Baze de date

Universitatea “Transilvania” din Brasov

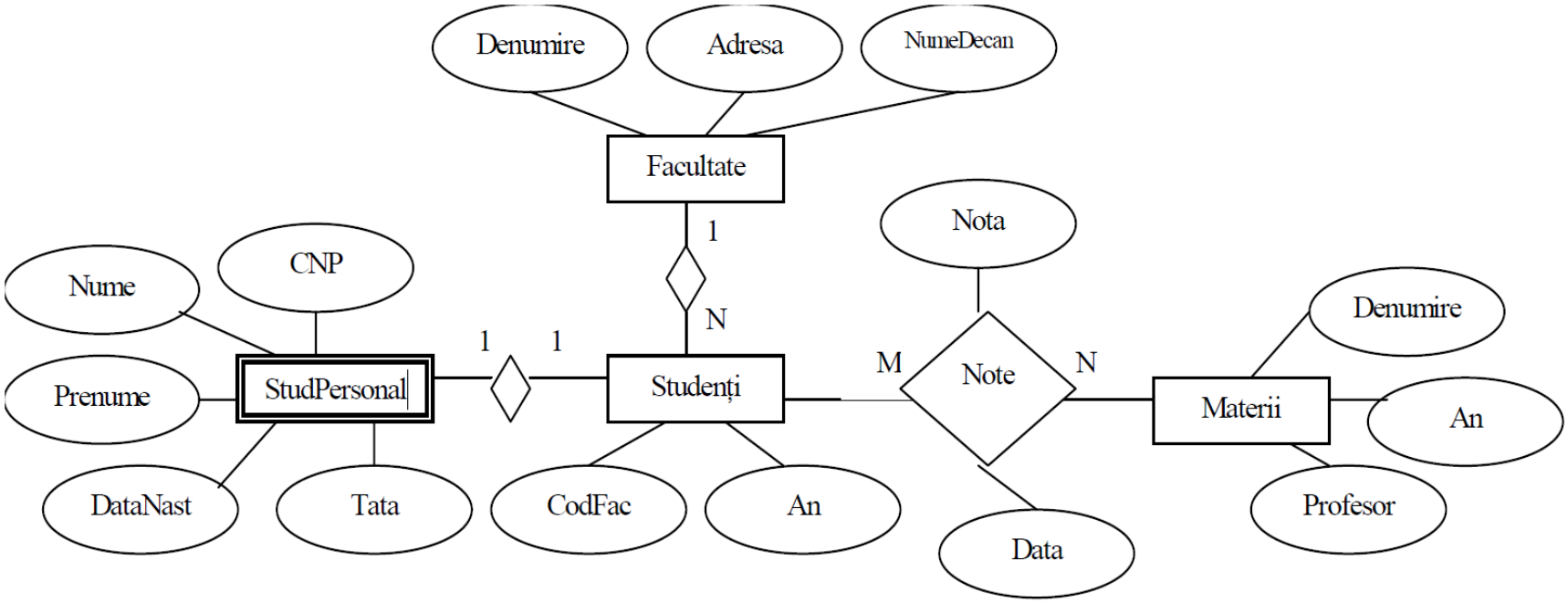
Lect.dr. Costel Aldea

costel.aldea@gmail.com

Normalizarea

- La proiectarea bazelor de date relaţionale se stabilesc entităţile din realitatea modelată.
- Modul în care se pot stabili entităţile unei baze de date nu este unic şi de accea este necesar să existe criterii de evaluare a calităţii entităţilor, astfel încât acestea să asigure integritatea datelor.
- Procesul de normalizare propus de E.F. Codd în 1970 urmăreşte execuţia asupra unei tabele a unor serii de teste pentru a cerceta apartenenţa la forma normală.
- Codd propune trei forme normale (3NF), cea mai bună definiţie fiind dată mai târziu de Boyce şi Codd, fiind cunoscută sub numele de forma normală Boyce-Codd.

Model conceptual - bd. universitate



Forma normală

- Normalizarea datelor este un proces în timpul căruia tipurile de entitati nesatisfăcătoare sunt descompuse prin împărţirea atributelor în tabele cu atribute mai puţine ce posedă proprietăţile dorite.
- Forma normală oferă proiectantului bazei de date :
  - un schelet formal pentru analiza relaţiilor bazat pe chei şi pe dependenţa funcţională între atribute
  - serie de teste ce pot elimina tabelele individuale astfel încât baza de date relaţională poate fi normalizată în orice grad.
- Când un test nu este trecut, tabela va fi descompusă în tabele ce trec testele de normalitate

Forma normală de ordin 1 (FN1)

- Forma normală de ordin 1 este considerată ca fiind parte a definiţiei formale a unei tabele.
- Aceasta nu permite atribute cu mai multe valori, atribute compuse sau combinaţii ale lor. Aceasta stabileşte ca domeniul atributelor trebuie să includă numai valori atomice şi valoarea oricărui atribut într-un tuplu este o valoare unică în domeniul atributului respectiv.
- FN1 nu permite un set de valori, un tuplu de valori sau o combinaţie a acestora ca valoare a unui atribut pentru un tuplu.
- FN1 nu permite tabele în tabele sau tabele ca atribute ale tuplurilor. Valorile permise de FN1 sunt atomice sau indivizibile, pentru un domeniu specificat de valori.

Exemplu FN1 (1)

- Considerăm că în tabela

  Materii (CodMaterie, Denumire, An, NumeProfesor),

- unde cheia primară este CodMaterie este introdusă o înregistrare ca in exemplul de mai jos.

CodMaterie	Denumire	An	NumeProfesor
1	Analiză matematică	1	O.Stanăşilă, P.Flondor, M.Olteanu

Exemplu FN1 (2)

- Această înregistrare reprezintă o disciplină care este predată de trei profesori diferiţi.
- Aceasta înregistrare nu îndeplineşte FN1, deoarece pentru atributul NumeProfesor nu sunt valori atomice, ci un set de valori.
- Pentru a rezolva această problemă vom introduce mai multe înregistrări, care vor îndeplini cerinţele FN1, considerând trei materii diferite astfel:

CodMaterie	Denumire	An	NumeProfesor
1	Analiză matematică	1	O.Stanăşilă
2	Analiză matematică	1	P.Flondor
3	Analiză matematică	1	M.Olteanu

Forma normală de ordin 2 (FN2)

- A doua formă normală impune ca fiecare atribut (coloană) să fie dependent de fiecare parte a cheii principale.
- O tabelă îndeplineşte FN2 dacă îndeplineşte FN1 şi conţine numai atribute care dau informaţii despre cheia tabelei.

Exemplu FN2

- Considerând că în tabela Materii (CodMaterie, Denumire, An, NumeProfesor), unde cheia primară este CodMaterie că ar mai exita şi alte câmpuri cum ar fi: Nume Student, Nota, DataExaminării. Acea structură nu ar fi proiectată bine, neîndeplinind FN2. Pentru a soluţiona această problemă trebuie împărţită acea tabelă în mai multe, astfel:

Studenti(CodStud, Nume, ….)

Materii (CodMaterie, Denumire, An, NumeProfesor),

Note(Nota, Data, CodStud, Cod Materie)

Forma normală de ordin 3 (FN3)

- Pentru a ajunge la a treia formă normală, tabelul trebuie să fie deja în prima şi a doua formă normală. Pentru a fi în a treia formă normală, trebuie ca toate câmpurile non-primare să depindă numai de câmpurile primare.
- Dependenţă tranzitivă: Dacă atributele A, B, C sunt în relaţiile A->B şi B->C, atunci spunem că atributul C este dependent tranzitiv de atributul A, via B.
- Ca regula de stil pentru normalizare, de obicei nu este recomandabil să includeţi câmpuri care pot fi derivate din alte câmpuri situate în acelaşi tabel sau în tabelele aflate în relaţie.

