13. NORMALIZAREA BAZEI DE DATE ȘI FORME NORMALE

I.OBIECTIVE

1. Forme normale ale bazelor de date
2. Normalizarea unei baze de date.Algoritmi de descompunere a schemelor de relații către FN3/FNBC

II. FUNDAMENTARE TEORETICĂ

2.1.FORME NORMALE

Dacă schema îndeplinește cerințele unui anumit set de condiții se spune că este în ***formă normală*** asociată acelui set.

Forma normală Boyce-Codd (FNBC)

***FN BC- Definiție*. Fie R o schemă de relație și F mulțimea de dependențe funcționale asociată. Se spune că R este în forma normală Boyce-Codd dacă și numai dacă oricare ar fi o dependență netrivială X $\rightarrow$Y din F , atunci X este supercheie pentru R.**

Rezultă că o schemă de relatie este in FNBC dacă și numai dacă fiecare dependență din F are în partea stânga o supercheie. Nu este necesar ca F să fie în forma canonică dar nu trebuie sa conțină dependențe triviale (obținute din prima axiomă - de reflexivitate, de tipul AB $\rightarrow$A sau AB $\rightarrow$AB)

Forma normala FN3

Pentru definiția formei normale 3 este necesară definirea noțiunii de ***atribut prim:*** fie R o schemă de relație și F mulțimea de dependențe funcționale asociată. Un atribut A din R se numeste atribut prim daca el aparține unei chei a lui R.

***FN 3- Definiție.* Fie R o schemă de relație si F mulțimea de dependențe funcționale asociată. Se spune că R este în FN3 daca și numai dacă oricare ar fi o dependență netriviala X $\rightarrow$ A din F atunci : X este supercheie pentru R sau A este atribut prim**

De remarcat că dacă în F avem dependențe care conțin mai multe atribute în partea dreaptă putem aplica regula de descompunere pentru a obține dependențe care în partea dreapta au câte un singur atribut.

Observație: Conditia de FNBC este inclusă în definiția FN3. Din acest motiv orice relație care este în FNBC este implicit și în FN3. Reciproca nu este adevarată. Rezultă de asemenea că dacă o schema de relație nu este în FN3 ea nu poate fi nici în FNBC.

Formele normale FN1 si FN2

Aceste forme normale nu garantează eliminarea anomaliilor deci ele nu sunt de dorit pentru schemele de relație ale unei baze de date a unei aplicații.

FN1-definiţie: **O relație R este in forma normală 1 (FN1) dacă pentru toate atributele sale există doar valori atomice ale datelor.**

Semnificația termenului 'atomic' este similară cu cea de la modelul entitate -asociere: valoarea respectivă este întotdeauna folosită ca un întreg şi nu se utilizează niciodată doar porțiuni din aceasta. De exemplu, dacă într-o relație continand date despre persoane avem atributul ***Adresa*** acesta este atomic dacă niciodată nu este nevoie să fie folosite doar anumite porțiuni ale sale (strada, numar, etc).

Fiind dată o relație R și mulțimea de dependențe asociată F ,putem defini încă două concepte:

Dependența parțială: O dependență funcțională X →A se numește ***dependenţă parţială*** dacă X este strict inclusă într-o cheie a relației R.

Dependență tranzitivă: O dependență funcțională X → A se numește ***dependența tranzitivă*** dacă X nu este inclusa în nici o cheie a relației R.

Pe baza acestor noțiuni putem defini forma normală 2:

FN2 -definiţie: **Fie R o schemă de relaţie şi F mulţimea de dependenţe funcţionale asociată. Se spune că R este în forma normală 2 (FN2) dacă și numai dacă F nu conține dependențe parțiale (*dar poate conține dependențe tranzitive).***

2.2.DESCOMPUNEREA SCHEMELOR DE RELAȚII

În cazul în care o relație din baza de date nu este într-o formă normală bună (FNBC, FN3) pot să apară diverse anomalii. Soluția este înlocuirea relației respective cu două sau mai multe relații care să conțină aceleași informații dar care, fiecare în parte, este în forma normală dorită . Procesul prin care se restructrează o relație in mai multe relații se numește ***descompunerea unei scheme de relație.*** Formal putem defini acest concept astfel:

Fie R o schemă de relatie, R = A1 A2 … A_m .

Se spune ca ρ = (R1, R2, …, R_n) este o ***descompunere*** a lui R dacă şi numai dacă :

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

Schemele R1, R2, …, R_n contin deci atribute din R, fiecare atribut Ai al schemei inițiale trebuind să se regăsească în cel puțin una dintre ele. *Nu este necesar ca schemele să fie disjuncte* (în practică ele au aproape întotdeauna atribute comune).

