentilor care au urmat ambele cursuri si au obtinut aceeasi nota la fiecare dintre ele)

Nr_matricol	Nume_student	Nota
1022	Popescu Daniel	7
1033	Popa Cornel	10
1022	Popescu Daniel	7
1033	Popa Cornel	10

- In urma operatiei de intersectie se obtine multiset-ul T care contine linii dupicat.



Operatori pe multiset-uri

■ Proiectia

Are acelasi mod de calcul, ca si in cazul relatiilor, dar la final nu se elimina liniile dupicat.

Exemplu:

S

Nr_matricol	Nume_student	Disciplina	Nota	Data_examen
1011	Ionescu Silvia	BD1	9	6.06.2020
1022	Popescu Daniel	BD1	7	6.06.2020
1033	Popa Cornel	BD1	10	6.06.2020
1044	Toma Diana	BD1	8	6.06.2020
1011	Ionescu Silvia	BD2	8	1.02.2021
1022	Popescu Daniel	BD2	7	1.02.2021
1033	Popa Cornel	BD2	10	1.02.2021



Operatori pe multiset-uri

$T = \pi_{nr_matricol, nume_student, nota}(S)$

Nr_matricol	Nume_student	Nota
1011	Ionescu Silvia	9
1022	Popescu Daniel	7
1033	Popa Cornel	10
1044	Toma Diana	8
1011	Ionescu Silvia	8
1022	Popescu Daniel	7
1033	Popa Cornel	10

- Se observa ca multisetul T , obtinut prin proiectie, contine linii dupicat pentru studentii care au obtinut aceeasi nota la disciplinele BD1 si BD2. In acest caz o multime conventionala S (care nu contine linii dupicat) se poate transforma prin proiectie intr-un multiset T .



Operatori pe multiset-uri

- În mod similar relațiilor se calculează și **diferenta, produsul cartezian, selectia, joinul, joinul natural, joinul extern**, dar cu următoarele observații:
 - multiset-urile operand pot să contină linii dupăcat;
 - multiset-ul rezultat poate să contină liniile dupăcat.
- Observație: În cazul acestor operații nu pot apărea linii dupăcat decât dacă operanții contin linii dupăcat.



Operatori extinsi ai algebrei relationale

- Majoritatea limbajelor de interogare moderne se bazeaza pe definitiile operatiilor relationale, precum si pe cele privind tratarea multiset-urilor. Totodata, in practica actuala, limbajele de interogare SQL permit efectuarea unor operatii suplimentare, importante in aplicatii, cum ar fi:

- **Redenumirea**

Exista doua modalitati de a face redenumirea tabelelor si/sau coloanelor:

- **Operatorul de redenumire (ρ)** - permite atat redenumirea relatiilor/multiseturilor cat si a atributelor acestora:

Fiind data relata $R(A_1, A_2, \dots, A_n)$, putem obtine alta relatie $S(B_1, B_2, \dots, B_n)$, care are acelasi continut ca si R dar atributele se numesc B_1, B_2, \dots, B_n , unde $S = \rho_{R(A_1, A_2, \dots, A_n)} \cdot$ (ρ - rho)

- Intr-un limbaj SQL se foloseste pentru definirea alias-urilor de coloana/tabela folosite in cererile de interogare SELECT.



Operatori extinsi ai algebrei relationale

- **Constructorul** - permite redenumirea atributelor in rezultatul unei expresii relationale sau pe multiseturi, de exemplu putem redenumi intr-un rezultat un atribut prin constructia:

Nume_vechi \rightarrow Nume_nou

Exemplu: Fie o relatie

$R=(\text{cod_grupa}, \text{nr_matricol}, \text{nume_stud}, \text{nota}).$

In rezultatul proiectiei

$\pi_{\text{cod_grupa} \rightarrow \text{Grupa}, \text{nume_stud} \rightarrow \text{Nume}, \text{media_notelor} \rightarrow \text{Media_gen}}(R)$
atributele se vor numi : Grupa, Nume, Media_gen.

- In SQL se foloseste pentru aliasul de coloana intr-o cerere SELECT.



Operatori extinsi ai algebrei relationale

■ Eliminare duplicate

Acest operator se poate aplica doar pe multiseturi (relatiile contin tupluri duplikat) iar efectul este eliminarea duplikatelor din multiset.

- Notatia operatorului este urmatoarea:

Fiind dat un multiset R , atunci $\delta(R)$ este un multiset fara duplicate (practic devine o relatie).

(δ - *delta*)

- In SQL se foloseste optiunea DISTINCT intr-o cerere SELECT.



Operatori extinsi ai algebrei relationale

Exemplu:

T – multiset

Nr_matricol	Nume_student	Nota
1011	Ionescu Silvia	9
1022	Popescu Daniel	7
1033	Popa Cornel	10
1044	Toma Diana	8
1011	Ionescu Silvia	8
1022	Popescu Daniel	7
1033	Popa Cornel	10

$\delta(T)$ – relatie obtinuta din multiset-ul T

Nr_matricol	Nume_student	Nota
1011	Ionescu Silvia	9
1022	Popescu Daniel	7
1033	Popa Cornel	10
1044	Toma Diana	8
1011	Ionescu Silvia	8



Operatori extinsi ai algebrei relationale

■ Grupare

Acest operator este folosit pentru gruparea rezultatelor operatiilor dupa anumite criterii si se poate aplica atat relatiilor cat si multiseturilor.

Sintaxa operatorului de grupare este urmatoarea:

$\gamma_{\text{atribute\&functii}} (R)$ *(γ - gamma)*

unde:

- **attribute** - sunt criterii de grupare si apar in rezultatul returnat de operator;
- **functii** - sunt functii de grup (de ex.: MIN, MAX, SUM, AVG, COUNT) care se calculeaza la nivelul fiecarui grup si de asemenea apar in rezultatul operatorului.
- In SQL operatorul se aplica prin functii de grup impreuna cu clauza GROUP BY.



Operatori extinsi ai algebrei relationale

Exemplu:

S

Nr_matricol	Nume_student	Disciplina	Nota	Data_examen
1011	Ionescu Silvia	BD1	9	6.06.2020
1022	Popescu Daniel	BD1	7	6.06.2020
1033	Popa Cornel	BD1	10	6.06.2020
1044	Toma Diana	BD1	8	6.06.2020
1011	Ionescu Silvia	BD2	8	1.02.2021
1022	Popescu Daniel	BD2	7	1.02.2021
1033	Popa Cornel	BD2	10	1.02.2021

$T = \gamma_{disciplina, Count(*) \rightarrow Nr_stud, AVG(nota) \rightarrow Media_discip}(S)$

Disciplina	Nr_stud	Media_discip
BD1	4	8.50
BD2	3	8.33



Operatori extinsi ai algebrei relationale

▪ Sortare

Sintaxa operatorului de sortare este urmatoarea:

$$\tau_{\text{lista_atribute}}(\mathbf{R}) \quad (\tau - tau)$$

- Operatorul are ca efect sortarea(ordonarea) relatiei sau multisetului \mathbf{R} in functie de atributele din lista.
- Intr-o relatie, sau multiset, datele nu sunt neaparat ordonate (si nici nu este nevoie). Operatorul se aplica, de regula, atunci cand se fac interogari pentru intocmirea de liste ordonate dupa anumite atributе.
- In limbajul SQL se foloseste clauza ORDER BY dintr-o cerere SELECT.



Operatori extinsi ai algebrei relationale

Exemplu:

R

Nr_matricol	Nume_student	Cod_grupa
1011	Ionescu Silvia	331CC
1022	Popescu Daniel	332CC
1033	Popa Cornel	333CC
1044	Ene Diana	334CC

$$T = \tau_{\text{nume_student}}(R)$$

Nr_matricol	Nume_student	Cod_spec
1044	Ene Diana	334CC
1011	Ionescu Silvia	331CC
1033	Popa Cornel	333CC
1022	Popescu Daniel	332CC



Operatori extinsi ai algebrei relationale

▪ Proiectia extinsa

Acest operator este analog proiectiei obisnuite dar permite definirea de atribute(coloane) noi, obtinute prin calcule cu ajutorul unor expresii pe relatii sau multiseturi. Sintaxa este urmatoarea:

$$\pi_{\text{expresie}_1, \text{expresie}_2, \dots \text{expresie}_n} (R)$$

- Observatie: Dupa ce se calculeaza rezultatele, duplicatele se elimina sau nu se elimina, in functie de rezultatul dorit (relatie sau multiset).
- In SQL acest operator este implementat in cererea SELECT folosind operatori sau functii aplicate pe atribute(coloanele) deja definite.



Operatori extinsi ai algebrei relationale

Exemplu:

S

Nr_matricol	Nume_student	Disciplina	Nota	Data_examen
1011	Ionescu Silvia	BD1	9	6.06.2020
1022	Popescu Daniel	BD1	7	6.06.2020
1033	Popa Cornel	BD1	10	6.06.2020
1044	Toma Diana	BD1	8	6.06.2020
1011	Ionescu Silvia	BD2	8	1.02.2021
1022	Popescu Daniel	BD2	7	1.02.2021
1033	Popa Cornel	BD2	10	1.02.2021

$T = \pi_{nr_matricol, nume_student, AVG(nota) \rightarrow media_BD} (S)$

Nr_matricol	Nume_student	Media_BD
1011	Ionescu Silvia	8.50
1022	Popescu Daniel	7
1033	Popa Cornel	10
1044	Toma Diana	8



Capitolul 3

Modelul relational



Modelul relational

- Modul de organizarea a datelor intr-o baza de date trebuie sa respecte un anumit model, iar operatia in sine se numeste **modelarea datelor**.
- Asa cum am aratat intr-un capitol anterior, au existat mai multe modele de baze de date in evolutia sistemelor de gestiune a bazelor de date, fiecare aducand ceva nou pentru modelarea datelor si o crestere a performantelor.
- **Modelul relational** a fost introdus in anul 1970 de Edgar Frank Codd si este modelul cel mai utilizat de catre sistemele de gestiune aparute dupa anul 1980.
- Obiectele bazei de date se numesc **entitati** si sunt relationate intre ele prin anumite **constrangeri** care formeaza asa numita **diagrama de relatii**.



Modelul relational

- In **Modelul relational (MR)** datele sunt stocate sub forma tabelara si aduce o serie de avantaje fata de modelele anterioare:
 - datele sunt stocate sub forma de linii intr-o tabela;
 - s-a renuntat la folosirea pointerilor si navigarea prin structuri arborescente;
 - pentru accesarea datelor exista limbaje specializate de nivel inalt, de exemplu limbajul SQL;
 - permite introducerea de constrangeri de integritate, ceea ce duce la reducerea anomalilor si a redundantei datelor;
 - asigura un control riguros al drepturilor de acces;
 - ofera facilitati sporite de securitate a datelor.



Elementele modelului relational

■ Domeniul

Domeniul reprezinta o multime de valori si este identificat printr-un nume.

- Un domeniu se poate defini fie prin enumerarea elementelor sale, fie prin specificarea unor caracteristici definitorii ale acestora.

Exemplu:

- Culori = {rosu, galben, albastru, violet, verde}
- Note = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10} sau
 $\text{Note} = \{n \in \mathbb{N}^* \mid n \geq 1 \text{ si } n \leq 10\}$
- Char40 = {Multimea sirurilor de maxim 40 caractere}
- Intreg5 = {Multimea numerelor intregi pozitive din intervalul [0, 99999]}



Elementele modelului relational

- **Produsul cartezian pe domenii** este definit similar cu cel din teoria multimilor:

Fiind date n domenii D_1, D_2, \dots, D_n produsul cartezian al domeniilor este multimea sirurilor de domenii:

$$D_1 \times D_2 \times \dots \times D_n = \{ (v_1, v_2, \dots, v_n) \mid v_i \in D_i, i = 1, \dots, n \}$$

- Trebuie mentionat ca in sirul de domenii care participa la un produs cartezian unele se pot gasi in mod repetat, de exemplu:

$$PC = \text{Intreg} \times \text{Char40} \times \text{Intreg} \times \text{Char40}$$



Elementele modelului relational

■ Relatia

Relatia reprezinta o submultime a unui produs cartezian avand asociat un nume.

- Un exemplu de relatie apartinand produsului cartezian $PC = \text{Studenti} \times \text{Specializari}$ este:

Specializari _stud

```
{   (1011, 'Ionescu Silvia', 1, 'CTI'),  
    (1022, 'Popescu Daniel', 2, 'AI'),  
    (1033, 'Popa Cornel', 1, 'CTI') }
```

- **Elementele unei relatii** sunt denumite in literatura de specialitate **tupluri** (tuple). Relatia de mai sus contine 3 dintre elementele produsului cartezian din care provine (3 tupluri).



Elementele modelului relational

- O reprezentare intuitiva pentru relatia de mai sus, in care fiecare element al relatiei devine o linie a unei tabele si fiecare coloana corespunde unui domeniu din produsul cartezian de baza, este urmatoarea :

Specializari_stud

1011	Ionescu Silvia	1	CTI
1022	Popescu Daniel	2	AII
1033	Popa Cornel	1	CTI

- Deci o relatie se poate reprezenta ca o tabela de date, fiecare coloana avand asociat un anumit tip de date, impus de domeniul din care provine.



Elementele modelului relational

■ Atributul

Atributul reprezinta o coloana a unei relatii avand asociat un nume.

- Deoarece o relatie are o reprezentare tabelara putem vorbi de “coloana a unei relatii” si putem sa facem urmatoarea similitudine: “atributele unei relatii” reprezinta “coloanele unei tabele”.
- Pentru relativa **Specializari_stud** putem defini urmatoarele atribute:
 - Nr_matricol – matricolul studentului
 - Nume_student – numele studentului
 - Cod_spec – codul specializarii
 - Den_specializare – denumirea specializarii



Elementele modelului relational

■ Schema

Schema unei relatii reprezinta structura relatiei compusa din **nume, attribute, domeniul atributelor si constrangerile de integritate asociate.**

- Datele dintr-o relatie se pot schimba in timp deoarece se pot adauga, modifica sau sterge linii, dar structura relatiei ramane constanta.
- Exista mai multe modalitati prin care se poate specifica schema unei relatii. In exemplele urmatoare prezentam cateva dintre acestea cu referire la relatia Medii:



Elementele modelului relational

Medii = Nr_matricol, Nume_student, Cod_grupa, Media

Medii (Nr_matricol, Nume_student, Cod_grupa, Media)

Medii (Nr_matricol: Intreg, Nume_student: Char40,
Cod_grupa: Char5, Media: Numar)

Medii (Nr_matricol intreg, Nume_student char(40),
Cod_grupa char(5), Media numar)

- În teoria bazelor de date relationale se foloseste și notatia urmatoare pentru o relatie:

R = ABCDE ,

care specifică ca schema relației **R** conține 5 attribute notate cu A, B, C, D și E.



Elementele modelului relational

- **Cheia**

Cheia unei relatii este o multime minimala de atribute ale caror valori identifica in mod unic un tuplu al relatiei respective. Deoarece o relatie nu accepta linii dupicat inseamna ca ele pot fi identificate in mod unic prin una sau mai multe coloane. Cheia unei relatii este o caracteristica a schemei acesteia si se defineste ca o constrangere de integritate.

- In tabela **Medii** tuplurile pot fi identificate in mod unic prin atributele (Nr_matricol, Nume_student) sau (Nr_matricol, Nume_student, Cod_grupa) dar ele nu indeplinesc si conditia de minimalitate.



Elementele modelului relational

- Daca consideram ca numarul matricol este unic pentru fiecare student atunci putem defini atributul Nr_matricol ca fiind cheia relatiei Medii, fiind indeplinita si conditia de minimalitate.
- Niciunul dintre celelalte atribute, luate separat, nu poate fi cheie, de exemplu Nume_student nu este cheie deoarece pot exista doi studenti cu acelasi nume si in acest caz nu pot fi identificate unic tuplurile .
- Daca facem referire la tabele, rezulta ca nu pot exista doua linii avand aceeasi combinatie de valori pe coloanele care formeaza cheia tabelei respective (proprietaate denumita in literatura de specialitate si unicitatea cheii).



Elementele modelului relational

- Trebuie specificat ca intr-o relatie se pot identifica uneori mai multe chei. Sa luam ca exemplu relatia: **Studenti(Nr_matricol, Nume_student, Adresa, CNP, Data_nastere)**
- Dupa cum se stie CNP-ul unei persoane este unic asa ca relatia Studenti poate avea doua chei Nr_matricol si CNP, ambele indeplinind conditia de unicitate si minimalitate.
- Observatie: Deoarece intr-o relatie nu pot exista doua tupluri identice, rezulta ca multimea tuturor atributelor relatiei formeaza o cheie sau contine cel putin o cheie, deci orice relatie are cel putin o cheie.



Elementele modelului relational

- În literatura de specialitate și în sistemele de gestiune a bazelor de date există alte trei concepte legate de cheie și care vor fi prezentate în acest capitol și capitolele următoare:
- **Cheie primara** (*Primary Key*)
- **Cheie străină** (*Foreign Key*)
- **Supercheie** (*Superkey*)



Elementele modelului relational

▪ Valori nule

Valoare nula (null value) este o valoare care modeleaza o informatie nespecificata si este o informatie inaplicabila pentru prelucrari de date(nu se pot aplica operatori aritmetici pe ea).

- Uneori, unele atribute ale unei relatii nu au nicio valoare specificata si in acest caz se spune ca atributul respectiv are o valoare nula.

Exemplu:

Studenti

Nr_matricol	Nume_student	CNP	Cod_Grupa
1011	Ionescu Silvia	2901028400772	331CC
1022	Popescu Daniel	<null>	332CC
1033	Popa Cornel	1911115451664	<null>



Elementele modelului relational

- În exemplul de mai sus studentul Popescu Daniel nu are specificat CNP-ul și spunem că atributul CNP are valoare nula. Înregistrarea poate fi totuși identificată unic după atributul Nr_matricol care este cheie. Dacă se dorește însă să se identifice studentul după CNP atunci cererea de interogare nu va returna nicio linie.
- Valorile nule pot fi modificate ulterior prin comenzi DML, de exemplu în SQL se folosește comanda UPDATE.
- Observație: Într-o tabelă o cheie care identifică unic un tuplu nu trebuie să contină attribute cu valoare nula.



Elementele modelului relational

- **Constrangeri de integritate**

Constrangerile de integritate reprezinta un mecanism prin care un SGDB poate controla corectitudinea datelor in cazul operatiilor DML de manipulare a datelor. In cazul violarii acestor constrangeri de integritate SGBD-ul va rejecta orice tranzactie pe tabela respectiva.

- ✓ **Constrangerea de valori nenule(NOT NULL)**

Este o constrangere care se aplica numai la nivel de atribut(coloana) si verifica daca inregistrarile au valori nule pe coloanele respective, fortand un cod de eroare care anuleaza tranzactia. Orice incercare de a adauga o linie care contine valori nule pe acea coloana sau de a modifica o valoare nenula intr-una nula va genera o eroare sistem.



Elementele modelului relational

- Cand se creeaza constrangeri pe o cheie primara se creeaza automat si o costrangere NOT NULL pe atributele respective (o cheie primara nu trebuie sa contina valori nule pe coloanele care o definesc).

Exemplu:

Daca definim atributul CNP de tip NOT NULL atunci cand se insereaza o linie sau se modifica valoarea CNP-ul in valoare nula (se sterge valoarea) SGDB-ul va genera o eroare si operatia se anuleaza.

✓ **Constrangerea de unicitate (UNIQUE)**

Se foloseste cand vrem ca un atribut(coloana) sau perechi de atribut sa nu contina valori duplicate.



Elementele modelului relational

- Verificarea se face numai pentru inregistrari cu valori nenule deoarece constrangerea de unicitate permite inserarea de valori nule in coloanele respective.
- Daca coloanele definite unice sunt si not null, atunci putem considera unicitatea ca o cheie unica.
- In mod automat se creeaza si un index pe coloanele definite unice, ceea ce duce la cresterea vitezei de interogare pe tabela.

Exemplu: Daca definim atributul CNP de tip UNIQUE, la orice operatie de inserare sau modificare de date, SGDB-ul face o verificare automata de unicitate si in caz de duplicare se genereaza o eroare care anuleaza operatia.



Elementele modelului relational

✓ **Constrangerea de cheie primara (PRIMARY KEY)**

Se foloseste pentru definirea unei chei primare la nivel de coloana (cheia contine o singura coloana) sau la nivel de tabela (cheia contine una sau mai multe coloane).

- O tabela poate avea o singura cheie primara.
- Nu se accepta valori nule pentru niciuna dintre coloanele care definesc o cheie primara.
- Cand se creeaza o cheie primara timpul de raspuns in cazul unei interogari se reduce foarte mult.



Elementele modelului relational

Exemplu:

- Daca definim atributul Nr_matricol cheie de tip PRIMARY KEY atunci cand se face o inserare sau o modificare de linie se face o verificare de unicitate dar si de valori nenule(nu se accepta valori nule ale atributelor care compun cheia primara) pe coloana de matricole.
- In caz de violare a constrangerii se anuleaza operatia.
- Daca avem o cheie primara compusa(pe mai multe coloane) verificarea se face pe toate coloanele care compun cheia primara, nu pe coloane individuale.



Elementele modelului relational

✓ **Constrangerea de cheie strina (FOREIGN KEY)**

Acest tip de constrangere se foloseste pentru relationarea unei tabele cu una sau mai multe tabele, verificand daca valorile atributelor definite ca FOREIGN KEY (cheie strina sau cheie externa) sunt cuprinse in valorile coloanelor altel tabele care trebuie sa fie definite UNIQUE sau PRIMARY KEY.

Mai exact, o relationare cu cheie externa pe o tabela se poate face numai daca coloanele de referinta formeaza o cheie primara sau au unicitate.



Elementele modelului relational

- Cand se face relationarea intre doua tabele trebuie avute in vedere urmatoarele reguli de functionare :
 - Inserarea unei linii intr-o tabela relationata (pe care am definit FOREIGN KEY) se poate face daca exista numai o singura linie in tabela de referinta (in care am definit PRIMARY KEY sau UNIQUE) corespunzator coloanelor de relationare;
 - Coloanele definite FOREIGN KEY accepta valori nule;
 - Stergerea unei linii din tabela de referinta nu se poate face atata timp cat exista linii relationate in tabela relationata, pe linia respectiva;
 - Regulile de mai sus sunt valabile si in cazul relationarii pe coloane.



Elementele modelului relational

Exemplu: Consideram relatiile:

Specializari (Cod_spec, Den_specializare) si

Studenti (Nr_matricol, Nume_student, Cod_grupa, Cod_spec).

- Tabela **Specializari** va contine toate specializarile din facultate, fiecare avand un cod unic.
- Daca definim in tabela **Studenti** atributul Cod_spec ca fiind FOREIGN KEY cu referinta pe atributul Cod_spec din tabela **Specializari**, in momentul in care se face o operatie de inserare sau modificare de linie in tabela **Studenti** SGDB-ul face o verificare daca valoarea Cod_spec se afla printre valorile de pe coloana Cod_spec din tabela **Specializari** , iar daca rezultatul este negativ operatia se anuleaza. Coloana Cod_spec din tabela **Specializari** trebuie sa contine valori unice, adica trebuie obligatoriu sa fie o cheie primara/unica.



Elementele modelului relational

✓ **Constrangerea de conditie (CHECK)**

Acum tip de constrangere se foloseste pentru a forta valorile unei coloane sa verifice o conditie prestabilita.

Conditia poate sa contina si functii, cu unele exceptii (sysdate, user, unele functii de tip data calendaristica etc.) .

Exemplu: Consideram relatia

**Note(Nr_matricol, Nume_student, Cod_grupa,
Disciplina, Data_examen, Nota).**

- Pentru evitarea erorilor de operare se poate defini coloana Nota de tip CHECK cu conditia ($Nota \geq 1$ and $Nota \leq 10$) sau conditia $\text{max}(Nota) \leq 10$.



Expresii sigure. Domenii de siguranta

- **Definitie:** Relatia $P = \{ t \mid \neg R(t) \}$ in calculul relational reprezinta toate tuplurile posibile de lungime t (aritatea lui R) care nu apartin relatiei R .
- Se pune problema cum putem afla “toate tuplurile posibile” si care este domeniul lor de valori.
- **Definitie:** O **expresie** $\{ t \mid \Psi(t) \}$ poate fi considerata **sigura**, daca se poate demonstra ca fiecare componenta a oricarui tuplu t , care satisface formula Ψ , apartine domeniului $DOM(\Psi)$.
- In expresie, t este o **variabila tuplu** iar Ψ este o **formula** si semnifica “multimea tuturor tuplurilor t care verifica formula Ψ ”.
- Se doreste eliminarea expresiilor $\{ t \mid \neg \Psi(t) \}$ considerate “nesigure” si considerarea numai a expresiilor care sunt “sigure”.



Expresii sigure. Domenii de siguranta

- **Definitie:** Domeniul expresiei, notat $\text{DOM}(\Psi)$ reprezinta multimea elementelor care:
 - fie apar explicit in formula Ψ ,
 - fie sunt componente ale unui tuplu oarecare, al unei relatii R oarecare, mentionata in formula Ψ .
- Trebuie remarcat ca $\text{DOM}(\Psi)$ nu se determina prin simpla inspectare a formulei Ψ , ci este determinat in functie de relatiile care se substituie efectiv variabilelor din Ψ .
- $\text{DOM}(\Psi)$ este intotdeauna finit deoarece consideram ca toate relatiile sunt finite.

Exemple:

- 1) Daca R este o relatie cu atributele A_1, A_2 si daca $\Psi[t] = R[t] \wedge t[1] = c \wedge R(t)$, atunci $\text{DOM}(\Psi)$ este o relatie unara data de formula algebraica relationala: $\{c\} \cup \pi_{A_1}(R) \cup \pi_{A_2}(R)$.



Expresii sigure. Domenii de siguranta

2) Fie $\Psi(t)$ o formula care nu incalca regulile de siguranta.

- Atunci, orice expresie de forma $\{ t \mid R(t) \wedge \Psi(t) \}$ este sigura, deoarece orice tuplu t care satisface $R(t) \wedge \Psi(t)$ este in R , de unde rezulta ca fiecare dintre componentele sale este in $\text{DOM}(R(t) \wedge \Psi(t))$.

3) Formula pentru diferenta a doua multimi R si S

$\{ t \mid R(t) \wedge \neg S(t) \}$, este de forma mentionata mai sus, cu $\Psi(t) = \neg S(t)$.

Formula pentru selectie este, de asemenea, de aceeasi forma.



Expresii sigure. Domenii de siguranta

- Generalizare a celor de mai sus: observam ca orice formula $\{ t \mid R_1(t) \vee R_2(t) \vee \dots \vee R_k(t) \wedge \Psi(t) \}$ este, de asemenea, sigura; $t[i]$ trebuie sa fie un simbol care apare in componenta i a unui tuplu oarecare al unei relatii oarecare R_j . Remarcam, de asemenea, ca formula data pentru reuniune intr-un capitol anterior, este de aceasta forma, dar lipsind Ψ . Cu alte cuvinte, putem lua pe Ψ ca fiind o formula intotdeauna adevarata, ca $t[1] = t[1]$.
- O expresie sigura se poate aplica si pe o proiectie a lui R :
$$\{ t^{(m)} \mid (\exists u_1) (\exists u_2) \dots (\exists u_k) (R_1(u_1) \wedge R_2(u_2) \wedge \dots \wedge R_k(u_k) \wedge t[1] = u_{i1}[j_1] \wedge t[2] = u_{i2}[j_2] \wedge \dots \wedge t[m] = u_{im}[j_m] \wedge \Psi(t, u_1, u_2, \dots, u_k)) \}$$
. Aici, componenta $t[1]$ este constransa la a fi un simbol care apare in cea de-a j_1 componenta a unui tuplu.



Calcul relational pe tupluri

- Pe langa algebra relationala, cererile de regasire a informatiei intr-o baza de date relationala pot fi exprimate si prin **calcul relational pe tupluri** (CRT) sau **calcul relational pe domenii** (CRD). In acest paragraf sunt prezentate pe scurt aceste doua modalitati de exprimare a cererilor.
- **Calcul relational pe tupluri**

In calcul relational pe tupluri o cerere se exprima printr-o **expresie** de forma: $\{ t \mid \Psi(t) \}$ unde **t** este o variabila tuplu, iar **Ψ** o formula. Semnificatia expresiei este “multimea tuturor tuplurilor **t** care verifica formula **Ψ** ”.



Calcul relational pe tupluri

Formula Ψ este compusa din elemente (numite si atomi) care pot fi de trei tipuri:

- Elemente de tip $R(s)$, unde R este un nume de relatie iar s o variabila tuplu. Semnificatia este “ s este un tuplu din R ”;
- Elemente de tip $s[i] \theta v[j]$, unde s si v sunt variabile tuplu iar θ un operator prin care se poate compara componenta i a variabilei tuplu s cu componenta j a variabilei tuplu v ;
- Elemente de tip $s[i] \theta a$ sau $a \theta s[i]$, prin care componenta i a variabilei tuplu s se compara cu constanta a .
(θ - theta)



Calcul relational pe tupluri

Pe baza acestor atomi se poate defini recursiv ce este o formula si ce sunt aparitiile libere sau legate ale variabilelor tuplu:

- Orice atom este in acelasi timp formula. Toate aparitiile unei variabile tuplu intr-un atom sunt aparitii libere.
- Daca Ψ si \emptyset sunt doua formule, atunci $\Psi \vee \emptyset$, $\Psi \wedge \emptyset$ si $\neg\Psi$ sunt formule cu semnificatia “ Ψ sau-logic \emptyset ”, “ Ψ si-logic \emptyset ” si respectiv “not Ψ ”. Aparitiile de variabile tuplu sunt libere sau legate in aceste formule, dupa cum ele sunt libere sau legate in componentele acestora. Este permis ca o aceeasi variabila tuplu sa aiba o aparitie libera in Ψ si o alta legata in \emptyset .



Calcul relational pe tupluri

- Daca Ψ este o formula atunci si $(\exists s)(\Psi)$ este formula.
 - Aparitiile variabilelor tuplu s , care sunt libere in Ψ sunt legate in $(\exists s)(\Psi)$.
 - Celealte aparitii de variabile tuplu din Ψ raman la fel (libere sau legate) in $(\exists s)(\Psi)$.
 - Semnificatia acestei formule este urmatoarea: exista o valoare concreta a lui s care inlocuita in toate aparitiile libere din Ψ face ca formula sa fie adevarata.
- Parantezele pot fi folosite in formule dupa necesitati. Precedenta este: intai comparatiile, apoi \exists si \forall si in final \neg , \wedge , \vee (in aceasta ordine).



Calcul relational pe tupluri

- Daca Ψ este o formula atunci si $(\forall s)(\Psi)$ este formula.
 - Aparitiile variabilei tuplu s care sunt libere in Ψ sunt legate in $(\forall s)(\Psi)$.
 - Celelalte aparitii de variabile tuplu din Ψ raman la fel (libere sau legate) in $(\forall s)(\Psi)$.
 - Semnificatia acestei formule este urmatoarea: orice valoare concreta a tuplului s , pusa in locul aparitiilor libere ale acestuia din Ψ , face ca formula sa fie adevarata.



Calcul relational pe tupluri

Exemple de expresii si formule:

- Expresia $\{t \mid R(t) \vee S(t)\}$ este echivalenta reuniunii a doua relatii din algebra relationala. Analog $\{t \mid R(t) \wedge S(t)\}$ reprezinta intersectia a doua relatii.
- Expresia pentru proiectia lui R pe atributele i_1, i_2, \dots, i_k se poate scrie astfel: $\{t^{(k)} \mid (\exists u)(R(u) \wedge t[1] = u[i_1] \wedge t[2] = u[i_2] \wedge \dots \wedge t[k] = u[i_k])\}$
- Formula $(\exists s)(R(s))$ spune ca relatia R este nevida.
- Din pacate, unele dintre expresiile scrise in calcul relational pe tupluri duc la rezultate infinite. De exemplu, daca R este o relatie finita, expresia $\{t \mid R(t)\}$ este de asemenea finita, dar expresia $\{t \mid \neg R(t)\}$ este infinita (exista o infinitate de tupluri care nu apartin lui R).



Calcul relational pe tupluri

- Pentru a evita astfel de rezultate s-a introdus notiunea de **expresii sigure**. Pentru definirea lor se poate folosi si un alt concept similar, numit **domeniul unei formule**:
- **Definitie:** Daca Ψ este o **formula** atunci domeniul sau, notat cu **DOM(Ψ)**, este **multimea tuturor valorilor** care, fie apar explicit in Ψ , fie sunt componente ale tuplurilor relatiilor prezente in Ψ . Cum orice relatie este finita rezulta ca si domeniul oricarei formule este finit.



Calcul relational pe tupluri

Exemplu:

Fie formula $\Psi = R(t) \wedge t[1] > 100$

care reprezinta conditia pentru o selectie din R dupa conditia “valoarea pe prima coloana este mai mare decat 100”.

Atunci:

$\text{DOM}(\Psi) = \{ 100 \} \cup \{\text{multimea valorilor care apar in tuplurile lui } R\}$.



Calcul relational pe tupluri

- Se poate deci afirma ca o expresie Ψ este sigura daca rezultatul sau este compus doar din valori apartinand lui $\text{DOM}(\Psi)$. Conform acestei definitii:
 - expresiile $\{t \mid R(t)\}$, $\{t \mid R(t) \wedge t[1] > 100\}$,
 $\{t \mid R(t) \vee S(t)\}$, $\{t \mid R(t) \wedge S(t)\}$ sau
 $\{t^{(k)} \mid (\exists u)(R(u) \wedge t[1] = u[i_1] \wedge t[2] = u[i_2] \wedge \dots \wedge t[k] = u[i_k])\}$ sunt sigure;
 - expresiile $\{t \mid \neg R(t)\}$ sau $\{t \mid \neg R(t) \wedge \neg S(t)\}$ nu sunt sigure.
- In literatura de specialitate se gaseste demonstratia faptului ca expresiile sigure din CRT sunt echivalente cu expresii din algebra relationala si reciproc.



Calcul relational pe domenii

▪ **Calcul relational pe domenii**

In calculul relational pe domenii in locul variabilor tuplu folosim **variabile de domeniu** care sunt elementele ale tuplurilor. In acest caz rescriem regulile de formare pentru o formula astfel:

- Un **atom** poate fi:
 - $R(x_1, x_2, \dots, x_n)$, unde R este o relatie iar x_i sunt variabile de domeniu sau constante;
 - $x \theta y$, unde x si y sunt variabile de domeniu sau constante , iar θ este in continuare un operator de comparatie.



Calcul relational pe domenii

- Formulele din CRD sunt construite analog cu cele din CRT utilizand de asemenea \neg , \wedge , \vee precum si \exists , \forall .
- Notiunea de aparitie libera sau legata a unei **variabile de domeniu** este analoaga cu cele din CRT.
- Analog cu CRT se definesc: **domeniul unei variabile de domeniu** $\text{DOM}(x)$ si **expresii sigure** in CRD.
- In literatura de specialitate se poate gasi demonstratia faptului ca expresiile sigure din CRD sunt echivalente cu expresii din algebra relationala si reciproc.



Calcul relational

Exemplu:

✓ Expresiile de mai jos sunt sigure:

- Reuniunea a două relații R și S:

$$\{x_1 x_2 \dots x_n \mid R(x_1 x_2 \dots x_n) \vee S(x_1 x_2 \dots x_n) \}$$

- Intersecția a două relații R și S:

$$\{x_1 x_2 \dots x_n \mid R(x_1 x_2 \dots x_n) \wedge S(x_1 x_2 \dots x_n) \}$$

- Selectia după condiția “valoarea pe prima coloană este mai mare decat 100”:

$$\{x_1 x_2 \dots x_n \mid R(x_1 x_2 \dots x_n) \wedge x_1 > 100 \}$$

✓ Analog cu CRT, expresiile de mai jos nu sunt sigure:

$$\{x_1 x_2 \dots x_n \mid \neg R(x_1 x_2 \dots x_n) \} \text{ si}$$

$$\{x_1 x_2 \dots x_n \mid \neg R(x_1 x_2 \dots x_n) \wedge \neg S(x_1 x_2 \dots x_n) \}$$



Capitolul 4

Modelul entitate-asociere



Modelul Entitate-Asociere

- **Modelul Entitate-Asociere** (*Entity Relationship*) este un model de date folosit în proiectarea conceptuală de nivel înalt a bazelor de date și poate fi transformat în model relational prin aplicarea unor reguli specifice.
- Modelul EA se constituie într-o abordare grafică a proiectării bazelor de date și a fost adoptat de comunitatea științifică, precum și de producătorii de software din domeniu, datorită simplității și expresivității sale.
- Acest model a fost introdus de P. P. Chen în 1976 și a fost utilizat de multe sisteme CASE pentru proiectarea bazelor de date DB2, Oracle, SQL Server, Sybase, s.a.



Modelul Entitate-Asociere

- Pana la aparitia unor astfel de metodologii se pornea de la o colectie de tabele si de la dependentele functionale sau multivalorice asociate si se ajungea, prin aplicarea unor algoritmi sau metode de normalizare, la schema dorita a bazei de date.
- Aceasta abordare de tip *bottom-up* creeaza insa dificultati in cazul bazelor de date complexe in care numarul de tabele si de dependente este mare.
- Probabilitatea ca unele interdependente intre date sa nu fie sesizate in procesul de proiectare este in aceste cazuri ridicata si pot duce la anomalii in exploatarea aplicatiilor.



