

BAZE DE DATE RELAȚIONALE

Vitalie Cotelea

PREFAȚĂ

A trecut mai bine de un sfert de secol de la prima publicație a lui Codd E., cu care a început dezvoltarea teoriei bazelor de date relaționale. Însă, numai acum această teorie a devenit un fundament real pentru construirea sistemelor informatice eficiente. Adevăratele sisteme relaționale s-au afirmat pe piață începând cu anul 1984: sistemul DB2 (IBM Corp.) funcționează pe mainframe-urile IBM și, prin urmare, e portabil pe RS/6000; Oracle (Oracle Systems) a devenit un sistem de gestiune relațional portabil pe mai multe platforme; sistemul RDBMS (AWB) funcționează pe calculatoarele AT&T 3B; sistemele de gestiune Informix (Informix Software) și Sybase (Sybase Inc.) funcționează în mediul Unix. Mai multe sisteme de gestiune ale bazelor de date relaționale (FoxPro, Paradox etc.) au fost elaborate pentru calculatoarele personale.

Dezvoltarea teoriei bazelor de date relaționale a căpătat o amploare nemaivăzută în domeniul aplicării tehnicii de calcul. Au apărut o serie de reviste specializate în domeniul respectiv. Între elaborările teoretice și producerea sistemelor comerciale s-a creat un spațiu de cel puțin 20 ani. Acesta e un rar exemplu când necesitățile software considerabil depășesc capacitățile hardware. Rezultatele obținute în teoria relațională au influențat esențial sistemele de gestiune ce se bazează pe celelalte două modele de date: ierarhic și rețea. Modelul relațional de date e aplicat pe larg și în bazele de date deductive. Pe de altă parte, se observă convergența modelului relațional și tehnologiilor orientate pe obiecte.

"Revoluția relațională" a introdus mai multe idei valoroase în lumea bazelor de date. Printre acestea progrese tehnologice și beneficii ale sistemelor de gestiune ale bazelor de date pot fi menționate:

- **Tabelele** sunt un mijloc simplu de reprezentare a datelor. Ele permit programatorilor și utilizatorilor finali să-și organizeze datele în mod acceptabil. Extinderea modelului relațional a confirmat puterea de atracție a acestei reprezentări.
- **SQL** este un standard de limbaje de interpelări foarte comod. El e un limbaj nonprocedural de manipulare a datelor și a contribuit mult la creșterea popularității sistemelor de gestiune ale bazelor de date relaționale.
- **Operațiile orientate pe mulțimi** permit programatorilor și utilizatorilor ordinari să găsească și să actualizeze mari colecții de înregistrări fără a scrie programe speciale.
- **Joncțiunile** sunt instrumente puternice de asociere a înregistrărilor anterior independente. Utilizatorii pot crea noi seturi de înregistrări (așa-numitele tabele virtuale), apelând la joncțiune.
- **Interpelările interactive.** Căutarea și prelucrarea datelor în mod dinamic a adus la utilizarea largă a bazelor de date relaționale. Gestionarea tabelor, vizualizarea interactivă și îmbunătățirea interactivă a contribuit ca utilizatorul să-și dea votul pentru sistemele relaționale.
- **Consistența datelor.** Sistemele de gestiune relaționale asigură că nici un utilizator și nici o aplicație nu pot modifica baza de date, dacă modificarea e în contradicție cu constrângerile de integritate.

Modelul relațional are un obiectiv simplu: "... utilizatorii ... trebuie să fie protejați de cunoaștere a ... reprezentării interne (a datelor) ... Activitățile utilizatorilor la terminale și majoritatea aplicațiilor trebuie să rămână intacte, dacă reprezentarea internă ... este modificată". Acest fragment este din primul articol al lui Codd consacrat modelului relațional de date.

A contribuit, oare, modelul relațional ca bazele de date elaborate astăzi să realizeze acest obiectiv? Pe cât de adecvat modelul relațional de date poate exprima semantica lumii înconjurătoare? Cum poate fi proiectată o bază de date ce ar satisface unele criterii formulate apriori? Ce limbaje de interpelări pot fi utilizate în bazele de date relaționale?

Răspunsurile la aceste și la alte întrebări pot fi găsite în prezenta lucrare. Lucrarea, scrisă într-o manieră strictă, poate ajuta atât pe profesioniști cât și pe amatori să se familiarizeze cu principalele repere ale teoriei bazelor de date relaționale. Ideile expuse sunt însoțite de exemple luate dintr-un cadru real. Iar fiecare capitol (cu excepția primului capitol) sfârșește cu exerciții ce pot contribui la consolidarea cunoștințelor.

Lucrarea acoperă șapte subiecte principale.

Capitolul 1 este consacrat modelului relațional de date. Sunt considerate cele trei componente sau aspecte ale modelului relațional: structura de date, integritatea datelor și manipularea datelor. Capitolul include de asemenea convenția asupra termenilor și notațiilor utilizate în lucrare.

Capitolul 2 este o continuare a capitolului 1 în discuția asupra aspectului trei al modelului relațional de date. Limbajele de interpelări în bazele de date relaționale pot fi divizate în două clase: limbaje algebrice și limbaje bazate pe calculul predicatelor. Capitolul doi se referă numai la algebra relațională.

A doua clasă de limbaje are două versiuni cunoscute sub denumirile de calcul relațional orientat pe tuplu și calcul relațional orientat pe domeniu. Aceste versiuni și echivalența lor cu algebra relațională sunt studiate în capitolul 7.

În capitolul 3 este descrisă cea mai simplă și mai larg răspândită constrângere de integritate a modelului relațional - dependența funcțională. El include reguli de inferență (reguli independente, mulțimi închise și complete de reguli de inferență), derivări și diverse tipuri de acoperiri.

Capitolul 4 e consacrat dependențelor multivaloare, regulilor de inferență, dependențelor multivaloare incluse, dependențelor multivaloare noncontradictorii și problemei calității de membru pentru dependențele multivaloare. Acest capitol consideră de asemenea și cea mai generală constrângere de integritate, numită dependență joncțiune și problemele legate de utilizarea acestui tip de dependențe în schemele relaționale.

Capitolul 5 acoperă în detalii diverse forme normale și metode de proiectare ale schemelor bazelor de date.

Iar capitolul 6 e consacrat clasei de scheme aciclice ale bazelor de date relaționale. Sunt descrise proprietățile dezirabile ale schemelor aciclice, gradele de aciclicitate și sunt formulate condițiile sintactice de aciclicitate.

MODELUL RELAȚIONAL

Modelul relațional ca și orice alt model de date utilizat în proiectarea logică a bazelor de date eliberează utilizatorul de cunoașterea detaliilor despre structura fizică și metodele de acces la date. În afară de aceasta, el are două avantaje suplimentare: e simplu și elegant. Simplitatea sa constă în structurile de date omogene în formă de relații tabelare. Iar eleganța modelului se explică prin temelia sa științifică. El este riguros din punct de vedere matematic grație faptului că se sprijină pe bine puse la punct teoriile matematica relațiilor și logica de ordinul unu.

Modelul relațional a fost primul exemplu de model de date formal și a fost propus de E. Codd în 1970. Prin model datele utilizatorului sunt reprezentate și manipulate în mod abstract. Modelul de asemenea presupune tehnici ce ajută administratorul de a detecta și corecta posibilele probleme de proiectare ce pot apărea o dată cu pregătirea datelor pentru implementare într-un SGBD concret.

Orice model de date, conform unei sugestii a lui Codd, trebuie să se bazeze pe trei componente: structurile de date, constrângerile de integritate și operatorii de manipulare a datelor.

- **Structurile de date.** Structurile sunt definite de un limbaj de definire a datelor (data definition language). Datele în modelul relațional sunt structurate în relații bidimensionale. Elementele principale ale structurii relaționale sunt relațiile, tuplurile, atributele, domeniile.
- **Constrângerile de integritate.** Prin integritatea datelor se subînțelege că datele rămân stabile, în siguranță și corecte. Integritatea în modelul relațional este menținută de constrângeri interne care nu sunt cunoscute utilizatorului.
- **Manipularea datelor.** Relațiile pot fi manipulate utilizând un limbaj de manipulare a datelor (data manipulation language). În modelul relațional, limbajul folosește operatorii relaționali bazați pe conceptul algebrei relaționale. În afară de aceasta, există limbaje echivalente algebrei relaționale, cum ar fi calculul relațional orientat pe tuplu și calculul relațional orientat pe domeniu.

1.1. Structura relațională a datelor

Unul din avantajele modelului relațional rezidă în omogenitatea lui. Toate datele sunt structurate în tabele, fiecare linie ale căror are același format. Linia într-un tabel reprezintă un obiect (sau o relație dintre obiecte) din lumea înconjurătoare.

1.1.1. Atribute și domenii

În sistemele obișnuite de gestionare a fișierelor câmpul este cea mai mică unitate accesibilă de date. Se presupune că fiecare câmp poate conține un anumit tip de date (integer, real, character, string etc.), pentru care se specifică numărul necesar de octeți de memorie. Câmpul, bineînțeles, are și un nume. Făcând analogie, în modelul

relațional fiecare coloană a unei linii dintr-un tabel corespunde noțiunii de câmp din fișiere.

Definiția 1.1. Fie U o mulțime nevidă de elemente A_1, A_2, \dots, A_n , numite *nume de atribut* sau simplu *atribut*. Mulțimea $U = \{A_1, A_2, \dots, A_n\}$ se numește *universul* unei baze de date relaționale sau *mulțime universală*.

Definiția 1.2. *Domeniul* unui atribut A_i din U , $1 \leq i \leq n$, notat cu $\text{dom}(A_i)$, este o mulțime finită de valori de același tip care le poate primi atributul A_i .

Domeniul este *simplu*, dacă elementele sale sunt atomice (adică nu pot fi descompuse din punctul de vedere al SGBD-ului). Atributul ce are un domeniu de valori simplu se numește *atribut atomic*. Domeniul unei submulțimi R a universului U , se notează $\text{dom}(R)$, este uniunea tuturor domeniilor atributelor din R , adică $\text{dom}(R) = \bigcup_{A_i \in R} \text{dom}(A_i)$, unde $\text{dom}(\emptyset) = \emptyset$, dacă $R = \emptyset$.

Remarcă. Cu toate că elementele unui domeniu trebuie să fie de același tip, această restricție nu se extinde asupra elementelor din $\text{dom}(R)$.

În modelul relațional fiecare tabel se spune că reprezintă o relație. Atributele sunt niște identificatori pentru a diferenția și marca coloanele tabelului. Deci, câmpul sau numele de coloană e și atribut. Toate atributele ce apar într-un tabel trebuie să fie distincte și să fie incluse în universul U . Atributele au un caracter global în baza de date: dacă un nume denotă două coloane în tabele distincte în aceeași bază de date, atunci el reprezintă același atribut.

Relația tabelară cu i linii și j coloane are $i \cdot j$ elemente. Fiecare element este o valoare dintr-un domeniu simplu. Cu toate că atributele în universul U trebuie să fie distincte, domeniile acestor atribute nu trebuie neapărat să fie disjuncte. De exemplu, managerul în același timp poate fi funcționar. Deci domeniile atributelor MANAGER și FUNCȚIONAR nu sunt disjuncte, adică MANAGER și FUNCȚIONAR sunt definite pe același domeniu (cu toate că atributele respective pot avea și valori distincte).

Convenție. Mai departe vom utiliza următoarele notații. Universul $U = \{A_1, A_2, \dots, A_n\}$ și orice submulțime a lui, $R = \{A_{i1}, \dots, A_{ik}\}$, vor fi reprezentate ca string-uri, adică

$$U = A_1 \dots A_n, \\ R = A_{i1} \dots A_{ik}.$$

Vom folosi o notație mai simplă $A_i \subseteq U$, în loc de $\{A_i\} \subseteq U$ pentru " A_i este o submulțime a mulțimii U ". Reuniunea $Y \cup Z$ a două mulțimi Y și Z va fi reprezentată de simbolul YZ , unde operația binară uniunea, " \cup ", este omisă. Mulțimile Y și Z pot fi și mulțimi vide, fiindcă $\emptyset Z = Z$, $Y \emptyset = Y$ și $\emptyset \emptyset = \emptyset$.

Cu litere majuscule de la începutul alfabetului latin vom nota atributele singulare, iar cu cele de la sfârșitul alfabetului latin - mulțimi de atribute.

1.1.2. Tupluri

În sistemele cu fișiere o mulțime de câmpuri ce e concepută ca o unitate de salvare și căutare se numește înregistrare. Înregistrarea are un format specific și depinde de tipurile de date ale câmpurilor. O linie dintr-o relație tabelară corespunde înregistrării din fișiere și în teoria relațională se numește tuplu.

Definiția 1.3. Fie R o submulțime a universului U , $R \subseteq U$, unde $R \neq \emptyset$ și fie $\text{dom}(R)$ domeniul mulțimii R . *Tuplu* se numește o funcție, $t: R \rightarrow \text{dom}(R)$, din R în $\text{dom}(R)$, adică

$$t = \{(A_{i1}, a_1), \dots, (A_{ik}, a_k)\},$$

unde orice A_{ij} , $1 \leq j \leq k$, este un atribut în R și un argument al lui t , iar orice a_j , $1 \leq j \leq k$, este o valoare în $\text{dom}(A_{ij})$.

Considerăm o restricție asupra tuplului t .

Definiția 1.4. Fie $X = B_1 \dots B_m$ o submulțime proprie a mulțimii R , $X \subset R$, unde $X \neq \emptyset$. X -valoare a tuplului t , notată cu $t[X]$, este $t[X] = \{(B_j, b_j) | b_j = t(B_j) = t(A_{jp}), 1 \leq j \leq m, p \in \{1, \dots, k\}\}$. Dacă $X = A_{ij}$, $j \in \{1, \dots, k\}$, atunci A_{ij} -valoarea tuplului t se mai numește A_{ij} -componentă a tuplului t .

Ultima definiție ne spune, că $t(A_{ij}) = t[A_{ij}] = a_j$ pentru $a_j \in \text{dom}(A_{ij})$. Deci nu vom diferenția simbolurile $t(A_{ij})$ și $t[A_{ij}]$ pentru un atribut singular A_{ij} din U .

Pentru comoditate tuplul t și X -valoarea tuplului t vor fi notate

$$t = \langle a_1 \dots a_k | A_{i1} \dots A_{ik} \rangle \text{ și}$$

$$t[X] = \langle b_1 \dots b_m | B_1 \dots B_m \rangle,$$

respectiv. Însă, dacă coloanele tabelului ce corespund mulțimilor R și X sunt marcate cu attribute din R și X , iar ordinea atributelor ce marchează corespund ordinii atributelor în R și X , atunci notațiile tuplului t și X -valorii tuplului t pot fi simplificate respectiv

$$t = \langle a_1 \dots a_k \rangle \text{ și}$$

$$t[X] = \langle b_1 \dots b_m \rangle.$$

Deci putem reprezenta printr-un string nu numai o mulțime de attribute, dar și o mulțime de valori. Dar permutarea atributelor într-un tabel va trebui reflectată în tupluri, permutându-le componentele. Cu toate că string-urile ce reprezintă tuplurile inițial și final vor fi diferite, vom considera că aceste tupluri sunt identice.

În tuplul $t = \langle a_1 \dots a_k | A_{i1} \dots A_{ik} \rangle$ distingem două componente – string-ul de attribute $A_{i1} \dots A_{ik}$ care este invariant în timp și string-ul de valori $a_1 \dots a_k$, care este foarte dinamic. Partea invariantă a tuplului vom numi-o schema tuplului (uneori se notează $\text{sch}(t)$). Îndată ce am definit schema tuplului, expresia “tuplul asupra R ” devine clară și este echivalentă expresiei “tuplul t cu schema R ”.

Pentru comoditate notațională, un tuplu cu numele t și schema R se va nota uneori

$$t(R) = t(A_{i1}) t(A_{i2}) \dots t(A_{ik}).$$

Deci putem concepe tuplul $t(R)$ ca un *tuplu variabilă* asupra R și fiecare componentă $t(A_{ij})$, $1 \leq j \leq k$, ca un *domeniu variabilă*. Dacă tuplul $t(R)$ are o formă constantă, adică string-ul lui de valori este $\langle c_1 \dots c_k \rangle$ și aceste valori sunt în $\text{dom}(R)$, el se numește *tuplu constantă* asupra R .

1.1.3. Relații și scheme

Definiția 1.5. Fie R o submulțime a universului U . *Relația* r asupra R este o mulțime finită de tupluri cu schema R . *Aritatea* relației r este egală cu cardinalitatea mulțimii R . *Cardinalitatea* relației r este numărul de tupluri în ea.

Definiția 1.6. Fie $R \subseteq U$ și relația r asupra R . Mulțimea de attribute R se numește *schema* relației r (notată cu $\text{sch}(r) = R$).

Definiția 1.7. *Baza de date* relațională (sau simplu bază de date) este o mulțime finită de relații, $db = \{r_1, \dots, r_m\}$, unde r_i este o relație cu schema R_i , $1 \leq i \leq m$.

Definiția 1.8. Fie baza de date $db = \{r_1, \dots, r_m\}$. *Schema bazei de date* este mulțimea schemelor relațiilor ce formează baza de date, $Db = \{R_1, \dots, R_m\}$, unde $R_i = \text{sch}(r_i)$.

Deci schema unei relații este o expresie a proprietăților comune și invariante ale tuplurilor ce compun relația. Schema unei relații mai este cunoscută sub denumirea de *intensia* unei relații. Relația se mai numește *extensie*. Extensia reprezintă mulțimea tuplurilor care compun la un moment dat relația, mulțime care este variabilă în timp.

Din definițiile de mai sus putem conchide următoarele:

- (1) Într-o relație nu există coloane cu nume duplicate, fiindcă attributele A_{ij} , $1 \leq j \leq k$, sunt elemente ale mulțimii R_i .
- (2) Relația r_i nu are tupluri identice, fiindcă r_i este o mulțime de tupluri.
- (3) Ordinea tuplurilor în r_i este nesemnificativă, fiindcă r_i este o mulțime.
- (4) Ordinea coloanelor e nesemnificativă.
- (5) Valorile atributelor în r_i sunt atomice fiindcă domeniile sunt simple.

Relațiile ce se stochează fizic și formează baza de date se numesc *relații de bază*. Există, însă, și situații în care extensia nu se memorează în baza de date. Este cazul așa-numitelor *relații virtuale*, cunoscute și sub numele de *relații derivate* sau *viziuni*. Relația virtuală nu este definită explicit ca relația de bază, prin mulțimea tuplurilor componente, ci implicit pe baza altor relații. Relațiile de bază sunt proiectate de administratorul bazei de date, în timp ce viziunile sunt definite de utilizatorii bazei de date.

Relațiile asupra unei mulțimi de attribute pot avea un nume, sau pot să nu aibă, dacă ele sunt identificate în mod unic de schemele sale. Numele relației, de obicei, se scrie cu minuscule, de exemplu, relația r .

<i>studenți</i>	NUME	NOTĂ_MED	FACULTATE	DECAN
	Vasilache	7.8	Cibernetică	Popovici

<i>discipline</i>	FACULTATE	DISCIPLINĂ

<i>corp_didac</i>	DISCIPLINA	PROFESOR

<i>șarjă</i>	DISCIPLINA	TIP	ORE

Fig.1.1. Baza de date Universitatea

Exemplul 1.1. Fie baza de date “Universitatea” constă din patru relații *studenți*, *discipline*, *corp_didactic* și *șarjă* (vezi fig.1.1).

Primul tabel, reprezentând relația *studenți*, stochează numele, nota medie, facultatea și decanul asociate fiecărui student. Relația are patru attribute: NUME, NOTĂ_MED, FACULTATE, DECAN. Relația *discipline* afișează disciplinele ce se predau la diverse facultăți. Ea are două attribute FACULTATE și DISCIPLINĂ. Relația *corp_didac* specifică disciplinele predate de diferiți profesori. Ea are două attribute: DISCIPLINĂ și PROFESOR. Relația *șarjă* descrie disciplinele cu formele sale de predare și numărul de ore. Ea antrenează trei attribute: DISCIPLINĂ, TIP și ORE.

Datele în fiecare relație sunt atomice și sunt luate din domeniile (simple) atributelor corespunzătoare. FACULTATE în relațiile *studenți* și *discipline* reprezintă același atribut. De asemenea atributul DISCIPLINĂ figurează în trei relații: *discipline*, *corp_didac* și *șarjă*. Cele opt attribute prezente în relațiile descrise constituie universul:

$U = \text{NUME NOTĂ_MED FACULTATE DECAN DISCIPLINĂ PROFESOR TIP ORE}$. Așadar, attributele oricărei relații formează o submulțime a mulțimii universale U .

Domeniul atributului NUME constă din eventualele nume de familii, dar trebuie să conțină numaidecât și valori active, adică numele studenților ce actualmente își fac studiile la facultate. Domeniul atributului NOTĂ_MED conține numere pozitive. Dat fiind faptul că mulțimea de numere pozitive e infinită, fiecare NOTĂ_MED -valoare nu poate depăși valoarea maximă 10. Deci $\text{dom}(\text{NOTĂ_MED})$ poate fi menținut finit. Celelalte domenii se definesc similar. Nu e exclus faptul că un decan să fie student la o altă facultate. În cazul acesta domeniile active ale atributelor NUME și DECAN nu vor fi disjuncte.

Tuplurile relației *studenți* sunt definite pe mulțimea de attribute $R = \text{NUME NOTĂ_MED FACULTATE DECAN}$. Ele sunt concepute ca tupluri constante-valori ale tuplului variabilă $t(R) = t(\text{NUME}) t(\text{NOTĂ_MED}) t(\text{FACULTATE}) t(\text{DECAN})$. De exemplu, tuplul constantă $\langle \text{Vasilache 7.8. Cibernetică Popovici} \rangle$ arată că Vasilache este student la facultatea Cibernetică a cărei decan este Popovici și are nota medie 7.8. Considerăm $X = \text{NUME DECAN}$. Tuplul variabilă va fi $t[X] = t(\text{NUME}) t(\text{DECAN})$. Tuplul constantă definit pe schema X este derivat din tuplul $\langle \text{Vasilache 7.8. Cibernetică Popovici} \rangle$ al relației *studenți* și este $\langle \text{Vasilache Popovici} \rangle$.

În baza de date “Universitatea”, *studenți* este nume de relație. Schema relației *studenți* e $\text{NUME NOTĂ_MED FACULTATE DECAN}$.

Baza de date “Universitatea” constă din patru relații $\text{db} = \{\text{studenți, discipline, corp_didactic, șarjă}\}$. Schema bazei de date este mulțimea schemelor celor patru relații $\text{Db} = \{\text{NUME NOTĂ_MED FACULTATE DECAN, FACULTATE DISCIPLINĂ, DISCIPLINĂ PROFESOR, DISCIPLINĂ TIP ORE}\}$.

Relațiile *studenți, discipline, corp_didactic, șarjă* au aritate 4, 2, 2 și 3 respectiv. Ele formează relațiile de bază. Relația definită pe attributele $X = \text{NUME DECAN}$ e o relație virtuală.

1.2. Constrângeri de integritate

1.2.1. Tipuri de constrângeri

Constrângerile de integritate, numite și restricții de integritate definesc cerințele pe care trebuie să le satisfacă datele din baza de date pentru a putea fi considerate corecte, coerente în raport cu lumea reală pe care o reflectă.

Constrângerile sunt principalul mod de integrare a semanticii datelor în cadrul modelului relațional. Mecanismele de definire și verificare ale acestor restricții reprezintă instrumentele principale de control al semanticii datelor. În modelul relațional, constrângerile sunt studiate mai ales sub aspectul puterii lor de modelare și al posibilității de verificare a respectării lor.

Constrângerile de integritate pot fi divizate în linii mari în două grupuri: *constrângeri de comportament* și *dependențe* între date.

Constrângerile de comportament specifică caracteristicile independente ale unui atribut (sau domeniu). Ele exprimă semantica elementelor domeniilor. De exemplu, toate valorile atributului NOTĂ_MED trebuie să fie mai mare decât zero, dar nu poate depăși zece. Sau nici o persoană de vârstă 25 ani nu poate avea o vechime în muncă de 37 ani. Deci, conform acestei restricții, valorile atributului trebuie să se încadreze între anumite limite.

Al doilea tip de constrângeri specifică legătura dintre attribute (sau domenii). Aici putem identifica așa-numita dependență de submulțime. Considerăm, de exemplu, relația *personal* definită pe mulțimea de attribute {ANGAJAT SALARIU DEPARTAMENT MANAGER}. În relația *personal*, un manager este în același timp un angajat, dar nu orice angajat este manager. Deci avem că $\text{dom}(\text{MANAGER}) \subseteq \text{dom}(\text{ANGAJAT})$.

Dacă presupunem că angajatul are un singur salariu, se subordonează unui singur manager direct, lucrează deci într-un singur departament, atunci ANGAJAT-valorile determină în mod unic toate tuplurile în relația *personal*. Această constrângere este numită dependență funcțională. Dependențele funcționale vor fi studiate detaliat în capitolul 3. Alte tipuri de dependențe cum ar fi cele multivaloare și de joncțiune vor fi studiate în capitolul 4.

1.2.2. Chei

Aici vom examina constrângerile legate de noțiunea de cheie a relațiilor. Întrucât relația reprezintă o mulțime de tupluri, iar o mulțime nu poate conține elementele duplicate, relația nu poate prezenta tupluri identice. Deci tuplurile sunt unice și trebuie să existe posibilitatea identificării lor în cadrul unei relații. Identificarea unui tuplu fără a consulta toate componentele tuplului a dus la apariția noțiunii de cheie.

Definiția 1.9. Fie U mulțimea universală de attribute, $R \subseteq U$ și $R \neq \emptyset$. Mulțimea K de attribute, unde $K \subseteq R$, se numește *cheie* pentru schema R (sau pentru relația r cu schema R), dacă ea posedă următoarele proprietăți:

- (1) pentru orice două tupluri t_1 și t_2 din r avem $t_1[K] \neq t_2[K]$;
- (2) nici o submulțime K^1 proprie a lui K nu posedă proprietatea (1)

Proprietatea (1), numită restricția de unicitate a cheii, permite K -valorilor să identifice în mod unic toate tuplurile dintr-o relație. Însă respectarea proprietății de unicitate poate fi complicată, dacă însăși K conține o cheie K^1 și $K \neq K^1$. În acest caz, cu toate că attributele din K sunt suficiente de a atinge scopul, unele din ele nu sunt necesare, deci pot fi eliminate din cheie fără a se afecta unicitatea. Dacă, însă, K este o submulțime proprie a unei chei, atunci utilizarea a astfel de K -valori pentru căutarea datelor va descoperi tupluri ce coincid pe toate valorile atributelor din K .

Proprietatea (2) ne asigură că o cheie K constituie numai acele attribute ce sunt necesare și suficiente pentru a determina univoc pe celelalte. Cu alte cuvinte K -valorile întotdeauna asigură un grad exact de informație nici mai mult, nici mai puțin, pentru a găsi un tuplu unic într-o relație.

Definiția 1.10. Mulțimea de attribute ce posedă proprietatea (1) se numește *supercheie*.

Deci cheia este o supercheie minimală. Orice cheie e și supercheie. Afirmatia inversă nu e corectă.

Este evident că o mulțime vidă nu poate servi drept cheie a unei relații ce conține mai mult de un tuplu. Orice relație are cel puțin o cheie. La limită cheia este constituită fie dintr-un singur atribut, fie din totalitatea atributelor din schema relației respective.

Într-o relație pot exista mai multe chei. Se spune în acest caz că relația posedă mai multe *chei candidate*. În această situație administratorul bazei de date va stabili una din cheile candidate să servească în mod efectiv la identificarea unică a tuplurilor. Ea va

primi numele de *cheie primară*. Primare se vor numi și domeniile atributelor ce formează o cheie primară.

Definiția 1.11. Cheia primară a unei relații se numește *cheie simplă*, dacă este constituită dintr-un singur atribut, iar atunci când este formată din mai multe atribute este denumită *cheie compusă*.

Remarcă. Nu toate atributele unei chei compuse pot fi definite pe domenii primare.

Definiția 1.12. O *cheie externă* reprezintă un atribut (grup de atribute) dintr-o schemă R_i definit (definite) pe același (aceleași) domeniu (domenii) ca și cheia primară a altei scheme R_j . Relația r_i se numește *relație care referă*, iar r_j poartă numele de *relație referită*.

Unele atribute pot avea așa numitele valori nedefinite sau necunoscute notate cu "null". Însă sunt bine cunoscute constrângerile formulate prin următoarele reguli numite regulile de actualizare (inserare, modificare și eliminare) a relațiilor.

- (1) **Constrângerea entității:** cheia primară a unei relații de bază nu poate conține valori "null";
- (2) **Constrângerea referirii:** dacă atributul A al unei chei compuse a relației r_i este definit pe un domeniu primar, atunci trebuie să existe o relație de bază r_j cu o cheie primară B încât orice A-valoare din r_i să apară în calitate de B-valoare în r_j .

Constrângerea entității impune ca la înserarea unui tuplu, valoarea cheii să fie cunoscută, pentru a se putea verifica faptul că această valoare nu este deja încărcată (respectarea constrângerii de unicitate a cheii). Cu valori "null" cheia își pierde rolul de identificator de tuplu.

Constrângerea referențială impune ca într-o relație r_i , care referă o relație r_j , valorile cheii compuse să figureze printre valorile cheii primare din relația r_j pentru atributele compatibile.

Exemplul 1.2. Considerăm relațiile *studenți*, *discipline*, *corp_didac.*, *șarjă*.

În relația *studenți* singura cheie candidat este NUME, deci NUME e și cheia primară, iar dom(NUME) este domeniu primar pentru baza de date "Universitatea".

Considerăm relațiile *discipline*(FACULTATE DISCIPLINĂ) și *corp_didac*(DISCIPLINĂ PROFESOR). Fiindcă la orice facultate se predă cel puțin o disciplină și orice profesor predă cel puțin o disciplină, și similar, orice disciplină se predă măcar la o facultate și se predă cel puțin de un profesor, cheile primare ale acestor relații sunt compuse și constau din toate atributele corespunzătoare fiecărei relații.

În relația *șarjă*(DISCIPLINĂ TIP ORE), orice disciplină poate fi de trei tipuri (prelegeri, practică, laborator) și poate avea diferite ore de predare; unele discipline pot avea același tip și același număr de ore. E puțin probabil ca cheia relației *șarjă* să fie simplă. Putem presupune că cheia ei este compusă.

În acest exemplu, domeniul dom(NUME) este primar. Cheile compuse ale relațiilor *discipline*, *corp_didac*, *șarjă* nu sunt definite pe acest domeniu primar.

Conform regulii (1) atributele celor patru chei nu pot avea valori "null". Dat fiind faptul că nici o cheie din cele trei compuse nu sunt definite pe domeniul primar dom(NUME), exemplul dat nu ne demonstrează regula (2).

Exemplul 1.3. Considerăm relațiile *studenți* și *facultăți* din fig.1.2.

Presupunem că la o facultate își fac studiile mai mulți studenți și un student poate studia la mai multe facultăți concomitent. Cheia primară a relației *studenți* este compusă și constă din atributele NUME FACULTATE. Relația *facultăți* posedă două chei candidate: FACULTATE și DECAN. Fie FACULTATE cheia primară. Atunci atributul FACULTATE al relației *studenți* este cheie externă. Conform regulii (2) toate valorile atributului FACULTATE al relației care referă trebuie să se conțină în relația referită.

<i>studenți</i>	NUME	FACULTATE	AN
	n ₁	f ₁	a ₁
	n ₂	f ₁	a ₂
	n ₃	f ₂	a ₃

<i>facultăți</i>	FACULTATE	DECAN
	f ₁	d ₁
	f ₂	d ₂

Fig.1.2.

Spuneam mai sus că extensiile relațiilor se schimbă pe parcursul timpului. S-ar părea că pentru fiecare instanță a relației pot fi determinate cheile și supercheile. Dar schemele relațiilor, adică intensiile, trebuie să fie invariante și e de dorit ca cheile pe parcursul timpului să nu se schimbe. Cheile trebuie să rămână chei pentru orice eventuale extensii. Prin urmare determinarea cheii unei relații necesită cunoașterea semanticii relației respective, nu numai celei din momentul în care se stabilește cheia.

Convenție. Dacă relația posedă o singură cheie sau dorim să evidențiem numai cheia primară mai departe vom sublinia atributele ce formează această cheie. De exemplu, relația *r* cu schema ABCD și cheia AC se scrie *r*(A B C D). În cazul că relația posedă mai multe chei, atunci le vom scrie explicit: relația *r*(ABCD) are două chei candidate K₁=AC, K₂=B.

1.3. Operații de actualizare

Regulile de actualizare a bazei de date fac parte din cele trei componente ale modelului relațional de date. Vom examina cele trei operații de actualizare a datelor: *inserarea* datelor, *ștergerea* datelor și *modificarea* datelor.

Fie că în relația *r*(A₁ A₂ ...A_n) vrem să introducem date. Operația de inserție, a unui tuplu în relația *r* poate avea forma:

Add (*r*; <a₁a₂...a_n|A₁ A₂...A_n>).

În cazul că ordinea atributelor în relație e cunoscută, e acceptabilă o formă mai scurtă a operației:

Add (*r*; <a₁a₂...a_n>).

Scopul acestei operații constă în adăugarea unui tuplu într-o relație concretă. Rezultatul operației poate să eșueze din următoarele cauze:

- (1) tuplul de inserție e definit pe o mulțime de attribute ce nu corespunde schemei relației;
- (2) valorile componentelor tuplului nu sunt luate din domeniile corespunzătoare;

(3) în relație deja se găsește un tuplu cu asemenea componente cheie.
În toate aceste cazuri operația Add păstrează relația r intactă.

Operația de ștergere a datelor se utilizează pentru eliminarea conținutului relațiilor. Pentru relația r de mai sus, operația de ștergere se reprezintă:

$$\text{Del}(r; \langle a_1 a_2 \dots a_n | A_1 A_2 \dots A_n \rangle).$$

În cazul când numele de attribute sunt sortate, poate fi utilizată următoarea notăție scurtă:

$$\text{Del}(r; \langle a_1 a_2 \dots a_n \rangle).$$

În realitate, o parte de date din operația de mai sus poate fi redundantă pentru determinarea tuplului destinat ștergerii. E suficientă definiția valorilor atributelor cheie. Dacă $K=B_1 B_2 \dots B_m$ este cheia relației r , atunci e utilă următoarea formă a operației Del:

$$\text{Del}(r; \langle b_1 b_2 \dots b_m | B_1 B_2 \dots B_m \rangle).$$

Rezultatul operației de ștergere a tuplurilor nu se lasă mult așteptat. Tuplul e eliminat, dacă el este relație. În cazul că tuplul lipsește - relația rămâne intactă. Nu se pune nici o restricție asupra eliminării ultimului tuplu în relație: relația vidă se admite.

Uneori, în loc de eliminarea unui tuplu din relație și includerea unui alt tuplu e mai efectivă schimbarea unei părți a tuplului. Schimbarea se face cu operația de modificare. Dacă $C_1 C_2 \dots C_k \subseteq A_1 A_2 \dots A_n$, atunci operația de modificare poate avea forma:

$$\text{Ch}(r; \langle a_1 a_2 \dots a_n | A_1 A_2 \dots A_n \rangle; \langle c_1 c_2 \dots c_k | C_1 C_2 \dots C_k \rangle).$$

Dacă mulțimea $K=B_1 B_2 \dots B_m$ este cheia relației r , atunci expresia de mai sus poate fi redusă la:

$$\text{Ch}(r; \langle b_1 b_2 \dots b_m | B_1 B_2 \dots B_m \rangle; \langle c_1 c_2 \dots c_k | C_1 C_2 \dots C_k \rangle).$$

Operația de modificare este foarte utilă. Același rezultat poate fi obținut prin intermediul operațiilor de inserare și ștergere. Prin urmare, toate eșecurile operațiilor inserare și ștergere sunt specifice și operației modificare.

ALGEBRA RELAȚIONALĂ

Algebra relațională deseori e concepută ca un limbaj abstract de formulare a interpelărilor (cererilor) sau ca o colecție de operații pe relații având drept operanzi una sau mai multe relații și producând ca rezultat altă relație. Operațiile algebrei relaționale pot fi divizate în două grupuri: operațiile tradiționale pe mulțimi (vezi fig.2.1) ce consideră relațiile ca mulțimi de tupluri și operațiile relaționale native (fig.2.2).

Denumire	Simbol
Uniunea	\cup
Intersecția	\cap
Diferența	\setminus
Produsul (cartezian)	\times

Fig. 2.1. Operațiile tradiționale pe mulțimi

Denumire	Simbol
Proiecția	π
Selecția	σ
Joncțiunea	\bowtie
θ - joncțiunea	\bowtie_{θ}
Semijoncțiunea	\bowtie_{F}
Diviziunea	\div

Fig.2.2. Operațiile relaționale native

2.1. Operațiile tradiționale

2.1.1. Scheme relaționale compatibile

Operațiile binare asupra relațiilor: uniunea, intersecția și diferența, necesită ca operanzii (relațiile) să fie definiți pe scheme compatibile. Compatibilitatea schemelor se definește în felul următor.

Definiția 2.1. Vom spune că două relații $r(R)$ și $s(S)$ sunt *compatibile* (sau au scheme compatibile), dacă între R și S există o corespondență biunivocă f : pentru orice atribut A din R , există un atribut B în S încât $\text{dom}(A)=\text{dom}(B)$, $B=f(A)$ și $A=f^{-1}(B)$, unde f^{-1} este funcția inversă funcției f .

Remarcă. Două relații cu aceeași schemă sunt compatibile.

<i>vânzări</i>	FIRMĂ	ARTICOL
	f_1	a_1
	f_1	a_2

<i>articole</i>	ARTICOL	CULOARE
	a_1	c_1
	a_1	c_2

f_2	a_1	a_1	c_3
f_3	a_1	a_3	c_2
		a_2	c_1

<i>furnizori</i>	ARTICOL	FURNIZOR
	a_1	f_1
	a_1	f_3
	a_2	f_1
	a_3	f_3

Fig.2.3.

Exemplul 2.1. Fie baza de date din fig.2.3 ce constă din trei relații: *vânzări*(FIRMĂ ARTICOL), *articole*(ARTICOL CULOARE), *furnizori*(ARTICOL FURNIZOR). Schemele relațiilor *vânzări* și *articole* nu sunt compatibile, în timp ce schemele relațiilor *vânzări* și *furnizori* sunt compatibile. Ultimele relații sunt definite pe attribute ce primesc valori din aceleași domenii. Valorile active sunt totuși diferite, fiindcă un furnizor poate să nu fie firmă și viceversa.

2.1.2. Uniunea

Uniunea a două relații presupune că schemele lor sunt compatibile.

r	A	B	C
	a_1	b_1	c_1
	a_1	b_2	c_3
	a_2	b_1	c_2

s	A	B	C
	a_1	b_1	c_1
	a_1	b_1	c_2
	a_1	b_2	c_3
	a_3	b_2	c_3

Fig.2.4. Relațiile $r(A\ B\ C)$ și $s(A\ B\ C)$

q	A	B	C
	a_1	b_1	c_1
	a_1	b_1	c_2
	a_1	b_2	c_3
	a_2	b_1	c_2
	a_3	b_2	c_3

Fig.2.5. Relația $q = r \cup s$

Definiția 2.2. *Uniunea* a două relații compatibile $r(R)$ și $s(S)$, notată cu $r \cup s$, e o relație definită pe schema R sau S și constă din tuplurile ce aparțin relațiilor r sau s , adică

$$r \cup s = \{t \mid t \in r \vee t \in s\}.$$

Exemplul 2.2. Fie relațiile $r(A\ B\ C)$ și $s(A\ B\ C)$ din fig.2.4. Relația din fig.2.5 este $q = r \cup s$.

Operația uniunea are două proprietăți. Ea e comutativă, adică $r \cup s = s \cup r$. Ea este și asociativă, adică $(r \cup s) \cup q = r \cup (s \cup q)$ pentru relațiile mutual compatibile r, s

și q . Prin urmare, în expresiile ce conțin o cascadă de operații uniunea, parantezele pot fi omise fără a provoca ambiguități. Deci, dacă avem k relații compatibile r_1, r_2, \dots, r_k , uniunea acestor relații poate fi notată cu $\cup(r_1, r_2, \dots, r_k)$.

Operația uniunea are două cazuri speciale. Pentru orice relație $r(R)$ au loc: $r \cup \emptyset = r$ și $r \cup s = s$, dacă $r \subseteq s$.

2.1.3. Intersecția

Similar uniunii, intersecția a două relații cere ca operandii să fie relații cu scheme compatibile.

Definiția 2.3. Intersecția a două relații compatibile $r(R)$ și $s(S)$, notată cu $r \cap s$, este o relație definită pe schema R sau S și constă din tuplurile ce aparțin concomitent relațiilor r și s , adică

$$r \cap s = \{t \mid t \in r \ \& \ t \in s\}.$$

Exemplul 2.3. Fie relațiile $r(A \ B \ C)$ și $s(A \ B \ C)$ din fig.2.4. Relația $q = r \cap s$ este prezentată în fig.2.6.

q	A	B	C
	a ₁	b ₁	c ₁
	a ₁	b ₂	c ₃

Fig.2.6

2.1.4. Diferența

Operația diferența presupune că operandii sunt relații cu scheme compatibile.

Definiția 2.4. Diferența a două relații compatibile $r(R)$ și $s(S)$, notată cu $r \setminus s$, este o relație definită pe mulțimea de atribute R sau S și are în calitate de tupluri, toate tuplurile din relația r ce nu sunt în s , adică

$$r \setminus s = \{t \mid t \in r \ \& \ t \notin s\}.$$

Exemplul 2.4. Fie relațiile $r(A \ B \ C)$ și $s(A \ B \ C)$ din fig.2.4. Relațiile $q_1 = r \setminus s$, și $q_2 = s \setminus r$ sunt prezentate în fig.2.7.

q ₁	A	B	C
	a ₂	b ₁	c ₂

q ₂	A	B	C
	a ₁	b ₁	c ₂
	a ₃	b ₂	c ₃

Fig.2.7.

Din exemplul de mai sus observăm că diferența nu se bucură de proprietatea comutativă, adică $r \setminus s \neq s \setminus r$. Totodată, nu e nici asociativă, adică $(r \setminus s) \setminus q \neq r \setminus (s \setminus q)$, fiindcă $(r \setminus s) \setminus q = r \setminus (s \cup q)$ pentru orice relații mutual compatibile r, s , și q .

tup	A	B	C
	a ₁	b ₁	c ₁
	a ₁	b ₁	c ₂

\bar{r}	A	B	C
	a ₁	b ₁	c ₂
	a ₁	b ₁	c ₃

a ₁	b ₁	c ₃
a ₁	b ₂	c ₁
a ₁	b ₂	c ₂
a ₁	b ₂	c ₃
a ₂	b ₁	c ₁
a ₂	b ₁	c ₂
a ₂	b ₁	c ₃
a ₂	b ₂	c ₁
a ₂	b ₂	c ₂
a ₂	b ₂	c ₃
a ₃	b ₁	c ₁
a ₃	b ₁	c ₂
a ₃	b ₁	c ₃
a ₃	b ₂	c ₁
a ₃	b ₂	c ₂
a ₃	b ₂	c ₃

a ₁	b ₂	c ₁
a ₁	b ₂	c ₂
a ₂	b ₁	c ₁
a ₂	b ₁	c ₃
a ₂	b ₂	c ₁
a ₂	b ₂	c ₂
a ₂	b ₂	c ₃
a ₃	b ₁	c ₁
a ₃	b ₁	c ₂
a ₃	b ₁	c ₃
a ₃	b ₂	c ₁
a ₃	b ₂	c ₂
a ₃	b ₂	c ₃

Fig.2.8.