In exemplele de mai jos sunt prezentate câteva descompuneri valide ale unor scheme de relații, unele pot incorecte din punct de vedere al păstrării datelor și/sau dependențelor inițiale :

1.Fie relatia R = ABCDE având F = { A → B, B → A , A →C, D → E } .
Putem avea descompuneri ca:

$$\rho_1 = (ABC, DE),\ \rho 2 = (ABCD, DE),\ \rho_3 = (AB, CD, DE)$$

2.Fie relația Produse = IdP, NumeP, Cant, IdF, NumeF, AdresaF având dependențele funcționale:
F = { IdP→NumeP, IdP→Cant, IdP→ IdF, IdF→NumeF, IdF→AdresaF }

Putem avea o multitudine de descompuneri printre care:

ρ_1 = **(** (IdP, NumeP, Qty, IdF); (NumeF, AdresaF) **)**
ρ_2 = **(** (IdP, NumeP, Qty, IdF); (IdF, NumeF, AdresaF) **)**
ρ_3 = **(** (IdP, NumeP); (Qty, IdF); (NumeF, AdresaF) **)**

Descompunerea acționează deci la nivelul ***schemei*** relației. Ce se intamplă însă cu ***conținutul*** acesteia in cazul unei descompuneri? Fiecare relație rezultată va moșteni o parte dintre datele relației descompuse și anume proiecția acesteia pe mulțimea de atribute a relației rezultată din descompunere. Se poate constata din conținutul de date pentru diverse descompuneri dacă se pot reconstrui prin join sau alți operatori relaționali relația inițială. In cazul în care descompunerea nu s-a facut corect putem pierde datele relației inițiale sau dependențele funcționale ale relației inițiale.
În continuare sunt prezentați algoritmi prin care putem detecta dacă prin descompunere se pierd date sau dependențe.

A)Descompuneri fără pierderi de date

Condiția pentru a nu se pierde date prin descompunere este ca relația inițială să poată fi reconstruită exact prin **joinul natural** al relațiilor rezultate, fără tupluri în minus sau în plus.

Definiție: Fie R o schemă de relație, F mulțimea de dependențe funcționale asociată și o descompunere $\rho = (R1, R2, \ldots, R_n)$ a lui R. Se spune ca ρ este o descompunere ***cu join fără pierderi in raport cu F*** (prescurtat j.f.p.) dacă și numai dacă pentru orice instanță **r** a lui R care satisface dependențele F avem ca:

$$r_1 \;|><|\; r_2 \;|><|\ldots\; |><|\; r_n = r$$

unde $r_i = \pi_{Ri}(r)$

In exemplul anterior,doar descompunerea $\rho 2$ = **(** (IdP, NumeP, Qty, IdF); (IdF, NumeF, AdresaF)**)** are aceasta proprietate, în cazul celorlalte, *din cauza inexistenței coloanelor comune*, joinul natural nu se poate efectua. Faptul că o descompunere are sau nu această proprietate se poate testa pornind doar de la lista atributelor relației inițiale, lista atributelor relațiilor din descompunere și a mulțimii de dependențe funcționale asociată folosind următorul algoritm:

Algoritm de testare a proprietăţii de j.f.p. pentru o descompunere

Intrare: Schema de relaţie R = A1 A2 . . . A_m , mulţimea de dependențe funcţionale F și o descompunere $\rho = (R_1, R_2, ..., R_n)$
Ieșire: proprietatea j.f.p.
Metoda:
Se construieşte o tabelă având **n** linii si **m** coloane. Liniile sunt etichetate cu elementele descompunerii ρ iar coloanele cu atributele relației R. Elementul (i,j) al tabelei va fi egal cu a_j daca $A_j \in R$ sau b_{ij} altfel.
Se parcurg dependențele X →Y din F. Dacă două (sau mai multe) linii din tabelă au aceiași simboli pe coloanele X aceste linii se egalează și pe coloanele din Y astfel:

- Dacă pe o coloană din Y apare un a_j atunci toate elementele de pe acea coloana din liniile respective devin a_j
- Dacă pe o coloană din Y nu apare nici un a_j atunci se alege unul dintre elementele

de tip b_{ij} si toate elementele de pe acea coloană din liniile respective devin egale cu acel b_{ij}

Procesul se opreste:

- fie când s-a obținut o linie în tabelă care conține doar a-uri, caz în care descompunerea ρ **are** proprietatea j.f.p.
- fie când la o parcurgere a dependențelor nu mai apar schimbări în tabela și nu s-a obtinut o linie doar cu a-uri. In acest caz descompunerea ρ **nu are** proprietatea j.f.p.

În unele cazuri este fost suficientă o singură trecere prin dependențe. Există însă situații când sunt necesare mai multe treceri până procesul se oprește. In cazul în care descompunerea are doar două elemente se poate testa dacă are proprietatea de join fără pierderi și astfel:

Fie R o schemă de relație, F mulțimea de dependențe funcționale asociată și ρ= (R1, R2) o descompunere a sa. *Atunci ρ are proprietatea de join fără pierderi dacă una din dependențele următoare se poate deduce din F:*

$(R_1 \cap R_2) \rightarrow (R_1 - R_2)$ sau
$(R1 \cap R2 \rightarrow (R2 - R1)$

Pentru a testa daca dependentț se poate deduce din F este suficient să calculăm închiderea lui ($R_1 \cap R_2$). Dacă ea conține fie pe ($R_1 - R_2$) fie pe ($R_2 - R_1$) atunci descompunerea este cu join fără pierderi.