Modelul Entitate-Asociere

- Impactul conceptelor de abstractizare si generalizare precum si elaborarea unui model de descriere informala a datelor, prin modelul Entitate-Asociere (EA), au dus la gasirea unor cai moderne de proiectare a bazelor de date. Modelul EA este unul dintre cele mai utilizate modele datorita intuitivitatii si simplitatii elementelor sale. Im bunatatirile aduse ulterior prin folosirea abstractizarilor si generalizarilor au dus la crearea de variante ale modelului, doua dintre acestea fiind descrise in acest capitol.
- Extensiile modelului EA au aparut si pentru alte necesitati:
 - modelarea cerintelor de secretizare a datelor;
 - elaborarea documentatiei pentru aplicatiile cu baze de date si usurarea comunicarii intre dezvoltatorul de sistem si utilizator;
 - proiectarea bazelor de date complexe pe portiuni si integrarea ulterioara a acestora (asa numita integrare a vederilor).



Modelul Entitate-Asociere

- Prin aplicarea modelului se obtine *diagrama entitate-asociere* care poate fi apoi translatata in modelul de date folosit de sistemul de gestiune a bazei de date.
- Cateva caracteristici ale modelului sunt urmatoarele:
 - Nu este legat direct de niciunul dintre modelele folosite de sistemele de gestiune a bazelor de date (relational sau orientat obiect) dar exista algoritmi bine pusi la punct de transformare din model EA in celealte modele de date;
 - Este intuitiv, rezultatul modelarii fiind o diagrama care defineste atat datele stocate in baza de date cat si interdependentele dintre acestea;
 - Proiectarea se poate face pe module, diagramele partiale rezultate putand fi apoi integrate pe baza unor algoritmi si metode bine puse la punct;
 - Este usor de inteles si interpretat.



Elementele modelului Entitate-Asociere

- Modelul entitate-asociere permite reprezentarea informatiilor despre structura bazelor de date folosind trei elemente de constructie: **entitati**, **attribute** ale entitatilor si **asocieri** intre entitati.
- **Entitati**

Entitatile modeleaza clase de obiecte concrete sau abstracte despre care se colecteaza informatii, au existenta independenta si pot fi identificate in mod unic. Ele definesc de obicei obiecte sau evenimente cu importanta informationala. Membrii unei clase care formeaza o astfel de entitate poarta numele de **instante** ale acelei entitati.

Exemple de entitati: Studenti, Produse, Facturi, etc.



Elementele modelului Entitate-Asociere

Entitatea este un obiect generic care reprezinta multimea tuturor instantelor sale si sunt de doua categorii:

- **Entitati independente** (sau tari) sunt cele care au existenta independenta de alte entitati.
- **Entitati dependente** (sau slabe) sunt formate din instante care isi justifica incadrarea in clasa respectiva doar atata timp cat intr-o alta entitate (tata) exista o anumita instanta de care sunt dependente.

Exemplu:

Daca consideram o baza de date universitara putem considera ca entitatea STUDENTI este entitate independenta iar entitatea NOTE entitate dependenta, deoarece notele unui student sunt dependente de existenta studentului in facultatea respectiva.



Elementele modelului Entitate-Asociere

Element	Tip	Reprezentare	Exemplu
Entitate	Tare		
	Slaba		
Atribut	Identificare		
	Descriere		
Asociere	Asociere $n=2$ entitati	Nume asociere 	Inscris_la
	Asociere $n > 2$ entitati	 	Alocare

Figura 1. Convenția de reprezentare a elementelor modelului EA



Elementele modelului Entitate-Asociere

■ Atribute

Atributele modeleaza proprietati atomice distincte ale entitatilor. In procesul de modelare vor fi luate in considerare doar acele proprietati ale entitatilor care sunt semnificative pentru aplicatia respectiva.

Exemplu:

Entitatea STUDENTI poate avea ca atribute Matricol, Nume, CNP, Grupa, etc. Din acest motiv, la entitatea STUDENTI nu vom lua in considerare caracteristici cum ar fi Talia, nefiind necesara pentru baza de date a universitatii (astfel de atribute ar putea exista insa intr-o baza de date privind personalul militar).



Elementele modelului Entitate-Asociere

Atributele unei entitati sunt de doua feluri:

- **Atribute de identificare** (formand impreuna identificatorul entitatii) - reprezinta acea multime de atribute care permite identificarea unica a instantelor unei entitati;
- **Atribute de descriere** (sau **descriptori**) - sunt folosite pentru specificarea informatiilor suplimentare ale instantelor.
- In cazul entitatii STUDENTI atributul Matricol este atribut de identificare (deoarece nu pot exista doi studenti cu acelasi numar matricol intr-o facultate) pe cand celelalte atribute sunt atribute de descriere.



Elementele modelului Entitate-Asociere

■ **Asocieri**

Asocierile modeleaza interdependentele dintre clasele de obiecte reprezentate prin entitati.

De exemplu, intre entitatile STUDENTI si FACULTATI se poate figura o asociere Inscris_la care descrie apartenenta studentilor pe facultati.

- In crearea diagramei nu vor fi luate in consideratie decat interdependentele care sunt necesare aplicatiei respective, in lumea reala putand exista intre entatile diagramei si alte asocieri care nu sunt semnificative in contextul dat.



Elementele modelului Entitate-Asociere

- În construcția unei diagrame EA trebuie avute în vedere următoarele:
 - Entitatile se reprezinta prin dreptunghiuri în care este inscris numele entitatii. În cazul entitatilor dependente (slabe), conturul va fi cu linie dubla.
 - Atributele se reprezinta prin cercuri (sau ovale) în interiorul cărora apare numele atributului. Ele sunt conectate cu un segment de dreapta la entitatea de care aparțin. Pentru a distinge attributele de identificare de cele de descriere, numele primelor va fi subliniat.
 - Asocierile se reprezinta prin romburi (daca conecteaza una sau doua entitati) sau poligoane regulate (daca conecteaza mai mult de doua entitati) conectate prin segmente de dreapta la entitatile asociate, avand inscris în interior (sau alături) numele asocierii.



Extensii ale modelului Entitate-Asociere

- Modelul entitate-asociere clasic are unele lipsuri în ceea ce privește posibilitatea modelării caracteristicilor asociate unor subclase de obiecte modelate prin simple entități.
- Pentru aceasta, la modelul original au fost adăugate două noi concepte: **iерархия de generalizare** și **iерархия de incluziune**. Prima definește partitionarea instantelor unei entități în **n** subclase diferite, iar a doua permite clasarea unor dintre instantele unei entități în **k** subclase care nu reprezintă o partitie în sens matematic.
- Din punct de vedere formal, cele două concepte se pot defini astfel:



Extensii ale modelului Entitate-Asociere

■ Ierarhia de incluziune

Definitie: O entitate E_k este o submultime a entitatii E (sau este inclusa in entitatea E) daca fiecare instanta a lui E_k este de asemenea o instanta a lui E .

- Un exemplu de incluziune este definirea in cadrul entitatii ANGAJATI a unor subclase modelate prin entitatile INGINERI, ECONOMISTI si COLABORATORI.
- In cazul ierarhiei de incluziune entitatile fiu pot sa nu fie disjuncte, de exemplu, pot exista angajati ingineri dar care sunt incadrati cu contract de colaborare. De asemenea, reuniunea lor poate sa nu acopere in intregime entitatea tata, de exemplu, exista angajati care nu sunt ingineri, economisti sau colaboratori.



Extensii ale modelului Entitate-Asociere

■ Ierarhia de generalizare

Definitie: O entitate E este generalizarea entitatilor E_1, E_2, \dots, E_n daca orice instanta a lui E este de asemenea instanta in una, si numai una, dintre entitatile E_1, E_2, \dots, E_n .

- Un exemplu de generalizare este clasarea instantelor entitatii ANGAJATI in subclasele BARBATI si FEMEI.
- O caracteristica a ierarhiei de generalizare este ca din punct de vedere matematic entitatile *fiu* reprezinta o partitie a entitatii *tata*:
 - $E_1 \cup E_2 \cup \dots \cup E_n = E$ si
 - $E_i \cap E_j = \emptyset$ pentru orice $i \neq j$ din intervalul 1..n
- Ierarhiile de inclusiune si generalizare se folosesc doar in cazul in care, pentru subclasele unor clase modelate prin entitati, este nevoie de stocarea unor informatii suplimentare specifice.



Extensii ale modelului Entitate-Asociere

Element	Reprezentare	Exemplu
Ierarhie de incluziune	<pre>graph TD; E[E] --> E1[E1]; E --> E2[E2]; E --> E3[E3]</pre>	<pre>graph TD; Angajati[Angajati] --> Economisti[Economisti]; Angajati --> Ingineri[Ingineri]; Angajati --> Colaboratori[Colaboratori]</pre>
Ierarhie de generalizare	<pre>graph TD; E[E] --> Criteriu{Criteriu}; E1[E1] --> Criteriu; E2[E2] --> Criteriu</pre>	<pre>graph TD; Angajati[Angajati] --> Sex{Sex}; Barbati[Barbati] --> Sex; Femei[Femei] --> Sex</pre>

Figura 2. Convenția grafică de reprezentare grafică a ierarhiilor



Extensiile ale modelului Entitate-Asociere

Exemplu:

- În cazul unei baze de date de personal este nevoie să fie specificat numărul de copii ai fiecarui angajat. Acest fapt se poate modela în două feluri: fie prin adăugarea la entitatea ANGAJATI a unui atribut suplimentar Numar_copii (care va avea valoarea 0 pentru angajatii fără copii), sau prin crearea unei entități suplimentare COPII aflată într-o relație de incluziune cu entitatea ANGAJATI, care va avea ca atribute de identificare pe cele ale angajatului iar ca atribut descriptiv numărul de copii, acesta fiind atributul specific subclasei.
- Vom alege a doua variantă în cazul în care nu avem nevoie de informații suplimentare despre copii(nume, CNP, etc.) sau numărul angajatilor care au date specifice aceluia atribut este mult mai mic decât numărul total al angajatilor(de exemplu angajati absolvenți a două facultăți), fapt care va duce la economie de spațiu pe disc. În acest caz o nouă entitate(o nouă tabela a bazei de date) va ocupa mai puțin spațiu decât un atribut suplimentar (o coloană suplimentară) a tabelei ANGAJATI.



Caracteristici ale elementelor modelului

Așa cum entitatile au atribută care specifică diverse proprietăți ale clasei de obiecte modelate, și asociările au caracteristici care aduc informații suplimentare.

Acestea sunt următoarele:

- **Gradul asocierii**

Este o valoare numerică întreagă și este dat de numarul de entități care participă la acea asociere.

Asociările de grad 1, 2 și 3 se mai numesc și asocieri **unare**, **binare** și respectiv **ternare**.

Exemplu:

Vom considera o bază de date continând informații despre studenți, proiectele realizate de acestia, calculatoarele pe care au alocate ore de lucru și facultățile la care sunt înscrise. De asemenea vom considera că unii dintre studenți au un **tutor** care îi îndrumă, acesta fiind un student dintr-un an mai mare.



Caracteristici ale elementelor modelului

- Diagrama EA a bazei de date este prezentata in Figura 3. Pentru simplificarea figurii nu s-au reprezentat decat entitatile si asocierile dintre ele, nu si atributele fiecarei entitati in parte.

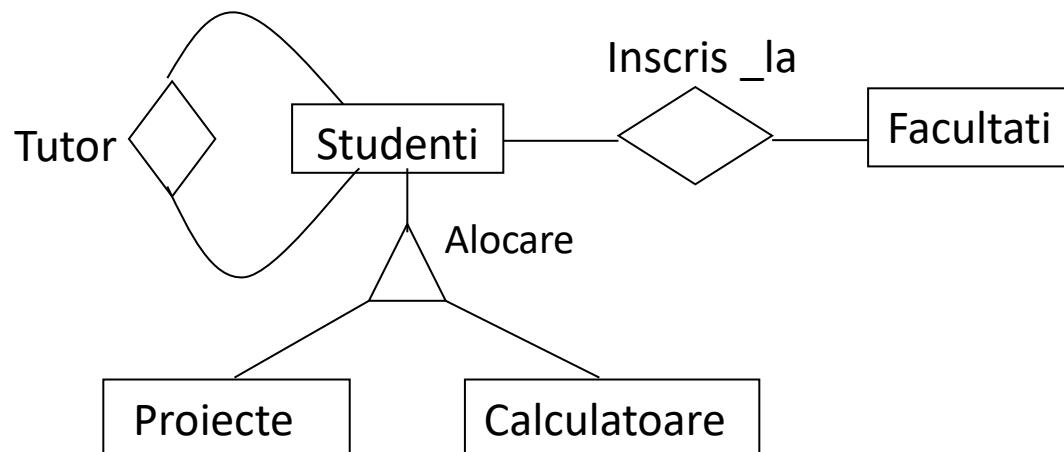


Figura 3. Exemple de asocieri de grad 1, 2 si 3



Caracteristici ale elementelor modelului

- Asocierea **Tutor** este o asociere unara deoarece la ea participa doar entitatea STUDENTI. Aceasta asociere arata cine este tutorul fiecarui student (daca exista).
- Asocierea **Inscris_la** este o asociere binara intre entitatile STUDENTI si FACULTATI si arata la ce facultate/facultati este inscris fiecare student.
- Asocierea **Alocare** este o asociere ternara intre entitatile STUDENTI, PROIECTE si CALCULATOARE. Ea modeleaza pe ce calculatoare are alocate ore de lucru fiecare student pentru fiecare proiect.



Caracteristici ale elementelor modelului

Un exemplu de asociere de grad mai mare ca trei este orarul unui an de studiu al unei facultati. Aceasta este o asociere intre urmatoarele entitati:

- GRUPE - Fiecare grupa are un cod unic.
- SALI - Salile sunt etichetate printr-un indicativ alfanumeric.
- ORE - Un interval orar este un triplet (Zi, De la ora, La ora).
- ACTIVITATE - Este o activitate prezenta in orar (curs, laborator, seminar, proiect).
- PROFESOR - Este cadrul didactic titular pentru o activitate.



Caracteristici ale elementelor modelului

- Modelarea acestor clase de obiecte ca entitati presupune faptul ca in baza de date sunt stocate si alte informatii despre ele. Diagrama EA este prezentata in Figura 4. Ea modeleaza programarea activitatilor didactice efectuate de profesori pe intervale orare, sali si grupe.

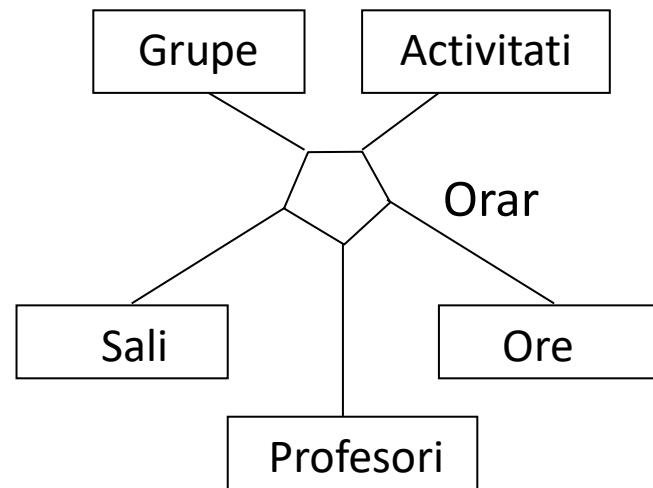


Figura 4. Asociere de grad 5



Caracteristici ale elementelor modelului

■ Conectivitatea asocierii

Conectivitatea este specifică fiecărei ramuri a unei asocieri și poate avea una dintre urmatoarele două valori: **unu** sau **multi**. Determinarea ei se face astfel:

- Se alege o entitate **E**;
- Se fixează o entitate **F** care participă la asociere;
- Se determină cu cate instante din entitatea **F** se poate conecta o instantă din **E** ;
- Dacă poate fi cel mult una, conectivitatea ramurii este **unu**, altfel conectivitatea este **multi**.
- Pentru stabilirea conectivității verificarea se face în ambele sensuri între **E** și **F** .



Caracteristici ale elementelor modelului

Convenții de reprezentare a conectivității:

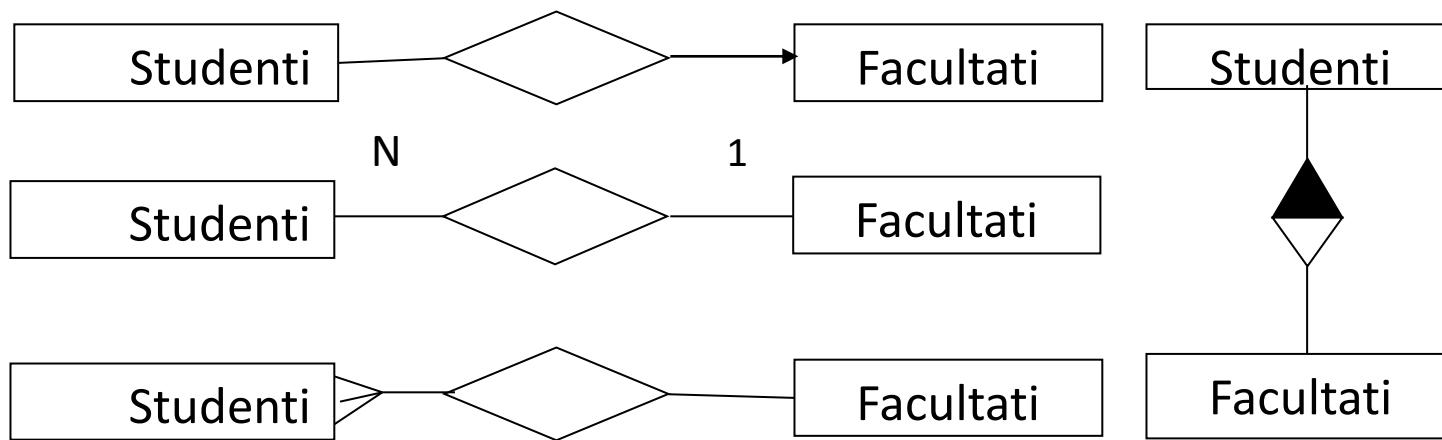


Figura 5. Exemple de reprezentare a conectivitatii



Caracteristici ale elementelor modelului

Exemplu:

Pentru determinarea conectivitatii trebuie stabilite mai intai cerintele de functionare si modelare.

Revenim la exemplul din Figura 3:

- Asocierea **Tutor** este **unu-unu** sau **multi-unu** dupa cum un student poate fi tutor pentru un singur student sau pentru mai multi studenti de an inferior.
- Asocierea **Inscris_la** este **multi-unu** (multi spre STUDENTI) sau **multi-multi** dupa cum un student poate fi inscris la una sau mai multe facultati.
- Asocierea ternara **Alocare** (aplicam definitia):



Caracteristici ale elementelor modelului

- Ramura spre STUDENTI: fiind dat un proiect si un calculator, cati studenti au ore alocate pe acel calculator pentru respectivul proiect? Considerand ca mai multi studenti lucreaza pentru acelasi proiect, pe acelasi calculator, ramura va fi **multi**;
- Ramura spre PROIECTE: fiind dat un student si un calculator, la cate proiecte are acesta alocate ore pe acel calculator? Considerand ca pentru fiecare proiect exista un calculator dedicat, ramura va fi **unu**;
- Ramura spre CALCULATOARE: fiind dat un student si un proiect, pe cate calculatoare are alocate acesta ore pentru realizarea proiectului? Considerand ca la un proiect se lucreaza pe un singur calculator, ramura va fi **unu**.
- Deci asocierea **Alocare** este **multi-unu-unu**.



Caracteristici ale elementelor modelului

- Observam ca raspunsul la fiecare dintre cele trei intrebari se da in functie de realitatea modelata. Aceeasi asociere poate avea conectivitati diferite in cazuri diferite: daca exista chiar si un singur proiect la care un student are ore alocate pe mai mult de un calculator, ramura spre CALCULATOARE va fi **multi** iar asocierea va fi **multi-unu-multi**.
- Convenția de reprezentare grafica a conectivitatii folosita in aceasta prezentare este urmatoarea: ramurile **unu** vor fi reprezentate cu sageata, iar ramurile **multi** fara segeata.
- In Figura 6 sunt prezentate cele trei asociieri avand figurata si conectivitatea. S-a presupus ca asocierea **Tutor** este **unu-unu** (un student are un singur tutor) iar **Inscris_la** este **multi-unu** (mai multi studenti sunt inscrisi la o facultate).



Caracteristici ale elementelor modelului

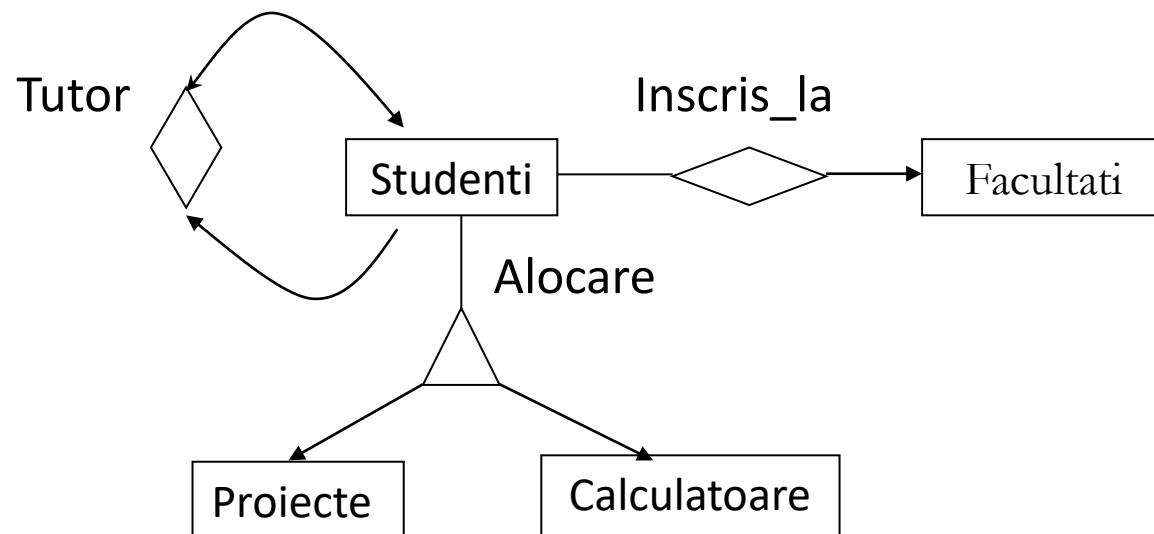


Figura 6. Reprezentarea conectivitatii



Caracteristici ale elementelor modelului

■ Obligativitatea asocierii

Ca si conectivitatea, **obligativitatea** se determina pentru fiecare ramura si poate avea doar una dintre urmatoarele valori: **obigatorie** sau **optionala**.

Determinarea ei se face astfel:

- Se alege ramura spre o entitate **E**;
- Se fixeaza o entitate **F** care participa la asociere;
- Este obligatoriu sa existe o instanta a lui **E** asociata cu o instanta din **F**?
- Daca raspunsul este DA ramura este obligatorie, altfel este optionala.



Caracteristici ale elementelor modelului

Exemplu:

In exemplul anterior ramurile asocierilor **Tutor** si **Alocare** sunt optionale iar cele ale asocierii **Inscris_la** sunt obligatorii deoarece:

- Pentru asocierea **Tutor**:
 - exista studenti care nu au un tutor si nici nu sunt tutori pentru alti studenti;
- Pentru asocierea **Alocare**:
 - un student poate sa nu aiba alocate ore pe niciun calculator la un proiect (de exemplu in cazul unui proiect la o materie umanista);
 - un student si un calculator, respectiv un calculator si un proiect, pot sa nu fie asociati prin alocare de ore de lucru (de exemplu pentru calculatoarele din birourile cadrelor didactice).
- Pentru asocierea **Inscris_la**:
 - nu exista studenti care nu sunt inscrisi la nicio facultate si nici facultati fara studenti inscrisi.



Caracteristici ale elementelor modelului

- Convenția de reprezentare grafică a clasei de apartenență folosită în continuare este urmatoarea: ramurile **obligatorii** vor fi reprezentate prin linie continuă iar cele **optionale** prin linie interrupță. În Figura 7 este reprezentată obligativitatea.

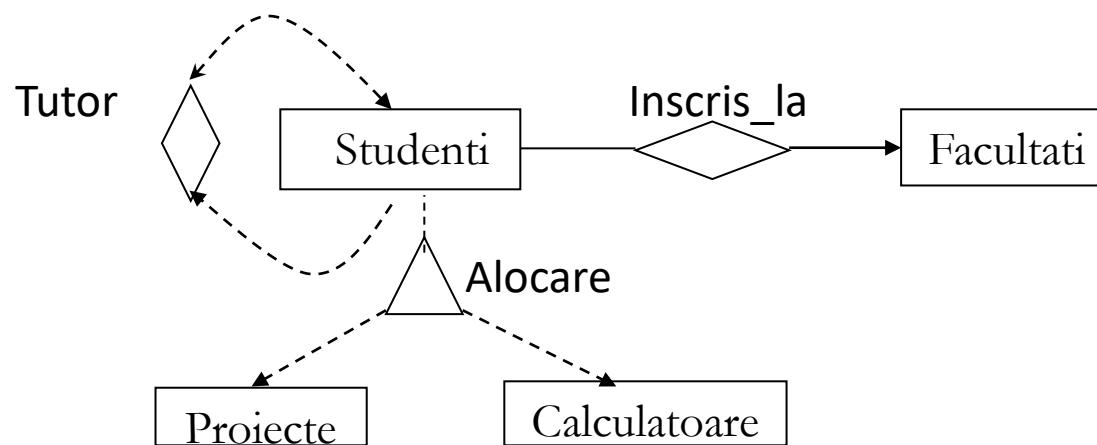


Figura 7. Reprezentarea obligativitatii



Caracteristici ale elementelor modelului

- Daca gradul si conectivitatea unei asocieri sunt folosite in proiectarea conceptuala a schemei bazei de date, obligativitatea se modeleaza pentru definirea unui criteriu de integritate specificand posibilitatea de aparitie a valorilor nule.
- La transformarea diagramei EA in model relational atributele tabelelor care modeleaza informatia reprezentata de asocieri pot avea sau nu valori nule, dupa cum ramurile acestora sunt optionale sau obligatorii.



Caracteristici ale elementelor modelului

■ Atributele asociierilor

In unele cazuri o anumita informatie descriptiva este asociata cu un ansamblu de clase diferite, nu cu o singura clasa de obiecte, modelate fiecare prin entitati. In acest caz aceasta va fi modelata ca un **atribut al asocierii** dintre entitatile respective.

Exemplu:

Sa luam cazul unei asocieri **A_absolvit** de tip **multi-multi** intre entitatile STUDENTI si FACULTATI care contine informatii privind facultatile absolvite anterior de unii studenti, asa cum este prezentata in Figura 8.



Caracteristici ale elementelor modelului

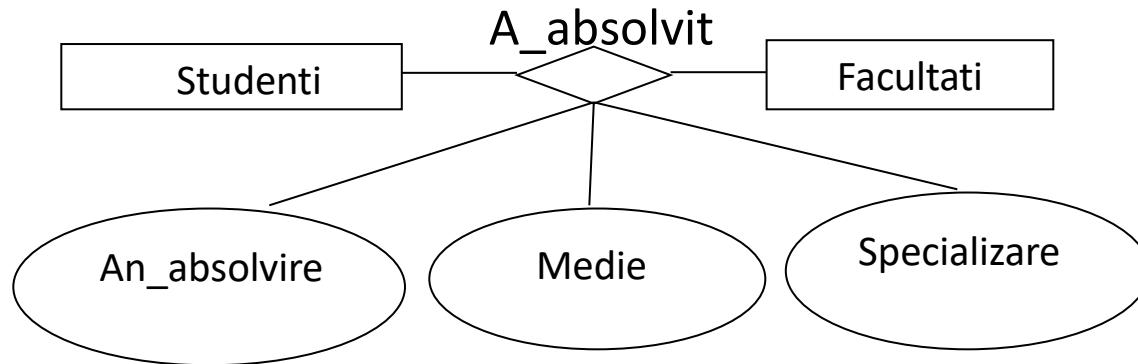


Figura 8. Reprezentarea atributelor asociierilor

- In acest caz informatii ca anul absolvirii, media, specializarea nu pot fi conectate nici la STUDENTI (pentru ca un student poate fi absolventul mai multor facultati in ani diferiti, cu medii diferite, etc.) si din motive similare nici la FACULTATI. Ele descriu asocierea unui student cu o facultate si de aceea vor fi atasate asocierii **A_absolvit**. Toate atributele unei asocieri sunt attribute descriptive, neexistand in acest caz un identificator al asocierii.



Caracteristici ale elementelor modelului

■ Rolul

In cazul in care de la o asociere pornesc mai multe ramuri catre aceeasi entitate, fiecareia dintre acestea i se poate asocia un **rol**. Acesta arata semnificatiile diferite pe care le are aceeasi entitate in cadrul asocierii respective, asa cum se vede in Figura 9.

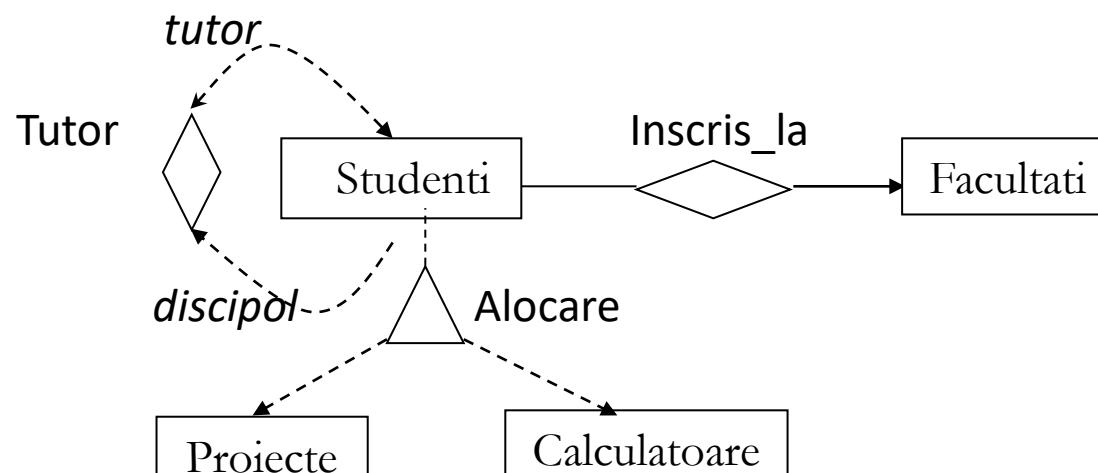


Figura 9. Reprezentarea obligativitatii si a rolurilor

- In cazul asocierii **Tutor** cele doua ramuri pot fi etichetate, de exemplu, cu **tutor** si **discipol**, aratand ca instante diferite ale aceleiasi entitati au roluri diferite.



Criterii de modelare

■ Clasificarea in entitati si attribute

Desi definitia notiunilor de entitate, atribut, asociere este destul de simpla, in practica modelarii apar dificultati in clasificarea diverselor informatii intr-una din aceste categorii. De exemplu, in cazul sediilor unei banchi localizate in diverse orase: obiectul ORASE este entitate distincta sau atribut descriptiv al entitatii SEDII ?

- Pentru a putea clasifica corect informatiile, exista cateva reguli care trebuie respectate si pe care le presentam in continuare.
- Prima regula da un criteriu general de impartire in entitati si attribute, urmatoarele doua semnaleaza exceptii iar ultimele doua reguli au un caracter mai degraba orientativ si mai putin normativ.



Criterii de modelare

- ***Regula 1.*** Entitatile au informatii descriptive, pe cand atributele nu poseda astfel de informatii.

Daca exista informatii descriptive despre o anumita clasa de obiecte, aceasta va fi modelata ca o entitate.

In cazul in care pentru o clasa de obiecte nu este nevoie decat de un identificator (codul, denumirea, etc), ea va fi modelata ca un atribut.

De exemplu, daca despre un oras este necesara stocarea in baza de date unor informatii ca judet, codul strazilor, numar locuitori, etc. atunci ORASE va fi o entitate. Daca singura informatie necesara este numele atunci Nume_oras va fi un atribut al altei entitati.



Criterii de modelare

- **Regula 2. Atributele multivalorice vor fi reclasificate ca entitati.**

Daca la o valoare a unui **identificator** corespund mai multe valori ale unui **descriptor**, atunci descriptorul va fi clasat ca entitate. De exemplu, in cazul unei baze de date privind localizarea in teritoriu a unor banchi, daca se stocheaza date doar despre banci care au un singur sediu, **Localitate** este atribut al entitatii BANCI, asa cum este prezentat in Figura 10.

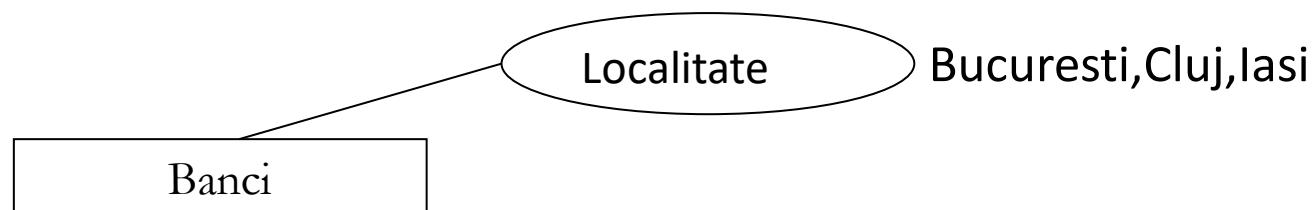


Figura 10. Reprezentarea atributelor multivalorice



Criterii de modelare

- Daca insa se stocheaza date despre banci care au sucursale si filiale in diverse localitati, deci pentru o singura banca (o valoare a identificatorului entitatii BANCI) avem mai multe localitati in care aceasta are sedii (mai multe valori ale descriptorului Localitate), atunci **LOCALITATI** va fi entitate distincta desi are numai un singur atribut. Pentru a modela localizarea sediilor in diverse localitati intre cele doua entitati va exista o asociere binara unu-multi (unu spre BANCI) numita de exemplu **Are_sediul_in**, asa cum se vede in Figura 11.

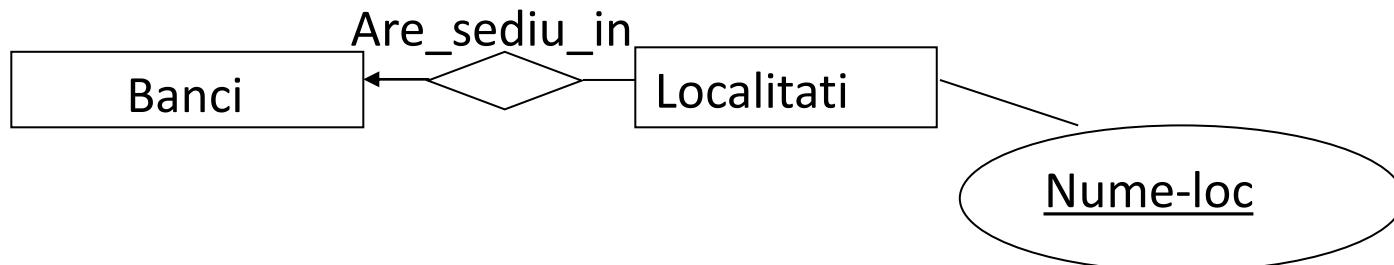


Figura 11. Reprezentarea atributelor multivalorice reclasificate ca entitati



Criterii de modelare

- **Regula 3.** Atributele unei entitati care au o asociere multi-unu cu o alta entitate vor fi reclasificate ca entitati.

Asa cum am vazut, asocierile pot lega doar entitati. Daca un descriptor al unei entitati este intr-o relatie multi-unu cu o alta entitate, acel descriptor va fi trecut in categoria entitatilor.

De exemplu, daca avem entitatile BANCI avand ca atribut descriptiv monovaloric Localitate si Judet, daca se doreste modelarea apartenentei la judete a localitatilor va exista o asociere multi-unu intre atributul Localitate si entitatea JUDETE, reprezentata in Figura 12.

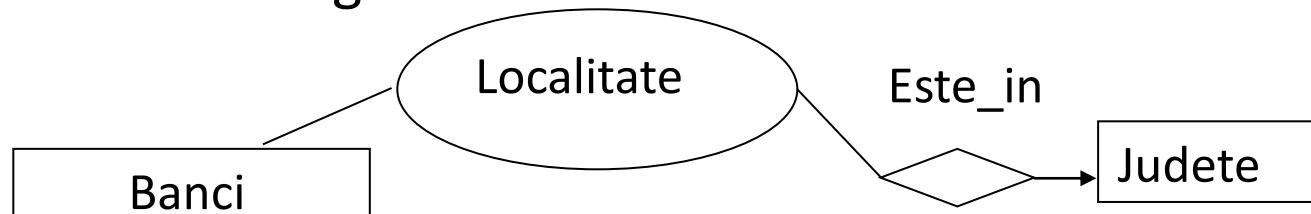


Figura 12. Reprezentarea atributelor multivalorice multi-unu



Criterii de modelare

- În acest caz atributul Localitate va fi reclasificat ca entitatea LOCALITATI desi nu sunt necesare alte informații în afara numelui localității, ca în Figura 13.