Diferența a două relații are patru cazuri speciale. Un caz este $\emptyset \setminus r = \emptyset$; altul e $r \setminus \emptyset = r$ pentru orice relație $r(R)$. Celelalte cazuri le vom examina în următoarele două secțiuni.

2.1.5. Complementul

Definiția 2.5. Fie relația $r(R)$. Notăm prin $\text{tup}(R)$ mulțimea tuturor tuplurilor asupra atributelor schemei R și a domeniilor lor. *Complementul* relației r , notat cu $\neg r$, este

$$\neg r = \text{tup}(R) \setminus r.$$

Exemplul 2.5. Fie relația $r(A \ B \ C)$ din fig.2.4 și fie $\text{dom}(A) = \{a_1, a_2, a_3\}$, $\text{dom}(B) = \{b_1, b_2\}$, $\text{dom}(C) = \{c_1, c_2, c_3\}$. Atunci $\text{tup}(A \ B \ C)$ și $\neg r$ sunt cele din fig.2.8.

Este clar că, dacă pentru un atribut A din R domeniul $\text{dom}(A)$ este infinit, atunci și $\neg r$ va fi infinită și deci nu va fi o relație conform definiției noastre. Pentru lichidarea acestui dezavantaj se introduce noțiunea de complement activ.

atup	A	B	C
	a ₁	b ₁	c ₁
	a ₁	b ₁	c ₂
	a ₁	b ₁	c ₃
	a ₁	b ₂	c ₁
	a ₁	b ₂	c ₂
	a ₁	b ₂	c ₃
	a ₂	b ₁	c ₁
	a ₂	b ₁	c ₂
	a ₂	b ₁	c ₃
	a ₂	b ₂	c ₁
	a ₂	b ₂	c ₂
	a ₂	b ₂	c ₃

$\neg r$	A	B	C
	a ₁	b ₁	c ₂
	a ₁	b ₁	c ₃
	a ₁	b ₂	c ₁
	a ₁	b ₂	c ₂
	a ₂	b ₁	c ₁
	a ₂	b ₁	c ₃
	a ₂	b ₂	c ₁
	a ₂	b ₂	c ₂
	a ₂	b ₂	c ₃

Fig.2.9.

2.1.6. Complementul activ

Versiunea modificată a complementului unei relații, complementul activ, întotdeauna va produce o relație.

Definiția 2.6. Fie r o relație asupra schemei R , $A \in R$ și $\text{adom}(A) = \{a | a \in \text{dom}(A) \text{ și } \exists t \in r \text{ și } t[A] = a\}$. Mulțimea de valori $\text{adom}(A)$ se numește *domeniul activ* al atributului A . Notăm cu $\text{atup}(R)$ mulțimea tuturor tuplurilor asupra atributelor schemei R și a domeniilor lor active. Atunci *complementul activ*, notat cu $\sim r$, este $\sim r = \text{atup}(R) \setminus r$.

Exemplul 2.6. Fie relația $r(A \ B \ C)$ din fig.2.4. Atunci $\text{adom}(A) = \{a_1, a_2\}$, $\text{adom}(B) = \{b_1, b_2\}$, $\text{adom}(C) = \{c_1, c_2, c_3\}$. Relațiile $\text{atup}(A \ B \ C)$ și $\sim r$ sunt prezentate în fig.2.9.

2.1.7. Produsul cartezian

Definiția 2.7. *Produsul cartezian* a două relații $r(A_1 \dots A_n)$ și $s(B_1 \dots B_m)$, notat cu $r \times s$, este o mulțime de tupluri (și nu întotdeauna o relație) definite pe mulțimea de attribute $A_1 \dots A_n \ B_1 \dots B_m$. Tuplurile reprezintă toate posibilele asociații de tupluri din r și s : dacă $t_r \in r$ și $t_s \in s$, atunci concatenația $t_r t_s$ este un tuplu în $r \times s$; pentru orice pereche de tupluri t_r și t_s din r și s , respectiv, există un tuplu t în $r \times s$ încât $t[A_i] = t_r[A_i]$, $1 \leq i \leq n$ și $t[B_j] = t_s[B_j]$, $1 \leq j \leq m$.

r	A	B	C
	a ₁	b ₁	c ₁
	a ₁	b ₂	c ₃
	a ₂	b ₁	c ₂

s	D	E
	d ₁	e ₁
	d ₁	e ₂

Fig.2.10.

Exemplul 2.7. Fie relațiile $r(A \ B \ C)$ și $s(D \ E)$ din fig.2.10. Produsul cartezian $q = r \times s$ arată ca în fig.2.11.

Produsul cartezian a două relații nevide nu întotdeauna produce o relație. Rezultatul e o relație, dacă ambii operanzi sunt relații cu scheme nevide și disjuncte (vezi exemplul 2.7). Dacă, însă, relațiile operanzi au scheme vide sau nondisjuncte, atunci produsul cartezian nu este o relație. Această problemă poate fi soluționată cu ajutorul operației atribuirea, care de fapt produce schimbarea numelor atributelor.

q	A	B	C	D	E
	a ₁	b ₁	c ₁	d ₁	e ₁
	a ₁	b ₁	c ₁	d ₁	e ₂
	a ₁	b ₂	c ₃	d ₁	e ₁
	a ₁	b ₂	c ₃	d ₁	e ₂
	a ₂	b ₁	c ₂	d ₁	e ₁
	a ₂	b ₁	c ₂	d ₁	e ₂

Fig.2.11.

Produsul cartezian nu este o operație comutativă. În schimb se bucură de proprietatea asociativă. Prin urmare, dacă avem o cascadă de produse carteziane $r_1 \times (r_2 \times (\dots r_k))$, ea poate fi reprezentată în formă de prefix $\times (r_1, r_2, \dots, r_k)$. Schemele relațiilor r_i , $1 \leq i \leq k$, trebuie să fie disjuncte două câte două. În caz contrar rezultatul nu va fi o relație.

2.1.8. Atribuirea

Fie $r(R)$ și $s(S)$ două relații cu scheme compatibile și $R \neq S$. Pentru a aplica asupra lor operațiile binare tradiționale se face reatribuirea relațiilor pentru a renumi diferența de atribute $R \setminus S$ sau $S \setminus R$. Atribuirea se utilizează și pentru păstrarea rezultatelor intermediare.

Definiția 2.8. Fie r o relație cu schema $A_1 \dots A_n$ și $\{B_1, \dots, B_n\}$ o mulțime de atribute compatibile, adică $\text{dom}(B_i) = \text{dom}(A_i)$, $1 \leq i \leq n$. O nouă relație $s(B_1 \dots B_n)$ compatibilă cu r se poate defini prin atribuirea: $s(B_1 \dots B_n) := r(A_1 \dots A_n)$. Extensia relației s este extensia relației r : $t_s \in s$ atunci și numai atunci, când $\exists t_r \in r$, unde $t_s[B_i] = t_r[A_i]$, $1 \leq i \leq n$.

r	A	B	C
	a ₁	b ₁	c ₁
	a ₁	b ₂	c ₃
	a ₂	b ₁	c ₂

s	A	D	C
	a ₁	b ₁	c ₁
	a ₁	b ₂	c ₃
	a ₂	b ₁	c ₂

Fig.2.12.

Următorul exemplu ilustrează utilizarea operației atribuirea pentru renumirea atributelor unei scheme.

Exemplul 2.9. Fie relația $r(A \ B \ C)$ din fig.2.12. Atunci, $s(A \ D \ C) := r(A \ B \ C)$.

r	A	B	C
	a ₁	b ₁	c ₁
	a ₁	b ₂	c ₃
	a ₂	b ₁	c ₂

s	A	B	C
	a ₁	b ₁	c ₁
	a ₁	b ₁	c ₂
	a ₁	b ₂	c ₃
	a ₃	b ₂	c ₃

q ₁	A	B	C
	a ₁	b ₁	c ₁
	a ₁	b ₁	c ₂
	a ₁	b ₂	c ₃
	a ₂	b ₁	c ₂
	a ₃	b ₂	c ₃

q ₂	A	B	C
	a ₁	b ₁	c ₁
	a ₁	b ₂	c ₃

q	A	B	C
	a ₁	b ₁	c ₂
	a ₂	b ₁	c ₂
	a ₃	b ₂	c ₃

Fig.2.13.

În următorul exemplu, operația atribuirea se folosește pentru păstrarea rezultatelor intermediare.

Exemplul 2.10. Fie relațiile $r(A\ B\ C)$ și $s(A\ B\ C)$ din fig.2.13. Atunci $q := (r \cup s) \setminus (r \cap s)$. Relația q s-a construit aplicând consecutivitatea de operații: $q_1 := r \cup s$, $q_2 := r \cap s$, $q = q_1 \setminus q_2$.

2.2. Operațiile relaționale native

2.2.1. Proiecția

Proiecția e o operație unară.

Definiția 2.9. Fie r o relație cu schema R și $X \subseteq R$. *Proiecția* relației r asupra mulțimii de atribute X , notată cu $\pi_X(r)$, e o relație cu schema X ce constă din X -valorile tuturor tuplurilor din r :

$$\pi_X(r) = \{t[X] \mid t \in r\}.$$

Tupluri distincte din r pot deveni identice, când se proiectează pe o mulțime de atribute. Tuplurile duplicate în relația rezultat se elimină.

Exemplul 2.11. Fie relația $r(A\ B\ C)$ din fig.2.14. Atunci $s = \pi_{A,C}(r)$.

r	A	B	C
	a	10	1
	a	20	1
	b	30	1
	b	40	2

s	A	C
	a	1
	b	1
	b	2

Fig.2.14.

Există două cazuri speciale:

- (1) $X = R$. Atunci $\pi_X(r) = r$.
- (2) $r = \emptyset$. Atunci $\pi_X(r) = \emptyset$.

Dacă mulțimea de atribute $X = \emptyset$, atunci proiecția $\pi_{\emptyset}(r)$ este indefinită, fiindcă schema unei relații nu poate fi o mulțime vidă. Schema unei relații, produsă de operația proiecția, are cel puțin un atribut.

Pentru cazul când $R \subset X$, operația proiecția iarăși este indefinită. Mulțimea asupra căreia se face proiecția nu poate fi mai largă decât schema relației inițiale.

Fie relația $r(R)$ și $Y \subseteq X \subseteq R$. Atunci $\pi_Y(\pi_X(r)) = \pi_Y(r)$. Dacă $X = Y$, atunci $\pi_X(\pi_Y(r)) = \pi_X(r) = \pi_Y(r)$.

2.2.2. Selecția

Selecția este o operație unară. Pentru selectarea unor tupluri dintr-o relație e necesară specificarea condițiilor de selectare. În rezultat se obține o relație ce e o submulțime de tupluri a relației inițiale. Fie că condiția de selecție se notează prin formula calculului propozițional, F , definită recursiv:

- (1) $A \theta B$ și $A \theta a$ sunt formule, unde A și B sunt atribute compatibile și $a \in \text{dom}(A)$, iar $\theta \in \{=, \neq, <, \leq, >, \geq\}$. Aceste formule sunt atomice.

- (2) Dacă G și H sunt formule, atunci conjuncția $G \& H$, disjuncția $G \vee H$, negațiile $\neg G$ și $\neg H$ sunt formule.
 (3) Nimic altceva nu e formulă.

Definiția 2.10. Vom spune că formula F e *aplicabilă* relației $r(R)$, dacă orice constantă c din F este în $\text{dom}(R)$, și orice atribut A din F este în R . O relație r satisface F (sau F e validă în r), dacă F e aplicabilă relației r și orice tuplu $t \in r$ satisface formula F în sensul că formula G obținută prin substituirea oricărui atribut A din F cu A -valoarea tuplului t are valoarea adevăr.

Definiția 2.11. Selecția relației $r(R)$ conform formulei F , unde F e aplicabilă relației $r(R)$, e o submulțime a relației $r(R)$, notată cu $\sigma_F(r)$, ce constă din toate tuplurile $t \in r$ ce satisfac F , adică

$$\sigma_F(r) = \{t \mid t \in r \ \& \ F(t)\}.$$

Exemplul 2.12. Fie $r(A \ B \ C \ D)$ din fig.2.15. Atunci $s = \sigma_{((A=B) \ \& \ (D>5))}(r)$.

r	A	B	C	D
	a	a	1	7
	b	b	5	5
	b	b	12	3
	b	b	23	10

s	A	B	C	D
	a	a	1	7
	b	b	23	10

Fig.2.15.

Există două cazuri speciale ale selecției.

- (1) Dacă F e o formulă ce nu e compusă nici dintr-o formulă atomică, adică e o formulă nulă, atunci $\sigma_F(r) = r$. În acest caz asupra tuplurilor $t \in r$ nu se impune nici o constrângere pentru selecție.
 (2) Dacă $r(R) = \emptyset$, atunci $\sigma_F(r) = \emptyset$ pentru orice formulă F , fiindcă F e validă în orice relație vidă.

Este evident că $\sigma_G(\sigma_F(r)) = \sigma_{G \& F}(r)$. Întrucât conjuncția este comutativă, adică $\sigma_{F \& G}(r) = \sigma_{G \& F}(r)$, atunci și compoziția a două selecții este comutativă. Deci $\sigma_G(\sigma_F(r)) = \sigma_F(\sigma_G(r))$.

Operația selecția este distributivă în raport cu operațiile binare tradiționale pe mulțimi. Fie r și s două relații compatibile și $\gamma \in \{\cup, \cap, \setminus\}$, atunci $\sigma_F(r \gamma s) = \sigma_F(r) \gamma \sigma_F(s)$. Să arătăm, de exemplu, că $\sigma_F(r \cup s) = \sigma_F(r) \cup \sigma_F(s)$. Într-adevăr:

$$\begin{aligned} \sigma_F(r \cup s) &= \sigma_F(\{t \mid t \in r \vee t \in s\}) = \\ &= \{t^1 \mid t^1 \in \{t \mid t \in r \vee t \in s\} \ \& \ F(t^1)\} = \\ &= \{t \mid t \in r \ \& \ F(t)\} \cup \{t \mid t \in s \ \& \ F(t)\} = \sigma_F(r) \cup \sigma_F(s). \end{aligned}$$

Trebuie menționat că selecția nu comutează cu operația complement. Însă selecția comutează cu proiecția, dacă sunt respectate unele condiții. Fie r o relație cu schema R , $X \subseteq R$, și fie F o formulă ce e satisfăcută de tuplurile $t[X]$. Atunci $\pi_X(\sigma_F(r)) = \sigma_F(\pi_X(r))$. Într-adevăr:

$$\pi_X(\sigma_F(r)) = \pi_X(\{t \mid t \in r \ \& \ F(t)\}) = \{t^1[X] \mid t^1 \in \{t \mid t \in r \ \& \ F(t)\}\} =$$

$$\{t[X] \mid t \in r \& F(t)\} = \{t^1 \mid t^1 \in \{t[X] \mid t \in r\} \& F(t^1)\} = \\ \{t^1 \mid t^1 \in \pi_x(r) \& F(t^1)\} = \sigma_F(\pi_x(r)).$$

2.2.3. θ -joncțiunea

Definiția 2.12. Fie $r(R)$ și $s(S)$ două relații, $R \cap S = \emptyset$, $A \in R$, $B \in S$ și fie θ un element din mulțimea $\{=, \neq, <, \leq, >, \geq\}$. Presupunem că atributele A și B sunt compatibile, adică $\text{dom}(A) = \text{dom}(B)$. θ -joncțiunea relațiilor $r(R)$ și $s(S)$, notată cu $r \mid x \mid_{A\theta B S}$, este o mulțime de tupluri concatenate de forma $t_r t_s$, unde $t_r \in r$, $t_s \in s$ și $t_r(A) \theta t_s(B)$, adică:

$$r \mid x \mid_{A\theta B S} = \{t_r t_s \mid t_r \in r \& t_s \in s \& t_r(A) \theta t_s(B)\}.$$

Condiția $R \cap S = \emptyset$ în definiție este necesară. Dacă $R \cap S \neq \emptyset$, atunci θ -joncțiunea nu este o relație. În cazul când θ este "=", θ -joncțiunea se mai numește *echijoncțiune*.

r	A	B	C	s	D	E
	a ₁	b ₁	4		3	e ₁
	a ₁	b ₂	2		4	e ₁
	a ₂	b ₁	6		1	e ₂

Fig.2.16.

q	A	B	C	D	E
	a ₁	b ₁	4	1	e ₂
	a ₁	b ₁	4	3	e ₁
	a ₁	b ₂	2	1	e ₂
	a ₂	b ₁	6	3	e ₁
	a ₂	b ₁	6	4	e ₁
	a ₂	b ₁	6	1	e ₂

Fig.2.17.

Exemplul 2.13. Fie relațiile $r(A \ B \ C)$ și $s(D \ E)$ din fig.2.16, unde $\text{dom}(C) = \text{dom}(D)$. În fig.2.17 relația $r \mid x \mid_{C>D S}$ este prezentă.

Operația θ -joncțiunea poate fi exprimată prin operațiile produsul cartezian și selecția. Rezultatul unei θ -joncțiuni este același cu rezultatul unei selecții operate asupra unui produs cartezian, adică $r \mid x \mid_{A\theta B S} = \sigma_{A\theta B}(r \times s)$.

Să observăm că operația selecția poate fi simulată prin operațiile θ -joncțiunea și proiecție. Fie relația $r(R)$. Pentru a calcula $\sigma_{A\theta a}(r)$ se construiește o relație s definită pe un singur atribut, A . Relația s conține un singur tuplu componenta căruia are valoarea a pentru atributul A . Atunci $\sigma_{A\theta a}(r) = \pi_R(r \mid x \mid_{A\theta a S})$.

2.2.4. Joncțiunea (Joncțiunea naturală)

Definiția 2.13. Fie două relații $r(R)$ și $s(S)$. *Joncțiunea* relațiilor r și s (notația uzuală $r \bowtie s$) este o relație cu schema RS . Tuplul t aparține relației rezultat, dacă există tuplurile t_r și t_s în r și s , respectiv, și satisfac $t[R]=t_r$ și $t[S]=t_s$, adică

$$r \bowtie s = \{t \mid t[R] = t_r \ \& \ t[S] = t_s \ \& \ t_r \in r \ \& \ t_s \in s\}.$$

Deci, fiecare tuplu din relația rezultat este o concatenare a unui tuplu din r cu un tuplu din s ce au $(R \cap S)$ -valori egale. Atributele cu același nume în schema relației rezultat se iau o singură dată.

Exemplul 2.14. În fig.2.18 sunt afișate relațiile $r(A \ B \ C)$, $s(B \ C \ D)$ și $q(A \ B \ C \ D)$, unde $q = r \bowtie s$.

Dacă R și S sunt disjuncte, $R \cap S = \emptyset$, atunci joncțiunea relațiilor r și s este identică cu produsul cartezian al lor, adică $r \bowtie s = r \times s$.

Dacă $R \cap S \neq \emptyset$ și $|R \cap S| = k$, atunci joncțiunea poate fi redată prin operațiile proiecția, selecția și produsul cartezian: $r \bowtie s = \pi_{RS} (\sigma_F (r \times s))$, unde $F = (r.A_1 = s.A_1) \ \& \ (r.A_2 = s.A_2) \ \& \ \dots \ \& \ (r.A_k = s.A_k)$ pentru $A_i \in R \cap S$, $1 \leq i \leq k$.

Dacă $R=S$, atunci $r \bowtie s = r \cap s$. Într-adevăr:

$$\pi_{RS} (\sigma_F (r \times s)) = \pi_R (\sigma_F (r \times s)) = \pi_R ((r \cap s) \times (r \cap s)) = r \cap s.$$

Operația joncțiunea nu este comutativă. În schimb, ea se bucură de proprietatea asociativă. Prin urmare, o cascadă de joncțiuni $(r_1 \bowtie (r_2 \bowtie (\dots r_k)))$ poate fi prefixată, adică $\bowtie (r_1, r_2, \dots, r_k)$.

Din exemplul 2.14 se vede că nu toate tuplurile relațiilor r și s participă la joncțiune.

r	A	B	C
	a ₁	b ₁	c ₁
	a ₁	b ₂	c ₁
	a ₂	b ₁	c ₂

s	B	C	D
	b ₁	c ₁	d ₁
	b ₁	c ₁	d ₂
	b ₁	c ₂	d ₃
	b ₂	c ₂	d ₄

q	A	B	C	D
	a ₁	b ₁	c ₁	d ₁
	a ₁	b ₁	c ₁	d ₂
	a ₂	b ₁	c ₂	d ₃

Fig.2.18.

s ₁	A	B
	a ₁	b ₁
	a ₂	b ₁

s ₂	B	C
	b ₁	c ₁
	b ₁	c ₂

s ₃	A	C
	a ₁	c ₁
	a ₂	c ₁

q	A	B	C
	a ₁	b ₁	c ₁
	a ₂	b ₁	c ₂

Fig.2.19

Definiția 2.14. Fie relațiile $s_1(S_1), \dots, s_k(S_k)$. Considerăm o consecutivitate de tupluri t_1, \dots, t_k , unde $t_i \in s_i(S_i)$, $1 \leq i \leq k$. Tuplurile t_1, \dots, t_k se numesc *jonctionabile*, dacă există un tuplu t definit pe mulțimea de atribute $S_1 \cup \dots \cup S_k$ și $t[S_i] = t_i$, $1 \leq i \leq k$. În caz contrar se numesc *nonjonctionabile*.

Exemplul 2.15. Fie relațiile $s_1(A B)$, $s_2(B C)$, $s_3(A C)$ și $q(A B C)$ din fig.2.19, unde $q = s_1 \bowtie s_2 \bowtie s_3$. Tuplurile $\langle a_1, b_1 \rangle$, $\langle b_1, c_1 \rangle$, și $\langle a_1, c_1 \rangle$ sunt jonctionabile. De asemenea $\langle a_2, b_1 \rangle$, $\langle b_1, c_2 \rangle$, și $\langle a_2, c_1 \rangle$ sunt jonctionabile, dar tuplurile $\langle a_2, b_1 \rangle$, $\langle b_1, c_2 \rangle$, și $\langle a_1, c_1 \rangle$ sunt nonjonctionabile.

Să examinăm acum legătura dintre joncțiune și uniune.

Fie relațiile $r(R)$, $r^1(R)$ și $s(S)$. Atunci are loc următoarea egalitate:

$$(r^1 \cup r) \bowtie s = (r \bowtie s) \cup (r^1 \bowtie s).$$

Într-adevăr, notăm părțile stânga și dreapta cu q și q^1 respectiv. Fie $t \in q$. Atunci există în $r^1 \cup r$ și s două tupluri jonctionabile t_r și t_s pentru care $t[R] = t_r$ și $t[S] = t_s$. Dacă $t_r \in r$, atunci $t_s \in r \bowtie s$; dacă, însă $t_r \in r^1$, atunci $t \in r^1 \bowtie s$. Deci $t \in q^1$ și prin urmare $q \subseteq q^1$.

Acum presupunem că $t \in q^1$. Atunci $t \in r \bowtie s$ sau $t \in r^1 \bowtie s$. În ambele cazuri rezultă că $t \in (r^1 \cup r) \bowtie s = q$. Deci $q^1 \subseteq q$. Prin urmare $q = q^1$.

Să cercetăm acum legătura dintre proiecție și joncțiune. Fie două relații $r(R)$ și $s(S)$. Notăm $q = r \bowtie s$ (din definiția operației joncțiunea schema relației q este RS). Proiectăm relația q asupra mulțimii de atribute R : $r^1 = \pi_R(q)$. În ce corelație se află r^1 și r ? Răspuns: $r^1 \subseteq r$. Într-adevăr, fie t un tuplu arbitrar în q . Atunci conform definiției operației proiecția $r^1 = \{t[R] | t \in q\}$. Dar, pe de altă parte (în virtutea definiției operației joncțiunea), $t[R]$ trebuie să fie și tuplu în r .

Exemplul 2.16. Fie relațiile $r(A B)$ și $s(B C)$. Notăm $q = r \bowtie s$ și $r^1 = \pi_{AB}(q)$. În urma operațiilor, observăm că tuplurile relației r^1 constituie o submulțime proprie a relației r (vezi fig.2.20.).

r	A	B
	a ₁	b ₁
	a ₁	b ₂

s	B	C
	b ₁	c ₁

q	A	B	C
	a ₁	b ₁	c ₁

r ¹	A	B
	a ₁	b ₁

Fig.2.20.

Exemplul 2.17. În acest exemplu $r^1 = r$. Sunt date relațiile $r(R)$ și $s(S)$ cu extensiile ca în fig.2.21, $q = r \bowtie s$, și $r^1 = \pi_{AB}(q)$. Este evident că semnul dintre relațiile r^1 și r este "=", dacă pentru orice tuplu t_r din r există un tuplu t_s în s ce satisfac egalitatea $t_r[R \cap S] = t_s[R \cap S]$. Cu alte cuvinte, dacă $\pi_{R \cap S}(r) = \pi_{R \cap S}(s)$.

r	A	B
	a ₁	b ₁
	a ₂	b ₂

s	B	C
	b ₁	c ₁
	b ₂	c ₂

q	A	B	C
	a ₁	b ₁	c ₁
	a ₂	b ₂	c ₂

r ¹	A	B
	a ₁	b ₁
	a ₂	b ₂

Fig.2.21.

Să considerăm acum legătura dintre proiecție și joncțiune, schimbând ordinea de aplicare a acestor operații.

Fie relația q e definită pe mulțimea de attribute RS . Notăm $r = \pi_R(q)$ și $s = \pi_S(q)$. Fie $q^1 = r \bowtie s$. Care e relația dintre q^1 și q ? Răspuns: $q \subseteq q^1$. Într-adevăr, fie t un tuplu arbitrar al relației q . Atunci tuplurile $t[R]$ și $t[S]$ vor fi în relațiile r și s , corespunzător. Tuplurile $t[R]$ și $t[S]$ sunt joncționabile cu rezultatul t . Deci tuplul t este și în relația q^1 .

În cazul când $q = q^1$, se spune că relația q se descompune fără pierderi pe mulțimile de attribute R și S .

Exemplul 2.18. Relația q din fig.2.22 se descompune fără pierderi pe mulțimile de attribute AB și BC , fiindcă $q = q^1$, unde $r = \pi_{AB}(q)$, $s = \pi_{BC}(q)$, $q^1 = r \bowtie s$.

q	A	B	C
	a ₁	b ₁	c ₁
	a ₁	b ₂	c ₁

r	A	B
	a ₁	b ₁
	a ₁	b ₂

s	B	C
	b ₁	c ₁

q ¹	A	B	C
	a ₁	b ₁	c ₁

b_2	c_1	a_1	b_2	c_1
-------	-------	-------	-------	-------

Fig.2.22.

Să continuăm procesul de descompunere a relației q^1 de acum. Fie $r^1 = \pi_R(q^1)$, și $s^1 = \pi_S(q^1)$. Construim joncțiunea $q^{11} = r^1 \bowtie s^1$.

Care este corelația dintre q^1 și q^{11} ? Răspuns: $q^1 = q^{11}$. Într-adevăr, proiectând $r = \pi_R(q)$, și $s = \pi_S(q)$ asupra mulțimii de atribute $R \cap S$ obținem egalitățile: $\pi_{R \cap S}(r) = \pi_{R \cap S}(\pi_R(q))$ și $\pi_{R \cap S}(s) = \pi_{R \cap S}(\pi_S(q))$. Dar $\pi_{R \cap S}(\pi_R(q)) = \pi_{R \cap S}(q)$ și $\pi_{R \cap S}(\pi_S(q)) = \pi_{R \cap S}(q)$.

Din aceste egalități rezultă că $\pi_{R \cap S}(r) = \pi_{R \cap S}(s)$. Ceea ce înseamnă că $r = r^1$ și $s = s^1$, de unde urmează că $q^1 = q^{11}$.

Corelația dintre joncțiune și alte operații se propun în calitate de exerciții la sfârșitul acestui capitol.

2.2.5. Diviziunea

Definiția 2.15. Fie $r(R)$ și $s(S)$ două relații și $S \subseteq R$. Notăm $Q = R \setminus S$. *Diviziunea* relației r la relația s , notată cu $r \div s$, este o relație definită pe mulțimea de atribute Q :

$$r \div s = \{t \mid \text{pentru } \forall t_s \in s(S) \exists t_r \in r(R) \text{ ce satisface } t_r[Q] = t \text{ și } t_r[S] = t_s\}.$$

Remarcă. Operația diviziunea poate fi concepută drept operație inversă produsului cartezian. Fie $q = r \div s$. Atunci $q \times s$ produce o relație cu schema R și relația q va conține numărul maximal de tupluri ce ar satisface expresia $q \times s \subseteq r$.

Teorema 2.1. Fie două relații $q(Q)$ și $s(S)$. Dacă $r = q \times s$, atunci $q = r \div s$.

Demonstrație. Relația r este definită pe schema QS . E suficient să demonstrăm că $q \subseteq r \div s$ și $r \div s \subseteq q$.

Să arătăm că $q \subseteq r \div s$. Fie $w := r \div s$ și t e un tuplu în q . Fiindcă pentru orice tuplu t_s din s concatenarea tuplurilor t și t_s este în $r = q \times s$, urmează (din definiția operației diviziunea) că tuplul t este în relația w . Deci $q \subseteq w$.

Să arătăm acum că $r \div s \subseteq q$. Fie $w := r \div s$ și t_w e un tuplu arbitrar din w . Atunci, pentru orice tuplu t_s din s (din definiția operației diviziunea), concatenarea tuplurilor t_w și t_s este un tuplu din r . Orice tuplu în $q \times s$ constă din concatenarea unui tuplu din q cu un tuplu din s . Prin urmare, există în q un tuplu t , încât $t = t_w$. Deci $t_w \in q$ și atunci $w \subseteq q$.

Exemplul 2.19. Fie relația $r(A \ B \ C)$ din fig.2.23. În fig.2.24(a)-2.24(e) sunt prezentate relațiile s și q , unde $q = r \div s$. Să se observe că $q \times s \subseteq r$ și că q conține un număr maximal de tupluri ce posedă această proprietate.

r	A	B	C
	a ₁	b ₁	c ₁
	a ₂	b ₁	c ₁
	a ₁	b ₂	c ₁
	a ₁	b ₂	c ₂
	a ₂	b ₁	c ₂
	a ₁	b ₂	c ₃
	a ₁	b ₂	c ₄

a ₁	b ₁	c ₅
----------------	----------------	----------------

Fig.2.23.

s	C		q	A	B
	c ₁			a ₁	b ₁
				a ₂	b ₁
				a ₁	b ₂

Fig.2.24(a).

s	C		q	A	B
	c ₁			a ₁	b ₂
	c ₂			a ₂	b ₁

Fig.2.24(b).

s	C		q	A	B
	c ₁			a ₁	b ₂
	c ₂				
	c ₃				
	c ₄				

Fig.2.24(c).

s	B	C	q	A
	b ₁	c ₁		a ₁
				a ₂

Fig.2.24(d).

s	B	C	q	A
	b ₁	c ₁		a ₁
	b ₂	c ₁		

Fig.2.24(e).

Teorema 2.2. Operația diviziunii poate fi exprimată în termenii produsului cartezian, diferenței și proiecției: $r \div s = \pi_Q(r) \setminus \pi_Q((\pi_Q(r) \times s) \setminus r)$.

Demonstrație. Fie tuplul $t \in \pi_Q(r) \setminus \pi_Q((\pi_Q(r) \times s) \setminus r)$. Atunci $t \in \pi_Q(r)$ și $t \notin \pi_Q((\pi_Q(r) \times s) \setminus r)$. Presupunem că există în s un tuplu t_s încât tuplul format prin concatenarea tuplurilor t și t_s , adică tt_s , nu este în relația r . Atunci tuplul t trebuie să aparțină relației $\pi_Q((\pi_Q(r) \times s) \setminus r)$ ce e contradicție. Prin urmare, tuplul tt_s este în r , adică $t \in r \div s$.

Invers, dacă t este un tuplu în $r \div s$, atunci t este în $\pi_Q(r)$. Tuplul t nu poate aparține relației $\pi_Q((\pi_Q(r) \times s) \setminus r)$, fiindcă atunci ar trebui să existe în s un tuplu t_s , încât concatenția tt_s să fie în $\pi_Q(r) \times s$ și să nu fie relația r .

Deci $t \in \pi_Q(r) \setminus \pi_Q((\pi_Q(r) \times s) \setminus r)$.

Să subliniem că operația diviziunii nu este nici comutativă, nici asociativă.

Din definiția diviziunii și remarcă urmează că, dacă relația $r[R]$ este o submulțime proprie de tupluri a relației $s(S)$, atunci $r \div s$ este o relație vidă.

Dacă $S = \emptyset$ atunci $r \div s$ este indefinită, fiindcă indefinită este $s(\emptyset)$.

2.2.6. Semijoncțiunea

Semijoncțiunea e o operație binară. Ea constă în construirea unei relații din cele două și e formată numai din tuplurile unei singure relații ce participă la joncțiune.

Definiția 2.16. Fie două relații $r(R)$ și $s(S)$. *Semijoncțiunea* relației r și s , notată cu $r \bowtie s$, este o mulțime de tupluri determinată de expresia $r \bowtie s = \pi_R(r \bowtie s)$.

Exemplul 2.20. Fie relațiile $r(A \ B)$, $s(B \ C)$ și $q(A \ B)$ din fig.2.25. Relația $q = r \bowtie s$.

r	A	B
	a ₁	b ₁
	a ₁	b ₂
	a ₂	b ₁

s	B	C
	b ₁	c ₁

q	A	B
	a ₁	b ₁
	a ₂	b ₁

Fig.2.25.

2.3. Expresii algebrice

Uniunea, diferența, produsul cartezian, proiecția, selecția și atribuirea sunt operațiile de bază ale algebrei relaționale. Celelalte operații, precum s-a văzut, se exprimă prin aceste șase. Dar un limbaj algebric de interpelări ce ar folosi numai operațiile de bază e destul de neeficient. Prin urmare, sunt utile și operațiile adiționale: intersecția, θ -joncțiunea, joncțiunea, semijoncțiunea și diviziunea.

Definiția 2.17. Fie $U = A_1 \dots A_n$ mulțimea universală de attribute, $\text{dom}(U) = \{\text{dom}(A_i) \mid A_i \in U \ \& \ 1 \leq i \leq n\}$ și, $f_{\text{dom}}: U \rightarrow \text{dom}(U)$, o funcție ce pune în corespondență fiecărui atribut din U un singur domeniu (unele attribute pot avea aceleași domenii de valori). Fie $Db = \{R_1, \dots, R_m\}$ schema bazei de date asupra U , unde $U = R_1 \dots R_m$, și $db = \{r_1, \dots, r_m\}$ o bază de date cu schema Db . Fie $\Theta = \{=, \neq, <, \leq, >, \geq\}$ mulțimea de operații aritmetice de comparare asupra domeniilor din $\text{dom}(U)$ și fie O mulțimea de operații ce include cel puțin cele șase operații relaționale de bază. *Algebra relațională* asupra U , $\text{dom}(U)$, f_{dom} , Db , db , Θ și O este tuplul $A = (U, \text{dom}(U), f_{\text{dom}}, Db, db, \Theta, O)$. *Expresie algebrică* asupra A este orice expresie bine formată (în corespundere cu restricțiile impuse operațiilor relaționale) de relații din db și relații constante cu scheme din Db legate cu operații din O .

În expresiile algebrice, se admit parantezele și se presupune că operațiile binare nu au prioritate una față de alta, cu excepția priorității operației \cap față de \cup . Într-o

consecutivitate de relații legate cu aceeași operație parantezele pot fi omise, dacă, bineînțeles, operația este asociativă. Numele de relații r_1, \dots, r_m servesc drept variabile, unde r_i parcurge mulțimea de relații cu schema R_i , $1 \leq i \leq m$. Să nu luăm în seamă ambiguitatea că r_i denotă atât numele unei relații, cât și o instanță curentă a unei relații cu schema R_i .

Exemplul 2.21. Fie relațiile *articole*(ART_ID ART_NUME ORAȘ BUCĂȚI PREȚ), *clienți*(CL_ID, CL_NUME, CL_ORAȘ, REDUCERE), *agenți*(AG_ID, AG_NUME, AG_ORAȘ, COMISION), *comenzi*(LUNĂ, CL_ID, AG_ID, ART_ID, BUCĂȚI, SUMĂ). Expresiile algebrei relaționale ce corespund unor interpelări puse la baza de date $db = (articole, clienți, agenți, comenzi)$ sunt redate mai jos.

- (a) Să se găsească numele clienților ce au comandat cel puțin un articol la prețul 0.50 lei.

$$\pi_{CL_NUME} ((\pi_{ART_ID} (\sigma_{PREȚ=0.50} (articole))) \mid \times \mid comenzi) \mid \times \mid clienți)$$

- (b) Să se găsească numele clienților ce n-au dat nici o comandă prin agentul a03.

$$\pi_{CL_NUME} (clienți \mid \times \mid (\pi_{CL_ID} (clienți) \setminus \pi_{CL_ID} (\sigma_{AG_ID="a03"} (comenzi))))$$

- (c) Să se găsească clienții ce-și plasează comenzile numai prin agentul a03.

$$\pi_{CL_ID} (clienți) \setminus \pi_{CL_ID} (\sigma_{AG_ID \neq "a03"} (comenzi))$$

- (d) Să se găsească articolele ce nu au fost comandate de vreun client din Chișinău printr-un agent cu sediul la Bălți.

$$\pi_{ART_ID} (articole) \setminus \pi_{ART_ID} ((\pi_{CL_ID} (\sigma_{CL_ORAȘ="Chișinău"} (clienți))) \mid \times \mid comenzi) \mid \times \mid \sigma_{AG_ORAȘ="Bălți"} (agenți))$$

- (e) Să se găsească numele clienților ce au comandat toate tipurile de articole la prețul 0.50 lei.

$$\pi_{CL_NUME} (clienți \mid \times \mid (\pi_{CL_ID \mid ART_ID} (comenzi) \div \pi_{ART_ID} (\sigma_{PREȚ=0.50} (articole))))$$

- (f) Să se găsească clienții ce au comandat toate tipurile de articole comandate de oricine.

$$\pi_{CL_ID \mid ART_ID} (comenzi) \div \pi_{ART_ID} (comenzi)$$

- (g) Să se găsească agenții ce au primit comenzi de livrare ale articolelor comandate și de clientul c004.

$$\pi_{AG_ID \mid ART_ID} (comenzi) \div \pi_{ART_ID} (\sigma_{CL_ID="c004"} (comenzi))$$

- (h) Să se găsească clienții ce au comandat articolele art01 și art07.

$$\pi_{CL_ID} (\sigma_{ART_ID="art01"} (comenzi)) \cap \pi_{CL_ID} (\sigma_{ART_ID="art07"} (comenzi))$$

- (i) Să se găsească clienții ce au plasat comenzi prin agenții care au livrat articolul art03.

$$\pi_{CL_ID} (comenzi \mid \times \mid \pi_{AG_ID} (\sigma_{ART_ID="art03"} (comenzi)))$$

- (j) Să se găsească clienții ce au comandat articole cu o reducere la preț ca a clienților din Chișinău sau Iași.

$$\pi_{CL_ID} (clienți \mid \times \mid \pi_{REDUCERE} (\sigma_{CL_ORAȘ="Chișinău" \vee CL_ORAȘ="Iași"} (clienți)))$$

- (k) Să se găsească articolele, ce au fost comandate prin agenții, ce au primit comenzi de la clienții, ce au comandat cel puțin un articol printr-un agent, ce a servit clientul c001.

$$\pi_{ART_ID} (comenzi \mid \bowtie \mid (\pi_{AG_ID} (comenzi \mid \bowtie \mid (\pi_{CL_ID} (comenzi \mid \bowtie \mid (\pi_{AG_ID} (\sigma_{CL_ID} = "c001" (comenzi))))))))))$$

Evaluarea unei expresii algebrice presupune efectuarea operațiilor asupra relațiilor din expresie în ordinea indicată de operatorii din expresie sau de paranteze. Rezultatul evaluării unei expresii este o relație. Deci orice expresie algebrică definește o funcție, ce aplică mulțimea de relații din expresie într-o singură relație. Schema acestei relații depinde de schemele relațiilor ce compun expresia algebrică. Notăm cu $sch(E)$ schema expresiei algebrice E .

Definiția 2.18. Schema expresiei algebrice $sch(E)$ se definește recursiv:

- (1) dacă E este o relație constantă, atunci $sch(E)$ este schema relației;
- (2) dacă E este relația r_i cu schema R_i , atunci $sch(E) = R_i$;
- (3) dacă E este una din expresiile $E_1 \cup E_2$, $E_1 \cap E_2$, $E_1 \setminus E_2$, $\sim E_1$ și $\sigma_F(E_1)$, unde F este o formulă aplicabilă lui E_1 , atunci $sch(E) = sch(E_1)$;
- (4) dacă E este $\pi_X(E_1)$, atunci $sch(E) = X$;
- (5) dacă E este $E_1 \div E_2$, atunci $sch(E) = sch(E_1) \setminus sch(E_2)$;
- (6) dacă E este una din expresiile $E_1 \times E_2$, $E_1 \mid \bowtie \mid_F E_2$ și $E_1 \mid \bowtie \mid E_2$, atunci $sch(E) = sch(E_1) \cup sch(E_2)$;
- (7) dacă E este $r_i = r_j$, unde relațiile r_i și r_j au schemele R_i și R_j , respectiv, atunci $sch(E) = R_j$.

2.4. Exerciții

- 2.1. Fie relațiile $r(A \ B \ C)$ și $s(B \ C \ D)$, $a \in \text{dom}(A)$ și $b \in \text{dom}(B)$. Care din expresiile algebrei relaționale de mai jos sunt corecte?

- (a) $r \cup s$;
- (b) $\pi_B(r) \setminus \pi_B(s)$;
- (c) $\sigma_{B=b}(r)$;
- (d) $\sigma_{A=a, B=b}(s)$;
- (e) $r \mid \bowtie \mid s$;
- (f) $\pi_A(r) \mid \bowtie \mid \pi_D(s)$.

r	A	B	C
	a	b	c
	a	b ₁	c ₁
	a	b ₂	c ₁
	a ₁	b ₁	c

s	B	C	D
	b ₁	c ₁	d
	b ₂	c ₁	d ₁
	b ₂	c	d

Fig.2.26.