B)Descompuneri care păstrează dependențele

O a doua problemă în cazul descompunerii unei scheme de relație R având dependențele F în mai multe relații R_1, R_2, . , R_n este aceea a păstrării corelațiilor între date, corelații date de dependențele funcționale din F.

Fiecare relație R_i va moșteni o mulțime de dependențe dată de proiecția mulțimii de dependențe funcționale F pe Ri : $F_i = \pi_{Ri}(F)$

Fie relația Produse = (IdP, NumeP, Cant, IdF, NumeF, AdresaF 0
Având dependențele funcționale:

F = { IdP → NumeP, IdP → Cant, IdP → IdF, IdF → NumeF, IdF → AdresaF }

In cazul descompunerii ρ2 = (Rl, R2)

unde: R1 = (IdP, NumeP, Cant, IdF) și R2 =(IdF,NumeF, AdresaF)

cele două relații moștenesc următoarele dependențe:

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

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

După cum se observă toate dependențele relației inițiale sunt păstrate fie în F_{R1} fie în F_{R2}. Există însă și cazuri în care unele dependențe din F nu mai pot fi regăsite în mulțimile de dependențe asociate

schemelor din descompunere şi nu se pot deduce din acestea. In primul caz se spune că descompunerea păstrează dependenţele , iar in al doilea ca descompunerea nu păstrează dependenţele.

Definiţie: Fie R o schemă de relatie, F mulţimea de dependenţe funcţionale asociată, o descompunere $\rho = (R1, R2, \ldots, R_n)$ a lui R şi $Fi = \pi_{Ri}(F)$ mulţimile de dependenţe funcţionale ale elementelor descompunerii. Se spune ca ρ ***păstrează dependenţele*** din F , dacă şi numai dacă orice dependenţă din F poate fi dedusă din $\cup_{i=1..n} (F_i)$.
Rezultă că o descompunere păstrează dependenţele dacă şi numai dacă: $F \subseteq (\cup_{i=1..n} (F_i))^+$

Însă atât proiectia unei mulţimi de dependenţe cât şi incluziunea de mai sus implică un calcul de închidere a unei mulţimi de dependente (F si respectiv reuniunea mulţimilor Fi). Există şi în acest caz un algoritm pentru a testa daca o dependenţă este sau nu păstrată după descompunere fără a fi necesar efectiv calculul mulţimilor Fi.

Algoritm de testare a păstrării dependenţelor
Intrare: o schema de relaţie R, multimea de dependenţe funcţionale asociată F şi o descompunere $p = (R_1, R_2, . , R_n)$
Iesire: verdictul dacă p păstrează sau nu dependenţele
Metoda: Pentru fiecare dependenţă $X \rightarrow Y$ din F se procedează astfel:

- Se porneşte cu o mulţime de atribute Z = X
- Se parcurg repetat elementele descompunerii ρ. Pentru fiecare Ri se calculează o noua valoare a lui Z astfel : $Z = Z \cup ((Z \cap R_i)^+ \cap R_i)$
- Procesul se opreste în momentul când Z rămâne neschimbat la o parcurgere a elementelor R_i. Daca $Y \subseteq Z$ atunci dependenţa $X \rightarrow Y$ este păstrată, altfel nu e păstrată.

Dacă toate dependenţele din F sunt păstrate, înseamnă că ρ păstrează dependenţele din F.

C)Algoritmi de descompunere directă către FN3 sau FN-BC

Algoritmii de testare a păstrarii dependenţelor şi a joinului fără pierderi pot fi aplicaţi atunci când descompunerea unei scheme de relaţie se face 'de mână', pe baza experienţei pe care o are proiectantul bazei de date. Există însă şi niste algoritmi simpli care, pornind de la o schemă de relaţie şi mulţimea de dependenţe funcţionale asociată ne duc direct la o descompunere care este în FN3 sau FNBC şi în plus au proprietatea de join fără pierderi (deci nu se pierd date prin descompunere) şi/sau de păstrare a dependenţelor.

C1.Algoritm de descompunere in FN3 cu păstrarea dependenţelor

Fie R o schema de relaţie si F mulţimea de dependenţe funcţionale asociată, cu

$$F = \{ Xi \rightarrow Yi, X2 \rightarrow Y2, \ldots Xn \rightarrow Yn \}$$

Atunci descompunerea $\rho = (X1Y1, X2Y2, \ldots X_nY_n)$ este o descompunere in FN3 cu păstrarea dependenţelor.
Se observă din definiţia de mai sus a descompunerii ρ că:

- Toate dependenţele sunt păstrate: dependenţa $X \rightarrow Y_i$ este in proiectia lui F pe $X_i Y_i$
- Pentru a minimiza numărul de elemente din descompunere se aplică regula reuniunii: dacă avem mai multe dependenţe care au aceeaşi parte stângă le reunim într-una singură.
- Dacă în descompunere exista două elemente XiYi si XjYj astfel incat $XiYi \subseteq XjYj$ atunci XiYi se elimină.