Exemplu FN3

- Fie tipul de entitate

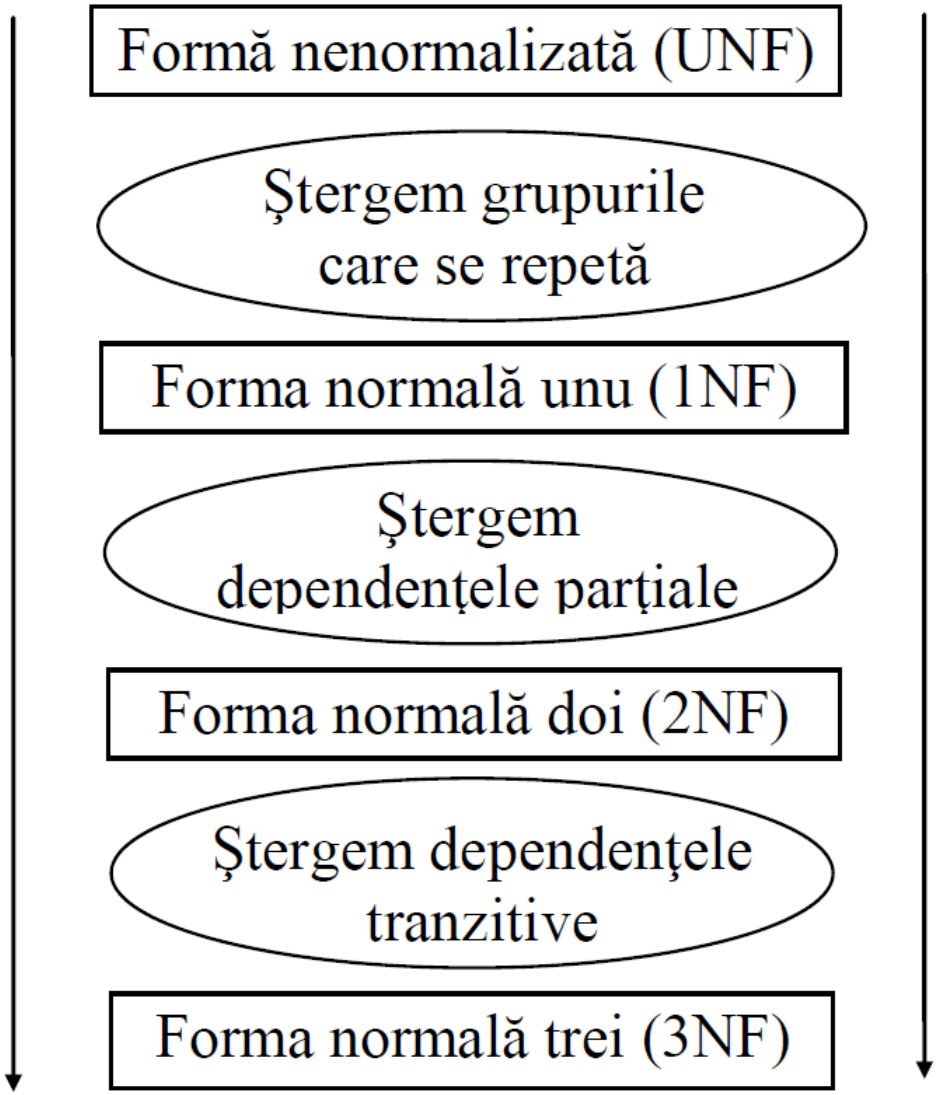
StudPersonal (CodStud, CNP, Nume, Init, Prenume, DataNasterii, LocNaşt, Tata, Mama, Adresa)

- Nu are rost să stocăm un alt câmp numit Vârsta, care se poate calcula din DataNasterii.

Forma normală Boyce-Codd (FNBC)

- Forma normală Boyce-Codd este o formă strictă FN3,
- fiecare tabelă FNBC este în acelaşi timp o tabelă FN3, cu toate că o tabelă FN3 nu este în mod necesar şi o tabelă FNBC.
- Cele două forme sunt asemănătoare, ambele impunând condiţia ca atributul care determină funcţional alte atribute să fie o cheie a tabelei.
- Forma normală Boyce-Codd este mai restrictivă decât FN3, deoarece în FNBC se impune această condiţie tuturor atributelor, prime sau neprime, pe când în FN3 condiţia se impune numai atributelor neprime.
- Atributele prime sunt atributele care aparţin unei chei, iar celelalte se numesc atribute neprime.
- Orice tabelă formată din două atribute este FNBC, FN2 şi FN3.

Normalizare



Dependentele de date

- **Dependenţele de date** (data dependencies) reprezintă constrângeri care se impun valorilor atributelor unei relaţii şi care determină proprietăţile relaţiei în raport cu operaţiile de inserare, ştergere şi actualizare a tuplurilor.
- **O formă normală a unei relaţii** (normal form) presupune anumite condiţii pe care le îndeplinesc valorile atributelor şi dependenţele de date definite pe acea relaţie
- Dependentele de date:
  - Dependente functionale: E.F. Codd a propus trei forme normale: FN1, FN2, FN3; apoi a fost introdusă forma normală Boyce-Codd (FNBC)
  - Dependenţelor multivalorice: forma normala 4 (FN4)
  - Dependenţelor de joncţiune: forma normala 5 (FN5)

- Formele normale ale relaţiilor formează o colecţie ordonată (FN1, FN2, FN3, FNBC, FN4, FN5), şi ele impun condiţii din ce în ce mai restrictive asupra dependenţelor de date
- Ordonarea formelor normale de la FN1 la FN5 înseamnă că orice relaţie aflată în FN2 este în FN1, orice relaţie în FN3 este în FN1 şi FN2 etc.
- *Normalizarea relaţiilor* (normalization) constă în descompunerea lor, astfel încât relaţiile să fie in forme normale cât mai avansate

Redundanta datelor si anomaliile de actualizare

- In relația AngajatProiect, cu cheia primara PK={IdAngajat, IdProiect} sunt valori redundante ale atributelor: Nume, Prenume, Adresa
- *Anomalii de inserare*: nu se pot introduce date despre un angajat (numele, prenumele, adresa) dacă nu există cel puţin un proiect la care acesta să lucreze
- *Anomalii de ştergere*: dacă se şterg toate tuplurile referitoare la un anumit, se pot pierde toate datele referitoare la acei angajaţi care lucrează doar la proiectul respectiv
- *Anomalii de actualizare*: dacă se modifică într-un tuplu valoarea unuia din atributele care au valori redundante, starea relaţiei poate deveni inconsistentă

IdAngajat	Nume	Prenume	Adresa	IdProiect	Ore
1	Ionescu	Ion	Bucuresti	P1	100
2	Popescu	Petre	Ploiesti	P1	80
3	Marinescu	Marin	Craiova	P1	200
1	Ionescu	Ion	Bucuresti	P2	100
2	Popescu	Petre	Ploiesti	P2	120

Eliminarea anomaliilor datorate redundantei

Eliminarea anomaliilor provocate de redundanta datelor se poate face:

- Fie prin prevederea unor proceduri stocate (sau triggere) care sa verifice corectitudinea fiecarei operatii de actualizare a relatiilor
- Fie prin descompunerea relatiilor care prezinta redundante in relatii mai simple; exemplu: descompunerea relatiei AP in 2 relatii, A si P

A

IdAngajat	**Nume**	**Prenume**	**Adresa**
1	Ionescu	Ion	Bucuresti
2	Popescu	Petre	Ploiesti
3	Marinescu	Marin	Craiova

P

IdAngajat	***IdProiect***	**Ore**
1	P1	100
2	P1	80
3	P1	200
1	P2	100
2	P2	120

- Relatiile A si P au un grad de normalizare mai ridicat si nu mai au anomalii
- Dar unele interogari sunt mai ineficiente in A si P decat in AP

Exemplu: “Care este numărul de ore lucrate de Ionescu la proiectul P1 ?”:

$Q1 = \Pi_{Ore}\, \sigma_{Nume='Ionescu' \text{ AND } IdProiect='P1'}\ (AP)$

$Q2 = \Pi_{Ore}\, \sigma_{Nume='Ionescu' \text{ AND } IdProiect='P1'}\ (A \bowtie P)$

Dependente functionale

- ***Dependentele funcţionale*** (*functional dependencies*) sunt constrângeri in relatii, prin care valoarea unui anumit set de atribute determina in mod unic valoarea altor atribute. Fie R o schema de relație si doua submulțimi ale atributelor sale: $X \subseteq R$ *and* $Y \subseteq R$
- Dependenta functionala (DF) $X \rightarrow Y$ exista in R daca si numai daca pentru orice stare a relației r(R), egalitatea valorilor atributelor X din doua tupluri t_1 si t_2 din r ($t_1 \in r$ si $t_2 \in r$) implica egalitatea valorilor atributelor Y din acele tupluri, adică: $t_1[X] = t_2[X] \Rightarrow t_1[Y] = t_2[Y]$
- Dependentele funcționale sunt generalizarea noțiunilor de chei ale relatiilor: orice cheie determina o DF in acea relație
  - Daca Y = R, atunci X este o cheie a relației
  - Reciproc, daca X este o cheie, Y = R ($X \rightarrow R$)
  - In acest caz $t_1[X] = t_2[X] \Rightarrow t_1 = t_2$, dar, cum intr-o relație nu pot exista doua tupluri identice, rezulta ca t_1 si t_2 sunt unul si același tuplu
- Cheile relaţiilor:
  - se pot preciza explicit (şi atunci ele implică DF corespunzătoare)
  - pot fi deduse din mulţimea DF stabilite de proiectant, folosind diferiti algoritmi