- 2.2. Fie r și s sunt relațiile din fig.2.26. Să se calculeze expresiile:

- (a) $\sim r$;

- (b) $\sim s$;
- (c) $\sigma_{A=a}(r)$;
- (d) toate expresiile formulate corect din exercițiul 2.1.
- 2.3. Fie relațiile $r(R)$ și $s(S)$, și fie $K \subseteq R$ este cheia relațiilor r și s . Care din relațiile de mai jos au în calitate de cheie mulțimea K ?
- (a) $r \cup s$;
- (b) $r \cap s$;
- (c) $r \setminus s$;
- (d) $\sim r$;
- (e) $\pi_K(r)$;
- (f) $r|X|s$.
- 2.4. Fie relația $r(R)$, $A \in R$, $a, b \in \text{dom}(A)$. Care din expresiile de mai jos sunt corecte?
- (a) $\sigma_{A=a, A=b}(r) = \emptyset$;
- (b) $\sigma_{A=a, A=b}(r) = \sigma_{A=a}(r)$.
- 2.5. Fie relațiile $r(R)$ și $s(S)$, $A \in R$ și $a \in \text{dom}(A)$. Confirmați sau infirmați următoarele egalități:
- (a) $\sigma_{A=a}(\sim r) = \sigma_{A=a}(r)$;
- (b) $\sigma_{A=a}(r \cap s) = \sigma_{A=a}(r) \cap s$.
- 2.6. Fie relația $r(\underline{A} \ B \ C)$. Ce se poate de spus despre numărul de tupluri ale relației $\sigma_{A=a}(r)$?
- 2.7. Fie relația $r(R)$, $X \subseteq R$, $A \in R$ și $A \notin X$. Să se găsească un exemplu ce ar infirma egalitatea $\pi_X(\sigma_{A=a}(r)) = \sigma_{A=a}(\pi_X(r))$.
- 2.8. Fie relațiile $r(R)$, $s(R)$ și $X \subseteq R$. Confirmați sau infirmați egalitățile:
- (a) $\pi_X(r \cap s) = \pi_X(r) \cap \pi_X(s)$;
- (b) $\pi_X(r \cup s) = \pi_X(r) \cup \pi_X(s)$;
- (c) $\pi_X(r \setminus s) = \pi_X(r) \setminus \pi_X(s)$;
- (d) $\pi_X(\sim r) = \sim \pi_X(r)$.
- 2.9. Fie relația $r(R)$, $A \in R$. Notăm $S = R \setminus A$. Care sunt corelațiile dintre dimensiunile relațiilor r , $\sigma_{A=a}(r)$, $\pi_A(r)$, $\pi_S(r)$, $\sigma_{A=a}(\pi_A(r))$?
- 2.10. Fie relațiile r și s . Să se arate că:
- (a) $s|X|s=s$;
- (b) $r|X|s = r|X|(r|X|s)$.
- 2.11. Fie relațiile $r(R)$, $s(S)$ și $A \in R$. Să se arate că $\sigma_{A=a}(r|X|s) = \sigma_{A=a}(r)|X|s$.
- 2.12. Fie relațiile $r(R)$, $r^1(R)$ și $s(S)$. Confirmați sau infirmați egalitățile:
- (a) $(r \cap r^1)|X|s = (r|X|s) \cap (r^1|X|s)$;
- (b) $(r \setminus r^1)|X|s = (r|X|s) \setminus (r^1|X|s)$;
- (c) $\sim r|X|\sim s = r|X|s$.
- 2.13. Fie relațiile $r(R)$, $s(S)$ și $R \cap S = \emptyset$. Să se arate că $(r|X|s) \div s = r$.

- 2.14. Fie relațiile $r(R)$, $s(S)$, $s^1(S)$ și $S \subseteq R$. Să se arate că $s \subseteq s^1$ implică $r \div s^1 \subseteq r \div s$. Să se arate că afirmația inversă nu e corectă.
- 2.15. Fie relațiile $r(ABC)$ și $s(BCD)$. Care este schema expresiei algebrice
- $$E = \pi_A(\sigma_{B=b}(\tilde{r})) \mid \bowtie \mid \pi_B(\pi_{BC}(r) \setminus \pi_{BC}(s)).$$

DEPENDENȚE FUNCȚIONALE

Proiectarea logică a bazei de date urmărește printre altele diminuarea redundanței și asigurarea securității datelor. Acest scop se poate atinge, dacă se cunosc a priori constrângerile ce pot fi aplicate asupra datelor. Dependentele sunt constrângeri impuse datelor în baza de date. Ba mai mult, mulțimea de dependente este partea esențială a schemei unei relații, deci și a schemei bazei de date. Dependentele funcționale au fost primele constrângeri logice considerate în modelul relațional. Ele formează cel mai simplu și cel mai larg răspândit tip de dependente.

Prezentul capitol e consacrat regulilor de inferență, închiderilor și diverselor forme de acoperiri ale dependențelor funcționale.

3.1. Noțiuni generale

Să considerăm relația *orar* din fig. 3.1.

<i>orar</i>	PROFESOR	DISCIPLINĂ	ZI	ORĂ	GRUPĂ	SALĂ
	Petrescu	Baze de date	Luni	8:00	C941	402
	Petrescu	Baze de date	Mierc.	14:30	C941	216
	Petrescu	Baze de date	Mierc.	16:00	C941	216
	Vasilache	Progr.logică	Luni	9:30	C941	404

Fig.3.1. Relația *orar*

Această relație arată care profesor predă disciplina dată, cărei grupe, în ce zi a săptămânii, la ce oră și în ce sală. Atributele ce formează schema acestei relații nu pot primi orice valori. Atributele se află într-o interdependență. Aici, în particular, se suprapun asupra atributelor următoarele constrângeri:

- (1) o disciplină este predată unei grupe de studiu de un singur profesor;
- (2) profesorul, în ziua dată, la ora dată se găsește într-o singură sală;
- (3) în ziua dată, la ora dată, în sala dată se predă o singură disciplină.

Aceste constrângeri ce reflectă o interdependență între atribute sunt exemple de dependente funcționale. Dependența funcțională este o generalizare a noțiunii de cheie.

Constrângerile de mai sus pot fi formulate:

- (1) DISCIPLINĂ GRUPĂ determină funcțional PROFESOR sau, ce e echivalent PROFESOR e determinat funcțional de DISCIPLINĂ GRUPĂ;
- (2) PROFESOR ZI ORĂ determină funcțional SALĂ;
- (3) ZI ORĂ SALĂ determină funcțional DISCIPLINĂ;

și notate respectiv:

- (1) DISCIPLINĂ GRUPĂ \rightarrow PROFESOR;
- (2) PROFESOR ZI ORA \rightarrow SALA;
- (3) ZI ORA SALĂ \rightarrow DISCIPLINA.

Unica posibilitate de a determina dependențele funcționale constă într-o analiză cu luare-aminte a semanticii atributelor. În acest sens dependențele sunt de fapt aserțiuni despre lumea reală. Ele nu pot fi demonstrate. Dar ele pot și trebuie să fie susținute de SGBD-uri. Majoritatea sistemelor susțin numai dependențele funcționale determinate de cheile relației. Dar sunt și sisteme ce susțin dependențe funcționale arbitrare.

Trebuie menționat că declararea dependențelor funcționale într-o bază de date este o decizie pe care o ia numai proiectantul bazei de date. Odată declarate SGBD-ul va susține aceste constrângeri. În afară de aceasta, după cum se va vedea în celelalte secțiuni, grație dependențelor, există o structură mai eficientă de păstrare a datelor. Dependențele funcționale vor servi la proiectarea schemelor bazelor de date cu anumite proprietăți dezirabile.

Definiția 3.1. Fie relația r cu schema R și $X, Y \subseteq R$. Vom spune că dependența funcțională $X \rightarrow Y$ este validă în relația r (sau relația r satisface dependența funcțională $X \rightarrow Y$), dacă, pentru orice două tupluri din r , fie t_1 și t_2 , din condiția că tuplurile au X -valori identice, urmează că au și Y -valori identice, adică $t_1[X]=t_2[X] \Rightarrow t_1[Y]=t_2[Y]$.

Dacă $X \rightarrow Y$ e validă în $r(R)$, vom spune că X *determină funcțional* Y sau, că Y e *determinat funcțional* de X . În această definiție (și mai departe) simbolul " \Rightarrow " notează "implică".

Deci dependența funcțională $X \rightarrow Y$ reprezintă o restricție de integritate aplicată tuplurilor relației $r(R)$, în sensul că oricare două tupluri din r care prezintă o aceeași valoare pentru X trebuie să prezinte o aceeași valoare pentru Y .

Definiția 3.1 poate fi interpretată și în felul următor: relația $r(R)$ satisface dependența funcțională $X \rightarrow Y$, dacă relația $\pi_Y(\sigma_{X=x}(r))$ conține nu mai mult de un tuplu pentru orice valoare x a atributului X .

Partea stângă a dependenței poartă numele de *determinant*, iar partea dreaptă a dependenței poartă numele de *determinat*. Astfel în cadrul dependenței $X \rightarrow Y$, X este determinantul, iar Y determinatul.

Exemplul 3.1. Considerăm relațiile din fig.3.2. În ele sunt valide următoarele dependențe funcționale. În relația r_1 : $A \rightarrow B$; în relația r_2 : $A \rightarrow B$, $B \rightarrow A$; în relația r_3 : $A \rightarrow B$.

r_1	A	B
	a_1	b_1
	a_2	b_2
	a_3	b_1
	a_4	b_1
	a_5	b_2
	a_6	b_2

r_2	A	B
	a_1	b_1
	a_2	b_4
	a_1	b_1
	a_3	b_2
	a_2	b_4
	a_4	b_3

r_3	A	B
	a_1	b_1
	a_2	b_4
	a_1	b_1
	a_3	b_2
	a_2	b_4
	a_4	b_4

Fig.3.2. Relațiile r_1 , r_2 și r_3 .

Pentru a verifica dacă o dependență e validă într-o relație dată, se utilizează următorul algoritm.

Algoritmul SATISF ($r, X \rightarrow Y$)

Intrare: Relația $r(R)$, dependența funcțională $X \rightarrow Y$, unde $X, Y \subseteq R$.

Ieșire: *Adevăr*, dacă relația r satisface dependența funcțională $X \rightarrow Y$; *fals* – în caz contrar.

- (1) Se sortează tuplurile relației r în așa fel, ca tuplurile cu X -valori egale să fie grupate împreună.
- (2) Se verifică dacă mulțimea de tupluri cu X -valori egale are și Y -valori egale, atunci la ieșire obținem adevăr; în caz contrar - fals.

Menționăm că nu ne interesează dependențele funcționale întâmplătoare, dar numai acele ce decurg din semantica atributelor. De exemplu, în relația *orar* e validă și dependența funcțională PROFESOR \rightarrow DISCIPLINĂ. Dar ea nu reprezintă o constrângere ce reflectă lumea reală, fiindcă în realitate un profesor poate și, de regulă, predă mai multe discipline.

Numai dependențele neîntâmplătoare, asigură integritatea semantică a bazei de date. De exemplu, dacă un utilizator dorește să insereze în relația *orar* un tuplu:

Add(*orar*; <Vasilache "Structuri de date" luni 8:00 c942 402>),

sistemul de gestiune va stopa efectuarea acestei operații, fiindcă va fi violată dependența funcțională ZI ORĂ SALĂ \rightarrow DISCIPLINĂ. Dacă SGBD-ul nu susține dependențele funcționale, atunci se poate întâmpla că într-o sală în același timp se vor preda două discipline diferite.

3.2. Reguli de inferență

Într-o relație $r(R)$ în orice moment sunt valide o mulțime de dependențe funcționale, să zicem F . Adică F este o mulțime de dependențe satisfăcută de relația $r(R)$. Aici apare aceeași problemă ca și în cazul cheilor. O extensie a relației satisface mulțimea F de dependențe funcționale, în timp ce altă extensie nu satisface. Pe noi ne interesează numai dependențele ce sunt satisfăcute de orice extensie a relației $r(R)$. Dar pentru aceasta sunt necesare cunoștințe asupra semanticii relației $r(R)$. Vom considera că mulțimea F de dependențe funcționale este definită pe mulțimea R de attribute ce formează schema relației r și orice extensie a relației r satisface mulțimea F .

Evident că mulțimea de dependențe valide în $r(R)$ este finită, fiindcă finită este schema R . Prin urmare, s-ar putea verifica dacă r satisface dependențele din F , aplicând algoritmul SATISF. Însă astfel de soluție este foarte laborioasă. Există o altă cale. Dacă sunt cunoscute niște dependențe, din ele pot fi deduse altele.

Definiția 3.2. Fie relația r cu schema R , F o mulțime de dependențe funcționale și f o dependență asupra R . Notăm cu $SAT(F)$ mulțimea tuturor relațiilor asupra R ce satisface orice dependență din F . Vom spune că F *logic implică* f , sau f este *consecință logică* a F , scriem $F \models f$, dacă orice relație $r(R)$ ce satisface dependențele din F satisface și f , adică

$$r(R) \in SAT(F) \Rightarrow r(R) \in SAT(F \cup \{f\}).$$

O regulă de inferență stabilește că, dacă o relație satisface anumite dependențe, ea trebuie să satisfacă și alte dependențe. Vom considera șase reguli de inferență a dependențelor funcționale.

Fie relația r definită pe mulțimea de attribute R și fie $W, X, Y, Z \subseteq R$.

DF1. Regula reflexivității. Dacă $Y \subseteq X$, atunci $X \rightarrow Y$.

Demonstrarea acestei afirmații este evidentă. Nu putem avea într-o relație două tupluri cu X-valori egale și să nu fie egale componentele lor pentru o submulțime a lui X.

Definiția 3.3. O dependență $X \rightarrow Y$ este *trivială* dacă $Y \subseteq X$.

Regula DF1 generează numai dependențe triviale și ea nu depinde de F. Întrucât $\emptyset \subseteq X \subseteq R$, atunci $X \rightarrow \emptyset$ și $R \rightarrow X$ sunt dependențe triviale. Deoarece $X \subseteq X$, $X \rightarrow X$ e dependență trivială. Dintre aceste dependențe prima, $X \rightarrow \emptyset$, nu are nici o aplicare practică.

DF2. Regula incrementării. Dacă $X \rightarrow Y$ și $Z \subseteq W$, atunci $XW \rightarrow YZ$.

Demonstrație. Fie r satisface dependența funcțională $X \rightarrow Y$, însă în ea există două tupluri t_1 și t_2 cu XW-valori egale, dar componentele YZ nu coincid. Conform regulii DF1 tuplurile t_1 și t_2 au Z-valori egale (fiindcă $Z \subseteq W$). Atunci tuplurile trebuie să nu coincidă măcar pe un atribut din Y. Dar aceasta înseamnă că tuplurile t_1 și t_2 au X-valori egale și nu au Y-valori egale, ce contrazice ipoteza că relația r satisface dependența funcțională $X \rightarrow Y$.

Să observăm că regula DF2 are mai multe cazuri speciale. Dacă $Z = \emptyset$ și $X \rightarrow Y$, atunci $XW \rightarrow Y$ pentru orice submulțime W din R. Dacă $W = Z$ și $X \rightarrow Y$, atunci $XW \rightarrow YW$. Dacă $X = W$, $Z = X$ și $X \rightarrow Y$, atunci $XX \rightarrow XY$, adică $X \rightarrow XY$.

r	A	B	C	D
	a ₁	b ₁	c ₁	d ₁
	a ₂	b ₂	c ₁	d ₁
	a ₁	b ₁	c ₁	d ₂
	a ₃	b ₃	c ₂	d ₃

Fig.3.3.

Exemplul 3.2. Considerăm relația din figura 3.3. Relația $r(ABCD)$ satisface dependența funcțională $A \rightarrow B$. Conform regulii DF2 în această relație sunt valide și dependențele $AB \rightarrow B$, $AC \rightarrow B$, $AD \rightarrow B$, $ABC \rightarrow B$, $ABD \rightarrow B$, $ACD \rightarrow B$, $ABCD \rightarrow B$, $AC \rightarrow BC$, $AD \rightarrow BD$, $ABC \rightarrow BC$, $ABD \rightarrow BD$, $ACD \rightarrow BC$, $ACD \rightarrow BD$, $ACD \rightarrow BCD$, $ABCD \rightarrow BC$, $ABCD \rightarrow BD$, $ABCD \rightarrow BCD$.

DF3. Regula aditivității. Dacă $X \rightarrow Y$ și $X \rightarrow Z$, atunci $X \rightarrow YZ$.

Demonstrație. Presupunem contrariul: în relația $r(R)$ există două tupluri t_1 și t_2 , pentru care $t_1[X] = t_2[X]$, dar $t_1[YZ] \neq t_2[YZ]$. Atunci, sau $t_1[Y] \neq t_2[Y]$, sau $t_1[Z] \neq t_2[Z]$, sau ambele concomitent. Dar aceasta contrazice presupunerea, că relația r satisface dependențele $X \rightarrow Y$ și $X \rightarrow Z$.

Exemplul 3.3. Relația $r(ABCD)$ reprezentată în fig.3.3 satisface dependențele funcționale $A \rightarrow B$ și $A \rightarrow C$. Conform regulii DF3, relația r trebuie să satisfacă și dependența $A \rightarrow BC$.

DF4. Regula proiectivității. Dacă $X \rightarrow YZ$, atunci $X \rightarrow Y$.

Demonstrație. Afirmatia este adevărată, fiindcă, dacă r satisface $X \rightarrow YZ$, atunci pentru orice două tupluri t_1 și t_2 din $t_1[X]=t_2[X]$ urmează $t_1[YZ]=t_2[YZ]$ și tuplurile vor coincide și pe orice submulțime ale mulțimii YZ . Deci $t_1[Y]=t_2[Y]$.

Exemplul 3.4. Relația din fig.3.3 satisface dependența funcțională $A \rightarrow BC$. Conform regulii DF4, ea satisface și dependențele $A \rightarrow B$ și $A \rightarrow C$.

DF5. Regula tranzitivității. Dacă $X \rightarrow Y$ și $Y \rightarrow Z$, atunci $X \rightarrow Z$.

Demonstrație. Pentru a demonstra regula DF5, vom presupune contrariul. Fie relația r satisface dependențele $X \rightarrow Y$ și $Y \rightarrow Z$, dar nu satisface dependența $X \rightarrow Z$. Atunci relația r are cel puțin două tupluri t_1 și t_2 , pentru care $t_1[X]=t_2[X]$, iar $t_1[Z] \neq t_2[Z]$. Dacă $t_1[Y]=t_2[Y]$, atunci inegalitatea $t_1[Z] \neq t_2[Z]$ contrazice presupunerea că în r e validă dependența funcțională $Y \rightarrow Z$. Dacă $t_1[Y] \neq t_2[Y]$, atunci egalitatea $t_1[X]=t_2[X]$ contrazice presupunerea că $X \rightarrow Y$ e validă în r .

r	A	B	C	D
	a ₁	b	c ₂	d
		1		1
	a ₂	b	c ₁	d
		2		2
	a ₃	b	c ₂	d
		1		1
	a ₄	b	c ₂	d
		1		3

Fig.3.4.

Exemplul 3.5. Relația $r(ABCD)$, reprezentată în fig.3.4, satisface dependențele funcționale $A \rightarrow B$ și $B \rightarrow C$. Conform regulii tranzitivității, relația r satisface și dependența funcțională $A \rightarrow C$.

DF6. Regula pseudotranzitivității. Dacă $X \rightarrow Y$ și $YW \rightarrow Z$, atunci $XW \rightarrow Z$.

Demonstrație. Presupunem contrariul: relația $r(R)$ satisface dependențele funcționale $X \rightarrow Y$ și $YW \rightarrow Z$, dar nu satisface dependența funcțională $XW \rightarrow Z$. Adică există două tupluri $t_1, t_2 \in r$ pentru care $t_1[XW]=t_2[XW]$ și $t_1[Z] \neq t_2[Z]$. Din egalitatea $t_1[XW]=t_2[XW]$ urmează $t_1[X]=t_2[X]$ și $t_1[W]=t_2[W]$. Pot avea loc două cazuri: sau $t_1[YW]=t_2[YW]$, sau $t_1[YW] \neq t_2[YW]$.

- (1) Fie $t_1[YW]=t_2[YW]$. Atunci din condiția că $t_1[Z] \neq t_2[Z]$ reiese că dependența funcțională $YW \rightarrow Z$ nu e validă în r .
- (2) Fie $t_1[YW] \neq t_2[YW]$. Întrucât $t_1[W]=t_2[W]$ rezultă că $t_1[Y] \neq t_2[Y]$. Din ultima inegalitate și din condiția $t_1[X]=t_2[X]$ urmează că relația r nu satisface dependența funcțională $X \rightarrow Y$.

Și într-un caz și în altul am ajuns la contradicție. Deci din $X \rightarrow Y$ și $YW \rightarrow Z$ urmează $XW \rightarrow Z$.

Așadar, dacă $W, X, Y, Z \subseteq R$, atunci în orice relație r cu schema R sunt valide regulile de inferență din fig.3.5.

Simbol	Denumire	Regulă
DF1	Reflexivitatea	$Y \subseteq X \Rightarrow X \rightarrow Y$
DF2	Incrementarea	$X \rightarrow Y, Z \subseteq W \Rightarrow XW \rightarrow YZ$
DF3	Aditivitatea	$X \rightarrow Y, X \rightarrow Z \Rightarrow X \rightarrow YZ$
DF4	Proiectivitatea	$X \rightarrow YZ \Rightarrow X \rightarrow Y$
DF5	Tranzitivitatea	$X \rightarrow Y, Y \rightarrow Z \Rightarrow X \rightarrow Z$
DF6	Pseudotranzitivitatea	$X \rightarrow Y, YW \rightarrow Z \Rightarrow XW \rightarrow Z$

Fig.3.5. Reguli de inferență a dependențelor funcționale

Să observăm că proiectivitatea (DF4) este, într-un sens, regula inversă regulii aditivității (DF3). Regula DF3 se utilizează pentru a uni două dependențe cu determinante egale în una, în timp ce regula DF4 - pentru descompunerea unei dependente.

3.3. Axiomele Armstrong

Definiția 3.4. Fie F o mulțime de dependențe asupra R și fie f o dependență funcțională asupra R . *Derivație* a dependenței f din F , notată cu $F|-f$, este o consecutivitate finită de dependențe funcționale f_1, f_2, \dots, f_k unde:

- (1) orice dependență f_i poate fi dedusă din (o submulțime a mulțimii) $F \cup \{f_1, f_2, \dots, f_{i-1}\}$, aplicând regulile de inferență DF1-DF6;
- (2) f este ultimul element, f_k , în consecutivitate.

Remarcă. Condiția (1) mai poate fi formulată în felul următor. Orice dependență f este element al mulțimii F sau se deduce din consecutivitatea $\{f_1, f_2, \dots, f_{i-1}\}$, aplicând regulile de inferență DF1-DF6.

Dacă $F|-f$, vom spune că F *derivă* f sau că f e *derivabilă* din F . Dacă G este mulțime de dependențe funcționale, atunci prin $F|-G$ se subînțelege că orice dependență funcțională din G e derivabilă din F .

E clar că, dacă în condiția (1) a definiției 3.4 mulțimea de dependențe funcționale F e vidă, adică $\emptyset|-f$, atunci f e dependență trivială, fiindcă singura regula de inferență corespunzătoare poate fi doar DF1.

Cu ajutorul regulilor de inferență putem deduce noi dependențe funcționale din cele date.

Exemplul 3.4. Fie r o relație cu schema R și $X, Y, Z \subseteq R$. Presupunem că în $r(R)$ sunt valide dependențele $XY \rightarrow Z$ și $X \rightarrow Y$. Atunci, conform regulii DF6, relația r satisface și dependența $XX \rightarrow Z$ care se reduce la $X \rightarrow Z$.

Pentru a combate o afirmație despre validitatea unei dependențe funcționale, e suficient de a aduce un exemplu de relație ce nu satisface afirmația dată.

r	A	B	C	D
	a ₁	b ₁	c ₁	d ₁
	a ₁	b ₂	c ₂	d ₁

Fig.3.6.

Exemplul 3.5. Să combatem afirmația că dependența $XY \rightarrow ZW$ implică dependența $X \rightarrow Z$. Relația $r(ABCD)$ din fig.3.6 satisface dependența funcțională $AB \rightarrow CD$, dar nu satisface dependența $A \rightarrow C$.

Unele reguli de inferență pot fi deduse din altele. Armstrong a arătat că regulile DF1, DF2 și DF5 formează o mulțime de reguli independente, iar regulile DF3, DF4 și DF6 pot fi deduse din DF1, DF2 și DF5. Mulțimea $\{DF1, DF2, DF5\}$ de reguli de inferență este cunoscută sub denumirea de axiome Armstrong.

Teorema 3.1. Regulile DF3, DF4 și DF6 se deduc din regulile DF1, DF2, DF5.

Demonstrație. Să arătăm că regula DF3 se deduce din regulile DF1, DF2, DF5, adică $\{X \rightarrow Y, X \rightarrow Z\} \mid -X \rightarrow YZ$, aplicând doar DF1, DF2, DF5. Într-adevăr, această aserțiune are loc, fiindcă putem construi următoarea consecutivitate de derivare.

- $f_1 := X \rightarrow Y$ (dependență dată),
- $f_2 := X \rightarrow XY$ ($f_1 \mid -f_2$ aplicând DF2),
- $f_3 := X \rightarrow Z$ (dependență dată),
- $f_4 := XY \rightarrow YZ$ ($f_3 \mid -f_4$, aplicând DF2),
- $f_5 := X \rightarrow YZ$ ($\{f_2, f_4\} \mid -f_5$, aplicând DF5).

Fiindcă $X \rightarrow YZ$ este ultimul, f_5 , element în consecutivitate, DF3 se deduce din DF2 și DF5.

Să ne convingem că regula DF4 se deduce din DF1, DF2, DF5, adică $X \rightarrow YZ \mid -X \rightarrow Y$, aplicând regulile DF1, DF2, DF5. Aceasta se confirmă de următoarea consecutivitate de derivare.

- $f_1 := X \rightarrow YZ$ (dependență dată),
- $f_2 := YZ \rightarrow Y$ (dependență trivială, aplicând DF1),
- $f_3 := X \rightarrow Y$ ($\{f_1, f_2\} \mid -f_3$, aplicând DF5).

Deci, regula DF4 se deduce din DF1 și DF5.

În sfârșit, să arătăm că $\{X \rightarrow Y, YW \rightarrow Z\} \mid -XW \rightarrow Z$, aplicând regulile DF1, DF2 și DF5. Putem construi următoarea consecutivitate de dependențe funcționale.

- $f_1 := X \rightarrow Y$ (dependență dată),
- $f_2 := XW \rightarrow YW$ ($f_1 \mid -f_2$ aplicând DF2),
- $f_3 := YW \rightarrow Z$ (dependență dată),
- $f_4 := XW \rightarrow Z$ ($\{f_2, f_3\} \mid -f_4$, aplicând DF5).

Deci, regula DF6 se deduce din regulile DF1, DF2, DF5.

Definiția 3.5. Fie F o mulțime de dependențe funcționale asupra schemei R și $X, Y \subseteq R$. Închiderea mulțimii F , notată cu F^+ , se definește recursiv:

- (1) $F \subseteq F^+$;
- (2) Dacă $F^1 \subseteq F$ și $F^1 \mid -X \rightarrow Y$, atunci $X \rightarrow Y \in F^+$;
- (3) Nimic altceva nu e în F^+ .

Deci, $F^+ = F \cup \{X \rightarrow Y \mid F^1 \mid -X \rightarrow Y \text{ pentru } F^1 \subseteq F \text{ și } X, Y \subseteq R\}$. Cu alte cuvinte, închiderea unei mulțimi de dependențe funcționale, F^+ , reprezintă mulțimea tuturor

dependențelor funcționale care se pot deriva din mulțimea F , aplicând axiomele Armstrong. Este clar că $F^+ = (F^+)^+$.

Exemplul 3.6. Fie relația $r(ABC)$ și $F = \{AB \rightarrow C, C \rightarrow B\}$. Atunci $F^+ = \{A \rightarrow A, AB \rightarrow A, AC \rightarrow A, ABC \rightarrow A, B \rightarrow B, AB \rightarrow B, BC \rightarrow B, ABC \rightarrow B, C \rightarrow C, AC \rightarrow C, BC \rightarrow C, ABC \rightarrow C, AB \rightarrow AB, ABC \rightarrow AB, AC \rightarrow AC, ABC \rightarrow AC, BC \rightarrow BC, ABC \rightarrow BC, ABC \rightarrow ABC, AB \rightarrow C, AB \rightarrow AC, AB \rightarrow BC, AB \rightarrow ABC, C \rightarrow B, C \rightarrow BC, AC \rightarrow B, AC \rightarrow AB\}$.

În F^+ primele nouăsprezece dependențe sunt triviale și se derivă din mulțimea \emptyset de dependențe, aplicând DF1, adică $\emptyset \vdash \{X \rightarrow Y \mid Y \subseteq X \subseteq ABC\}$. Alte dependențe $X \rightarrow Y$ se derivă din $Z \rightarrow Y$, unde $Z \subset X$, aplicând regula incrementării (DF2), adică $\{Z \rightarrow Y\} \vdash \{X \rightarrow Y \mid Z \subset X \subseteq R \text{ și } Z \not\subseteq X \text{ și } Y \subseteq R\}$. În F^+ avem șase dependențe deduse, aplicând regula DF2 asupra F . Deci $AB \rightarrow C \vdash \{AB \rightarrow AC, AB \rightarrow BC, AB \rightarrow ABC\}$ și $\{C \rightarrow B\} \vdash \{C \rightarrow BC, AC \rightarrow B, AC \rightarrow AB\}$. Regula tranzitivității nu generează dependențe netriviale. În afară de aceasta, F^+ conține cele două dependențe din F . În total F^+ constă din douăzeci și șapte dependențe.

Din exemplul de mai sus, se observă că numărul de dependențe ce alcătuiesc F^+ este destul de mare în raport cu F .

Dacă $F = \emptyset$, atunci F^+ constă numai din dependențe triviale. Fiindcă orice relație $r(R)$ satisface orice dependență trivială asupra R , dependențele triviale, bineînțeles, nu se consideră constrângeri asupra relației. Întrucât R are $2^{|R|}$ submulțimi, numărul de dependențe triviale într-o relație este exponențial. Nu vom considera de asemenea dependențele de forma $X \rightarrow \emptyset$, fiindcă ele nu au aplicare practică.

3.4. Completitudinea regulilor de inferență

Definiția 3.6. Fie RI o mulțime de reguli de inferență asupra mulțimii de dependențe F . Mulțimea RI de reguli este *închisă*, dacă $F \vdash f$, utilizând regulile din RI , implică $F \models f$. Mulțimea de reguli de inferență RI este *completă*, dacă $F \models f$ implică $F \vdash f$, utilizând regulile din RI .

Că mulțimea de reguli DF1-DF6 este închisă, adică ele au loc în orice relație, s-a demonstrat pentru fiecare regulă în parte în secțiunea 3.2. Pentru a arăta că mulțimea de reguli este completă mai întâi introducem noțiunea de închidere a unei mulțimi de atribute.

Definiția 3.7. Fie F o mulțime de dependențe asupra R și $X \subseteq R$. Închiderea mulțimii de atribute X în raport cu mulțimea de dependențe F , notată cu X^+ , se definește astfel:

- (1) $X \subseteq X^+$ (X e o submulțime a închiderii);
- (2) Dacă $Z \subseteq X^+$ și $Z \rightarrow Y \in F$, atunci $Y \subseteq X^+$;
- (3) Nici un alt atribut nu face parte din X^+ .

Adică $X^+ = X \cup \{Y \mid Z \subseteq X^+ \text{ și } Z \rightarrow Y \in F\}$.

Condiție definiția 3.6.	Comparare atribute, dependențe	$(AB)^+$
----------------------------	-----------------------------------	----------

(1)	$AB \subseteq (AB)^+$	AB
(2)	$AB \subseteq AB$ și $AB \rightarrow DE \in F$	ABDE
(2)	$D \subseteq ABDE$ și $D \rightarrow C \in F$	ABCDE
(2)	$CE \subseteq ABCDE$ și $CE \rightarrow G \in F$	ABCDEG

Fig.3.7.

Exemplul 3.7. Fie $R=ABCDEG$ și $F=\{D \rightarrow C, AB \rightarrow DE, CE \rightarrow G\}$. Să se arate că $(AB)^+ = ABCDEG$. În fig.3.7 este reprezentat procesul de construire a $(AB)^+$.

r	X^+	$R \setminus X^+$
t_1	a a ... a	a a ... a
t_2	a a ... a	b b ... b

Fig.3.8.

Teorema 3.2. Mulțimea de reguli de inferență DF1-DF6 este completă.

Demonstrație. Fie F o mulțime de dependențe asupra R . Trebuie să arătăm că, dacă $F \models X \rightarrow Y$, atunci $F \models -X \rightarrow Y$ (sau, ce e echivalent, dacă $F \models -X \rightarrow Y$, atunci $F \models X \rightarrow Y$).

Fie $X \rightarrow Y$ o dependență funcțională ce nu se deduce din F , adică $F \models -X \rightarrow Y$. Să arătăm că relația ce satisface toate dependențele din F nu satisface $X \rightarrow Y$, adică $F \not\models X \rightarrow Y$.

Definim relația $r(R)$ în felul următor (vezi fig.3.8). Ea constă din două tupluri t_1 și t_2 . Tuplul $t_1 = aa...a$; tuplul t_2 se definește astfel $t_2[A] = \{a, \text{dacă } A_i \in X^+; b, \text{dacă } A_i \in R \setminus X^+\}$.

Mai întâi, vom arăta că relația construită satisface toate dependențele din F . Presupunem contrariul. Există în F o dependență $V \rightarrow W$ ce nu e validă în $r(R)$. Atunci $V \subseteq X^+$ și $W \not\subseteq X^+$, altminteri relația r va satisface dependența $V \rightarrow W$. Întrucât $W \not\subseteq X^+$, există în W cel puțin un atribut A și $A \notin X^+$. Conform regulii reflexivității $V \subseteq X^+$ implică $X^+ \rightarrow V$. Dependențele $X^+ \rightarrow V$ și $V \rightarrow W$ implică, conform regulii DF5, $X^+ \rightarrow W$. Din $A \in W$ urmează $W \rightarrow A$. Aplicând asupra $X^+ \rightarrow W$ și $W \rightarrow A$ regula tranzitivității obținem $X^+ \rightarrow A$. Din ultima dependență, $X^+ \rightarrow A$, urmează că $A \in X^+$, ce contrazice faptului că $A \notin X^+$. Deci relația construită satisface orice dependență din F .

Acum să arătăm că relația noastră nu satisface dependența $X \rightarrow Y$. Presupunem că relația $r(R)$ satisface dependența $X \rightarrow Y$. Atunci $t_1[X] = t_2[X]$ implică $t_1[Y] = t_2[Y]$. Aceasta se poate întâmpla doar când $Y \subseteq X^+$. Din $Y \subseteq X^+$, aplicând regula DF1, urmează $X^+ \rightarrow Y$. Dependența $X \rightarrow X^+$ este în F^+ . Aplicând asupra $X \rightarrow X^+$ și $X^+ \rightarrow Y$ regula DF5, obținem $X \rightarrow Y$. Dar aceasta contrazice ipoteza că $F \models -X \rightarrow Y$.

Deci, dacă $F \models X \rightarrow Y$, atunci $F \models -X \rightarrow Y$.

Luând în considerație că (în compartimentul 3.2.) s-a demonstrat că mulțimea de reguli este închisă, adică $F \models -X \rightarrow Y$ implică $F \models X \rightarrow Y$ putem spune că mulțimea de reguli de inferență DF1-DF6 este închisă și completă, adică $F \models X \rightarrow Y$, dacă și numai dacă $F \models -X \rightarrow Y$. Prin urmare, putem utiliza " \models " și " $\models -$ " deopotrivă.

Dat fiind faptul că regulile DF1-DF6 se deduc din axiomele Armstrong, vom spune că și axiomele Armstrong sunt închise și complete.

3.5. Modele de derivări

Noțiunea de consecutivitate de derivare introdusă în secțiunea 3.3 are o serie de dezavantaje. De regulă, pentru o dependență f pot exista o serie de derivări din mulțimea de dependențe date F . Aceste derivări, fiind în esență echivalente, diferă prin ordinea și tipul regulilor aplicate pentru derivarea dependențelor din consecutivitate. În plus, consecutivitățile de derivare pot conține dependențe derivate redundante. Mai jos vom examina modele de derivări ce într-o măsură sau alta sunt lipsite de aceste dezavantaje.

3.5.1. RAP–consecutivități de derivare

Axiomele Armstrong sunt o submulțime completă de reguli de inferență a mulțimii DF1-DF6. Există, însă, mulțimi complete de reguli de inferență ce nu sunt submulțimi ale mulțimii DF1-DF6. Considerăm o astfel de mulțime de reguli de inferență denumită B-axiome.

Pentru relația $r(R)$, submulțimile W, X, Y și Z ale mulțimii R și $C \in R$ avem:

B1. Regula reflexivității. Dacă $Y \subseteq X$, atunci $X \rightarrow Y$.

B2. Regula acumulării. Dacă $X \rightarrow YZ$ și $Z \rightarrow CW$, atunci $X \rightarrow YZC$.

B3. Regula proiectivității. Dacă $X \rightarrow YZ$, atunci $X \rightarrow Y$.

Teorema 3.3. Mulțimea de reguli B1-B3 este o mulțime închisă.

Demonstrație. În secțiunea 3.2 s-a demonstrat că regulile reflexivității și proiectivității au loc în orice relații r cu schema R . Să examinăm regula acumulării. Presupunem contrariul: relația r satisface dependențele $X \rightarrow YZ$, $Z \rightarrow CW$ și nu satisface $X \rightarrow YZC$. Atunci există două tupluri t_1 și t_2 pentru care $t_1[X] = t_2[X]$, dar $t_1[YZC] \neq t_2[YZC]$. Din $t_1[YZC] \neq t_2[YZC]$ urmează sau $t_1[YZ] \neq t_2[YZ]$, sau $t_1[C] \neq t_2[C]$. Dacă $t_1[YZ] \neq t_2[YZ]$, atunci dependența $X \rightarrow YZ$ nu e validă în r . Dacă $t_1[C] \neq t_2[C]$ atunci r nu satisface dependența $Z \rightarrow CW$. Prin urmare, dependența $X \rightarrow YZC$ e validă în orice relație, în care sunt valide dependențele $X \rightarrow YZ$ și $Z \rightarrow CW$.

Teorema 3.4. Mulțimea de reguli B1-B3 este completă.

Demonstrație. Pentru a arăta că regulile B1-B3 sunt complete, e de ajuns să arătăm că axiomele Armstrong se deduc din B-axiome.

Regula reflexivității DF1 coincide cu B1.

Să arătăm că regula incrementării, DF2, urmează din regulile B1-B3. Fie $X \rightarrow Y$. Din regula B1 urmează $XZ \rightarrow XZ$. Aplicând asupra $XZ \rightarrow XZ$ și $X \rightarrow Y$ regula B2 de atâtea ori câte atribute sunt în Y , obținem $XZ \rightarrow XZY$. Conform regulii proiectivității, B3, obținem $XZ \rightarrow Y$.

Să demonstrăm că regula tranzitivității DF5 urmează din B1-B3. Fie relația r satisface dependențele funcționale $X \rightarrow Y$ și $Y \rightarrow Z$. Din B1 obținem $X \rightarrow X$. Consecutiv, aplicăm mai întâi regula B2 asupra $X \rightarrow X$ și $X \rightarrow Y$ de atâtea ori câte atribute sunt în Y și obținem $X \rightarrow XY$. Aplicând asupra $X \rightarrow XY$ și $Y \rightarrow Z$, regula B2, obținem $X \rightarrow XYZ$. O singură aplicare a regulii B3 produce $X \rightarrow Y$.

Definiția 3.8. Consecutivitatea de dependențe funcționale se numește RAP-consecutivitate de derivare (după primele litere ale denumirilor B-axiomele):

Reflexivitate, Acumulare, Proiectivitate) a unei dependențe $X \rightarrow Y$ din mulțimea de dependențe F , dacă:

- (1) prima dependență în consecutivitate e $X \rightarrow X$;
- (2) ultima dependență în consecutivitate e $X \rightarrow Y$ (obținută, aplicând regula B3);
- (3) orice dependență din consecutivitate (în afară de prima și ultima) sau este o dependență din F , sau are forma $X \rightarrow Z$ și e obținută, aplicând regula B2 asupra dependențelor precedente.

Exemplul 3.8. Fie $F = \{AB \rightarrow E, AG \rightarrow J, BE \rightarrow I, E \rightarrow G, GI \rightarrow H\}$. Consecutivitatea de mai jos este RAP-consecutivitate de derivare a dependenței $AB \rightarrow GH$ din F .

- | | |
|--------------------------------|--------------------|
| $f_1 := AB \rightarrow AB$ | (B1), |
| $f_2 := AB \rightarrow E$ | (dată), |
| $f_3 := AB \rightarrow ABE$ | (B2: f_1, f_2), |
| $f_4 := BE \rightarrow I$ | (dată), |
| $f_5 := AB \rightarrow ABEI$ | (B2: f_3, f_4), |
| $f_6 := E \rightarrow G$ | (dată), |
| $f_7 := AB \rightarrow ABEIG$ | (B2: f_5, f_6), |
| $f_8 := GI \rightarrow H$ | (dată), |
| $f_9 := AB \rightarrow ABEIGH$ | (B2: f_7, f_8), |
| $f_{10} := AB \rightarrow GH$ | (B3: f_9). |

3.5.2. DDA-grafuri de derivare

DDA-graful (Derivation Directed Acyclic-graph) este o interpretare grafică a RAP-consecutivității de derivare.

Definiția 3.9. Fie F o mulțime de dependențe funcționale asupra schemei R . DDA-graf asupra F este un graf orientat fără cicluri ce se definește recursiv:

- R1. O mulțime de noduri notate cu atribute din R este DDA-graf.
- R2. Dacă H este DDA-graf și B_1, \dots, B_n sunt noduri în H și dependența funcțională $B_1 \dots B_n \rightarrow CZ$ este în F , atunci H^1 , obținut din H prin adăugarea nodului C (dacă astfel de nod nu există) și muchiilor $(B_1C) \dots (B_nC)$ orientate spre C , este DDA-graf.
- R3. DDA-graf este numai graful obținut prin aplicarea regulilor R1 și R2.


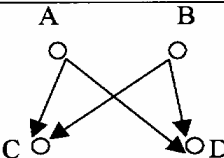
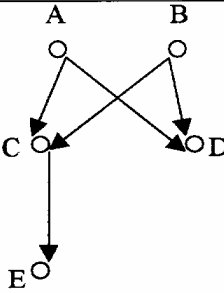
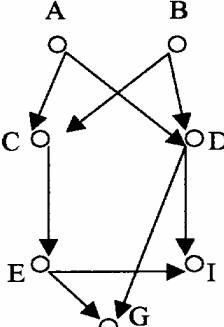
DDA-graf	Regulă, dependență
	R1
	R2, $AB \rightarrow CD$
	R2, $C \rightarrow E$
	R2, $DE \rightarrow GI$

Fig.3.9.

Definiția 3.10. Nodul B al unui DDA-graf H se numește *inițial*, dacă în el nu intră nici o muchie (Nodurile inițiale se adaugă în H prin regula R1).

Definiția 3.11. Fie H un DDA-graf asupra F. Graful H se numește DDA-graf de derivare a dependenței $X \rightarrow Y$ din F, dacă:

- (1) X este mulțimea de noduri inițiale în H;
- (2) orice atribut din Y este un nod în H.