Se poate demonstra faptul că fiecare schemă din descompunere este în **FN3.**

Exemplul 1.R = ABCDE, F = { A $\rightarrow$ B, A $\rightarrow$ C, A $\rightarrow$D, D $\rightarrow$ E } . Rescriem prin reuniune mulțimea de dependențe funcționale: F = { A $\rightarrow$BCD, D $\rightarrow$ E } . Rezultă din algoritm descompunerea ρ = (ABCD, DE)

Exemplul 2 Fie relația Produse =(IdP, NumeP, Cant, IdF, NumeF, AdresaF)având dependențele funcționale:

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

Rescriem mulțimea de dependențe. Rămân doar două dependențe:

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

Descompunerea în FN3 cu păstrarea dependențelor va fi:

ρ = ((IdP, NumeP, Cant, IdF), (IdF, NumeF, AdresaF))

C2.Algoritm de descompunere in FN3 cu păstrarea dependențelor și join fără pierderi

Dacă la descompunerea obtinuta prin algoritmul anterior *adaugăm o cheie a relației* (ca element al descompunerii) vom obține o descompunere care are **atât proprietatea de join fără pierderi** cât si pe cea a **păstrării dependențelor**. Formal putem scrie algoritmul astfel:

Algoritm. Fie R o schema de relație și F mulțimea de dependențe funcționale asociată, cu
F = { Xi $\rightarrow$ Yi, X2 $\rightarrow$Y2, … Xn $\rightarrow$ Yn }
și X o cheie pentru R

Atunci descompunerea ρ = (X, X1Y1, X2Y2, … X_nY_n) este o descompunere in FN3 cu păstrarea dependențelor și join fara pierderi.

Păstrarea dependențelor este evidentă, ca mai sus. Dacă vreunul dintre elementele de forma XiYj conțin deja o cheie a lui R atunci nu este necesară adăugarea unui element suplimentar în descompunere.

Exemplul 3 : Pentru relațiile din exemplele de mai sus descompunerea rămâne aceeași deoarece:

- In cazul relatiei R = ABCDE cheia este A, deja inclusă în ABCD, deci descompunerea rămâne ρ = (ABCD, DE).
- In cazul relației PRODUSE de asemenea cheia este IdP, inclusă deja într-unul din elementele descompunerii.

C3.Algoritm de descompunere în FNBC cu join fără pierderi

Fie R o schemă de relație și F mulțimea de dependențe funcționale asociată, F în forma canonică: F = { $X_1 \rightarrow A_1$, $X_2 \rightarrow A_2$, . $X_n \rightarrow A_n$ } . Putem calcula descompunerea în FNBC cu join fără pierderi iterativ:

- Inițial ρ = (R)

- La fiecare pas se alege o schemă T care conține o dependență de forma X □A care violează condițiile de FNBC. Schema respectivă este înlocuită în ρ prin T_1 si T_2 unde T = XA si T_2 = T - {A}
- Procesul se opreste când în p nu mai există elemente care nu sunt în FNBC

D)Definiții funcționale pentru FORMELE NORMALE

O relație este normalizată în prima formă normală (FN1) dacă fiecare atribut ia numai valori atomice și scalare din domeniul său de definiție.

O relație este normalizată în a doua formă normală (FN2) în raport cu o mulțime de dependențe funcționale *F*, dacă este în FN1 și dacă in *F+* nu există nici o dependență funcțională parțială a unui atribut neprim față de o cheie a relației.

Ca exemplu de normalizare in FN2, se va studia relația cu schema AP(idAngajat, Nume,Prenume,Adresa, IdProiect,ore) si cu multimea dependentelor functionale F_{AP}:

F_{AP} = {IdAngajat→Nume,IdAngajat→Prenume,

IdAngajat→Adresa,{IdAngajat,IdProiect} →Ore}

Cheia primară a relației este {IdAngajat,IdProiect} și se poate deduce din mulțimea dependențelor funcționale. Se presupune că toate atributele iau valori atomice și scalare, deci relația este în FN1. Atributele prime sunt IdAngajat, IdProiect iar toate celelalte atribute sunt neprime.

Se observă ca dependența funcțională {IdAngajat, IdProiect}→Nume este parțială, datorită faptului ca submulțimea {IdAngajat} a atributului compus cheie primară determină funcțional atributul Nume: IdAngajat→Nume.

Rezultă că relația AP nu este în FN2. Intr-o astfel de relație există date redundante (de exemplu, numele, prenumele și adresa unui angajat se înregistreaza pentru fiecare proiect la care lucrează angajatul) care produc anomalii de actualizare.