Tipuri de dependenţe funcţionale

- O mulțime F de dependente funcționale in R definește constrângerile pe care trebuie sa le respecte orice relație r(R) pentru a fi corecta
  - Se spune ca F se menține in R daca toate relațiile legale pe R (r(R)) satisfac mulțimea de dependente funcționale F
  - Reciproc, o relație r este legala in raport cu o mulțime de dependente funcționale F, daca r satisface F
- Dependenţe funcţionale triviale si ne-triviale
  - O dependenta funcțională este triviala dacă este satisfăcută de orice stare a relației; in general $X \rightarrow Y$ este triviala dacă $Y \subseteq X$
  - Celelalte dependente (in care Y nu este o submulțime a lui X) sunt ne-triviale
- Dependenţele funcţionale parţiale si totale
  - O dependenţă funcţională $X \rightarrow Y$ este parţială (partial dependency) dacă există o submulţime proprie Z a lui X ($Z \subseteq X$) care determină funcţional pe Y ($Z \rightarrow Y$)
  - O dependenţă funcţională $X \rightarrow Y$ este totală (full dependency), dacă nu există nici o submulţime proprie Z a lui X care să determine funcţional pe Y . Rezulta:
    - dacă atributul X este simplu, dependenţa funcţională $X \rightarrow Y$ este totală
    - dacă X este o cheie candidată a lui R, atunci dependenţa $X \rightarrow R$ este totală
- Atributele care aparţin unei chei se numesc atribute prime, iar celelalte se numesc atribute neprime

Multimi de dependente functionale (1)

- Proiectantul bazei de date specifică acele dependenţe funcţionale între atribute care sunt evidente din punct de vedere semantic; de ex, in AP:
  - (a) IdAngajat →Nume
  - (b) IdAngajat →Prenume
  - (c) IdAngajat →Adresa
  - (d) {IdAngajat,IdProiect} →Ore
- Din acestea se pot deduce si alte DF, folosind regulile de inferenţă (inferenţă - *inference rules*) ale lui Armstrong:

1. Reflexivitatea (*reflexivity*): dacă Y ⊆ X, atunci X →Y; prin această regulă se deduc DF *triviale*; de exemplu în relaţia AP:

- (e) {IdAngajat, IdProiect} →IdAngajat
- (f) {IdAngajat, IdProiect} →IdProiect

2. Augmentarea (*augmentation*): dacă X →Y, atunci (X ∪ Z) →(Y ∪ Z); următoarele DF (g şi h) sunt deduse prin augmentare, pornind de la dependenţele funcţionale (a) şi respectiv (b), augmentate cu Z = {Nume}, respectiv Z = {Prenume} (se stie ca Nume ∪ Nume = Nume etc.)

- (g) {IdAngajat, Nume} →Nume
- (h) {IdAngajat, Prenume} →Prenume

Multimi de dependente functionale (2)

3. Tranzitivitatea (*transitivity*): dacă X→Y şi Y→Z, atunci X→Z. De exemplu, din DF (a) şi (e) rezultă:

(i) {IdAngajat,IdProiect} →Nume

Aceste trei reguli sunt suficiente pentru calculul tuturor dependenţelor funcţionale pe care le implică o mulţime dată de DF

Din aceste reguli de bază se pot deduce şi alte reguli; de exemplu:

Proiecţia (*projection*): dacă X→(Y ∪ Z), atunci X→Y şi X→Z

Reuniunea (*union*): dacă X→Y şi X→Z, atunci X→(Y ∪ Z)

Pseudo-tranzitivitatea (*pseudo-transitivity*): dacă X→Y şi (W ∪ Y) →Z, atunci (W ∪ X) →Z.

Inchiderea unei multimi de DF

- *Fiind dată o mulţime* F *de DF, mulţimea tuturor DF care sunt implicate de* F *se numeşte* ***închiderea mulţimii F*** *(closure set of F) şi se notează* F^+
  - F^+ se poate deduce prin aplicarea repetată a regulilor de inferenţă asupra DF din F
  - Pentru a testa daca o DF $X \rightarrow Y$ poate fi dedusă dintr-o mulţime F de DF, se afla închiderea F^+ a mulţimii F şi se testează dacă $(X \rightarrow Y) \in F^+$
  - *Două mulţimi de DF,* E şi F *sunt echivalente dacă închiderile lor sunt egale* ($E^+ = F^+$); *adica:* $\forall DF \in E \Rightarrow DF \in F^+$ şi $\forall DF \in F \Rightarrow DF \in E^+$
- ***O mulţime G de DF este minimă*** *dacă satisface următoarele condiţii:*
  - *membrul drept al oricărei DF din* G *este un atribut simplu;*
  - *orice dependenţă funcţională din* G *este totală*;
  - *mulţimea* G *este ireductibilă, adică, dacă se exclude o DF din* G, *mulţimea rezultată* H *nu este echivalentă cu G* (*adică* $H^+ \neq G^+$).
- ***Acoperirea minimă a unei mulţimi F de DF*** (*minimal cover of a set* F *of DFs*) *este o mulţime minimă de dependenţe funcţionale* G *care este echivalentă cu* F, *adică* $G^+ = F^+$
  - Pot exista mai multe acoperiri minime ale unei mulţimi de DF

Dependenţele funcţionale şi cheile relaţiilor

- In orice relaţie pot exista două categorii de DF:
  - DF determinate de cheile relaţiei; astfel de DF nu produc redundanţa datelor şi nici anomalii de actualizare a relaţiei
  - DF în care atributul determinant nu este o cheie a relaţiei; astfel de DF produc redundanţa datelor şi anomalii de actualizare a relaţiei
- DF determinate de chei (primară sau secundara) sunt constrângeri implicite, conţinute în definiţia relaţiei şi sunt verificate şi impuse automat de SGBD
  - Proiectantul bazei de date nu trebuie să prevadă nimic suplimentar pentru ca aceste constrângeri să fie satisfăcute de orice stare a relaţiei
- DF în care atributul determinant nu este o cheie sunt constrângeri explicite, care nu sunt verificate şi nici impuse de SGBD
- Pentru DF în care atributul determinant nu este o cheie, se aplica una din urmatoarele solutii:
  - (a) Se accepta astfel de DF, dar se asigura verificarea şi impunerea lor procedurala, prin triggere, proceduri stocate, sau funcţii în programele de aplicaţii
  - (b) Se descompune relatia in relatii mai simple, in care nu mai exista astfel de DF, dar trebuie verificate conditiile de descompunere reversibila

Inchiderea unui atribut fata de o multime de DF (1)

Închiderea unui atribut faţă de o mulţime F de DF (*the closure of an atribute under F*). *Fie un atribut* X *al unei relaţii pe care este definită mulţimea* F *de DF*; *mulţimea* X^+ *a tuturor atributelor determinate de* X (X→X^+) *se numeşte închiderea lui* X *în raport cu* F

- Algoritmul prin care *se poate deduce inchiderea unui atribut*:

  1. se setează $X^+ = X$
  2. repetă
     $P\_X^+ = X^+$
     pentru fiecare DF Y→Z din F
     dacă $Y \subseteq X^+$ atunci $X^+ = X^+ \cup Z$
     până când $P\_X^+ == X^+$

- Se aplică acest algoritm pentru dependenţele funcţionale din relatia AP:

  F_{AP} = {IdAngajat → Nume, IdAngajat → Prenume, IdAngajat → Adresa, {IdAngajat, IdProiect} → Ore}

- Se obtin inchiderile atributelor astfel:

  $\{IdAngajat\}^+ = \{IdAngajat, Nume, Prenume, Adresa\}$

  $\{IdAngajat, IdProiect\}^+ = \{IdAngajat, IdProiect, Nume, Prenume, Adresa, Ore\}$

  $\{Nume\}^+ = \{Nume\}$, $\{Prenume\}^+ = \{Prenume\}$, $\{IdProiect\}^+ = \{IdProiect\}$;