Definiția 3.12. Mulțimea de dependențe din F utilizate de regula R2 pentru construirea DDA-grafului H de derivare a unei dependențe $X \rightarrow Y$ din F se numește *mulțime utilizabilă*, notată cu $U(H)$.

Exemplul 3.9. În fig.3.9 sunt prezentate etapele de construire a DDA-grafului de derivare a dependenței $AB \rightarrow CG$ din mulțimea de dependențe funcționale $F = \{AB \rightarrow CD, A \rightarrow I, C \rightarrow E, DE \rightarrow GI\}$.

Nodurile inițiale sunt A și B. Mulțimea utilizabilă $U(H) = \{AB \rightarrow CD, C \rightarrow E, DE \rightarrow GI\}$. Graful H obținut este DDA-graf de derivare a dependenței $AB \rightarrow CG$ din F, fiindcă mulțimea de attribute AB formează nodurile inițiale din H, iar CG sunt noduri în H.

3.5.3. Derivația maximală

Modelele de derivare descrise mai sus poartă mai mult un caracter teoretic. Ele nu sunt lipsite de neajunsul că pentru o dependență dată există mai multe consecutivități de derivare. Aici vom considera un model, numit derivare maximală, liber de acest dezavantaj. Acest model este foarte aproape de noțiunea de închidere a unei mulțimi de atribute în raport cu o mulțime de dependențe funcționale. El va fi utilizat în demonstrarea diverselor rezultate privind acoperirile de dependențe funcționale.

Definiția 3.13. Fie F o mulțime de dependențe funcționale asupra schemei R și fie $X \subseteq R$. Derivația maximală a mulțimii de atribute X în raport cu mulțimea de dependențe funcționale F este o consecutivitate de mulțimi de atribute $\langle X_0, X_1, \dots, X_n \rangle$, unde

- (1) $X_0 = X$;
- (2) $X_i = X_{i-1} \cup Z$, $1 \leq i \leq n$, unde $Z = \cup_j W_j$ pentru orice dependență $V_j \rightarrow W_j \in F$ ce satisface $V_j \subseteq X_{i-1}$ și $W_j \not\subseteq X_{i-1}$;
- (3) în F nu există nici o dependență $V_j \rightarrow W_j$ pentru care $V_j \subseteq X_n$ și $W_j \not\subseteq X_n$.

Înainte de a arăta că derivația maximală este un instrument puternic de modelare a derivării dependențelor funcționale, considerăm două proprietăți ale ei.

Lema 3.1. Dacă $X \subseteq Y$ și consecutivitățile $\langle X_0, X_1, \dots, X_n \rangle$, $\langle Y_0, Y_1, \dots, Y_m \rangle$ sunt derivații maxime ale mulțimilor X și Y , corespunzător, în raport cu F , atunci pentru orice X_i există o mulțime Y_j încât $X_i \subseteq Y_j$ și $j \leq i$.

Demonstrație. Vom arăta aceasta, aplicând inducția matematică asupra i . Când $i=0$ avem $X_0 \subseteq Y_0$, fiindcă $X \subseteq Y$. Fie că presupunerea noastră e justă pentru $i=k$: $X_k \subseteq Y_p$ și $p \leq k$. Să arătăm că ceea ce trebuie de demonstrat are loc și pentru $i=k+1$. Într-adevăr, la pasul $k+1$, $X_{k+1} = X_k \cup Z$, unde $Z = \cup_j W_j$ pentru toate dependențele $V_j \rightarrow W_j$ din F determinanții și determinații cărora satisfac $V_j \subseteq X_k$ și $W_j \not\subseteq X_k$ corespunzător. Conform ipotezei inducției $X_k \subseteq Y_p$. Prin urmare, toți determinanții dependențelor $V_j \rightarrow W_j$ ce se conțin în X_k se vor conține și în Y_p . Dat fiind faptul că mulțimea Y_p e mai "largă" ea poate conține toți determinații W_j și atunci $X_{k+1} \subseteq Y_p$. Dacă nu, atunci în derivația mulțimii Y în raport cu F se execută următorul $p+1$ pas, în rezultatul căruia vom obține Y_{p+1} care va conține X_{k+1} .

Lema 3.2. Dacă $\langle X_0, X_1, \dots, X_n \rangle$ este derivația maximală a mulțimii X în raport cu mulțimea de dependențe funcționale F , atunci $X \rightarrow X_i \in F^+$, $0 \leq i \leq n$.

Demonstrație. Vom face demonstrarea, utilizând inducția asupra numărului de aplicări a regulii (2) în construirea derivației maxime.

Fie că în construirea derivației maxime nu s-a aplicat regula (2). Atunci ea are un singur element X_0 , unde $X_0 = X$ și conform regulii reflexivității, $DF1$, $X \rightarrow X_0 \in F^+$.

Presupunem că la aplicarea $i-1$ a regulii (2) are loc $X \rightarrow X_{i-1} \in F^+$. Să se arate veridicitatea afirmației pentru pasul i . Fără a constrânge generalitatea, presupunem că la acest pas avem o singură dependență $V \rightarrow W$ ce satisface $V \subseteq X_{i-1}$, $W \not\subseteq X_{i-1}$. Conform regulii reflexivității $X_{i-1} \rightarrow V \in F^+$. Dar $X_{i-1} \rightarrow V \in F^+$ și $V \rightarrow W \in F^+$ (regula tranzitivității) implică $X_{i-1} \rightarrow W \in F^+$. Adăugăm la determinantul și determinatul ultimei dependențe mulțimea X_{i-1} . Obținem (conform regulii incrementării) $X_{i-1} \rightarrow X_{i-1} W \in F^+$. Din $X \rightarrow X_{i-1} \in F^+$ (ipoteza inducției) și $X_{i-1} \rightarrow X_{i-1} W \in F^+$ urmează $X \rightarrow X_i \in F^+$.

În baza acestor două proprietăți vom demonstra

Teorema 3.5. Fie $\langle X_0, X_1, \dots, X_n \rangle$ derivația maximală a mulțimii X în raport cu mulțimea de dependențe funcționale F . Atunci $X \rightarrow Y \in F^+$ dacă și numai dacă $Y \subseteq X_n$.

Demonstrație. Necesitatea. Să arătăm că, dacă $X \rightarrow Y \in F^+$, atunci există o mulțime X_i în derivația maximală $\langle X_0, X_1, \dots, X_n \rangle$ încât $Y \subseteq X_i$ și, prin urmare, $Y \subseteq X_n$. Vom utiliza inducția asupra (lungimii derivării) numărului de dependențe folosite în derivarea dependenței $X \rightarrow Y$ în raport cu F , unde dependența de derivare sau este în F , sau se deduce din regula reflexivității sau din regula incrementării, aplicate asupra unei dependențe precedente, sau cu ajutorul regulii tranzitivității, aplicate asupra a două dependențe precedente. Ultima dependență în derivare, bineînțeles, e $X \rightarrow Y$.

Fie că derivarea dependenței $X \rightarrow Y$ are lungimea 1, adică constă din însuși $X \rightarrow Y$. Sunt două cazuri. Sau $X \rightarrow Y$ se deduce din axioma reflexivității, sau $X \rightarrow Y \in F$. În primul caz, $Y \subseteq X$ și, prin urmare, $Y \subseteq X_0$. În al doilea caz, dependența $X \rightarrow Y$ va participa la formarea elementului al doilea a derivației maxime a mulțimii X în raport cu F . Deci $Y \subseteq X_1$.

Presupunem acum că afirmația noastră e justă pentru o derivare cu lungimea mai mică decât k și să demonstrăm veridicitatea afirmației noastre pentru derivarea cu lungimea k . Considerăm consecutiv regulile de inferență ce pot fi aplicate la acest pas.

Dacă pentru deducerea dependenței $X \rightarrow Y$ se aplică regula reflexivității sau $X \rightarrow Y \in F$, atunci Y se comportă ca și pentru derivări cu lungimea unu, adică Y se pomeniște corespunzător în X_0 și X_1 .

Dacă $X \rightarrow Y$ urmează din regula incrementării asupra unei dependențe precedente $V \rightarrow W$, atunci există S și T , unde $T \subseteq S$ și $VS = X$, $WT = Y$. Întrucât $V \rightarrow W$ are o lungime mai mică decât k , atunci conform ipotezei inducției există în derivația maximală o mulțime V_j , unde $W \subseteq V_j$. Fiindcă $V \subseteq X$, atunci conform lemei 3.1 există în derivația maximală pentru X în raport cu F o mulțime X_i , unde $W \subseteq X_i$. Din $T \subseteq S \subseteq X$ urmează $T \subseteq X_0$ și $T \subseteq X_i$.

Considerăm ultimul caz, când dependența $X \rightarrow Y$ e obținută, aplicând regula tranzitivității asupra a două dependențe precedente $X \rightarrow Z$ și $Z \rightarrow Y$, derivațiile cărora au lungimi mai mici decât k .

Urmând ipoteza inducției, pentru $X \rightarrow Z$ și $Z \rightarrow Y$ avem corespunzător $Z \subseteq X_j$ și $Y \subseteq Z_p$. Însă Z_p este element al derivației maxime a mulțimii Z în raport cu F . Fiindcă $Z \subseteq X_j$, conform lemei 3.1 $Z_p \subseteq X_{j+m}$, unde X_{j+m} este elementul $m+1$ al derivației maxime a mulțimii X_j în raport cu F pe care o vom nota $\langle X_{j+0}, X_{j+1}, \dots, X_{j+m}, \dots, X_{j+p} \rangle$. Este evident că derivația maximală a mulțimii X_j nu e altceva decât o subconsecutivitate ce constă din ultimele $n-j+1$ elemente ale derivației maxime a mulțimii X în raport cu F . Prin urmare, $Y \subseteq X_i$, unde $i = j+m$.

Suficiența. Fie $\langle X_0, X_1, \dots, X_n \rangle$ e derivația maximală a mulțimii X în raport cu F . Conform lemei 3.2 $X \rightarrow X_n \in F^+$. Întrucât $Y \subseteq X_n$, atunci aplicând regula proiectivității asupra dependenței $X \rightarrow X_n$ obținem $X \rightarrow Y \in F^+$. Teorema e demonstrată.

Definiția 3.14. Fie $X \rightarrow Y \in F^+$ și $\langle X_0, X_1, \dots, X_n \rangle$ derivația maximală a mulțimii X în raport cu F . Fie X_i este primul element din consecutivitate ce conține mulțimea Y . Subconsecutivitatea $\langle X_0, X_1, \dots, X_i \rangle$ se numește *derivația* dependenței funcționale $X \rightarrow Y$ în raport cu F .

Din teorema 3.5 și definiția 3.14 urmează

Consecința 3.1. $X \rightarrow Y \in F^+$ atunci și numai atunci când există derivația dependenței $X \rightarrow Y$ în raport cu F .

Consecința 3.2. Dacă $X \rightarrow Y \in F^+$ și dependența $V \rightarrow W \in F$ e utilizată în construirea derivației dependenței $X \rightarrow Y$ în raport cu F , atunci $X \rightarrow V \in F^+$.

Justețea acestei afirmații decurge imediat din lema 3.2 și regula proiectivității.

Trebuie menționat că derivația maximală este un model de derivare liber de dezavantajele menționate la începutul acestei secțiuni. Existența a unei singure derivații pentru o dependență dată va fi utilă în expunerea de mai departe a materiei.

3.5.4. Algoritmi

Pentru a determina dacă $F \models X \rightarrow Y$, e suficient de verificat dacă $X \rightarrow Y \in F^+$. Însă, F^+ este excesiv de mare în raport cu F . E dezirabilă o metodă de verificare, dacă $X \rightarrow Y$ aparține F^+ , fără a deduce toate dependențele funcționale din F . Un astfel de algoritm e prezentat mai jos. Nucleul algoritmului constă din procedura de construire a închiderii mulțimii de atribute X în raport cu F . După ce se găsește X^+ , se verifică dacă $Y \subseteq X^+$.

Este evident că ultimul element, X_n , din derivația maximală nu este altceva decât X^+ . Iar teorema 3.5 ne sugerează că $X \rightarrow Y$ urmează logic din F , dacă $Y \subseteq X^+$. Deci, derivația maximală servește drept model teoretic pentru următorul algoritm de determinare a lui X^+ .

Algoritmul CLOSURE caută în F o dependență funcțională pentru care determinantul reprezintă o submulțime a lui X_i , iar determinatul nu este inclus în X_i . Dacă se găsește o astfel de dependență funcțională, atunci se adaugă la X_i atributele care constituie determinatul dependenței. Dacă nu se găsește, atunci închiderea căutată, X^+ este reprezentată de mulțimea de atribute X_i .

Algoritmul CLOSURE (F, X, X^+)

Intrare: F - o mulțime de dependențe funcționale asupra schemei R ; X - o mulțime de atribute, $X \subseteq R$.

Ieșire: X^+ - închiderea mulțimii X în raport cu F

begin

$i := 0; X_i := X;$

repeat

$i := i + 1;$

$X_i := X_{i-1};$

For all $V \rightarrow W$ in F

if $V \subseteq X_i$ then $X_i := X_i \cup W;$

until $X_i = X_{i-1};$

return ($X^+ := X_i$);

end

Exemplul 3.10. Fie $F = \{B \rightarrow CD, AD \rightarrow E, B \rightarrow A\}$, $X = B$. Să se calculeze X^+ .

Inițial $X_0 = B$.

În ciclul repeat:

$X_1 = B$.

În ciclul for:

$X_1 = BCD$ (aplicând $B \rightarrow CD$),

$X_1 = ABCD$ (aplicând $B \rightarrow A$).
 $X_2 = ABCD$
 În ciclul for:
 $X_2 = ABCDE$ (aplicând $AD \rightarrow E$)
 $X_3 = ABCDE$
 După ciclul for: $X_3 = X_2$.
 Rezultat: $X^+ = ABCDE$.
 Deci închiderea lui B în raport cu F este $(B)^+ = ABCDE$.

Algoritmul CLOSURE realmente construiește derivația maximală a mulțimii de attribute X în raport cu F. Apelând la CLOSURE e ușor de construit algoritmul de verificare a apartenenței unei dependențe funcționale la F^+ . Această verificare e realizată în algoritmul MEMBERSHIP.

Algoritmul MEMBERSHIP($F, X \rightarrow Y$)

Intrare: F- o mulțime de dependențe funcționale;

$X \rightarrow Y$ – o dependență funcțională.

Ieșire: adevăr, dacă $F \models X \rightarrow Y$, fals - în caz contrar.

begin

CLOSURE (F, X, X^+);

if $Y \subseteq X^+$ then return (adevăr) else return (fals);

end

Corectitudinea algoritmilor CLOSURE și MEMBERSHIP rezultă imediat din teorema 3.5.

3.6. Acoperiri

În această secțiune se consideră diverse moduri de reprezentare a mulțimilor de dependențe, cum ar fi mulțimile nonredundante, reduse, canonice, minimale și optimale.

3.6.1. Mulțimi echivalente de dependențe funcționale

Definiția 3.15. Două mulțimi de dependențe funcționale F și G se numesc *echivalente*, notat cu $F \equiv G$, dacă $F^+ = G^+$. Vom mai spune în acest caz că F *acoperă* G (sau G acoperă F).

Dacă $F \equiv G$, adică $F^+ = G^+$, atunci orice dependență $X \rightarrow Y$ ce urmează logic din F urmează și din G. Deci pentru a verifica dacă F și G sunt echivalente se ia orice dependență $X \rightarrow Y$ din F și se verifică dacă $G \models X \rightarrow Y$. Dacă o oarecare dependență $X \rightarrow Y$ nu aparține lui G^+ , atunci $F^+ \neq G^+$. Apoi analogic se verifică dacă orice dependență $V \rightarrow W$ din G se deduce din F. Dacă toate dependențele se deduc, mulțimile F și G sunt echivalente.

Exemplul 3.11. Mulțimile $F = \{AB \rightarrow C, AC \rightarrow D, AD \rightarrow B, C \rightarrow B\}$ și $G = \{AD \rightarrow C, AB \rightarrow D, C \rightarrow B\}$ sunt echivalente, însă F nu este echivalentă mulțimii $G^1 = \{AB \rightarrow C, AC \rightarrow D, AD \rightarrow B, AC \rightarrow B\}$ (a se verifica în calitate de exercițiu).

3.6.2. Acoperiri nonredundante

Definiția 3.16. Mulțimea de dependențe funcționale F este *nonredundantă*, dacă nu există o submulțime proprie F^1 a mulțimii F și $F^1 \equiv F$. Dacă o astfel de submulțime

există, atunci F se numește *redundantă*. Mulțimea F este *acoperire nonredundantă* a mulțimii G , dacă F este acoperire pentru G și F este nonredundantă.

Exemplul 3.12. Fie $G = \{A \rightarrow BC, B \rightarrow C\}$. Mulțimea $F = \{A \rightarrow B, A \rightarrow C, B \rightarrow C\}$ este acoperire a mulțimii G , dar nu e acoperire nonredundantă, fiindcă $F^1 = \{A \rightarrow B, B \rightarrow C\}$ e acoperire pentru G , însă $F^1 \subset F$.

Să considerăm o altă interpretare a noțiunii de mulțime nonredundantă.

Definiția 3.17. Mulțimea F de dependențe funcționale se numește nonredundantă, dacă în ea nu există nici o dependență $X \rightarrow Y$ încât $(F \setminus \{X \rightarrow Y\}) \models X \rightarrow Y$. În caz contrar, F se numește redundanță.

Această definiție este pusă în baza următorului algoritm de construire a acoperirii nonredundante. E de menționat că rezultatul obținut în urma aplicării algoritmului depinde de ordinea considerării dependențelor funcționale.

Algoritmul NONREDUN (F, G)

Intrare: F – o mulțime de dependențe funcționale.

Ieșire: G – o acoperire nonredundantă a mulțimii F .

begin

$G := F$;

for all $X \rightarrow Y$ in G

if MEMBERSHIP ($G \setminus \{X \rightarrow Y\}, X \rightarrow Y$) then

$G := G \setminus \{X \rightarrow Y\}$;

return (G);

end

Exemplul 3.13. Fie $F = \{A \rightarrow B, B \rightarrow A, B \rightarrow C, A \rightarrow C\}$. În rezultatul aplicării algoritmului NONREDUN obținem acoperirea nonredundantă $G = \{A \rightarrow B, B \rightarrow A, A \rightarrow C\}$. Dacă mulțimea F e prezentată în altă ordine $\{A \rightarrow B, A \rightarrow C, B \rightarrow A, B \rightarrow C\}$ se obține rezultatul $G = \{A \rightarrow B, B \rightarrow A, B \rightarrow C\}$.

3.6.3. Acoperiri reduse

Dacă F e o mulțime nonredundantă, atunci nu poate fi eliminată din F nici o dependență funcțională fără a afecta echivalența mulțimii obținute cu cea anterioară. În schimb poate fi micșorată dimensiunea mulțimii F , eliminând unele atribute din dependențele funcționale.

Definiția 3.18. Fie F o mulțime de dependențe funcționale asupra schemei R și $X \rightarrow Y \in F$. Atributul A este *redundant* în dependența $X \rightarrow Y$ în raport cu F , dacă

(1) $A \in X, V = X \setminus A$ și $F \setminus \{X \rightarrow Y\} \cup \{V \rightarrow Y\} \equiv F$

sau

(2) $A \in Y, W = Y \setminus A$ și $F \setminus \{X \rightarrow Y\} \cup \{X \rightarrow W\} \equiv F$.

Cu alte cuvinte, atributul A este redundant în dependența $X \rightarrow Y$, dacă el poate fi eliminat din determinant sau determinat, fără a fi schimbată închiderea mulțimii F . Procesul de eliminare a atributelor redundante se numește, corespunzător, reducere în stânga și reducere în dreapta a dependențelor.

Exemplul 3.14. Fie $F = \{AC \rightarrow B, A \rightarrow C, A \rightarrow BD\}$. Atributul C este redundant în $AC \rightarrow B$, fiindcă $(AC)^+ = A^+ = ACBD$. Adică $A \rightarrow B$ este în F^+ . Deci dependența $AC \rightarrow B$

poate fi înlocuită cu $A \rightarrow B$ în mulțimea de dependențe funcționale F . Atributul B este redundant în partea dreaptă a dependenței $A \rightarrow BD$.

Definiția 3.19. Fie F o mulțime de dependențe funcționale asupra schemei R . Mulțimea F se numește *redușă în stânga* (dreapta), dacă orice dependență din F nu are attribute redundante în partea stângă (dreaptă). Mulțimea de dependențe redusă în stânga și în dreapta se numește *redușă*.

Exemplu 3.15. Mulțimea $F = \{AC \rightarrow B, A \rightarrow C, A \rightarrow BD\}$ nu este redusă nici în stânga, nici în dreapta. Mulțimea $F_1 = \{A \rightarrow B, A \rightarrow C, A \rightarrow BD\}$ e redusă în stânga și nu e redusă în dreapta, dar $F_2 = \{AC \rightarrow B, A \rightarrow C, A \rightarrow D\}$ e redusă în dreapta și nu în stânga. Mulțimea de dependențe funcționale $F_3 = \{A \rightarrow B, A \rightarrow C, A \rightarrow D\}$ e redusă în stânga și în dreapta, deci e redusă.

Mai jos se aduc algoritmi de reducere a unei mulțimi nonredundante.

Algoritmul LEFTRED (F, G)

Intrare: F – o mulțime nonredundantă de dependențe funcționale.

Ieșire: G – o mulțime de dependențe funcționale redusă în stânga.

```
begin
  G:=F;
  for all  $X \rightarrow Y$  in  $G$ 
    for all  $A$  in  $X$ 
      if MEMBERSHIP ( $G, (X \setminus A) \rightarrow Y$ ) then
         $G := G \setminus \{X \rightarrow Y\} \cup \{(X \setminus A) \rightarrow Y\}$ ;
  return ( $G$ );
end
```

Algoritmul RIGHTRED (F, G)

Intrare: F – o mulțime nonredundantă de dependențe funcționale.

Ieșire: G – o mulțime de dependențe funcționale redusă în dreapta

```
begin
  G:=F;
  for all  $X \rightarrow Y$  in  $G$ 
    for all  $A$  in  $Y$ 
      if MEMBERSHIP ( $G \setminus \{X \rightarrow Y\} \cup \{X \rightarrow (Y \setminus A)\}, X \rightarrow A$ ) then  $G := \{G \setminus \{X \rightarrow Y\} \cup \{X \rightarrow (Y \setminus A)\}$ ;
  return ( $G$ );
end
```

Algoritmul REDUCE (F, G)

Intrare: F – o mulțime de dependențe funcționale nonredundantă.

Ieșire: G – o mulțime redusă de dependențe de dependențe funcționale.

```
begin
  LEFTRED ( $F, F^1$ );
  RIGHTRED ( $F^1, G$ );
  eliminarea din  $G$  a dependențelor  $X \rightarrow \emptyset$ ;
  return ( $G$ );
end
```

În algoritmul LEFTRED de mai sus, este inclusă condiția de verificare $G|=(X \setminus A) \rightarrow Y$. Este evident că, dacă $(X \setminus A) \rightarrow Y$ se deduce din G , atunci $X \rightarrow Y$ poate fi substituită cu $(X \setminus A) \rightarrow Y$, fiindcă $(X \setminus A) \rightarrow Y| = X \rightarrow Y$.

E evident că, dacă o dependență este redundantă, atunci toate atributele ei sunt redundante. Pentru a evita apariția dependențelor de forma $\emptyset \rightarrow \emptyset$ se presupune că mulțimea destinată reducerii este nonredundantă.

S-ar părea că acoperirile reduse pot fi calculate, găsind și eliminând în mod aleator atributele redundante. Însă, examinând părțile stângi și drepte ale dependențelor în ordine diferită, obținem rezultate diferite. În părțile drepte odată examinate pot apărea atribute redundante după examinarea părților stângi. Deci eliminarea atributelor redundante trebuie să înceapă cu partea stângă. Dar și în cazul acesta pot apărea dependențe de forma $X \rightarrow \emptyset$. Ele se elimină la urmă din mulțimea rezultat.

3.6.4. Acoperiri canonice

Definiția 3.20. Mulțimea de dependențe funcționale F este *canonică*, dacă F este nonredundantă, redusă în stânga și orice dependență din F are forma $X \rightarrow A$.

Întrucât mulțimea canonică este nonredundantă și redusă în stânga, iar determinatul oricărei dependente constă dintr-un singur atribut, ea este redusă și în dreapta, adică este redusă.

Exemplul 3.16. Mulțimea $F = \{A \rightarrow B, A \rightarrow C, B \rightarrow D\}$ este o acoperire canonică a mulțimii $G = \{A \rightarrow BC, B \rightarrow D\}$.

Teorema 3.6. Fie F o acoperire redusă. Se formează mulțimea G , dezagregând orice dependență de forma $X \rightarrow A_1 \dots A_n$ în $X \rightarrow A_1, \dots, X \rightarrow A_n$. Atunci G este canonică. Fie G o acoperire canonică. Formăm F , agregând dependențele cu determinanți egali. Mulțimea F este acoperire redusă.

Demonstrație. Fie mulțimea G este formată din F prin dezagregarea dependențelor. Presupunem că G nu e canonică. Dacă dependența $X \rightarrow A_i$ e redundantă, atunci atributul A_i e redundant în $X \rightarrow A_1 \dots A_n$. Dacă $X \rightarrow A_i$ conține un atribut redundant în X , fie B , atunci $G|=(X \setminus B) \rightarrow A_i$. Dar $G|=(X \setminus B) \rightarrow X$, fiindcă $X \rightarrow A_i$ e nonredundantă. De unde conchidem că $F|=(X \setminus B) \rightarrow X$ și, prin urmare, B e redundant în partea stângă a dependenței $X \rightarrow A_1 \dots A_n$ din F .

Să demonstrăm a doua parte a teoremei. Presupunem contrariul: F nu e redusă. Dacă F nu e redusă în dreapta, atunci G nu e nonredundantă. Dar dacă F nu e redusă în stânga, atunci și G nu e redusă din stânga, ce contravine presupunerii că G e canonică.

3.6.5. Clase de echivalență

Definiția 3.21. Fie F o mulțime de dependențe funcționale asupra schemei R și $X, Y \subseteq R$. Mulțimile de atribute X și Y sunt *echivalente*, notăm cu $X \leftrightarrow Y$, în raport cu F , dacă $F| = X \rightarrow Y$ și $F| = Y \rightarrow X$.

Această definiție ne sugerează că mulțimea F poate fi partiționată în clase de echivalență. Adică asupra F se poate defini o relație de echivalență: dependențele $X \rightarrow Y$ și $V \rightarrow W$ din F aparțin unei clase de echivalență, dacă și numai dacă $X \leftrightarrow V$ în raport cu F .

Notăm cu $E_F(X)$ mulțimea de dependențe cu determinanții echivalenți lui X în raport cu F , adică $E_F(X) = \{V \rightarrow W \mid F \models V \rightarrow X \text{ \& } F \models X \rightarrow V\}$. $E_F(X)$ se numește *clasă de echivalență* a dependențelor cu determinanții echivalenți mulțimii de atribut X .

Notăm cu $\bar{E}_F = \{E_F(X) \mid X \subseteq R \text{ \& } E_F(X) \neq \emptyset\}$, adică \bar{E}_F este mulțimea tuturor claselor de echivalență nevide, în care este partiționată mulțimea de dependențe funcționale F .

Exemplul 3.17. Fie $F = \{AB \rightarrow C, AC \rightarrow D, AD \rightarrow B, C \rightarrow B\}$. Atunci $(AB)^+ = (AC)^+ = (AD)^+ = ABCD$ și $C^+ = BC$. Deci $AB \leftrightarrow AC$, $AD \leftrightarrow AC$, $AB \leftrightarrow AD$, adică $\bar{E}_F = \{E_F(AB) = \{AB \rightarrow C, AC \rightarrow D, AD \rightarrow B\}, E_F(C) = \{C \rightarrow B\}\}$.

Următoarea leamnă arată corelația dintre structurile a două acoperiri nonredundante.

Lema 3.3. Fie F și G două mulțimi de dependențe funcționale nonredundante echivalente asupra schemei R . Fie $X \rightarrow Y$ o dependență în F . Atunci în G există o dependență $V \rightarrow W$, unde $X \leftrightarrow V$ în raport cu F (și în raport cu G).

Demonstrație. Din faptul că $F = G$ urmează $G \models X \rightarrow Y$. Atunci există derivația H a dependenței funcționale $X \rightarrow Y$, în raport cu G . Considerăm toate dependențele utilizate în construirea derivației H , adică mulțimea $U(H)$. În același timp, orice dependență $V \rightarrow W$ din $U(H)$ se deduce din F . Fie H^1 este derivația dependenței $V \rightarrow W$ în F . Trebuie să existe în $U(H)$ o dependență $V \rightarrow W$ pentru deducerea căreia se utilizează dependența $X \rightarrow Y$, adică $X \rightarrow Y \in U(H^1)$. În caz contrar pentru dependența $X \rightarrow Y$ va exista o derivație H^{11} asupra $F \setminus \{X \rightarrow Y\}$ și, prin urmare, $X \rightarrow Y$ va fi redundanță în F ce contrazice ipotezei că F este o mulțime nonredundantă.

Întrucât $X \rightarrow Y \in U(H^1)$, conform consecinței 3.2, $F \models V \rightarrow X$. Dar $V \rightarrow W \in U(H)$ și atunci $G \models X \rightarrow V$. Prin urmare, $X \leftrightarrow V$.

Lema de mai sus poate fi parafrazată în felul următor. În două acoperiri nonredundante F și G pentru orice dependență din F există o dependență în G ce are partea stângă echivalentă celei din F . Prin urmare, mulțimile nonredundante echivalente au același număr de clase de echivalență.

Exemplul 3.18. Mulțimile $F = \{A \rightarrow BC, B \rightarrow A, AD \rightarrow E\}$ și $G = \{A \rightarrow ABC, B \rightarrow A, BD \rightarrow E\}$ sunt nonredundante și echivalente. Să observăm că $A \leftrightarrow A$, $B \leftrightarrow B$, $AD \leftrightarrow BD$ și $A \leftrightarrow B$. Deci $\bar{E}_F = \{E_F(A) = \{A \rightarrow BC, B \rightarrow A\}, E_F(AD) = \{AD \rightarrow E\}\}$ și $\bar{E}_G = \{E_G(A) = \{A \rightarrow ABC, B \rightarrow A\}, E_G(BD) = \{BD \rightarrow E\}\}$.

3.6.6. Acoperiri minimale

Definiția 3.22. Mulțimea de dependențe funcționale F este *minimală*, dacă nu există o mulțime echivalentă ei cu mai puține dependențe funcționale.

Este evident că orice mulțime minimală de dependențe funcționale este și nonredundantă. Afirmația inversă nu este corectă.

Exemplul 3.19. Mulțimea $F = \{A \rightarrow B, A \rightarrow C\}$ este nonredundantă, dar nu este minimală, fiindcă $G = \{A \rightarrow BC\}$ este acoperire pentru F și are o singură dependență.

Fie $e_F(X)$ este mulțimea părților stângi ale dependențelor ce formează clasa de echivalență $E_F(X)$. Atunci are loc

Lema 3.4. Fie F o mulțime nonredundantă de dependențe funcționale, X determinantul unei dependențe din F și Y o mulțime de atribute echivalentă lui X (adică $Y \leftrightarrow X$ în raport cu F). Atunci există în $e_F(X)$ o mulțime Z , încât $(F \setminus E_F(X)) \models Y \rightarrow Z$.

Demonstrație. Dacă $Y \in e_F(X)$, atunci lema e demonstrată, fiindcă $Y \rightarrow Y$ urmează din orice mulțime de dependențe funcționale. Fie $Y \notin e_F(X)$. Întrucât $Y \rightarrow Z$ pentru orice Z din $e_F(X)$, atunci există derivația $H = \langle Y_0, Y_1, \dots, Y_m \rangle$, unde $Y_0 = Y$ și $Z \subseteq Y_m$. Dacă în construirea lui H nu s-a utilizat nici o dependență din $E_F(X)$, atunci lema e demonstrată. Presupunem că pentru construirea derivației H s-a utilizat dependența $V \rightarrow W \in E_F(X)$ și fie $V \rightarrow W$ e prima dependență din $E_F(X)$ utilizată în H . Atunci în H există un Y_i încât $V \subseteq Y_i$ dar $W \not\subseteq Y_i$. Dar $Y \rightarrow Y_i \in (F \setminus E_F(X))^+$ și, conform reflexivității, $Y_i \rightarrow V$ are loc în orice mulțime de dependențe. Aplicând regula tranzitivității asupra ultimelor dependențe, obținem că $(F \setminus E_F(X)) \models Y \rightarrow V$, adică $Z = V$.

Lema 3.5. Fie F și G două mulțimi nonredundante echivalente de dependențe funcționale. Fie X e determinantul unei dependențe din F și Y o mulțime de atribute, unde $Y \leftrightarrow X$. Dacă $Y \rightarrow Z \in (F \setminus E_F(X))^+$, atunci $Y \rightarrow Z \in (G \setminus E_G(X))^+$.

Demonstrație. Întrucât $Y \rightarrow Z \in (F \setminus E_F(X))^+$, atunci există derivația $H = \langle Y_0, Y_1, \dots, Y_m \rangle$, pentru $Y \rightarrow Z$ în raport cu $F \setminus E_F(X)$. Fie $V \rightarrow W$ o dependență utilizată în construirea lui H . Din $F \equiv G$ urmează $V \rightarrow W \in G^+$.

Să arătăm că în derivația dependenței $V \rightarrow W$ în raport cu G nu sunt utilizate dependențe din $E_G(X)$. Presupunem contrariul: pentru derivarea $V \rightarrow W$ este utilizată dependența $T \rightarrow S$ din $E_F(X)$. Atunci, conform lemei 3.3, $Y \leftrightarrow T$, iar din consecința 3.2 $V \rightarrow T \in G^+$. Din $V \rightarrow T \in G^+$ și $T \rightarrow Y \in G^+$ urmează $V \rightarrow Y \in G^+$. Însă $Y \rightarrow V \in G^+$ și atunci obținem că $Y \leftrightarrow V$ în raport cu F . Contrazicere. Deci, orice dependență utilizată în derivația lui $Y \rightarrow Z$ în raport cu $F \setminus E_F(X)$ se deduce din $G \setminus E_G(H)$.

Deci, derivația dependenței $Y \rightarrow Z$ va utiliza numai dependențe din $G \setminus E_G(H)$.

Teorema 3.7. O mulțime nonredundantă F este minimală, dacă și numai dacă nici o clasă de echivalență $E_F(X)$ nu conține două dependențe diferite $X \rightarrow Y$ și $V \rightarrow W$, unde $X \rightarrow V \in (F \setminus E_F(X))^+$.

Demonstrație. Necesitatea. Fie F este o mulțime minimală, și presupunem contrariul: în clasa de echivalență $E_F(X)$ sunt două dependențe diferite $X \rightarrow Y$ și $V \rightarrow W$ încât $X \rightarrow V \in (F \setminus E_F(X))^+$. Construim o mulțime de dependențe G , care se deosebește de F , prin aceea că în clasa de echivalență $E_G(X)$ dependențele $X \rightarrow Y$ și $V \rightarrow W$ sunt substituite de $V \rightarrow YW$. Vom arăta ca $F \equiv G$. Pentru aceasta e suficient să verificăm dacă $X \rightarrow Y \in G^+$. Conform lemei 3.5 $X \rightarrow V \in (G \setminus E_G(X))^+$. Din $X \rightarrow V \in (G \setminus E_G(X))^+$ și $V \rightarrow YW \in G^+$ urmează că $X \rightarrow YW \in G^+$. Deci $F \equiv G$, însă G conține cu o dependență mai puțin, ce contrazice ipotezei că F e o mulțime minimală de dependențe funcționale.

Suficiența. Vom arăta că, dacă orice clasă de echivalență a unei mulțimi F nu conține două dependențe diferite, încât părțile stângi să se determine funcțional în afara propriei clase de echivalență, atunci F este minimală. Să demonstrăm că nu există o mulțime nonredundantă G și $G \equiv F$, încât careva clasă de echivalență din G să conțină mai puține dependențe decât clasa corespunzătoare din F .

Presupunem contrariul: există o mulțime nonredundantă G și $G \equiv F$, iar clasa de echivalență $E_G(X)$ conține mai puține dependențe decât clasa $E_F(X)$.

Să evidențiem dependențele acestor două clase de echivalență, unde $m < n$ (vezi fig.3.10).

$E_F(X)$	$E_G(X)$
$X_1 \rightarrow Y_1$	$V_1 \rightarrow W_1$
$X_2 \rightarrow Y_2$	$V_2 \rightarrow W_2$
\dots	\dots
$X_n \rightarrow Y_n$	$V_m \rightarrow W_m$

Fig.3.10.

În corespundere cu lema 3.4 pentru orice $X_j \in e_F(X)$ există în $e_G(X)$ un determinant V_k și $X_j \rightarrow V_k \in (G \setminus E_G(X))^+$. Apelând la lema 3.5. obținem $X_j \rightarrow V_k \in (F \setminus E_F(X))^+$. Întrucât $n > m$, atunci se vor găsi în $e_F(X)$ cel puțin două mulțimi X_j și X_l , unde $j \neq l$, ce satisfac $X_j \rightarrow V_k \in (F \setminus E_F(X))^+$ și $X_l \rightarrow V_k \in (F \setminus E_F(X))^+$. La rândul său, în $e_F(X)$ există un determinant X_h și $V_k \rightarrow X_h \in (F \setminus E_F(X))^+$. Considerăm două cazuri posibile: $h \neq j$ sau $h = j$. Dacă $h \neq j$, atunci $X_j \rightarrow V_k \in (F \setminus E_F(X))^+$ și $V_k \rightarrow X_h \in (F \setminus E_F(X))^+$ implică $X_j \rightarrow X_h \in (F \setminus E_F(X))^+$. Dacă, însă, $h = j$, atunci obținem $V_l \rightarrow X_h \in (F \setminus E_F(X))^+$.

În ambele cazuri, clasa de echivalență $E_F(X)$ va conține două dependențe diferite, părțile stângi ale cărora se determină funcțional în afara clasei de echivalență examinată. Contradicție.

Consecința 3.3. Dacă F și G sunt mulțimi minimale echivalente, atunci clasele de echivalență corespunzătoare conțin același număr de dependențe funcționale.

Consecința 3.4. Dacă F și G sunt două mulțimi minimale echivalente, atunci pentru orice determinant $X_j \in e_F(X)$ există un singur determinant V_k în $e_G(X)$ pentru care au loc $X_j \rightarrow V_k \in (F \setminus E_F(X))^+$ și $V_k \rightarrow X_j \in (F \setminus E_F(X))^+$.

Remarcă. Existența corespondenței biunivoce, indicate în consecința 3.4, permite substituirea unor părți stângi ale mulțimii minimale cu părți stângi ale altei acoperiri minimale, neafectând echivalența mulțimilor. Mai mult ca atât, mulțimea nouă de dependențe funcționale va continua să fie minimală.

În teorema de mai sus se afirmă că, dacă o mulțime nonredundantă G are două dependențe $X \rightarrow Y$ și $V \rightarrow W$ pentru care $X \leftrightarrow V$ și $(G \setminus E_G(X)) \models X \rightarrow V$, atunci G nu este minimală. Aceste două dependențe pot fi substituite cu altă dependență $V \rightarrow YW$. În rezultat obținem o mulțime echivalentă de dependențe funcționale ce conține cu o dependență mai puțin.

Acest proces este pus la baza următorului algoritm de minimizare a unei mulțimi de dependențe funcționale.

Algoritmul MINIMIZE (f,g)

Intrare: F – o mulțime de dependențe funcționale.

Ieșire: G – o mulțime minimală de dependențe funcționale.

begin

NONREDUN (F, G);

get \bar{E}_G ;

for all $E_G(X)$ în \bar{E}_G

for all $X \rightarrow Y$ în $E_G(X)$

for all $V \rightarrow W \neq X \rightarrow Y$ în $E_G(X)$

```

        if MEMBERSHIP (G \ EG(X), X→V) then G:= G \ {X→Y,
        V→W} ∪ {V→YW};
    return (G);
end

```

Exemplul 3.20. Fie $F = \{AB \rightarrow D, AB \rightarrow C, AC \rightarrow D, AD \rightarrow B, C \rightarrow B\}$. Să construim acoperirea minimală a mulțimii F .

Nu e greu de observat că, dacă examinăm dependențele funcționale din F de la stânga la dreapta, dependența funcțională $AB \rightarrow D$ e redundantă în F . Într-adevăr, $(AB)^+$ în raport cu $F \setminus \{AB \rightarrow D\}$ este $ABCD$. Deci $(F \setminus \{AB \rightarrow D\})^+ = AB \rightarrow D$. Am obținut mulțimea nonredundantă $G = \{AB \rightarrow C, AC \rightarrow D, AD \rightarrow B, C \rightarrow B\}$.

Să partiționăm G în clase de echivalență. Pentru aceasta construim închiderile determinanților tuturor dependențelor din G . Dependențele ce au închideri ale determinanților egale fac parte din aceeași clasă de echivalență. Așadar,

$(AB)^+ = ABCD,$
 $(AC)^+ = ABCD,$
 $(AD)^+ = ABCD,$
 $C^+ = BC.$

Deci, mulțimea G conține următoarele două clase de echivalență $G = \{E_G(AB) = \{AB \rightarrow C, AC \rightarrow D, AD \rightarrow B\}, E_G(C) = \{C \rightarrow B\}\}$.

Întrucât clasa $E_G(C)$ conține o singură dependență, se examinează numai clasa de echivalență $E_G(AB)$. Nu e greu de verificat că în $E_G(AB)$ sunt două dependențe $AC \rightarrow D$, $AB \rightarrow C$ și $(G \setminus E_G(AB))^+ = AC \rightarrow AB$. Atunci în clasa de echivalență $E_G(AB)$ aceste două dependențe se substituie cu $AB \rightarrow CD$.

Am obținut mulțimea de dependențe $G = \{E_G(AB) = \{AB \rightarrow CD, AD \rightarrow B\}, E_G(C) = \{C \rightarrow B\}\}$. În clasa de echivalență $E_G(AB)$ nu se mai găsesc dependențe determinanții cărora se determină funcțional în afara clasei $E_G(AB)$. Prin urmare, mulțimea $G = \{AB \rightarrow CD, AD \rightarrow B, C \rightarrow B\}$ este o acoperire minimală a mulțimii F .

3.6.7. Acoperiri optimale

Mulțimea de dependențe funcționale F poate fi estimată după numărul de attribute (inclusiv repetate) antrenate de dependențele funcționale din F . De pildă, mulțimea $F = \{AB \rightarrow C, C \rightarrow B\}$ are aritatea cinci.

Definiția 3.23. Mulțimea de dependențe funcționale F se numește *optimală*, dacă nu există o mulțime echivalentă ei cu o aritate mai mică.

Exemplul 3.21. Mulțimea $F = \{ABC \rightarrow E, BC \rightarrow D, D \rightarrow BC\}$ nu este acoperire optimală, fiindcă mulțimea $G = \{AD \rightarrow E, BC \rightarrow D, D \rightarrow BC\}$ are aritatea mai mică decât F și $G = F$. Mulțimea G este optimală.