Relatia AP se poate descompune in relațiile A(IdAngajat,Nume,Prenume,Adresa) și P(IdAngajat, IdProiect,Ore), care sunt în FN2 (de fapt, sunt chiar in FNBC, se poate verifica usor acest lucru) si nu prezinta redundanța a datelor și anomalii de actualizare a tuplurilor.

O relatie este normalizata in a treia forma normala (FN3) in raport cu **o mulțime de dependențe functionale *F* daca este în FN2 si daca in *F+* nu există nici o dependență funcțională netrivială a unui atribut neprim față de alt atribut neprim al relației.**

Orice relație formată din două atribute este în FN3 deoarece ea se află în FN2 (s-a demonstrat în secțiunea precedentă), și *nu poate exista nici un atribut neprim care să determine funcțional un alt atribut neprim,* deoarece o relație cu două atribute nu poate avea decât cel mult un atribut neprim.

Ca exemplu de normalizare a unei relații in a treia forma normală, se consideră schema relației

AFS(IdAngajat, Nume, Prenume, Adresa, Functie, Salariu),

cu mulțimea dependențelor funcționale :

F_{AFS} = {IdAngajat→Nume, IdAngajat→Prenume, IdAngajat→Functie, Functie→Salariu}.

Cheia primară a relației este atributul ***IdAngajat***, si ea poate fi dedusă din mulțimea dependențelor funcționale.

Se consideră ca fiecare atribut ia numai valori atomice și scalare, deci relația este in FN1. Primele patru dependente functionale sunt dependențele functionale totale ale unor atribute neprime față de cheia primară a relatiei, deci relația este în FN2.

Dependența funcțională **(***Functie→Salariu)* semnifică faptul că în instituția respectivă toți salariații cu aceeasi funcție au același salariu (adică funcția determiă salariul, ceea ce este plauzibil). Aceasta dependență funcționala a atributului neprim *Salariu* fata de alt atribut neprim *(Functie)*, arată că relația nu este în a treia forma normală (FN3).

Chiar dacă relația AFS este in FN2, în aceasta încă mai există redundanta a datelor, deoarece valoarea salariului corespunzător unei funcții se înregistrează de mai multe ori, pentru fiecare salariat care deține acea funcție. Astfel de redundanțe și anomaliile de actualizare pe care le provoacă se pot elimina dacă se descompune relația în două (sau mai multe) relații, care să nu conțină date redundanțe.

Relația AFS se poate descompune în relațiile :

AF(IdAngajat,Nume,Prenume,Adresa,Functie) si
FS(Functie,Salariu)**.**

Se poate verifica usor că relațiile rezultate sunt în FN3.

O relație cu schema R este în forma normală Boyce-Codd (FNBC) în raport cu o mulțime *F* de dependente funcționale dacă este în FN1 și dacă, pentru orice dependență funcțională netrivială *X→Y* din *F*+,*X* este o cheie a relației *R*.

Se poate observa asemănarea dintre formele normale FN3 si FNBC, ambele impunând condiția ca *atributul care determină funcțional alte atribute să fie o cheie a relației*. Forma normală Boyce-Codd este mai restrictivă decat FN3, deoarece în FNBC se impune aceasta condiție *tuturor* atributelor, prime sau neprime, iar pentru FN3, condiția se impune numai atributelor neprime. Este evident faptul ca o relație în FNBC este, de asemenea în FN3, dar o relație în FN3 poate să fie sau nu în FNBC.

Ca exemplu de normalizare in FNBC, se considera relatia EDP(IdElev, IdDisciplina, IdProfesor), cu cheia candidată {IdElev,IdDisciplina} și cu mulțimea F_{EDP} a dependențelor funcționale:

F_{EDP} = {{IdElev,IdDisciplina} →IdProfesor,IdProfesor→IdDisciplina}

Aceste dependențe funcționale semnifică o anumită organizare a activității de instruire a elevilor: (a) fiecare elev are un singur profesor la o disciplina (de regula) si (b) un profesor predă o singură disciplină (plauzibil).

Se consideră că atributele iau valori atomice și scalare, deci relația este în FN1. Din muțtimea dependențelor funcționale se observa ca nu exista dependențe funcționale parțiale fata de cheia relatiei (deci relatia este in FN2) și nu există nici o dependență funcțională a unui atribut neprim față de un alt atribut neprim, deci relația EDP este în FN3. Totuși, relația EDP nu este in FNBC, datorită dependenței funcționale a atributului prim *IdDisciplina* față de atributul neprim *IdProfesor***.**

O relație care nu este în FNBC prezintă, ca orice relație incomplet normalizată, redundanța datelor și anomalii de actualizare. De exemplu, în relația EDP, pot exista mai multe tupluri care conțin o anumită

valoare a atributului *IdProfesor* (deoarece mai multi elevi studiază cu acelaşi profesor) şi, de fiecare dată, este înregistrată disciplina (atributul IdDisciplina) pe care o predă profesorul respectiv. Din această redundanță a datelor rezultă mai multe anomalii de inserare, ştergere şi actualizare a tuplurilor.