  $\{Adresa\}^+ = \{Adresa\}$, $\{Ore\}^+ = \{Ore\}$

Inchiderea unui atribut fata de o multime de DF (2)

- Din algoritmul de mai sus se poate deduce că închiderea unui atribut care nu determină funcţional nici un alt atribut (nu apare în partea stângă a nici unei DF) este chiar atributul respectiv:

  $\{Nume\}^+ = \{Nume\}$, $\{Prenume\}^+ = \{Prenume\}$, $\{Adresa\}^+ = \{Adresa\}$, $\{Ore\}^+ = \{Ore\}$

- Algoritmul de calcul al închiderii unui atribut faţă de o mulţime F de DF poate fi folosit pentru a verifica dacă o DF dată $X \rightarrow Y$ este consecinţa logică a mulţimii F (daca $(X \rightarrow Y) \in F^+$)
- Pentru aceasta, se calculează X^+ faţă de F; dacă $Y \subseteq X^+$, atunci $(X \rightarrow Y) \in F^+$, deci $X \rightarrow Y$ este consecinţa logică a lui F
- Acest algoritm poate fi folosit şi pentru a *verifica dacă un atribut K (simplu sau compus) este supercheie a relaţiei R cu mulţimea F de DF*
- Pentru aceasta se calculează K^+ în raport cu F; daca $K^+ = R$, atunci K este supercheie în R
- Dintr-o supercheie se pot deduce cheile relaţiei, testând condiţia de ireductibilitate pentru fiecare din atributele componente ale supercheii
- Algoritmul de găsire a cheilor unei relaţii din mulţimea DF bazat pe inchiderea unui atribut fata de o multime de DF este prezentat in continuare

Aflarea cheilor unei relaţii din multimea DF

- Fiind dată o relaţie cu schema R şi mulţimea F a DF pe această relaţie, cheile candidate K ale relaţiei se pot afla cu urmatorul algoritm:

  1. se setează K = R şi se selectează din K un atribut X (preferabil nedeterminant);

  2. se testează (K – X) pe baza multimii F: dacă (K – X) este supercheie a relaţiei R, atunci K = (K – X), altfel K nu se modifica

  3. se selectează un alt atribut X din K şi se reia pasul 2, pana cand (K-X) nu mai este supercheie in R

- Prin parcurgerea repetată a paşilor 2 şi 3, se găseşte una din cheile candidate ale relaţiei. Dacă există mai multe chei candidate, atunci ordinea găsirii cheilor candidate depinde de atributul selectat în pasul 1 al algoritmului
- De exemplu, se aplică algoritmul de mai sus pentru găsirea cheii primare a relaţiei AP cu mulţimea F_{AP} a DF (definita la inceputul capitolului)
- Se porneşte cu K = R = {IdAngajat, Nume, Prenume, Adresa, IdProiect, Ore}, se selectează X= Nume şi se verifică dacă {K – X} este supercheie in R
  - Pt. aceasta se calculeaza $\{K - X\}^+$ = {IdAngajat, Nume, Prenume, Adresa, IdProiect, Ore} = R, deci {IdAngajat, Prenume, Adresa, IdProiect, Ore} este supercheie in R
- Se repeta pasul 2 pentru X = Prenume, apoi X = Adresa si X = Ore; aceste atribute pot fi eliminate, si se obtine K = {IdAngajat, IdProiect}
- Atributele IdProiect, IdAngajat nu se pot elimina din supercheia K, deci K e cheie

Descompunerea relatiilor

- *O descompunere D = {R1, R2,..Ri,...Rk} a schemei de relaţie R (relation schema decomposition) este formată din submulţimi proprii ale lui R (R1⊂ R, R2⊂ R,…Rk⊂ R a caror reuniune este egala cu R (R = R1∪ R2…∪ Rk)*
- *Proiectiile relatiei r(R) pe submultimile R1, R2,...Ri, …Rk (r1 = Π_{R1}(r), r2 = Π_{R2}(r), ….. ri = Π_{Ri}(r),… rk = Π_{Rk}(r)) reprezinta* descompunerea *relaţiei r(R) pe aceste submulţimi de atribute*
- Fie o relaţie cu schema R şi mulţimea F de DF ale acesteia; o descompunere a relaţiei r(R) este *reversibilă* dacă are proprietăţile de:
  - ***Joncţiune fără pierdere de informaţie*** *(conservarea informatiei); inseamna ca r = r1 ⋈ r2 ⋈ …ri ⋈ …rk*
  - ***Conservarea dependenţelor funcţionale****: dacă oricare din dependenţele din F se regăseşte sau poate fi dedusă din DF ale relaţiilor cu schemele R1,R2,...Ri,…Rk*
- ***Teorema lui Ullman****. Descompunerea* D = {R1, R2} *a unei scheme de relaţie R este o descompunere fără pierdere de informaţie la joncţiune în raport cu mulţimea F de DF, dacă şi numai dacă este îndeplinită una din condiţiile:* (a) ((R1 ∩ R2) → (R1 - R2)) ∈ F^+, sau (b) ((R1 ∩ R2) → (R2 - R1)) ∈ F^+
  - Dacă R1 ∩ R2 este o cheie a uneia dintre relaţiile R1 sau R2, atunci descompunerea este fără pierdere de informaţie la joncţiune;
    - Demonstratie: dacă (R1 ∩ R2) este o supercheie în R1, atunci ea determina functional orice submultime de atribute din R1, inclusiv (R1 - R2) adică (R1 ∩ R2) →(R1 - R2);
    - La fel se demonstreaza daca (R1 ∩ R2) este o supercheie în R2

Jonctiune fara pierdere de informatie

- Lipsa acestei proprietăţi se manifestă în mod paradoxal prin apariţia unor tupluri noi (parazite) în relaţia obţinută prin joncţiunea relaţiilor r1,r2,...ri,...rk, tupluri care nu existau în relaţia r(R)
- Exemplu: descompunerea relaţiei AP în relaţiile A şi P:

  $A \cap P$ = {IdAngajat},

  $A - P$ = {Nume,Prenume,Adresa}; $P - A$ = {IdProiect,Ore}

  Din DF IdAngajat→Nume, IdAngajat→Prenume, IdAngajat→ Adresa

  Se deduce: IdAngajat→{Nume, Prenume, Adresa} (cf. regulii de reuniune a DF)

  Rezultă: $((A \cap P) \rightarrow (A - P)) \in F^+$

  Deci, conform teoremei lui Ullman desc. este fără pierdere de informaţie la joncţiune
- Descompunerea succesivă fără pierdere de informaţie la joncţiune
  - Dacă o descompunere D = {R1,R2,...Ri,...Rk} a unei scheme R este fără pierdere de informaţie la joncţiune în raport cu mulţimea F de DF
  - şi D1 = {Q1,Q2,...Qm} este o descompunere fără pierdere de informaţie la joncţiune a lui R1 în raport cu proiecţia lui F pe R1,
  - atunci descompunerea D2 = {Q1,Q2,...Qm,R2,...Rk} este o descompunere fără pierdere de informaţie a lui R în raport cu F
- Această proprietate permite descompunerea fără pierdere de informaţie la joncţiune a unei relaţii în mai multe etape (mai întâi in două relaţii, apoi fiecare dintre acestea se descompune în alte două relaţii, ş.a.m.d)

Conservarea dependenţelor funcţionale

- *O descompunere* D = {R1, R2, ...Ri,...Rk} *a unei scheme de relaţie* R *prezintă proprietatea de conservare a mulţimii* F *de DF, dacă reuniunea proiecţiilor lui* F *pe schemele de relaţii* Ri (*unde* $1 \leq i \leq k$) *este echivalentă cu* F*, adică:* $((\Pi_{R1}(F)) \cup (\Pi_{R2}(F)) \ldots \cup (\Pi_{Rk}(F)))^+ = F^+$
- ***Proiecţia unei mulţimi de dependenţe funcţionale.*** *Fiind dată mulţimea* F *de DF definite pe* R*, şi o descompunere* D = {R1,R2,..Rk} *a lui* R*, proiecţia lui* F *pe* Ri (*unde* $1 \leq i \leq k$)*, notată* $\Pi_{Ri}(F)$*, este mulţimea de DF* $X \rightarrow Y \in F^+$*, astfel încât* $X \subseteq Ri$ *şi* $Y \subseteq Ri$
- Exemplu: descompunerea relaţiei AP in relaţiile A şi P:

  F_{AP} = {IdAngajat→Nume, IdAngajat→Prenume,
  IdAngajat→Adresa,{IdAngajat,IdProiect}→Ore}

  Proiecţiile mulţimii F_{AP} pe A si P sunt:

  $\Pi_A(F_{AP})$ = {IdAngajat→Nume, IdAngajat→Prenume, IdAngajat→Adresa}

  $\Pi_P(F_{AP})$ = {{IdAngajat,IdProiect}→Ore}

  Dat fiind că $\Pi_A(F_{AP}) \cup \Pi_P(F_{AP}) = F_{AP}$, rezultă că această descompunere conservă DF
- Dacă, prin descompunerea unei relaţii, se pierd una sau mai multe DF, pentru verificarea lor, trebuie să se efectueze mai întâi joncţiunea relaţiilor obtinute
- Cele două proprietăţi ale unei descompuneri, conservarea informaţiei şi conservarea dependenţelor funcţionale, sunt independente

Formele normale ale relatiilor (FN1 si FN2)

- *O relaţie este normalizată în prima formă normală (FN1) dacă fiecare atribut ia numai valori atomice şi scalare din domeniul său de definiţie*
  - Sistemele SGBD relaţionale nu admit relaţii care să nu fie cel puţin în FN1, dar proiectarea relaţiilor normalizate în FN1 este intotdeauna posibila
- *Fie o schema de relatie R si multimea F de DF definite pe aceasta. O relaţie r(R) este normalizată în FN2, dacă este în FN1 şi dacă în F^+ nu există nici o DF parţială a unui atribut neprim faţă de o cheie a relaţiei*
- Exemplu: relatia AP cu multimea F_{AP} de DF este in FN1, dar nu este in FN2:

  F_{AP} = {IdAngajat → Nume, IdAngajat → Prenume,
  IdAngajat → Adresa, {IdAngajat, IdProiect} → Ore}
  - Cheia primara este {IdAngajat, IdProiect}, deci {IdAngajat, IdProiect}→Nume
  - Această DF se poate deduce din F_{AP}; cf. reflexivitatii, {IdAngajat, IdProiect} → IdAngajat; cf.tranzitivitatii, daca IdAngajat→Nume, {IdAngajat, IdProiect}→Nume
  - DF {IdAngajat, IdProiect}→Nume este parţială deoarece exista: IdAngajat→Nume

  Rezultă că relaţia AP nu este în FN2, deci prezinta redundante si anomalii
- Anomaliile se pot evita prin (a) normalizare sau (b) prin impunerea DF
- Normalizarea relatiei AP prin descompunere in relatiile A si P:
  - A(IdAngajat, Nume, Prenume, Adresa)
  - P(IdAngajat, IdProiect, Ore),
  - S-a demonstrat ca aceasta descompunere prezinta proprietatile de jonctiune fara pierdere de informatie si conservarea DF si se poate arata ca A si P sunt in FN2

Impunerea DF in relatia nenormalizata AP (1)

- Dacă relaţia AP nu se normalizează, atunci trebuie să se prevadă proceduri speciale care să verifice şi să impună DF care nu sunt determinate cheia PK
- La orice operaţie de actualizare un trigger verifica valorile care urmează să fie introduse şi nu se admit doi sau mai mulţi angajaţi cu acelasi identificator (IdAngajat) dar cu nume, prenume sau adresă diferite
- Trigger-ul pentru relatia AP arata astfel:

```
DELIMITER $$ DROP TRIGGER `normalizare`.`trigger_AP`$$
CREATE TRIGGER `trigger_AP` AFTER INSERT ON `normalizare`.`AP` FOR EACH ROW
BEGIN DECLARE done INT DEFAULT 0;
DECLARE error INT DEFAULT 0;
DECLARE l_nume, l_prenume, l_adresa VARCHAR(20);
DECLARE cursor_AP CURSOR FOR SELECT Nume, Prenume, Adresa FROM AP
WHERE IdAngajat = NEW.IdAngajat;
DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;
OPEN cursor_AP;
REPEAT FETCH cursor_AP INTO l_nume, l_prenume, l_adresa;
        IF NEW.Nume <> l_nume OR NEW.Prenume != l_prenume OR NEW.Adresa <> l_adresa
        THEN SET error = 1; END IF;
        UNTIL done = 1 OR error = 1
        END REPEAT; CLOSE cursor_AP;
        IF error = 1 THEN
        DELETE FROM AP WHERE IdAngajat=NEW.IdAngajat AND IdProiect=NEW.IdProiect; END IF;
        END $$  DELIMITER ;
```

Impunerea DF in relatia nenormalizata AP(2)

- Trigger-ul este de tip AFTER INSERT, astfel ca linia dorita se insereaza mai intai in tabel, cu verificarea de catre SGBD a unicitatii cheii primare
- Dupa inserare se citesc intr-un cursor toate liniile care au pentru IdAngajat valoarea nou inserata (NEW.IdAngajat)
- Se parcurg liniile cursorului si se verifica daca valorile nou introduse ale atributelor Nume, Prenume, Adresa (NEW.Nume, NEW.Prenume, NEW.Adresa) difera de cele existente; daca difera, se seteaza error = 1
- Daca, dupa parcurgerea tuturor liniilor cursorului, error=1, se sterge linia nou introdusa, deci in final, linia nu este inserata in tabel.
- Exemplu:
  - Fie tabelul AP cu liniile (1,'Ionescu','Ion', 'Bucuresti',1,50), (1,'Ionescu','Ion', 'Bucuresti',2,100) si (1,'Ionescu','Ion','Bucuresti',3,80)
  - Se observa redundanta datelor: valorile 'Ionescu', 'Ion', 'Bucuresti' sunt memorate pentru fiecare proiect la care lucreaza angajatul cu IdAngajat = 1
  - Anomalia de inserare: daca nu se activeaza trigger-ul, se poate insera si linia (1,'Marinescu','Mihai','Bucuresti',4,60), ceea ce inseamna ca angajatul cu IdAngajat =1 este nedeterminat (este Ionescu Ion sau Marinescu Mihai?)
  - Aceasta linie nu se poate insera daca s-a activat trigger-ul trigger_AP

Forma normala 3 (FN3)

- *Fie o schema de relatie R si multimea F de DF definite pe aceasta. O relaţie r(R) este normalizată în FN3, dacă este în FN2 şi dacă oricare DF din F^+ a unui atribut neprim este determinata de o cheie a relatiei*
- Exemplu: AFS(IdAngajat, Nume, Prenume, Adresa, Functie, Salariu)

  F_{AFS}={IdAngajat→Nume, IdAngajat→Prenume, IdAngajat →Adresa,

  IdAngajat→Functie, Functie→Salariu}.
  - Cheia primară a relaţiei este IdAngajat, şi poate fi dedusă din F_{AFS}
  - Primele patru DF sunt totale, deci relaţia este în FN2
  - Relaţia nu este în FN3 datorita DF Functie→Salariu; prezinta redundante si anomalii
- Normalizare prin descompunere in:

  AF(IdAngajat, Nume, Prenume, Adresa, Functie)

  FS(Functie, Salariu)
- Se demonstreaza ca AF si FS sunt in FN3 si ca descompunerea este reversibila
  - Proiectiile multimii F_{AFS}:

    F_{AF}={IdAngajat→Nume,IdAngajat→Prenume,IdAngajat →Adresa,IdAngajat→Functie}

    F_{FS} = {Functie→Salariu} si se deduc (sau se verifica) cheile relatiilor
  - PK_{AF}={IdAngajat}, PK_{FS}={Functie}, deci AF si FS sunt in FN3
  - $F_{AF} \cup F_{FS} = F_{AFS}$ deci descompunerea conserva DF
  - AF ∩ FS = {Functie} si (Functie → Salariu) ∈ F_{AFS}, deci, cf. regulii lui Ullman, descompunerea este fara pierdere de informatie la jonctiune

Impunerea DF in relatia nenormalizata AFS (1)

Dacă relaţia AFS nu se normalizează, atunci trebuie să se prevadă proceduri speciale care să verifice şi să impună DF care nu sunt determinate de cheia PK

Se poate inlocui operatia de INSERT cu apelul unei proceduri stocate care verifica mai intai valorile si executa INSERT numai daca acestea respecta DF