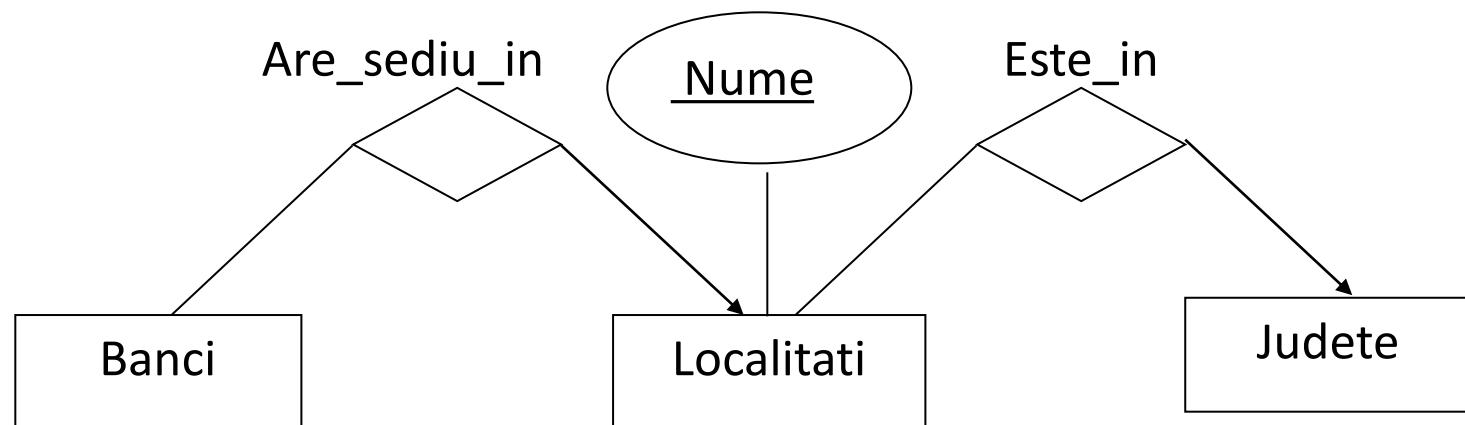


Figura 13. Reprezentarea atributelor multivalorice multi-unu reclasificate



Criterii de modelare

- ***Regula 4.*** Atributele vor fi atasate la entitatile pe care le descriu in mod nemijlocit. De exemplu, Universitate va fi atasat ca atribut al entitatii FACULTATI si nu al entitatilor STUDENTI sau PROFESORI.
- ***Regula 5.*** Folosirea identificatorilor compusi va fi evitata, pe cat posibil. Identificatorul unei entitati este acea submultime de atribute ale acesteia care identifica in mod unic fiecare instanta a sa. In modelul relational pentru atributele de acest fel se construiesc, de regula, structuri de cautare rapida (indecsi) care functioneaza cu atat mai lent cu cat complexitatea index-ului creste.
Aplicarea acestei reguli se poate face in diverse moduri:



Criterii de modelare

- Daca identificatorul unei entitati este compus din mai multe attribute care sunt toate identificatori in alte entitati, acea entitate se elimina. Informatia continuta de aceasta va fi modelata sub forma unei asocieri intre acele entitati.
- Daca identificatorul unei entitati este compus din mai multe attribute care nu sunt toate identificatori in alte entitati, exista doua solutii:
 - Entitatea respectiva se elimina si este inlocuita prin alte entitati si asocieri astfel incat per ansamblu informatia modelata in varianta initiala sa fie pastrata.
 - Entitatea respectiva ramane in forma initiala, cu dezavantaje insa in privinta vitezei tranzactiilor pe baza de date.



Criterii de modelare

- Se observa ca procedura clasificarii obiectelor in entitati si atribute este iterativa:
 - ✓ Se face o prima impartire conform primei reguli;
 - ✓ O parte dintre atributele astfel obtinute se reclasifica in entitati conform regulilor 2 si 3;
 - ✓ Se face o rafinare finala conform regulilor 4 si 5.



Criterii de modelare

- **Identificarea ierarhiilor de generalizare si incluziune**
In cazul in care despre anumite subclase ale unei clase de obiecte exista informatii specifice, clasa si subclasele (care la pasul anterior au fost catalogate ca entitati) sunt interconectate intr-o ierarhie de incluziune sau generalizare, dupa cum este cazul.
- La acest pas se face si o reatasare a atributelor pentru evitarea redundantei, astfel:
 - La entitatea *tata* vor fi atasate atributele care formeaza identificatorul si descriptorii care modeleaza informatii specifice intregii clase.
 - La entatile *fiu* vor fi atasate atributele de identificare (aceleasi ca ale tatalui) plus atributele care modeleaza informatii specifice doar acelei subclase de obiecte.



Criterii de modelare

Exemplu:

Sa consideram o ierarhie care imparte studentii unei facultati in doua subclase:

- caministi – daca locuiesc in campusul universitar;
- necaministi – daca nu locuiesc in campus.

Diagrama entitate-asociere si ierarhiile de asociere sunt prezentate in Figura 14.



Criterii de modelare

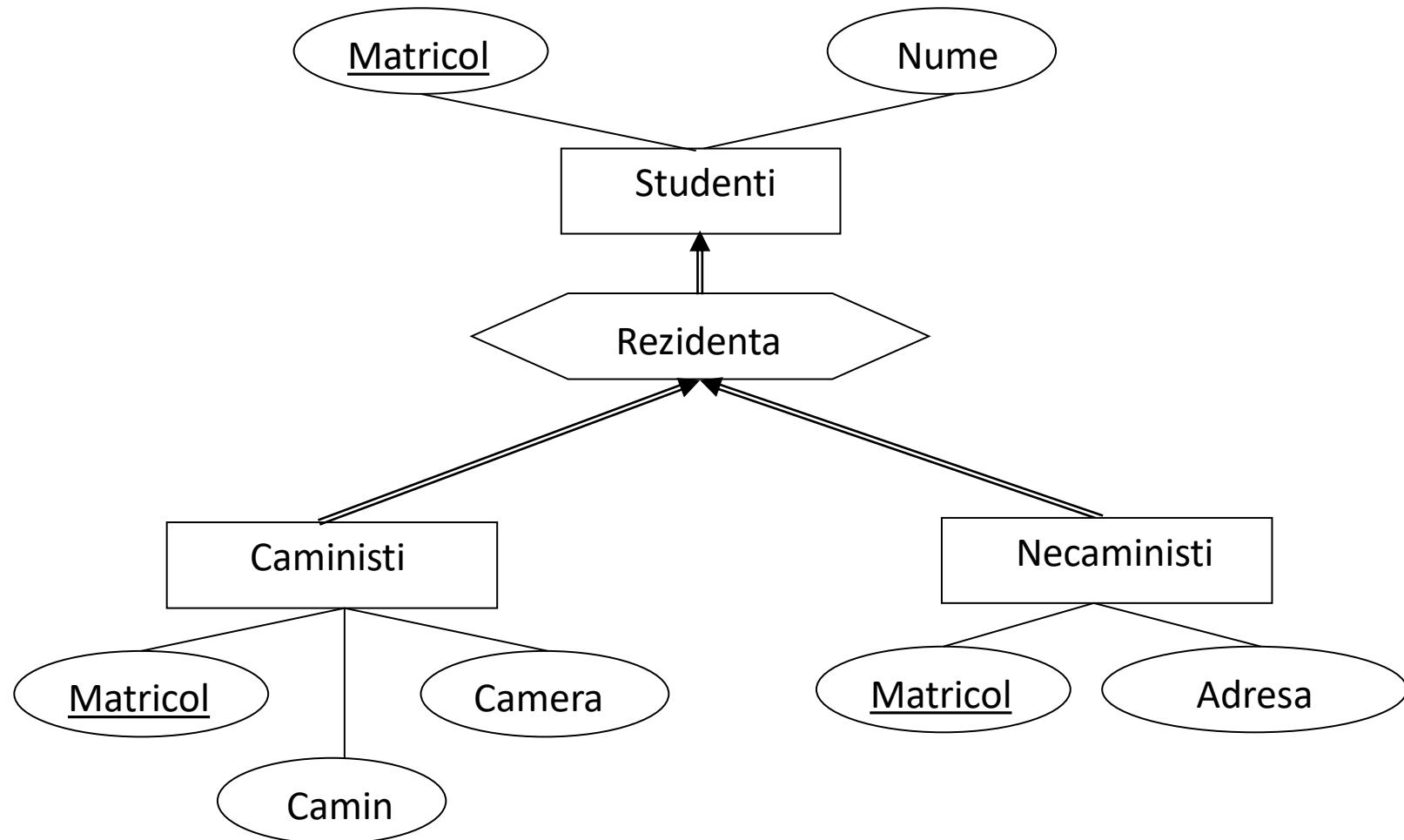


Figura 14. Atributele entitatilor unei ierarhii



Criterii de modelare

- În acest caz atributul Nume trebuie eliminat de la entitatile fiu CAMINISTI și NECAMINISTI deoarece el este prezent deja la entitatea tata STUDENTI.
- Rezulta urmatoarele reguli:
 - Identifierul tata din cadrul unei ierarhii se regaseste in identifierul tuturor fiilor ierarhiei;
 - Descriptorii care apar si la tata si la fii, se elimina de la fii;
 - Descriptorii care apar la toti fiile unei ierarhii de generalizare, dar nu apar la tata, se muta la tata.



Criterii de modelare

■ Identificarea asocierilor

In aceasta etapa se trateaza informatiile care nu au fost clasificate ca entitati sau atribute ci reprezinta interdependente intre clase de obiecte. Ele sunt modelate ca **asocieri intre entitati.** Pentru fiecare asociere se specifica gradul, conectivitatea, obligativitatea si daca este cazul si atributele asocierii precum si rolurile ramurilor sale.

- Ca si in cazul clasificarii in entitati si atribute, exista cateva reguli de urmat in operatia de definire a asocierilor:



Criterii de modelare

■ Eliminarea asocierilor redundante

In cazul in care o asociere poate fi dedusa din alte asocieri deja catalogate, aceasta se elimina. De retinut ca intre doua entitati pot sa existe oricate asocieri si ele nu sunt considerate redundante atata timp cat au semnificatie diferita.

- Un caz des intalnit de redundanta este cel al compunerii (tranzitivitatii) asocierilor. Prezentam in Figura 15 un exemplu:

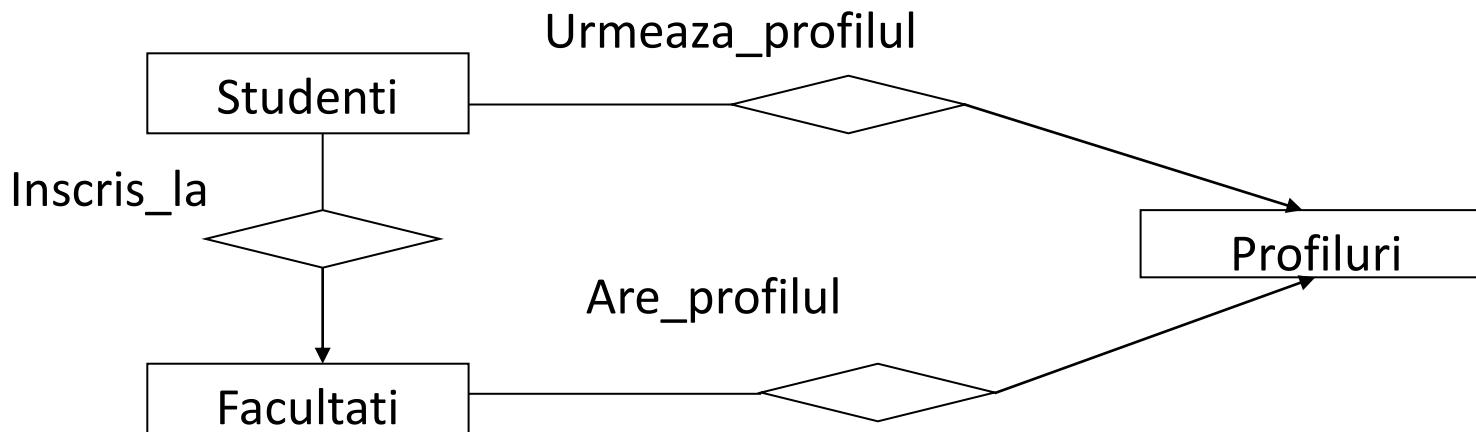


Figura 15. Asocieri redundante



Criterii de modelare

- În acest exemplu, asocierea **Inscris_la** modelează apartenența fiecarui student la o facultate a unei universități. Fiecare facultate are un profil unic descris de asocierea **Are_profilul** (de ex. Electric, Mecanic, Chimic, etc.). Ambele asocieri sunt multi-unu în sensul STUDENTI → FACULTATI → PROFILURI.
- Deoarece asocierile multi-unu (ca și cele unu-unu) sunt din punct de vedere matematic funcții, din compunerea asocierilor putem afla profilul la care este înscris fiecare student.
- Rezulta că asocierea **Urmeaza_profilul** care are chiar aceasta semnificație este redundanta si trebuie eliminata.



Criterii de modelare

■ Asocieri de grad mai mare ca doi

Asocierile ternare (sau de grad mai mare ca trei) se folosesc doar atunci cand sunt strict necesare.

- Este de multe ori posibil ca o aceeasi informatie sa fie modelata ca o asociere ternara sau ca un ansamblu de asocieri binare si unare. In cazul acesta, este de preferat ca sa se opteze pentru a doua varianta, asa cum se vede in Figura 16 si Figura 17.
- Doar cand asocierile binare nu pot modela intreaga semnificatie dorita se va opta pentru asocieri de grad mai mare decat doi. Aceasta cerinta deriva din faptul ca la trecerea in modelul relational asocierile de grad superior devin scheme de relatii de sine statatoare, marind numarul de tabele din baza de date pe cand cele de grad unu si doi (cu exceptia celor multi-multi) nu au acest efect.



Criterii de modelare

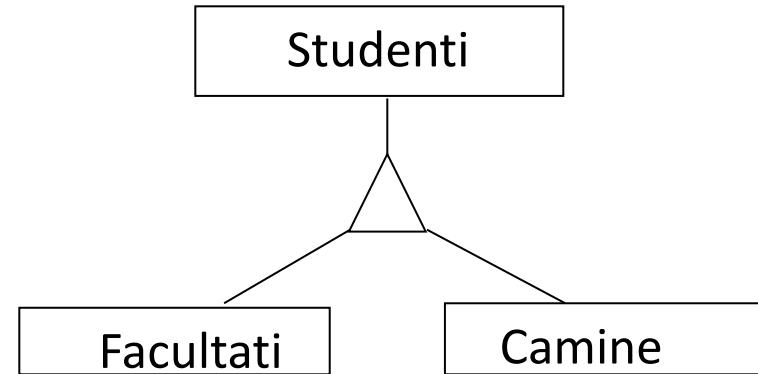


Figura 16. Exemplu de asociere ternara

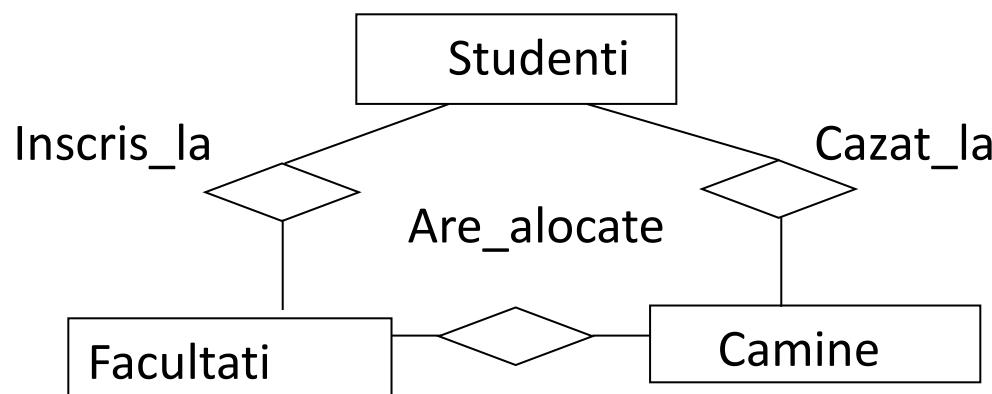


Figura 17. Exemplu de asociere ternara remodelata



Criterii de modelare

■ Integrarea vederilor

In cazul proiectarii bazelor de date complexe, activitatea se desfasoara uneori de catre mai multe colective simultan, fiecare modeland o portiune distincta a bazei de date.

Deoarece in final trebuie sa se obtina o singura diagrama a bazei de date, dupa terminarea modelarii pe portiuni diagramele rezultate sunt integrate, eliminandu-se redundantele si inconsistentele.



Transformarea diagramelor Entitate-Asociere in Modelul Relational

- In **procesul de transformare** vom pleca de la o diagrama E-A si vom obtine trei tipuri de scheme de relatie in M-R:
 - **Relatii provenite din entitati.** Ele contin aceleasi informatii ca si entitatile din care au rezultat.
 - **Relatii provenite din entitati si care contin chei straine.** Ele contin pe langa informatiile provenite din entitatile din care au rezultat si atributte care in alte entitati sunt identificatori. Este cazul acelor entitati care au asocieri multi-unu si partial din cele care au asocieri unu-unu cu alte entitati.
 - **Relatii provenite din asociieri.** Este cazul celor care apar din transformarea asociierilor binare multi-multi si a asociierilor de grad mai mare ca doi. Ele contin ca atributte reunirea identificatorilor entitatilor asociate si atributtele proprii ale asocierilor.

Procesul de transformare are un algoritm foarte precis si este din acest motiv etapa care se preteaza cel mai bine pentru crearea de instrumente software CASE care sa-l asiste.



Transformarea diagramelor E-A in M-R

■ Transformarea entitatilor

Fiecare entitate a diagramei se transforma intr-o schema de relatie avand:

- **Numele relatiei** = Numele entitatii
- **Atributele relatiei** = Atributele entitatii
- **Cheia relatiei** = Identifierul entitatii

Exemplu:

Fie entitatea ANGAJATI din Figura 18:

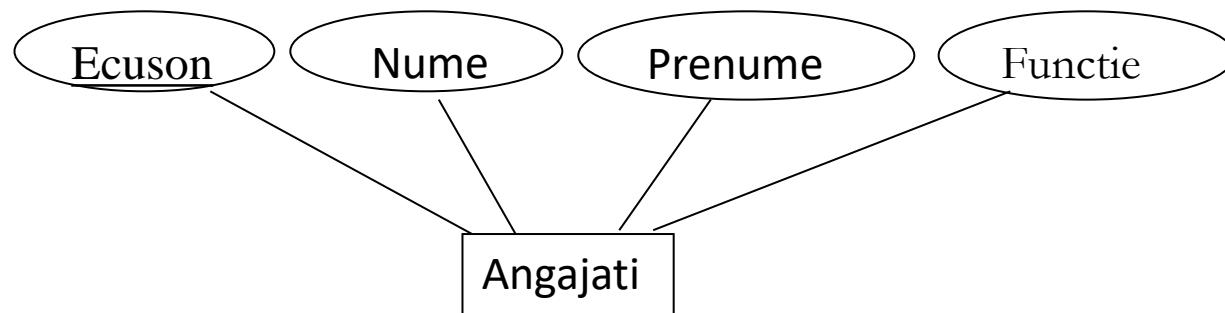


Figura 18. Exemplu de schema de relatie

Schema rezultata este ANGAJATI (Ecuson, Nume, Prenume, Functie).



Transformarea diagramelor E-A in M-R

- **Transformarea asocierilor unare si binare unu-unu si multi-unu**
Fiecare asociere din aceasta categorie va avea ca rezultat imbogatirea multimii de atribute descriptive ale uneia dintre cele doua scheme rezultate din entitatile asociate, cu cheia celeilalte scheme. Aceste atribute care se adauga sunt denumite in prezentarea de fata cheie straina (sau cheie externa) deoarece ele sunt cheie dar in alta schema de relatie.
• In cazul asocierilor **multi-unu**, se adauga identificatorul entitatii unu in schema rezultata din entitatea multi.
• In cazul asocierilor **unu-unu**, se adauga identificatorul unei entitati in schema rezultata din transformarea celeilalte. Alegerea schemei in care se face adaugarea se poate face dupa doua criterii:
 - fie in acea schema care defineste relatia cu cele mai putine tupluri dintre cele doua;
 - fie pastrandu-se, daca exista, filiatia naturala intre cele doua entitati: identificatorul tatalui se adauga la fiu.



Transformarea diagramelor E-A in M-R

- In cazul acestui tip de asociere, la schema de relatie care primeste cheia strina se ataseaza una sau doua dependente functionale primare, conform Tabelului 1.

Un exemplu este prezentat in Figura 19.

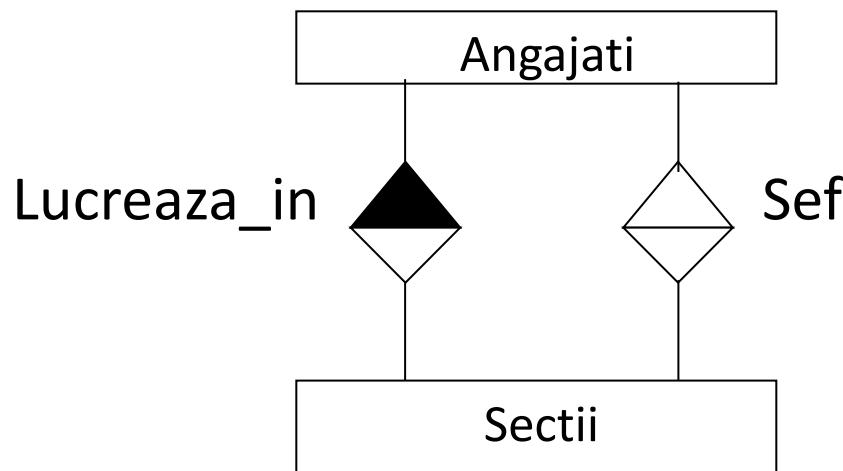
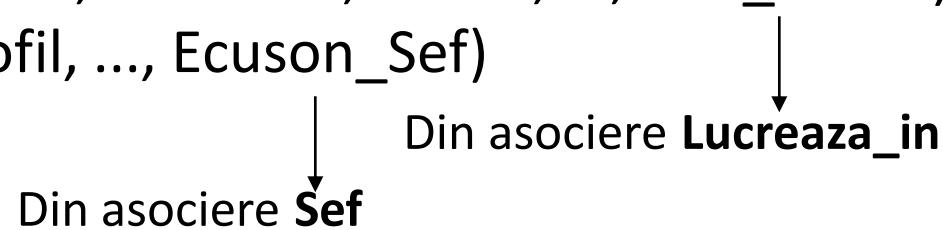


Figura 19. Exemplu de asociere cu doua dependente



Transformarea diagramelor E-A in M-R

- Intre entitatile ANGAJATI si SECTII exista doua asocieri:
 - **Sef** (unu-unu), care modeleaza faptul ca o sectie este condusa de un angajat (seful de sectie);
 - **Lucreaza_in** (multi-unu) care modeleaza faptul ca o sectie are mai multi angajati.
- Rezultatul transformarii este cel de mai jos. Atributele aflate dupa punctele se suspensie sunt cele adaugate (chei straine):
ANGAJATI (Ecuson, Nume, Prenume, Varsta, ..., Cod_Sectie)
SECTII (Cod_Sectie, Profil, ..., Ecuson_Sef)



In atributul Cod_Sectie din relatia ANGAJATI se va inregistra pentru fiecare angajat codul sectiei in care acesta lucreaza iar in atributul Ecuson_Sef din relatia SECTII se va inregistra, pentru fiecare sectie, ecusonul sefului de sectie. Pentru asocierea Sef s-a aplicat primul criteriu (relatia SECTII va avea mult mai putine inregistrari decat ANGAJATI), dar si al doilea criteriu este indeplinit.



Transformarea diagramelor E-A in M-R

- **Transformarea asocierilor unare si binare multi-multi si a celor de grad mai mare decat doi**

Fiecare asociere binara multi-multi si fiecare asociere cu grad mai mare decat doi se transforma intr-o schema de relatie astfel:

- **Nume relatie** = Nume asociere;
- **Atributele relatiei** = Reuniunea identificatorilor entitatilor asociate la care se adauga atributele proprii ale asocierii ;
- **Cheia relatiei** = Conform Tabelului 2.



Transformarea diagramelor E-A in M-R

Grad	Conecțivitate	Dependente Funcționale Primare (DFP)
Unare	<ul style="list-style-type: none"> • unu (E) - unu (E) • cheie(E) se introduce ca si cheie straina in tabela E, cu redenumire 	Cheie(E) => Cheie straina(E) Cheie straina(E) => Cheie(E)
	unu (E) - multi (E)	Cheie(E) => Cheie straina(E)
	multi (E) - multi (E)	Cheie(E) => AP
Binare	<ul style="list-style-type: none"> • unu (E1) - unu (E2) • cheie(E2) se introduce in E1 	Cheie(E1) => Cheie straina(E2) Cheie(E2) => Cheie straina(E1)
	unu (E1) - multi (E2)	Cheie(E1) => Cheie straina(E2)
	multi (E1) - multi (E2)	Cheie(E1) + Cheie(E2) => AP
Ternare	unu (E1) - unu (E2) - unu (E3)	Cheie(E1)+Cheie(E2)=>Cheie(E3) sau Cheie(E1)+Cheie(E3)=>Cheie(E2) sau Cheie(E2)+Cheie(E3)=>Cheie(E1)
	unu (E1) - unu (E2) - multi (E3)	Cheie(E1)+Cheie(E3)=>Cheie(E2) sau Cheie(E2)+Cheie(E3)=>Cheie(E1)
	unu (E1)- multi (E2) - multi (E3)	Cheie(E2)+Cheie(E3)=>Cheie(E1)
	multi (E1) - multi (E2) - multi (E3)	Cheie(E1)+Cheie(E2)+Cheie(E3)=>AP

Tabel 1. Dependente functionale primare rezultate din transformare

Legenda: X => Y: Multimea de coloane X determina functional multimea de coloane Y

AP : Atribute proprii transformarii sau multimea vida (daca nu exista)

X+Y : Coloanele X impreuna cu coloanele Y



Transformarea diagramelor E-A in M-R

Grad	Conecțivitate	Cheia relației obținuta din asociere
Unare	multi (E) - multi (E)	Cheie(E) + Cheie(E)
Binare	multi (E1) - multi (E2)	Cheie(E1)+Cheie(E2)
Ternare	unu (E1) - unu (E2) - unu (E3)	Cheie(E1)+Cheie(E2) sau Cheie(E1)+Cheie(E3) sau Cheie(E2)+Cheie(E3)
	unu (E1) - unu (E2) - multi (E3)	Cheie(E1)+Cheie(E3) sau Cheie(E2)+Cheie(E3)
	unu (E1)- multi (E2) - multi (E3)	Cheie(E2)+Cheie(E3)
	multi (E1)-multi(E2) - multi (E3)	Cheie(E1)+Cheie(E2)+Cheie(E3)

Tabel 2. Cheile schemelor de relație rezultate din asocieri

Legenda:

X + Y: Multimea de atribute X impreuna cu multimea de atribute Y



Transformarea diagramelor E-A in M-R

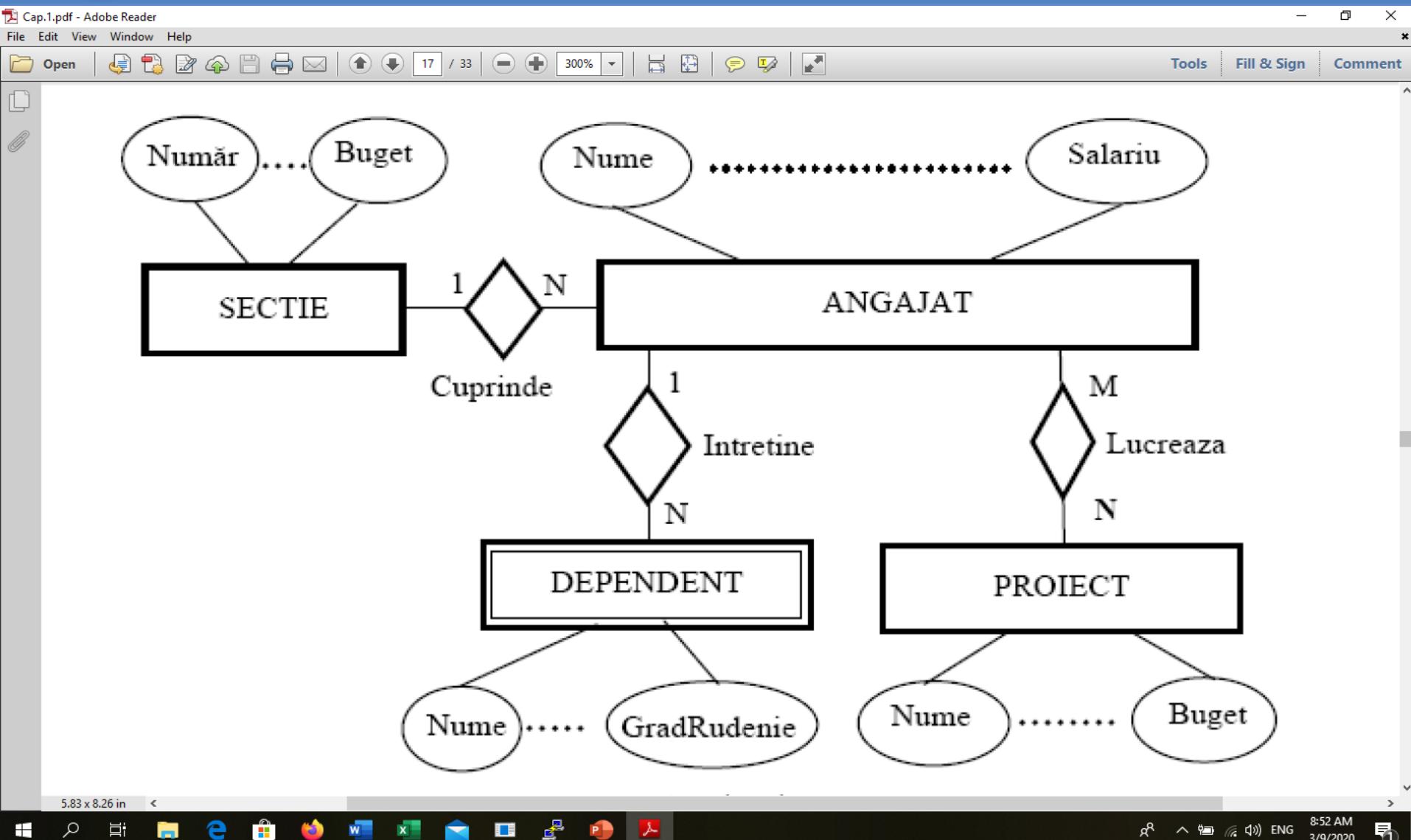


Figura 20. Exemplu de diagrama de relatii in modelul E-A



Transformarea diagramelor E-A in M-R

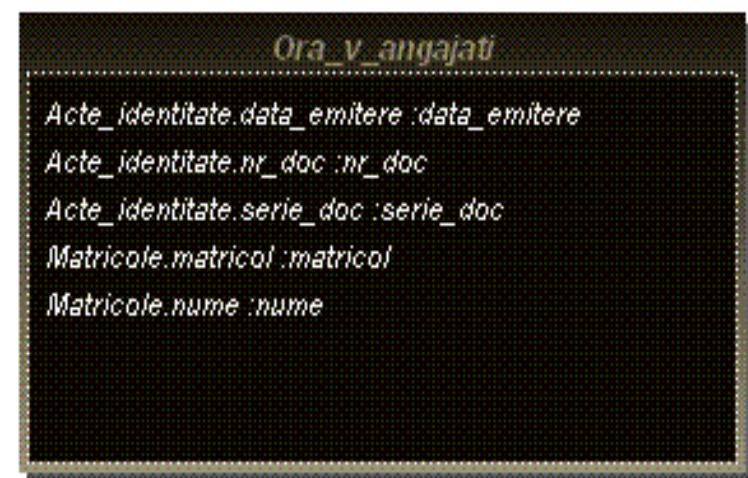
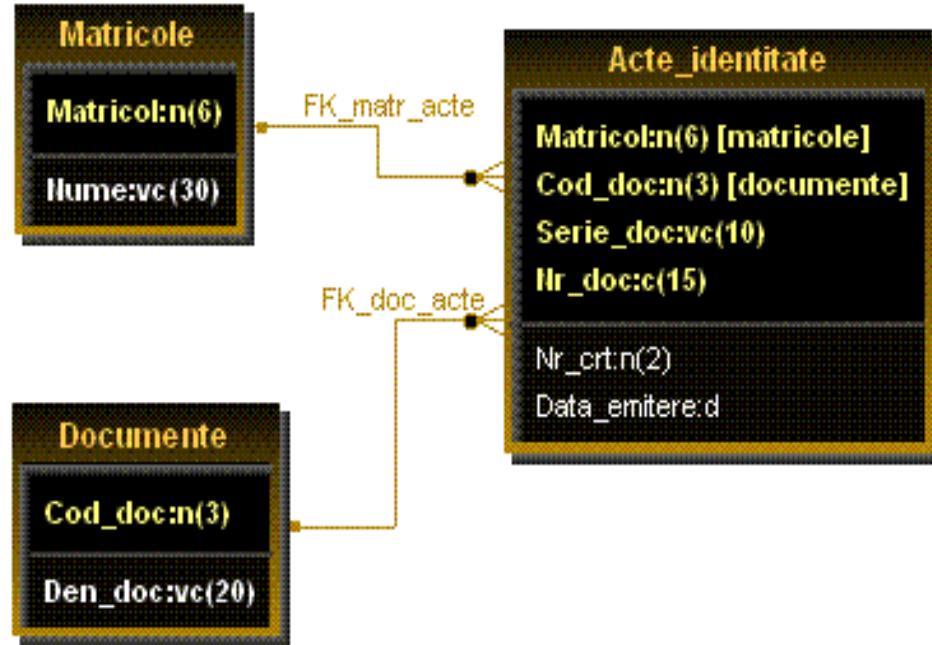


Figura 20. Exemplu de diagrama de relatii in modelul M-R



Capitolul 5

Proiectarea bazelor de date



Proiectarea bazelor de date

Proiectarea corecta a bazei de date este primul pas spre o implementare de succes a unei aplicatii software si de aceea trebuie respectate metodologiile care impun normalizarea schemelor de relatie, pastrarea integritatii datelor si eliminarea anomalilor.

Proiectarea trebuie vazuta sub doua aspecte:

- proiectare conceptuala;
 - proiectare logica.
- Deoarece o schema de relatie poate avea multiple forme, se pune problema alegerii corecte a structurii relatiei printr-o proiectare conceptuala adecvata, dar si respectarea cerintelor de functionare rezultate din analiza si regasite in proiectarea logica.
 - Schema conceptuala descrie in mod abstract forma in care dorim sa stocam datele reale iar procesul de realizare se numeste **modelare conceptuala**.



Dependente functionale

In paragrafele urmatoare vom folosi o conventie de notare intalnita in literatura de specialitate:

- R, S, T, ...: scheme de relatii;
- r, s, ...: instante ale relatiilor R respectiv S;
- A, B, C, D, ... (litere mari de la inceputul alfabetului): atribute ale unei relatii;
- X, Y, Z, W, U, ... (litere mari de la sfarsitul alfabetului): multimi de atribute dintr-o schema de relatie;
- $X \subseteq R$: Multimea de atribute X este inclusa in multimea atributelor relatiei R;
- $Y \subseteq X$: Multimea de atribute Y este inclusa in multimea de atribute X;
- $A \in X$: Atributul A apartine multimii de atribute X;
- t, t1, t2, ... tupluri ale unei relatii;
- $t[X]$: valorile atributelor din multimea X aflate in tuplul t;
- F, G, ...: multimi de dependente functionale atasate unei scheme de relatie.



Dependente functionale

- Termenul **relatie** semnifica atat **schema relatiei** (descrierea structurii acesteia) cat si o **instanta** a acesteia (datele propriu-zise).
- **Dependentă datelor** este o restrictie asupra relatiilor R care pot constitui valoarea curenta a unei scheme de relatie R .
- Dependentă trebuie vazuta ca o legatura intre două atribute, in sensul ca valoarea unui atribut determina valoarea celuilalt, de exemplu atributul Matricol determina unic atributul Nume al unui student.



Dependente functionale

- **Definitie:** Fie:
 - R o schema de relatie si
 - $X, Y \subseteq R$ doua multimi de atribute ale acesteia.
 - Spunem ca **X determina functional pe Y** (sau Y este determinata functional de X), prin notatia simbolica $X \rightarrow Y$ daca si numai daca oricare ar fi doua tupluri t_1 si t_2 din orice instanta a lui R atunci:
$$t_1[X] = t_2[X] \Rightarrow t_1[Y] = t_2[Y]$$
Cu alte cuvinte, daca doua tupluri au aceleasi valori pe atributele X atunci ele au aceleasi valori si pe atributele Y .



Dependente functionale

Exemple:

- Consideram relatia FURNIZORI cu urmatoarea structura:
FURNIZORI (ID_furniz, Den_furniz, Adresa) unde:
 - ID_furniz = cheia de identificare unica a unui furnizor;
 - Den_furniz = denumirea furnizorului;
 - Adresa = adresa furnizorului.
- Relatia COMPOENTE este un catalog de componente pentru calculatoare, cu urmatoarea structura:
COMPONENTE(ID_com, Den_com, ID_furniz, Pret, Um) unde:
 - ID_com = cheia de identificare unica a unei componente;
 - Den_com = denumirea componentei;
 - ID_furniz = cheia de identificare unica a unui furnizor pentru o componentă;
 - Pret = pretul unitar al componentei ;
 - Um = unitatea de masura (bucati, metri, set, etc.).