Normalizarea relației EDP astfel încât relațiile obținute să fie în FNBC se poate face prin descompunerea acesteia. Se pot incerca trei descompuneri:

D_1 = {EP,PD}: EP ={IdElev,IdProfesor},PD={IdProfesor,IdDisciplina};

D_2 = {ED,PD}: ED ={IdElev,IdDisciplina},PD ={IdProfesor,IdDisciplina};

D_3 = {EP,ED}: EP = {IdElev,IdProfesor}, ED= {IdElev,IdDisciplina}.

Se observă că relațiile rezultate în oricare din aceste descompuneri sunt relații in FNBC (fiind relații formate din doua atribute) şi ca, în oricare din aceste descompuneri, se pierde dependenta functionala **{**IdElev,IdDisciplina}—> IdProfesor**.** Rezultă că relatia EDP nu poate fi descompusă în mod reversibil iîn relatii FNBC şi că, pentru respectarea tuturor constrângerilor dintr-o relație ca relația EDP, sunt necesare măsuri speciale.

Analiza normalizării relațiilor trebuie să fie realizată pentru orice proiect de baze de date, pentru a asigura funcționarea corectă a acesteia. După ce se decide forma normală a unei relații, este necesar să se prevadă procedurile de verificare a tuturor constrângerilor explicite rămase: fie dependențe de date care nu sunt determinate de chei ale relației (într-o relație cu o formă de normalizare scazută), fie dependențele funcționale pierdute prin descompunerea relației, care devin constrângeri explicite între relațiile rezultate prin descompunere.

Dacă o relație se păstrează într-o formă de normalizare mai redusă, atunci trebuie să se prevadă *proceduri de verificare şi impunere a dependențelor de date care nu sunt determinate de chei ale relației sub forma de constrangeri explicite.(de ex. triggere).*

III.PROBLEME REZOLVATE

3.1. Relaţia Produse = IdP, NumeP, Cant, IdF având asociată mulţimea de dependențe funcționale $F_{PRODUSE} = \pi_{PRODUSE}(F)$ ={ IdP $\rightarrow$NumeP, IdP $\rightarrow$Cant, IdP $\rightarrow$IdF} este în forma normală Boyce-Codd: cheia unică a relației este IdP si toate dependențele au ân partea stângă o supercheie (asa cum s-a menționat orice cheie este în același timp şi supercheie)

3.2. Relația Produse = IdP, NumeP, Cant, IdF, NumeF, AdresaF având dependențele:
F = { IdP $\rightarrow$NumeP, IdP $\rightarrow$Cant, IdP $\rightarrow$ IdF, IdF $\rightarrow$NumeF, IdF $\rightarrow$AdresaF } nu este în forma normala Boyce-Codd: cheia unică este IdP dar exista dependențe care nu au în partea stângă o supercheie: IdF $\rightarrow$NumeF, IdF $\rightarrow$AdresaF

3.3. Relația R = ABCD avand F = { AB $\rightarrow$ C, AB $\rightarrow$D, D $\rightarrow$ A } are cheia unica AB.

- Relația este in FN3 deoarece primele două dependențe au în partea stanga o supercheie (AB) iar a treia dependență are în partea dreaptă atributul prim A.
- Relația nu este în FNBC deoarece a treia dependență violează definiția pentru această formă normală (nu are în partea stanga o supercheie).

3.4. Fie relația R = ABCDE având F = { A $\rightarrow$ B, B $\rightarrow$ A , A $\rightarrow$C, D $\rightarrow$E } are cheile AD şi BD. Rezulta că:

- R nu este în FN3 deoarece dependențele 3 si 4 nu au nici supercheie în partea stangă nici atribut prim în partea dreaptă
- R nu e în FNBC deoarece nu e în FN3

3.5.Relația R = ABCD având F = { A →B, B →C, C →D, D → A } are cheile A, B, C si D. Rezultă:

- R este în FNBC deoarece în partea stangă a dependențelor sunt numai superchei
- R este în FN3 deoarece este în FNBC

3.6. Fie R = ABCDE, F = { A →B, A → C, A → D, D →E } și o descompunere a lui R , ρ = (ABCD, DE)

Construim tabelul din algoritm:

	A	B	C	D	E
ABCD	**Al**	**a2**	**a3**	**a4**	**~~b15~~ a5**
DE	**b21**	**b22**	**b23**	**a4**	**a5**

La prima parcurgere, pentru dependențele A → B , A →C , A →D nu găsim două linii cu aceleași valori pe coloana A dar pentru dependența D→ E cele două linii sunt egale pe coloana D (simbolul a4). Le egalăm si pe coloana E: cum pe această coloană exista a5 rezulta ca b15 devine egal cu a5. S-a obținut o linie numai cu a-uri, deci descompunerea are proprietatea de join fără pierderi.