Procedura stocata pentru relatia AFS arata astfel:

```
DELIMITER $$ DROP PROCEDURE IF EXISTS sp_AFS $$
CREATE PROCEDURE `sp_AFS`(INOUT error INT, s_id INT, s_nume VARCHAR(20), s_prenume VARCHAR(20), s_adresa VARCHAR(20), s_functia VARCHAR(20), s_salariu DECIMAL)
BEGIN DECLARE done INT DEFAULT 0;
DECLARE l_functia VARCHAR(20); DECLARE l_salariu DECIMAL;
DECLARE cursor_AFS CURSOR FOR
SELECT Functia, Salariu FROM AFS WHERE Functia = s_functia;
DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;
OPEN cursor_AFS;
REPEAT FETCH cursor_AFS INTO l_functia, l_salariu;
IF s_salariu <> l_salariu THEN SET error = 1; END IF;
UNTIL done = 1 OR error = 1
END REPEAT;
CLOSE cursor_AFS;
IF error = 0 THEN
INSERT INTO AFS VALUES (s_id, s_nume, s_prenume, s_adresa, s_functia, s_salariu);END IF;
END$$  DELIMITER ;
```

Impunerea DF in relatia nenormalizata AFS (2)

- Fie tabelul AFS care contine liniile: (1, ‘Ionescu’,’Ion’,’Bucuresti’,’inginer’,2000), (2, ‘Popescu’,’Petre’,’Craiova’,’inginer’,2000)
- Se observa redundanta datelor: valoarea salariului este memorata de fiecare data pentru o functie anume, desi trebuie sa fie aceeasi (conform DF Functie →Salariu)
- Datorita redundantei exista si anomalii: se poate insera linia (3, ‘Mateescu’, ’Viorel’,’Bucuresti’,‘inginer’, 2100), care nu respecta DF Functie → Salariu: pentru functia ‘inginer’ se admite si salariul 2000 si salariul 2100; se sterge apoi linia
inserata eronat
- Inserarea liniilor in tabelul AFS prin apelul procedurii stocate sp_AFS inlatura aceasta anomalii
- De exemplu, la apelul procedurii stocate cu aceleasi valori:

```
SET @error = 0;
call sp_AFS(@error, 3, ‘Mateescu', ‘Viorel', 'Bucuresti','inginer', 2100);
select @error;
```

se obtine @error=1 si linia respectiva nu va fi introdusa in tabelul AFS

Forma normala Boyce-Codd (FNBC)

- *Fie schema de relatie R si multimea F de DF definite pe aceasta. O relaţie r(R) este în forma normală Boyce-Codd* (FNBC) *în raport cu* F *dacă este în* FN1 *şi dacă orice DF netrivială din* F^+ *este determinata de o cheie a relaţiei*
- Este evident că o relaţie în FNBC este în FN3, dar o relaţie în FN3 poate să fie sau nu în FNBC
- Exemplu: relaţia EDP(<u>IdElev, IdDisciplina</u>,IdProfesor), cu cheia PK= {IdElev, IdDisciplina} şi cu mulţimea F_{EDP} de DF:

  F_{EDP} = {{IdElev,IdDisciplina}→IdProfesor, IdProfesor→IdDisciplina}
- Se consideră că relaţia este în FN1; din F_{EDP} se observă că nu există DF parţiale faţă de cheia relaţiei (deci relaţia este în FN2) şi nu există nici o DF a unui atribut neprim faţă de un alt atribut neprim, deci relaţia este în FN3
- Relaţia EDP nu este în FNBC, datorită DF a atributului prim IdDisciplina faţă de atributul neprim IdProfesor; aceasta relatie prezintă redundante de date şi anomalii de actualizare
- Exemplu: fie starea in care tabelul EDP contine liniile (1,1,1) si (2,1,1)
  - redundanta: disciplina (1) predata de profesorul 1 este memorata de doua ori in tuplurile (1,1,1) si (2,1,1)
  - anomalie de inserare: se poate insera si tuplul (1,2,1) care nu respecta DF IdProfesor→IdDisciplina (profesorul 1 preda si disciplina 1 si disciplina 2)

Impunerea DF in relatia nenormalizata EDP

Dacă relaţia EDP nu se normalizeaza, atunci trebuie să se prevadă o procedura care să verifice şi să impună DF (IdProfesor→IdDisciplina) care nu este determinata de cheia relatiei, pentru operatiile de INSERT si UPDATE

De ex: se inlocuieste operatia INSERTcu apelul unei proceduri stocate care verifica mai intai valorile si executa INSERT numai daca acestea respecta DF

```
DELIMITER $$
DROP PROCEDURE IF EXISTS `sp_EDP`$$
CREATE PROCEDURE `sp_EDP`(INOUT error INT, s_Elev INT, s_Disciplina INT, s_Profesor INT)
BEGIN DECLARE done INT DEFAULT 0; DECLARE l_Disciplina INT;
DECLARE cursor_EDP CURSOR FOR
SELECT IdDisciplina FROM EDP WHERE IdProfesor = s_Profesor; DECLARE CONTINUE
HANDLER FOR NOT FOUND SET done = 1; OPEN cursor_EDP;
REPEAT FETCH cursor_EDP INTO l_IdProfesor;
IF s_Disciplina <> l_Disciplina THEN SET error = 1; END IF; UNTIL done = 1 OR error = 1
END REPEAT; CLOSE cursor_EDP;
IF error = 0 THEN INSERT INTO EDP VALUES (s_Elev, s_Disciplina, s_Profesor); END IF; END$$
DELIMITER ;
```

Pentru verificare: se sterge linia(1,2,1) (daca exista) si se executa:

```
set @error=0; call sp_EDP(@error,1,2,1); select @error;
```

se obtine @error = 1 si nu s-a inserat acest tuplu

Normalizarea relatiei EDP

Normalizarea relaţiei EDP (ElevDisciplinaProfesor) astfel încât relaţiile obţinute să fie în FNBC se poate face prin descompunerea acesteia. Se pot încerca trei descompuneri:

D1 = {EP,PD}, unde EP = {IdElev, IdProfesor},
PD = {IdProfesor, IdDisciplina};

D2 = {ED,PD}, unde ED = {IdElev, IdDisciplină},

PD = {IdProfesor, IdDisciplina};

D3 = {EP,ED}, unde EP = {IdElev, IdProfesor},

ED = {IdElev, IdDisciplina}.

Se poate observa că relaţiile rezultate în oricare din aceste descompuneri sunt relaţii în FNBC (fiind relaţii formate din două atribute).

Dintre cele trei descompuneri, descompunerea D1 prezintă proprietatea de joncţiune fără pierdere de informaţie. Într-adevăr, EP ∩ PD = {IdProfesor}, PD − EP = {IdDisciplina} şi IdProfesor→IdDisciplina, deci este îndeplinită condiţia lui Ullman de conservare a informaţiei.

Celelalte descompuneri, D2 şi D3, nu îndeplinesc această condiţie.

In oricare din aceste descompuneri se pierde dependenţa funcţională {IdElev,IdDisciplina}→IdProfesor, deci relaţia EDP nu poate fi descompusă în mod reversibil în relaţii FNBC.

Impunerea constrangerilor pierdute la descompunere (1)

- Dacă relaţia EDP se normalizeaza prin descompunerea (EP, PD) atunci trebuie să se prevadă o procedura care să verifice şi să impună constrangerea pierduta
- Se pot inlocui operatiile de INSERT in tabelele EP, PD cu apelul unei proceduri stocate care verifica mai intai valorile si executa INSERT numai daca acestea respecta constrangerea: {IdElev, IdDisciplina} → IdProfesor

```
DELIMITER $$ DROP PROCEDURE IF EXISTS `sp_EP_PD`$$
CREATE PROCEDURE `sp_EP_PD`(INOUT error INT, s_Elev INT, s_Disciplina INT, s_Profesor INT)
BEGIN DECLARE done INT DEFAULT 0;
DECLARE l_Elev, l_Profesor, l_Disciplina INT;
DECLARE cursor_EP_PD CURSOR FOR
SELECT IdElev, EP.IdProfesor, IdDisciplina FROM EP, PD
WHERE EP.IdProfesor = PD.IdProfesor ;
DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;
OPEN cursor_EP_PD;
REPEAT FETCH cursor_EP_PD INTO l_Elev, l_Profesor, l_Disciplina;
        IF s_Elev = l_Elev AND s_Disciplina = l_Disciplina AND s_Profesor <> l_Profesor        THEN
        SET error = 1; END IF;
        UNTIL done = 1 OR error = 1
        END REPEAT;
        CLOSE cursor_EP_PD;
        IF error = 0 THEN
        INSERT INTO EP VALUES (s_Elev, s_Profesor);
        INSERT INTO PD VALUES (s_Profesor, s_Disciplina);
    END IF;    END$$  DELIMITER ;
```