Dependente functionale

- Putem identifica urmatoarele dependente functionale:
- În relația FURNIZORI :
 - $ID_{furniz} \rightarrow Den_{furniz}$
 - $ID_{furniz} \rightarrow Adresa$
- În relația COMPONENTE :
 - $ID_{com} \rightarrow Den_{com}$
 - $ID_{com} \rightarrow ID_{furniz}$
 - $ID_{com} \rightarrow Pret, Um$
 - $ID_{com} \rightarrow ID_{furniz}, Pret$
- Observație: Singura cale de a gasi dependențele functionale valabile pentru o schema R este analiza atenta a inteleseului (semnificatiei) fiecarui atribut al acesteia si a modului in care sunt asignate valori atributelor.



Dependente functionale

- Dependentă funcțională $ID_{com} \rightarrow ID_{furniz}$, Pret specifică ca ID_{com} identifică unic un singur produs, achiziționat de la un singur furnizor și la un anumit pret. În acest caz, dacă un produs cu același nume este achiziționat de la un alt furnizor, eventual la același pret, trebuie să i se dea un alt ID_{com} .
- Dependentă funcțională $ID_{furniz} \rightarrow Den_{furniz}$ specifică ca un ID_{furniz} identifică unic un furnizor de componente și dacă două componente au același ID_{furniz} înseamnă că ele sunt achiziționate de la același furnizor.
- Observație: Dependentele funktionale nu se determină prin inspectarea valorilor relaiei ci din semnificatia atributelor acesteia.



Axiome si reguli

- Pornind de la o multime de dependente functionale atasate unei scheme de relatie se pot deduce alte dependente functionale valide, folosind reguli de inferenta.
- Exista mai multe reguli de inferenta. Pentru a se putea face o prezentare formală a acestora, trei dintre ele au fost alese ca axiome iar restul se pot deduce pornind de la ele.
- Cele trei axiome (numite Axiomele lui Armstrong) sunt urmatoarele:



Axiome si reguli

- **A1. Reflexivitate:** Fie R o schema de relatie si $X \subseteq R$.

Daca $Y \subseteq X$ atunci $X \rightarrow Y$.

Toate dependentele functionale care rezulta din aceasta axioma sunt numite si **dependente triviale**.
Ele nu spun nimic in plus fata de setul de dependente initial dar sunt dependente functionale valide.

- **A2. Augmentare:** Fie R o schema de relatie si

$X, Y, Z \subseteq R$. Daca $X \rightarrow Y$ atunci si $XZ \rightarrow YZ$.

Aceasta axioma arata ca se poate reuni o aceeasi multime Z in stanga si in dreapta unei dependente functionale valide, obtinand de asemenea o dependenta functionala valida.



Axiome si reguli

- **A3. Tranzitivitate:** Fie R o schema de relatie si $X, Y, Z \subseteq R$. Daca $X \rightarrow Y$ si $Y \rightarrow Z$ atunci si $X \rightarrow Z$.
- Pe baza acestor axiome se pot demonstra o serie de **reguli de inferenta** pentru dependente functionale dintre care cele mai importante sunt urmatoarele:
- **R1. Descompunere:** Fie R o schema de relatie si $X, Y, Z \subseteq R$. Daca $X \rightarrow Y$ si $Z \subseteq Y$ atunci si $X \rightarrow Z$.

Demonstratie:

$X \rightarrow Y$ este data. Prin ipoteza $Z \subseteq Y$, aplicand A1 rezulta $Y \rightarrow Z$.

Cu A3, din $X \rightarrow Y$ si $Y \rightarrow Z$ obtinem $X \rightarrow Z$.



Axiome si reguli

- Regula descompunerii ne permite sa rescriem un set de dependente functionale astfel incat sa obtinem doar dependente care au in partea dreapta doar un singur atribut. Sa presupunem ca avem o dependenta functionala de forma:

$$X \rightarrow A_1 A_2 A_3 \dots A_n$$

- Atunci ea poate fi inlocuita cu urmatoarele **n** dependente functionale:

$$X \rightarrow A_1$$

$$X \rightarrow A_2$$

$$X \rightarrow A_3$$

...

$$X \rightarrow A_n$$



Axiome si reguli

- **R2. Reuniune:** Fie R o schema de relatie si $X, Y, Z \subseteq R$.

Daca $X \rightarrow Y$ si $X \rightarrow Z$ atunci si $X \rightarrow YZ$.

Rezulta si faptul ca din cele n dependente obtinute prin descompunere se poate obtine dependenta initiala, deci inlocuirea acesteia nu duce la pierderea vreunei corelatii existente.

Demonstratie:

$X \rightarrow Y$ este data. Amplificam cu X si prin inferenta obtinem $XX \rightarrow XY$, sau $X \rightarrow XY$. De asemenea, $X \rightarrow Z$ este data; amplificam cu Y si $XY \rightarrow ZY$ sau $XY \rightarrow YZ$.

Aplicand A3 rezulta $X \rightarrow YZ$.



Axiome si reguli

- **R3. Pseudotranzitivitate:** Fie R o schema de relatie si $X, Y, Z, W \subseteq R$.

Daca $X \rightarrow Y$ si $YZ \rightarrow W$ atunci si $XZ \rightarrow W$.

Demonstratie:

$X \rightarrow Y$ este data. Amplificam cu Z si obtinem $XZ \rightarrow YZ$.

Dar $YZ \rightarrow W$ si aplicand A3 rezulta $XZ \rightarrow W$.



Inciderea unei multimi de DF

- Pornind de la un set de dependente functionale F si utilizand axiome si reguli obtinem o multitudine de alte dependente, triviale sau nu. Multimea tuturor dependentelor functionale care se pot deduce din F se numeste **inchiderea multimii de dependente functionale F** , notata cu F^+ . Definitia formală a acestei inchideri este urmatoarea:

$$F^+ = \{X \rightarrow Y \mid F \Rightarrow X \rightarrow Y\}$$

Unde prin \Rightarrow am notat faptul ca dependenta respectiva se poate deduce din F folosind axiome si reguli.



Inchiderea unei multimi de DF

- Multimea F^+ contine foarte multe dependente, inclusiv dependente triviale ca:
 $ABC \rightarrow A, ABC \rightarrow B, ABC \rightarrow C, ABC \rightarrow AB, ABC \rightarrow AC,$
 $ABC \rightarrow BC$ sau $ABC \rightarrow ABC$
- F^+ nu se calculeaza in totalitate, algoritmii care au nevoie de ea ocolesc, intr-un fel sau altul, calculul acesteia.
- Introducerea acestei notiuni s-a facut pentru a explica, in cazul descompunerii unei scheme de relatie, care sunt dependentele mostenite de elementele descompunerii de la relatia initiala si pentru a putea defini formal urmatoarele notiuni:



Acoperirea unei multimi de DF

- ✓ **Acoperirea unei multimi de DF** - Fie R o schema de relatie si F, G doua multimi de dependente pentru R. Se spune ca F **acopera** pe G daca si numai daca $G \subseteq F^+$.
- ✓ **Echivalenta a doua multimi de dependente** - Fie R o schema de relatie si F, G doua multimi de dependente pentru R. Se spune ca F este **echivalenta** cu G daca si numai daca F acopera pe G si G acopera pe F, adica, daca $G \subseteq F^+$ si $F \subseteq G^+$, rezulta $F^+ = G^+$.



Forma canonica a unei multimi de DF

- ✓ **Forma canonica a unei multimi de DF** - Din definitiile de mai sus rezulta ca o multime de dependente poate fi inlocuita cu alta echivalenta continand alte dependente. In cazul aceasta se poate spune ca:
 - **Definitie:** O multime de dependente este in **forma canonica** daca:
 - Orice dependenta are in partea dreapta un singur atribut. Acest lucru se poate obtine aplicand regula descompunerii prezentata anterior.
 - Multimea de dependente este minima, niciuna dintre dependente neputand sa fie dedusa din celelalte (altfel spus, nu exista dependente redundante).



Forma canonica a unei multimi de DF

Exemplu:

1) Fie $R = ABCDE$ o schema de relatie si F multimea de dependente functionale asociata, cu

$$F = \{ AB \rightarrow CDE, C \rightarrow DE \}:$$

Aplicam regula de descompunere si obtinem:

$$F = \{ AB \rightarrow C, AB \rightarrow D, AB \rightarrow E, C \rightarrow D, C \rightarrow E \}$$

Multimea nu este insa minimala deoarece $AB \rightarrow D$ si $AB \rightarrow E$ se pot deduce prin tranzitivitate din $AB \rightarrow C$ impreuna cu $C \rightarrow D, C \rightarrow E$.

Rezulta ca forma canonica a lui F este:

$$F = \{ AB \rightarrow C, C \rightarrow D, C \rightarrow E \}$$



Forma canonica a unei multimi de DF

2) Pentru relatia

COMPONENTE = (ID_com, Den_com, Pret, ID_furniz,
Den_furniz).

- Multimea de dependente functionale F este:

$$F = \{ ID_{com} \rightarrow Den_{com}, Pret, ID_{furniz}, Den_{furniz},$$
$$ID_{furniz} \rightarrow Den_{furniz} \}.$$

- Forma canonica a lui F este:

$$F = \{ ID_{com} \rightarrow Den_{com},$$
$$ID_{com} \rightarrow Pret,$$
$$ID_{com} \rightarrow ID_{furniz},$$
$$ID_{furniz} \rightarrow Den_{furniz} \}$$

A fost eliminata dependenta redundanta $ID_{com} \rightarrow Den_{furniz}$.



Implicatii logice ale dependentelor

➤ Implicatii logice ale dependentelor

- Fie F o multime de dependente functionale pentru R si fie $X \rightarrow Y$ o dependenta functionala, valabila tot pentru schema R . Atunci,
 F implica logic $X \rightarrow Y$, daca orice relatie r pentru R , care satisface dependentele din F , satisface si $X \rightarrow Y$.

Exemplu:

- Fie R o schema de relatie, iar A, B, C atribute in R . Avem, de exemplu, dependentele $A \rightarrow B$ si $B \rightarrow C$ valabile in R . Se poate arata ca, de asemenea, ca $A \rightarrow C$ este valabila in R (tranzitivitate).



Inciderea unei multimi de DF

- **Inciderea** multimii de dependente F , notata cu F^+ , se mai defineste ca **multimea dependentelor functionale implicate logic de catre F** .
Daca $F^+ = F$, F este o familie completa de dependente.
Exemplu:
Fie $R=ABC$ si F multimea de dependente. Atunci, F^+ contine toate dependentele $X \rightarrow Y$, astfel incat:
 - X contine pe A , de exemplu $ABC \rightarrow AB$, $AB \rightarrow BC$ sau $A \rightarrow C$.
 - X contine B dar nu A , iar Y nu contine A , de exemplu, $BC \rightarrow B$, $B \rightarrow C$, $B \rightarrow \emptyset$.
 - $X \rightarrow Y$ este una dintre cele doua dependente $C \rightarrow C$ sau $C \rightarrow \emptyset$.



Cheia si supercheia unei relatii

In acest moment putem sa dam o definitie echivalenta a cheii unei relatii pe baza dependentelor functionale:

- **Definitie:** Fie R o schema de relatie, F multimea de dependente functionale asociata si $X \subseteq R$. Atunci X este **cheie** pentru R daca si numai daca:
 - $F \Rightarrow X \rightarrow R$ (deci $X \rightarrow R$ se poate deduce din F);
 - X este minimala: oricare ar fi $Y \subset X$, $Y \neq X$ atunci $\neg(F \Rightarrow Y \rightarrow R)$ (deci orice submultime stricta a lui X nu mai indeplineste conditia anterioara).



Cheia si supercheia unei relatii

- Deci o cheie determina functional toate atributele relatiei, este minimala si nicio submultime stricta a sa nu determina functional pe R. Se observa faptul ca aceasta definitie este echivalenta cu cea din capitolul anterior: cunoscandu-se valorile pe atributele X, pot fi determinate unic valorile pentru toate atributele relatiei, deci este determinat unic un tuplu din relatie.
- In cazul in care doar prima conditie este indeplinita (fara minimalitate) multimea X se numeste **supercheie**.
- Observatie: Faptul ca o supercheie nu este constransa de minimalitate nu inseamna insa ca ea nu poate fi minimala. Rezulta ca orice cheie este in acelasi timp si supercheie, reciproca nefiind insa adevarata.



Cheia si supercheia unei relatii

Exemplu:

Fie $R = ABCDE$ si $F = \{ AB \rightarrow C, C \rightarrow D, C \rightarrow E \}$. Atunci AB este cheie pentru R:

- Din $AB \rightarrow C$, $C \rightarrow D$ si $C \rightarrow E$ obtinem prin tranzitivitate $AB \rightarrow D$ si $AB \rightarrow E$;
- Din $AB \rightarrow C$, $AB \rightarrow D$ si $AB \rightarrow E$ obtinem prin reuniune $AB \rightarrow CDE$;
- Din $AB \rightarrow CDE$ obtinem prin augmentare cu AB $AB \rightarrow ABCDE$, deci $AB \rightarrow R$;
- Rezulta ca AB este supercheie pentru R . Vom arata intr-un alt capitol cum se poate demonstra si ca AB este minimala, deci este chiar cheie pentru R , nu numai supercheie.



Proiectia unei multimi de DF

- Aşa cum s-a mentionat anterior, inchiderea unei multimi de dependente functionale F^+ a fost introdusa si pentru a putea defini setul de dependente functionale mostenite de o schema de relatie obtinuta prin descompunerea unei scheme incorrect proiectata.

- Sa analizam o noua structura a relatiei:

COMPONENTE_FURNIZORI =

(ID_com, Den_com, Pret, ID_furniz, Den_furniz, Adresa).

Multimea de dependente functionale F este:

$$F = \{ ID_com \rightarrow Den_com, ID_com \rightarrow Pret,$$
$$ID_com \rightarrow ID_furniz, ID_furniz \rightarrow Den_furniz,$$
$$ID_furniz \rightarrow Adresa \}$$



Proiectia unei multimi de DF

- Prin descompunerea acestei relatii in doua relatii, obtinem relatiile:

COMPONENTE = (ID_com, Den_com, Pret, ID_furniz)

FURNIZORI = (ID_furniz, Den_furniz, Adresa)

- Atributele relatiei initiale se regasesc fie doar intr-una dintre schemele rezultate, fie in amandoua.

Se pune problema identificarii dependentelor mostenite de cele doua relatii de la relatia initiala.

Pentru aceasta trebuie definita proiectia unei multimi de dependente pe o multime de atribute.



Proiectia unei multimi de DF

- **Definitie.** Fie o relatie R , o multime asociata de dependente functionale F si o submultime de atribute $S \subseteq R$.

Proiectia multimii de dependente F pe S , notata cu $\pi_S(F)$ este multimea dependentelor din F^+ care au atributele din partea stanga si cea dreapta incluse in S .

Formal, putem scrie:

$$\pi_S(F) = \{X \rightarrow Y \in F^+ \mid X, Y \subseteq S\}$$



Proiectia unei multimi de DF

Exemplu:

Fie relatia $\text{COMPONENTE_FURNIZORI} =$

(ID_com , Den_com , Pret , ID_furniz , Den_furniz , Adresa)
avand multimea de dependente functionale:

$F = \{ \text{ID_com} \rightarrow \text{Den_com}, \text{ID_com} \rightarrow \text{Pret}, \text{ID_com} \rightarrow \text{ID_furniz}, \text{ID_furniz} \rightarrow \text{Den_furniz}, \text{ID_furniz} \rightarrow \text{Adresa} \}.$

- Aplicand definitia se obtin proiectiile urmatoare:

$F_{\text{COMPONENTE}} = \pi_{\text{COMPONENTE}}(F) = \{ \text{ID_com} \rightarrow \text{Den_com}, \text{ID_com} \rightarrow \text{Pret}, \text{ID_com} \rightarrow \text{ID_furniz} \}$

$F_{\text{FURNIZORI}} = \pi_{\text{FURNIZORI}}(F) = \{ \text{ID_furniz} \rightarrow \text{Den_furniz}, \text{ID_furniz} \rightarrow \text{Adresa} \}$



Proiectia unei multimi de DF

- Observatie: Atunci cand descompunem o schema se poate intampla ca unele dintre dependentele schemei initiale sa se piarda.
Exemplu: Fie $R = ABCD$ si $F = \{ A \rightarrow B, B \rightarrow C, C \rightarrow D, D \rightarrow A \}$.
In cazul in care descompunem R in $R_1 = AB$ si $R_2 = CD$
atunci: $F_{R1} = \pi_{R1}(F) = \{ A \rightarrow B, B \rightarrow A \}$ si
 $F_{R2} = \pi_{R2}(F) = \{ C \rightarrow D, D \rightarrow C \}$
- A doua dependenta din fiecare multime nu este in F dar este in F^+ (obtinuta prin tranzitivitate).
- Observam insa ca dependentele $B \rightarrow C$ si $D \rightarrow A$ nu mai pot fi obtinute nici din F_{R1} , nici din F_{R2} si nici din reuniunea lor. Intr-un alt capitol va fi prezentata o metoda prin care se poate verifica daca prin descompunere dependentele initiale sunt pastrate sau nu.



Inciderea unei multimi de atribute

- **Definitie :** Fie R o schema de relatie, F multimea de dependente asociata si $X \subseteq R$.

Se poate defini X^+ ca fiind **incliderea multimii de atribute X in raport cu F** astfel:

$$X^+ = \{ A \mid X \rightarrow A \in F^+ \}$$

- Deci X^+ contine toate atributele care apar in partea dreapta a dependentelor din F sau care se pot deduce din F folosind reguli si axiome.



Inciderea unei multimi de attribute

➤ **Algoritm de calcul pentru X^+**

Fie R o schema de relatie, F multimea de dependente asociata si $X \subseteq R$.

Pentru calculul inciderii multimii de attribute X^+ se aplica urmatorul **algoritm** iterativ:

- Se porneste cu $X^{(0)} = X$
- Pentru $i \geq 1$,

$$X^{(i)} = X^{(i-1)} \cup \{ A \mid (\exists) Y \rightarrow A \in F \text{ cu } Y \subseteq X^{(i-1)} \}$$

- Daca $X^{(i)} = X^{(i-1)}$ sau $X^{(i)} = R$, atunci

STOP.

- Scopul introducerii acestei notiuni este si acela de a putea ocoli calculul lui F^+ in alti algoritmi sau definitii.₃₂



Inciderea unei multimi de attribute

Exemplu: Fie $R = ABCDE$ si $F = \{ A \rightarrow B, A \rightarrow C, D \rightarrow E \}$.

Pentru a calcula A^+ , D^+ si $(AD)^+$ procedam astfel:

✓ Calcul A^+ :

- $X^{(0)} = \{A\}$
- Din $A \rightarrow B$ si $A \rightarrow C$ rezulta ca $X^{(1)} = X^{(0)} \cup \{B, C\} = \{A\} \cup \{B, C\} = ABC$
- Sigurele dependente care au partea dreapta in $X^{(1)}$ sunt tot primele doua, deci

$$X^{(2)} = X^{(1)} \cup \{B, C\} = \{A, B, C\} \cup \{B, C\} = ABC$$

- Deoarece $X^{(2)} = X^{(1)}$ \Rightarrow STOP.

- Rezulta ca $A^+ = ABC$

✓ Calcul D^+ : In mod similar, rezulta $D^+ = DE$.



Inciderea unei multimi de attribute

✓ Calcul $(AD)^+$:

- $X^{(0)} = \{A, D\}$
- Din $A \rightarrow B$, $A \rightarrow C$ si $D \rightarrow E$ rezulta ca
$$X^{(1)} = X^{(0)} \cup \{ B, C, E \} = \{ A, D \} \cup \{ B, C, E \} = ABCDE$$
- Deoarece $X^{(1)} = R \Rightarrow$ STOP (oricate iteratii am face nu mai pot sa apara noi attribute).
- Rezulta ca $(AD)^+ = ABCDE$



Inciderea unei multimi de atribut

Avem urmatorul rezultat teoretic:

- **Lema.** Fie R o schema de relatie, F multimea de dependente asociata si $X, Y \subseteq R$.

Atunci $X \rightarrow Y$ se poate deduce din F daca si numai daca $Y \subseteq X^+$.

Demonstratie:

Fie $Y = A_1 A_2 A_3 \dots A_n$. Facem ipoteza ca $Y \subseteq X^+$. Prin definitia lui X^+ , $X \rightarrow A_i$ este implicat de axiomele Armstrong pentru orice i .

Dar cum $Y = A_1 A_2 \dots A_n = \{A_1 \cup A_2 \cup \dots \cup A_n\}$, prin regula de reuniune rezulta $X \rightarrow Y$, deoarece $X \rightarrow A_i$ pentru orice i .



Inciderea unei multimi de atrbute

- Exista teoreme care arata ca “**sistemul axiomelor lui Armstrong este complet si corect**”.
- **Complet** – daca sunt date dependentele functionale din F , prin axiome deducem toate dependentele functionale din F^+ .
- **Corect** – plecand de la F , aplicand regulile de inferenta reprezentate de axiomele Armstrong , nu putem deduce dependente functionale care nu sunt in F^+ .



Inciderea unei multimi de atribute

Consecinte:

- X^+ a fost definit ca multimea de atribute A, astfel incat $X \rightarrow A$ decurge din F, folosind axiomele lui Armstrong.
- O definitie echivalenta este: X^+ este multimea atributelor A astfel incat F implica logic pe $X \rightarrow A$.
- F^+ a fost introdusa ca multimea dependentelor implicate logic de catre F. Se poate insa defini F^+ ca multimea dependentelor care decurg din F prin axiomele Armstrong.
- Consecintele pot fi demonstate pe baza informatiilor anterioare.



O alta definitie pentru cheie

- Pe baza propozitiei din paragraful anterior se poate da o alta definitie pentru cheia sau supercheia unei relatii pe inchiderea unei multimi de atrbute X^+ (si nu pe F^+ , ca in subcapitolul anterior).
- **Definitie:** Fie R o schema de relatie, F multimea de dependente functionale asociata si $X \subseteq R$.
- X este **cheie** pentru R daca si numai daca:
 - $X^+ = R$;
 - X este minimala: oricare ar fi $Y \subset X$, $Y \neq X$ atunci $Y^+ \neq R$ (deci orice submultime stricta a lui X nu mai indeplineste conditia anterioara).
 - X este **supercheie** pentru R daca este indeplinita numai prima conditie.



O alta definitie pentru cheie

Echivalenta acestei definitii cu cea anterioara este evidenta:

- $X^+ = R$ inseamna ca $X \rightarrow R$ conform lemei;
- Minimalitatea este de asemenea definita echivalent:

Daca $Y \subset X$, $\neg(F \Rightarrow Y \rightarrow R)$ este echivalenta cu $\neg(Y^+ = R)$, adica $Y^+ \neq R$.

- Folosind aceasta definitie se poate defini o euristică de gasire a cheilor unei relații:



Euristica de gasire a cheilor unei relatii

➤ **Euristica de gasire a cheilor unei relatii**

Pentru gasirea cheilor unei relatii pornim de la observatia ca atributele care nu sunt in partea dreapta a niciunei dependente nu pot sa apară in procesul de inchidere a unei multimi de atribut, deci ele aparțin oricărei chei a relației.

- Fie **R** o schema de relatie si **F** multimea de dependente functionale asociata (F in forma canonica).
 - **Cheia unica sau cheile alternative ale lui R** se calculeaza in pasii urmatori:



Euristica de gasire a cheilor unei relatii

- 1) Se porneste de la multimea de atribute $X \subseteq R$ care nu apar in partea dreapta a niciunei dependente.
- 2) Se calculeaza X^+ . Daca $X^+ = R$ atunci X este o cheie minimala a relatiei R si calculul se opreste aici.
Pasii urmatori se efectueaza doar daca $X^+ \neq R$.
- 3) Se adauga la X cate un atribut din $R - X^+$ obtinandu-se o multime de chei candidat.
- 4) Se calculeaza X^+ pentru fiecare dintre candidate. Daca se obtin toate atributele lui R atunci acel X este o cheie a lui R .
- 5) Se repeta pasii 3 si 4 pornind de la acele multimi candidat X care nu sunt gasite ca si chei la pasul anterior. Dintre multimile candidat nu luam niciodata in considerare pe cele care contin o cheie gasita anterior.
- 6) Procesul se opreste cand nu se mai pot face augmentari.



Euristica de gasire a cheilor unei relatii

Exemple:

1) Fie $R = ABCDE$ si $F = \{ A \rightarrow B, A \rightarrow C, D \rightarrow E \}$.

1.1) Multimea atributelor care nu apar in partea dreapta a niciunei dependente este $X = (AD)$.

1.2) Calculam $(AD)^+ = ABCDE = R$. Nu mai trebuie verificata minimalitatea deoarece A si D trebuie sa faca parte din orice cheie (oricum, $A^+ = ABC$ si $D^+ = DE$) .

1.3) Procesul se opreste. Rezulta ca AD este cheie pentru R.

2) Fie $R = ABCDE$ si $F = \{A \rightarrow B, B \rightarrow A, A \rightarrow C, D \rightarrow E\}$.

2.1) Multimea atributelor care nu apar in partea dreapta a niciunei dependente este $X = D$.

2.2) Calculam $(D)^+$. Obtinem $(D)^+ = DE$. Rezulta ca D nu este cheie pentru R.



Euristica de gasire a cheilor unei relatii

- 2.3) Calculam multimea de candidate: augmentam D cu atribute din $R - D^+ = ABCDE - DE = ABC$. Obtinem AD, BD si CD
- 2.4) Calculam inchiderile lor. Obtinem $(AD)^+ = R$, $(BD)^+ = R$ si $(CD)^+ = CDE \neq R$. Rezulta ca AD si BD sunt chei ale lui R dar CD nu este cheie.
- 2.5) Calculam o noua multime de candidate pornind de la CD. Putem augmenta CD cu atribute din $R - (CD)^+ = ABCDE - CDE = AB$. Niciuna dintre augmentari nu este insa posibila pentru ca atat ACD cat si BCD contin o cheie gasita anterior (AD respectiv BD).
- 2.6) Procesul se opreste. Singurele chei ale lui R raman AD si BD.



Acoperiri de multimi de dependente

- **Definitie:** Fie F, G – multimi de dependente functionale. Daca $F^+ = G^+$, spunem ca F si G sunt **echivalente**. Daca F si G sunt echivalente, spunem ca F **acopera** G (si G acopera F).
- Pentru a stabili ca F si G sunt echivalente (sau $F = G$), pentru fiecare dependenta $Y \rightarrow Z$ din F se verifica daca $Y \rightarrow Z$ este in G^+ , folosind algoritmul anterior pentru a calcula Y^+ si pentru a verifica apoi daca $Z \subseteq Y^+$ (potrivit axiomelor, daca $Z \subseteq Y^+$, atunci $Y \rightarrow Z$). Daca o dependenta $Y \rightarrow Z$ din F nu este in G^+ , atunci sigur $F^+ \neq G^+$.



Acoperiri de multimi de dependente

- Daca fiecare dependenta din F este in G^+ , atunci fiecare dependenta $V \rightarrow W$ din F^+ este in G^+ ; pentru a arata ca $V \rightarrow W$ este in G^+ , se demonstreaza ca fiecare $Y \rightarrow Z$ din F este in G^+ , apoi ca $V \rightarrow W$ este in F^+ .
- Pentru a arata ca fiecare dependenta din G este, de asemenea, si in F^+ , se procedeaza in mod analog.
- F si G vor fi echivalente daca fiecare dependenta din F este si in G^+ , iar fiecare dependenta din G este si in F^+ .



Acoperiri de multimi de dependente

- **Lema.** Fiecare multime de dependente functionale F este acoperita de o multime de dependente G , in care nicio dependenta nu are in parte dreapta mai mult de un atribut.

Demonstratie:

Fie G o multime de dependente $X \rightarrow A$, astfel incat pentru o dependenta $X \rightarrow Y$ in F , A este in Y . Atunci $X \rightarrow A$ decurge din $X \rightarrow Y$ prin regula de descompunere. Asadar, $G \subseteq F^+$. Dar $F \subseteq G^+$, deoarece, daca $Y = A_1 A_2 \dots A_n$, atunci $X \rightarrow Y$ rezulta din $X \rightarrow A_1$, $X \rightarrow A_2$, ..., $X \rightarrow A_n$, prin regula de reuniiune.



Multime minimala de dependente

O multime de dependente F este **minimală** daca:

1. Partea dreapta a fiecarei dependente din F contine un singur atribut;
2. Pentru nicio dependenta $X \rightarrow A$ din F , multimea $F - \{X \rightarrow A\}$ nu este echivalenta cu F ;
3. Pentru nicio dependenta $X \rightarrow A$ din F si pentru nicio submultime $Z \subseteq X$, multimea $F - \{X \rightarrow A\} \cup \{Z \rightarrow A\}$ nu este echivalenta cu F .

De fapt, conditia a 2-a garanteaza ca nicio dependenta din F nu este redundanta, iar a 3-a ca niciun atribut din partea stanga nu este redundant. Desigur, niciun atribut din dreapta nu este redundant, deoarece fiecare parte dreapta contine un singur atribut (conditia 1).



Multime minimală de dependente

- **Teorema.** Fiecare multime de dependente F este echivalentă cu o multime F' , care este minimală.

Exemplu:

Fie: $F = \{AB \rightarrow C, D \rightarrow EG, C \rightarrow A, BE \rightarrow C, BC \rightarrow D, CG \rightarrow BD, ACD \rightarrow B, CE \rightarrow AG\}$.

- În ultima lema a fost indicat – la demonstrație – “un fel” de algoritm pentru fragmentarea partilor drepte ale dependentelor funcționale.

Aplicându-l obținem următoarele dependente:

$AB \rightarrow C, C \rightarrow A, BC \rightarrow D, ACD \rightarrow B, D \rightarrow E, D \rightarrow G, BE \rightarrow C, CG \rightarrow B, CG \rightarrow D, CE \rightarrow A, CE \rightarrow G$.



Multime minimală de dependente

- Dependentele $CG \rightarrow B$ și $CE \rightarrow A$ sunt redundante.
- ✓ Sa aratam ca $CG \rightarrow B$ este redundanta cu $ACD \rightarrow B$:
 - Deoarece $C \rightarrow A$, rezulta ca $ACD \rightarrow B$ este echivalenta cu $CD \rightarrow B$;
 - Acum trebuie sa aratam ca $CG \rightarrow B$ este redundanta, fata de $CD \rightarrow B$:

Pornind de la $CD \rightarrow B$,
avem $CG \rightarrow D$,
deci inlocuim pe D in $CD \rightarrow B$ si obtinem
 $C(CG) \rightarrow B$, adica $CG \rightarrow B$.



Multime minimala de dependente

- Procedand astfel, se obtine **multimea de dependente minimele**:
 $\{AB \rightarrow C, C \rightarrow A, BC \rightarrow D, CD \rightarrow B, D \rightarrow E, D \rightarrow G, BE \rightarrow C, CG \rightarrow D, CE \rightarrow G\}$.
- Daca din F eliminam dependentele
 $CE \rightarrow A, CG \rightarrow D$ si $ACD \rightarrow B$,
obtinem **acoperirea minima**:
 $F' = \{AB \rightarrow C, C \rightarrow A, BC \rightarrow D, D \rightarrow E, D \rightarrow G, BE \rightarrow C, CG \rightarrow B, CE \rightarrow G\}$.
- Se observa ca cele doua acoperiri minimele contin un numar diferit de dependente.



Capitolul 6

Normalizarea schemelor de relatie



Normalizarea schemelor de relatie

- Dupa cum am aratat in capitolele anterioare, proiectarea unei baze de date se poate face in mai multe feluri si se pune problema care este cea mai buna optiune.
- Un obiectiv important in alegerea modelului de date este realizarea unei reprezentari corecte a datelor, a relatiilor dintre ele si a restrictiilor impuse asupra lor.
- Pentru realizarea acestui obiectiv se utilizeaza tehnica normalizarii, care are ca scop principal identificarea schemei de relatii care sa modeleze cat mai corect realitatea functionala.
- Proiectarea incorecta a schemelor de relatie poate duce la aparitia diferitelor anomalii care complica procesul de dezvoltare sau exploatare a unei aplicatii software.



Normalizarea schemelor de relatie

- Ca metoda de testare a corectitudinii unei scheme de relatie este verificarea dependentelor functionale atasate schemei.
- Una dintre regulile de baza in proiectare este aceea ca datele nu trebuie sa fie redundante, adica aceleasi date sa fie stocate de mai multe ori in aceeasi relatie sau relatii diferite.
- O alta regula este ca datele care se pot deduce prin prelucrare din alte date nu trebuie stocate in baza de date.
- **Normalizarea** reprezinta un proces de descompunere a unei relatii in mai multe relatii cu scopul eliminarii anomalilor de proiectare si exploatare a bazei de date. Procesul de normalizare se realizeaza cu ajutorul **formelor normale** care impun **regulile de descompunere**.
- O schema de relatie obtinuta in urma normalizarii , care indeplineste un set de standarde si cerintele specifice, se spune ca este intr-o **forma normala**.



Normalizarea schemelor de relatie

- In anul 1970 Edgar F. Codd a definit formalismul primelor forme normale: Forma Normala 1(FN1), Forma Normala 2(FN2) si Forma Normala 3(FN3).
- In anul 1974, Edgar F. Codd impreuna cu Raymond F. Boyce, au definit FNBC(Forma Normala Boyce-Codd).
- Ulterior au fost definite formele normale FN4, FN5 si FN6 care au avut aplicabilitate destul de redusa in proiectarea bazelor de date.



Forma normala 1 (FN1)

- **Definitie:** O relatie R se gaseste in **Forma Normala 1 (FN1)** daca si numai daca:
 - Pe toate atributele sale exista doar valori atomice ale datelor(nu exista attribute cu valori multiple);
 - Nu exista attribute sau grupuri de attribute care se repeta.
 - Semnificatia termenului “atomic” este similara cu cea de la modelul entitate asociere: valoarea respectiva este intotdeauna folosita ca un intreg si nu se utilizeaza niciodata doar portiuni din aceasta. In prima forma normala, domeniul fiecarui atribut este construit din valori indivizibile. Cu alte cuvinte, toate atributele trebuie sa fie atomice, adica sa contina o singura informatie.

Exemplu: ABONATI (Cod_abonat, Nume, Adresa, Tip_abon). Aceasta relatie este in FN1 daca atributul Adresa este atomic, adica niciodata nu este nevoie sa fie folosite doar anumite parti ale sale (oras, sector, strada, etc.).



Forma normala 2 (FN2)

- Fiind data o schema de relatie R si multimea de dependente functionale asociata F, putem defini inca doua concepte.
Fie A un atribut care nu face parte din cheie si X o multime de atribute din R care formeaza o cheie a relatiei. Atunci:
 - **Definitie:** O dependenta functionala $Y \rightarrow A$ se numeste **dependenta paritala** daca Y este strict inclusa intr-o cheie a relatiei R.
(X este cheie, $Y \subset X$, $Y \rightarrow A$ si A nu face parte din cheie) sau
(avem dependenta $X \rightarrow Y \rightarrow A$, unde: $X=cheie$, $Y \subset X$ si $A \notin X$).
 - **Definitie:** O dependenta functionala $Y \rightarrow A$ se numeste **dependenta tranzitiva** daca Y nu este inclusa in nicio cheie a relatiei R.
(X este cheie, $Y \not\subset X$, $Y \rightarrow A$ si A nu face parte din cheie) sau
(avem dependenta $X \rightarrow Y \rightarrow A$, unde: $X=cheie$, $Y \not\subset X$ si $A \notin X$).



Forma normala 2 (FN2)

- **Definitie:** Fie R o schema de relatie si F multimea de dependente functionale asociata. Relatia R este in **Forma Normala 2 (FN2)** daca si numai daca F respecta cerintele FN1 si nu contine dependente partiale (dar poate contine dependente tranzitive).

Cu alte cuvinte, o relatie se gaseste **Forma Normala 2** daca si numai daca:

- Se gaseste in FN1 si
- Orice atribut care nu face parte din cheie va fi identificat de intreaga cheie, nu doar de unele atribute care fac parte din cheie.



Forma normala 2 (FN2)

Observatie:

- Daca o entitate se gaseste in FN1 si cheia sa este formata dintr-un singur atribut, atunci se gaseste automat si in FN2.
- FN1 si FN2 nu garanteaza eliminarea anomalilor si nu sunt recomandate pentru proiectarea schemelor de relatii ale unei baze de date.



Forma normala 2 (FN2)

Exemplu:

1) Relatia PRODUSE = Cod_prod, Den_prod, Pret,
Cod_furniz, Den_furniz, Adresa

cu dependentele functionale:

$F = \{ \text{Cod_prod} \rightarrow \text{Den_prod}, \text{Cod_prod} \rightarrow \text{Pret},$
 $\text{Cod_prod} \rightarrow \text{Cod_furniz}, \text{Cod_furniz} \rightarrow \text{Den_furniz},$
 $\text{Cod_furniz} \rightarrow \text{Adresa} \}$ si cheia unica Cod_prod.

- Relatia este in FN2 deoarece cheia are un singur atribut. Relatia nu are dependente partiale iar ultimele doua dependente sunt dependente tranzitive deoarece Cod_furniz nu apartine cheii unice Cod_prod.



Forma normala 2 (FN2)

2) Relatia

NOTE = Nr_matricol, Nume_stud, Cod_discip,
Den_discip, Nota, Data_ex

are cheia (Nr_matricol, Cod_discip),

considerand ca un student are o singura nota la o disciplina (nota finala).