Trebuie menționat că problema construirii unei acoperiri optimale aparține clasei de probleme NP-complete, pentru care încă nu au fost găsiți algoritmi polinomiali.

Teorema 3.8. Mulțimea optimală este minimală și redusă.
 Demonstrarea acestei afirmații se lasă în calitate de exercițiu.

3.7. Exerciții

- 3.1. Să se aducă un exemplu de două atribute ce se găsesc într-o dependență funcțională și un exemplu de două atribute ce nu sunt funcțional dependente.

r	A	B	C	D
	a ₁	b ₁	c ₁	d ₁
	a ₁	b ₁	c ₂	d ₂
	a ₂	b ₁	c ₁	d ₃
	a ₂	b ₁	c ₃	d ₄

Fig.3.11.

- 3.2. Să se găsească dependențele funcționale valide în relația r din fig.3.11.
- 3.3. Fie r e definită pe mulțimea ABC. Să se aducă o extensie a relației r ce ar satisface dependența funcțională $A \rightarrow B$ și nu ar satisface dependența $C \rightarrow A$.
- 3.4. Să se arate că $\{WX \rightarrow Y\} \mid -X \rightarrow Y$.
- 3.5. Fie relația r(R) și $V, W, X, Y, Z \subseteq R$. Să se demonstreze sau să se combată următoarele reguli de inferență.
- $\{X \rightarrow Y, Z \rightarrow W\} \mid -XZ \rightarrow YW$;
 - $\{XY \rightarrow Z, Z \rightarrow X\} \mid -Z \rightarrow Y$;
 - $\{X \rightarrow Y, Y \rightarrow Z\} \mid -X \rightarrow YZ$;
 - $\{ZV \rightarrow W, W \rightarrow XV, V \rightarrow Y\} \mid -ZV \rightarrow XY$;
 - $\{X \rightarrow Y, W \rightarrow Z\} \mid -X \rightarrow Z$, unde $W \subseteq Y$.
- 3.6. Să se arate că regulile DF1, DF2 și DF6 sunt independente, adică nici una din ele nu se deduce din celelalte.
- 3.7. Să se arate că pentru orice mulțime de dependențe funcționale F are loc egalitatea $F^+ \equiv (F^+)^+$.
- 3.8. Fie F o mulțime de dependențe funcționale asupra schemei R. Care este F^+ , dacă $F = \emptyset$?
- 3.9. Fie mulțimea $F = \{AB \rightarrow C, C \rightarrow B\}$ definită asupra atributelor ABC. Să se găsească F^+ .
- 3.10. Să se demonstreze că sistemul de reguli DF1, DF3, DF4 și DF5 este complet. Sunt aceste reguli independente?
- 3.11. Fie $F = \{AB \rightarrow C, B \rightarrow D, CD \rightarrow E, CE \rightarrow GH, G \rightarrow A\}$.
- Să se construiască consecutivitatea de derivare a dependenței $AB \rightarrow E$ din F.
 - Să se construiască consecutivitatea de derivare a dependenței $AB \rightarrow G$ din F, utilizând axiomele Armstrong.

- (c) Să se construiască RAP–consecutivitatea de derivare a dependenței $AB \rightarrow G$ din F .
- (d) Să se construiască DDA-graful de derivare a dependenței $AB \rightarrow G$ din F .
- 3.12. Să se arate că mulțimile $\{AB \rightarrow E, CD \rightarrow F, A \rightarrow C, B \rightarrow D, C \rightarrow A, D \rightarrow B, F \rightarrow AD\}$ și $G = \{A \rightarrow C, B \rightarrow D, C \rightarrow A, D \rightarrow B, F \rightarrow AD, CD \rightarrow EF\}$ sunt echivalente.
- 3.13. Să se găsească mulțimea claselor de echivalență \bar{E}_G , dacă $G = \{AB \rightarrow EF, A \rightarrow C, B \rightarrow D, C \rightarrow A, D \rightarrow B, F \rightarrow AD\}$.
- 3.14. Să se construiască o acoperire nonredundantă a mulțimii de dependențe funcționale $F = \{A \rightarrow C, AB \rightarrow C, C \rightarrow DI, CD \rightarrow I, EC \rightarrow AB, EI \rightarrow G\}$.
- 3.15. Să se construiască o mulțime de dependențe funcționale în care o dependență funcțională $X \rightarrow Y$ are toate atributele redundante.
- 3.16. Să se construiască două mulțimi de dependențe funcționale F și G , unde F e o acoperire nonredundantă a mulțimii G , dar conține un număr mai mare de dependențe decât G .
- 3.17. Fie F mulțimea tuturor dependențelor posibile asupra schemei $R = A_1 \dots A_n$, în afară de $\emptyset \rightarrow X$. Să se găsească o acoperire nonredundantă a mulțimii F .
- 3.18. Să se găsească două mulțimi nonredundante echivalente cu un număr diferit de dependențe.
- 3.19. Fie $F = \{A \rightarrow B, B \rightarrow C, C \rightarrow A\}$ și $G = \{A \rightarrow BC, B \rightarrow A, C \rightarrow A\}$.
- (a) Să se arate că mulțimile F și G sunt echivalente.
- (b) Să se găsească o acoperire redusă a mulțimii G .
- 3.20. Fie $G = \{AF \rightarrow C, C \rightarrow D, D \rightarrow BE, B \rightarrow CE, CE \rightarrow A\}$. Să se găsească o acoperire canonică a mulțimii de dependențe G .
- 3.21. Să se găsească acoperire minimală a mulțimii $G = \{AB \rightarrow C, BC \rightarrow D, BE \rightarrow C, CD \rightarrow B, CE \rightarrow AF, CF \rightarrow BD, C \rightarrow A, D \rightarrow EF\}$.
- 3.22. Să se construiască o acoperire optimală a mulțimii $G = \{A \rightarrow BC, BC \rightarrow A, ABD \rightarrow EF, BCD \rightarrow EF\}$.

DEPENDENȚE MULTIVALOARE ȘI DEPENDENȚE JONCȚIUNE

Modelul relațional utilizează dependențele pentru exprimarea constrângerilor pe care datele din baza de date trebuie să le satisfacă. Schema bazei de date relaționale este definită de o varietate de constrângeri ce sunt impuse componentelor sale. Dependențele funcționale sunt un exemplu de astfel de constrângeri de integritate. Ele au fost studiate detaliat în capitolul 3.

O generalizare a dependențelor funcționale, numite dependențe multivaloare, a fost descoperită de mai mulți cercetători în domeniu. Cea mai importantă proprietate a dependenței multivaloare constă în faptul că existența ei într-o relație este o condiție necesară și suficientă pentru ca relația să poată fi înlocuită fără pierderi de informații, independent de extensia curentă, cu două proiecții ale sale. Această proprietate face ca dependența multivaloare să joace un rol important în teoria și practica proiectării bazelor de date relaționale.

O dată ce dependențele multivaloare au devenit parte a teoriei relațiilor, o cerință de bază ce trebuie să fie satisfăcută este cunoașterea proprietăților lor și, în particular, metodelor de manipulare. Întrucât dependențele multivaloare sunt o generalizare a celor funcționale, metodele aplicate asupra ultimelor pot servi drept ghid în susținerea acestei cerințe.

Este bine cunoscut că existența într-o relație a dependențelor funcționale implică că în ea există dependențe funcționale adiționale. Aceasta e valabil și pentru dependențele multivaloare. Noțiunea de implicare este formalizată în conceptul de reguli de inferență. Sunt cunoscute mulțimi închise și complete de reguli de inferență pentru dependențele multivaloare.

Dependențele joncțiune sunt o generalizare a dependențelor multivaloare. E cunoscut faptul că o mulțime de dependențe funcționale plus o dependență joncțiune se consideră suficiente pentru exprimarea dependențelor dintre atributele unei scheme a bazei de date.

Acest capitol cuprinde noțiuni generale despre dependențele multivaloare, regulile de inferență, dependențele multivaloare incluse, regulile de inferență ale dependențelor joncțiune etc.

4.1. Dependențe multivaloare

Definiția 4.1. Fie relația r cu schema R și $X, Y \subseteq R$. Notăm $Z = R \setminus XY$. Vom spune că relația $r(R)$ satisface *dependența multivaloare* $X \twoheadrightarrow Y$ (sau $X \twoheadrightarrow Y$ e validă în $r(R)$), dacă pentru orice pereche de tupluri t_1 și t_2 din $r(R)$ ce satisfac $t_1[X] = t_2[X]$ există în $r(R)$ un tuplu t_3 pentru care au loc egalitățile $t_3[X] = t_1[X]$, $t_3[Y] = t_1[Y]$ și $t_3[Z] = t_2[Z]$.

Remarcă. Din proprietatea de simetrie a acestei definiții urmează că în $r(R)$ mai există un tuplu t_4 ce satisface egalitățile $t_4[X] = t_1[X]$, $t_4[Y] = t_2[Y]$ și $t_4[Z] = t_1[Z]$.

Teorema 4.1. O dependență multivaloare $X \twoheadrightarrow Y$ e validă în relația $r(R)$ dacă și numai dacă $X \twoheadrightarrow Z$ e validă în $r(R)$, unde $Z = R \setminus XY$.

Demonstrație. Din remarcă definiției 4.1 urmează că, dacă relația $r(R)$ satisface dependența multivaloare $X \twoheadrightarrow Y$, atunci de fiecare dată când $t_1[X] = t_2[X]$ în $r(R)$ există nu numai un tuplu t_3 ce satisface $t_3[X] = t_1[X]$, $t_3[Y] = t_1[Y]$ și $t_3[Z] = t_2[Z]$, dar și un tuplu t_4 pentru care au loc egalitățile $t_4[X] = t_1[X]$, $t_4[Y] = t_2[Y]$ și $t_4[Z] = t_1[Z]$. În consecință, tupluri distincte cu aceleași X-valori și cu Y-valori (Z-valori) identice trebuie să aibă diferite Z-valori (Y-valori) pentru a menține toate tuplurile distincte. Din această proprietate simetrică rezultă că relația $r(R)$ satisface dependența multivaloare $X \twoheadrightarrow Y$ dacă și numai dacă satisface dependența multivaloare $X \twoheadrightarrow Z$.

Exemplul 4.1. Relația $r(ABCD)$ din fig.4.1 satisface dependența multivaloare $BC \twoheadrightarrow A$. În relația $r(ABCD)$ e validă de asemenea dependența multivaloare $BC \twoheadrightarrow D$. Dacă, însă, din relația $r(ABCD)$ este eliminat un tuplu, atunci dependențele multivaloare $BC \twoheadrightarrow A$ și $BC \twoheadrightarrow D$ devin invalide în $r(ABCD)$.

r	A	B	C	D
	a_1	b_1	c_1	d_1
	a_1	b_1	c_1	d_2
	a_1	b_1	c_2	d_1
	a_1	b_1	c_2	d_2
	a_2	b_1	c_1	d_1
	a_2	b_1	c_1	d_2
	a_2	b_1	c_2	d_1
	a_2	b_1	c_2	d_2

Fig.4.1

În definiția 4.1 nu s-au pus condiții asupra mulțimilor X și Y . Deci $X \cap Y \neq \emptyset$ în caz general. Determinatul Y poate fi redus. Să demonstrăm că varianta redusă, $X \twoheadrightarrow Y \setminus X$, e echivalentă dependenței $X \twoheadrightarrow Y$.

Teorema 4.2. Dependența funcțională $X \twoheadrightarrow Y$ e validă în relația $r(R)$, dacă și numai dacă $X \twoheadrightarrow Y \setminus X$ e validă în $r(R)$.

Demonstrație. *Necesitatea.* Fie relația $r(R)$ satisface dependența multivaloare $X \twoheadrightarrow Y$. Notăm $Y^1 = Y \setminus X$. Atunci $Z = R \setminus XY = R \setminus XY^1$. Fie t_1 și t_2 două tupluri cu X-valori egale, adică $t_1[X] = t_2[X]$. Fiindcă $X \twoheadrightarrow Y$ e validă în $r(R)$, atunci în r trebuie să existe un tuplu t_3 ce satisface $t_3[X] = t_1[X]$, $t_3[Y] = t_1[Y]$ și $t_3[Z] = t_2[Z]$. Egalitatea $t_3[Y] = t_1[Y]$ implică egalitatea $t_3[Y^1] = t_1[Y^1]$. Prin urmare, relația r satisface și dependența multivaloare $X \twoheadrightarrow Y^1$.

Suficiența. Fie $r(R)$ satisface dependența multivaloare $X \twoheadrightarrow Y^1$, unde $Y^1 = Y \setminus X$ și fie $X^1 \subseteq X$. Să arătăm că dependența $X \twoheadrightarrow Y^1 X^1$ e validă în $r(R)$. Întrucât r satisface $X \twoheadrightarrow Y^1$ și dacă $t_1, t_2 \in r$ și $t_1[X] = t_2[X]$, atunci există un tuplu t_3 , pentru care $t_3[X] = t_1[X]$, $t_3[Y^1] = t_1[Y^1]$ și $t_3[Z] = t_2[Z]$. Din $X^1 \subseteq X$ și $t_3[Y^1] = t_1[Y^1]$ urmează $t_3[Y^1 X^1] = t_1[Y^1 X^1]$. Deci $X \twoheadrightarrow Y^1 X^1$.

Exemplul 4.2. Relația $r(ABCD)$ din fig.4.1 satisface dependența multivaloare $BC \twoheadrightarrow A$. Conform teoremei 4.2 în r e validă și dependența multivaloare $BC \twoheadrightarrow AB$.

Teorema ce urmează poate fi considerată o metodă de verificare dacă o dependență multivaloare e validă într-o relație.

Teorema 4.3. Fie relația $r(R)$, $X, Y \subseteq R$ și $Z = R \setminus XY$. Dependența multivaloare $X \twoheadrightarrow Y$ e validă în $r(R)$ dacă și numai dacă r este joncțiunea proiecțiilor sale $\pi_{XY}(r)$ și $\pi_{XZ}(r)$.

Demonstrație. Necesitatea. Fie dependența multivaloare $X \twoheadrightarrow Y$ e validă în $r(R)$ și fie $r_1 = \pi_{XY}(r)$, $r_2 = \pi_{XZ}(r)$. Fiindcă întotdeauna are loc corelația $r(R) \subseteq r_1 \bowtie r_2$, pentru a demonstra necesitatea, e de ajuns să arătăm că orice tuplu t din joncțiunea $r_1 \bowtie r_2$ este și în $r(R)$, adică $r_1 \bowtie r_2 \subseteq r(R)$. Fie t un tuplu în $r_1 \bowtie r_2$. Atunci r_1 și r_2 trebuie să conțină corespunzător tuplurile t_1 și t_2 încât $t[X] = t_1[X] = t_2[X]$, $t[Y] = t_1[Y]$ și $t[Z] = t_2[Z]$. Întrucât r_1 și r_2 sunt proiecții ale relației r , atunci r conține tuplurile t_1^1 și t_2^1 pentru care $t_1[XY] = t_1^1[XY]$ și $t_2[XZ] = t_2^1[XZ]$. În r este un tuplu t_3 (dat fiind faptul că $X \twoheadrightarrow Y$ e validă în r) ce satisface $t_3[X] = t_1^1[X]$, $t_3[Y] = t_1^1[Y]$ și $t_3[Z] = t_2^1[Z]$. Se vede că acest tuplu t_3 este t .

Suficiența. Presupunem acum că relația r se descompune în două relații r_1 și r_2 fără pierderi. Să arătăm că pentru orice două tupluri t_1^1 și t_2^1 ce satisfac $t_1^1[X] = t_2^1[X]$ există un tuplu t încât $t[X] = t_1^1[X]$, $t[Y] = t_1^1[Y]$ și $t[Z] = t_2^1[Z]$, adică în r e validă dependența multivaloare $X \twoheadrightarrow Y$.

Fie t_1^1 și t_2^1 două tupluri în $r(R)$. Întrucât $r(R)$ se descompune fără pierderi asupra XY și XZ (adică $r = \pi_{XY}(r) \bowtie \pi_{XZ}(r)$), atunci în r_1 și r_2 se găsesc, respectiv, tuplurile t_1 și t_2 și $t_1 = t_1^1[XY]$, $t_2 = t_2^1[XZ]$. Fiindcă $r = r_1 \bowtie r_2$, r conține un tuplu t ce satisface $t[XY] = t_1[XY]$ și $t[XZ] = t_2[XZ]$. Întrucât tuplurile t_1^1 , t_2^1 și t satisfac definiția 4.1 relația $r(R)$ satisface dependența multivaloare $X \twoheadrightarrow Y$.

Din teorema 4.3 se poate face următoarea concluzie.

Concluzie. Relația $r(R)$ se descompune fără pierderi în relațiile $r_1(R_1)$ și $r_2(R_2)$ dacă și numai dacă $R_1 \cap R_2 \twoheadrightarrow R_1$ (sau $R_1 \cap R_2 \twoheadrightarrow R_2$).

Este ineficientă utilizarea metodei din această teoremă pentru a verifica dacă o relație satisface sau nu o dependență multivaloare. Testarea necesită două proiecții și o joncțiune. Să examinăm un alt procedeu de verificare, dacă o dependență multivaloare e validă într-o relație.

Fie relația $r(R)$ satisface dependența multivaloare $X \twoheadrightarrow Y$, atunci conform teoremei 4.3 $r(R) = \pi_{XY}(r) \bowtie \pi_{XZ}(r)$, unde $Z = R \setminus XY$.

Expresiile $|\pi_{XY}(\sigma_{X=x}(r))|$ și $|\pi_{XZ}(\sigma_{X=x}(r))|$ reprezintă numerele de tupluri în proiecțiile relației r asupra mulțimilor XY și XZ , corespunzător, pentru X -valoarea dată egală cu x . Este evident că, dacă relația r se descompune fără pierderi în relațiile $\pi_{XY}(r)$ și $\pi_{XZ}(r)$, atunci

$$|\sigma_{X=x}(r)| = |\pi_{XY}(\sigma_{X=x}(r))| \cdot |\pi_{XZ}(\sigma_{X=x}(r))|. \quad (4.1)$$

Întrucât $|\pi_{XW}(\sigma_{X=x}(r))| = |\pi_W(\sigma_{X=x}(r))|$, atunci egalitatea (4.1) poate fi simplificată:

$$|\sigma_{X=x}(r)| = |\pi_Y(\sigma_{X=x}(r))| \cdot |\pi_Z(\sigma_{X=x}(r))|. \quad (4.2)$$

Această procedură de verificare a validității unei dependențe multivaloare este mai puțin laborioasă decât precedenta. Ea presupune sortarea tuplurilor după X -valori, apoi pentru orice X -valoare x se testează egalitatea de mai sus.

Exemplul 4.3. Relația $r(ABCD)$ din fig.4.1 satisface teorema 4.3. Proiecțiile $\pi_{BCA}(r)$ și $\pi_{BCD}(r)$ sunt prezentate în fig.4.2. Joncțiunea lor este egală cu $r(ABCD)$. Atunci egalitatea (4.2) e satisfăcută fiindcă:

(1) $|\pi_A(\sigma_{BC=b_1c_1}(r))| = |\pi_D(\sigma_{BC=b_1c_1}(r))| = 2$, și $|\sigma_{BC=b_1c_1}(r)| = 4$
și

(2) $|\pi_A(\sigma_{BC=b_1c_2}(r))| = |\pi_D(\sigma_{BC=b_1c_2}(r))| = 2$, și $|\sigma_{BC=b_1c_2}(r)| = 4$.

$\pi_{BCA}(r)$	B	C	A
	b_1	c_1	a_1
	b_1	c_2	a_1
	b_1	c_1	a_2
	b_1	c_2	a_2

$\pi_{BCD}(r)$	B	C	D
	b_1	c_1	d_1
	b_1	c_1	d_2
	b_1	c_2	d_1
	b_1	c_2	d_2

Fig.4.2.

Proprietatea de mai sus poate fi utilizată într-o nouă definiție a dependenței multivaloare.

Definiția 4.2. Fie r o relație asupra schemei R , $X, Y \subseteq R$ și $Z = R \setminus XY$. Relația $r(R)$ satisface dependența multivaloare $X \twoheadrightarrow Y$, dacă pentru orice X -valoare x și XZ -valoare xz e satisfăcută egalitatea $\pi_Y(\sigma_{X=x}(r)) = \pi_Y(\sigma_{XZ=xz}(r))$.

Cu alte cuvinte, în cadrul relației $r(R)$ există o dependență multivaloare $X \twoheadrightarrow Y$, dacă și numai dacă mulțimea valorilor lui Y corespunzătoare unei perechi xz depinde numai de X -valoarea x nu și de valoarea lui Z .

4.2. Reguli de inferență ale dependențelor multivaloare

Primele șase reguli sunt similare regulilor de inferență omonime ale dependențelor funcționale, însă numai primele trei conțin aceleași aserțiuni.

Fie o relație r cu schema R și $W, X, Y, Z \subseteq R$.

DM1. Regula reflexivității. Dacă $Y \subseteq X$, atunci $X \twoheadrightarrow Y$.

Validitatea acestei reguli urmează din definiția dependenței multivaloare.

DM2. Regula incrementării. Dacă $Z \subseteq W$ și $X \twoheadrightarrow Y$, atunci $XW \twoheadrightarrow YZ$.

Validitatea acestei afirmații reiese din definiția 4.1 și teorema 4.2.

DM3. Regula aditivității. Dacă $X \twoheadrightarrow Y$ și $X \twoheadrightarrow Z$, atunci $X \twoheadrightarrow YZ$.

Demonstrație. Fie în r sunt două tupluri t_1 și t_2 ce au X -valori egale, $t_1[X] = t_2[X]$. Trebuie arătat că în r există un tuplu t , încât $t[X] = t_1[X]$, $t[YZ] = t_1[YZ]$ și $t[U] = t_2[U]$, unde $U = R \setminus XYZ$.

Întrucât $r(R)$ satisface dependența $X \twoheadrightarrow Y$, atunci r conține un tuplu t_3 și $t_3[X] = t_1[X]$, $t_3[Y] = t_1[Y]$, $t_3[V] = t_2[V]$ pentru orice t_1 și t_2 ce satisfac egalitatea $t_1[X] = t_2[X]$, unde $V = R \setminus XY$. Același raționament este și pentru $X \twoheadrightarrow Z$: există în $r(R)$ un tuplu t_4 care satisface $t_4[X] = t_1[X]$, $t_4[Z] = t_1[Z]$ și $t_4[W] = t_3[W]$ (fiindcă $t_1[X] = t_3[X]$), unde $W = R \setminus XZ$.

Să arătăm că $t = t_4$. E evident că $t[X] = t_4[X]$. De asemenea $t_4[Z] = t_1[Z] = t[Z]$. Dar $t_4[Y \cap W] = t_3[Y \cap W] = t_1[Y \cap W] = t[Y \cap W]$ și atunci $t_4[YZ] = t[YZ]$. Din $U \subseteq W \cap V$, urmează $t_4[U] = t_3[U] = t_2[U] = t[U]$. Întrucât $R = XYZU$, atunci $t_4 = t$.

DM4. Regula proiectivității. Dacă $X \twoheadrightarrow Y$ și $X \twoheadrightarrow Z$, atunci $X \twoheadrightarrow Y \cap Z$, $X \twoheadrightarrow Y \setminus Z$.

Demonstrație. Conform regulii DM3, $X \twoheadrightarrow Y$ și $X \twoheadrightarrow Z$ implică $X \twoheadrightarrow YZ$. Aplicând teorema 4.1 asupra dependenței multivaloare $X \twoheadrightarrow YZ$, obținem $X \twoheadrightarrow R \setminus XYZ$. Aplicând regula DM3 asupra dependențelor $X \twoheadrightarrow R \setminus XYZ$ și $X \twoheadrightarrow Z$, obținem $X \twoheadrightarrow (R \setminus XYZ)Z$. Dar conform teoremei 4.1, $X \twoheadrightarrow (R \setminus XYZ)Z$ implică $X \twoheadrightarrow R \setminus X(R \setminus XYZ)Z$. Determinatul ultimei dependențe se simplifică în felul următor (vezi fig. 4.3): $R \setminus X(R \setminus XYZ)Z = R \setminus X(R \setminus Y)Z = Y \setminus XZ = (Y \setminus Z) \setminus X$. Deci $X \twoheadrightarrow (Y \setminus Z) \setminus X$ și conform teoremei 4.2 dependența $X \twoheadrightarrow Y \setminus Z$ este validă în $r(R)$.

Am demonstrat validitatea regulii: dacă $X \twoheadrightarrow Y$ și $X \twoheadrightarrow Z$, atunci $X \twoheadrightarrow Y \setminus Z$.

Să arătăm acum că, dacă $X \twoheadrightarrow Y$ și $X \twoheadrightarrow Z$, atunci $X \twoheadrightarrow Y \cap Z$.

Dependența $X \twoheadrightarrow Y$ implică dependența $X \twoheadrightarrow R \setminus XY$. Combinând, conform regulii DM3, dependențele $X \twoheadrightarrow Y \setminus Z$ și $X \twoheadrightarrow R \setminus XY$, obținem $X \twoheadrightarrow (R \setminus XY)(Y \setminus Z)$. Aplicând teorema 4.1 asupra ultimei dependențe multivaloare, obținem $X \twoheadrightarrow R \setminus X(R \setminus XY)(Y \setminus Z)$. Să examinăm partea dreaptă a dependenței multivaloare obținute, utilizând diagrama din fig.4.3.

$$R \setminus X(R \setminus XY)(Y \setminus Z) = R \setminus X(R \setminus Y)(Y \setminus Z) = Y \setminus X(Y \setminus Z) = (Y \cap Z) \setminus X.$$

Prin urmare, relația $r(R)$ satisface dependența multivaloare $X \twoheadrightarrow (Y \cap Z) \setminus X$ și, conform teoremei 4.2, satisface dependența $X \twoheadrightarrow Y \cap Z$.

DM5. Regula tranzitivității. Dacă $X \twoheadrightarrow Y$ și $Y \twoheadrightarrow Z$, atunci $X \twoheadrightarrow Z \setminus Y$.

Demonstrație. Dacă vom arăta că $X \twoheadrightarrow YZ$ e validă în relația r , atunci asupra acestei dependențe și dependenței $X \twoheadrightarrow Y$ poate fi aplicată regula DM4, pentru a obține dependența $X \twoheadrightarrow YZ \setminus Y$ sau $X \twoheadrightarrow Z \setminus Y$.

Notăm $W = R \setminus XYZ$ și să arătăm că $X \twoheadrightarrow Y$ și $Y \twoheadrightarrow Z$ implică $X \twoheadrightarrow YZ$. Adică, dacă în r sunt două tupluri t_1, t_2 și $t_1[X] = t_2[X]$, atunci r conține un tuplu t ce satisface egalitățile $t[X] = t_1[X]$, $t[YZ] = t_1[YZ]$ și $t[W] = t_2[W]$.

Întrucât dependența $X \twoheadrightarrow Y$ e validă în r , relația r conține un tuplu t_3 ce satisface $t_3[X] = t_1[X]$, $t_3[Y] = t_1[Y]$ și $t_3[V] = t_2[V]$, unde $V = R \setminus XY$. Dar dependența $Y \twoheadrightarrow Z$ presupune că în r este un tuplu t_4 ce satisface condițiile $t_4[Y] = t_1[Y]$, $t_4[Z] = t_1[Z]$ și $t_4[U] = t_3[U]$, unde $U = R \setminus YZ$.

Să arătăm că tuplul t_4 este tuplul căutat t . Întrucât $t_1[X] = t_2[X] = t_3[X] = t_4[X]$ e evident că $t_4[YZ] = t_1[YZ]$. Dat fiind faptul că $t_4[U] = t_3[U]$ și $W \subseteq U \setminus X$, atunci $t_4[W] = t_3[W]$. Din $t_3[V] = t_2[V]$ și $(U \setminus X) \subseteq V$ reiese că $t_3[W] = t_2[W]$. Deci, tuplul t_4 este cel căutat.

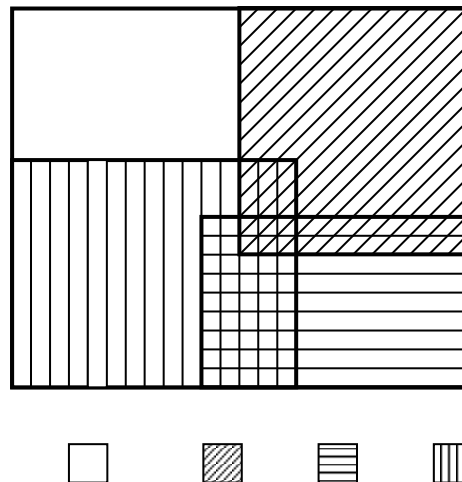


Fig. 4.3. - R, - X, - Y, - Z.

DM6. Regula pseudotranzitivității. Dacă $X \rightarrow \rightarrow Y$ și $YW \rightarrow \rightarrow Z$, atunci $XW \rightarrow \rightarrow Z \setminus YW$.

Demonstrarea acestei reguli e similară regulii tranzitivității și se propune în calitate de exercițiu.

DM7. Regula complementării. Dacă $X \rightarrow \rightarrow Y$, atunci $X \rightarrow \rightarrow Z$, unde $Z = R \setminus XY$. Validitatea acestei reguli este demonstrată de teorema 4.1.

Simbol	Denumire	Regulă
DM1	Reflexivitatea	$Y \subseteq X \Rightarrow X \rightarrow \rightarrow Y$
DM2	Incrementarea	$Z \subseteq W, X \rightarrow \rightarrow Y \Rightarrow XW \rightarrow \rightarrow YZ$
DM3	Aditivitatea	$X \rightarrow \rightarrow Y, X \rightarrow \rightarrow Z \Rightarrow X \rightarrow \rightarrow YZ$
DM4	Proiectivitatea	$X \rightarrow \rightarrow Y, X \rightarrow \rightarrow Z \Rightarrow X \rightarrow \rightarrow Y \cap Z, X \rightarrow \rightarrow Y \setminus Z$
DM5	Tranzitivitatea	$X \rightarrow \rightarrow Y, Y \rightarrow \rightarrow Z \Rightarrow X \rightarrow \rightarrow Z \setminus Y$
DM6	Pseudotranzitivitatea	$X \rightarrow \rightarrow Y, YW \rightarrow \rightarrow Z \Rightarrow XW \rightarrow \rightarrow Z \setminus YW$
DM7	Complementarea	$X \rightarrow \rightarrow Y \Rightarrow X \rightarrow \rightarrow R \setminus XY$

Fig.4.4. Reguli de inferență ale dependențelor multivaloare

În fig.4.4 sunt prezentate regulile de inferență ale dependențelor multivaloare.

După cum s-a observat din demonstrațiile validității regulilor de inferență, unele reguli se pot deduce din celelalte. Similar mulțimii de reguli {DF1, DF2, DF5}, pentru dependențele funcționale, există submulțimi de reguli pentru dependențele multivaloare, din care se deduc celelalte.

Teorema 4.4. Regulile DM3, DM4 și DM6 se deduc din regulile DM1, DM2, DM5 și DM7.

Demonstrație. Să arătăm validitatea regulii DM3 utilizând DM1, DM2, DM5, DM7, adică $\{X \rightarrow \rightarrow Y, Y \rightarrow \rightarrow Z\} \vdash X \rightarrow \rightarrow YZ$. Într-adevăr:

$$\begin{aligned}
 m_1 &:= X \rightarrow \rightarrow Z && \text{(dată),} \\
 m_2 &:= X \rightarrow \rightarrow XZ && \text{(DM2:} m_1 \text{),} \\
 m_3 &:= X \rightarrow \rightarrow Y && \text{(dată),} \\
 m_4 &:= XZ \rightarrow \rightarrow YZ && \text{(DM2:} m_3 \text{),} \\
 m_5 &:= XZ \rightarrow \rightarrow R \setminus XYZ && \text{(DM7:} m_4 \text{),} \\
 m_6 &:= X \rightarrow \rightarrow R \setminus XYZ && \text{(DM5:} m_2, m_5 \text{ și fiindcă } (R \setminus XYZ) \setminus XZ = R \setminus XYZ \text{)),} \\
 m_7 &:= X \rightarrow \rightarrow R \setminus X(R \setminus XYZ) && \text{(DM7:} m_6 \text{).}
 \end{aligned}$$

Din fig.4.3 se vede că $R \setminus X(R \setminus XYZ) = YZ$, deci $m_7 = X \rightarrow \rightarrow YZ$.

Să demonstrăm că regula DM4 urmează din DM1, DM2, DM5, DM7. Validitatea expresiei $\{X \rightarrow \rightarrow Y, X \rightarrow \rightarrow Z\} | - \{X \rightarrow \rightarrow Y \setminus Z, X \rightarrow \rightarrow Y \cap Z\}$ este confirmată de următoarea consecutivitate de inferență.

$$\begin{aligned} m_1 &:= X \rightarrow \rightarrow Y && \text{(dată),} \\ m_2 &:= X \rightarrow \rightarrow Z && \text{(dată),} \\ m_3 &:= X \rightarrow \rightarrow R \setminus XY && \text{(DM7: } m_1), \\ m_4 &:= X \rightarrow \rightarrow Z(R \setminus XY) && \text{(DM3: } m_2, m_3), \\ m_5 &:= X \rightarrow \rightarrow Y \setminus Z && \text{(DM7: } m_4 \text{ (vezi fig.4.3.)),} \\ m_6 &:= X \rightarrow \rightarrow (Y \setminus Z)(R \setminus XY) = \\ &\quad X \rightarrow \rightarrow R \setminus (X \cap Y) && \text{(DM3: } m_3, m_5 \text{ (vezi fig.4.3.)),} \\ m_7 &:= X \rightarrow \rightarrow X \cap Y && \text{(DM7: } m_6). \end{aligned}$$

Întrucât regula DM3 urmează din DM1, DM2, DM5 și DM7 substituirea dependențelor m_4 și m_6 cu consecutivități de inferență corespunzătoare se propune în calitate de exercițiu.

Și, în sfârșit, să arătăm că pseudotranzitivitatea, DM6, urmează din DM1, DM2, DM5, DM7. Pentru aceasta vom defini o consecutivitate de inferență pentru a arăta validitatea expresiei $\{X \rightarrow \rightarrow Y, YW \rightarrow \rightarrow Z\} | - XW \rightarrow \rightarrow Z \setminus YW$, aplicând doar regulile DM2 și DM5.

$$\begin{aligned} m_1 &:= X \rightarrow \rightarrow Y && \text{(dată),} \\ m_2 &:= XW \rightarrow \rightarrow YW && \text{(DM2: } m_1), \\ m_3 &:= YW \rightarrow \rightarrow Z && \text{(dată),} \\ m_4 &:= XW \rightarrow \rightarrow Z \setminus YW && \text{(DM5: } m_2, m_3). \end{aligned}$$

4.3. Reguli de inferență mixte

Fie M o mulțime de dependențe funcționale și multivaloare asupra schemei R și m o dependență funcțională sau multivaloare. Sunt reguli de inferență ce pot fi utilizate pentru a verifica dacă $M | = m$.

Fie relația $r(R)$ și $W, X, Y, Z \subseteq R$.

DFM1. Regula replicării. Dacă $X \rightarrow Y$, atunci $X \rightarrow \rightarrow Y$.

Validitatea acestei reguli este evidentă. Apelând la definiția 4.2 a dependenței multivaloare are loc egalitatea $\pi_Y(\sigma_{X=x}(r)) = \pi_Y(\sigma_{XZ=xz}(r))$, unde $Z = R \setminus XY$, fiindcă dependența funcțională presupune că pentru orice X -valori egale corespund Y -valori egale ale tuplurilor. Deci valorile pe care le primește atributul Z sunt imateriale.

Dependența funcțională reprezintă un tip particular de dependență multivaloare, pentru care mulțimea valorilor dependente este constituită dintr-o singură valoare, adică $\pi_Y(\sigma_{XZ=xz}(r))$ este o relație ce conține nu mai mult de un tuplu.

DFM2. Regula coalescenței. Dacă $X \rightarrow \rightarrow Y$ și $Z \rightarrow W$, unde $W \subseteq Y$ și $Y \cap Z = \emptyset$, atunci $X \rightarrow W$.

Demonstrație. Deoarece $X \rightarrow \rightarrow Y$ și dacă în r sunt două tupluri t_1, t_2 , pentru care $t_1[X] = t_2[X]$, atunci în r există un tuplu t_3 ce satisface egalitățile $t_3[X] = t_1[X]$, $t_3[Y] = t_1[Y]$ și $t_3[R \setminus XY] = t_2[R \setminus XY]$. Să observăm că, deoarece $Y \cap Z = \emptyset$, avem $Z \subseteq X(R \setminus XY)$ și întrucât $t_3[X] = t_1[X] = t_2[X]$, rezultă că $t_3[Z] = t_2[Z]$.

Conform dependenței funcționale $Z \rightarrow W$ avem $t_3[W] = t_2[W]$. Însă $W \subseteq Y$, de unde urmează că $t_3[W] = t_1[W] = t_2[W]$ și prin urmare dependența funcțională $X \rightarrow W$ e validă în r .

În fig.4.5 sunt prezentate regulile mixte de inferență.

Simbol	Denumire	Regulă
DFM1	Replicarea	$X \rightarrow Y \Rightarrow X \rightarrow \rightarrow Y$
DFM2		$X \rightarrow \rightarrow Y, Z \rightarrow W, W \subseteq Y, Y \cap Z = \emptyset, \Rightarrow X \rightarrow W$

Fig.4.5. Regulile mixte de inferență

Definiția 4.3. Fie relația $r(R)$ și $X, Y \subseteq R$. Dependența multivaloare $X \rightarrow \rightarrow Y$ se numește *trivială*, dacă $X \subseteq Y$ sau $XY = R$.

Astfel, regula de inferență DM1, generează numai dependențe multivaloare triviale.

Aici ne vom limita numai la formularea unor rezultate despre completitudinea regulilor de inferență fără a aduce demonstrațiile corespunzătoare.

Teorema 4.4. Regulile DM1-DM7 formează o mulțime completă de reguli de inferență a dependențelor multivaloare.

Teorema 4.5. Sistemul de reguli DF1, DF2, DF5, DM1, DM2, DM5, DM7, DFM1, DFM2 formează o mulțime închisă și completă de reguli de inferență a dependențelor funcționale și multivaloare.

4.4. Problema calității de membru

Fie M o mulțime de dependențe funcționale și multivaloare și m o dependență funcțională sau multivaloare. Deseori e necesar de determinat dacă urmează logic dependența m din M . Această problemă se numește problema calității de membru (membership problem). Bineînțeles că asemănător cu cazul numai dependențelor funcționale calcularea M^+ , adică a mulțimii tuturor dependențelor ce se deduc din M , este o procedură destul de laborioasă. Procesul necesită un timp care depinde exponențial de dimensiunile mulțimii M . Similar noțiunii de închidere a unei mulțimi de atribute în raport cu o mulțime de dependențe funcționale, pentru mulțimea M se introduce noțiunea de bază a dependențelor (dependency basis).

Definiția 4.4. Fie relația r cu schema R , o mulțime M de dependențe multivaloare și funcționale și $X \subseteq R$. *Baza dependențelor* a mulțimii de atribute X , notată cu $DEP(X)$, în raport cu M este o partiție a lui R : $DEP(X) = \{W_1, \dots, W_k | 1 \leq k \leq n, n = |R|\}$ ce satisface

$$(1) \quad X \rightarrow \rightarrow W_i \in M^+, 1 \leq i \leq k$$

și

$$(2) \quad X \rightarrow \rightarrow Y \in M^+, \text{ dacă și numai dacă } Y \text{ este uniunea a unor mulțimi } W_i \text{ din } DEP(X).$$

Să remarcăm că regula proiectivității pentru dependențele multivaloare este mai slabă decât omologul său pentru dependențele funcționale. Proiectivitatea pentru dependențele funcționale ne spune că, dacă dependența $X \rightarrow Y$ e validă într-o relație,

atunci e validă în această relație și dependența $X \rightarrow A$, pentru orice $A \in Y$. Regula proiectivității pentru dependențele multivaloare ne permite să spunem că, dacă $X \twoheadrightarrow Y$ e validă într-o relație, atunci dependența $X \twoheadrightarrow A$ e validă în aceeași relație numai dacă există în schema relației o mulțime de atribute Z ce satisface condițiile: dependența $X \twoheadrightarrow Z$ e validă și sau $Z \cap Y = A$, sau $Y \setminus Z = A$.

Cu toate acestea regulile aditivității (DM3) și proiectivității (DM4) ne permit să formulăm următoarea teoremă despre mulțimea de atribute Y , ca Y să depindă de o mulțime de atribute X , adică $X \twoheadrightarrow Y$. Această teoremă stă la temelia noțiunii de bază a dependențelor și a algoritmului de calculare a bazei dependențelor descris ceva mai jos.

Teorema 4.6. Fie relația $r(R)$ și $X \subseteq R$. Mulțimea de atribute $R \setminus X$ poate fi partiționată în submulțimi disjuncte W_1, \dots, W_k , încât pentru $Y \subseteq R \setminus X$ are loc $X \twoheadrightarrow Y$ atunci și numai atunci când Y este uniunea a unor mulțimi W_i , $1 \leq i \leq k$.

Demonstrație. La început fie că avem o singură submulțime constituită din atributele $R \setminus X$. Presupunem că la un oarecare pas de partiționare am obținut submulțimile W_1, \dots, W_m și e validă dependența $X \twoheadrightarrow W_i$, $1 \leq i \leq m$. Dacă $X \twoheadrightarrow Y$, dar Y nu este uniunea unor W_i , atunci substituim toate mulțimile W_i care satisfac expresiile $W_i \cap Y \neq \emptyset$ și $W_i \setminus Y \neq \emptyset$, cu mulțimile $W_i \cap Y$ și $W_i \setminus Y$, respectiv. Conform regulii proiectivității, dependențele $X \twoheadrightarrow W_i \cap Y$ și $X \twoheadrightarrow W_i \setminus Y$ sunt valide în $r(R)$. Fiindcă o mulțime finită de atribute nu poate fi partiționată la infinit, în cele din urmă, vom avea o partiție încât Y din dependența $X \twoheadrightarrow Y$, va fi uniunea unor submulțimi W_i . Conform regulii aditivității, X va determina uniunea oricărei submulțimi din partiție.

Remarcă. Dependențele multivaloare triviale $X \twoheadrightarrow Y$, unde $Y \subseteq X$, ce se obțin din regula reflexivității nu sunt considerate în teorema de mai sus. Din definiția bazei dependențelor mulțimii de atribute X urmează că $DEP(X)$ conține și toate atributele singulare A , unde $A \in X$.

Algoritmul DEPBASIS ($R, X, M, DEP(X)$)

Intrare: R – o schemă relațională;

X – o mulțime de atribute;

M – o mulțime de dependențe funcționale și multivaloare definite pe schema R .

Ieșire: $DEP(X)$ – baza dependențelor mulțimii X în raport cu M .

begin

```

0   DEP(X) := {A1, ..., An} ( unde A1...An=X);
1   W1 = R \ X;
2   k := 1;
3   for i=1 until k do
4       while  $\exists S \twoheadrightarrow T \in M$  &  $S \cap W_i = \emptyset$  &  $\emptyset \subset T \cap W_i \subset W_i$  do
           begin
5               k := k+1;
6               Wk := T ∩ Wi;
7               Wi := Wi \ Wk;
           end
8   DEP(X) := DEP(X) ∪ {W1, ..., Wk};
   return (DEP(X));
```

end

Algoritmul DEPBASIS construiește baza dependențelor pentru o mulțime dată de attribute X în raport cu o mulțime de dependențe M și este de complexitate polinomială.