Impunerea constrangerilor pierdute la descompunere (2)

- Procedura sp_EP_PD primeşte ca argumente un flag de eroare şi valorile celor trei atribute IdElev, IdDisciplină, IdProfesor care trebuie sa respecte constrângerea {IdElev, IdDisciplina} → IdProfesor
- În această procedura se creează un cursor în care se încarcă rezultatul joncţiunii naturale între relaţiile EP şi PD pe atributul comun IdProfesor
- Pentru fiecare linie a rezultatului joncţiunii EP ⋈ PD se verifică respectarea constrângerii dorite, testând valorile existente în linia respectivă cu noile valori care urmează să fie introduse:
  - daca această constrângere este respectată, atunci se execută două instrucţiuni INSERT, cate una in foiecare din tabelele EP si PD
  - daca această constrângere nu este respectată, nu se introduce nici o linie
- Exemplu: daca tabelul EP contine linia (1,1) si tabelul (PD) contine linia (1,1), prin INSERT se pot introduce si valorile (1,1,2) pentru (IdElev, IdDisciplina, IdProfesor) adica liniile: (1,2) in EP si (2,1) in PD; dar aceste valori nu respecta constrangerea {IdElev, IdDisciplina} → IdProfesor deoarece:
  - in liniile existente (IdElev, IdDisciplina) = (1,1), iar IdProfesor = 1
  - in valorile de inserat (IdElev, IdDisciplina) = (1,1), iar IdProfesor = 2
- In schimb apelul procedurii: set @error = 0; call sp_EP_PD(@error, 1,1,2); select @error; produce @error=1 si nu se introduce nici o linie

Dependente multivalorice

- *O dependenţă multivalorică* - DMV- (*multivalued dependency*) X→→Y *specificată pe schema de relatie* R = {X,Y,Z} *stabileşte următoarele constrângeri pe care trebuie să le respecte orice relaţie r(R)*: *dacă există două tupluri* t1 *şi* t2 *în* r *astfel ca* t1[X] = t2[X]= x*, atunci vor exista şi tuplurile* t3 *şi* t4 *cu următoarele proprietăţi:*

  t3[X] = t4[X] = t1[X] = t2[X] = x;

  t3[Y] = t1[Y] = y1 *şi* t4[Y] = t2[Y] = y2 ;

  t3[Z] = t2[Z] = z2 *şi* t4[Z] = t1[Z] = z1 .

- Datorită simetriei egalităţilor de mai sus, rezulta că, dacă într-o relaţie cu schema R există DMV X→→Y, atunci există şi X→→Z, unde Z = R−(X∪Y)
- O DF este un caz particular al unei DMV: DF X→Y este o DMV X →→ Y cu restricţia că unei valori a lui X îi corespunde o singură valoare a lui Y
- *O relaţie cu schema* R *este în a patra formă normală* (FN4) *în raport cu o mulţime* F *de DF şi DMV dacă este în* FN1 *şi dacă, pentru orice DMV netrivială* X→→Y *din* F^+*,* X *este cheie a relaţiei r(R).*
- Asemănarea acestei definiţii cu definiţia FNBC: pentru FNBC se impun restricţii DF, iar pentru FN4 se impun restricţii DMV
- Dacă o schemă de relaţie respectă condiţia de FN4, atunci înseamnă că ea respectă şi condiţia de FNBC

Dependente de jonctiune

Fie o schema de relaţie R *şi* R1,R2,...Rk *submulţimi de atribute ale lui* R, *unde* R1 ∪ R2 ∪...∪ Rk = R. *Se spune că există o dependenţă de joncţiune (DJ) pe* R, *notată* *(R1,R2,...Rk), *dacă şi numai dacă orice relaţie* r(R) *este egală cu joncţiunea proiecţiilor relaţiei pe submulţimile* R1,R2,..Rk, *adică*

$r = \Pi_{R1}(r) \bowtie \Pi_{R2}(r) ... \bowtie \Pi_{Rk}(r)$.

- Rezulta ca:
  - *(R1,R2,...Rk) este o DJ pe R dacă şi numai dacă descompunerea lui R în proiecţiile pe R1,R2,...Rk este fără pierdere de informaţie la jonctiune
  - Relaţia r(R) este *k*-decompozabilă fără pierdere de informaţie dacă există o DJ *(R1,R2,...Rk)
- Tipuri de DJ (dupa valoarea lui k):
  - Cazul k = 1 reprezintă o DJ trivială
  - Cazul k = 2 al unei DJ este o DMV; o DMV X→→Y în relaţia r(R) reprezintă o DJ *(XY, XZ), unde Z = R–(X ∪ Y).
  - In cazul k > 2, DJ nu mai sunt echivalente cu DMV
- DJ sunt dificil de identificat şi nu există reguli de deducere (inferenţă) care să genereze toate DJ pornind de la o mulţime dată
- Datorită acestor aspecte, analiza DJ are un pronunţat caracter intuitiv

Forma normala FN5

*O relaţie cu schema R este în a cincea formă normală (FN5) în raport cu o mulţime F de dependenţe funcţionale, multivalorice sau de joncţiune dacă este în FN1 şi dacă, pentru orice dependenţă de joncţiune *(R1,R2,... Ri,...Rk) din F^+, Ri (unde 1 ≤ i ≤ k) este o cheie a relaţiei r(R)*

- Având în vedere faptul că o dependenţă multivalorică este un caz special de dependenţă de joncţiune, iar o dependenţă funcţională este un caz special de dependenţă multivalorică, se poate afirma că o relaţie care este în FN5, este implicit în FN4, deci şi în FNBC, ş.a.m.d.

- S-a demonstrat că orice relaţie poate fi transformată în relaţii FN5 (FN4, FNBC etc.) printr-o descompunere fără pierdere de informaţie, dar nu se asigura conservarea tuturor DF
- Conditiile de normalizare in FNBC, FN4 si FN5 se pot rezuma la faptul căîntr-o relaţie nu trebuie să existe decât dependenţe determinate de chei

Proiectarea relatiilor normalizate

Normalizarea relaţiilor asigură un proiect al bazei de date mai concis şi de aceea se consideră că a normaliza este avantajos, chiar dacă normalizarea nu este o garanţie că s-a realizat cel mai bun model

Proiectarea bazelor de date pornind de la diagrama Entitate-Asociere conduce, în general, la relaţii normalizate, deoarece:

- Relaţiile corespunzătoare mulţimilor de entităţi sunt, de regulă, relaţii normalizate, dat fiind că toate atributele descriu tipul de entitate respectiv.
- Relaţiile de asociere binară, care conţin două chei străine care referă cheile primare din cele două relaţii pe care le asociază, rezultă tot ca relaţii normalizate
- Relaţiile care modelează asocierile multiple pot să rezulte nenormalizate şi necesită operaţii de normalizare suplimentare

„ Dar, cu cât nivelul de normalizare creşte, cu atât se obţin mai multe relaţii cu grad mai mic şi pentru fiecare interogare sunt necesare mai multe operaţii de joncţiune, ceea ce face ca timpul de execuţie a interogărilor să crească; in general, se recomandă ca:
  - relaţiile asupra cărora se efectuează operaţii de actualizare frecvente să fie normalizate într-o formă normală cât mai avansată
  - relaţiile asupra cărora se efectuează interogări frecvente pot fi păstrate într-o formă de normalizare mai redusă

„ Mentinerea unei relaţii într-o formă de normalizare mai redusă se numeşte “denormalizare”, şi are scopul de a obţine performanţe ridicate la interogări

Algoritmi de normalizare

- Analiza normalizării relaţiilor trebuie să fie facuta pentru orice proiect de baze de date, pentru a asigura funcţionarea corectă a acesteia:
  - Dacă o relaţie se păstrează într-o formă de normalizare mai redusă, atunci trebuie să se prevadă proceduri de verificare şi impunere a dependenţelor de date care nu sunt determinate de cheile relaţiei (ca si constrângeri explicite)
  - Dacă se normalizeaza o relaţie, dar prin descompunere se pierd unele DF, acestea pot fi impuse explicit prin proceduri stocate sau funcţii în programele de aplicaţie, care execută joncţiunea între relaţiile rezultate şi impun constrângerea respectiva
- S-a demonstrat si exista algoritmi prin care orice relatie poate fi descompusa reversibil (cu conservarea informatiei si conservarea DF) in relatii in formele normale FN2 sau FN3
- S-a demonstrat si exista algoritmi prin care orice relatie poate fi descompusa in relatii FN2, FN3, FNBC, FN4 sau FN5 fara pierdere de informatie la jonctiune, dar se pot pierde unele DF; algoritmul foloseste notiunea de inchiderea unui atribut fata de o multime F de DF