- Relatia NOTE nu este in FN2 deoarece dependentele
 $\text{Nr_matricol} \rightarrow \text{Nume_stud}$ si $\text{Cod_discip} \rightarrow \text{Den_discip}$
sunt dependente partiale (Nr_matricol , Cod_discip fac parte din cheie).



Forma normala 3 (FN3)

- Pentru a defini forma FN3 este necesara definirea notiunii de **atribut prim**:
- **Definitie.** Fie R o schema de relatie si F multimea de dependente functionale asociata.

Un atribut $A \in R$ se numeste **atribut prim** daca el apartine unei chei a lui R.

Exemplu:

Fie $R = ABCDE$ avand $F = \{ A \rightarrow B, B \rightarrow A, A \rightarrow C, D \rightarrow E \}$. Cum cheile relatiei sunt AD si BD rezulta ca in R sunt trei atribute prime: A, B si D.



Forma normala 3 (FN3)

- **Definitie.** Fie R o schema de relatie si F multimea de dependente functionale asociata.

Relatia R este in **Forma Normala 3(FN 3)** daca si numai daca oricare ar fi o dependenta netriviala

$X \rightarrow A$ din F, atunci:

- X este supercheie pentru R
sau
- A este atribut prim.



Forma normala 3 (FN3)

- De remarcat ca daca in F avem dependente care contin mai multe atribute in partea dreapta putem aplica regula de descompunere pentru a obtine dependente care in partea dreapta au cate un singur atribut.
- Observatie : O relatie se gaseste in FN3 daca si numai daca se gaseste in FN2 si, in plus, niciun atribut care nu este parte a unei chei nu depinde de un alt atribut care nu face parte din cheie. Cu alte cuvinte, nu se accepta dependente tranzitive, adica un atribut sa depinda de o cheie in mod indirect.



Forma normala 3 (FN3)

Exemple:

- 1) Fie $R=NLAP$, o schema de relatie pentru furnizori cu atributele (Nume, Localitate, Articol, Pret) si dependentele $F=\{NA \rightarrow P \text{ si } N \rightarrow L\}$.
 - Relatia are cheia NA si nu respecta FN3 deoarece are dependenta partiala $N \rightarrow L$ (de asemenea nu respecta nici FN2). Intr-adevar, fie $X=NA$, $Y=N$. Atributul L (localitate) este neprim, deoarece singura cheie este NA. Atunci $X \rightarrow Y$ si $Y \rightarrow L$ sunt dependente valabile, pe cand $Y \rightarrow X$ (adica $N \rightarrow NA$) nu este valabila. Vom observa ca in acest caz, $X \rightarrow Y$ si $Y \rightarrow L$ nu numai ca “functioneaza” in R , dar ele sunt dependente date. In general, este suficient ca $X \rightarrow Y$ si $Y \rightarrow L$ sa decurga dintr-o multime data de dependente, chiar daca sunt date ca atare.
- 1) Fie $R=OSC$, o schema de relatie pentru localitati cu atributele (Oras, Strada, Cod) si dependentele $F=\{ OS \rightarrow C \text{ si } C \rightarrow O \}$. Cheile sunt OS si SC. Relatia are toate atributele prime si este in FN3.



Forma normala 3 (FN3)

3) Fie $R=MADS$, o relatie pentru magazine cu atributele (Magazin, Articol, Departament, Sef). Presupunem ca functioneaza urmatoarele dependente functionale:

- $MA \rightarrow D$ (fiecare articol, in fiecare magazin , este vandut de cel mult un departament/raion);
 - $MD \rightarrow S$ (fiecare departament/raion , din fiecare magazin, are un sef).
-
- Relatia nu este in FN3 deoarece:
 - Relatia are o singura cheie , pe MA.
 - Daca notam $X=MA$ si $Y=MD$, atunci $X \rightarrow D$ si $Y \rightarrow S$ nu respecta regulile care definesc FN3 deoarece D si S nu sunt attribute prime.
 - Relatia este in FN2 pentru ca nu exista dependente partiale (nicio submultime proprie a cheii MA nu determina functional attributele D sau S).



Forma normala 3 (FN3)

✓ Necesitatea FN3

- Asa cum am aratat, prin FN3 se evita multe dintre probleme legate de redundanta si de anomaliiile de actualizare.
- Putem presupune, astfel, ca dependentele functionale $X \rightarrow Y$ nu reprezinta numai o restrictie de integritate asupra relatiilor, ci reprezinta, in acelasi timp, o legatura (asociere) pe care baza de date "are intentia sa o memoreze". Cu alte cuvinte, daca atributelor din X le este asignata o multime de valori, consideram important sa stim ce valoare, pentru fiecare atribut din Y, este asociata cu aceasta "asignare" de valori pentru atributele din X.



Forma normala 3 (FN3)

- Daca avem o dependenta partiala $Y \rightarrow A$, X fiind o cheie iar Y o submultime proprie a lui X , atunci in fiecare tuplu folosit pentru a asocia o valoare, din multimea asignata lui X , cu valori pentru alte atrbute in afara de A si de atrbutele din X , trebuie sa apară aceeasi asociere intre X si A .
- Aceasta situatie este usor de evideniat in relatia $R=NLAP$, cu cheia NA in care $N \rightarrow L$ este o dependenta partiala, iar localitatea furnizorului trebuie sa fie repetata pentru fiecare articol livrat de furnizor.
Evident ca FN3 elimina aceasta posibilitate, precum si redundantele respective si anomaliile de actualizare.



Forma normala 3 (FN3)

- In caz ca exista o dependenta tranzitiva $X \rightarrow Y \rightarrow A$,
atunci nu putem asocia o valoare Y cu o valoare X,
daca nu exista o valoare A asociata cu valoarea Y.
Aceasta situatie conduce la anomalii de inserare si de
stergere deoarece nu putem insera o asociere
X-la-Y fara o asociere Y-la-A, iar daca stergem
valoarea A asociata cu o valoare Y data vom “pierde
legatura” unei asocieri X-la-Y.
- De exemplu, in schema de relatie R= MADS, cu
dependentele F={MA \rightarrow D si MD \rightarrow S}, nu putem
inregistra un departament oarecare daca acel
departament nu are sef, iar daca stergem un sef
“dispare” si departamentul asociat lui.



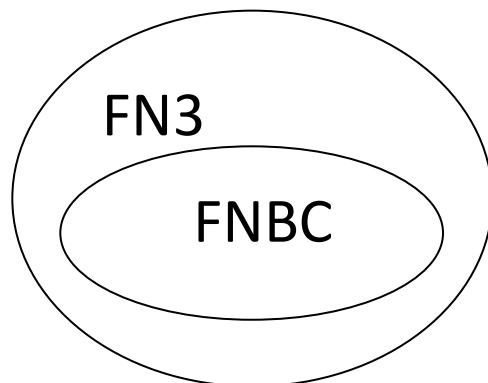
Forma normala Boyce-Codd (FNBC)

- **Definitie.** Fie R o schema de relatie si F multimea de dependente functionale asociata. Se spune ca R este in **Forma Normala Boyce-Codd (FNBC)** daca si numai daca oricare ar fi o dependenta netriviala $X \rightarrow Y$ din F atunci X este supercheie pentru R.
- Rezulta ca o relatie este in FNBC daca si numai daca fiecare dependenta din F are in partea stanga o supercheie.
- Nu este obligatoriu ca F sa fie in forma canonica, dar trebuie sa nu contine dependente triviale (obtinute din prima axioma de reflexivitate, de tipul $AB \rightarrow A$ sau $AB \rightarrow AB$).
- Dependenta triviala $R \rightarrow R$ este admisa in FNBC(o relatie nu contine linii dupicat si cheia este compusa din toate atributele).



Forma normala Boyce-Codd (FNBC)

- Observatie: Conditia de FNBC este inclusa in definitia FN3. Din acest motiv orice relatie care este in FNBC este implicit si in FN3. Reciproca nu este adevarata. Rezulta de asemenea ca daca o schema de relatie nu este in FN3 ea nu poate fi nici in FNBC.





Forma normala Boyce-Codd (FNBC)

Exemplu:

- 1) Relatia $R = ABCDE$ avand $F = \{ A \rightarrow B, B \rightarrow A, A \rightarrow C, D \rightarrow E \}$ nu este in forma normala Boyce-Codd deoarece are cheile AD si BD dar nicio dependenta nu are in partea stanga o supercheie a lui R.
- 2) Relatia $R = ABCD$ avand $F = \{ AB \rightarrow C, AB \rightarrow D, D \rightarrow A \}$ are cheia unica AB.
 - Relatia este in FN3 deoarece primele doua dependente au in partea stanga o supercheie (AB) iar a treia dependenta are in partea dreapta atributul prim A.
 - Relatia nu este in FNBC deoarece a treia dependenta violeaza definitia pentru aceasta forma normala (nu are in partea stanga o supercheie).



Forma normala Boyce-Codd (FNBC)

3) Relatia $R = ABCDE$ avand $F = \{ A \rightarrow B, B \rightarrow A, A \rightarrow C, D \rightarrow E \}$ are cheile AD si BD. Rezulta ca:

- R nu este in FN3 deoarece dependentele 3 si 4 nu au nici supercheie in partea stanga si nici atribut prim in partea dreapta;
- R nu este in FNBC deoarece nu e in FN3.

4) Relatia $R = ABCD$ avand $F = \{ A \rightarrow B, B \rightarrow C, C \rightarrow D, D \rightarrow A \}$ are cheile A, B, C si D. Rezulta ca:

- R este in FNBC deoarece in partea stanga a dependentelor sunt numai superchei;
- R este in FN3 deoarece este in FNBC.



Forma normala Boyce-Codd (FNBC)

5) Relatia PRODUSE_FURNIZORI = Cod_prod, Den_prod, Pret,
Cod_furniz, Den_furniz, Adresa
cu dependentele functionale:

$$F = \{ \text{Cod_prod} \rightarrow \text{Den_prod}, \text{Cod_prod} \rightarrow \text{Pret}, \text{Cod_prod} \rightarrow \text{Cod_furniz}, \text{Cod_furniz} \rightarrow \text{Den_furniz}, \text{Cod_furniz} \rightarrow \text{Adresa} \}$$

- Relatia nu este in FNCB deoarece cheia unica este Cod_prod dar exista dependente care nu au in partea stanga o supercheie:

$$\text{Cod_furniz} \rightarrow \text{Den_furniz}, \text{Cod_furniz} \rightarrow \text{Adresa}$$

- Relatia nu este nici in FN3 deoarece Den_furniz si Adresa nu sunt atribute prime, in plus aceste dependente sunt si tranzitive.



Forma normala Boyce-Codd (FNBC)

6) Relatia PRODUSE_FURNIZORI = Cod_prod, Den_prod, Pret, Cod_furniz, Den_furniz, Adresa cu dependentele:
 $F = \{ \text{Cod_prod} \rightarrow \text{Den_prod}, \text{Cod_prod} \rightarrow \text{Pret},$
 $\text{Cod_prod} \rightarrow \text{Cod_furniz}, \text{Cod_furniz} \rightarrow \text{Den_furniz},$
 $\text{Cod_furniz} \rightarrow \text{Adresa} \}$

- Consideram proiectia

PRODUSE= Cod_prod, Den_prod, Pret, Cod_furniz

$F_{\text{PRODUSE}} = \pi_{\text{PRODUSE}}(F) = \{ \text{Cod_prod} \rightarrow \text{Den_prod},$
 $\text{Cod_prod} \rightarrow \text{Pret}, \text{Cod_prod} \rightarrow \text{Cod_furniz} \}$ care este in FNCB deoarece cheia relatiei este Cod_prod si toate dependentele au in partea stanga o supercheie (asa cum s-a mentionat, orice cheie este in acelasi timp si supercheie). Proiectia este in FN3 deoarece este in FNCB.



Anomalii in baze de date

Anomaliiile care pot sa apară într-o bază de date sunt de două feluri:

- Anomalii de **proiectare**: redundante, tipuri gresite de atribut, constrangeri incorecte, etc.
- Anomalii de **funcționare**: sunt cele care apar la operații DML (inserare, modificare, stergere).
- Pentru a înțelege mai bine anomaliiile să considerăm relația NOTE(în care un student are o singură nota la o disciplină):

NOTE = Matricol, Nume, Disciplina, Nota, Data_ex,
Cod_spec, Den_spec
cu următoarele date:



Anomalii in baze de date

Matricol	Nume	Disciplina	Nota	Data_ex	Cod_spec	Den_spec
1011	Ionescu Silvia	BD1	9	08.06.2020	1	CTI
1011	Ionescu Silvia	BD2	10	10.02.2021	1	CTI
1022	Popescu Daniel	PM	8	11.02.2021	2	AII
1022	Popescu Daniel	SO	7	03.06.2020	2	AII
1033	Popa Cornel	BD1	8	08.06.2020	1	CTI
1033	Popa Cornel	BD2	8	10.02.2021	1	CTI



Anomalii in baze de date

➤ **Redundanta** apare atunci cand datele sunt stocate de mai multe ori in baza de date(in aceeasi relatie sau in relatii diferite).

In relatia NOTE se observa ca atributul Den_spec este redundant pentru ca este suficient atributul Cod_spec care specifica codul specializarii studentului si determina unic denumirea acesteia. In acest caz ar trebui sa se descompuna relatia in doua relatii , NOTE si SPECIALIZARI .



Anomalii in baze de date

Existenta redundantei intr-o baza de date produce urmatoarele efecte negative:

- Alocare suplimentara de spatiu fizic;
- Inconsistenta datelor;
- Cresterea costurilor de acces la baza de date;
- Prelucrari eronate ale datelor;
- Creste probabilitatea erorii umane in timpul actualizarilor pe baza de date.



Anomalii in baze de date

- **Tipurile atributelor** pot genera anomalii, de exemplu daca atributul Data_ex este definit de tip Number, in loc de tip Date, atunci cand se insereaza data se va genera eroare de format. Daca este definit Text pot aparea erori la anumite functii de tip data calendaristica.
- **Constrangerile** incorecte apar in momentul definirii relatiilor dar efectul lor se vede, de regula, in timpul functionarii. De exemplu, daca in tabela NOTE se defineste cheia relatiei ca fiind Matricol, atunci nu se poate insera decat nota la o singura disciplina, a doua operatie va genera o eroare de violare de cheie.



Anomalii in baze de date

- **Inserarea** poate genera anomalii, de exemplu codurile specializarilor sa nu corespunda cu denumirile de specializare sau sa fie folosite coduri diferite pentru aceeasi specializare. Anomalia se inlatura daca se creeaza relatia SPECIALIZARI(Cod_spec, Den_spec). O alta anomalie este ca nu se stiu numarul matricol si specializarea unui student pana cand nu se insereaza cel putin o nota la o disciplina.
- **Modificarea** datelor poate duce la inconsistenta datelor, de exemplu daca un student isi schimba specializarea trebuie sa se modifice Cod_spec, Den_spec la toate inregistrarile si este posibil sa fie omise unele dintre ele.



Anomalii in baze de date

- **Stergerea** poate produce anomalii, de exemplu daca se sterg toate inregistrarile aferente unui student atunci nu se mai stie numarul matricol si nici specializarea lui. In acest caz se poate crea relatia STUDENTI(Matricol, Nume, Cod_spec) si anomalia este evitata.
- Majoritatea anomalilor semnalate au aparut deoarece structura relatiei NOTE contine doua clase de obiecte care trebuie separate. Daca folosim modelul Entitate-Asociere diagrama corecta este cea din Figura 2.



Anomalii in baze de date

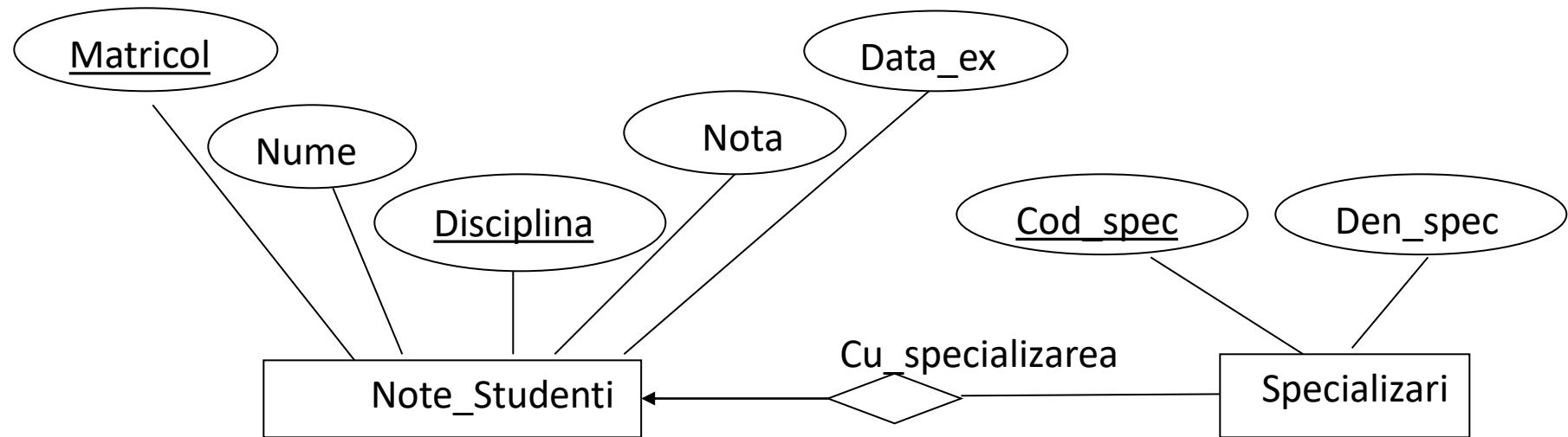


Figura 2. Diagrama entitate-asociere reproiectata



Anomalii in baze de date

- Prin transformarea diagramei initiale se obtin urmatoarele relatii:
NOTE_STUDENTI = Matricol, Nume, Disciplina, Nota,
Data_ex, Cod_spec
SPECIALIZARI = Cod_spec, Den_spec
- In acest caz tabelele vor contine urmatoarele date:

SPECIALIZARI

Cod_spec	Den_spec
1	CTI
2	AII



Anomalii in baze de date

NOTE_STUDENTI

Matricol	Nume	Disciplina	Nota	Data_ex	Cod_spec
1011	Ionescu Silvia	BD1	9	08.06.2020	1
1011	Ionescu Silvia	BD2	10	10.02.2021	1
1022	Popescu Daniel	PM	8	11.02.2021	2
1022	Popescu Daniel	SO	7	03.06.2020	2
1033	Popa Cornel	BD1	8	08.06.2020	1
1033	Popa Cornel	BD2	8	10.02.2021	1

- Procesul de “spargere” a unei tabele, care are o structura incorecta, in doua sau mai multe tabele se numeste descompunerea schemei de relatii si va fi prezentat intr-un alt capitol.



Anomalii in baze de date

Existenta anomalilor de proiectare poate crea o serie de probleme, cum ar fi:

- Cresterea costurilor prin cresterea spatiului de stocare a datelor;
- Aparitia inconsistentei datelor(valori diferite ale datelor pentru acelasi identificator);
- Nerespectarea standardelor pentru baze de date;
- Imposibilitatea aplicarii constrangerilor de integritate;
- Extragerea de informatii eronate in urma prelucrarii datelor.

Atunci cand se proiecteaza o baza de date trebuie facuta o analiza atenta pentru a vedea cum trebuie grupate atributele in relatii cu scopul de a minimiza redundanta datelor si implicit spatiul fizic alocat.



Independenta datelor

- O baza de date poate fi accesata de mai multe aplicatii, sau module ale aceleiasi aplicatii, in mod concurrent. In acest caz se pune problema in ce mod sunt afectate aplicatiile de modificarile efectuate in structura bazei de date. In mod normal, modificarile majore de structura(nivel inferior) implica si modificari la nivelul aplicatiei(nivel superior).
- Acest aspect tine de independenta datelor si poate fi vazuta sub doua aspecte:
 - independenta logica;
 - independenta fizica.



Independenta datelor

- **Independenta logica** a datelor se refera la imunitatea schemelor externe fata de modificarile efectuate in schema conceptuala, cum ar fi:
 - adaugarea de entitati sau atribute noi;
 - modificarea structurii entitatilor;
 - stergerea anumitor entitati sau atribute;
 - adaugarea, modificarea sau stergerea de obiecte in baza de date(tabele,view-uri,proceduri,functii, etc.);
 - activarea sau dezactivarea unor constrangeri.
- Cu alte cuvinte, independenta logica a datelor permite sa se faca modificari in structura bazei de date fara a implica rescrierea programelor aplicatiei.



Independenta datelor

- **Independenta fizica** de date se refera la imunitatea schemei conceptuale fata de modificarile efectuate in schema interna, cum ar fi:
 - reorganizarea fisierelor de date;
 - introducerea de noi dispozitive de stocare;
 - mutarea bazei de date de pe un server pe altul;
 - modificari in configurarea retelei, in cazul bazelor de date distribuite;
 - replicarea datelor din/pe mai multe servere;
 - utilizarea serverelor redundante pentru o securitate sporita a datelor.



Independenta datelor

- Independenta datelor, de cele mai multe ori, este o problema foarte grea chiar si in cazul sistemelor de ultima generatie. Limbajele relationale au adus imbunatatiri majore in acest sens insa complexitatea mare a aplicatiilor face ca problema independentei sa fie un concept care trebuie in continuare studiat si imbunatatit.
- Instrumentele moderne de proiectare software ofera facilitatea de modificare in cascada a obiectelor relate, atunci cand unul dintre obiecte sufera alterari de structura.



Independenta datelor

- În cazul instrumentelor CASE este posibil, uneori, ca o simplă recompilare sau o nouă generare a aplicatiei să preia toate modificările de structură intervenite la nivelul bazei de date. Sunt însă puține astfel de instrumente și de regulă rezolvă parțial problema, în sensul că trebuie făcute și intervenții manuale ale dezvoltatorilor. Pot fi și situații când o aplicatie nu este afectată de modificarea bazei de date, de exemplu dacă se adaugă entități sau atribută noi, funcționarea aplicatiei nu este afectată. Însă aceste modificări sunt făcute, de regulă, cu scopul de a îmbunătăți sau adăuga noi funcționalități în aplicatie și în acest caz trebuie intervenit și la nivel extern.



Independenta datelor

- In cazul cererilor de interogare sunt situatii cand nu trebuie rescrise, de exemplu daca se adauga noi entitati sau se modifica relatiile dintre entitati fara afectarea datelor existente in baza de date.
- Acelasi lucru se intampla si in cazul vederilor(*view*) create pe una sau mai multe entitati, daca atributele initiale nu sunt alterate nu trebuie recreata vederea, dar si aici trebuie studiate cu atentie implicatiile care pot sa apară.



Capitolul 7

Descompunerea schemelor de relatie



Descompunerea schemelor de relatie

Asa cum s-a mentionat anterior, in cazul in care o relatie din baza de date nu este intr-o forma normala adecvata (FN3, FNBC) pot sa apară diverse anomalii. Solutia este inlocuirea relatiei respective cu doua sau mai multe relatii care sa contina aceleasi informatii dar care, fiecare in parte, sa fie in forma normala dorita de proiectant.

- **Definitia descompunerii unei scheme de relatie**

Procesul prin care se divide o relatie in mai multe relatii se numeste ***descompunerea unei scheme de relatie***.

Formal, putem defini acest concept astfel:



Descompunerea schemelor de relatie

- **Definitie:** Fie R o schema de relatie, $R = A_1 A_2 \dots A_m$. Se spune ca $\rho = (R_1, R_2, \dots, R_n)$ este o **descompunere** a lui R daca si numai daca

$$R = R_1 \cup R_2 \cup \dots \cup R_n$$

- Schemele R_1, R_2, \dots, R_n contin deci atribute din R, fiecare atribut A_i al schemei initiale trebuind sa se regaseasca in cel putin una dintre ele.
- Nu este necesar ca schemele sa fie disjuncte, in practica ele au de multe ori atribute comune.



Descompunerea schemelor de relatie

Exemple:

In exemplele de mai jos sunt prezentate cateva descompuneri valide ale unor scheme de relatii (unele sunt insa incorecte din punct de vedere al pastrarii datelor si/sau dependentelor initiale):

- 1) Fie relatia $R = ABCDE$ avand

$$F = \{ A \rightarrow B, B \rightarrow A, A \rightarrow C, D \rightarrow E \}.$$

Putem avea descompuneri ca:

$$\rho_1 = (ABC, DE)$$

$$\rho_2 = (ABCD, DE)$$

$$\rho_3 = (AB, CD, DE)$$



Descompunerea schemelor de relatie

2) Fie relatia

Produse = IdP, NumeP, Cant, IdF, NumeF, AdresaF
avand dependentele functionale:

$$F = \{ IdP \rightarrow NumeP, IdP \rightarrow Cant, IdP \rightarrow IdF, \\ IdF \rightarrow NumeF, IdF \rightarrow AdresaF \}$$

- Putem avea mai multe descompuneri, printre care:

$$\rho_1 = ((IdP, NumeP, Cant, IdF); (NumeF, AdresaF))$$

$$\rho_2 = ((IdP, NumeP, Cant, IdF); (IdF, NumeF, AdresaF))$$

$$\rho_3 = ((IdP, NumeP); (Cant, IdF); (NumeF, AdresaF))$$



Descompunerea schemelor de relatie

- Descompunerea actioneaza deci la nivelul ***schemei*** relatiei. Ce se intampla insa cu ***continutul*** acesteia in cazul unei descompuneri?
- Fiecare relatie rezultata va mosteni o parte dintr-datele relatiei descompuse si anume proiectia acesteia pe multimea de atributi a relatiei rezultata din descompunere.
- Sa consideram o instanta **r** a relatiei din schema **R** (instanta unei relatii este o incarcare cu date corecte a acesteia). Atunci instantele pentru relatiile din descompunerea p sunt:

$$r_i = \pi_{R_i}(r)$$



Descompunerea schemelor de relatie

Exemplu:

Fie relatia PRODUSE de mai jos:

IdP	NumeP	Cant	IdF	NumeF	AdresaF
101	Imprimanta laser	30	20	Xerox	Str. D. Danielopolu, nr. 4-6, Sector 1, Bucuresti
105	Calculator PC	20	23	IBM	Bd. D.Cantemir, nr.1, Sector 3, Bucuresti
124	Copiator	10	20	Xerox	Str. D. Danielopolu, nr. 4-6, Sector 1, Bucuresti



Descompunerea schemelor de relatie

1. In cazul descompunerii

$\rho_1 = ((\text{IdP}, \text{NumeP}, \text{Cant}, \text{IdF}); (\text{NumeF}, \text{AdresaF}))$
obtinem:

r_1

IdP	NumeP	Cant	IdF
101	Imprimanta laser	30	20
105	Calculator PC	20	23
124	Copiator	10	20

r_2

NumeF	AdresaF
Xerox	Str. D. Danielopolu, nr. 4-6, Sector 1, Bucuresti
IBM	Bd. D.Cantemir, nr.1, Sector 3, Bucuresti



Descompunerea schemelor de relatie

2. In cazul descompunerii

$\rho_2 = ((\text{IdP}, \text{NumeP}, \text{Cant}, \text{IdF}); (\text{IdF}, \text{NumeF}, \text{AdresaF}))$

obtinem :

r_1

IdP	NumeP	Cant	IdF
101	Imprimanta laser	30	20
105	Calculator PC	20	23
124	Copiator	10	20

r_2

IdF	NumeF	AdresaF
20	Xerox	Str. D. Danielopolu, nr. 4-6, Sector 1, Bucuresti
23	IBM	Bd. D.Cantemir, nr.1, Sector 3, Bucuresti



Descompunerea schemelor de relatie

3. In cazul descompunerii

$\rho_3 = ((\text{IdP}, \text{NumeP}); (\text{Cant}, \text{IdF}); (\text{NumeF}, \text{AdresaF}))$
obtinem:

r_1

IdP	NumeP
101	Imprimanta laser
105	Calculator PC
124	Copiator

r_2

Cant	IdF
30	20
20	23
10	20

r_3

NumeF	AdresaF
Xerox	Str. D. Danielopolu, nr. 4-6, Sector 1, Bucuresti
IBM	Bd. D.Cantemir, nr.1, Sector 3, Bucuresti



Descompunerea schemelor de relatie

- Observam din aceste exemple ca in cazul primei si ultimei descompuneri nu putem reconstrui prin join, sau alti operatori relationali, relatia initiala. In cazul in care descompunerea nu s-a facut corect putem pierde:
 - datele relatiei initiale;
 - dependentele functionale ale relatiei initiale.
- In paragrafele urmatoare sunt prezentati algoritmi prin care putem detecta daca prin descompunere se pierd date sau dependente.



Descompunere cu join fara pierdere de date

➤ Descompunere cu join fara pierdere de date (j.f.p)

Conditia pentru a nu se pierde date prin descompunere este ca relatia initiala sa poata fi reconstruita in totalitate prin join-ul natural al relatiilor rezultante, fara tupluri in minus sau in plus.

Formal, definitia este urmatoarea:

- **Definitie:** Fie R o schema de relatie, F multimea de dependente functionale asociata si o descompunere $\rho = (R_1, R_2, \dots, R_n)$ a lui R. Se spune ca ρ este o descompunere ***cu join fara pierderi in raport cu F*** (prescurtat j.f.p.) daca si numai daca pentru orice instanta r a lui R, care satisface dependentele F, avem:

$$r_1 \bowtie r_2 \bowtie \dots \bowtie r_n = r \quad \text{unde} \quad r_i = \pi_{R_i}(r)$$



Descompunere cu join fara pierdere de date

- În exemplul de la paragraful anterior doar descompunerea $\rho_2 = ((\text{IdP}, \text{NumeP}, \text{Cant}, \text{IdF}); (\text{IdF}, \text{NumeF}, \text{AdresaF}))$ are proprietatea j.f.p, în cazul celorlalte, din cauza inexistentei coloanelor comune, joinul natural nu se poate efectua.
- Faptul că o descompunere are proprietatea j.f.p se poate testa pornind doar de la lista atributelor relației initiale, lista atributelor relațiilor din descompunere și a multimii de dependente funcționale asociată folosind algoritmul următor:



Algoritm de testare a proprietatii j.f.p.

- **Algoritm de testare a proprietatii de j.f.p. pentru o descompunere**
- **Intrare:** Schema de relatie $R = A_1 A_2 \dots A_m$, multimea de dependente functionale F si o descompunere $\rho = (R_1, R_2, \dots, R_n)$.
- **Iesire:** Decizia daca ρ are sau nu proprietatea j.f.p.
- **Metoda:**
 - ✓ Se construieste un tabel avand **n** linii si **m** coloane.
Liniile sunt etichetate cu elementele descompunerii ρ iar coloanele cu atributele relatiei R . Elementul (i,j) al tabelului va fi egal cu a_i daca $A_j \in R_i$ sau b_{ij} in caz contrar.



Algoritm de testare a proprietatii j.f.p.

- ✓ Se parcurg dependentele $X \rightarrow Y$ din F. Daca doua (sau mai multe) linii din tabel au aceleasi simboluri pe coloanele X aceste linii se egaleaza si pe coloanele din Y astfel:
 - Daca pe o coloana din Y apare un a_j atunci toate elementele de pe acea coloana din liniile respective devin a_j .
 - Daca pe o coloana din Y nu apare niciun a_j atunci se alege unul dintre elementele de tip b_{ij} si toate elementele de pe acea coloana din liniile respective devin egale cu acel b_{ij} .



Algoritm de testare a proprietatii j.f.p.

- ✓ Procesul se opreste:
 - Fie cand s-a obtinut o linie in tabel care contine doar **a-uri**, caz in care descompunerea **p are proprietatea j.f.p.**
 - Fie cand la o parcurgere a dependentelor nu mai apar schimbari in tabel si nu s-a obtinut o linie doar cu **a-uri**. In acest caz descompunerea **p nu are proprietatea j.f.p.**

In literatura de specialitate se poate gasi demonstratia faptului ca acest algoritm determina corect daca o descompunere are proprietatea j.f.p.



Algoritm de testare a proprietatii j.f.p.

Exemple:

- 1) Fie $R = ABCDE$, $F = \{ A \rightarrow B, A \rightarrow C, A \rightarrow D, D \rightarrow E \}$ si o descompunere a lui R , $\rho = (ABCD, DE)$
Construim tabelul aplicand algoritmul:

	A	B	C	D	E
ABCD	a1	a2	a3	a4	b15 a5
DE	b21	b22	b23	a4	a5

La prima parcurgere, pentru dependentele $A \rightarrow B$, $A \rightarrow C$, $A \rightarrow D$ nu gasim doua linii cu aceleasi valori pe coloana A dar pentru dependenta $D \rightarrow E$ cele doua linii sunt egale pe coloana D (simbolul a4). Le egalam si pe coloana E: cum pe aceasta coloana exista a5 rezulta ca b15 devine egal cu a5. S-a obtinut o linie numai cu **a**-uri, deci descompunerea are proprietatea de join fara pierderi.



Algoritm de testare a proprietatii j.f.p.

2) Fie relatia $R = ABCDE$ si $F = \{ A \rightarrow B, AC \rightarrow D, D \rightarrow E \}$ si descompunerea $\rho = (AB, BC, CDE)$. Tabelul este urmatorul:

	A	B	C	D	E
AB	a1	a2	b13	b14	b15
BC	b21	a2	a3	b24	b25
CDE	b31	b32	a3	a4	a5

La prima trecere nu apar modificari in tabel.

Procesul se opreste si ρ nu are proprietatea j.f.p.



Algoritm de testare a proprietatii j.f.p.

3) Fie $R = ABCDE$,

$$F = \{ C \rightarrow E (1), A \rightarrow C (2), B \rightarrow D (3), D \rightarrow E (4), E \rightarrow B (5) \}$$

(cu dependente numerotate intre paranteze) si $\rho = (BCE, AB, ACD)$

	A	B	C	D	E
BCE	b11	a2	a3	b14	a5
AB	a1	a2	b23 (2) a3	b24 (3) b14	b25 (4) a5
ACD	a1	b32 (5) a2	a3	a4	b35 (1) a5

Prima trecere:

- Din $C \rightarrow E$ rezulta $b35$ devine $a5$ (1)
- Din $A \rightarrow C$ rezulta $b23$ devine $a3$ (2)
- Din $B \rightarrow D$ rezulta $b24$ devine $b14$ (3)
- Din $D \rightarrow E$ rezulta $b25$ devine $a5$ (4)
- Din $E \rightarrow B$ rezulta $b32$ devine $a2$ (5)

Am obtinut o linie doar cu **a**-uri. Descompunerea este j.f.p.



Algoritm de testare a proprietatii j.f.p.

Observatie:

- In exemplele de mai sus a fost suficiente o singura trecere prin dependente. Exista insa situatii cand sunt necesare mai multe treceri pana cand procesul se opreste.
- In cazul in care descompunerea are numai doua elemente se poate testa daca are proprietatea de join fara pierderi si astfel:



Algoritm de testare a proprietatii j.f.p.

- **Definitie:** Fie R o schema de relatie, F multimea de dependente functionale asociata si $\rho = (R_1, R_2)$ o descompunere a sa.
Atunci ρ are proprietatea de **join fara pierderi** daca una dintre dependentele urmatoare se poate deduce din F :
 - $(R_1 \cap R_2) \rightarrow (R_1 - R_2)$ sau
 - $(R_1 \cap R_2) \rightarrow (R_2 - R_1)$
- Cu alte cuvinte, daca una dintre aceste dependente face parte din F (sau inchiderea F^+ aplicand axiome si reguli de inferenta) atunci descompunerea ρ este cu join fara pierderi.



Algoritm de testare a proprietatii j.f.p.

Exemplu: In prima exemplificare a aplicarii algoritmului de testare aveam o descompunere cu doua elemente: $R = ABCDE$, $F = \{ A \rightarrow B, A \rightarrow C, A \rightarrow D, D \rightarrow E \}$ si $\rho = (ABCD, DE)$. Avem :

- $R_1 = ABCD$, $R_2 = DE$,
- $(R_1 - R_2) = ABCD - DE = ABC$
- $(R_2 - R_1) = DE - ABCD = E$
- $(R_1 \cap R_2) = D$

Cele doua dependente sunt:

- $(R_1 \cap R_2) \rightarrow (R_1 - R_2)$ devine $D \rightarrow ABC$ (se pot obtine prin regula de inferenta – descompunere: $D \rightarrow A$, $D \rightarrow BC$ si $D \rightarrow C$, dar niciuna din aceste dependente nu face parte din F sau F^+).
- $(R_1 \cap R_2) \rightarrow (R_2 - R_1)$ devine $D \rightarrow E$. Cum $D \rightarrow E$ face parte din F rezulta ca ρ are proprietatea de join fara pierderi.