Exemplul 4.4. Fie $R=ABCDEFGH$, $X=HI$ și

$M=\{m_1:HI\rightarrow A,$
 $m_2:AEFH\rightarrow\rightarrow ABF,$
 $m_3:CFI\rightarrow\rightarrow EH,$
 $m_4:AI\rightarrow\rightarrow BCDI,$
 $m_5:AH\rightarrow\rightarrow B\}.$ Să se calculeze $DEP(X)$.

La început, conform liniilor 0,1,2,3 ale algoritmului, sunt setate următoarele valori pentru $DEP(HI)$, W_1 , k și i :

$DEP(HI)=\{H,I\},$
 $W_1=ABCDEFG,$
 $k=1, i=1.$

Variabila k indică numărul de blocuri, iar i blocul curent. Conform liniei 4 a algoritmului, vom considera pe rând dependențele din M în privința satisfacerii condițiilor corespunzătoare. Fiindcă $i=1$, este selectat blocul de attribute W_1 . Pentru W_1 , dependența m_1 satisface condițiile liniei 4: $HI\cap W_1=\emptyset$ și $\emptyset\subset A\cap W_1\subset W_1$, unde HI și A sunt, corespunzător, determinantul și determinatul dependenței m_1 . Deci, conform liniilor 5-7 pentru k , W_2 și W_1 sunt setate următoarele valori:

$k=2,$
 $W_2=A,$
 $W_1=BCDEFG.$

Pentru blocul W_1 dependența m_4 e prima care satisface condițiile din linia 4 (dependența m_5 , de asemenea, satisface). Prin urmare, după executarea liniilor 5-7, k , W_3 și W_1 obțin valorile:

$k=3,$
 $W_3=BCD,$
 $W_1=EFG.$

Blocul W_2 nu e satisfăcut de nici o dependență. El va intra în $DEP(HI)$. Atunci, în linia 3 variabila i crește cu o unitate, adică $i=2$. Dar pentru W_2 nu sunt dependențe în M ce satisfac condițiile liniei 4 și atunci i crește cu o unitate, devenind 3. Pentru W_3 există dependența m_2 . Prin urmare,

$k=4,$
 $W_4=B,$
 $W_3=CD.$

Blocul W_3 nu e satisfăcut de nici o dependență și atunci i devine egal cu 4. Pentru blocul W_4 , de asemenea, nu există nici o dependență. Aici generarea blocurilor sfârșește, și în final vom obține:

$DEP(HI)=\{H,I,W_1,W_2,W_3,W_4\}=\{H, I, EFG, A, CD, B\}.$

Exemplul 4.5. Fie R și M din exemplul 4.4. Să se confirme sau să se infirme că:

- (a) $M|=HI\rightarrow\rightarrow AB,$
- (b) $M|=HI\rightarrow\rightarrow H,$
- (c) $M|=HI\rightarrow\rightarrow ABC.$

Pentru a verifica dacă $X\rightarrow\rightarrow Y$, se deduce din M , conform definiției bazei dependențelor, Y trebuie să fie uniunea unor mulțimi din $DEP(X)$. În cazul nostru, $DEP(HI)=\{H, I, EFG, A, CD, B\}$. Deci:

- (a) $M \models HI \rightarrow \rightarrow AB$,
- (b) $M \models HI \rightarrow \rightarrow H$,
- (c) $M \not\models HI \rightarrow \rightarrow ABC$.

4.5. Acoperiri nonredundante cu dependențe multivaloare

Ca și în cazul dependențelor funcționale, o mulțime de dependențe multivaloare este redundantă, dacă cel puțin una din dependențe este derivabilă din celelalte. Vom spune, de asemenea, că această dependență este redundantă în mulțimea dată. O acoperire nonredundantă a unei mulțimi de dependențe este o mulțime nonredundantă echivalentă. Problema construirii acoperirilor nonredundante pentru dependențele multivaloare se soluționează similar acoperirilor nonredundante ale dependențelor funcționale.

Fie M o mulțime de dependențe multivaloare. O acoperire nonredundantă a mulțimii M se construiește de următoarea procedură simplă. Se selectează o dependență din M . Dacă ea se deduce din celelalte dependențe multivaloare, atunci se elimină din M . Acest proces continuă până nu poate fi găsită nici o dependență redundantă. Fiindcă în acest proces fiecare dependență multivaloare se examinează o singură dată, complexitatea acestui proces este proporțională complexității construirii mulțimii $DEP(X)$ înmulțită la numărul de dependențe în M .

Un proces similar poate fi aplicat asupra unei mulțimi de dependențe funcționale și multivaloare. Dar trebuie menționat că în acest caz ordinea, în care dependențele sunt examinate, determină care dependențe vor rămâne în acoperirea nonredundantă. Intuitiv, se observă că pentru utilizatori (de asemenea și pentru SGBD) dependențele funcționale poartă mai multă informație decât dependențele multivaloare. De aceea e rezonabil pentru eliminare mai întâi de examinat dependențele multivaloare.

Fie F și M mulțimi de dependențe funcționale și multivaloare, respectiv. Fie F^1 o acoperire nonredundantă a tuturor dependențelor funcționale din $(F \cup M)^+$. Orice dependență funcțională din $(F \cup M)^+$ poate fi dedusă din F^1 cu algoritmul MEMBERSHIP. Apoi se elimină dependențele redundante multivaloare din $F^1 \cup M$ pentru a obține $F^1 \cup M^1$. Orice dependență multivaloare din $(F \cup M)^+ = (F^1 \cup M^1)^+$ poate fi dedusă din $F^1 \cup M^1$, utilizând numai algoritmul DEPBASIS.

Să menționăm că $F^1 \cup M^1$ nu este neapărat nonredundantă, fiindcă unele dependențe funcționale pot deveni redundante în F^1 , fiindcă dependențele multivaloare M^1 n-au fost considerate când se construia F^1 .

De asemenea F nu este neapărat o acoperire pentru dependențele funcționale din $(F \cup M)^+$. Regulile DFM1 și DFM2 pot deduce dependențe funcționale noi ce pot fi adăugate la F . Dar această metodă e destul de inefficientă.

Există metode mai eficiente de generare a astfel de acoperiri ale dependențelor funcționale fiind prezente și dependențe multivaloare. Dar discuțiile asupra algoritmilor eficienți depășește cadrul acestei lucrări.

4.6. Dependențe multivaloare incluse

Vom considera o generalizare a clasei dependențelor multivaloare care se numește dependențe multivaloare incluse (embedded multivalued dependencies).

Definiția 4.5. Fie relația r asupra mulțimii de atribute R și $R^1 \subseteq R$, $X, Y \subseteq R^1$. Dependența $X \twoheadrightarrow Y$ se numește multivaloare inclusă, dacă $X \twoheadrightarrow Y$ este dependență multivaloare în $\pi_{R^1}(r)$.

Este evident din definiție că orice dependență multivaloare este dependență multivaloare inclusă. Exemplul de mai jos arată că afirmația inversă e greșită.

Exemplul 4.6. Considerăm relația $r(ABCD)$ și proiecțiile ei $\pi_{ABC}(r)$ și $\pi_{ABD}(r)$ din fig.4.6. Proiecțiile $\pi_{ABC}(r)$ și $\pi_{ABD}(r)$ satisfac respectiv dependențele $A \twoheadrightarrow C$ și $A \twoheadrightarrow D$. Conform regulii complementării (DM7), ambele proiecții satisfac dependența $A \twoheadrightarrow B$. În același timp relația $r(ABCD)$ nu satisface dependența $A \twoheadrightarrow B$, deci și $A \twoheadrightarrow CD$ nu e validă în r .

r	A	B	C	D
	a	b	c	d
	a	b ₁	c ₁	d ₁
	a	b	c ₁	d
	a	b ₁	c	d ₁
	a	b	c	d ₁
	a	b ₁	c ₁	d

$\pi_{ABC}(r)$	A	B	C
	a	b	c
	a	b ₁	c ₁
	a	b	c ₁
	a	b ₁	c

$\pi_{ABD}(r)$	A	B	D
	a	b	d
	a	b ₁	d ₁
	a	b	d ₁
	a	b ₁	d

Fig.4.6.

Trebuie menționat că nu există o mulțime completă de reguli de inferență pentru dependențele multivaloare incluse.

4.7. Dependențe multivaloare noncontradictorii

Definiția 4.6. Fie o mulțime M de dependențe multivaloare definite pe schema R . Vom spune că dependența multivaloare $X \twoheadrightarrow Y$ din M *separă* două atribute A și B , dacă unul din atribute, fie A , este în Y , iar B este în $R \setminus XY$. Mulțimea M de dependențe multivaloare separă o mulțime de atribute V , dacă M separă două atribute din V .

Exemplul 4.7. Fie schema $R=ABCD$. Dependența $AB \twoheadrightarrow C$ separă attributele C și D . Similar, dependența multivaloare $CD \twoheadrightarrow A$ separă A și B . Dacă schema $R=ABCD$ e proiectată în baza dependenței $AB \twoheadrightarrow C$ în două subscheme ABC și ABD , atunci în ele sunt valide doar dependențele multivaloare $AB \twoheadrightarrow C$ și $AB \twoheadrightarrow D$, respectiv. Nici într-o schemă nu e validă dependența $CD \twoheadrightarrow A$.

Problema descrisă în exemplul de mai sus e cunoscută sub denumirea de problemă a separării determinanților.

Definiția 4.7. Fie determinanții X, Y a două dependențe multivaloare și fie $DEP(X)$, $DEP(Y)$. Determinanții X și Y sunt *noncontradictorii* (conflict free), dacă

$$DEP(X) \setminus \{A \mid A \in X\} = \{V_1, \dots, V_k, X_1, \dots, X_i, Z_x, Y_1, \dots, Y_j\},$$

$DEP(Y) \setminus \{B \mid B \in Y\} = \{V_1, \dots, V_k, Y_1, \dots, Y_j, Z_y, X_1, \dots, X_i\}$,
unde $\{V_1, \dots, V_k\} \subseteq DEP(X \cap Y)$ și $Z_x \cup X = Z_y \cup Y$. Vom spune că o mulțime M de dependențe multivaloare este noncontradictorie, dacă sunt noncontradictorii determinanții ai oricăror două dependențe din M .

Din definițiile 4.6 și 4.7, urmează că mulțimea M de dependențe multivaloare este noncontradictorie, dacă nici o dependență din M nu separă determinantul altei dependențe multivaloare din M .

Exemplul 4.8. Fie $R = ABCDEF$ și $M = \{B \twoheadrightarrow A, B \twoheadrightarrow C, B \twoheadrightarrow DEF, D \twoheadrightarrow ABC, D \twoheadrightarrow EF, E \twoheadrightarrow ABCD, E \twoheadrightarrow F\}$. Să se arate că mulțimea M de dependențe multivaloare e noncontradictorie.

Trebuie să arătăm că orice pereche din mulțimea de determinanți $\{B, D, E\}$ este noncontradictorie. Într-adevăr, $DEP(B) \setminus B = \{A, C, DEF\}$, unde $X = B, X_1 = A, X_2 = C, Z_x = D, Y_1 = E, Y_2 = F$; $DEP(D) \setminus D = \{EF, BAC\}$, unde $Y = D, Y_1 = E, Y_2 = F, Z_y = B, X_1 = A, X_2 = C$. Întrucât $X \cap Y = B \cap D = \emptyset$ și $DEP(\emptyset) = \emptyset$, atunci $DEP(B) \cap DEP(D) \subseteq DEP(B \cap D)$. Este satisfăcută și condiția $Z_x \cup X = Z_y \cup Y$, fiindcă $Z_x \cup X = \{D, B\}$ și $Z_y \cup Y = \{B, D\}$. Prin urmare, B și D sunt determinanți noncontradictorii. Similar poate fi arătat că sunt noncontradictorii și perechile B, E și D, E . Verificarea acestor din urmă se lasă în calitate de exercițiu.

Exemplul 4.9. Fie $R = ABCD$ și $M = \{AB \twoheadrightarrow C, AB \twoheadrightarrow D, CD \twoheadrightarrow A, CD \twoheadrightarrow B\}$. Atunci $DEP(AB) \setminus \{A, B\} = \{C, D\}$, $DEP(CD) \setminus \{C, D\} = \{A, B\}$. Definiția 4.7 nu e satisfăcută și, prin urmare, mulțimea M de dependențe multivaloare e contradictorie.

Unii cercetători afirmă că, dacă o mulțime de dependențe multivaloare reflectă o parte a lumii reale, atunci mulțimea neapărat este noncontradictorie. Dar dacă mulțimea specificată nu e noncontradictorie, atunci înseamnă că o parte de semantică a lumii reale nu a fost capturată.

4.8. Dependențe joncțiune

Definiția 4.8. Fie U mulțimea universală de attribute și fie relațiile r_1, \dots, r_m definite pe schemele R_1, \dots, R_m , respectiv, unde $R_i \subseteq U$, $1 \leq i \leq m$. Joncțiunea relațiilor r_1, \dots, r_m , notată cu $|x|(r_1, \dots, r_m)$, este o relație definită pe schema $U^1 = R_1 \dots R_m \subseteq U$:

$$|x|(r_1, \dots, r_m) = \{t \mid t[R_i] = t_i \text{ \& } t_i \in r_i(R_i), 1 \leq i \leq m\}.$$

De noțiunea joncțiune $|x|(r_1, \dots, r_m)$ a relațiilor r_1, \dots, r_m e strâns legată noțiunea de dependență joncțiune asupra U^1 , care este o constrângere asupra U^1 de forma $|x|(R_1, \dots, R_m)$. Vom spune că dependența joncțiune este inclusă dacă $U^1 \subseteq U$. Dacă $U^1 = U$, vom spune simplu dependența joncțiune.

Definiția 4.9. Vom spune că relația $r(U)$ satisface dependența joncțiune $|x|(R_1, \dots, R_m)$ sau dependența joncțiune $|x|(R_1 \dots R_m)$ e validă în $r(U)$, dacă $r(U)$ se descompune fără pierderi pe schemele R_1, \dots, R_m , adică

$$r(U) = |x|(\pi_{R_1}(r), \dots, \pi_{R_m}(r)) \quad (4.3)$$

Exemplul 4.10. Relația $r(ABC)$ satisface (vezi relația și proiecțiile corespunzătoare în fig.4.7) dependența joncțiune $|x|(AB, AC, BC)$, fiindcă $r(ABC) = |x|(\pi_{AB}(r), \pi_{AC}(r), \pi_{BC}(r))$.

O condiție necesară, ca egalitatea (4.3) să fie satisfăcută, este $U=R_1...R_m$.

Este evident că dependența de joncțiune inclusă este o generalizare a dependenței joncțiune. La rândul său, apelând la teorema 4.3, despre condiția necesară și suficientă ca o relație să se descompună fără pierderi în două proiecții, conchidem că dependența joncțiune este o generalizare a dependenței multivaloare. Într-adevăr, teorema 4.3 ne spune că $r(R)$ satisface dependența multivaloare $X \twoheadrightarrow Y$, atunci și numai atunci când r se descompune fără pierderi pe schemele XY și XZ , unde $Z = R \setminus XY$. Condiția coincide cu definiția dependenței joncțiune $|x|(XY, XZ)$.

r	A	B	C
	a ₁	b ₁	c ₁
	a ₁	b ₂	c ₂
	a ₃	b ₃	c ₃
	a ₄	b ₃	c ₄
	a ₅	b ₅	c ₅
	a ₆	b ₆	c ₅

$\pi_{AB}(r)$	A	B
	a ₁	b ₁
	a ₁	b ₂
	a ₃	b ₃
	a ₄	b ₃
	a ₅	b ₅
	a ₆	b ₆

$\pi_{AC}(r)$	A	C
	a ₁	c ₁
	a ₁	c ₂
	a ₃	c ₃
	a ₄	c ₄
	a ₅	c ₅
	a ₆	c ₅

$\pi_{BC}(r)$	B	C
	b ₁	c ₁
	b ₂	c ₂
	b ₃	c ₃
	b ₃	c ₄
	b ₅	c ₅
	b ₆	c ₅

Fig.4.7.

4.9. Tablouri

Vom descrie o metodă tabelară de testare a dependenței joncțiune, bazată pe noțiunea de tablou. Tabloul, de fapt este o relație, cu numai o singură deosebire – în loc de valori tuplurile conțin variabile. Variabilele sunt luate din două mulțimi: mulțimea variabilelor distincte și mulțimea variabilelor nondistincte. Variabilele distincte sunt formate din litera a cu indice, iar cele nondistincte din litera b cu doi indici. Mulțimea de atribute ce denumesc coloanele este schema tabloului. O variabilă corespunde unei singure coloane. O coloană nu poate avea mai mult de o variabilă distinctă.

Definiția 4.10. Fie o dependență joncțiune $|x|(R_1,...,R_m)$. *Tablou* al dependenței $|x|(R_1,...,R_m)$ este o relație ce are un nume (fie tab) și $|R_1...R_m| = |U|$ coloane, notate cu atributele din U , și m tupluri câte unul pentru fiecare schemă R_i , $1 \leq i \leq m$. Tuplul t_i conține în coloana A_j variabila distinctă a_j , dacă A_j aparține lui R_i . Celelalte componente ale tuplului t_i sunt variabile nondistincte b_{ij} ce nu se repetă în alt tuplu. Deci $t_i[A_j] = a_j$, dacă $A_j \in R_i$ și $t_i[A_j] = b_{ij}$, dacă $A_j \notin R_i$.

tab	A	B	C	D
	a ₁	a ₂	b ₁₃	b ₁₄
	b ₂₁	a ₂	a ₃	b ₂₄
	a ₁	b ₃₂	a ₃	a ₄

Fig.4.8. Tabloul dependenței $|x|(AB, BC, ACD)$

Exemplul 4.11. Fie dependența joncțiune $|x|(AB, BC, ACD)$. Tabloul acestei dependențe arată ca în fig.4.8.

Tuplurile într-un tablou pot fi transformate conform unor reguli ce corespund anumitor clase de dependențe: funcționale și joncțiune. Scopul unor astfel de transformări constă în obținerea unui tuplu alcătuit numai din variabile distincte. Tuplul format exclusiv din variabile distincte se numește *tuplu-scop*. Dacă în urma transformărilor tabloul conține tuplul scop, atunci dependența joncțiune e validă, adică joncțiunea este fără pierderi. Sunt cunoscute două tipuri de reguli de transformare a tabloului: F-reguli și J-reguli.

F-reguli. Fie tab un tablou a unei dependențe joncțiune și J o mulțime de dependențe joncțiune, multivaloare și funcționale. Pentru orice dependență funcțională $X \rightarrow Y$ din J modificarea tabloului tab se produce în felul următor. În tab se caută tuplurile ce coincid pe atributele din X. Fiind descoperite astfel de tupluri, se egalează variabilele atributelor din Y. Fie $t_i[X] = t_j[X]$ și pentru $A \in Y$ $t_i[A] = v_1$, $t_j[A] = v_2$. Atunci,

- dacă una din variabilele v_1 și v_2 este distinctă, atunci variabila nondistinctă e substituită de cea distinctă;
- dacă ambele variabile sunt nondistincte, atunci variabila cu indice mai mare e substituită de variabila cu indice mai mic.

J-reguli. Fie tab un tablou determinat de dependența joncțiune $|x|(R_1, \dots, R_m)$ și fie J o mulțime de dependențe joncțiune, multivaloare și funcționale. Pentru orice dependență joncțiune $|x|(S_1, \dots, S_k)$ din J la tabloul tab se adaugă tuplul t (dacă el nu este deja un tab) dacă $t \in |x|(\pi_{S_1}(\text{tab}), \dots, \pi_{S_k}(\text{tab}))$.

Trebuie de menționat că S_1, \dots, S_k trebuie să formeze mulțimea universală U de atribute, adică $R_1, \dots, R_m = S_1, \dots, S_k$. Considerăm un raționament de executare eficientă a joncțiunii. Nu toate tuplurile ce vor participa la joncțiune sunt esențiale în formarea tuplului t ce se adaugă la tab. Dacă tuplurile t_i și t_j sunt în tab și dacă tuplul t_j are componente distincte pentru toate atributele pentru care are variabile distincte tuplul t_i , atunci t_j nu trebuie să participe la joncțiune.

Regulile de transformare a tabloului legate de dependențele multivaloare sunt de prisos, întrucât dependența multivaloare este un caz particular al dependenței joncțiune: dependența multivaloare $X \twoheadrightarrow Y$ este dependența joncțiune $|x|(XY, XZ)$, unde $Z = R \setminus XY$.

Să aducem următorul algoritm de testare a dependențelor joncțiune.

Algoritmul LOSSLESSTEST (J, j)

Intrare: J - o mulțime de dependențe joncțiune, multivaloare și funcționale;

$|x|(R_1, \dots, R_m)$ - o dependență joncțiune.

Ieșire: Adevăr, dacă dependența joncțiune e validă (sau joncțiunea e fără pierderi) și fals, în caz contrar.

- (1) Se definește tabloul dependenței joncțiune $|x|(R_1, \dots, R_m)$.
- (2) Se aplică F-regulile și J-regulile asupra tabloului până nu mai poate fi schimbat.
- (3) După substituirile produse de toate dependențele din J, se verifică dacă tabloul conține un tuplu ce are toate componentele distincte. Dacă există în tablou tuplul-scop, atunci return (adevăr) în caz contrar - return(fals).

Exemplul 4.12. Fie relația $r(ABCDEF)$ și $J=\{A \rightarrow B, F \rightarrow E\}$. Să se verifice dacă dependența joncțiune $|x|(ABDE, ACDF, BCEF)$ este validă în relația $r(ABCDEF)$.

Aplicând algoritmul de mai sus, pasul (1) produce tabloul din fig.4.9(a). Acest tabel are trei tupluri și șase coloane.

	A	B	C	D	E	F
t_1	a_1	a_2	b_{13}	a_4	a_5	b_{16}
t_2	a_1	b_{22}	a_3	a_4	b_{25}	a_6
t_3	b_{31}	a_2	a_3	b_{34}	a_5	a_6

Fig.4.9(a). Tabloul inițial al dependenței joncțiune $|x|(ABDE, ACDF, BCEF)$

Urmând F-regulile în aplicarea dependenței $A \rightarrow B$ din J (pasul (2)), obținem tabloul modificat din figura 4.9(b). În tablou tuplul $t_2[B] = a_2$, fiindcă $t_1[A] = t_2[A]$ în tabloul inițial și b_{22} a fost substituit de a_2 .

	A	B	C	D	E	F
t_1	a_1	a_2	b_{13}	a_4	a_5	b_{16}
t_2	a_1	a_2	a_3	a_4	b_{25}	a_6
t_3	b_{31}	a_2	a_3	b_{34}	a_5	a_6

Fig.4.9(b). Tabloul modificat de $A \rightarrow B$

Aplicând apoi dependența funcțională $F \rightarrow E$ în pasul (2), obținem tabloul modificat din fig.4.9(c). Aici $t_2[E] = a_5$, întrucât variabila b_{25} din tabloul precedent este substituită de a_5 . Examinând tabloul obținut, observăm că $t_2 = \langle a_1, a_2, a_3, a_4, a_5, a_6 \rangle$ este tuplul-scop. Deci relația $r(ABCDEF)$ satisface dependența joncțiune $|x|(ABDE, ACDF, BCEF)$ sau joncțiunea $|x|(\pi_{ABDE}(r), \pi_{ACDF}(r), \pi_{BCEF}(r))$ este fără pierderi.

	A	B	C	D	E	F
t_1	a_1	a_2	b_{13}	a_4	a_5	b_{16}
t_2	a_1	a_2	a_3	a_4	a_5	a_6
t_3	b_{31}	a_2	a_3	b_{34}	a_5	a_6

Fig.4.9(c). Tabloul modificat de $F \rightarrow E$

Exemplul 4.13. Fie relația $r(ABCDE)$ și mulțimea de dependențe joncțiune $J = \{ |x|(AC, ABD, BDE), |x|(ABD, CDE) \}$ validă în r . Să se arate că relația $r(ABCDE)$ satisface și dependența joncțiune $|x|(AC, ABD, DE)$.

Construim tabloul inițial tab pentru dependența joncțiune $|x|(AC, ABD, DE)$ din fig. 4.10(a) ce constă din trei tupluri t_1 , t_2 și t_3 , determinate respectiv de mulțimile AC , ABD și DE .

tab	A	B	C	D	E
t_1	a_1	b_{12}	a_3	b_{14}	b_{15}
t_2	a_1	a_2	b_{23}	a_4	b_{25}
t_3	b_{31}	b_{32}	b_{33}	a_4	a_5

Fig.4.10(a) Tabloul inițial tab

Aplicăm dependența joncțiune $|x|(ABD, CDE)$ din J asupra tabloului tab . Proiectăm tabloul tab asupra schemelor ABD și CDE . Obținem proiecțiile $\pi_{ABD}(tab)$ și $\pi_{CDE}(tab)$ din fig.4.10(b) și fig.4.10(c), respectiv.

$\pi_{ABD}(tab)$	A	B	D
	a_1	b_{12}	b_{14}
	a_1	a_2	a_4
	b_{31}	b_{32}	a_4

Fig.4.10(b). $\pi_{ABD}(tab)$

$\pi_{CDE}(tab)$	C	D	E
	a_3	b_{14}	b_{15}
	b_{23}	a_4	b_{25}
	b_{33}	a_4	a_5

Fig.4.10(c). $\pi_{CDE}(tab)$

În relația $\pi_{ABD}(tab)$ tuplurile $\langle a_1 \ b_{12} \ b_{14} \rangle$ și $\langle b_{31} \ b_{32} \ a_4 \rangle$ nu sunt esențiale, fiindcă tuplul $\langle a_1 \ a_2 \ a_4 \rangle$ le recapitulează. În relația $\pi_{CDE}(tab)$ tuplurile reprezentative sunt $\langle a_1 \ b_{14} \ b_{15} \rangle$ și $\langle b_{33} \ a_4 \ a_5 \rangle$. Tuplul $\langle b_{23} \ a_4 \ b_{25} \rangle$ nu va participa la joncțiune, fiindcă se substituie de tuplul $\langle b_{33} \ a_4 \ a_5 \rangle$. Joncționând tuplurile reprezentative din relațiile $\pi_{ABD}(tab)$ și $\pi_{CDE}(tab)$, obținem tuplul $t_4 = \langle a_1 \ a_2 \ b_{33} \ a_4 \ a_5 \rangle$. Formăm tabloul $tab_1 = tab \cup \{t_4\}$ din fig.4.10(d).

tab_1	A	B	C	D	E
---------	---	---	---	---	---

t ₁	a ₁	b ₁₂	a ₃	b ₁₄	b ₁₅
t ₂	a ₁	a ₂	b ₂₃	a ₄	b ₂₅
t ₃	b ₃₁	b ₃₂	b ₃₃	a ₄	a ₅
t ₄	a ₁	a ₂	b ₃₃	a ₄	a ₅

Fig.4.10(d) Tabloul $tab_1 = tab \cup \{t_4\}$

Să examinăm acum acțiunea asupra tabloului tab_1 a dependenței $|x|(AC, ABD, BDE)$ din J, proiectându-l pe schemele AC, ABD și BDE (vezi fig.4.10(e), fig.4.10(f), fig.4.10(g), respectiv)

$\pi_{AC}(tab_1)$	A	C
	a ₁	a ₃
	a ₁	b ₂₃
	b ₃₁	b ₃₃
	a ₁	b ₃₃

Fig.4.10(e). $\pi_{AC}(tab_1)$

$\pi_{ABD}(tab_1)$	A	B	D
	a ₁	b ₁₂	b ₁₄
	a ₁	a ₂	a ₄
	b ₃₁	b ₃₂	a ₄

Fig.4.10(f). $\pi_{ABD}(tab_1)$

$\pi_{BDE}(tab_1)$	B	D	E
	b ₁₂	b ₁₄	b ₁₅
	a ₂	a ₄	b ₂₅
	b ₃₂	a ₄	a ₅
	a ₂	a ₄	a ₅

Fig.4.10(g). $\pi_{BDE}(tab_1)$

Să observăm tuplurile ce vor participa la joncțiune. În $\pi_{AC}(tab_1)$ este tuplul $\langle a_1 a_3 \rangle$, în $\pi_{ABD}(tab_1)$ e tuplul $\langle a_1 a_2 a_4 \rangle$ și în $\pi_{BDE}(tab_1)$ e tuplul $\langle a_2 a_4 a_5 \rangle$. În urma joncțiunii acestor tupluri (câte unul din fiecare proiecție) obținem tuplul $t_5 = \langle a_1 a_2 a_3 a_4 a_5 \rangle$. Se construiește tabloul $tab_2 = tab_1 \cup \{t_5\}$ prezentat în fig.4.10(h).

tab ₂	A	B	C	D	E
t ₁	a ₁	b ₁₂	a ₃	b ₁₄	b ₁₅
t ₂	a ₁	a ₂	b ₂₃	a ₄	b ₂₅
t ₃	b ₃₁	b ₃₂	b ₃₃	a ₄	a ₅
t ₄	a ₁	a ₂	b ₃₃	a ₄	a ₅
t ₅	a ₁	a ₂	a ₃	a ₄	a ₅

Fig.4.10(h). Tabloul $tab_2 = tab_1 \cup \{t_5\}$

Tabloul tab_2 conține tuplul-scop, t_5 . Deci dependența joncțiune $|x|(AC, ABD, DE)$ este validă în relația $r(ABCDE)$ sau, ceea ce e echivalent, joncțiunea $|x|(\pi_{AC}(r), \pi_{ABD}(r), \pi_{DE}(r))$ este fără pierderi.

Deci tablourile pot fi utilizate pentru soluționarea problemei calității de membru pentru dependențele joncțiune. Adică, dacă J e o mulțime de dependențe joncțiune, multivaloare și funcționale și j e o dependență joncțiune, atunci cu ajutorul tablourilor putem determina dacă j urmează logic din J . Această metodă este aplicabilă, numai dacă dependențele joncțiune din J sunt definite pe toată mulțimea universală de attribute U

Pentru mulțimea de dependențe joncțiune nu există o mulțime completă de reguli de inferență.

Definiția 4.11. Dependența joncțiune $|x|(R_1, \dots, R_m)$ asupra $U = R_1 \dots R_m$ este *trivială*, dacă e validă în orice relație r cu schema U .

4.10. Reguli de inferență ale dependențelor joncțiune

Considerăm următoarea mulțime de reguli de inferență ale dependențelor joncțiune. Este clar că mulțimea este închisă, dar e puțin probabil să fie și completă.

DJ1. Dacă $R \subseteq U$, atunci $|x|(R)$.

Această regulă ne spune că orice relație $r(R)$ satisface dependența joncțiune $|x|(R)$, fiindcă $r(R) = |x|(\pi_R(r))$.

DJ2. Dacă $|x|(R_1, \dots, R_m)$ și $Y \subseteq R_1 \dots R_m$, atunci $|x|(R_1, \dots, R_m, Y)$.

Validitatea acestei reguli reiese din egalitatea $|x|(|x|(R_1, \dots, R_m)Y) = |x|(R_1, \dots, R_m, Y)$.

r	A	B	C
	a ₁	b ₁	c ₁
	a ₁	b ₂	c ₂
	a ₂	b ₁	c ₃

Fig. 4.11.

Exemplul 4.14. Fie dependența joncțiune $|x|(AC, BC)$ și fie $Y = AB$. Atunci $|x|(AC, BC) = |x|(AC, BC, AB)$, adică relația $r(ABC)$ ce satisface dependența (vezi fig.4.11) $|x|(AC, BC)$ satisface și dependența $|x|(AC, BC, AB)$. Aceste două dependențe sunt definite pe mulțimea universală de attribute, deci ele pot fi testate prin intermediul tabloului.

DJ3. Fie $|x|(R_1, \dots, R_m)$ și $Y, Z \subseteq U$. Atunci $|x|(R_1, \dots, R_m, Y, Z)$ implică $|x|(R_1, \dots, R_m, YZ)$.

Exemplul 4.15. Fie în relația $r(ABCDE)$ este validă dependența joncțiune $|x|(AC,DE)$. Atunci $|x|(AC,DE,ABD,BD) |= |x|(AC,DE,ABD)$. Întrucât ambele dependențe antrenează toate atributele, deducția poate fi verificată cu ajutorul tabloului.

DJ4. Fie $|x|(R_1, \dots, R_m)$, $|x|(S_1, \dots, S_k)$ și $Y = S_1 \dots S_k$. Atunci $|x|(R_1, \dots, R_m, Y)$ și $|x|(S_1, \dots, S_k)$ implică $|x|(R_1, \dots, R_m, S_1, \dots, S_k)$.

Exemplul 4.16. Fie $|x|(AC,ABD)$, $|x|(BD,DE)$ și $Y=BDE$. Atunci $\{|x|(AC,ABD,BDE), |x|(BD,DE)\} |= |x|(AC,ABD,BD,DE)$. Aici tabloul nu poate fi utilizat pentru verificarea implicației, fiindcă dependența joncțiune $|x|(BD,DE)$ este inclusă, adică nu e definită pe mulțimea universală.

DJ5. Fie $|x|(R_1, \dots, R_m)$, $A \notin R_1 \dots R_m$ și $Y \subseteq U$. Dacă $|x|(R_1, \dots, R_m, YA)$ atunci $|x|(R_1, \dots, R_m, Y)$.

Exemplul 4.17. Fie $|x|(BCD)$, $Y = DE$ și $A \notin BCD$. Atunci $|x|(BCD,DEA) |= |x|(BCD,DE)$. Întrucât dependența $|x|(BCD,DE)$ e inclusă, tabloul nu poate fi utilizat pentru verificarea acestei reguli.

Regulile DJ4 și DJ5 conțin dependențe joncțiune incluse. Vom combina aceste reguli pentru obținerea unei reguli DJ6 echivalente, dar care antrenează numai dependențe joncțiune definite pe mulțimea universală.

DJ6. Dacă $|x|(R_1, \dots, R_m, Y)$, $|x|(S_1, \dots, S_k)$ și $\{A | A \text{ aparține cel puțin la două scheme } S_i, S_j\} \subseteq Y$, atunci $|x|(R_1, \dots, R_m, S_1 \cap Y, \dots, S_k \cap Y)$.

Exemplul 4.18. $\{|x|(AC,AB,BDE) |x|(ABD,CDE)\} |= \{|x|(AC,ABD,BD,DE)$. Această inferență poate fi verificată, utilizând tabloul.

Pentru dependențele joncțiune nu este găsită o mulțime completă de reguli de inferență. Probabil că nici nu există.

4.11. Exerciții

4.1. Fie relația $r(ABC)$ din fig.4.12.

- Să se arate că relația $r(ABC)$ satisface dependența multivaloare $A \twoheadrightarrow B$.
- Să se construiască din $r(ABC)$ patru relații diferite, fiecare conținând câte trei tupluri și să se arate că dependența $A \twoheadrightarrow B$ nu e validă în nici o relație.
- Să se construiască din $r(ABC)$ șase relații diferite a câte două tupluri. Să se arate că patru din ele satisfac dependența multivaloare $A \twoheadrightarrow B$, iar două nu o satisfac.

r	A	B	C
	a	b ₁	c ₁
	a	b ₂	c ₂

a	b ₁	c ₂
a	b ₂	c ₁

Fig.4.12.

- 4.2. Să se infirme că, dacă $Z \subseteq W$ și $X \rightarrow \rightarrow Y$, atunci $XW \rightarrow YZ$.
- 4.3. Să se infirme că, dacă $X \rightarrow \rightarrow Y$, atunci $X \rightarrow Y$.
- 4.4. Fie relația $r(ABCDEFGH IJ)$ și $M = \{AB \rightarrow \rightarrow DEFG, CGJ \rightarrow \rightarrow ADHI\}$. Să se calculeze $DEP(ACGJ)$.
- 4.5. Fie relația $r(ABC)$ și $J = \{AB \rightarrow C, C \rightarrow A\}$. Să se arate că relația $r(ABC)$ nu satisface dependența joncțiune $|x|(AB, AC)$.
- 4.6. Fie $U = ABCDEFGH$ și $J = \{B \rightarrow E, C \rightarrow E, EF \rightarrow G, G \rightarrow ABH\}$. Să se arate că schema bazei de date $Db = \{ABFG, BC, CDFH, AEH\}$ se bucură de proprietatea joncțiunii fără pierderi.
- 4.7. Să se arate că dependența joncțiune $|x|(R_1, \dots, R_m)$ asupra $U = R_1 \dots R_m$ este trivială, dacă și numai dacă $R_i = U$ pentru careva $i, 1 \leq i \leq m$.
- 4.8. Să se completeze cu tupluri relația $r(ABCDE)$ din fig.4.12, ca să satisfacă dependențele multivaloare $A \rightarrow \rightarrow BC$ și $CD \rightarrow \rightarrow BE$

r	A	B	C	D	E
	a ₁	b ₁	C ₁	d ₁	e ₁
	a ₁	b ₂	C ₁	d ₂	e ₁
	a ₂	b ₁	C ₁	d ₁	e ₂

Fig.4.12.

- 4.9. Să se descrie clasa de dependențe multivaloare ce pot fi deduse din dependența funcțională $X \rightarrow Y$.
- 4.10. Să se găsească $DEP(AC)$ pentru mulțimea de dependențe multivaloare definite pe schema $R = ABCDEI$.
- 4.11. Să se arate că o relație $r(R)$ nu poate fi descompusă fără pierderi în două relații cu schemele R_1 și R_2 , unde $R_1 \neq R$ și $R_2 \neq R$, atunci și numai atunci când în r sunt valide numai dependențe multivaloare triviale.
- 4.12. Fie relația $r(ABCDE)$ și fie $F = \{A \rightarrow C, B \rightarrow C, C \rightarrow D, DE \rightarrow C, CE \rightarrow A\}$ o mulțime de dependențe funcționale valide în r . Să se arate că în r e validă și dependența joncțiune $|x|(AD, AB, BE, CDE, AE)$.
- 4.13. Fie $R = ABCDE$ și $M = \{E \rightarrow \rightarrow B, AE \rightarrow \rightarrow C\}$. Să se arate că mulțimea M de dependențe multivaloare este noncontradictorie.

PROIECTAREA BAZELOR DE DATE

Prin proiectarea bazei de date, aici se subînțelege proiectarea unei scheme logice care ar înlătura apariția unor anomalii în lucrul cu baza de date, asigurând totodată facilități și performanțe sporite la exploatarea ei.

Anomaliile care apar în lucrul cu baza de date sunt cunoscute sub anomaliile de actualizare a datelor. Ele sunt puse în legătură cu dependențele care se manifestă între atribute. O asemenea abordare a anomaliilor de actualizare permite caracterizarea riguroasă a gradului de perfecțiune a schemei bazei de date și face posibilă definirea unor tehnici formale de proiectare a unor astfel de scheme.

Prelucrarea datelor o perioadă de timp, cum se întâmplă în bazele de date, poate provoca o serie de probleme personalului responsabil de menținerea integrității datelor. Anomaliile în date cum ar fi datele duplicate sau pierderile de informații pot apărea, dacă datele nu sunt organizate într-un mod rezonabil. În ce constă o organizare rezonabilă a datelor? Cercetările la zi și experiența acumulată în domeniul proiectării bazelor de date au arătat că unele aranjări de date lucrează mai bine decât altele. S-au elaborat tehnici de analiză a datelor și organizare a lor într-o structură flexibilă și stabilă.

Procesul de normalizare constă în aplicarea unui set de reguli predefinite asupra unei aranjări a datelor cu scopul reducerii structurii complexe și transformării lor în structuri mai mici și stabile ce vor facilita manipularea și menținerea datelor.

La fiecare pas o regulă este aplicată, datele pot fi restructurate și când regula este satisfăcută se spune că datele sunt într-o formă normală.

Deci normalizarea este o abordare formală de analiză și grupare a datelor în structuri mai eficiente ce se pot acomoda viitoarelor actualizări. În afară de aceasta normalizarea minimizează impactul ce poate avea loc asupra aplicațiilor în procesul actualizării bazei de date.

Pentru a produce o bază de date bine proiectată de obicei se pornește de la relații nenormalizate și printr-o serie de pași se descompun structurile de date pentru a obține schema finală a bazei de date.

5.1. Prezumția schemei universale

Materia expusă până acum (și mai departe) presupune că toată mulțimea de atribute formează schema unei relații “mari” și toate constrângerile asupra atributelor sunt constrângeri ale acestei scheme.

Unul din scopurile, pe care și le propune să le atingă modelul relațional este eliberarea utilizatorului de a specifica căile de acces la date. Această problemă e cunoscută sub denumirea de problema navigației logice. Însă, dacă baza de date constă din mai multe relații independența navigației logice nu este asigurată.

De exemplu, fie baza de date are două relații *angajați*(FUNȚIONAR DEPARTAMENT) și *departamente*(DEPARTAMENT MANAGER). Pentru a obține asocierile FUNȚIONAR MANAGER se jonctionează relațiile *angajați* și *departamente* și apoi relația obținută se proiectează pe atributele FUNȚIONAR și

MANAGER. Dar indicarea operațiilor și este specificarea căilor de acces. Dacă baza de date se restructurează, reprezentându-se printr-o singură relație, atunci trebuie să se modifice corespunzător și programele ce specifică joncțiunea.

Formulând o interpelare (cerere) la baza de date ce se referă la mai multe relații din baza de date e comod a interpreta lumea reală ca o singură relație, schema căreia include toate atributele din schemele relațiilor bazei. Această relație se numește relație universală, iar schema ei – schema relației universale sau schema universală.

Modelul relației universale realizează complet independența navigației logice, excluzând astfel definirea unor căi de acces neoptimale din partea unor utilizatori neinițiați. Deci acest model facilitează interacțiunea sistem-utilizator, cerând de la ultimul doar cunoașterea atributelor și semanticii.

Cea mai strictă formă de realizare a relației universale constă în construirea propriu-zisă a bazei de date dintr-o singură relație pe mulțimea de atribute universală U . Dar aici apar multe dezavantaje. În primul rând, nu toate tuplurile vor avea valori definite. În al doilea rând, păstrarea tuturor datelor într-o singură relație inevitabil va genera o serie de anomalii de actualizare.

De aceea, realmente, baza de date constă dintr-o mulțime de relații normalizate definite pe submulțimi de atribute ale schemei relației imaginare universale. Însă în acest caz baza de date trebuie să satisfacă unele condiții.

Una din condiții presupune că baza de date trebuie să posede proprietatea joncțiunii fără pierderi. Această problemă a fost abordată în capitolul precedent.

A doua – că descompunerea relației universale imaginare conservă dependențele. Această cerință va fi considerată în acest capitol.

A treia condiție presupune ca atributele în schema universală joacă un singur “rol”. Astfel ambiguitățile sunt excluse. Dacă numele exprimă diverse noțiuni problema se soluționează prin renumirea atributelor sau divizarea unui atribut în mai multe.