3.7. Fie relația R = ABCDE şi F = { A →B, AC → D, D →E } şi descompunerea ρ = (AB, BC, CDE). Tabelul este următorul

	A	B	C	D	E
AB	**Al**	**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 descompunerea propusă nu are proprietatea j.f.p.

3.8. Fie R = ABCDE,
F = { C →E (1), A → C (2), B →D (3), D→ E (4), E→ B (5) } (cu dependențe numerotate între paranteze) si ρ = (BCE, AB, ACD)

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

Prima trecere:

- Din C → E b35 devine a5
- Din A → C b23 devine a3
- Din B →D b24 devine b14
- Din D →E b25 devine a5

- Din E $\rightarrow$B b32 devine a2

Am obținut încă o linie doar cu a-uri. Descompunerea are proprietatea de j.f.p.

3.9. In prima exemplificare a aplicării algoritmului de testare aveam o descompunere cu două elemente:
R= ABCDE, F = { A $\rightarrow$B, A $\rightarrow$ C, A $\rightarrow$ D, D $\rightarrow$E }
si ρ = (ABCD, DE).

Avem R_1 = ABCD, R_2 = DE

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

Cele două dependențe sunt:
($R_1 \cap R_2$) $\rightarrow$ (R_1 - R_2) devine D $\rightarrow$ ABC
($R_1 \cap R_2$) $\rightarrow$ (R_2 - R_1) devine D $\rightarrow$E

Ultima este chiar o dependență din F deci se poate deduce din F , *deci* ρ *are proprietatea de join fara pierderi.*

3.10. Fie R = ABCDE, ρ = (BCE, AB, ACD) si mulțimea de dependențe funcționale

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

Se observă că dependențele C $\rightarrow$E, A $\rightarrow$C si E $\rightarrow$ B sunt păstrate: ele aparţin proiecţiei lui F pe BCE (prima şi ultima) şi ACD (a doua). Rămân de testat dependențele B $\rightarrow$D şi D $\rightarrow$E. Să aplicăm algoritmul pentru B -- D:

Inițial Z = B
Trecerea 1 prin elementele lui ρ:
Pentru BCE: Z = B $\cup$ ((B $\cap$ BCE)$^+$ $\cap$BCE) = B $\cup$ (BDE $\cap$ BCE) = BE
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 $\cup \varnothing$ = BE.

La următoarea trecere Z rămâne neschimbat şi procesul se opreste. Cum {D} $\not\subset$BE rezultă că dependența B $\rightarrow$ D nu este păstrată , deci ρ nu păstrează dependențele.

3.11. Fie schema de relație R = ABCD, F = { A $\rightarrow$ B, A $\rightarrow$ C, C $\rightarrow$ D, D $\rightarrow$ A}
şi o descompunere ρ = (ABC, CD). Trebuie sa testăm dacă D $\rightarrow$ A este păstrată (celelalte dependențe se regăsesc direct în proiecțiile lui F pe elementele descompunerii).

Inițial Z = D
Prima trecere prin elementele lui ρ:
Pentru ABC: Z = D $\cup$ ((D $\cap$ ABC)$^+$ $\cap$ ABC) = D $\cup \varnothing$= D
Pentru CD: Z = D $\cup$ ((D $\cap$CD)$^+$ $\cap$ CD) = D $\cup$ (ABCD $\cap$ CD) = CD
A doua trecere prin elementele lui □:
Pentru ABC: Z = CD $\cup$ ((CD $\cap$ ABC)$^+$ $\cap$ ABC) = CD $\cup$ (ABCD $\cap$ ABC) =

ABCD. Stop. Am obținut ca $A \subseteq Z$, deci dependența $D \rightarrow A$ este păstrată, deci ρ păstrează dependențele.

3.12: Fie R = ABCDE, F = { $A \rightarrow B$, $B \rightarrow A$, $A \rightarrow C$, $D \rightarrow E$ } . Cheile relației sunt AD și BD.

- Rescriem mulțimea de dependențe: F = { $A \rightarrow BC$, $B \rightarrow A$, $D \rightarrow E$ } .
- Rezultă descompunerea cu păstrarea dependențelor: ρ = (ABC, AB, DE). Cum AB e inclus în ABC rezulta în final ρ = (ABC, DE).
- Cum elementele descompunerii nu conțin vreo cheie a lui R, o adaugăm. Obținem în final descompunerea ρ = (AD, ABC, DE).

3.13: Fie relația R = ABCD cu F = { $AB \rightarrow C$, $AB \rightarrow D$, $D \rightarrow A$ } . Cheia relației este AB. Relația *este în FN3 dar nu este în FNBC* din cauza dependenței $D \rightarrow A$, care nu are în partea stangă o supercheie a lui R.