Descompuneri care pastreaza dependentele

➤ Descompuneri care pastreaza dependentele

- O a doua problema, in cazul descompunerii unei scheme de relatie R avand dependentele F in mai multe relatii R₁, R₂, ..., R_n, este aceea a pastrarii corelatiilor intre date, corelatii impuse de dependentele functionale din F.
- Fiecare relatie R_i va mosteni o multime de dependente data de proiectia multimii de dependente functionale F pe R_i:

$$F_i = \pi_{R_i}(F)$$



Descompuneri care pastreaza dependentele

Exemplu:

Fie relatia Produse = $\{IdP, NumeP, Cant, IdF, NumeF, AdresaF\}$ avand dependentele functionale:

$$F = \{ IdP \rightarrow NumeP, IdP \rightarrow Cant, IdP \rightarrow IdF, IdF \rightarrow NumeF, IdF \rightarrow AdresaF \}$$

- În cazul descompunerii $\rho_2 = (R1, R2)$ unde:
 $R1 = (IdP, NumeP, Cant, IdF)$
 $R2 = (IdF, NumeF, AdresaF)$

cele două relații mostenesc următoarele dependente:

$$F_{R1} = \pi_{R1}(F) = \{ IdP \rightarrow NumeP, IdP \rightarrow Cant, IdP \rightarrow IdF \}$$

$$F_{R2} = \pi_{R2}(F) = \{ IdF \rightarrow NumeF, IdF \rightarrow AdresaF \}$$



Descompuneri care pastreaza dependentele

- Dupa cum se observa toate dependentele relatiei initiale sunt pastrate fie in F_{R1} , fie in F_{R2} . Exista insa si cazuri in care unele dependente din F nu mai pot fi regasite in multimile de dependente asociate schemelor din descompunere si nu se pot deduce din acestea.
- In primul caz se spune ca descompunerea pastreaza dependentele iar in al doilea ca descompunerea nu pastreaza dependentele.
- O definitie formală a acestui concept este urmatoarea:



Descompuneri care pastreaza dependentele

- **Definitie:** Fie R o schema de relatie, F multimea de dependente functionale asociata, o descompunere $\rho = (R_1, R_2, \dots, R_n)$ a lui R si $F_i = \pi_{R_i}(F)$ proiectia multimii de dependente functionale(multimile de dependente functionale ale elementelor descompunerii).

Se spune ca ρ ***pastreaza dependentele*** din F daca si numai daca orice dependenta din F poate fi dedusa din $\cup_{i=1..n} (F_i)$.

- Rezulta ca o descompunere pastreaza dependentele daca si numai daca:

$$F \subseteq (\cup_{i=1..n} (F_i))^+$$



Descompuneri care pastreaza dependentele

- Din pacate, atât proiecția unei multimi de dependente cat și incluziunea de mai sus implica un calcul de inchidere a unei multimi de dependente (F , respectiv reuniunea multimilor F_i).
- Există și în acest caz un algoritm pentru a testa dacă o dependență este pastrată după descompunere, sau nu este, fără a fi necesar calculul efectiv al multimilor F_i^+ .



Algoritm de testare a pastrarii dependentelor

➤ Algoritm de testare a pastrarii dependentelor

- **Intrare:** O schema de relatie R, multimea de dependente functionale asociata F si o descompunere $\rho = (R_1, R_2, \dots, R_n)$.
- **Iesire:** Decizia daca ρ pastreaza sau nu dependentele.



Algoritm de testare a pastrarii dependintelor

- **Metoda:** Pentru fiecare dependenta $X \rightarrow Y$ din F se procedeaza astfel:
 - ✓ Se porneste cu o multime de atribute $Z = X$;
 - ✓ Se parcurg repetat elementele descompunerii ρ . Pentru fiecare R_i se calculeaza o noua valoare a lui Z astfel:
$$Z = Z \cup ((Z \cap R_i)^+ \cap R_i);$$
 - ✓ Procesul se opreste in momentul cand Z ramane neschimbat la o parcurgere a elementelor R_i . Daca $Y \subseteq Z$ atunci dependenta $X \rightarrow Y$ este pastrata, altfel nu e pastrata.
 - ✓ Daca toate dependentele din F sunt pastrate inseamna ca ρ **pastreaza dependentele din F .**



Algoritm de testare a pastrarii dependintelor

Exemple:

1) Fie $R = ABCDE$, multimea de dependente functionale $F = \{ C \rightarrow E, A \rightarrow C, B \rightarrow D, D \rightarrow E, E \rightarrow B \}$ si descompunerea $\rho = (BCE, AB, ACD)$.

- Se observa ca dependentele $C \rightarrow E$, $A \rightarrow C$ si $E \rightarrow B$ sunt pastrate: ele apartin proiectiei lui F pe BCE (prima si ultima) si ACD (a doua).
- Raman de testat dependentele $B \rightarrow D$ si $D \rightarrow E$.
Sa aplicam algoritmul pentru $B \rightarrow D$:



Algoritm de testare a pastrarii dependentelor

- ✓ Initial $Z = B$
- 1) Prima trecerea prin elementele lui ρ :
 - Pentru BCE:
$$Z = B \cup ((B \cap BCE)^+ \cap BCE) = B \cup (BDE \cap BCE) = BE$$

deoarece $(B \cap BCE)^+ = B^+$ iar calculul lui B^+ este urmatorul:

- $X^{(0)} = \{B\}$
- $X^{(1)} = \{B\} \cup \{D\} = BD$
- $X^{(2)} = \{BD\} \cup \{DE\} = BDE$
- $X^{(3)} = \{BDE\} \cup \{DEB\} = BDE$



Algoritm de testare a pastrarii dependintelor

- Pentru AB:

$$Z = BE \cup ((BE \cap AB)^+ \cap AB) = BE \cup (BDE \cap AB) = BE$$

- Pentru ACD:

$$Z = BE \cup ((BE \cap ACD)^+ \cap AB) = BE \cup \emptyset = BE$$

- ✓ Deoarece $Z=BE$ ramane neschimbat dupa efectuarea trecerilor, procesul se opreste.
- ✓ Cum $\{D\} \not\subset BE$ rezulta ca dependenta $B \rightarrow D$ nu este pastrata, deci ρ nu pastreaza dependentele.



Algoritm de testare a pastrarii dependintelor

2) Fie schema de relatie $R = ABCD$, $F = \{ A \rightarrow B, A \rightarrow C, C \rightarrow D, D \rightarrow A \}$ si o descompunere $\rho = (ABC, CD)$. Trebuie sa testam daca $D \rightarrow A$ este pastrata (celelalte dependente se regasesc direct in proiectiile lui F pe elementele descompunerii).

- ✓ Initial $Z = D$
- 1) Prima trecere prin elementele lui ρ :
 - Pentru ABC : $Z = D \cup ((D \cap ABC)^+ \cap ABC) = D \cup \emptyset = D$
 - Pentru CD : $Z = D \cup ((D \cap CD)^+ \cap CD) = D \cup (ABCD \cap CD) = CD$
- 2) A doua trecere prin elementele lui ρ :
 - Pentru ABC : $Z = CD \cup ((CD \cap ABC)^+ \cap ABC) = CD \cup (ABCD \cap ABC) = ABCD$. Stop.
 - ✓ Am obtinut ca $A \subseteq Z$, deci dependenta $D \rightarrow A$ este pastrata, rezulta ca ρ pastreaza dependentele.



Algoritmi de descompunere a schemelor de relatie

➤ Algoritmi de descompunere

Algoritmii de testare a pastrarii dependentelor si a joinului fara pierderi pot fi aplicati atunci cand descompunerea unei scheme de relatie se face “manual”, pe baza experientei pe care o are proiectantul bazei de date.

- Exista insa si niste algoritmi simpli care, pornind de la o schema de relatie si multimea de dependente functionale asociata , ne duc direct la o descompunere care este in FN3 sau FNBC si, in plus, au proprietatea de join fara pierderi (nu se pierd date prin descompunere) si se pastreaza dependentele functionale.



Algoritmi de descompunere a schemelor de relatie in FN3

■ Algoritm de descompunere in FN3 cu pastrarea dependentelor

Fie R o schema de relatie si F multimea de dependente functionale asociata, cu $F = \{ X_1 \rightarrow Y_1, X_2 \rightarrow Y_2, \dots X_n \rightarrow Y_n \}$
Atunci descompunerea $\rho = (X_1Y_1, X_2Y_2, \dots X_nY_n)$ este o descompunere in FN3 cu pastrarea dependentelor.

Se observa din definitia de mai sus a descompunerii ρ ca:

- Toate dependentele sunt pastrate: dependenta $X_i \rightarrow Y_i$ este in proiectia lui F pe X_iY_i
- Pentru a minimiza numarul de elemente din descompunere se aplica regula reuniunii: daca avem mai multe dependente care au aceeasi parte stanga le reunim intr-una singura.
- Daca in descompunere exista doua elemente X_iY_i si X_jY_j astfel incat $X_iY_i \subseteq X_jY_j$ atunci X_iY_i se elimina.
- **Observatie:** In literatura de specialitate exista demonstratia faptului ca fiecare schema din descompunerea ρ este in FN3.



Algoritmi de descompunere a schemelor de relatie in FN3

- Exemple:
- 1) $R = ABCDE$, $F = \{ A \rightarrow B, A \rightarrow C, A \rightarrow D, D \rightarrow E \}$. Rescriem prin reuniune multimea de dependente functionale: $F = \{ A \rightarrow BCD, D \rightarrow E \}$. Rezulta din algoritm ca descompunerea $\rho = (ABCD, DE)$ este in FN3 cu pastrarea dependentelor.
- 2) Fie relata Produse = IdP, NumeP, Cant, IdF, NumeF, AdresaF avand dependentele functionale:
 $F = \{ IdP \rightarrow NumeP, IdP \rightarrow Cant, IdP \rightarrow IdF, IdF \rightarrow NumeF, IdF \rightarrow AdresaF \}$
- Rescriem multimea de dependente. Raman numai doua dependente:
 $F = \{ IdP \rightarrow NumeP, Cant, IdF; IdF \rightarrow NumeF, AdresaF \}$
- Descompunerea in FN3 cu pastrarea dependentelor va fi:
 $\rho = ((IdP, NumeP, Cant, IdF), (IdF, NumeF, AdresaF))$



Algoritmi de descompunere a schemelor de relatie in FN3

- **Algoritm de descompunere in FN3 cu pastrarea dependentelor si join fara pierderi**

Daca la descompunerea obtinuta prin algoritmul anterior adaugam o cheie a relatiei (ca element al descompunerii) vom obtine o descompunere care are atat proprietatea de join fara pierderi cat si pe cea a pastrarii dependentelor.

Formal putem scrie algoritmul astfel:

- **Definitie:** Fie R o schema de relatie si F multimea de dependente functionale asociata, cu

$F = \{ X_1 \rightarrow Y_1, X_2 \rightarrow Y_2, \dots X_n \rightarrow Y_n \}$ si o cheie X pentru R, cu $X \not\subset X_i Y_i$. Atunci descompunerea

$\rho = (X, X_1 Y_1, X_2 Y_2, \dots X_n Y_n)$ este o descompunere in FN3 cu pastrarea dependentelor si join fara pierderi.



Algoritmi de descompunere a schemelor de relatie in FN3

- Pastrarea dependentelor este evidentă, ca mai sus. Demonstratia faptului că descompunerea are și proprietatea de join fără pierderi se găsește în literatura de specialitate.
- **Observatie:**

Daca vreunul dintre elementele de forma X_iY_i contine deja o cheie a lui R atunci nu este necesara adaugarea unui element suplimentar in descompunere. Deci, daca o cheie este deja inclusa intr-o descompunere atunci nu trebuie adaugata ca element suplimentar.



Algoritmi de descompunere a schemelor de relatie in FN3

Exemple:

- 1) Pentru relatiile din exemplele de mai sus descompunerea ramane aceeasi, deoarece:
 - In cazul relatiei $R = ABCDE$ cu descompunerea $\rho = (ABCD, DE)$, cheia este A, deja inclusa in ABCD, deci ρ este in FN3 cu pastrarea dependentelor si join fara pierderi.
 - In cazul relatiei PRODUSE cu descompunerea $\rho = ((IdP, NumeP, Cant, IdF), (IdF, NumeF, AdresaF))$ cheia este IdP, inclusa de asemenea intr-unul dintre elementele descompunerii, deci ρ este in FN3 cu pastrarea dependentelor si join fara pierderi.



Algoritmi de descompunere a schemelor de relatie in FN3

2) Fie $R = ABCDE$, $F = \{ A \rightarrow B, B \rightarrow A, A \rightarrow C, D \rightarrow E \}$.
Cheile relatiei sunt AD si BD.

Rescriem multimea de dependente:

$$F = \{ A \rightarrow BC, B \rightarrow A, D \rightarrow E \}.$$

- Rezulta descompunerea cu pastrarea dependentelor:
 $\rho = (ABC, AB, DE)$. Dar AB este inclus in ABC si rezulta in final $\rho = (ABC, DE)$.
- Cum elementele descompunerii nu contin vreo cheie a lui R, o adaugam. Obtinem in final descompunerile $\rho_1 = (AD, ABC, DE)$ si $\rho_2 = (BD, ABC, DE)$ in FN3 care pastreaza dependentele si au proprietatea j.f.p.



Algoritmi de descompunere a schemelor de relatie in FNBC

- **Algoritm de descompunere in FNBC cu join fara pierderi**

Fie R o schema de relatie si F multimea de dependente functionale asociata, F in forma canonica: $F = \{ X_1 \rightarrow A_1, X_2 \rightarrow A_2, \dots X_n \rightarrow A_n \}$. Putem calcula descompunerea in FNBC cu join fara pierderi iterativ:

- ✓ Initial $\rho = (R)$;
- ✓ La fiecare pas se alege o schema T care contine o dependenta de forma $X \rightarrow A$ care violeaza conditiile de FNBC. Schema respectiva este inlocuita in ρ prin T_1 si T_2 unde $T_1 = XA$ si $T_2 = T - \{A\}$;
- ✓ Procesul se opreste cand in ρ nu mai exista elemente care nu sunt in FNBC.



Algoritmi de descompunere a schemelor de relatie in FNBC

Exemple:

1) Fie relatia $R = ABCD$ cu $F = \{ AB \rightarrow C, AB \rightarrow D, D \rightarrow A \}$. Cheia relatiei este AB. Relatia este in FN3 dar nu este in FNBC din cauza dependentei $D \rightarrow A$ care nu are in partea stanga o supercheie a lui R.

- ✓ Initial: $\rho = (R) = (ABCD)$;
- ✓ Alegem dependenta $D \rightarrow A$ care violeaza conditia de FNBC;
- Inlocuim $T = ABCD$ cu $T_1 = DA$ si $T_2 = ABCD - A = BCD$;
- T_1 mosteneste de la T dependenta $D \rightarrow A$, cheia va fi D si T_1 e in FNBC;
- T_2 mosteneste de la T dependenta $\{ BD \rightarrow C$ obtinuta prin augmentarea $DB \rightarrow AB$ si $AB \rightarrow C \}$. Cheia va fi BD si T_2 e in FNBC;
- ✓ Rezulta ca descompunerea in FNBC cu join fara pierderi este $\rho = (AD, BCD)$.



Algoritmi de descompunere a schemelor de relatie in FNBC

Observatii:

- Dependenta mostenita de T_2 este din F^+ . Ea se deduce astfel: Din $D \rightarrow A$ prin augmentare cu B obtinem $DB \rightarrow AB$ si impreuna cu dependenta $AB \rightarrow C$, prin tranzitivitate obtinem $DB \rightarrow C$.
- Analog din $AB \rightarrow D$ se deduce $DB \rightarrow D$ dar aceasta este o dependenta triviala (partea dreapta e inclusa in cea stanga).
- In multe cazuri este nevoie de mai multe iteratii, relatiile de tip T_2 (egale in algoritm cu $T - A$) nefiind uneori in FNBC. Ele se descompun din nou in acelasi fel.



Algoritmi de descompunere a schemelor de relatie in FNBC

2) Consideram R =(id_dis, id_stud, sesiune, data_ex, nota, id_prof) un catalog de note si F=(id_dis->id_prof, id_stud->id_dis, id_stud->nota, id_stud->id_dis, id_dis->data_ex, id_dis->sesiune, sesiune->data_ex) cu cheia=(id_dis, id_stud).

Obs: Consideram ca un student are o singura nota la o disciplina(nota finala) si o disciplina este predata de catre un singur profesor.

- Aplicam algoritmul de descompunere:

a) Alegem id_dis->id_prof care violeaza conditia FNBC (id_dis nu este supercheie) si rezulta R1=(id_dis, id_prof) , R2=(id_dis, id_stud, sesiune, data_ex, nota)

- R1 cheia=id_dis si id_dis->id_prof , deci este FNBC.
- R2 cheia=(id_dis,id_stud), (id_dis,id_stud)->(sesiune, data_ex, nota),deci este FNBC.

Putem sa ne oprim aici deoarece am obtinut o descompunere FNBC.

Obs: Daca continuam algoritmul putem obtine alte descompuneri tot in FNBC, dar se urmareste ca descompunerea sa contine cat mai putine relatii.

De exemplu, daca continuam cu id_dis->data_ex obtinem descompunerea:

R1=(id_dis, id_prof) , R2= (id_dis, data_ex), R3=(id_dis, id_stud, sesiune, nota) in care relatiile sunt tot in FNBC.



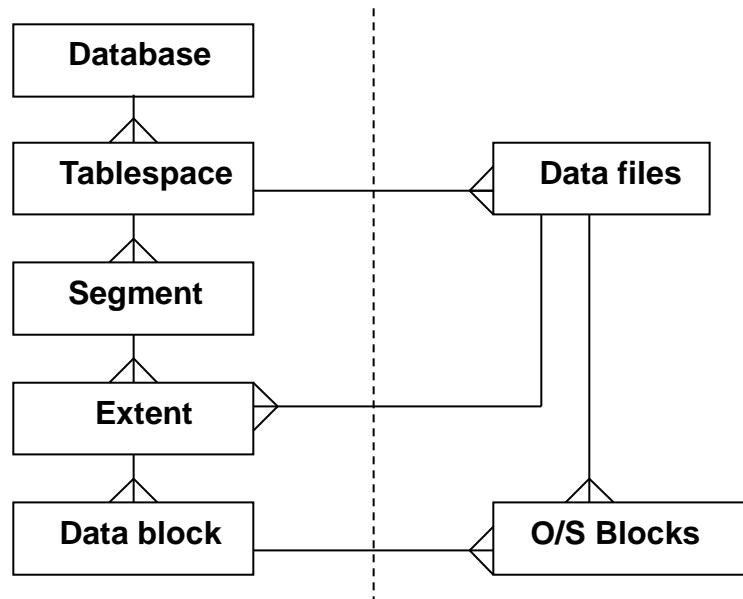
Limbajul SQL

Alexandru Boicea Curs: Baze de date



Structura bazei de date

- Structura logica si fizica a unei **Baze de Date** (*database*) relationale:



- Tablespace - este spatiul logic in care se creeaza obiectele (tabele, view-uri, indecsi, proceduri, etc.). O baza de date poate avea mai multe tablespace-uri, iar un tablespace poate avea alocat fizic mai multe fisiere de date (*data files*) .



Structura bazei de date

- Segmentul(*segment*) – reprezinta un spatiu logic de stocare alocat unui obiect intr-un tablespace. Pot fi de mai multe tipuri de segmente: permanente, temporare, index, rollback, etc.).
- Extensia(*extent*) - reprezinta o extensie logica a spatiului de stocare reprezentata printr-un numar continuu de blocuri.
- Blocul de date(*data block*) - reprezinta cea mai mica unitate logica de stocare.
- Fisiere de date(*data files*) - sunt fisierele organizate fizic pe un dispozitiv de stocare. Fisierele de date stocheaza fizic datele in baza de date.
- O/S block - reprezinta cea mai mica unitate fizica de organizare a datelor intr-o baza de date.



Sistemul de Gestiune a Bazei de Date

- Controlul asupra bazei de date este gestionat de catre **Sistemul de Gestiune a Bazei de Date**(SGBD) si verifica respectarea unor reguli:
 - O baza de date relationala apare ca o colectie de tabele definite de catre utilizator;
 - Utilizatorul nu controleaza felul cum este organizata fizic informatia;
 - controlul asupra fisierelor de date este gestionat exclusiv de catre sistemul de gestiune;
 - Utilizatorul poate defini anumiti parametri de sistem pentru optimizarea aplicatiilor sau pentru diferite setari;
 - Accesul la baza de date este gestionat exclusiv de catre sistem prin executarea de comenzi specifice;
 - Rularea aplicatiilor, atat pe server cat si pe masina client, este gestionata exclusiv de catre sistemul de gestiune.



Limbajul SQL

- Un sistem de gestiune a bazei de date necesita un limbaj de interogare pentru a permite utilizatorului sa acceseze datele.
- **Limbajul SQL (*Structured Query Language*)** este un limbaj de interogare structurat utilizat de majoritatea sistemelor de baze de date relationale.
- Cateva trasaturi caracteristice ale limbajului SQL:
 - Implementeaza setul standard de comenzi de manipulare a datelor(inserare, interogare, modificare, stergere);
 - Este un limbaj neprocedural care optimizeaza automat planul de executia a cererilor;
 - Cererile se executa secvential, linie cu linie, deci se prelucreaza o singura inregistrare dintr-o tabela la un moment dat .
 - Permite importul si exportul datelor;
 - Ofera suport pentru administrarea bazei de date.



Limbajul SQL

DB DB-Engines Ranking - popularit X + https://db-engines.com/en/ranking/relational+dbms

Complete ranking
Relational DBMS
Key-value stores
Document stores
Graph DBMS
Time Series DBMS
Object oriented DBMS
Search engines
RDF stores
Multivalue DBMS
Wide column stores
Native XML DBMS
Event Stores
Content stores
Navigational DBMS

Special reports

- Ranking by database model
- Open source vs. commercial

Featured Products

AllegroGraph
Graph Database Leader for AI Knowledge Graph Applications - The Most Secure Graph Database Available.

DB-Engines Ranking of Relational DBMS

The DB-Engines Ranking ranks database management systems according to their popularity. The ranking is updated monthly.

This is a partial list of the [complete ranking](#) showing only relational DBMS.

Read more about the [method](#) of calculating the scores.

include secondary database models

139 systems in ranking, January 2020

Rank			DBMS	Database Model	Score		
Jan 2020	Dec 2019	Jan 2019			Jan 2020	Dec 2019	Jan 2019
1.	1.	1.	Oracle +	Relational, Multi-model	1346.68	+0.29	+77.85
2.	2.	2.	MySQL +	Relational, Multi-model	1274.65	-1.01	+120.39
3.	3.	3.	Microsoft SQL Server +	Relational, Multi-model	1098.55	+2.35	+58.29
4.	4.	4.	PostgreSQL +	Relational, Multi-model	507.19	+3.82	+41.08
5.	5.	5.	IBM Db2 +	Relational, Multi-model	168.70	-2.65	-11.15
6.	6.	6.	Microsoft Access	Relational	128.58	-0.89	-13.04
7.	7.	7.	SQLite +	Relational	122.14	+1.78	-4.66
8.	8.	8.	MariaDB +	Relational, Multi-model	87.45	+0.66	+8.63
9.	9.	↑ 10.	Hive +	Relational	84.24	-1.81	+14.33
10.	10.	↓ 9.	Teradata +	Relational, Multi-model	78.29	-0.21	+2.10
11.	↑ 12.	11.	FileMaker	Relational	55.11	-0.03	-2.05
12.	↑ 13.	12.	SAP HANA +	Relational, Multi-model	54.69	+0.52	-1.95
13.	↓ 11.	13.	SAP Adaptive Server	Relational	54.59	-0.96	-0.45
14.	14.	14.	Microsoft Azure SQL Database	Relational, Multi-model	28.20	+0.32	+1.01
15.	↑ 16.	↑ 20.	Google BigQuery +	Relational	26.76	+1.25	+8.39
16.	↓ 15.	↓ 15.	Informix	Relational, Multi-model	25.14	-0.38	-1.63

trend chart

<https://db-engines.com/en/ranking/relational+dbms>



Limbajul SQL

- În acest capitol vom face o introducere în limbajul **Oracle SQL** utilizat pentru accesarea și administrarea unei baze de date Oracle. Comenzile și cererile SQL sunt folosite pentru :
 - Inserarea/extragerea/modificarea/stergerea randurilor într-o tabelă;
 - Crearea/modificarea/stergerea obiectelor din baza de date;
 - Controlul conexiunii și accesului la baza de date;
 - Prelucrarea datelor;
 - Relationarea datelor din mai multe tabele;
 - Formatarea datelor de ieșire;
 - Introducerea criteriilor de căutare și sortare a datelor;
 - Refacerea stării bazei de date la un moment anterior;
 - Importul, exportul și replicarea datelor.



Limbajul SQL

- ✓ Setul de comenzi standard SQL de manipulare a datelor DML (*Data Manipulation Language*) :
 - SELECT – folosita pentru extragerea datelor din baza de date;
 - INSERT – folosita pentru inserarea datelor in baza de date;
 - UPDATE – folosita pentru modificarea datelor in baza de date;
 - DELETE – folosita pentru stergerea inregistrarilor.
- ✓ Setul de comenzi standard SQL pentru definirea datelor DDL (*Data Definition Language*):
 - CREATE – folosita pentru crearea unui obiect (tabela, view, index, etc.) in baza de date ;
 - ALTER – folosita pentru modificarea structurii unui obiect din baza de date;
 - DROP – folosita pentru stergerea unui obiect din baza de date.



Limbajul SQL

- ✓ Setul de comenzi standard SQL pentru crearea/revocarea drepturilor de acces:
- GRANT – folosita pentru a grantifica drepturile de acces la un obiect din baza de date;
- REVOKE – folosita pentru revocarea drepturilor de acces.
- Acestea sunt numai o parte a comenziilor SQL(lista completa se gaseste in *Manualul de Referinta a Limbajului SQL*).
- Cateva reguli de scriere a comenziilor SQL :
 - Comenziile se pot edita pe una sau mai multe linii;
 - Clauzele sunt uʒual plasate pe linii separate, dar nu obligatoriu;
 - Cuvintele predefinite nu pot fi separate pe mai multe linii;
 - Comenziile nu sunt *case sensitive* (numai datele stocate in baza de date).



Operatori SQL

➤ Operatori de comparatie

Sunt operatori folositi pentru compararea valorilor coloanelor intre ele sau cu valori numerice si pot fi de doua feluri:

✓ Operatori logici

Operator	Semnificatie
=	egal cu
>	mai mare decit
>=	mai mare sau egal
<	mai mic decit
<=	mai mic sau egal



Operatori SQL

✓ Operatori SQL

Operator	Semnificatie
BETWEEN..AND...	intre doua valori(inclusiv)
IN(list)	compara cu o lista de valori
LIKE	compara cu un model de tip caracter
IS NULL	este o valoare nula



Operatori SQL

➤ Operatorii de negatie

Sunt operatori folositi pentru negarea valorilor coloanelor sau verificarea conditiilor de inegalitate si pot fi de doua feluri:

✓ Operatori logici

Operator	Semnificatie
-----	-----
!=	diferit de (VAX,UNIX,PC)
^=	diferit de (IBM)
<>	diferit de (toate OS)
NOT col_name =	diferit de
NOT col_name >	mai mic sau egal



Operatori SQL

✓ Operatori SQL

Operator	Semnificatie
-----	-----
NOT BETWEEN	nu se afla intre doua valori date
NOT IN	nu se afla intr-o lista data de valori
NOT LIKE	diferit de un sir
IS NOT NULL	nu este o valoare nula

- Folosind operatorul LIKE asociat cu simbolurile urmatoare, este posibil sa selectam randurile care se potrivesc cu un sir sau subsir de caractere :

<u>Simbol</u>	<u>Semnificatie</u>
%	orice secventa de mai multe caractere
_	un singur caracter



Operatori SQL

Observatii:

- Daca se compara o coloana sau expresie cu NULL, atunci operatorul de comparatie trebuie sa fie IS sau IS NOT. Daca se foloseste orice alt operator rezultatul va fi totdeauna FALSE (de exemplu expresia comision != NULL este intotdeauna falsa).
- Operatorii AND si OR pot fi utilizati pentru a compune expresii logice cu conditii multiple. Predicatul AND este adevarat numai daca ambele conditii sunt TRUE, iar predicatul OR este adevarat daca cel putin una dintre conditii este TRUE. Se pot combina AND sau OR in aceiasi expresie logica in clauza WHERE, iar in acest caz operatorii AND sunt evaluati primii si apoi operatorii OR (deci operatorul AND au o precedenta mai mare decat OR).



Operatori SQL

- Daca operatorii au precedenta egala, atunci ei se evaluateaza de la stanga la dreapta.
- Precedenta operatorilor logici este urmatoarea:
 1. Operatorii de comparatie si operatorii SQL au precedenta egala:
=, >= , <>, BETWEEN...AND, IN, LIKE, IS NULL.
 2. NOT - cand este folosit pentru a inversa rezultatul unei expresii logice (de exemplu SELECTWHERE not(sal>2000))
 3. AND
 4. OR.
- Pentru a fi siguri de ordinea de executie a doua operatii, se recomanda folosirea parantezelor rotunde pentru gruparea operatiilor.



Crearea unei tabele

➤ Comanda CREATE TABLE

- Este folosita pentru crearea unei tabele in baza de date. Aceasta comanda va fi prezentata in detaliu intr-un alt capitol.

Exemplu:

- Vom crea doua tabele pentru evidenta departamentelor si a angajatilor:

```
CREATE TABLE departamente  
( id_dep      number(2)  not null,  
  den_dep     varchar2(30),  
  telefon     varchar2(10) );
```

```
CREATE TABLE angajati  
( id_ang      number(4)  not null,  
  nume        varchar2(30),  
  functie    varchar2(20),  
  id_sef      number(4),  
  data_ang    date,  
  salariu    number(7,2),  
  comision   number(7,2),  
  id_dep      number(2) );
```



Inserarea datelor intr-o tabela

➤ Comanda INSERT

- Este folosita pentru inserarea datelor intr-o tabela si are urmatoarea sintaxa:

```
INSERT INTO [schema.] table_name[view_name][@dblink]
    [column 1, column 2, ....]
    VALUES (expr1, expr2. ....) subquery
```

unde :

- **schema** – este schema unde este creata tabela (specifica userul si baza de date);
- **table_name** – este numele tablei;
- **view_name** – este numele unui view creat pe o tabela;
- **column** – este numele coloanei;
- **expr** – reprezinta valoarea aferenta coloanei;
- **subquery** – este o subcerere care returneaza linii cu date din una sau mai multe tabele.



Inserarea datelor intr-o tabela

Exemple:

- Sa facem o inserare completa (toate coloanele au valori nenule) in tabelele create anterior. In acest caz nu trebuie sa specificam numele coloanelor dar valorile trebuie specificate in clauza VALUES in ordinea de creare a coloanelor in tabela.

```
SQL> INSERT INTO departamente VALUES (50, 'Proiectare Software',  
'0213262031');
```

```
SQL> INSERT INTO angajati VALUES (111, 'Popa Daniela', 'Inginer', 777, '1-  
OCT-2019', 750, 100, 10);
```

- Daca facem inserari numai in anumite coloane comanda arata astfel:

```
SQL> INSERT INTO angajati (id_ang, nume, functie) VALUES (777, 'Petrescu  
Florin', 'Contabil');
```

- Trebuie mentionat ca in acest caz trebuie specificate toate coloanele care fac parte dintr-o cheie primara sau sunt declarate NOT NULL la crearea tablei, altfel se va genera un cod de eroare.



Inserarea datelor intr-o tabela

- Urmatoarea comanda va genera o eroare deoarece nu se specifica valoare pentru coloana id_ang care este declarata not null:

```
SQL> INSERT INTO angajati (nume, functie, salariu) VALUES ('Tache  
Marius', 'Tehnician', 1300);
```

ERROR at line 1:

ORA-01400: cannot insert NULL into ("SCOTT"."ANGAJATI"."ID_ANG")

- Nu se accepta inregistrari cu valori care depasesc dimensiunea coloanei. Urmatoarea comanda va genera o eroare deoarece se specifica o valoare prea mare pentru id_dep:

```
SQL> INSERT INTO departamente VALUES (222, 'Contabilitate', '0213262032');  
*
```

ERROR at line 1:

ORA-01438: value larger than specified precision allowed for this column



Stergerea datelor dintr-o tabela

➤ Comanda DELETE

- Este folosita pentru stergerea liniilor dintr-o tabela sau view si are urmatoarea sintaxa:

DELETE FROM [schema.]

table_name[view_name][@dblink]

WHERE condition

(column1,column2,..) IN [NOT IN] subquery

unde :

- **schema** – este schema unde este creata tabela (specifica userul si baza de date);
- **table_name** – este numele tablei;
- **view_name** – este numele unui view creat pe o tabela;
- **condition** – conditia care trebuie indeplinita pentru liniile sterse;
- **column** – este numele coloanei;
- **subquery** – este o subcerere care returneaza linii cu date din una sau mai multe table.



Stergerea datelor dintr-o tabela

Exemple:

- Stergerea unui angajat din tabela angajati se face cu comanda:
SQL> DELETE FROM angajati WHERE nume='POPA DANIELA';
- Stergerea tuturor angajatilor care au venit in companie inainte de anul 1981 se face cu comanda:
- **SQL> DELETE FROM angajati WHERE data_ang < '1-JAN-1981';**
- Pentru a sterge toti angajatii care s-au angajat in luna Noiembrie, indiferent de an, folosim comanda:

SQL> DELETE FROM angajati WHERE data_ang LIKE '%NOV%';

- Stergerea angajatilor care nu au sef (id_sef este null) se face astfel:

SQL> DELETE FROM angajati WHERE id_sef is null;

- Pentru a sterge toate liniile din tabela angajati folosim comanda:

SQL> DELETE FROM angajati;

- Refacerea datelor sterse accidental se face cu comanda ROLLBACK:

SQL> ROLLBACK;



Modificarea datelor dintr-o tabela

➤ Comanda UPDATE

Este folosita pentru modificarea datelor in baza de date si are urmatoarea sintaxa:

UPDATE [schema.] table_name[view_name] [@dblink]

SET column=expr

column=subquery

(column1,column2,...) IN [NOT IN] subquery

WHERE condition

unde :

- **schema** – este schema unde este creata tabela (specifica userul si baza de date);
- **table_name** – este numele tablei;
- **column** – este numele coloanei;
- **condition** – conditia pentru modificarea liniilor;
- **subquery** – este o subcerere care returneaza linii cu date din una sau mai multe table.



Modificarea datelor dintr-o tabela

Exemple:

- Modificarea salariului si comisionului unui angajat se face astfel:

SQL> UPDATE angajati SET salariu=1200, comision=100 WHERE nume='IONESCU VICTOR';

SQL> UPDATE angajati SET salariu=1000 WHERE id_ang=7369;

- Modificarea tuturor salariilor prin indexare cu 10%:

SQL> UPDATE angajati SET salariu=salariu*1.1;

- Daca dorim sa crestem salariile doar pentru angajatii din departamentul 10 folosim comanda:

SQL> UPDATE angajati SET salariu=salariu*1.1 WHERE id_dep=10;

- Acordarea unui comision pentru angajatii veniti in companie in anul 1981 se face astfel:

SQL> UPDATE angajati SET comision=0.1*salariu WHERE data_ang>'1-JAN-1981' AND data_ang<'31-DEC-1981';

- In absenta clauzei WHERE toate liniile vor fi actualizate.



Modificarea datelor dintr-o tabela

- Cand se face actualizarea datelor intr-o tabela se verifica automat si constrangerile de integritate definite pe tabela respectiva, altfel comanda genereaza un cod de eroare si tranzactia esueaza.
- Situatiile in care pot sa apară erori sunt:
 - Noile valori fac duplicare de cheie primara sau unica;
 - Actualizarea valorii cu o valoare nula cand coloana este NOT NULL;
 - Valorile noi nu respecta o constrangere CHECK;
 - Valorile noi nu respecta o constrangere FOREIGN KEY;
 - Valorile vechi erau referite de alte tabele printr-o constrangere FOREIGN KEY;



Vizualizarea datelor dintr-o tabela

➤ Comanda SELECT

- Este folosita pentru vizualizarea datelor dintr-o tabela. Aceasta comanda va fi prezentata in detaliu intr-un alt capitol.

Exemple:

SQL> SELECT * FROM angajati;

SQL> SELECT nume, functie, salariu FROM angajati;

SQL> SELECT nume, functie, salariu, comision FROM angajati WHERE id_dep=10;

SQL> SELECT * FROM angajati WHERE functie='DIRECTOR';

- Datele din baza de date sunt case sensitive. Urmatoarea comanda nu va returna niciun rand, cu toate ca exista angajati cu functia DIRECTOR:

SQL> SELECT * FROM angajati WHERE functie='Director';

no rows selected



Cereri de interogare SQL



Cereri de interogare SQL*Plus

- Cererile de interogare SQL folosesc in exclusivitate comanda **SELECT**, fiind utilizate atat pentru interogarea obiectelor create de utilizatori cat si a celor sistem. Sintaxa comenzii este urmatoarea :

```
SELECT [DISTINCT,ALL]  [schema.table.]expresion expr_alias
FROM [schema.table@dblink] table_alias
[WHERE condition]
[START WITH condition][CONNECT BY condition]
[UNION,UNION ALL,INTERSECT,MINUS][SELECT command]
[GROUP BY expresion][ HAVING condition]
[ORDER BY expresion(position)][ASC,DESC]
[FOR UPDATE OF schema.table.column][NOWAIT]
```



Cereri de interogare SQL*Plus

- Parametrii comenzii au urmatoarea semnificatie(cei din paranteze sunt optionali):
- ***DISTINCT*** - returneaza o singura linie in cazul in care cererea returneaza linii duplicate;
- ***ALL*** – returneaza toate liniile simple si duplicate;
- ***schema.table*** – reprezinta schema de identificare a tabelei(sau view-lui) specificata prin *user.table_name*;
- ***expresion*** – reprezinta un nume de coloana sau o expresie care poate folosi functii sistem (* selecteaza toate coloanele tabelelor din clauza FROM);
- ***expr_alias*** – este un nume alocat unei expresii care va fi folosit in formatarea coloanei (apare in antetul listei);
- ***dblink*** – reprezinta numele complet sau partial de identificare a unei baze de date (database.domain@connection qualifier)



Cereri de interogare SQL*Plus

- ***table_alias*** – este un nume alocat unei tabele(view) care va fi folosit in cereri corelate;
- ***WHERE condition*** – reprezinta o clauza (inlantuire de conditii) care trebuie sa fie indeplinita in criteriul de selectie a liniilor;
- ***START WITH condition*** – stabileste criteriul de selectie pentru prima linie;
- ***CONNECT BY condition*** – stabileste o ierarhie de selectie a liniilor;
- ***GROUP BY expresion*** – stabileste criteriile de grupare a liniilor(numele coloanelor folosite in criteriul de grupare);
- ***HAVING condition*** – restrictioneaza liniile din grup la anumite conditii;



Cereri de interogare SQL*Plus

- ***UNION, UNION ALL, INTERSECT, MINUS*** – face operatii pe multimi de linii selectate de mai multe comenzi *SELECT* prin aplicarea anumitor restrictii;
- ***ORDER BY expresion(position)*** – ordoneaza liniile selectate dupa coloanele din expresie sau in ordinea coloanelor specificate prin pozitie;
- ***FOR UPDATE OF*** – face o blocare (*lock*) a liniilor in vederea modificarii anumitor coloane;
- ***NOWAIT*** – returneaza controlul userului daca comanda asteapta eliberarea unei linii blocata de un alt user.