5.2. Descompunerea relațiilor cu conservarea dependențelor

S-a constatat că e binevenit ca descompunerea unei relații (inclusiv celei universale) să posede proprietatea joncțiunii fără pierderi. Aceasta este garanția că relația poate fi refăcută din proiecțiile sale.

O altă importantă proprietate a descompunerii unei relații $r(R)$ pe mulțimea de scheme R_1, \dots, R_m , unde $R = R_1 \dots R_m$, constă că mulțimea de dependențe valide în $r(R)$ să se deducă din dependențele valide în proiecțiile $\pi_{R_1}(r)$, ..., $\pi_{R_m}(r)$.

Definiția 5.1. Fie F o mulțime de dependențe funcționale asupra schemei R . Proiecția mulțimii F asupra unei mulțimi de atribute Z , notată $\pi_Z(F)$, este mulțimea de dependențe $X \rightarrow Y$ din F^+ și $XY \subseteq Z$.

Să observăm că dependența $X \rightarrow Y$ nu neapărat trebuie să aparțină mulțimii F . E de ajuns ca ea să aparțină închiderii F^+ , adică $F \models X \rightarrow Y$.

Definiția 5.2. Fie relația $r(R)$ este descompusă pe mulțimea de scheme R_1, \dots, R_m , unde $R = R_1 \dots R_m$, și fie o mulțime F de dependențe asupra R . Vom spune că descompunerea relației $r(R)$ asupra R_1, \dots, R_m conservă dependențele F , dacă $\{\pi_{R_1}(F) \cup \dots \cup \pi_{R_m}(F)\} \models F$.

Tendința de conservare a dependențelor e firească. Dependențele sunt constrângeri de integritate asupra relațiilor cu schema dată. Dacă din dependențele proiectate nu ar urma mulțimea F , atunci s-ar găsi o descompunere $\pi_{R_1}(r), \dots, \pi_{R_m}(r)$ a relației $r(R)$ ce nu satisface mulțimea de dependențe F , dar posedă proprietatea joncțiunii fără pierderi.

Exemplul 5.1. Fie relația $r(\text{ORAȘ ADRESĂ COD})$, unde ADRESĂ este denumire de stradă, număr de casă, iar COD este codul oficiului poștal ce deservește o anumită adresă. În această relație avem valide următoarele dependențe funcționale

$$F = \{\text{ORAȘ ADRESĂ} \rightarrow \text{COD}, \text{COD} \rightarrow \text{ORAȘ}\}.$$

Adică adresa completă determină funcțional codul poștal, dar codul poștal e de ajuns pentru a determina orașul. Descompunerea relațiilor r asupra schemelor ORAȘ COD și ADRESĂ COD posedă proprietatea joncțiunii fără pierderi. Într-adevăr, întrucât $\text{COD} \rightarrow \text{ORAȘ}$, atunci $\text{COD} \rightarrow \rightarrow \text{ORAȘ}$. Dar conform teoremei 4.3 relația r se descompune fără pierderi în două relații definite pe schemele de mai sus.

Proiecția mulțimii de dependențe F asupra schemei ADRESĂ COD produce numai dependențe triviale ce urmează din axioma reflexivității, în timp ce proiecția mulțimii F pe schema COD ORAȘ produce dependența $\text{COD} \rightarrow \text{ORAȘ}$ și dependențele triviale. Deci

$$\{\pi_{\text{ADRESĂ COD}}(F) \cup \pi_{\text{COD ORAȘ}}(F)\} \neq F.$$

Din altă parte descompunerea unei relații poate conserva dependențele, dar să nu posedă proprietatea joncțiunii fără pierderi.

Exemplul 5.2. Fie relația $r(\text{ABCD})$ și fie mulțimea $F = \{A \rightarrow B, C \rightarrow D\}$ de dependențe funcționale valide în r . Descompunerea relației r în $\pi_{AB}(r)$ și $\pi_{CD}(r)$ conservă dependențele din F , însă ea nu posedă proprietatea joncțiunii fără pierderi. Într-adevăr, tabloul ce definește descompunerea arată ca în fig.5.1. Asupra acestui tablou nu pot fi aplicate F -regulile și întrucât el nu conține un tuplu-scop, descompunerea nu posedă proprietatea joncțiunii fără pierderi.

tab	A	B	C	D
	a_1	a_2	b_{13}	b_{14}
	b_{21}	b_{22}	a_3	a_4

Fig.5.1.

În capitolul 1 s-a definit noțiunea de cheie a unei relații sau a unei scheme și sau discutat problemele legate de această noțiune. E evident că noțiunea de cheie este în strânsă corelație cu noțiunea de dependență funcțională. Prin urmare, aici ne vom opri asupra repetării noțiunii de cheie în termenii dependențelor funcționale.

Vom presupune mai departe, când vom avea nevoie, că schema unei relații constă din două componente $S = (R, F)$, unde R este propriu-zis schema, iar F mulțimea de dependențe definite pe mulțimea R .

Definiția 5.3. Fie $S = (R, F)$ o schemă relațională. O submulțime de attribute K a schemei R se numește supercheie pentru schema S , dacă $K \rightarrow R$ este în F^+ . Submulțimea de attribute K din R se numește cheie, dacă K e supercheie și pentru orice submulțime

proprie K^1 a supercheii K dependența $K^1 \rightarrow R$ nu este în F^+ . Dependențele de forma $K \rightarrow R$, unde K este cheie sau supercheie a schemei S , le vom numi *dependențe cheie*.

Exemplul 5.3. Fie schema relațională $S=(ABCDEFG, \{A \rightarrow BCF, C \rightarrow D, BD \rightarrow E, EF \rightarrow G\})$. Să găsim cheile și supercheile acestei scheme.

Calculăm $A^+=ABCDEFG$, $C^+=CD$, $(BD)^+=BDE$ și $(EF)^+=EFG$. Întrucât atributul A determină funcțional toate attributele schemei, el este cheie. Uniunea atributului A cu orice submulțime din $BCDEFG$ formează supercheile schemei S .

5.3. Anomalii și redundanțe

De ce o schemă a bazei de date poate fi “rea”? Anomaliile, care apar în lucrul cu baza de date, se produc datorită dependențelor “nedorite” care se manifestă între attributele din cadrul schemelor relațiilor din baza de date. Aceste dependențe determină creșterea redundanței datelor și reducerea flexibilității structurii bazei de date, făcând extrem de dificil lucrul cu ea.

Deci, în primul rând, o schemă poate fi ineficientă fiindcă conține o mulțime de *date redundante*.

În al doilea rând, ca o consecință a primei cauze, actualizarea unei baze redundante poate duce la situația când ea va conține fapte logic contradictorii. O parte de date pot rămâne nemodificate. Deci o bază de date “rea” duce la apariția unor *inconsistențe la modificarea datelor*.

În al treilea rând, o bază de date “rea” *poate limita posibilitatea de inserare a datelor*. Într-o relație nu pot fi introduse date despre o entitate până nu se cunosc alte date conform restricțiilor de integritate ale entității.

În al patrulea rând, pot apărea *pierderi de date la ștergere*. În mod normal, prin operația de ștergere trebuie să se poată elimina din baza de date numai datele pe care dorim să le ștergem. Atunci când, concomitent cu aceste date sunt șterse și altele, care nu mai pot fi reconstruite din baza de date, spunem că la operația de ștergere se produc pierderi de date.

5.4. Forma normală unu

Precum s-a menționat în capitolul 1, domeniile atributelor sunt simple, adică ele sunt atomice (nu pot fi descompuse din punctul de vedere al sistemului de gestiune al bazei de date). Cu alte cuvinte, valorile ce le pot primi attributele nu sunt liste, mulțimi sau alte structuri complexe.

Definiția 5.4. Schema relațională R se găsește în *forma normală unu*, dacă pentru orice atribut A din R valorile din $\text{dom}(A)$ sunt atomice. Schema unei baze de date se găsește în forma normală unu, dacă orice schemă relațională din ea este în forma normală unu.

Forma normală unu este forma de bază a relațiilor, care figurează ca cerință minimală la majoritatea SGBD-urilor. Toate exemplele de relații considerate până aici au fost în forma normală unu.

Definirea noțiunii de valoare atomică e destul de dificilă. Valoarea atomică dintr-o aplicație în altă aplicație poate fi considerată nonatomică. De aceea ne vom conduce de următoarea regulă: atributul nu este atomic, dacă în aplicații el se utilizează pe părți.

În general se cunosc două tipuri de attribute nonatomice. Unul din ele sunt listele sau mulțimile de valori.

<i>cămin</i>	NUME_STUDENT	CAMERĂ
	Ionescu, Vasilachi	301
	Popovici	302
	Gârlea, Efim	303

Fig.5.2(a)

<i>cămin</i>	NUME_STUDENT	CAMERA
	Ionescu	301
	Vasilachi	301
	Popovici	302
	Gârlea	303
	Efim	303

Fig.5.2(b)

Exemplul 5.4. Relația *cămin* din fig.5.2(a) nu se află în forma normală unu, fiindcă atributul NUME_STUDENT nu e atomic.

Aducerea relației *cămin* în forma normală unu presupune eliminarea listelor de valori. Pentru orice valoare din listă pe care o poate primi atributul NUME_STUDENT se formează un tuplu aparte, conținând numele studentului și camera unde locuiește. Relația *cămin* adusă în forma normală unu arată ca în fig.5.2(b).

Alt tip de attribute nonatomice sunt attributele compuse.

<i>data naștere</i>	NUME_STUDENT	DATA_NAȘTERE
	Ionescu	9 ianuarie 1979
	Vasilachi	21 februarie 1978
	Popovici	15 decembrie 1977
	Gârlea	6 iunie 1979
	Efim	9 ianuarie 1978

Fig.5.3(a)

Exemplul 5.5. Relația *data naștere* din fig.5.3(a) nu este în forma normală unu, dacă dorim să avem accesul la unele componente ale atributului DATA_NAȘTERE.

Pentru a aduce relația *data naștere* în forma normală unu atributul compus DATA_NAȘTERE se divizează în trei attribute ZI, LUNĂ, AN. Noua relație *data naștere* din fig.5.3(b) se găsește în forma normală unu.

<i>data naștere</i>	NUME_STUDENT	ZI	LUNĂ	AN
	Ionescu	9	ianuarie	1979
	Vasilachi	21	februarie	1978
	Popovici	15	decembrie	1977

Gârlea	6	iunie	1979
Efim	9	ianuarie	1978

Fig.5.3(b)

Utilitatea formei normale unu este destul de evidentă. Listele de valori distrug structura naturală dreptunghiulară a unei relații. Este extrem de greu să te referi la un element din grupul de valori, fiindcă trebuie specificată cumva poziția valorii căutate. Și, bineînțeles, că operația de actualizare nu poate fi efectuată. Cu atât mai mult, că cheia NUME_STUDENT a relației *cămin* nu poate fi specificată în cazul unei liste de valori.

În afară de aceasta, diverse părți ale unui atribut partiționat pot să se comporte în mod diferit din punctul de vedere al dependențelor. Presupunem că în prima relație *data_naștere* din fig.5.3(a) s-a adăugat atributul SEMN valorile căruia sunt semnele zodiacului. Tot ce se poate de făcut în această relație este să stabilim dependența funcțională DATA_NAȘTERE→SEMN. Dar această constrângere de integritate permite ca doi indivizi născuți în aceeași zi și aceeași lună, dar ani diferiți, să aibă semne diferite ale zodiacului.

Relația a doua *data_naștere* din fig.5.3(b) este lipsită de acest dezavantaj, fiindcă aici se poate defini dependența funcțională ZI, LUNĂ→SEMN, ce corespunde semanticii semnului zodiacului, care nicidecum nu depinde de anul în care este născută persoana dată, ci numai de ziua și luna nașterii. Deci unul din avantajele formei normale unu constă în aceea că ea poate exprima dependențele la așa grad de detaliere, de care avem nevoie.

5.5. Forma normală doi

Apariția formei normale doi a fost motivată de reducerea redundanței și eliminarea unor anomalii ce apar la actualizarea schemelor în forma normală unu.

Considerăm relația *r* din fig. 5.4.

r	DISCIPLINĂ	PROFESOR	GRUPĂ	ȘEF_GR
	Baze de date	Popescu	CIB-941	Vasilachi
	Programarea logică	Petrache	CIB-942	Gârlea
	Structuri de date	Ciobanu	CIB-942	Gârlea
	Cerc. operaționale	Cazacu	CIB-942	Gârlea

Fig.5.4. Relația *r* în forma normală unu

Relația *r*(DISCIPLINĂ PROFESOR GRUPĂ ȘEF_GR) este constrânsă de două dependențe funcționale: GRUPĂ→ȘEF_GR, semnificând că grupa de studenți are un singur șef și GRUPĂ DISCIPLINĂ→PROFESOR ce presupune că o disciplină într-o grupă de studiu este predată de un singur profesor. Este evident că singura cheie a acestei relații este mulțimea {GRUPĂ DISCIPLINĂ}.

Relația dată se găsește în forma normală unu. Dar să observăm dezavantajele ce le posedă o relație cu astfel de schemă.

În primul rând, sunt limitate posibilitățile de inserare a datelor. În relația *r* nu pot fi introduse date despre o grupă, adică șeful grupei, decât atunci când se cunoaște măcar

o disciplină ce va fi predată în această grupă. Atributul DISCIPLINĂ face parte din cheie și nu poate avea valoare nedeterminată.

În al doilea rând, pot fi pierderi de date la ștergere. De exemplu, în situația când disciplina Baze de date nu se mai predă grupei 941 tuplul dat trebuie șters. Însă ștergerea acestui tuplu din relația considerată determină pierderea datelor despre șeful grupei 941, întrucât acestei grupe nu se mai predau de acum nici o disciplină (fie din cauza că pentru această grupă s-a terminat semestrul de studiu mai devreme în legătură cu plecarea la practică).

În al treilea rând, persistă o redundanță de date. De exemplu, faptul că Gârlea este șeful grupei CIB-942 se repetă de trei ori.

Această redundanță implică al patrulea dezavantaj: apariția unor inconsistențe la modificarea datelor. Presupunem că s-a schimbat șeful grupei 942. Modificarea tuplurilor poate duce la apariția inconsistențelor, dacă numele șefului de grupă nu este actualizat în toate tuplurile. În mod normal, numele șefului grupei trebuie de scris o singură dată, lucru posibil de realizat numai dacă datele despre grupă s-ar păstra într-o relație separată.

Din punctul de vedere al actualizării fără anomalii și îndepărtării redundanței, păstrarea datelor în două relații r_1 și r_2 (vezi fig. 5.5(a), 5.5(b)) e binevenită.

r_1	DISCIPLINĂ	PROFESOR	GRUPĂ
	Baze de date	Popescu	CIB-941
	Programare logică	Petrache	CIB-942
	Structuri de date	Ciobanu	CIB-942
	Cerc. operaționale	Cazacu	CIB-942

Fig.5.5(a) Relația r_1

r_2	GRUPĂ	ȘEF
	CIB-941	Vasilachi
	CIB-942	Gârlea

Fig.5.5(b) Relația r_2

Este evident că relația r se restabilește din r_1 și r_2 , adică $r = r_1 \bowtie r_2$. În afară de aceasta, au dispărut anomaliile de actualizare și ne-am eliberat de careva redundanță.

Să trecem la expunerea strictă a formei normale doi.

Definiția 5.4. Fie relația $r(R)$ și $A \in R$. Atributul A se numește *primar*, dacă el aparține unei chei a schemei R și *nonprimar* în caz contrar.

Definiția 5.5. Fie $X \rightarrow A$ o dependență funcțională netrivială. Atributul A este *parțial dependent* de X , dacă există o submulțime proprie Y a mulțimii X și $Y \rightarrow A$. Dacă nu există o astfel de submulțime proprie, se spune că A *depinde complet* de X .

Exemplul 5.6. Fie $F = \{DISCIPLINĂ \rightarrow GRUPĂ, GRUPĂ \rightarrow ȘEF\}$ asupra schemei $R = DISCIPLINĂ \rightarrow PROFESOR \rightarrow GRUPĂ \rightarrow ȘEF$. Aici atributul ȘEF depinde parțial de DISCIPLINĂ GRUPĂ, iar atributul PROFESOR depinde complet de DISCIPLINĂ GRUPĂ. Întrucât mulțimea de attribute $\{DISCIPLINĂ, GRUPĂ\}$ este

singura cheie a schemei R, attributele DISCIPLINĂ și GRUPĂ sunt primare, iar PROFESOR și ȘEF sunt nonprimare.

Definiția 5.6. Schema unei relații R se găsește în *forma normală doi* în raport cu mulțimea de dependențe funcționale F, dacă ea se găsește în forma normală unu și orice atribut nonprimar nu depinde parțial de careva cheie a schemei R. Schema bazei de date se găsește în forma normală doi, dacă orice schemă relațională a ei se găsește în forma normală doi.

Exemplul 5.7. Fie R și F din exemplul 5.6. Schema bazei de date $Db=\{R\}$ nu se găsește în forma normală doi, fiindcă atributul ȘEF depinde parțial de cheia $\{DISCIPLINĂ, GRUPĂ\}$. Dar schema $Db=\{DISCIPLINĂ \text{ PROFESOR } GRUPĂ, GRUPĂ \text{ ȘEF}\}$ bazei de date din fig.5.5(a) și 5.5(b) se găsește în forma normală doi.

Într-adevăr, în schema relațională $R_1 = DISCIPLINĂ \text{ PROFESOR } GRUPĂ$, atributul PROFESOR este nonprimar și el depinde complet de cheia DISCIPLINĂ GRUPĂ. Cheia schemei $R_2 = GRUPĂ \text{ ȘEF}$ este atributul GRUPĂ. Deci singurul atribut nonprimar e ȘEF care depinde complet de cheie.

Problema determinării, dacă un atribut e primar e legată de problema găsirii cheilor unei relații. Prin urmare, determinarea dacă o schemă se găsește în forma normală doi e o problemă NP-completă.

5.6. Forma normală trei

Considerăm relația $r(\text{STUDENT } DISCIPLINĂ \text{ PROFESOR } COD_PROF)$ din fig.5.6.

r	STUDENT	DISCIPLINĂ	PROFESOR	COD_PROF
	Vasilachi	Baze de date	Popescu	P.021
	Marin	Baze de date	Popescu	P.021
	Guțu	Baze de date	Popescu	P.021
	Vasilachi	Programarea logică	Petrache	P.024

Fig.5.6. Relația $r(\text{STUDENT } DISCIPLINĂ \text{ PROFESOR } COD_PROF)$ în forma normală doi

Relația r este constrânsă de următoarele dependențe funcționale:

$STUDENT \text{ } DISCIPLINĂ \rightarrow COD_PROF$,

$COD_PROF \rightarrow PROFESOR$,

$PROFESOR \rightarrow COD_PROF$.

Cheia acestei relații e formată din două attribute STUDENT și DISCIPLINĂ. Deci ele sunt primare. Attributele PROFESOR și COD_PROF sunt nonprimare. Întrucât ele depind complet de cheie, relația r se găsește în forma normală doi.

Dar să observăm că și această relație nu e lipsită de unele anomalii.

În relația r nu poate fi inserat numele unui profesor și codul lui, decât atunci când acest profesor predă măcar o disciplină în momentul dat. Deci în r se manifestă anomalia de inserare a datelor.

În situația, când profesorul Petrache nu mai predă disciplina Programarea logică, operația de ștergere a acestui tuplu determină pierderea codului acestui profesor. Deci schema dată nu e liberă nici de anomaliile de ștergere a datelor.

În afară de aceasta, perechea de date <Popescu P.021> se repetă de atâtea ori câți studenți ascultă prelegerile profesorului Popescu. Prin urmare, persistă o redundanță, care poate duce la apariția anomaliilor de modificare și inconsistență a datelor. De exemplu, dacă codul profesorului Popescu se va schimba cu P.022, atunci neapărat trebuie modificate toate tuplurile (dar nu se știe numărul lor) și de făcut substituția corespunzătoare. O singură greșeală poate duce la violarea dependențelor PROFESOR→COD_PROF și COD_PROF→PROFESOR.

Aceste anomalii pot fi eliminate, dacă relația r se descompune în două relații r_1 și r_2 din fig.5.7(a) și 5.7(b). În afară de aceasta, relația r poate fi restabilită prin joncțiunea proiecțiilor sale r_1 și r_2 . Existența joncțiunii fără pierderi este evidentă.

r_1	STUDENT	DISCIPLINĂ	COD_PROF
	Vasilachi	Baze de date	P.021
	Marin	Baze de date	P.021
	Guțu	Baze de date	P.021
	Vasilachi	Programarea logică	P.024

Fig.5.7(a). Relația r_2 (STUDENT DISCIPLINĂ COD_PROF)

r_2	PROFESOR	COD_PROF
	Popescu	P.021
	Petrache	P.024

Fig.5.7(b). Relația r_2 (COD_PROF PROFESOR)

Definiția 5.7. Fie relația $r(R)$, $X, Y \subseteq R$ și $A \in R$. Vom spune că atributul A *depinde tranzitiv* de X prin Y , dacă sunt satisfăcute condițiile:

- (1) $X \rightarrow Y$,
- (2) $Y \not\rightarrow X$ (adică X nu depinde funcțional de Y),
- (3) $Y \rightarrow A$
- (4) $A \notin XY$.

În această definiție, condițiile (1) și (3) implică $X \rightarrow A$ conform regulii tranzitivității. Condiția (2) este esențială, de altfel $X \leftrightarrow Y$. Condiția (4), de asemenea, e esențială, în caz contrar $X \rightarrow A$ poate fi dedusă cu ajutorul reflexivității (dacă $A \in X$) sau cu ajutorul regulii proiectivității (dacă $A \in Y$).

Exemplul 5.8. Considerăm relația din fig.5.6. Atributul nonprimar PROFESOR este tranzitiv dependent de cheia {STUDENT, DISCIPLINĂ} prin COD_PROF fiindcă

- (1) $STUDENT \text{ } DISCIPLINĂ \rightarrow COD_PROF$ (întrucât {STUDENT, DISCIPLINĂ} e cheie și deci determină toate atributele din schema relației r),
- (2) $COD_PROF \not\rightarrow STUDENT \text{ } DISCIPLINĂ$,
- (3) $COD_PROF \rightarrow PROFESOR$,
- (4) $PROFESOR \notin STUDENT \text{ } DISCIPLINĂ$.

Definiția 5.8. Schema R se găsește în *forma normală trei* în raport cu o mulțime de dependențe funcționale F , dacă R se găsește în forma normală unu și orice atribut

nonprimar nu depinde tranzitiv de careva cheie a schemei R . Schema bazei de date $Db = \{R_1, \dots, R_m\}$ se găsește în forma normală trei, dacă orice schemă relațională $R_i \in Db$, $1 \leq i \leq m$, se găsește în forma normală trei.

Exemplul 5.9. Schema relației din fig.5.6 nu se găsește în forma normală trei, fiindcă, cum s-a văzut în exemplul 5.8, atributul nonprimar PROFESOR depinde tranzitiv de cheia $\{STUDENT, DISCIPLINĂ\}$. În schimb, schema bazei de date din fig.5.7(a) și 5.7(b) $Db = \{STUDENT \text{ DISCIPLINĂ } COD_PROF, COD_PROF \text{ PROFESOR}\}$ se găsește în forma normală trei.

Într-adevăr, să examinăm pe rând schemele relaționale din Db . Cheia relației r_1 este $\{STUDENT, DISCIPLINĂ\}$. Atributele antrenate în această cheie sunt primare. Singurul atribut nonprimar este COD_PROF . El nu depinde tranzitiv de $\{STUDENT, DISCIPLINĂ\}$. Deci relația r_1 (sau schema ei) se găsește în forma normală trei.

Cât privește relația r_2 , în ea sunt valide dependențele funcționale $COD_PROF \rightarrow PROFESOR$ și $PROFESOR \rightarrow COD_PROF$. Deci r_2 are două chei COD_PROF și $PROFESOR$. Întrucât schema relației r_2 nu conține atribute nonprimare ea este în forma normală trei.

E firească întrebarea, care este corelația dintre forma normală trei și forma normală doi. Răspunsul îl dă următoarea teoremă.

Teorema 5.1. Schema unei relații ce se găsește în forma normală trei se găsește și în forma normală doi.

Demonstrație. Teorema poate fi reformulată în felul următor: dacă schema unei relații nu se găsește în forma normală doi, atunci ea nu se găsește nici în forma normală trei. Deci trebuie să arătăm că dependența parțială implică dependența tranzitivității.

Fie schema relațională $S = (R, F)$ și presupunem că atributul nonprimar A e parțial dependent de o cheie, fie K . Adică $K \rightarrow A \in F^+$ și $K^1 \rightarrow A \in F^+$, unde $K^1 \subset K$. Conform regulii reflexivității, $K^1 \subset K$ implică $K \rightarrow K^1 \in F^+$ și atunci condiția (1) a definiției 5.7 e satisfăcută. Condiția (2) tot e satisfăcută, adică $K^1 \not\subset K$, fiindcă în caz contrar K nu este cheie. Condiția (3) urmează din ipoteză, iar condiția (4) e satisfăcută din presupunerea că A este atribut nonprimar, adică nu aparține cheii K și deci nici lui K^1 . Prin urmare, atributul nonprimar A depinde tranzitiv de cheia K .

Exemplul 5.10. Schema bazei de date $Db = \{DISCIPLINĂ \text{ PROFESOR } GRUPĂ, GRUPĂ \text{ ȘEF}\}$ din fig.5.5(a) și 5.5(b) se găsește în forma normală trei, deci se găsește și în forma normală doi.

Într-adevăr, schema $R_1 = DISCIPLINĂ \text{ PROFESOR } GRUPĂ$ se găsește în forma normală trei, fiindcă cheia e $\{DISCIPLINĂ, GRUPĂ\}$, iar singurul atribut nonprimar este PROFESOR și el nu depinde tranzitiv de cheie. Cheia schemei $R_2 = GRUPĂ \text{ ȘEF}$ este atributul GRUPĂ. Atributul nonprimar ȘEF nu depinde tranzitiv de GRUPĂ.

E ușor de observat, din cele expuse mai sus, că definiția formei normale trei poate fi formulată și altfel.

Definiția 5.9. Schema unei relații se găsește în forma normală trei, dacă orice atribut ce depinde tranzitiv de cheie este primar.

Atunci putem formula următoarea teoremă.

Teorema 5.2. Schema R se găsește în forma normală trei în raport cu mulțimea de dependențe funcționale F , dacă pentru orice dependență netrivială $X \rightarrow A \in F^+$

(1) X este supercheie pentru R
sau

(2) A este atribut primar.

Demonstrație. Fie $X \rightarrow A$ o dependență netrivială și fie K o cheie a schemei R. Din definiția 5.9 reiese că nu e necesară examinarea cazului, când A este atribut primar. Condiția (2) e evidentă. Să arătăm că pentru orice atribut nonprimar A, dependența $K \rightarrow A$ nu este tranzitivă. Dacă presupunem că condiția (1) nu e satisfăcută, atunci $K \rightarrow X \in F^+$ (fiindcă K e cheie) și $X \not\rightarrow K$. Dar aceasta este dependența tranzitivă a atributului nonprimar de cheie.

Exemplul 5.11. Fie schema bazei de date $Db = \{(AC, \{C \rightarrow A\}), (ABE, \{AE \rightarrow B\}), (BCDEF, \{BF \rightarrow C, CD \rightarrow EF, EF \rightarrow CD\})\}$.

Este ușor de constatat că schema Db se găsește în forma normală trei. Într-adevăr, schemele relaționale $R_1 = (AC, \{C \rightarrow A\})$ și $R_2 = (ABE, \{AE \rightarrow B\})$ se găsesc în forma normală trei fiindcă nu au dependențe tranzitive. Să examinăm schema $R_3 = (BCDEF, \{BF \rightarrow C, CD \rightarrow EF, EF \rightarrow CD\})$. Schema R_3 are trei chei BCD, BDF și BEF. Dependențele tranzitive $BCD \rightarrow EF$, $BDF \rightarrow C$, $BEF \rightarrow CD$ includ numai atribute primare (ele toate fac parte dintr-o cheie). Prin urmare și schema R_3 este în forma normală trei.

5.7. Forma normală Boyce-Codd

Forma normală trei nu interzice dependența tranzitivă a atributelor primare de cheie. Însă și unele relații în forma normală trei nu sunt lipsite de anomalii de actualizare a datelor.

r	ORAȘ	ADRESĂ	COD
	o_1	a_1	c_1
	o_1	a_2	c_1
	o_1	a_3	c_1
	o_1	a_4	c_2

Fig.5.8. Relația $r(\text{ORAȘ ADRESĂ COD})$
în forma normală trei

Exemplul 5.12. Considerăm relația $r(\text{ORAȘ ADRESĂ COD})$ din fig.5.8 examinată și în exemplul 5.1. Relația r satisface dependențele funcționale $\text{ORAȘ ADRESĂ} \rightarrow \text{COD}$ și $\text{COD} \rightarrow \text{ORAȘ}$. Mulțimea de atribute $\{\text{ORAȘ, ADRESĂ}\}$ formează cheia relației. Atributul ORAȘ e primar. Nici un atribut nonprimar nu depinde tranzitiv de această cheie. Deci relația r se află în forma normală trei.

Însă, în ea nu putem introduce un tuplu ce conține date despre un oraș și codul lui, până nu cunoaștem adresa asociată de acest cod.

În afară de aceasta, în r se repetă perechea ORAȘ COD ce poate duce la apariția inconsistențelor la modificarea acestor date.

Dacă relația r e descompusă în două relații r_1 și r_2 (precum sunt prezentate în fig.5.9(a) și 5.9(b)), atunci aceste dezavantaje lipsesc.

r_1	COD	ADRES Ă
	c_1	a_1
	c_1	a_2
	c_1	a_3
	c_2	a_4

Fig.5.9(a). Relația $r_1(\text{COD ADRESĂ})$

r_2	ORAȘ	COD
	o_1	c_1
	o_1	c_2

Fig.5.9(b). Relația $r_2(\text{ORAȘ COD})$

Definiția 5.10. Schema R se găsește în forma normală Boyce-Codd în raport cu o mulțime de dependențe funcționale F , dacă pentru orice dependență $X \rightarrow Y \in F^+$, determinantul X este o supercheie a schemei R . Schema bazei de date se găsește în forma normală Boyce-Codd, dacă orice schemă relațională din ea se găsește în forma normală Boyce-Codd.

Exemplul 5.13. Considerând relația din fig.5.8 ce satisface dependența funcțională $\text{COD} \rightarrow \text{ORAȘ}$, conchidem că COD nu este supercheie. Deci schema acestei relații nu se găsește în forma normală Boyce-Codd.

Examinăm relațiile r_1 și r_2 din fig.5.9(a) și 5.9(b).

Cheia relației r_1 constă din toată mulțimea de attribute $\{\text{COD}, \text{ADRESĂ}\}$. În r_1 nu e validă nici o dependență funcțională, în afara celor triviale. Deci schema $R_1 = \text{COD ADRESĂ}$ se găsește în forma normală Boyce-Codd.

Relația r_2 satisface dependența funcțională $\text{COD} \rightarrow \text{ORAȘ}$. Atributul COD este cheie, deci și supercheie. Prin urmare, r_2 se găsește în forma normală Boyce-Codd.

Relația r poate fi restabilită din proiecțiile sale, r_1 și r_2 . Deci descompunerea dată posedă proprietatea joncțiunii fără pierderi. Însă descompunerea dată nu conservă dependențele funcționale. Dependența $\text{ORAȘ ADRESĂ} \rightarrow \text{COD}$ validă în relația r nu se deduce din dependențele valide în relațiile r_1 și r_2 .

Din exemplul 5.13 putem face concluzia că forma normală trei nu implică forma normală Boyce-Codd.

Următoarea afirmație stabilește legătura dintre forma normală Boyce-Codd și forma normală trei.

Teoremă 5.3. Dacă schema R se găsește în forma normală Boyce-Codd, atunci R se găsește și în forma normală trei.

Demonstrație. Validitatea acestei afirmații urmează direct din definiția formei normale Boyce-Codd și teorema 5.2.

5.8. Normalizarea prin descompunere

5.8.1. Aducerea schemelor în forma normală trei

Normalizarea este procesul de aducere a schemei într-o formă normală dată. Algoritmul FN3 aduce schema R în forma normală trei prin descompunere.

Algoritmul FN3 (R, F, Db)

Intrare: Schema R; F – o mulțime de dependențe funcționale definite pe schema R.

Ieșire: Db – schema bazei de date în forma normală trei.

```
begin
1   k:=1;
2   Rk:=R;
3   for i=1 to k do
4       keys (Ri,F,K);
5       AttrNP:=Ri \ ∪ Kj, ∀ Kj ∈ K;
6       while ∃ X → Y ∈ F+ & X ↛ Ri & X ∩ Y = ∅ & XY ⊆ Ri & Y ⊆ AttrNP
          do
              begin
2          k:=k+1;
3          Rk:=XY;
4          Ri:= Ri \ Y;
              end
10  Db:={R1,..., Rk};
    return (Db);
end.
```

La început vom porni de la ideea că orice schemă ce nu se găsește în forma normală trei poate fi descompusă într-o serie de scheme ce se găsesc în forma normală trei. Descompunerea presupune divizarea unei scheme R în două scheme R₁ și R₂ astfel că orice relație r(R) ce satisface mulțimea dată de dependențe F se proiectează fără pierderi asupra schemelor R₁ și R₂, adică $r = \pi_{R_1}(r) \bowtie \pi_{R_2}(r)$. Acest proces se repetă asupra schemelor R₁ și R₂, bineînțeles, până subschemele formate sunt aduse în forma normală trei.

În linia 3 variabila i denotă schema curentă, iar k numărul de scheme deja create.

Linia 4 determină mulțimea K de chei a schemei curente R_i.

Linia 5 construiește mulțimea AttrNP de attribute nonprimare din R_i.

Linia de bază a algoritmului este 6. Ea analizează dacă schema curentă se găsește în forma normală trei. Pentru fiecare dependență validă în schema curentă cu determinatul format numai din attribute nonprimare se verifică, dacă determinantul ei este supercheie. Dacă nu, atunci conform teoremei 5.2 schema R_i nu se găsește în forma normală trei. În acest caz (liniile 7-9) se produce descompunerea schemei R_i: se formează o nouă schemă din attributele implicate în dependență, R_i=XY, iar schema R_i e substituită de R_i \ Y. Conform teoremei 4.3, această descompunere posedă proprietatea joncțiunii fără pierderi. Apoi continuă analiza schemelor deja formate. Dacă în cadrul lor nu se mai manifestă dependențe ce satisfac condițiile din linia 6, atunci schema obținută se găsește în forma normală trei.

Exemplul 5.14. Considerăm schema relațională $R = \text{DPOCSN}$, unde D e disciplină, P – profesor, O – ora, C – clasă, S – student și N – nota. Presupunem că pe schema dată sunt definite următoarele dependențe funcționale:

- $D \rightarrow P$ – orice disciplină e predată de un singur profesor;
 - $OC \rightarrow D$ – într-o clasă în același timp se predă o singură disciplină;
 - $OP \rightarrow C$ – profesorul într-un anumit timp se găsește într-o singură clasă;
 - $DS \rightarrow N$ – orice student are o singură notă finală la o disciplină;
 - $OS \rightarrow C$ – studentul se găsește la ora dată într-o singură clasă.
- Să se aducă schema R în forma normală trei.

De la începutul algoritmului se formează schema $R_1 = R$. În linia 4 a fost găsită o singură cheie pentru R_1 și anume OS . Deci $\{D, P, C, N\}$ formează mulțimea de attribute nonprimare. Pentru a aduce schema R_1 la forma normală trei considerăm dependența funcțională $D \rightarrow P$ care satisface toate condițiile din linia 6. Formăm o nouă schemă (linia 8) $R_2 = DP$, iar R_1 este substituită de $R_1 = \text{DOCSN}$. Schema R_2 se găsește în forma normală trei, fiindcă nu există vre-o dependență definită pe această schemă și să satisfacă condițiile liniei 6. Schema R_1 nu este în forma normală trei, fiindcă există o dependență, de exemplu, $OC \rightarrow D$ ce satisface condițiile liniei 6. Se formează a treia schemă $R_3 = OCD$, dar R_1 devine de acum egală cu OSCN . Este evident că schema R_3 se găsește în forma normală trei. Schema R_1 de asemenea se găsește în forma normală trei, fiindcă singura dependență, $OS \rightarrow C$, definită pe attributele schemei R_1 nu satisface condițiile liniei 6.

Deci schema R s-a descompus fără pierderi în R_1 , R_2 și R_3 . Schema bazei de date $Db = \{R_1, R_2, R_3\}$ se găsește în forma normală trei.

Să menționăm că schema $Db = \{R_1, R_2, R_3\}$ se găsește și în forma normală Boyce-Codd.

5.8.2. Aducerea schemei la forma normală Boyce-Codd

Adresându-ne la definiția formei normale Boyce-Codd, un algoritm de aducere a schemelor în această formă poate fi prezentat astfel.

Algoritmul FNBC (R, F, Db)

Intrare: Schema R ;

F - o mulțime de dependențe funcționale asupra schemei R .

Ieșire: Db – schema bazei de date în forma normală Boyce-Codd.

begin

$k := 1$;

$R_k := R$;

 for $i = 1$ to k do

 while $\exists X \rightarrow Y \in F^+ \ \& \ X \not\rightarrow R_i \ \& \ X \cap Y = \emptyset \ \& \ XY \subseteq R_i$ do

 begin

$k := k + 1$;

$R_k := XY$;

$R_i := R_i \setminus Y$;

 end

$Db := \{R_1, \dots, R_k\}$;

 return (Db);

end

Exemplul 5.15. Fie $R = (ABCDEF, \{AB \rightarrow E, AC \rightarrow F, AD \rightarrow B, B \rightarrow C, C \rightarrow D\})$. Să se aducă schema R în forma normală Boyce-Codd.

$R_1 = ABCDEF$. Considerăm pe rând dependențele funcționale valide în R_1 . Dependența $AB \rightarrow E$ nu participă la descompunere, fiindcă $(AB)^+ = ABCDEF$. Considerăm dependența $AC \rightarrow F$. $(AC)^+ = ABCDEF$, deci $AC \rightarrow F$, de asemenea, nu poate fi aplicată la descompunerea schemei R_1 . Același lucru putem spune și despre dependența $AD \rightarrow B$, fiindcă $(AD)^+ = ABCDEF$.

Determinantul dependenței $B \rightarrow C$ nu este supercheie, fiindcă $B^+ = BC$. Deci R_1 se descompune în două scheme: $R_2 = BC$ și $R_1 = ABDEF$. Schema R_2 evident se găsește în forma normală Boyce-Codd, fiindcă constă numai din două atribute. Altă dependență, ce poate fi aplicată de acum asupra schemei R_1 modificate, este $B \rightarrow D \in F^+$, unde $BD \subseteq R_1$. Construim schemele $R_3 = BD$ și $R_1 = ABDEF$. Cheia schemei R_3 este B , iar a schemei R_1 - AB . Schema bazei de date în forma normală Boyce-Codd este $Db = \{(ABEF, \{AB\}), (BC, \{B\}), (BD, \{B\})\}$.

5.8.3. Dezavantajele normalizării prin descompunere

Dat fiind faptul că algoritmul FN3 necesită calcularea cheilor schemei și determinarea atributelor nonprimare, iar algoritmi FN3 și FNBC necesită examinarea dependențelor din F^+ valide în schema curentă, complexitatea procesului de normalizare nu e polinomială. Acesta e primul dezavantaj.

Într-al doilea rând, nu întotdeauna putem obține un număr minimal de scheme relaționale normalizate dintr-o schemă dată.

Exemplul 5.16. Fie schema $R = ABCDE$ și $F = \{AB \rightarrow CDE, AC \rightarrow BDE, B \rightarrow C, C \rightarrow B, C \rightarrow D, B \rightarrow E\}$. Să se aducă R în forma normală trei.

Cheile schemei R sunt $K = \{AB, AC\}$. Aplicăm pentru descompunere dependența $C \rightarrow D$. Atunci

$R_2 = CD, K_2 = \{C\}$;

$R_1 = ABCE, K_1 = \{AB, AC\}$.

Mai departe pentru descompunerea schemei R_1 utilizăm dependența $B \rightarrow E$:

$R_3 = BE, K_3 = \{B\}$;

$R_1 = ABC, K_1 = \{AB, AC\}$.

Schema finală în forma normală trei este

$Db = \{(ABC, \{AB, AC\}), (CD, \{C\}), (BE, \{B\})\}$.

Există, în schimb, altă descompunere cu mai puține scheme relaționale. Dacă utilizăm dependența funcțională $B \rightarrow DE \in F^+$, atunci obținem schemele

$R_2 = BDE, K_1 = \{B\}$;

$R_1 = ABC, K_1 = \{AB, AC\}$.

Deci, $Db = \{(ABC, \{AB, AC\}), (BDE, \{B\})\}$.

A treia problemă constă în apariția dependențelor parțiale în procesul descompunerii schemelor. Aceste dependențe generează scheme cu mai multe scheme relaționale decât e nevoie.

Exemplul 5.17. Fie schema $R = ABCD$ și $F = \{A \rightarrow BCD, C \rightarrow D\}$. Să se aducă schema R în forma normală trei.

Singura cheie a schemei R este A . Utilizăm dependența $BC \rightarrow D$ pentru descompunerea schemei R . Atunci

$R_2 = BCD, K_2 = \{BC\};$

$R_1 = ABC, K_1 = \{A\}.$

În R_2 atributul D depinde parțial de cheia BC, deci poate fi aplicată dependența $C \rightarrow D$ pentru descompunerea schemei R_2 :

$R_3 = CD, K_3 = \{C\};$

$R_2 = BC, K_2 = \{BC\}.$

Deci, schema bazei de date în forma trei este $Db = \{(ABC, \{A\}), (BC, \{BC\}), (CD, \{C\})\}.$

Însă, dacă pentru descompunere asupra schemei R e aplică deodată dependența $C \rightarrow D$, atunci obținem

$R_2 = CD, K_2 = \{C\};$

$R_1 = ABC, K_1 = \{A\}.$

În acest caz $Db = \{(ABC, \{A\}), (CD, \{C\})\}.$ Ultima schemă a bazei de date este o submulțime a primei scheme.

Acest dezavantaj poate fi evitat, dacă dependențele utilizate în descompunere sunt reduse în stânga.

A patra problemă este că normalizarea prin descompunere nu întotdeauna conservă dependențele funcționale.

Exemplul 5.18. Schema bazei de date obținută în exemplul 5.14 nu conservă dependențele $OP \rightarrow C$ și $DS \rightarrow N$. Schema bazei de date obținută în exemplul 5.15 nu conservă dependențele $AC \rightarrow F$ și $AD \rightarrow B$.

A cincea problemă constă în faptul că normalizarea prin descompunere poate produce scheme, în care dependențele, ce pot fi utilizate mai departe în descompunere, sunt latente.

Exemplul 5.19. Fie pe schema $R = ABCD$ e definită mulțimea de dependențe funcționale $F = \{A \rightarrow B, B \rightarrow C\}$. Cheia relației R este AD. Dependența $A \rightarrow B$ poate fi utilizată în descompunerea schemei R:

$R_2 = AB, K_2 = \{A\};$

$R_1 = ACD, K_1 = \{AD\}.$

S-ar părea că schema R_1 se găsește în forma normală trei, însă în R_1 există dependența funcțională latentă $A \rightarrow C$, ce ar trebui să fie utilizată în descompunerea de mai departe.