Descompunerea fara pierdere de informatie la jonctiune (1)

„ Fiind dată o relaţie cu schema R şi mulţimea F de DF, o descompunere D fără pierdere de informaţie la joncţiune în relaţii într-o formă normală dorită (FNBC, FN4, FN5) se poate obţine aplicând algoritmul următor:

1. se setează D = {R};

2. atât timp cât în D există o relaţie Q (cu mulţimea F_Q a DF) care nu se află în forma normală dorită:

   - se alege o DF $X \rightarrow W$ din F_Q care nu respecta forma normala dorita şi se construieşte închiderea X^+ a atributului X şi mulţimea $Y = X^+ – X$;

   - în D se înlocuieşte relaţia Q cu două relaţii: $Q1 = X \cup Y$ şi $Q2 = X \cup Z$, unde $Z = Q – (X \cup Y)$;

„ Demonstraţie:

- La fiecare pas de execuţie a algoritmului, o relaţie Q se descompune în două relaţii Q1 şi Q2, astfel încât $Q1 \cap Q2 = X$, şi $Q1 - Q2 = Y$
- Din definiţia închiderii unui atribut rezultă că $X \rightarrow Y$, deci, conform teoremei lui Ullman, această descompunere este fără pierdere de informaţie la joncţiune
- Astfel de descompuneri succesive păstrează caracterul de descompuneri fără pierdere de informaţie la joncţiune

Descompunerea fara pierdere de informatie la jonctiune (2)

„Exemplu de aplicare a algoritmului pentru descompunerea in relatii FNBC a relatiei R = {dAngajat, Nume, Prenume, Adresa, Functie, Salariu, IdProiect, Ore} cu PK = {IdAngajat, IdProiect}, aflata în FN1 şi cu mulţimea F de DF

F = {IdAngajat→Nume,IdAngajat→Preume, IdAngajat→Adresa,IdAngajat→Functie, Functie→Salariu, {IdAngajat,IdProiect}→Ore}

„Executia algoritmului:

- Se setează D = {R}
- Din F se alege DF IdAngajat→Nume care nu respectă condiţia impusă de FNBC
- Închiderea atributului X=IdAngajat este $[IdAngajat]^+$= {IdAngajat, Nume, Prenume, Adresa, Functie, Salariu} şi rezultă $Y = X^+ - X$ = {Nume,Prenume, Adresa, Functie, Salariu} şi $Z = R - (X \cup Y)$ = {IdProiect,Ore}.
- Se obtine descompunerea D a relaţiei R: D = {R_{11}, R_{12}}, unde

  $R_{11} = X \cup Y$ = {IdAngajat, Nume, Prenume, Adresa, Functie, Salariu}; in FN2

  $R_{12} = X \cup Z$ = {IdAngajat, IdProiect, Ore}; in FNBC

  - In acelasi mod se descompune relaţia R_{11}, in relatiile R_{111} si R_{112}:

    R_{111} = {Functie, Salariu}; este in FNBC

    R_{112} = {IdAngajat, Nume, Prenume, Adresa, Functie}; este in FNBC

„Se poate demonstra usor ca descompunerea obtinuta D= (R_{111}, R_{112}, R_{12}) conserva si dependentele functionale

Relatia furnizori-cheltuieli nenormalizata

Cod Furn.	Denumire	Cod fiscal	Nr. Crt.	Cod Chelt.	Denumire Cheltuială	Valoare
F100	Romgaz	R1234567	1	C15	Chelt pt. Încălzire	1500000
			2	C16	Chelt pt. Bucătării	500000
F110	Renel	R7654321	3	C10	Chelt cu iluminatul	3000000
			4	C11	Chelt pt. Func. liftului	200000

FN1

Cod Furn.	Denumire	Cod fiscal	Nr. Crt.	Cod Chelt.	Denumire Cheltuială	Valoare
F100	Romgaz	R1234567	1	C15	Chelt pt. Încălzire	1500000
F100	Romgaz	R1234567	2	C16	Chelt pt. Bucătării	500000
F110	Renel	R7654321	3	C10	Chelt cu iluminatul	3000000
F110	Renel	R7654321	4	C11	Chelt pt. Func. liftului	200000

Cod Furn.	Nr. Crt.	Cod Chelt.	Denumire Cheltuială	Valoare
F100	1	C15	Chelt pt. Încălzire	1500000
F100	2	C16	Chelt pt. Bucătării	500000
F110	3	C10	Chelt cu iluminatul	3000000
F110	4	C11	Chelt pt. func. Liftului	200000

Cod Furn.	Denumire	Cod fiscal
F100	Romgaz	R1234567
F100	Romgaz	R1234567
F110	Renel	R7654321
F110	Renel	R7654321

FN2

Relatia Cheltuieli

Cod Furn.	Nr. Crt.	Cod Chelt.	Valoare
F100	1	C15	1500000
F100	2	C16	500000
F110	3	C10	3000000
F110	4	C11	200000

Relaţia Furnizori

Cod Furn.	Denumire	Cod fiscal
F100	Romgaz	R1234567
F100	Romgaz	R1234567
F110	Renel	R7654321
F110	Renel	R7654321

Relaţia Tip Cheltuială

Cod Chelt.	Denumire Cheltuială
C15	Chelt pt. încălzire
C16	Chelt pt. bucătării
C10	Chelt cu iluminatul
C11	Chelt pt. func. liftului

FN3

- Nu exista dependente tranzitive, rezulta ca relatiile sunt in forma normala 3.

FN3 + Boyce-Codd

Să căutăm determinanţii din exemplul de mai sus:

Cod Furn. → Denumire, Cod fiscal
Cod Chelt. → Denumire cheltuială
Nr. Crt., Cod Furn., Cod Chelt. → Valoare

Toţi cei trei determinanţi sunt şi chei candidat. Deci exemplul de mai sus este în formă normală Boyce-Codd. Relaţiile în formă normală trei sunt în general şi în formă normală Boyce-Codd. În cazul în care relaţia nu este în formă normală Boyce-Codd, trecerea la BCNF se realizează prin ştergerea din relaţia iniţială a atributelor care sunt asociate unui determinant care nu este cheie candidat şi crearea unei noi relaţii cu aceste atribute şi determinantul lor.

Forma Normală Unu (1NF)

- Trebuie să căutăm toate intersecţiile de linii şi coloane, unde există repetiţii. Aceste repetiţii se pot elimina prin două modalităţi:
  - Crearea a noi înregistrări pentru fiecare valoare a repetiţiei, după care se caută o nouă cheie primară.
  - Ştergerea atributelor care conţin repetiţii şi crearea a unor noi relaţii care vor conţine atributele şterse, precum şi cheia principală din relaţia iniţială.

Forma Normală 2 (2NF)

- Se caută dependenţele parţiale de cheia principală, adică toate atributele care depind funcţional de un subset de atribute a cheii primare.
- Dacă cheia primară este compusă dintr-un singur atribut, atunci releţia este în forma normală doi.
- Dacă există dependenţe parţiale, vom şterge atributele care depind parţial de cheia principală şi creăm o relaţie nouă care să se compună din atributele şterse împreună cu determinantul lor.

Forma Normală 3 (3NF)

- Pentru a trece la forma normală trei, trebuie să eliminăm dependenţele tranzitive.
- Eliminarea se realizează prin ştergerea câmpurilor dependente tranzitiv de cheia primară din relaţia iniţială şi crearea unei noi relaţii cu aceste atribute şi determinantul lor.

Forma Normală Boyce-Codd (BCNF)

- Cerinţa la forma normală Boyce-Codd este ca fiecare determinant din relaţie să fie cheie candidat.
- În cazul în care nu este îndeplinită această cerinţă, vom şterge atributele dependente funcţional de determinantul care nu este cheie candidat şi creăm o nouă relaţie în care să avem atributele şterse şi determinantul lor.
- În unele cazuri trecerea la forma normală Boyce-Codd complică foarte mult baza de date, caz în care este de preferat rămânerea la forma normală trei.