Cereri simple de interogare

1. Cereri simple

- Cererea urmatoare returneaza toate coloanele si toate inregistrarile continute de o tabela:

```
SQL> SELECT * FROM angajati;
```

- Interogari pe anumite coloane ale unei tabele:

```
SQL> SELECT id_dep, den_dep  
      FROM departamente;
```

- Interogari care returneaza calcule pe anumite coloane si asigneaza un alias:

```
SQL> SELECT id_ang ecuson, nume,  
           salariu*12+nvl(comision,0) venit_anual  
      FROM angajati;
```



Cereri simple de interogare

- Interogari care concateneaza anumite coloane:

```
SQL> SELECT id_ang||'-'||nume_angajat , functie, data_ang  
      FROM angajati;
```

- Interogari care adauga coloane noi in lista:

```
SQL> SELECT id_ang||'-'||nume_angajat , functie,  
           salariu*12+nvl(comision,0) venit_lunar, ' ' semnatura  
      FROM angajati;
```



Cereri cu clauza WHERE

2. Cereri cu clauza WHERE

- Clauza WHERE poate compara valori de coloana ,valori literale, expresii aritmetice (sau functii) si poate avea patru tipuri de parametri:
 - nume de coloana;
 - operator de comparatie;
 - operator de negatie;
 - lista de valori.



Cereri cu clauza WHERE

- Lista persoanelor angajate in anul 1980:

```
SQL> SELECT id_ang ecuson, nume, functie, data_ang  
      FROM angajati  
     WHERE data_ang LIKE '%80';
```

- Lista persoanelor al caror nume incepe cu litera **F** si au numele functiei pe 7 caractere :

```
SQL> SELECT id_ang ecuson, nume, functie, data_ang  
      FROM angajati  
     WHERE nume LIKE 'F%' and functie LIKE '_____';
```



Cereri cu clauza WHERE

- Lista angajatilor din departamentul 20 care nu au primit comision:

```
SQL> SELECT id_ang ecuson, nume, functie, salariu FROM angajati  
      WHERE (comision=0 OR comision IS NULL) AND id_dep=20  
      ORDER BY nume;
```

- Lista angajatilor care au primit comision si nu sunt directori:

```
SQL> SELECT id_ang ecuson, nume, functie, salariu,comision  
      FROM angajati  
      WHERE comision IS NOT NULL and functie NOT LIKE 'DIRECTOR'  
      ORDER BY nume;
```



Cereri cu variabile substituite

3. Cерери cu variabile substituite

- Cерерile SQL pot fi executate folosind anumiti parametri, care se mai numesc si variabile substituite(sau de substitutie).

➤ Variabile ampersand (&)

O astfel de variabila se defineste sub forma **&nume_variabila** si este un parametru care se va introduce de la tastatura in timpul executiei comenzii in care este utilizata. Parametrul cu un singur ampersand trebuie introdus de fiecare data, chiar daca este folosit de mai ori in aceeasi comanda SQL.

Exemplu:

```
SQL> SELECT id_ang,nume,functie,salariu FROM angajati  
      WHERE id_sef=&ecuson_sef;
```

Enter value for ecuson_sef: 7698

- old 1: SELECT id_ang,nume,functie,salariu FROM angajati WHERE id_sef=&ecuson_sef
- new 1: SELECT id_ang,nume,functie, salariu FROM angajati WHERE id_sef=7698;



Cereri cu variabile substituite

➤ Variabile dublu ampersand (&&)

- Spre deosebire de variabila cu un singur ampersand, o variabila cu dublu ampersand va fi stocata si va putea fi apelata pe toata sesiunea de lucru.
- Definirea se face similar **&&nume_variabila** si va fi ceruta o singura data, folosirea ei de mai multe ori in cadrul comenzii se face apeland-o cu **&nume_variabila**.

Exemplu:

```
SQL> SELECT nume,functie,&&venit venit_lunar  
      FROM angajati WHERE &venit>2000;
```

Enter value for venit: salariu+nvl(comision,0)

- Pentru a reseta o variabila cu dublu ampersand se utilizeaza comanda UNDEFINE :

```
SQL>UNDEFINE venit
```



Cereri cu variabile substituite

➤ Variabile de sistem (&n)

- Sunt variabile definite numeric (1-9) care sunt predefinite de catre sistem si care functioneaza similar cu variabilele cu dublu ampersand. Avantajul folosirii acestor variabile este ca pot fi apelate direct dintr-un fisier de comenzi indirecte, fara a fi definite in prealabil. Suporta valori numerice, alfanumerice si data calendaristica.

Exemplu:

```
SQL> SELECT id_ang,nume,functie,data_ang FROM angajati  
      WHERE functie='&1' and data_ang>'&2'  
      ORDER BY data_ang;
```

```
SQL> SAVE angajari.sql
```

Created file angajari.sql

```
SQL> START angajari PROGRAMATOR 15-AUG-1981
```



Cereri definite cu ACCEPT

➤ Variabile definite cu ACCEPT

- Cand definim o variabila cu ampersand, totdeauna promptul va fi numele variabilei.
- Folosind comanda ACCEPT, se poate redefini promptul si chiar se pot ascunde caracterele introduse de la tastatura(facilitate utila in cazul unei parole).

Exemplu:

Se vor edita urmatoarele comenzi in fisierul **c:\temp\functia.sql**

SQL> ACCEPT functie_sef CHAR

PROMPT 'Introduceti functia sefului:' ;

SQL> SELECT nume,salariu,comision FROM angajati

WHERE functie='&functie_sef';

Fisierul se va rula in SQL*Plus si se va introduce functia sefului:

SQL> @c:\temp\functia.sql

Introduceti functia sefului: DIRECTOR



Cereri definite cu DEFINE

- **Variabile definite cu DEFINE si resetate cu UNDEFINE**
 - Variabilele definite cu DEFINE nu vor mai afisa promptul atunci cand sunt setate si raman setate pana cand vor fi resetate cu comanda UNDEFINE.

Exemple:

```
SQL> DEFINE procent_prima= 1.15
```

```
SQL> SELECT nume, salariu, salariu*&procent_prima prima  
      FROM angajati WHERE id_dep=20;
```

```
SQL> DEFINE venit= salariu+nvl(comision,0)
```

```
SQL> SELECT nume, data_ang, &venit venit_lunar FROM angajati  
      WHERE functie='DIRECTOR';
```

```
SQL> UNDEFINE procent
```

```
SQL> UNDEFINE venit
```



Cereri de interogare SQL*Plus

Exercitii:

1. Sa se faca o lista cu toti angajatii care s-au angajat inainte de anul 1982 si nu au primit comision.

a)

```
SQL> SELECT * FROM angajati
```

```
      WHERE data_ang<'1-JAN-1982' and  
            (comision is null or comision =0);
```

b)

```
SQL> SELECT * FROM angajati
```

```
      WHERE data_ang<'1-JAN-1982' and nvl(comision,0)=0;
```



Cereri de interogare SQL*Plus

2. Sa se faca o lista cu toti angajatii care au salariul peste 3000 \$ si nu au sefi, ordonati dupa departamente si nume.

SQL> SELECT * FROM angajati

WHERE salariu>3000 and id_sef is null

ORDER BY id_dep,nume;

3. Sa se faca o lista cu numele, functia si venitul anual al angajatilor care nu sunt directori pentru un departament introdus de la tastatura.
a)

**SQL> SELECT nume, functie, salariu*12+nvl(comision,0) venit_anual
FROM angajati**

WHERE functie not like 'DIRECTOR' and id_dep=&nr_depart;



Cereri de interogare SQL*Plus

b)

```
SQL> DEFINE venit_anual='salariu*12+nvl(comision,0)';  
SQL> SELECT nume,functie,&venit_anual FROM angajati  
      WHERE functie not like 'DIRECTOR' and id_dep=&nr_depart;  
SQL> UNDEFINE venit_anual ;
```

4. Sa se faca o lista cu departamentul, numele, data angajarii si salariul tuturor persoanelor angajate in anul 1981, din doua departamente pentru care id_dep se introduce de la tastatura .

a)

```
SQL> SELECT id_dep,nume,data_ang,salariu FROM angajati  
      WHERE data_ang like '%81' and  
            (id_dep=&nr_depart1 or id_dep=&nr_depart2)  
      ORDER BY id_dep;
```



Cereri de interogare SQL*Plus

5. Sa se scrie o comanda SQL care listeaza toti angajati dintr-un departament (introdus ca parametru de la tastura), care au venitul anual peste un venit mediu anual (introdus tot de la tastatura).
- a)

```
SQL> SELECT id_dep,id_ang,nume||'-'||functie  
          nume_functie,salariu,comision,  
          salariu*12+nvl(comision,0) venit_anual  
FROM angajati  
WHERE id_dep=&departament and  
      (salariu*12+nvl(comision,0))> &venit;
```



Cereri de interogare SQL*Plus

b)

```
SQL> DEFINE venit_anual='(salariu*12+nvl(comision,0))';
SQL> SELECT id_dep,id_ang,nume||'-'||functie
          nume_functie,salariu, comision,
          salariu*12+nvl(comision,0) venit_anual
     FROM angajati
 WHERE id_dep=&departament and &venit_anual> &venit;
SQL> UNDEFINE venit_anual ;
```



Cereri de interogare SQL*Plus

c)

```
SQL> DEFINE venit_anual= '(salariu*12+nvl(comision,0))';
SQL> ACCEPT depart number PROMPT 'Departament:';
SQL> ACCEPT val_venit number PROMPT 'Venit anual:';
SQL> SELECT id_dep,id_ang, nume||'-'||functie
          nume_functie, salariu, comision,
          salariu*12+nvl(comision,0) venit_anual
     FROM angajati
 WHERE id_dep=&depart and &venit_anual> &val_venit;
SQL> UNDEFINE venit_anual ;
SQL> UNDEFINE val_venit ;
SQL> UNDEFINE depart ;
```



Metode de JOIN



Metode de JOIN

- Pentru extragerea datelor din mai multe tabele din baza de date, comanda SELECT foloseste una sau mai multe metode de JOIN. Sintaxa pentru un join simplu este urmatoarea:

```
SELECT [DISTINCT,ALL] [table].expresion expr_alias  
FROM [schema.table1] table1_alias,  
      [schema.table2] table2_alias,  
WHERE table1_alias.column=table2_alias.column  
ORDER BY expresion(position)] [ASC,DESC]
```



Metode de JOIN

- Parametrii comenzii au urmatoarea semnificatie(cei din paranteze sunt optionali):
 - ***DISTINCT*** - returneaza o linie in cazul in care cererea returneaza linii duplicate;
 - ***ALL*** – returneaza toate liniile simple si duplicate;
 - ***schema.table*** – reprezinta schema de identificare a tablei(sau view-lui) specificata prin *user.table_name*;
 - ***expresion*** – reprezinta un nume de coloana sau o expresie care poate folosi functii sistem (* selecteaza toate coloanele tabelelor din clauza FROM);
 - ***expr_alias*** – este un nume alocat unei expresii care va fi folosit in formatarea coloanei (apare in antetul listei);



Metode de JOIN

- ***table_alias*** – este un nume alocat unei tabele(sau view) care va fi folosit in cereri corelate;
- ***WHERE condition*** – reprezinta o clauza (inlantuire de conditii) care trebuie sa fie indeplinita in criteriul de selectie a liniilor;
- ***ORDER BY expresion(position)*** – ordoneaza liniile selectate dupa coloanele din expresie sau in ordinea coloanelor specificate prin pozitie.



Equi JOIN

➤ Equi-join

- Daca in conditia de join apar numai egalitati, avem de-a face cu un *equi-join*. Pentru a putea sa realizam un join pe mai multe tabele, este obligatoriu ca ele sa contina coloane de acelasi tip, cu date comune sau corelate.

Exemplu:

- Pentru a lista angajatii dintr-un departament folosim tabela *angajati*, insa in aceasta tabela gasim asociat fiecarui angajat un id_dep, iar denumirea departamentului respectiv o gasim in tabela *departamente*.
- Se impune un join intre cele doua tabele, pentru ca in lista sa apara si denumirea departamentului.



Equi JOIN

```
SQL> SELECT a.id_dep ,b.den_dep depart,  
      a.nume, a.functie  
    FROM angajati a, departamente b  
   WHERE  a.id_dep=b.id_dep and  
         a.id_dep=10;
```

- Se observă ca au fost folosite aliasuri pentru tabele pentru a nu crea ambiguitate cand referim coloane cu aceeasi denumire.



Non Equi JOIN

➤ Non Equi-join

- Se foloseste cand in conditia de join avem alti operatori in afara de operatorul de egalitate sau cand doua sau mai multe tabele nu au coloane comune, dar trebuie totusi relationate.

Exemple:

- Relatia dintre tabelele *angajati* si *grila_salariu* este un non-equi-join, in care nicio coloana din prima tabela nu corespunde direct cu o coloana din cealalta.
- Relatia se obtine folosind tot un operator, altul decat = , de exemplu *between*.
- Pentru a evalua gradul de salarizare al unui angajat, trebuie sa consultam grila de salarizare pentru a identifica in ce plaja de salariu se incadreaza salariul:



Non Equi JOIN

```
SQL> SELECT a.nume, a.salariu, b.grad  
      FROM angajati a, grila_salariu b  
     WHERE a.salariu BETWEEN b.nivel_inf  
       AND b.nivel_sup AND a.id_dep=20;
```

- Sunt situatii cand trebuie sa folosim si equi-join si non equi-join intr-o cerere . In exemplul anterior, daca dorim sa listam si departamentul, va trebui sa facem join intre trei tabele:

```
SQL> SELECT c.den_dep,a.nume, a.salariu, b.grad  
      FROM angajati a, grila_salariu b, departamente c  
     WHERE a.salariu BETWEEN b.nivel_inf AND  
       b.nivel_sup AND a.id_dep=c.id_dep AND  
         a.id_dep=20;
```



Self JOIN

➤ Joinul unei tabele cu ea insasi (*self join*)

- Sunt situatii cand avem nevoie sa extragem date corelate din aceeasi tabela. De exemplu, daca dorim sa afisam care sunt sefii angajatilor trebuie sa extragem din tabela *angajati* si numele sefului (*id_sef*).

```
SQL> SELECT a.nume nume_ang,a.functie functie_ang,  
      b.nume nume_sef,b.functie functie_sef  
    FROM angajati a, angajati b  
   WHERE a.id_sef=b.id_ang AND a.id_dep=10;
```



Cross JOIN

➤ Produs cartezian

- Produsul cartezian a doua tabele se obtine prin concatenarea fiecarei linii dintr-o tabela cu fiecare linie din cealalta, rezultand un numar de linii egal cu produsul numarului de linii din fiecare tabela. Aceasta situatie este mai putin practica si se intalneste, de regula, cand sunt puse gresit conditii in clauza WHERE.

Exemplu:

```
SQL> SELECT nume ,functie ,den_dep  
      FROM angajati , departamente  
      WHERE functie='DIRECTOR';
```



Outer JOIN

➤ Join extern (*outer join*)

- Folosind equi-join putem selecta toate liniile care indeplinesc conditiile din clauza WHERE. Apar situatii cand cererea trebuie sa selecteze si linii care nu indeplinesc toate conditiile din clauza.

Exemple:

- Sa construim structura organizatorica a firmei selectand toate departamentele si angajatii care fac parte din fiecare departament. Exista, in tabela departamente, departamentul 40 care nu are niciun angajat si folosind equi-join acesta nu apare in lista. Pentru a depasi situatia se foloseste un *join extern* (+) asa cum se vede mai jos:



Outer JOIN

```
SQL> SELECT a.id_dep, a.den_dep, b.nume, b.functie  
      FROM departamente a, angajati b  
     WHERE a.id_dep=b.id_dep(+);
```

- În lista va apărea și departamentul VANZARI care nu are niciun angajat. Totdeauna semnul (+) se pune în dreptul tabelei deficitare ca informații.
- Putem să folosim și operatorul *BETWEEN* într-un join extern. Dacă vrem să aflăm care angajați raman în grila de salarizare prin dublarea salariilor, executăm urmatoarea cerere:



Outer JOIN

```
SQL> SELECT c.den_dep,a.nume, a.salariu, b.grad  
      FROM angajati a, grila_salariu b, departamente c  
     WHERE a.salariu*2 BETWEEN b.nivel_inf(+) AND  
          b.nivel_sup(+) AND a.id_dep=c.id_dep ;
```

- Daca dorim sa selectam toate departamentele care au primul caracter din id_dep din tabela departamente folosit printre id_dep din tabela angajati, se poate folosi si operatorul *LIKE* :

```
SQL> SELECT a.id_dep,a.den_dep,b.nume,b.functie  
      FROM departamente a, angajati b  
     WHERE b.id_dep(+) LIKE substr(a.id_dep,1,1)|| '%';
```



Vertical JOIN

➤ Join vertical (*vertical join*)

- Join-ul vertical este folosit pentru prelucrarea liniilor returnate de mai multe cereri *SELECT* si foloseste operatorii ***UNION*** (reuniune), ***INTERSECT*** (intersectie), ***MINUS*** (diferenta). In acest caz, join-ul se face dupa coloane de acelasi tip, nu dupa randuri, de aceea se mai numeste si vertical.

Exemple:

- Daca dorim lista angajatilor din departamentele 10 si 30, se poate utiliza cererea urmatoare:



Vertical JOIN

```
SQL> SELECT id_dep,nume,functie,salariu  
      FROM angajati WHERE id_dep=10  
UNION  
SELECT id_dep,nume,functie,salariu  
      FROM angajati WHERE id_dep=30;
```

- Trebuie retinut ca reuniunea se poate face pe coloane declarate de acelasi tip (number, varchar, date), chiar daca au semnificatii diferite. Sa construim o cerere care reuneste pe aceeasi coloana salariile angajatilor din departamentul 10 si cu comisioanele celor din departamentul 30.



Vertical JOIN

```
SQL> SELECT id_dep,nume,functie,'are salariu'  
      salar_comision, salariu sal_com  
      FROM angajati WHERE id_dep=10  
UNION  
      SELECT id_dep,nume,functie,'are comision ',  
      comision FROM angajati WHERE id_dep=30;
```

- Folosind operatorul *UNION ALL* se selecteaza si liniile duplicate:

```
SQL> SELECT functie FROM angajati WHERE id_dep=10  
      UNION ALL  
      SELECT functie FROM angajati WHERE id_dep=20;
```



Vertical JOIN

- Operatorul *INTERSECT* este folosit pentru a selecta liniile comune. Daca dorim sa aflam care sunt functiile angajatilor care au primit acelasi comision si care se regasesc in toate departamentele, scriem urmatoarea cerere:

```
SQL> SELECT functie, comision FROM angajati  
      WHERE id_dep=10  
INTERSECT  
SELECT functie,comision FROM angajati  
      WHERE id_dep=20  
INTERSECT  
SELECT functie,comision FROM angajati  
      WHERE id_dep=30;
```



Vertical JOIN

- Pentru a afla care sunt functiile din departamentul 10 care nu se regasesc in departamentul 30, folosim operatorul *MINUS* :

```
SQL> SELECT functie FROM angajati WHERE id_dep = 10  
MINUS
```

```
SELECT functie FROM angajati WHERE id_dep = 30;
```



Reguli de JOIN

- Observatii:
 - Atunci cand conditia de join lipseste, fiecare linie a unei tabele din lista FROM este asociata cu fiecare linie a celoralte tabele, obtinandu-se de fapt ***produsul cartezian*** al acestora.
 - Daca in conditia de join apar numai egalitati operatia este numita si ***equi-join***. In celelalte cazuri avem un ***non-equи-join***.
 - In lista de tabele care participa la join o tabela poate sa apara repetat. O astfel de operatie este numita si ***joinul unei tabele cu ea insasi (self-join)***.
 - In cazul in care o linie a unei tabele nu se coreleaza prin conditia de join cu nicio linie din celelalte tabele ea nu va participa la formarea rezultatului. Se poate insa obtine ca aceasta sa fie luata in considerare pentru rezultat, folosind un ***join extern (outer join)***.



Reguli de JOIN

- În cazul general al unui **join pe N tabele, condiția de join este compusa din N - 1 subconditii conectate prin AND** care relatează întreg ansamblul de tabele. Altfel spus, dacă se construiește un graf al condiției în care nodurile sunt tabele și arcele subconditii de join care leagă două tabele, atunci acest graf trebuie să fie conex.
- Marcajul de **join extern se poate folosi și atunci cand condiția de join este compusa**, cu excepția cazului în care se folosește OR sau operatorul de incluziune IN urmat de o listă care conține mai mult de o valoare.
- Joinul extern **se poate folosi și în conjuncție cu operatorii specifici SQL**.



Metode de JOIN in SQL-3

- Pana la versiunea Oracle 9i sintaxa joinului in Oracle era diferita de standardul ANSI (*American National Standards Institute*). Incepand cu aceasta versiune au fost introduse in limbaj si tipurile de join din standardul SQL-3(anul 1999) printre care ***cross-join***, ***join natural*** si mai multe ***variante de join extern***:
 - **CROSS JOIN** – join pe produs cartezian
 - **[INNER] JOIN ... USING** – join pe coloane comune
 - **[INNER] JOIN ... ON** – join general
 - **NATURAL JOIN** – join natural
 - **OUTER JOIN ... ON** – join extern
- In clauza FROM avem perechi de tabele care participa la join.



Cross JOIN in SQL-3

➤ CROSS JOIN

- Este folosit pentru obtinerea produsul cartezian si are urmatoarea sintaxa:

```
SELECT [DISTINCT|ALL]
[[table|table_alias].]{column|expression}
[column_alias]

FROM
[schema.]table1 [table1_alias] CROSS JOIN
[schema.]table2 [table2_alias]
[other clauses]
```



Cross JOIN in SQL-3

Exemplu:

```
SQL> SELECT a.nume, b.den_dep FROM angajati a  
      CROSS JOIN departamente b;
```

- Pentru a face JOIN si CROSS JOIN adaugam conditia de join in clauza WHERE:

```
SQL> SELECT a.nume, a.data_ang, b.den_dep, b.sediu  
      FROM angajati a  
      CROSS JOIN departamente b  
      WHERE a.id_dep=b.id_dep;
```

- Observatie: Conditia CROSS JOIN poate sa lipseasca in acest caz, rezultatul va fi acelasi.



Inner JOIN in SQL-3

➤ JOIN ... USING

- Este un equi-join dupa coloane cu acelasi nume specificate in USING (dar nu toate). Sintaxa este urmatoarea:

SELECT [DISTINCT|ALL]

{ {table|table_alias}. }{column|expression}

[column_alias]

FROM

[schema.]table1 [table1_alias]

[INNER] JOIN [schema.]table2 [table2_alias]

USING (column 1, column 2...)

[other clauses]



Inner JOIN in SQL-3

Exemple:

- Deoarece in ANGAJATI si DEPARTAMENTE avem o coloana cu acelasi nume (ID_DEP) putem face un equi-join astfel:

```
SQL> SELECT id_dep, den_dep, nume, data_ang, sediu  
      FROM angajati  
      INNER JOIN departamente USING (id_dep);
```

- Dupa cum se observa in lista USING numele coloanelor dupa care se efectueaza joinul nu trebuie prefixat cu numele sau aliasul vreunei dintre tabele (coloanele comune se afiseaza o singura data).
- Daca se doreste selectia doar dintr-un departament se va adauga conditia suplimentara pe WHERE:

```
SQL> SELECT id_dep, nume, den_dep, data_ang, sediu  
      FROM angajati  
      JOIN departamente USING (id_dep)  
      WHERE den_dep='CONTABILITATE';
```



Inner JOIN in SQL-3

➤ JOIN .. ON

- Prin aceasta clauza se implementeaza un join general. Conditia de join (si eventual si conditiile suplimentare) se pun in clauza ON. Sintaxa este urmatoarea:

SELECT [DISTINCT | ALL]

{ {table | table_alias}. } {column | expression} [column_alias]

FROM

[schema.]table1 [table1_alias]

[INNER] JOIN [schema.]table2 [table2_alias]

**ON { {table1 | table1_alias}. } column_fromTable1 =
 { {table2 | table2_alias}. } column_fromTable2**

[other clauses]



Inner JOIN in SQL-3

Exemplu:

- Cererea urmatoare efectueaza joinul dupa coloana ID_DEP si contine si o conditie suplimentara(filtru) dupa functia DIRECTOR:

```
SQL> SELECT a.nume, a.data_ang, b.den_dep, b.telefon  
      FROM angajati a JOIN departamente b  
     ON (a.id_dep=b.id_dep AND a.functie='DIRECTOR');
```

Conditia suplimentara se putea pune si in clauza WHERE.

- Intr-un JOIN ON se poate folosi si non-equi-join. Daca se doreste o lista a angajatilor cu salariul intre 1000 si 3000 se poate introduce urmatorul filtru:

```
SQL> SELECT a.nume, a.data_ang, b.den_dep  
      FROM angajati a JOIN departamente b  
     ON ( a.id_dep=b.id_dep AND  
          a.salariu BETWEEN 1000 AND 3000);
```



Inner JOIN in SQL-3

- Observatie: Un join general se poate aplica pe mai multe tabele.

Exemplu:

- Cererea urmatoare efectueaza joinul tablei de angajati cu tabelele de departamente si grila de salarii:

```
SQL> SELECT b.den_dep , a.nume, a.salariu, c.grad  
      FROM angajati a  
      JOIN departamente b  
        ON (a.id_dep=b.id_dep  
             AND salariu+nvl(comision,0) >1000 )  
      JOIN grila_salariu c  
        ON (a.salariu>=c.nivel_inf AND  
             a.salariu<=c.nivel_sup)  
      ORDER BY 1;
```



Natural JOIN in SQL-3

➤ NATURAL JOIN

- Este un equi-join dupa coloane cu acelasi nume si are sintaxa urmatoare:

SELECT [DISTINCT|ALL]

[[table|table_alias].]{column|expression}

[column_alias]

FROM [schema.]table1 [table1_alias]

NATURAL JOIN [schema.]table2 [table2_alias]

[other clauses]



Inner JOIN in SQL-3

Exemplu:

- Tabele DEPARTAMENTE si ANGAJATI au o coloana comună (ID_DEP) după care se poate face un join natural:

```
SQL> SELECT id_dep, nume, data_ang, den_dep, sediu
FROM angajati
NATURAL JOIN departamente ;
```

- Observații:
 - În cazul folosirii clauzei NATURAL JOIN cele două coloane trebuie să aibă același nume.
 - Dacă coloanele au același nume, nu se tine cont de tipul și semnificația coloanelor.
 - Nu se acceptă aliasuri pentru coloanele comune.



Outer JOIN in SQL-3

➤ OUTER JOIN ... ON

- Join-ul extern se foloseste cand se doreste ca in rezultat sa apara si liniile cu valori nule pe coloanele corespondente din tabelele relationate. Sintaxa este urmatoarea:

```
SELECT [DISTINCT] lista_de_expresii
FROM tabela1
LEFT \
RIGHT | OUTER JOIN tabela2
FULL /
ON (tabela1.nume_coloana1 =
tabela2.numecoloana2)
```



Left Outer JOIN in SQL-3

➤ LEFT OUTER JOIN

- În cazul join-ului extern stanga valorile nule provin din tabela2(cea deficitara ca date).

```
SELECT [DISTINCT|ALL]
{{table|table_alias}.}{column|expression}
[column_alias]

FROM

[schema.]table1 [table1_alias]

LEFT [OUTER ] JOIN [schema.]table2 [table2_alias]

ON {{table1|table1_alias}.}column_fromTable1 =
    {{table2|table2_alias}.}column_fromTable2

[other clauses]
```



Left Outer JOIN in SQL-3

Exemplu:

- Cererea de mai jos face o lista cu toate departamente si angajatii lor, dar afiseaza si departamentele care nu au niciun angajat:

```
SQL> SELECT a.nume nume_ang, a.data_ang, a.salariu, b.den_dep  
      departament  
      FROM departamente b  
      LEFT OUTER JOIN angajati a ON(a.id_dep=b.id_dep);
```

- Urmatoarea cerere este echivalenta cu prima:

```
SQL> SELECT a.nume nume_ang, a.data_ang, a.salariu, b.den_dep  
      departament  
      FROM angajati a , departamente b  
      WHERE a.id_dep(+) = b.id_dep;
```

- Notatia pentru join extern stanga este: R \bowtie_L S. In rezultat apar toate liniile tablei din stanga operatorului, inclusiv valorile nule.



Right Outer JOIN in SQL-3

➤ **RIGHT OUTER JOIN**

- In cazul join-ului extern dreapta valorile nule provin din tabela1(cea deficitara ca date). Sintaxa este similara cu join extern stanga.

Exemplu:

- Cererea urmatoare face o lista cu angajatii si sefii lor, inclusiv angajatii care nu au sefi:

```
SQL> SELECT b.id_ang sef, b.nume nume_sef, a.nume nume_ang,  
          a.data_ang, a.salariu  
        FROM angajati b
```

```
      RIGHT OUTER JOIN angajati a ON(a.id_sef= b.id_ang);
```

- Urmatoarea cerere este echivalenta cu prima:

```
SQL> SELECT b.id_ang sef, b.nume nume_sef, a.nume nume_ang,  
          a.data_ang, a.salariu  
        FROM angajati b, angajati a  
      WHERE a.id_sef= b.id_ang(+);
```

- Notatia pentru join extern dreapta este: R <o>_RS . In rezultat apar toate liniile tablei din dreapta operatorului, inclusiv valorile nule.



Outer JOIN in SQL-3

- Observatie: Sa analizam urmatoarele cereri:

```
SQL> SELECT b.id_ang sef, b.nume nume_sef, a.nume nume_ang,  
      a.data_ang, a.salariu  
      FROM angajati a  
      LEFT OUTER JOIN angajati b ON (a.id_sef = b.id_ang);
```

```
SQL> SELECT b.id_ang sef, b.nume nume_sef, a.nume nume_ang,  
      a.data_ang, a.salariu  
      FROM angajati b  
      RIGHT OUTER JOIN angajati a ON (a.id_sef = b.id_ang);
```

- Daca executam cele doua cereri vom obtine acelasi rezultat.
- Practic, daca inversam tipul de join si aliasurile de coloana vom obtine doua cereri identice.



Full Outer JOIN in SQL-3

➤ FULL OUTER JOIN

- În cazul unui join extern complet rezultatul contine toate liniile din rezultatul joinului general și joinul extern LEFT/RIGHT, obținut cu aceeași condiție. Sintaxa este urmatoarea:

SELECT [DISTINCT|ALL]

{ {table|table_alias}. }{column|expression} [column_alias]

FROM

[schema.]table1 [table1_alias]

FULL [OUTER] JOIN [schema.]table2 [table2_alias]

**ON { {table1|table1_alias}. }column_fromTable1 =
 { {table2|table2_alias}. }column_fromTable2**

[other clauses]



Full Outer JOIN in SQL-3

Exemplu:

- Cererea urmatoare returneaza o lista cu toti angajatii si toate departamentele:

```
SQL> SELECT a.nume, a.data_ang, b.den_dep, b.sediu  
      FROM angajati a  
      FULL OUTER JOIN departamente b ON (a.id_dep=b.id_dep);
```

- Observatie: Urmatoarea cerere **NU ESTE** echivalenta cu prima:

```
SQL> SELECT a.nume, a.data_ang, b.den_dep, b.sediu  
      FROM angajati a, departamente b  
      WHERE a.id_dep(+) = b.id_dep(+);
```

ERROR at line 3: ORA-01468: a predicate may reference only one outer-joined table

- Notatia pentru join extern complet este: R \bowtie S. In rezultat apar toate liniile tabelelor din stanga si dreapta operatorului.



Full Outer JOIN in SQL-3

- Observatie: Un join extern se poate face pe mai multe tabele.
Exemplu:
- Cererea urmatoare returneaza o lista cu toate departamentele (inclusiv cele fara angajati), toti angajatii si toate gradele de salarizare(inclusiv pe cele care nu au corespondent in salariile angajatilor):

```
SQL> SELECT b.den_dep, a.nume, a.data_ang, c.grad  
      FROM angajati a  
      FULL OUTER JOIN departamente b  
          ON (a.id_dep=b.id_dep)  
      FULL OUTER JOIN grila_salariu c  
          ON a.salariu between c.nivel_inf and c.nivel_sup  
      ORDER BY b.den_dep;
```



Functii SQL



Functii SQL

- Functia poate fi vazuta ca un operator de manipulare a datelor care accepta unul sau mai multe argumente (constanta , variabila , referire de coloana, etc.) si returneaza un rezultat.

Sintaxa de apelare este urmatoarea:

function_name(arg1, arg2,...)

- Rolul lor este de a face mai puternice cererile Oracle SQL*Plus si se pot folosi pentru:
 - Efectuarea calculelor numerice;
 - Prelucrare de siruri de caractere;
 - Prelucrarea datei calendaristice;
 - Schimbarea formatului pentru datele de afisare;
 - Conversia tipurilor de date.



Functii SQL

- Dupa specific si tipuri de argumente, functiile se pot imparti in urmatoarele categorii:
 - **Functii numerice**
 - **Functii pentru siruri**
 - **Functii pentru data calendaristica**
 - **Functii de conversie**
 - **Functii diverse** (accepta diverse tipuri de argumente)
 - **Functii de grup**



Functii numerice

➤ Functii numerice

- Aceste functii au ca parametri valori numerice si returneaza tot o valoare numERICA.

Exemple:

- **ABS (n)** – returneaza valoarea absoluta a lui n

SQL> SELECT abs(-12) FROM dual; -- returneaza valoarea 12

- **CEIL (n)** – returneaza cel mai mic intreg $\geq n$

SQL> SELECT ceil(14.7) FROM dual; -- returneaza valoarea 15

- **COS (n)** – returneaza **cosinus (n)** unde n este in radiani

SQL> SELECT cos(180 * 3.14/180) FROM dual; -- returneaza -1

- **COSH (n)** – returneaza cosinus hiperbolic

SQL> SELECT cosh(0) FROM dual; -- returneaza valoarea 1



Functii numerice

- **EXP (n)** – returneaza e la puterea n ($e=2.7182..$)

SQL> SELECT exp(4) FROM dual; -- returneaza valoarea 54.5981

- **FLOOR (n)** – returneaza cel mai mare intreg $\leq n$

SQL> SELECT floor(11.6) FROM dual; -- returneaza valoarea 11

- **LN (n)** – returneaza logaritmul natural al lui n ($n>0$)

SQL> SELECT ln(95) FROM dual; -- returneaza valoarea 4.5538

- **LOG (m,n)** – returneaza logaritmul in baza m al lui n

SQL> SELECT log(10,100) FROM dual; -- returneaza valoarea 2

- **MOD (m,n)** – returneaza restul impartirii lui m la n

SQL> SELECT mod(14,5) FROM dual; -- returneaza valoarea 4

- **POWER (m,n)** – returneaza m la puterea n

SQL> SELECT power(3,2) FROM dual; -- returneaza valoarea 9



Functii numerice

- **ROUND (n[,m])** – returneaza **n** rotunjit la :

m zecimale daca $m > 0$;

0 zecimale daca m este omis;

m cifre inainte de virgula daca $m < 0$;

SQL> SELECT round(15.193, 1) FROM dual; -- returneaza 15.2

SQL> SELECT round(15.193) FROM dual; -- returneaza 15

SQL> SELECT round(15.193, -1) FROM dual; -- returneaza 20

- **SIGN (n)** – returneaza -1 daca $n < 0$, 0 daca $n = 0$ si 1 daca $n > 0$

SQL> SELECT sign(-17.5) FROM dual; -- returneaza valoarea -1

- **SIN (n)** – returneaza **sinus (n)** unde n este in radiani

SQL> SELECT sin(30 * 3.14/180) FROM dual; -- returneaza 0.5

- **SINH (n)** – returneaza cosinus hiperbolic

SQL> SELECT sinh(1) FROM dual; -- returneaza valoarea 1.1752



Functii numerice

- **SQRT (n)** – returneaza radacina patrata a lui **n** unde $n > 0$
SQL> SELECT sqrt(26) FROM dual; -- returneaza valoarea 5.099
- **TAN (n)** – returneaza tangenta lui **n** , unde n este in radiani
SQL> SELECT tan(135 * 3.14/180) FROM dual; -- returneaza -1
- **TANH (n)** – returneaza tangenta hiperbolica a lui **n** ,unde n este in radiani

SQL> SELECT tanh(0.5) FROM dual; -- returneaza 0.4621

- **TRUNC (n[,m])** – returneaza **n** trunchiat la :
 - m** zecimale daca $m > 0$;
 - 0** zecimale daca m este omis;
 - m** cifre inainte de virgula daca $m < 0$.