- Inițial: ρ = (R) = (ABCD)
- Alegem dependența $D \rightarrow A$ care violează condiția de FNBC.
- Inlocuim T = ABCD cu T_1 = DA si T_2 = ABCD - A = BCD.
- T_1 moșteneste de la T dependența $D \rightarrow A$, cheia va fi D si T_1 e in FNBC
- T_2 moșteneste de la T dependența { $DB \rightarrow C$ } . Cheia va fi DB si T_2 e în FNBC.
- Rezultă că descompunerea ăn FNBC cu join fără pierderi este p = (AD, BCD).

Observații:

1. Dependența moștenită de T_2 este din F^+. Ea se deduce astfel: Din $D \rightarrow A$ prin augmentare cu B obținem $DB \rightarrow AB$ și împreună cu dependența $AB \rightarrow C$, prin tranzitivitate obținem $DB \rightarrow C$. Analog din $AB \rightarrow D$ se deduce $DB \rightarrow D$ dar aceasta este o dependență trivială (partea dreapta e inclusă în cea stângă).
2. In multe cazuri este nevoie de mai multe iterații, relațiile de tip T_2 (egale în algoritm cu T — A) nefiind uneori în FNBC. Ele se descompun din nou în acelasi mod.

IV.PROBLEME PROPUSE

4.1.Să se identifice cheile relației R(ABCDEFG) ce posedă următoarele dependențe functionale:

$AB \rightarrow C$, $CD \rightarrow E$, $EF \rightarrow G$, $FG \rightarrow E$, $DE \rightarrow C$, si $BC \rightarrow A$

$AEF \rightarrow C$, $BF \rightarrow C$, $EF \rightarrow D$, and $ACDE \rightarrow F$.

4.2.Care din relații este în FNBC?

a. R(ABCD) și dependențele funcționale : $BD \rightarrow C$; $AB \rightarrow D$; $AC \rightarrow B$; $BD \rightarrow A$
b. R(ABCD) și dependențele funcționale : $C \rightarrow B$; $BC \rightarrow A$; $A \rightarrow C$; $BD \rightarrow A$

4.3.Care din relații este în FN3?

c. R(ABCD) și dependențele : $ACD \rightarrow B$; $AC \rightarrow D$; $D \rightarrow C$; $AC \rightarrow B$
d. R(ABCD) și dependențele : $ABD \rightarrow C$; $CD \rightarrow A$; $AC \rightarrow B$; $AC \rightarrow D$
e. R(ABCD) și dependențele : $A \rightarrow B$; $B \rightarrow A$; $A \rightarrow D$; $D \rightarrow B$
f. R(ABCD) și dependențele : $AB \rightarrow C$; $ABD \rightarrow C$; $ABC \rightarrow D$; $AC \rightarrow D$

g. R(ABCD) FD's: AB → C ; BCD → A ; D → A ; B → C
h. R(ABCD) FD's: B → C ; AC → D ; ABD → C ; BCD → A
i. R(ABCD) FD's: C → B ; A → B ; CD → A ; BCD → A
j. R(ABCD) FD's: C → B ; B → A ; AC → D ; AC → B

4.4.Să se descompună următoarele relații într-un set de relații in FN3 , menținând toate dependențele funcționale :

R(ABCD) FD's: AB → C ; ABD → C ; ABC → D ; AC → D
R(ABCD) FD's: AB → C ; BCD → A ; D → A ; B → C
R(ABCD) FD's: B → C ; AC → D ; AB → D; BD → A
R(ABCDE) FD's: D → C, D → E, BC → A, BC → D, BCD → A
R(ABCDEF), FD's: B→ F, BCD→ A, C→ D, D→ B, EF→ C, AC → B

4.5.Fie următoarea schemă de relații: R= {A,B,C,D,E,G},{C→D,A→G,AB→E,G→B,AD→B})

a) Care este cheia/cheile relatiei? Argumentati
b) Normalizati pana la cel mai inalt nivel posibil cu/fara mentinerea dependentelor functionale.

4.6.Fie următoarea schemă R=({A,B,C,D,E,F}, {A→BE,AE→BD,F→CD,CD→BEF,CF→B})

a) Sa se gaseasca toate cheile candidate ale lui R si sa se argumenteze ca sunt chei candidate.
b) Normalizați până la cel mai înalt nivel cu menținerea dependențelor funcționale.

4.7. Fie schema de relație:R=({A,B,C,D,E,G}, {A→BC, BE→G,G→CD,AD→BG,AE→G})

a) Gasiti cheile candidate ale relatiei R si argumentati
b) Normalizați până la cel mai înalt nivel, cu/fără mentinerea dependentelor functionale.

4.8.Se consideră schema de relatii R=({A,B,C,D,E,G}, {AC→D, BC→EG,D→B,AD→CB,ABD→E})

a) Sa se identifice cheile candidate ale relatiei R , argumentând.
b) In ce formă normală se afla relatia R? Transformați schema de relație în forma normală 3 și argumentați