SQL> SELECT trunc(15.193, 1) FROM dual; -- returneaza 15.1

SQL> SELECT trunc(15.193) FROM dual; -- returneaza 15

SQL> SELECT trunc(15.193, -1) FROM dual; -- returneaza 10



Functii alfanumerice

➤ Functii pentru siruri

- Aceste functii accepta la intrare valori alfanumerice si returneaza o valoare numerica sau tot o valoare alfanumerica. Cele care returneaza valori de tip VARCHAR2 pot avea lungimea maxima 4000 caractere iar cele de tip CHAR pot avea lungimea maxima 2000 caractere.

Exemple:

- **CHR (n)** -returneaza caracterul care are reprezentarea binara **n**
- SQL> **SELECT chr(75) FROM dual;** -- returneaza valoarea K
- **CONCAT (char1,char2)** – returneaza concatenarea lui **char1** cu **char2**

SQL> **SELECT concat(concat(nume,' este '),functie) nume_functie**
FROM angajati WHERE id_ang=7839;



Functii alfanumerice

- **INITCAP (char)** – returneaza char cu majuscule

SQL> SELECT initcap('popescu mihai') FROM dual; -- returneaza
Popescu Mihai

- **REPLACE (string,string1,[string2])** – inlocuieste in string
caracterele din string1 cu caracterele din string2

SQL> SELECT replace('JACK si JUE','J','BL') FROM dual;
-- returneaza valoarea BLACK si BLUE

- **TRANSLATE (col/string,string1,string2)** – translateaza in
col/string caracterele specificate in string1 in caracterele
specificate in string2

SQL> SELECT nume,translate(nume,'C','P') FROM angajati
WHERE id_dep=10; -- translateaza in nume litera C in P

- **RPAD/LPAD (char1,n,[char2])** – adauga la dreapta/stanga lui
char1 caracterele char2 pana la lungimea n

SQL> SELECT rpad(nume,20,'*') completare_nume FROM
angajati WHERE id_dep=10;



Functii alfanumerice

- **RTRIM (char[,set])** sterge din **char** ultimele caractere daca sunt in set

SQL> SELECT rtrim('Popescu','scu') FROM dual;--returneaza **Pope**

- **SUBSTR (char,m[,n])** – returneaza **n** caractere din **char** incepand cu pozitia **m**

SQL> SELECT substr ('Popescu ',2,3) FROM dual; -- returneaza **ope**

- **INSTR (char1,char2[,n[,m]])** – returneaza pozitia lui **char2** incepand cu pozitia **n** la a **m**-a aparitie

SQL> SELECT instr('Protopopescu','op',3,2) FROM dual;

-- returneaza **7**

- **LENGTH (char)** – returneaza lungimea lui **char** ca numar de caractere

SQL> SELECT length('analyst') FROM dual; --returneaza **7**

- **SIGN(col/expr/value)**-returneaza **-1** daca col/exp/valoarea este un numar negativ , **0** daca este zero si **+1** daca este numar pozitiv



Functii pentru data calendaristica

➤ Functii pentru data calendaristica

- O baza de date ORACLE stocheaza datele calendaristice in urmatorul format intern:

Secol / An / Luna / Zi / Ora / Minut / Secunda

- Formatul implicit de afisare sau intrare pentru o data calendaristica este *DD-MON-YY*.
- Plaja datei calendaristice este intre '1-JAN-4712' i.e.n si '31-DEC-4712' e.n.
- Toate functiile de tip data calendaristica intorc o valoare de tip *DATE* cu exceptia lui *MONTHS_BETWEEN* care intoarce o valoare numerica.



Functii pentru data calendaristica

Exemple:

- **ADD_MONTHS (date,n)** – returneaza o data prin adaugarea a n luni la **date**

```
SQL> SELECT nume, data_ang,add_months(data_ang,3)  
      data_mod FROM angajati WHERE id_dep=10;
```

- **LAST_DAY (date)** – returneaza data ultimei zile din luna cuprinsa in **date**

```
SQL> SELECT nume,data_ang,last_day(data_ang) ultima_zi  
      FROM angajati WHERE id_dep=10;
```

- **MONTHS_BETWEEN(date1,date2)** – returneaza numarul de luni (si fractiuni de luna) cuprinse intre **date1** si **date2**. Daca **date1>date2** rezultatul va fi pozitiv altfel va fi negativ.

```
SQL> SELECT nume, data_ang, sysdate,  
      months_between(sysdate,data_ang) luni_vechime  
      FROM angajati WHERE id_dep=10;
```



Functii pentru data calendaristica

- **NEXT_DAY(date,char)** – returneaza data urmatoarei zile **char** dupa **date**

```
SQL> SELECT next_day('17-NOV-2006', 'TUESDAY')  
      marti_urmatoare FROM dual;
```

- **ROUND(date,fmt)** – returneaza data prin rotunjirea lui **date** la formatul **fmt**

```
SQL> SELECT round(to_date('17-NOV-2006'), 'YEAR') rot_an  
      FROM dual;
```

```
SQL> SELECT round(to_date('17-NOV-2006'), 'MONTH')  
      rot_luna FROM dual;
```

- **TRUNC(date,fmt)** – returneaza data prin trunchierea lui **date** la formatul **fmt**

```
SQL> SELECT trunc(to_date('17-NOV-2006'), 'YEAR') trunc_an  
      FROM dual;
```



Functii pentru data calendaristica

- Folosind operatorii aritmetici + si – se pot face diferite operatii cu date calendaristice:
 - **data + numar** - aduna un numar de zile la data, returnand tot o data calendaristica;
 - **data - numar** - scade un numar de zile la data, returnand tot o data calendaristica;
 - **data1 – data2** - scade data2 din data 1, obtinand un nr. de zile;
 - **data + numar/24** - aduna la data un numar de ore pentru a obtine tot o data calendaristica.

Exemplu:

```
SQL> SELECT data_ang,data_ang+7,data_ang-7,  
           sysdate-data_ang FROM angajati  
      WHERE data_ang LIKE '%JUN%';
```



Functii de conversie

➤ Functii de conversie

- Aceste functii fac conversia unui tip de data in alt tip de data.
- **TO_CHAR(date[,fmt[,nlsparams]])** – face conversia unei date de tip DATE in format VARCHAR2

```
SQL> SELECT nume, to_char(data_ang, 'Month DD, YYYY')  
      data_ang  FROM angajati  
      WHERE to_char(data_ang,'YYYY') LIKE '1987';
```

- **TO_DATE(char[,fmt[,nlsparams]])** – face conversia unei variabile de tip CHAR sau VARCHAR2 in format DATE

```
SQL> SELECT to_date('15112006' , 'DD-MM-YYYY') data  
      FROM dual;
```



Functii de conversie

- **TO_NUMBER(char[,fmt[,nlsparams]])** – face conversia unei variabile de tip CHAR sau VARCHAR2 in format NUMBER

Exemplu:

- Pentru calculul primei in functie de anul angajarii, folosim cererea:

```
SQL> SELECT nume, functie, salariu,  
           salariu+to_number(to_char(data_ang,'yyyy'))/10 prima  
      FROM angajati  
     WHERE to_number(to_char(data_ang , 'YYYY'))=1987;
```



Functii de conversie

- Exemple de format pentru date numerice:

Format	Semnificatie	Exemple	
9	numere(nr.de 9 determina lungimea de afisare)	999999	1234
0	afiseaza zerourile de la inceput	099999	001234
\$	simbolul dolar	\$999999	\$1234
.	punct zecimal	999999.99	1234.00
,	virgula	999,999	1,234
MI	semnele minus la dreapta(valori negative)	999999MI	1234-
PR	paranteze pentru numere negative	999999PR	<112>
EEEE	notatie stiintifica	99.999EEEE	1.234E+03
V	inmultire cu 10**n (n=nr de 9 dupa V)	9999V99	123400
B	afiseaza valori zero ca blancuri(nu 0)	B9999.99	1234.00

- Formatul de afisare a datelor se poate seta cu comanda COLUMN:
SQL>COLUMN nume FORMAT a30 -- format alfanumeric max 30 car
SQL>COL salariu FOR 99999.99 -- format numeric cu 2 zecimale



Functii de conversie

- **EXTRACT ([YEAR], [MONTH], [DAY], [HOUR], [MINUTE], [SECONDE] from datetime)** – extrage anul, luna, ziua, minutul, secunda din data calendaristica **datetime**.

```
SQL> SELECT nume, data_ang,  
           extract (year from data_ang) ANUL,  
           extract(month from data_ang) LUNA,  
           extract (day from data_ang) ZIUA  
      FROM angajati  
     WHERE functie ='ANALIST' ;
```



Functii diverse

➤ Functii diverse

- Aceste functii accepta orice tip de argumente.
- **GREATEST(expr1 [,expr2]...)** – returneaza cea mai mare valoare din lista de argumente numerice, sau expresia care are prima litera pozitionata ultima in alfabet , pentru date alfanumerice.

SQL> SELECT greatest(12,34,77,89) from dual;

- **LEAST(expr1 [,expr2]...)** – similar cu GREATEST dar returneaza cea mai mica valoare din lista de argumente.
- **DECODE(expr,search1,result1,...default)** - **expr** este comparata cu fiecare valoare **search** si intoarce **rezult** daca **expr** este egala cu valoarea **search**, iar daca nu gaseste nicio egalitate intoarce valoarea **default**.
- Functia DECODE are efectul unui **CASE** sau a unei constructii **IF-THEN-ELSE**.



Functii diverse

- Tipurile de parametri pot fi :
 - **expression** poate fi orice tip de data;
 - **search** este de același tip ca *expression*;
 - **result** este valoarea întoarsă și poate fi orice tip de data;
 - **default** este de același tip ca *result*.

Exemplu:

- În exemplul urmator se calculează prima în raport de funcția angajatului în companie:

```
SQL> SELECT nume,functie,salariu,  
        decode(functie,'DIRECTOR', salariu*1.35, 'ANALIST',  
        salariu*1.25, salariu/5) prima  
        FROM angajati WHERE id_dep=20  
        ORDER BY functie;
```



Functii diverse

- **CASE expr** **WHEN value1/expr1 THEN statements_1;**
WHEN value2/expr2 THEN statements_2;
...
[ELSE statements_k;]
END
- CASE functioneaza similar cu functia DECODE. Parametrii sunt:
 - **expr** – este expresia care se va evalua
 - **statements_1** – este valoarea returnata cand **expr = value1/expr1**;
 - **statements_2** – este valoarea returnata cand **expr = value2/expr2**;
 - **statements_k** – este valoarea implicită returnata cand
expr <> value1, value2, ...

Observatii:

- Expresia **expr_i** poate fi diferita pentru fiecare ramura WHEN;
- Expresia **expr_i** poate contine mai multi operatori de comparative;
- Functia CASE poate fi folosita in cererea SELECT sau clauza WHERE si are mai multe tipuri de sintaxa.



Functii diverse

Exemple:

```
SQL> SELECT nume,data_ang, salariu,  
CASE  
WHEN salariu <= 1000 THEN salariu*0.1  
WHEN salariu > 1000 THEN salariu*0.2  
END prima  
FROM angajati;
```

```
SQL> SELECT id_ang, nume,functie, data_ang, salariu  
FROM angajati  
WHERE functie= (CASE id_ang  
WHEN 7839 then 'PRESEDINTE'  
WHEN 7698 then 'DIRECTOR'  
END);
```



Functii diverse

- **NVL(expr1 ,expr2)** – returneaza **expr2** daca **expr1** este **null**

```
SQL> SELECT nume,data_ang,comision,nvl(comision,0) nvl_com  
      FROM angajati WHERE id_dep=30  
      ORDER BY nume;
```

- **COALESCE(expr1 ,expr2, ...)** – returneaza prima expresie **not null** din lista de argumente

```
SQL> SELECT nume,data_ang, comision, coalesce(comision,0)  
      FROM angajati WHERE id_dep=30  
      ORDER BY nume;
```



Functii diverse

- De retinut ca valoarea null trebuie obligatoriu convertita la zero atunci cand se fac operatii aritmetice, altfel rezultatul va fi null.
- Functiile pot fi imbricate pana la orice nivel, ordinea de executie fiind din interior spre exterior.

USER – returneaza userul curent Oracle



Functii de grup

➤ Functii de grup

- Sintaxa pentru functiile de grup este urmatoarea:

SELECT [column,] group_function(column)...

FROM table

[WHERE condition]

[GROUP BY [CUBE] [ROLLUP] group_expression]

[HAVING having_expression]

[ORDER BY column(position)] [ASC,DESC];



Functii de grup

- Parametrii comenzii au urmatoarea semnificatie(cei din paranteze sunt optionali):
- ***group_function*** – specifica numele functiei/functiilor de grup;
- ***WHERE condition*** – reprezinta o inlantuire de conditii care trebuie sa fie indeplinita in criteriul de selectie a liniilor;
- ***GROUP BY group_expresion*** - specifica coloanele dupa care se face gruparea rezultatelor;
- ***HAVING***– este folosita pentru a specifica conditiile de grupare a rezultatelor;
- ***having_expresion*** – reprezinta un nume de coloana sau o expresie care poate folosi functii si coloane;
- ***ORDER BY***– specifica coloanele(sau ordinea coloanelor specificate prin pozitie) dupa care se face ordonarea rezultatelor;



Functii de grup

- **CUBE** – este un operator folosit impreuna cu o funcție de grup pentru a genera randuri suplimentare in rezultate(produce subtotaluri pentru toate combinatoriile posibile ale gruparii specificate in clauza GROUP BY);
- **ROLLUP** – este un operator folosit pentru a obtine un set de rezultate care contin subtotaluri, pe langa valorile pentru randurile grupate in mod obisnuit.



Functii de grup

- Aceste functii returneaza rezultate bazate pe grupuri de inregistrari, nu pe o singura inregistrare ca cele prezentate la punctele anterioare.
- Gruparea se face folosind clauza GROUP BY intr-o cerere SELECT si in acest caz toate elementele listei trebuie cuprinse in clauza de grupare.
- Functiile de grup pot fi apelate si in clauza HAVING, dar nu pot fi apelate in clauza WHERE in mod explicit.
- Se poate folosi operatorul DISTINCT pentru a sorta numai elementele distincte din lista, dar se poate folosi si operatorul ALL pentru a considera si inregistrarile duplicate.



Functii de grup

- **AVG([DISTINCT/ALL] expr)** – returneaza valoarea medie a lui **expr**, ignorand valorile nule.

Exemplu: Sa calculam valoarea medie a salariului si comisionului pe toate departamentele:

```
SQL> SELECT avg(salariu), avg(comision) FROM  
      angajati;
```

Exemplu: Daca dorim sa aflam valorile medii pe fiecare departament, trebuie sa folosim obligatoriu clauza GROUP BY:

```
SQL> SELECT id_dep, avg(salariu), avg(comision)  
      FROM angajati  
      GROUP BY id_dep;
```

- Prin folosirea operatorilor DISTINCT si ALL rezultatul interogarii poate fi diferit:

```
SQL> SELECT id_dep, avg(salariu), avg(all salariu),  
      avg(distinct salariu)  
      FROM angajati GROUP BY id_dep;
```



Functii de grup

- **COUNT(* | [DISTINCT/ALL] expr)** – returneaza numarul de linii introarse de interogare. Daca se fosese * se numara toate liniile, inclusiv cele care contin valori nule pe anumite coloane, iar daca se foloseste **expr** se numara numai valorile **not null** ale lui **expr**.

Exemplu: Pentru a afla numarul angajatilor care au primit comision pe fiecare departament, folosim urmatoarea cerere:

```
SQL> SELECT id_dep, count(*), count(comision),
      count(all comision), count(distinct comision)
      FROM angajati GROUP BY id_dep;
```

- **MAX([DISTINCT/ALL] expr)** – returneaza valoarea maxima pentru **expr**.

Exemplu: Sa calculam salariul maxim pe fiecare departament:

```
SQL> SELECT a.id_dep, b.den_dep, max(salariu)
      FROM angajati a, departamente b
      WHERE a.id_dep=b.id_dep
      GROUP BY a.id_dep, b.den_dep;
```



Functii de grup

- **MIN([DISTINCT/ALL] expr)** – returneaza valoarea minima pentru **expr**.

Exemplu: Sa calculam venitul minim pe fiecare departament:

```
SQL> SELECT id_dep,  
           min(salariu + nvl(comision,0)) venit_minim  
      FROM angajati GROUP BY id_dep;
```

- Trebuie mentionat ca operatorii DISTINCT si ALL nu au vreun efect pentru functiile MIN si MAX.
- **SUM([DISTINCT/ALL] expr)** – returneaza suma valorilor pentru **expr**.

Exemplu: Suma salariilor si comisioanelor pe fiecare departament:

```
SQL> SELECT id_dep, sum(salariu),  
           sum(distinct salariu), sum(comision)  
      FROM angajati GROUP BY id_dep;
```



Functii de grup

- **CUBE**– Genereaza un superset de grupuri prin referiri incruisate pe coloanele incluse in clauza GROUP BY.

Exemplu: Cererea urmatoare calculeaza salariul total pe toate departamentele, pe fiecare tip de functie, pe fiecare departament si pe fiecare tip de functie din cadrul departamentelor 10 si 20:

```
SQL> SELECT id_dep, functie, SUM(salariu) salariu_total  
      FROM angajati  
     WHERE id_dep < 30  
   GROUP BY CUBE (id_dep, functie);
```

ID_DEP	FUNCTIE	SALARIU_TOTAL
		19625
	ANALIST	6000
	DIRECTOR	5425
	TEHNICIAN	3200
	PRESEDINTE	5000
10		8750
10	DIRECTOR	2450
10	TEHNICIAN	1300
10	PRESEDINTE	5000
20		10875
20	ANALIST	6000
20	DIRECTOR	2975
20	TEHNICIAN	1900



Functii de grup

- **ROLLUP** – Genereaza un superset de grupuri pe coloanele incluse in clauza GROUP BY.

Exemplu: Cererea urmatoare calculeaza salariul total pe fiecare tip de functie din cadrul departamentelor, pe fiecare departament si pe toate departamentele care indeplinesc restrictia de id_dep(10 si 20):

```
SQL> SELECT id_dep, functie, SUM(salariu) salariu_total  
      FROM angajati  
     WHERE id_dep < 30  
   GROUP BY ROLLUP (id_dep, functie);
```

ID_DEP	FUNCTIE	SALARIU_TOTAL
10	DIRECTOR	2450
10	TEHNICIAN	1300
10	PRESEDINTE	5000
10		8750
20	ANALIST	6000
20	DIRECTOR	2975
20	TEHNICIAN	1900
20		10875
		19625



Sucereri SQL

Alexandru Boicea Curs: Baze de date



Subcereri SQL

- În Oracle SQL*Plus subcererile sunt cereri SQL incluse în clauzele SELECT, FROM, WHERE, HAVING sau ORDER BY ale altor cereri numite și cerere principală.
- Rezultatele returnate de o subcerere sunt folosite de o altă subcerere sau de cererea principală în situații cum ar fi:
 - Crearea de tabele sau view-uri;
 - Inserarea, modificarea și stergerea înregistrărilor din tabele;
 - În furnizarea valorilor pentru condiții puse în comenziile SELECT, UPDATE, DELETE, INSERT și CREATE TABLE.



Subcereri SQL

- Din punct de vedere al rolului pe care il au intr-o comanda SQL si a modului de a face constructia comenzii, subcererile pot fi:
 - **Subcereri ascunse**
 - **Subcereri corelate**
 - **Subcereri pe tabela temporara**
 - **Subcereri pe clauza HAVING**
 - **Subcereri pe clauza SELECT**
 - **Subcereri pe clauza ORDER BY**



Subcereri ascunse

➤ Subcereri ascunse

- Subcererile ascunse pot fi impartite in mai multe categorii, in functie de numarul de coloane sau linii pe care le returneaza:
 - Subcereri care returneaza o valoare
 - Subcereri care returneaza o coloana
 - Subcereri care returneaza o linie
 - Subcereri care returneaza mai multe linii

✓ Subcereri care returneaza o valoare

- Aceste subcereri se folosesc intr-o alta cerere si au urmatoarea constructie:

SELECT column 1, column 2, ...

FROM table 1

WHERE column =

**(SELECT column FROM table 2
WHERE condition);**



Subcereri ascunse

- Subcererea se executa prima pentru a verifica conditia din clauza WHERE si poate intoarce o singura valoare dintr-o tabela, prin conditii care depind de datele din tabela folosita in cererea principala. Daca conditia din clauza WHERE a subcererii nu returneaza o singura valoare, atunci se va genera o eroare.

Exemplu:

- Pentru a afla care sunt angajatii care au cel mai mare salariu in firma, putem sa folosim urmatoarea cerere:

```
SQL> SELECT id_dep, nume, functie, salariu  
      FROM angajati  
     WHERE salariu=( SELECT max(salariu)  
                      FROM angajati );
```



Subcereri care returneaza o coloana

✓ Subcereri care returneaza o coloana

- Sunt subcereri care returneaza mai multe valori si folosesc operatorii **IN** si **NOT IN** intr-o constructie ca mai jos:

SELECT column 1, column 2, ...

FROM table 1

WHERE column IN [NOT IN]

(SELECT column

FROM table 2

WHERE condition);

- In cazul in care am folosi operatorul **=** in locul operatorului **IN** s-ar genera o eroare pentru ca subcererea returneaza mai mult de o valoare (ORA-01427). Trebuie specificat ca operatorul **IN** se poate folosi in locul operatorului **=**, dar invers este gresit.



Subcereri care returneaza o coloana

Exemplu:

- Daca vrem sa aflam salariile angajatilor care au functii similare functiilor aferente departamentului 20, folosim urmatoarea cerere :

```
SQL> SELECT nume, functie, salariu
FROM angajati
WHERE functie IN
  (SELECT DISTINCT functie
FROM angajati
WHERE id_dep=20);
```



Subcereri care returneaza o linie

✓ Subcereri care returneaza o linie

- Sunt subcereri care returneaza mai multe coloane dar o singura linie si folosesc operatorii **IN** si **NOT IN**, intr-o constructie ca mai jos:

SELECT column 1, column 2, ...

FROM table 1

WHERE (column 1,column 2...) = [<>][IN] [NOT IN]

(SELECT column 1,column 2..

FROM table 2

WHERE condition);

- Numarul, ordinea si tipul coloanelor din clauza WHERE trebuie sa coincida cu cele din subcerere.



Subcereri care returneaza o linie

Exemplu:

- Daca dorim care angajati din departamentul 30 au aceeasi functie ca cea a lui Tache Ionut, folosim urmatoarea cerere:

```
SQL> SELECT id_dep, nume, functie, data_ang  
      FROM angajati  
     WHERE (id_dep, functie) =  
           (SELECT id_dep, functie FROM angajati  
            WHERE nume='TACHE IONUT');
```

- In acest caz subcererea returneaza o singura linie, deoarece Tache are o singura functie si este angajat la un singur departament. In eventualitatea ca pot exista mai multi angajati cu acelasi nume, se va folosi clauza DISTINCT in subcerere(daca au acelasi depart si job).



Subcereri care returneaza mai multe linii

- ✓ **Subcereri care returneaza mai multe linii**
 - Aceste subcereri au o constructie asemanatoare ca la punctul precedent, dar intorc mai multe linii cu mai multe coloane, drept pentru care se mai numesc si subcereri care intorc o tabela.

SELECT expr 1,... expr n

FROM table 1

WHERE (expr 1,...expr k) IN [NOT IN]

(SELECT expr 1, ...expr k

FROM table 2

WHERE condition);

- Trebuie specificat ca in locul coloanelor se pot folosi si expresii. Numarul de expresii si ordinea lor specificate in clauza WHERE trebuie sa coincida cu numarul de expresii din subcerere si sa fie de acelasi tip.



Subcereri care returneaza mai multe linii

Exemplu:

- Pentru a afla persoanele care au venit maxim pe fiecare departament, folosim urmatoarea cerere:

```
SQL> SELECT id_dep, nume, functie, salariu+nvl(comision,0) venit
      FROM angajati
      WHERE (id_dep, salariu+nvl(comision,0)) IN
            (SELECT id_dep,max(salariu+nvl(comision,0)) venit_max
              FROM angajati GROUP BY id_dep);
```

- In cazul unei subcereri ascunse, cererea SELECT din subcerere ruleaza prima si se executa o singura data, intorcand valori ce vor fi folosite de cererea principala.



Subcereri corelate

➤ Subcereri corelate

- Subcererile corelate se executa o data pentru fiecare linie candidat prelucrata de cererea principala si la executie folosesc cel putin o valoare dintr-o coloana din cererea principala.
- O subcerere corelata se relateaza cu cererea exterioara prin folosirea unor coloane ale cererii exterioare in clauza predicatului cererii interioare.

Constructia unei subcereri corelate este urmatoarea:

SELECT expr 1,... expr n

FROM table 1

WHERE (expr 1,...expr k) IN [NOT IN]

(SELECT expr 1, ...expr k

FROM table 2

WHERE table 2.expr x = table 1.expr y [AND,OR] ..);



Subcereri corelate

- Pentru o cerere corelata, subcererea se executa de mai multe ori, cate o data pentru fiecare linie prelucrata de cererea principala, deci cererea interioara este condusa de cererea exterioara.
- Pasii de executie ai unei subcereri corelate sunt:
 - se obtine linia candidat prin procesarea cererii exterioare;
 - se executa cererea interioara corelata cu valoarea liniei candidat;
 - se folosesc valorile rezultate din cererea interioara pentru a prelucra linia candidat;
 - se repeta pana nu mai ramane nicio linie candidat.
- Trebuie specificat ca desi subcererea corelata se executa repeatat, aceasta nu inseamna ca subcererile corelate sunt mai putin eficiente decat subcererile ascunse.



Subcereri corelate

Exemplu:

- Daca vrem sa aflam persoanele care au salariul peste valoarea medie a salariului pe departamentul din care fac parte, folosim cererea urmatoare:

```
SQL> SELECT a.id_dep, a.nume, a.functie, a.salariu  
      FROM angajati a WHERE a.salariu >  
          (SELECT avg(salariu) salariu_mediu FROM angajati b  
           WHERE b.id_dep=a.id_dep )  
      ORDER By id_dep;
```

- Cererea principala proceseaza fiecare linie din tabela angajati si pastreaza toti angajatii care au salariul peste salariul mediu pe departamentul respectiv returnat de subcerere, care se coreleaza cu cererea principala prin conditia din clauza WHERE. In acest exemplu se foloseste un join pe aceeasi tabela.



Subcereri imbricate

- Subcererile pot fi imbricate. De exemplu, pentru a afla care angajati din firma au salariul mai mare decat salariul maxim din departamentul de proiectare, folosim cererea urmatoare:

```
SQL> SELECT nume, functie, data_ang, salariu  
      FROM angajati  
     WHERE salariu > ( SELECT max(salariu)  
                           FROM angajati  
                          WHERE id_dep= (SELECT id_dep  
                                         FROM departamente  
                                         WHERE  
                                           den_dep= 'PROIECTARE'));
```



Subcereri pe tabela temporara

➤ Subcereri pe tabela temporara

- Aceste subcereri se intalnesc in cazul in care se foloseste o subcerere la nivelul clauzei FROM din cererea principala.
Constructia este urmatoarea:

```
SELECT t1.column1,t1.column2,.. t2.expr 1,... t2.expr k  
FROM table t1,  
      ( SELECT expr 1, ...expr n  
        FROM table  
        WHERE conditions ) t2  
WHERE conditions  
ORDER BY expr;
```



Subcereri pe tabela temporara

Exemplu:

- Pentru a afla salariul maxim pe fiecare departament, se poate face o constructie ca mai jos, unde subcererea intoarce o tabela temporara cu salariul maxim pe fiecare departament:

```
SQL> SELECT b.id_dep, a.den_dep, b.sal_max_dep  
      FROM departamente a, (SELECT id_dep, max(salariu)  
                            sal_max_dep FROM angajati GROUP BY id_dep) b  
     WHERE a.id_dep = b.id_dep  
     ORDER BY b.id_dep;
```



Subcereri pe clauza HAVING

➤ Subcereri pe clauza HAVING

- Aceste subcereri sunt specificate in clauza HAVING intr-o constructie ca cea de mai jos :

SELECT expr 1,... expr n

FROM table 1

WHERE conditions

HAVING expr (operator)

(**SELECT expr 1, ...expr k**

FROM table 2

WHERE conditions)

GROUP BY expresion;

- Trebuie retinut ca o clauza HAVING se foloseste intotdeauna impreuna cu clauza GROUP BY si intr-o clauza HAVING se pot folosi functii de grup in mod explicit.



Subcereri pe clauza HAVING

Exemplu:

- Pentru a afla ce departamente au cei mai multi angajati pe aceeasi functie, folosim urmatoarea cerere:

```
SQL> SELECT d.den_dep, a.functie, count(a.id_ang) nr_ang
      FROM angajati a, departamente d
      WHERE a.id_dep = d.id_dep
      HAVING count(a.id_ang) = (SELECT max(count(id_ang))
                                FROM angajati GROUP BY id_dep, functie)
      GROUP BY d.den_dep, a.functie;
```

- Clauza HAVING poate fi folosita si intr-o subcerere pe tabela temporara.



Subcereri pe clauza SELECT

➤ Subcereri pe clauza SELECT

- Se poate construi o subcerere si pe clauza SELECT, avand urmatoarea constructie :

SELECT expr 1,... expr n,

(SELECT expr FROM table 2

WHERE table 2.expr x = table 1.expr y [AND,OR] ..) alias

FROM table 1

WHERE conditions

ORDER BY expr 1,...expr k

- Aceste subcereri pot fi corelate sau ascunse dar trebuie sa returneze intotdeauna o singura valoare.



Subcereri pe clauza SELECT

Exemplu:

- Daca vrem sa aflam care sunt sefii directi ai angajatilor din departamentul 20, folosim cererea urmatoare:

```
SQL> SELECT nume nume_ang,
           (SELECT nume FROM angajati b
            WHERE b.id_ang=a.id_sef) nume_sef
      FROM angajati a
     WHERE id_dep=20
    ORDER BY nume ;
```



Subcereri pe clauza ORDER BY

➤ Subcereri pe clauza ORDER BY

- O subcerere pe clauza ORDER BY se foloseste numai pentru cereri corelate si trebuie sa returneze intotdeauna o singura valoare, avand urmatoarea constructie :

SELECT expr 1,... expr n,

FROM table 1

WHERE conditions

ORDER BY

(SELECT expr FROM table 2

WHERE table 2.expr x = table 1.expr y [AND,OR] ..)

ASC/DESC



Subcereri pe clauza ORDER BY

Exemplu:

- Daca vrem sa facem o lista cu angajatii din departamentele 10 si 20, ordonati descrescator tinand cont de numarul lor pe fiecare departament, folosim cererea urmatoare:

```
SQL> SELECT id_dep, nume, functie FROM angajati a  
      WHERE id_dep in (10,20)  
      ORDER BY  
            (SELECT count(*)  
             FROM angajati b  
             WHERE a.id_dep=b.id_dep) DESC;
```



Operatori in subcereri

➤ Operatori in subcereri

- Operatorii prezentati pentru cereri sunt valabili si pentru subcereri, dar mai sunt si alti operatori specifici.
- ✓ **Operatorii SOME/ANY si ALL**
- Acesti operatori sunt folositi in subcereri care intorc mai multe linii si sunt folositi impreuna cu operatorii logici in clauzele WHERE si HAVING.
- Operatorul SOME (sau sinonimul lui ANY) compara o expresie cu fiecare valoare returnata de o subcerere si pastreaza liniile unde expresia indeplineste conditia impusa de operatorul logic.
- Daca este folosit impreuna cu operatorul logic $>$, atunci are semnificatia de *mai mare decat minim*.



Operatori in subcereri

Exemplu:

- Pentru a afla care sunt angajatii care au salariul mai mare decat cel mai mic salariu pentru functia de programator, folosim cererea urmatoare:

```
SQL> SELECT id_dep, nume, functie, salariu
FROM angajati
WHERE salariu > SOME (SELECT DISTINCT salariu
FROM angajati
WHERE functie='PROGRAMATOR')
ORDER BY id_dep, nume;
```



Operatori in subcereri

- Operatorul ALL lucreaza similar cu operatorii SOME/ANY, iar daca este folosit impreuna cu operatorul logic `>`, atunci are semnificatia de *mai mare decat maxim*.

```
SQL> SELECT id_dep, nume, functie, salariu  
      FROM angajati  
     WHERE salariu > ALL ( SELECT DISTINCT salariu  
                           FROM angajati  
                          WHERE functie='PROGRAMATOR')  
          ORDER BY id_dep, nume;
```



Operatori in subcereri

✓ Operatorii EXISTS si NOT EXISTS

- Acesti operatori sunt folositi adesea in subcereri corelate si testeaza daca subcererea returneaza cel putin o valoare pentru EXISTS, sau niciuna in cazul lui NOT EXISTS, returnand TRUE sau FALSE.

Exemplu:

- Pentru a afla care departamente au cel putin un angajat, folosim cererea urmatoare:

```
SQL> SELECT d.id_dep, d.den_dep
      FROM departamente d
      WHERE EXISTS ( SELECT nume
                      FROM angajati
                     WHERE id_dep=d.id_dep)
      ORDER BY id_dep;
```



Operatori in subcereri

- Trebuie specificat ca o constructie cu EXISTS este mult mai performanta decat o constructie cu IN, SOME/ANY sau ALL, deoarece in cazul in care folosim tabele temporare acestea nu sunt indexate, ducand la scaderea considerabila a performantelor.
- Performanta depinde de folosirea indecsilor, de dimensiunea tabelelor din baza de date, de numarul de linii returnate de subcerere si daca sunt necesare tabele temporare pentru a evalua rezultatele returnate.
- Desi o subcerere cu o constructie pe operatorul NOT IN poate fi la fel de eficienta ca si in cazul unei constructii pe NOT EXISTS, cea din urma este totusi mult mai sigura, daca subcererea intoarce si valori NULL.



Operatori in subcereri

- In cazul operatorului NOT IN, conditia se evalueaza la FALSE cand in lista de comparatii sunt incluse valori NULL.

Exemplu:

- Daca vrem sa facem o lista cu angajatii care nu sunt sefi si folosim o constructie cu NOT IN, cererea nu va intoarce nicio inregistrare, ceea ce este fals:

```
SQL> SELECT id_dep, id_ang, nume, functie, id_sef  
      FROM angajati a  
     WHERE id_ang NOT IN ( SELECT DISTINCT id_sef  
                           FROM angajati)  
        ORDER BY id_dep;
```

- **no rows selected**



Actualizarea datelor prin subcereri

Exemple:

- Urmatoarea cerere actualizeaza comisionul angajatilor la 10% din salariul minim din departamentul din care fac parte, dar numai pentru angajatii care nu au primit comision:

SQL>UPDATE angajati a

```
SET a.comision=(SELECT min(salariu)*0.1 FROM angajati b  
                      WHERE a.id_dep=b.id_dep)  
                      WHERE nvl(comision,0)=0 ;
```

- Pentru a crea o tabela cu angajatii care au ecusonul mai mic decat cel mai mic ecuson de sef pe fiecare departament folosim comanda:

SQL> CREATE TABLE ecusoane_old (id_dep,id_sub,nume_sub, id_sef) as

```
SELECT id_dep, id_ang, nume,id_sef FROM angajati a  
                      WHERE a.id_ang <(SELECT min(b.id_sef)  
                      FROM angajati b WHERE b.id_dep = a.id_dep);
```

- Pentru a face o copie a tableei angajati executam comanda:

SQL> CREATE TABLE angajati_copy AS SELECT * FROM angajati;



Actualizarea datelor prin subcereri

- Daca vrem sa stergem angajatii care nu sunt sefi din tabela copie, executam urmatoarea comanda:

```
SQL> DELETE FROM angajati_copy
```

```
WHERE id_ang NOT IN (SELECT DISTINCT id_sef FROM angajati  
WHERE id_sef IS not null);
```

8 rows deleted.

Observatie:

- Daca nu se pune conditia in subcerere ca ecusonul sefului sa nu fie null, rezultatul va fi eronat:

```
SQL> DELETE FROM angajati_copy
```

```
WHERE id_ang NOT IN (SELECT DISTINCT id_sef FROM angajati);
```

0 rows deleted.

- Rezultatul eronat este din cauza ca exista un angajat care nu are sef, adica are valoarea NULL pentru ID_SEF (presedintele companiei).



Reguli in subcereri

- Cand folosim subcereri, trebuie sa respectam cateva reguli:
 - Cererea interioara trebuie sa fie inclusa intre paranteze si trebuie sa fie in partea dreapta a conditiei;
 - Expresiile din lista de expresii a subcererii trebuie sa fie in aceeasi ordine ca cele din lista din clauza WHERE a cererii principale (avand acelasi tip si numar de expresii);
 - Subcererile nu pot fi ordonate, deci nu contin clauza ORDER BY;
 - Clauza ORDER BY poate sa fie pusa la sfarsitul cererii principale;
 - Subcererile sunt executate de la cea mai adanca imbricare pana la nivelul principal de imbricare(cu exceptia cererilor corelate);
 - Subcererile pot folosi functii de grup si clauza GROUP BY ;
 - Subcererile pot fi inlantuite cu predicate multiple AND sau OR in aceeasi cerere externa;
 - In subcereri se pot folosi operatori de multimi;
 - Subcererile pot fi imbricate pana la nivelul 255.