5.9. Normalizarea prin sinteză

În această secțiune va fi prezentată o altă metodă de aducere a schemelor la forma normală trei, ce nu generează problemele descrise în secțiunea precedentă.

Metoda propusă este o procedură de sinteză, fiindcă pleacă de la mulțimea de dependențe funcționale F cu determinații dintr-un singur atribut și produce schema bazei de date $Db = \{R_1, \dots, R_m\}$ asupra $R = R_1 \dots R_m$. Schema bazei de date trebuie să satisfacă următoarele patru condiții:

- (1) Mulțimea de dependențe formate de cheile fiecărei scheme R_i trebuie să fie o acoperire a mulțimii inițiale F de dependențe funcționale, adică $F \equiv \{K \rightarrow R_i \mid R_i \in Db, 1 \leq i \leq m, K - \text{cheie}\}.$
- (2) Orice schemă relațională R_i din Db se află în forma normală trei.

- (3) Nu există o schemă a bazei de date ce satisface condițiile (1) și (2) cu mai puține scheme relaționale.
- (4) Orice relație $r(R)$ ce satisface F se descompune fără pierderi asupra schemei Db , adică $r = \pi_{R_1}(r) \bowtie \dots \bowtie \pi_{R_m}(r)$.

Să considerăm aceste condiții.

Condiția (1) garantează că descompunerea relației r asupra schemei Db conservă dependențele funcționale F . În afară de aceasta, condiția (1) ne asigură că unicele dependențe valide în R_i , $1 \leq i \leq m$, au în calitate de determinanți cheile schemei R_i . Satisfacerea condiției (1) este soluția problemei patru din secțiunea precedentă.

Condiția (2) este scopul principal al normalizării și necesitatea ei a fost detaliat studiată.

Condiția (3) ne ocrotește de scheme redundante, deci ea soluționează problemele doi și trei din secțiunea precedentă.

Condiția (4) de asemenea a fost considerată.

Problema cinci din secțiunea precedentă nu apare grație îndeplinirii concomitente a condițiilor (1) și (3). Iar problema unu nu se mai pune, fiindcă complexitatea algoritmului de sinteză descris mai jos este polinomială.

Algoritmul SYNT (F, Db)

Intrare: F – o mulțimea de dependențe funcționale

Ieșire: Db schema bazei de date în forma normală trei.

Se găsește o acoperire nonredundantă F_n a mulțimii F .

Se construiește o acoperire redusă în stânga F_r a mulțimii F_n .

Mulțimea F_r se partiționează în clase de echivalență.

Se construiește o mulțime J în felul următor. Fie $J = \emptyset$. Pentru orice două dependențe funcționale din F_r cu determinații X și Y , unde $X \leftrightarrow Y$, se modifică J , $J := J \cup \{X \rightarrow Y, Y \rightarrow X\}$. Pentru orice $A \in Y$, dacă $X \rightarrow A$ se găsește în F_r , atunci $F_r := F_r \setminus \{X \rightarrow A\}$. Același lucru e valabil și pentru orice $B \in X$. Dacă $Y \rightarrow B \in F_r$, atunci $F_r := F_r \setminus \{Y \rightarrow B\}$.

Se elimină dependențele tranzitive. Se găsește o mulțime $F_r^1 \subseteq F_r$, ce satisface $(F_r^1 \cup J)^+ \equiv (F_r \cup J)^+$ și nici o submulțime proprie a mulțimii F_r^1 nu satisface condiția dată. Apoi se includ dependențele din J în clasele de echivalență a mulțimii F_r^1 și fie că obținem mulțimea G de dependențe funcționale.

Se construiesc schemele R_1, \dots, R_m . Fiecare schemă R_i include atributele dependențelor funcționale din clasa de echivalență i și în final obținem schema bazei de date $Db := \{R_1, \dots, R_m\}$.

Să ne oprim acum asupra corectitudinii algoritmului. Algoritmul expus formează un număr minimal de scheme relaționale în Db .

Teorema 5.4. Dacă Db este schema bazei de date sintetizate din mulțimea F , atunci Db conține cel puțin $|\bar{E}_{Fn}|$ scheme relaționale.

Demonstrație. Dependențele ce sunt incluse într-o schemă relațională R_i , $1 \leq i \leq m$, au determinanți echivalenți. Deci Db conține atâtea scheme în câte clase de echivalență este partiționată mulțimea G (vezi algoritmul SYNT). Din lema 3.3 urmează $|\bar{E}_G| = |\bar{E}_{Fn}|$. Dar e cunoscut faptul că $|\bar{E}_F| \geq |\bar{E}_{Fn}|$, adică mulțimea nonredundantă constă dintr-un număr minimal de clase de echivalență.

Această teoremă ne garantează că algoritmul de sinteză satisface condiția (3).

Condiția (1) este asigurată, fiindcă $F \equiv F_n \equiv F_r \equiv G$.

Condiția (2) e satisfăcută de pasul 5 al algoritmului ce elimină dependențele tranzitive.

Acum să vedem dacă e satisfăcută și condiția (4). Cu toate că algoritmul de sinteză soluționează toate cele cinci probleme din secțiunea 5.8.3, nu întotdeauna schema bazei de date posedă proprietatea joncțiunii fără pierderi. Adică nu întotdeauna e satisfăcută condiția (4). Acest lucru nu se petrecea în cazul normalizării prin descompunere.

Exemplul 5.20. Fie $F = \{A \rightarrow C, B \rightarrow C\}$. Algoritmul de sinteză generează schema $Db = \{R_1, R_2\}$, unde

$R_1 = AC, K_1 = \{A\}$;

$R_2 = BC, K_2 = \{B\}$.

Însă e ușor de văzut că relația $r(ABC)$ din fig.5.10 nu se descompune fără pierderi asupra R_1 și R_2 .

r	A	B	C
	a_1	b_1	c_1
	a_2	b_2	c_1

Fig.5.10.

Un alt dezavantaj al algoritmului de sinteză e legat de atributele ce nu sunt antrenate de mulțimea de dependențe funcționale F . Aceste două dezavantaje pot fi eliminate, introducând așa-numita cheie universală.

Definiția 5.11. Fie $Db = \{R_1, \dots, R_m\}$ o schemă a bazei de date asupra atributelor $R = R_1 \dots R_m$ și F o mulțime de dependențe funcționale. Mulțimea $X \subseteq R$ se numește *cheie universală*, dacă $F \models X \rightarrow R$ și nu există X^1 , unde $X^1 \subset X$, ce ar satisface $F \models X^1 \rightarrow R$.

Deci, ca schema bazei de date să posedă proprietatea joncțiunii fără pierderi, ea trebuie să conțină o schemă relațională în care o cheie a ei e universală.

Vom modifica algoritmul de sinteză pentru a elimina cele două dezavantaje menționate mai sus.

La mulțimea inițială de dependențe funcționale F se adaugă o dependență funcțională $R \rightarrow C$, unde $R = R_1 \dots R_m$, iar $C \notin R$.

Este clar că la primul pas al algoritmului dependența $R \rightarrow C$ nu va fi eliminată, fiindcă ea nu e redundantă în F .

La al doilea pas ea va fi redusă în stânga, fie $R^1 \rightarrow C$.

La etapa de partiție, dacă ea va intra într-o clasă de echivalență cu alte dependențe, atunci ea se elimină din F_r și algoritmul continuă mai departe. Dacă ea

singură formează o clasă de echivalență, atunci $R^1 \rightarrow C$ generează schema $R_m = R^1 C$ cu cheia R^1 .

La sfârșitul algoritmului se elimină atributul C din schema R_m , deci $R_m = R^1$.

Exemplul 5.21. Fie F ca în exemplul 5.20, adică $F = \{A \rightarrow C, B \rightarrow C\}$. Adăugăm la F dependența $ABC \rightarrow D$.

Algoritmul de sinteză va genera schema $Db = \{(AC, \{A\}), (BC, \{B\}), (ABD, \{AB\})\}$. Apoi eliminând din ultima schemă relațională atributul D, obținem schema bazei de date în forma normală trei ce posedă proprietatea joncțiunii fără pierderi $Db = \{(AC, \{A\}), (BC, \{B\}), (AB, \{AB\})\}$.

Cu părere de rău, trebuie să recunoaștem că algoritmul modificat violează condiția (3) de minimalitate a schemei.

5.10. Forma normală patru

Definiția 5.12. Schema R se găsește în *forma normală patru* în raport cu mulțimea de dependențe multivaloare și funcționale M , dacă ea se găsește în forma normală unu și orice dependență $X \twoheadrightarrow Y$ din M^+ satisface.

(1) $X \twoheadrightarrow Y$ este trivială (adică $Y \subseteq X$ sau $XY = R$)

sau

(2) X este supercheie pentru schema R .

Schema bazei de date se găsește în forma normală patru, dacă orice schemă a ei se găsește în forma normală patru.

Procedura de aducere în forma normală patru se bazează pe teorema 4.3, care ne spune că o dependență $X \twoheadrightarrow Y$ e validă într-o relație $r(R)$, dacă și numai dacă $r(R)$ este joncțiunea proiecțiilor $\pi_{XY}(r)$ și $\pi_{XZ}(r)$, unde $Z = R \setminus XY$.

Procesul de normalizare decurge în felul următor. Se începe cu schema inițială R . Dacă în această schemă e validă dependența multivaloare netrivială $X \twoheadrightarrow Y$ și X nu e supercheie, atunci schema R se descompune în două scheme $R_1 = XY$ și $R_2 = XZ$, unde $Z = R \setminus XY$. La rândul lor, schemele R_1 și R_2 sunt considerate în privința satisfacerii condițiilor de a fi în forma normală patru. Dacă careva schemă nu e în forma normală patru, procesul de descompunere continuă până toate schemele sunt normalizate. Evident că procesul este finit.

Exemplul 5.22. Fie mulțimea de dependențe $M = \{C \twoheadrightarrow DE, A \rightarrow BC\}$ definită pe schema $R = ABCDE$. Schema R nu se află în forma normală patru, fiindcă $C \twoheadrightarrow DE$ nu e trivială și C nu este cheie. Schema bazei de date ce constă din două scheme $R_1 = CDE$ și $R_2 = ABC$ se găsește în forma normală patru. Într-adevăr, cu toate că $A \rightarrow B \in M^+$, deci $A \twoheadrightarrow B \in M^+$ și $A \twoheadrightarrow B$ nu este trivială, însă A este supercheie pentru R_2 .

Exemplul 5.23. Să se normalizeze schema $R = ABCDEI$, dacă pe ea e definită mulțimea de dependențe $M = \{A \twoheadrightarrow BCD, B \rightarrow AC, C \rightarrow D\}$. Fiindcă $A \twoheadrightarrow BCD$ nu este trivială și A nu e supercheie schema R se descompune fără pierderi în schemele $R_1 = ABCD$ și $R_2 = AEI$. Schema R_2 se găsește în forma normală patru: pe ea e definită o singură dependență multivaloare trivială $A \twoheadrightarrow EI$. Cu toate că în M^+ este $B \twoheadrightarrow AC$, din $B \rightarrow AC$ și $C \rightarrow D$ urmează $B \rightarrow ACD$. Deci, B este cheia schemei R_1 și $B \twoheadrightarrow AC$ nu participă la descompunerea de mai departe a schemei R_1 . Însă, din $C \rightarrow D$ urmează

dependența netrivială $C \rightarrow \rightarrow D$ ce descompune schema R_1 în două: $R_3 = CD$ și $R_1 = ABC$. Schema bazei de date în forma normală patru este $DB = \{ABC, AEI, CD\}$.

Teorema 5.5. Dacă schema R se găsește în forma normală patru, atunci R se găsește în forma normală Boyce-Codd.

Demonstrație. Vom demonstra o afirmație echivalentă: dacă R nu se găsește în forma normală Boyce-Codd, atunci R nu se găsește nici în forma normală patru.

Fie R nu se găsește în forma normală Boyce-Codd. Adică trebuie să existe o dependență funcțională netrivială $X \rightarrow A$ și X nu este supercheie a schemei R . Atunci există un atribut B în R , încât $B \notin AX$ și $X \not\rightarrow B$. Prin urmare, $B \in R \setminus AX$ și atunci dependența $X \rightarrow \rightarrow A$ care este alter ego al dependenței $X \rightarrow A$ nu e trivială. Din condițiile că dependența $X \rightarrow \rightarrow A$ nu e trivială și X nu e supercheie, urmează că R nu se găsește în forma normală patru.

5.11. Forma normală proiecție-joncțiune

Definiția 5.13. Dependența joncțiune $|x|(R_1, \dots, R_m)$ este aplicabilă schemei R , dacă $R = R_1 \dots R_m$.

Definiția 5.14. Schema R se găsește în *forma normală proiecție-joncțiune* în raport cu o mulțime de dependențe joncțiune (dependențele multivaloare sunt aceleași dependențe joncțiune) și funcționale J , dacă ea se găsește în forma normală unu și orice dependență joncțiune $|x|(R_1, \dots, R_m)$ aplicabilă din J^+ este trivială sau orice R_i este supercheie pentru R .

Exemplul 5.24. Fie mulțimea de dependențe $J = \{|x|(ABCD, CDE, BDF), |x|(AB, BCD, AD), A \rightarrow BCDE, BC \rightarrow A\}$ definită pe schema $R = ABCDEF$.

Schema R nu se găsește în forma normală proiecție-joncțiune din cauza dependenței aplicabile $|x|(ABCD, CDE, BDF)$.

Schema bazei de date $Db = \{ABCD, CDE, BDF\}$ se găsește în forma normală proiecție-joncțiune în raport cu J . Cu toate că dependența joncțiune $|x|(AB, BCD, AD)$ e aplicabilă schemei relaționale $ABCD$, Db se găsește în forma normală proiecție-joncțiune datorită faptului că orice subschemă AB , BCD și AD este supercheie pentru schema $ABCD$.

Exemplul 5.25. Fie $R = ABCDEF$ și $J = \{|x|(ABC, ADEF), A \rightarrow BCDE, BC \rightarrow AF\}$.

Schema R se găsește în forma normală proiecție-joncțiune, fiindcă cu toate că dependența $|x|(ABC, ADEF)$ este aplicabilă schemei R , subschemele ei sunt superchei pentru R .

Teorema 5.6. Dacă schema R se găsește în forma normală proiecție-joncțiune în raport cu mulțimea de dependențe J , atunci R se află în forma normală patru.

Demonstrația acestei teoreme urmează direct din condiția că dependența multivaloare netrivială $X \rightarrow Y$ definită asupra schemei R este dependența de joncțiune $|x|(XY, XZ)$, unde $Z = R \setminus XY$.

5.12. Concluzii

Etapele de proiectare a bazei de date pot fi cele de mai jos. Fiecare din aceste etape produce o bază de date mai "bună" decât precedenta. Corelația dintre diverse forme normale este reprezentată în fig.5.11.

- (1) Inițial datele sunt nenormalizate.
- (2) Se elimină atributele ce formează mulțimi de valori sau sunt complexe. În consecință se obține schema relației universale. Se spune că schema acestei relații se găsește în forma normală unu (FN1).
- (3) Pentru a ajunge la forma normală doi (FN2) se elimină dependențele parțiale de chei ale atributelor nonprimare.
- (4) Forma normală trei (FN3) cere eliminarea dependențelor tranzitive ale atributelor nonprimare de chei. De obicei mulți profesioniști în proiectarea bazelor de date se limitează la această formă normală.
- (5) După înlăturarea tuturor dependențelor tranzitive, se obține forma normală Boyce-Codd (FNBC).
- (6) Forma normală patru (FN4) soluționează problemele cauzate de dependențele multivaloare netriviiale.
- (7) Forma normală proiecție-joncțiune (FNPJ) se referă la soluționarea problemei descompunerii fără pierderi a relațiilor.

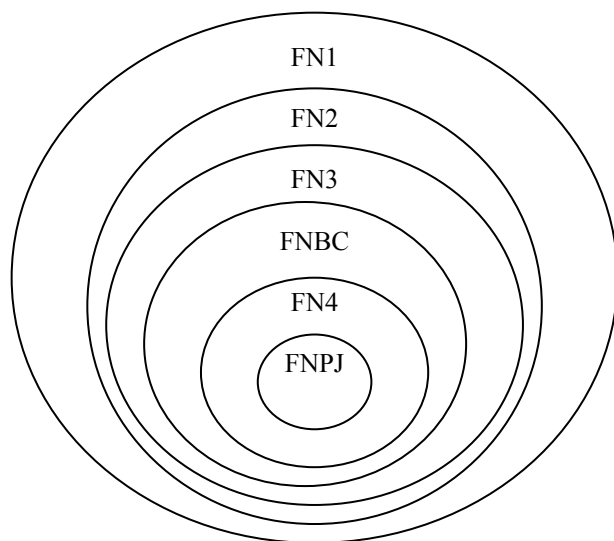


Fig.5.11. Corelația dintre formele normale

În afară de facilitățile pe care ni le oferă o schemă a bazei de date, construită conform rigorilor științifice, normalizarea creează și două probleme: micșorarea eficienței căutării datelor și apariția unor duplicate de date.

Un efect adițional al normalizării este creșterea numărului de structuri de date în baza de date. Aceasta însă afectează eficiența de căutare a datelor în sistemul informatic. Deși normalizarea reduce spațiul total necesar de păstrare a datelor, însă crește timpul în care poate fi căutată informația. Pentru procesarea interpelărilor și extragerii răspunsurilor apare necesitatea rejonționării relațiilor. Prin aceasta și se explică faptul că primele baze de date relaționale au apărut pe calculatoare performante, iar mai târziu au apărut sisteme instalate pe microcomputere.

În ceea ce privește duplicatele de date trebuie de menționat că ele nu pot fi comparate cu redundanța de date ce este redusă în procesul de normalizare. Redundanța generează anomalii de actualizare a datelor. Pe când duplicatele apărute după normalizare, nu generează asemenea anomalii. Aici e vorba de atributele ce formează determinantul dependenței care participă la descompunere. Atributele determinantului apar în ambele scheme produse din schema precedentă.

5.13. Exerciții

- 3.23. Fie mulțimea de dependențe funcționale $G = \{AB \rightarrow EF, A \rightarrow C, D \rightarrow B, C \rightarrow F, F \rightarrow B\}$. Să se determine cheile schemelor de mai jos.
 - (a) $R_1 = ABCDEF$;
 - (b) $R_2 = ABDF$;
 - (c) $R_3 = ACE$;
 - (d) $R_4 = BCD$;
 - (e) $R_5 = DEF$.
- 5.2. Fie mulțimea de dependențe funcționale $G = \{A \rightarrow D, AB \rightarrow E, BF \rightarrow E, CD \rightarrow F, E \rightarrow C\}$ definită pe schema $R = ABCDEF$.
 - (a) Să se determine închiderile oricărei combinații de atribute din schema R .
 - (b) Să se identifice atributele primare.
- 5.3. Să se aducă un exemplu de schemă (alta decât cele descrise în secțiunea curentă), în care se manifestă anomalii de inserare, ștergere și modificare a datelor.
- 5.4. Să se aducă schema din exercițiul 5.3 la forma normală necesară, încât anomaliile să fie eliminate.
- 5.5. Fie pe schema $R = ABCDE$ e definită mulțimea de dependențe funcționale $F = \{A \rightarrow C, B \rightarrow C, C \rightarrow D, DE \rightarrow C, CE \rightarrow A\}$. Să determine dacă descompunerea $R_1=AD, R_2=AB, R_3=BE, R_4=CDE, R_5=AE$ a schemei R , posedă proprietatea joncțiunii fără pierderi.
- 5.6. Fie $F = \{AC \rightarrow BE, BC \rightarrow AD, C \rightarrow DE, A \rightarrow D, D \rightarrow B\}$. Să se determine dacă descompunerea $R_1=ABC, R_2=AB, R_3=BDE$ a schemei $R=ABCDE$, se bucură de proprietatea joncțiunii fără pierderi.

- 5.7. Să se aducă schema relațională $R = ABCDEF$ în forma normală doi, dacă pe ea e definită mulțimea de dependențe funcționale $G = \{AB \rightarrow CE, BC \rightarrow A, C \rightarrow A, ACE \rightarrow B, E \rightarrow DF, BD \rightarrow C, CF \rightarrow BE, CD \rightarrow AF, E \rightarrow F\}$.
- 5.8. Să se descompună schema $R = ABCDEF$ în forma normală trei, dacă pe ea e definită mulțimea de dependențe funcționale $G = \{AB \rightarrow C, C \rightarrow A, BC \rightarrow D, ACD \rightarrow B, CD \rightarrow B, BE \rightarrow C, CF \rightarrow BD, CE \rightarrow AF\}$.
- 5.9. Să se aducă un exemplu de schemă în forma normală trei cu un atribut primar ce depinde tranzitiv de cheie.
- 5.10. Fie $F = \{C \rightarrow T, HR \rightarrow C, CS \rightarrow G, HS \rightarrow R, HRS \rightarrow T\}$. Să se sintetizeze schema bazei de date în forma normală trei.
- 5.11. Să se construiască schema bazei de date în forma normală trei din mulțimea de dependențe $F = \{A \rightarrow CF, B \rightarrow ED, E \rightarrow F, F \rightarrow BC\}$.
- 5.12. Să se aducă un exemplu de relație ce se descompune fără pierderi în trei relații, dar nu se descompune în două. Bineînțeles că toate schemele trebuie să fie diferite.
- 5.13. Fie mulțimea de dependențe funcționale $F = \{AB \rightarrow C, A \rightarrow D, BD \rightarrow C\}$ valide în relația $r(ABCD)$.
- Să se aducă trei exemple de anomalii ce apar în actualizarea relației r .
 - Să se descompună schema acestei relații în două scheme astfel ca schemele obținute să se găsească în forma normală trei și descompunerea să conserve dependențele funcționale.
- 5.14. Fie că relația $r(ABCD)$ satisface mulțimea de dependențe funcționale $F = \{AC \rightarrow B, AB \rightarrow D\}$.
- Să se găsească cheile schemei relației r .
 - Să se arate că schema $R = ABCD$ se găsește în forma normală doi și nu se găsește în forma normală trei.
 - Să se aducă exemple de anomalii ce pot apărea în procesul de actualizare a relației r .
- 5.15. Considerăm schema $R = ABCD$, mulțimea de dependențe funcționale $F = \{A \rightarrow B, C \rightarrow B, D \rightarrow ABC, AC \rightarrow D\}$ definită pe ea și descompunerea lui R în două scheme $R_1 = AB$ și $R_2 = BCD$.
- Descompunerea dată posedă proprietatea joncțiunii fără pierderi? Dacă nu, atunci să se găsească o descompunere fără pierderi.
 - Descompunerea dată conservă mulțimea de dependențe F ?
- 5.16. Considerăm schema $R = ABCD$ și mulțimea de dependențe funcționale $F = \{AC \rightarrow B, AB \rightarrow D\}$.
- Care sunt cheile schemei R ?
 - În ce formă normală cea mai înaltă se găsește schema R ?

- (c) Care este cea mai înaltă formă la care poate fi adusă schema R, dar să posedă proprietatea joncțiunii fără pierderi și să conserve dependențele?
- 5.17. Utilizând algoritmul de normalizare prin descompunere să se aducă schema $R = ABCDEF$ în forma normală Boyce-Codd, dacă pe ea e definită mulțimea de dependențe funcționale $G = \{A \rightarrow B, CD \rightarrow A, CB \rightarrow D, AE \rightarrow F\}$.
- 5.18. Fie schema $R = ABCDEGHIKLMN$ și mulțimea de dependențe funcționale $F = \{AC \rightarrow ED, A \rightarrow BGHL, G \rightarrow HIK, L \rightarrow MN, N \rightarrow L\}$. Să se aducă schema R în forma normală Boyce-Codd prin descompunere, astfel ca descompunerea să posedă proprietatea joncțiunii fără pierderi. Schema normală obținută conservă dependențele?
- 5.19. De ce o schemă relațională cu cel mult două atribute se găsește în forma normală Boyce-Codd?
- 5.20. Să se arate că, dacă pentru orice pereche distinctă de atribute A și B din schema R, atributul A nu se conține în închiderea mulțimii $R \setminus AB$, atunci R se găsește în forma normală Boyce-Codd.
- 5.21. Să se aducă un exemplu de schemă în forma normală Boyce-Codd, dar care nu se găsește în forma normală patru.

BAZE DE DATE ACICLICE

Concluzionând cele descrise în secțiunile precedente, o schemă “bună” a bazei de date trebuie să posede mai multe calități dezirabile. Printre aceste calități putem menționa, în primul rând, formele normale, proprietatea joncțiunii fără pierderi și conservarea dependențelor.

Însă, asupra schemei bazei de date mai pot fi definite niște constrângeri sintactice cum ar fi, spre exemplu, aciclicitatea. Se cunosc diferite tipuri de aciclicitate. Similar unei ierarhii de forme normale ale schemelor, fiecare formă fiind mai restrictivă decât predecesorul, există și o ierarhie de tipuri de aciclicități. După cum se știe, proiectantul bazei de date trebuie să țină cont că, dacă schema relațională nu se găsește în forma normală corespunzătoare, atunci pot apărea diverse probleme de actualizare a bazei de date. De asemenea, de competența proiectantului ține și selectarea gradului de aciclicitate în care dorește ca schema să fie proiectată.

Schemele aciclice se bucură de o serie de proprietăți. Cu cât gradul de aciclicitate este mai înalt, cu atât mai “bună” este schema. Mai mult decât atât, unii algoritmi, ce au o complexitate exponențială asupra schemelor ciclice, asupra schemelor aciclice, sunt polinomiale.

Schemele aciclice ale bazelor de date pot fi caracterizate în diferite moduri. În primul rând, definiția de aciclicitate poate fi formulată prin forme echivalente. Toate aceste forme se bazează pe reprezentarea schemelor bazelor de date cu ajutorul hipergrafurilor. Unele definiții de aciclicități se aduc, utilizând componentele hipergrafurilor în timp ce altele sunt bazate pe grafuri ordinare construite din hipergrafuri.

Multe din proprietățile schemelor aciclice pot fi concepute în calitate de caracteristici, în sens că schema are o proprietate particulară, dacă și numai dacă schema este aciclică. O parte din proprietăți sunt strâns legate de procesarea interpretărilor la baza de date.

6.1. Scheme hipergrafuri

Întrucât schema bazei de date este o mulțime de scheme relaționale, e foarte comod de a asocia schemei bazei de date un hipergraf.

Vom aduce noțiunea de hipergraf. Hipergraful este analogic grafului ordinar neorientat, cu excepția că o muchie a lui nu unește numai două noduri, ci o mulțime arbitrară de noduri.

FURNIZOR	CONTRACT	DATĂ

FURNIZOR	ARTICOL	PREȚ

FURNIZOR	ARTICOL	CONTRACT

Fig.6.1. Schema bazei de date $Db = \{FURNIZOR, CONTRACT, DATA, FURNIZOR\ ARTICOL\ PREȚ, FURNIZOR\ ARTICOL\ CONTRACT\}$

Definiția 6.1. Hipergraful H este o pereche (N, E) , unde N este o mulțime finită de noduri și E o mulțime de muchii (sau hipermuchii), care sunt submulțimi nevide ale mulțimii de noduri N .

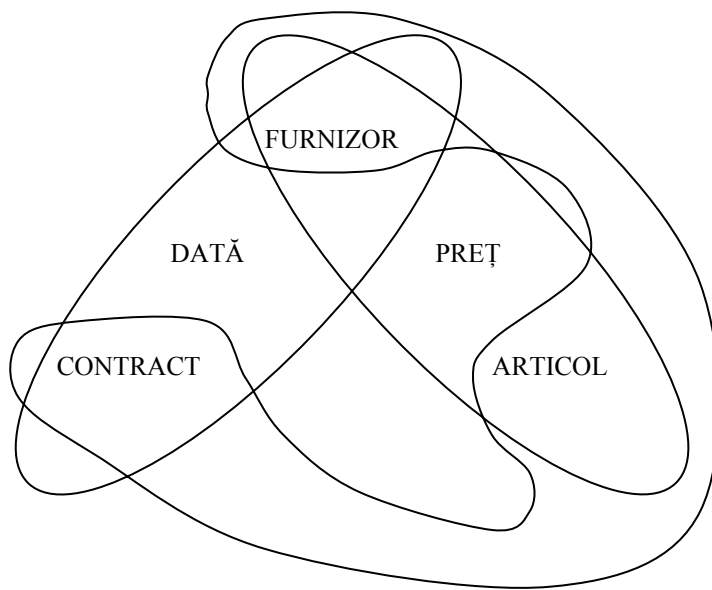


Fig.6.2. Hipergraful schemei bazei de date din fig.6.1

Mai departe vom identifica un hipergraf numai prin menționarea muchiilor sale și implicit vom presupune că mulțimea de noduri este exact mulțimea nodurilor ce aparțin tuturor muchiilor.

Schemei bazei de date îi vom pune în corespondență un hipergraf în felul următor. Mulțimea de atribute U ce formează mulțimea universală este mulțimea de noduri, iar fiecare schemă relațională din schema bazei de date reprezintă o muchie ce include nodurile notate cu atributele din schema relațională. Hipergraful din fig.6.2 corespunde schemei bazei de date din fig.6.1.

Considerăm două scheme ale bazelor de date din fig.6.3 și fig.6.4, fiecare conținând trei scheme relaționale. Unica diferență dintre aceste scheme este că a doua schemă a bazei de date are atributul D în ultima schemă relațională, în timp ce prima schemă a bazei de date conține atributul E . Cu toate că această diferență la prima vedere pare neesențială, în realitate, schema din figura 6.3 este aciclică, iar cea din fig.6.4 e ciclică. Mai departe vom vedea că prima schemă posedă o serie de priorități dezirabile, dar a doua - nu. Pentru o vizualizare a faptului că a doua schemă este ciclică considerăm hipergrafurile corespunzătoare din fig.6.5.

A	B	C

B	C	D

A	E

Fig.6.3. Schema $Db_1 = \{ABC, BCD, AE\}$

A	B	C

B	C	D

A	D

Fig.6.4. Schema $Db_2 = \{ABC, BCD, AD\}$

Nu vom aduce aici definiția aciclicității, vom spune doar că primul hipergraf este aciclic, iar al doilea - ciclic. Adică vom spune că schema Db_1 este aciclică și Db_2 este ciclică.

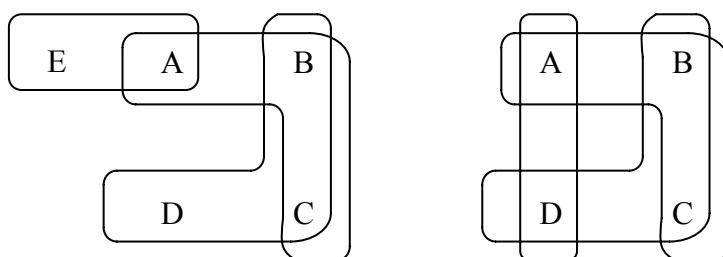


Fig.6.5. Hipergrafurile schemelor Db_1 și Db_2

În acest capitol, nu vom examina toate proprietățile pe care le posedă o schemă aciclică a bazelor de date. Aceste proprietăți au fost studiate de mulți cercetători în diferit context. Cel mai remarcabil fapt este însă că toate aceste proprietăți sunt echivalente (în sens că ele sunt echivalente aciclicității).

6.2. Algoritmul Graham

Există un simplu algoritm de determinare a aciclicității schemei bazei de date. Din considerente pedagogice, expunerea formală a aciclicității o lăsăm pentru secțiunea următoare.

Algoritmul Graham de determinare a aciclicității constă în reducerea pas cu pas a hipergrafului, conform a două reguli până nici una din reguli nu mai poate fi aplicată asupra hipergrafului, reprezentând schema bazei de date.

Dacă în urma aplicării regulilor obținem un hipergraf vid, atunci schema bazei de date este aciclică, în caz contrar este ciclică.

Fie hipergraful $H=(N, E)$. Regulele de reducere sunt următoarele.

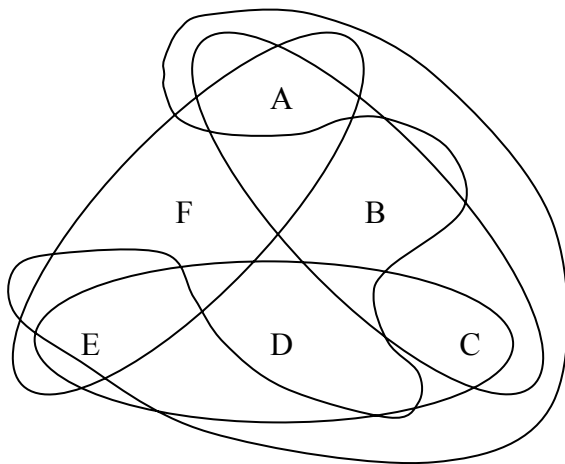


Fig.6.6. Hipergraful schemei $Db=\{ABC,EDC,AEF,ACE\}$

EM (eliminare muchie). Muchia E_i se elimină din E , dacă există o muchie E_j , $i \neq j$, încât $E_i \subseteq E_j$.

EN (eliminare nod). Dacă nodul n_i aparține numai unei singure muchii, el este eliminat din N , deci și din muchia din care face parte.

Exemplul 6.1. Considerăm hipergraful din fig.6.6 și să verificăm, aplicând algoritmul de mai sus, dacă el este aciclic sau nu.

Pentru comoditate, vom reprezenta fiecare muchie a hipergrafului prin nodurile sale amplasate pe o linie, nodurile comune ale muchiilor fiind puse unul sub altul.

Deci algoritmul începe cu considerarea hipergrafului:

A	B	C			
		C	D	E	
A				E	F
A		C		E	

Se aplică regulile EM și EN, până nu mai pot fi făcute modificări asupra hipergrafului.

Aplicăm regula EN, pentru a înlătura nodurile izolate (ce aparțin unei singure muchii). În exemplul nostru, se înlătură nodurile B, D și F, fiind izolate. Au rămas:

A	C	
	C	E
A		E
A	C	E

Cu ajutorul regulii EM, eliminăm muchiile nodurile cărora se găsesc în alte muchii. Așadar, prima muchie AC se conține în ultima ACE și, înlăturând-o pe prima, obținem:

C	E
---	---

A		E
A	C	E

Similar, din aceeași cauză, înlăturăm prima, CE, și a doua, AE, muchii. Atunci obținem un hipergraf format dintr-o singură muchie:

A	C	E
---	---	---

Apelând din nou la regula EN, sunt eliminate nodurile A, C și E.

Am obținut o mulțime vidă. Deci schema bazei de date reprezentată de hipergraful din fig.6.6 este aciclică.

Teorema 6.1. Un hipergraf (sau o schemă) este aciclică, dacă aplicând algoritmul Graham obținem o mulțime vidă.

În legătură că demonstrațiile unor teoreme sunt destul de complexe și necesită cunoștințe suplimentare ce depășesc cadrul acestei lucrări, demonstrațiile nu vor fi aduse. Teoremele vor fi pur și simplu numai formulate.

Exemplul 6.2. Pentru hipergraful din fig.6.7, algoritmul Graham nu produce o

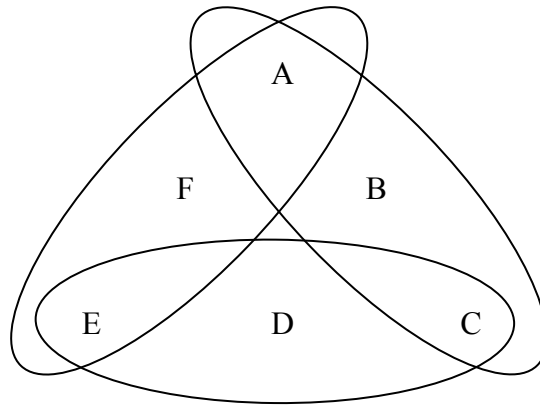


Fig.6.7. Hipergraful schemei $Db = \{ABC, CDE, AEF\}$

mulțime vidă. Deci schema reprezentată de el este ciclică.

Într-adevăr, algoritmul începe, considerând următorul hipergraf:

A	B	C			
		C	D	E	
A				E	F

După înlăturarea nodurilor izolate B, D și F obținem:

A	C	
	C	E
A		E

Aici aplicarea regulilor EM și EN nu mai e posibilă și, prin urmare, hipergraful considerat este ciclic. Deci ciclică este și schema bazei de date asociată acestui hipergraf.

Acest exemplu este instructiv, din punctul de vedere, că el ne demonstrează că nu orice subhipergraf al unui hipergraf aciclic este aciclic. Hipergraful din fig.6.7 este un subhipergraf al hipergrafului din fig.6.6.

Prin subhipergraf al unui hipergraf vom înțelege o submulțime de muchii și noduri al hipergrafului. Acest fenomen contraintuitiv nu are loc pentru grafurile ordinare, adică nu e posibil ca un subgraf al unui graf ordinar aciclic să fie ciclic.

În secțiunile următoare, vor fi considerate alte tipuri de aciclicitate pentru hipergrafuri, în care fenomenul de mai sus nu are loc.

6.3. Consistențe

Vom examina o proprietate a bazei de date ce este echivalentă aciclicității.

Fie schema $Db = \{R_1, \dots, R_m\}$ a bazei de date $db = \{r_1, \dots, r_m\}$. În secțiunile precedente spuneam că verificarea, dacă o schemă a bazei de date posedă proprietatea joncțiunii fără pierderi, este o problemă laborioasă.

Definiția 6.2. Vom spune că baza de date $db = \{r_1, \dots, r_m\}$ este *consistentă*, dacă posedă proprietatea joncțiunii fără pierderi și *consistentă câte două*, dacă orice pereche r_i, r_j de relații din db posedă proprietatea joncțiunii fără pierderi.

r_1	A	B	C
	a_0	b_0	c_1
	a_1	b_0	c_1
	a_2	b_3	c_4

r_2	B	C	D
	b_0	c_1	d_1
	b_3	c_4	d_5

r_3	A	D
	a_0	d_1
	a_1	d_1
	a_2	d_5

Fig.6.8. Baza de date $db = \{r_1(ABC), r_2(BCD), r_3(AD)\}$

Exemplul 6.3. Baza de date din fig.6.8 este joncționabilă fără pierderi fiindcă relațiile r_1, r_2, r_3 sunt proiecțiile relației universale din fig.6.9. E ușor de verificat că și orice două relații din $db = \{r_1, r_2, r_3\}$ sunt joncționabile fără pierderi.

r	A	B	C	D
	a_0	b_0	c_1	d_1
	a_1	b_0	c_1	d_1
	a_2	b_3	c_4	d_5

Fig.6.9. Relația universală $r(ABCD)$

Exemplul 6.4. Să observăm că baza de date din fig.6.10 nu este joncționabilă fără pierderi.

Într-adevăr tuplurile $\langle a_0 \ b_0 \ c_0 \rangle$ și $\langle b_0 \ c_0 \ d_0 \rangle$ din r_1 și r_2 sunt joncționabile, formând tuplul $\langle a_0 \ b_0 \ c_0 \ d_0 \rangle$. Dar, proiectând tuplul $\langle a_0 \ b_0 \ c_0 \ d_0 \rangle$ pe mulțimea de

atribute AD, nu poate fi obținut nici un tuplu din relația r_3 . Orice două relații din fig.6.10 sunt jonctionabile fără pierderi, dar baza de date în întregime nu este jonctionabilă fără pierderi.

Aceasta are loc, fiindcă schema bazei de date $Db=\{ABC, BCD, AD\}$ este ciclică.

r_1	A	B	C
	a_0	b_0	c_0
	a_1	b_1	c_1

r_2	B	C	D
	b_0	c_0	d_0
	b_1	c_1	d_1

r_3	A	D
	a_0	d_1
	a_1	d_0

Fig.6.10. Baza de date $r = \{r_1, r_2, r_3\}$

Teorema 6.2. Dacă schema este aciclică, atunci baza de date este consistentă, dacă și numai dacă e consistentă câte două.

Teorema 6.3. Dacă schema este ciclică, atunci există o bază de date consistentă câte două, dar care nu e consistentă.

Deci, verificarea dacă o bază de date este jonctionabilă fără pierderi, în cazul când schema ei este aciclică, constituie o procedură simplă. Se verifică dacă relațiile ei sunt jonctionabile două câte două. Această procedură are complexitate polinomială, spre deosebire de cazul când schema este ciclică.

6.4. Program semijoncțiune de reducere completă

Să examinăm o problemă legată de altă proprietate echivalentă noțiunii de aciclicitate. Fie că avem o bază de date distribuită, geografic plasată în mai multe orașe, câte o relație în fiecare oraș.

r	A	B	C	D	E	F	G
t_1	a_0	b_0	c_1	d_2	e_3	f_4	g_5
t_2	a_1	b_3	c_9	d_0	e_6	f_3	g_{17}
t_3	a_2	b_1	c_{17}	d_4	e_{19}	f_2	g_8

Fig.6.11. Relația r_1 în orașul O_1

r_2	A	B	H	I	K	L	M
t_1	a_0	b_0	h_{101}	i_{101}	k_{103}	l_{104}	m_{105}
t_2	a_1	b_3	h_{201}	i_{102}	k_{203}	l_{204}	m_{205}
t_3	a_2	b_1	h_{14}	i_{14}	k_3	l_6	m_{47}

Fig. 6.12. Relația r_2 în orașul O_2

De exemplu, fie relația din fig.6.11 se găsește în orașul O_1 , iar cea din fig.6.12 în orașul O_2 .

Presupunem că fiecare din relațiile r_1 și r_2 conțin multe tupluri (sunt prezentate doar câteva). Pentru a obține răspuns la o interpelare ce antrenează atribute din ambele relații, se poate întâmpla că transmiterea datelor între orașe costă mult mai scump, decât prelucrarea datelor în fiecare punct aparte. Deci, vom încerca să soluționăm problema de minimizare a volumului de date, transferate de la un punct la altul.

Transmiterea relațiilor într-un punct, apoi efectuarea operațiilor necesare pentru extragerea răspunsului la interpelare, apoi returnarea răspunsului poate fi foarte inefficientă. La joncțiunea relațiilor pot participa doar o parte de tupluri, celelalte fiind nerelevante interpretării date. În cazul relațiilor noastre la joncțiune participă doar tuplul t_1 din r_1 și tuplurile t_1 și t_2 din r_2 , rezultatul fiind reprezentat în fig.6.13.

r	A	B	C	D	E	F	G	H	I	K	L	M
	a_0	b_0	c_1	d_2	e_3	f_4	g_5	h_{101}	i_{102}	k_{103}	l_{104}	m_{105}
	a_0	b_0	c_1	d_2	e_3	f_4	g_5	h_{201}	i_{202}	k_{203}	l_{204}	m_{205}

Fig. 6.13. Joncțiunea relațiilor r_1 și r_2

Pentru soluționarea acestei probleme se propune așa numita strategie a semijoncțiunii. Vom descrie lucrul acestei strategii pentru exemplul nostru.

Pasul 1. Se taie proiecția relației r_2 din orașul O_2 asupra atributelor AB (acestea sunt attributele comune pentru relațiile din orașele O_1 și O_2 și se transmite în orașul O_1 . Deci, se transmite un singur tuplu $\langle a_0 b_0 \rangle$.

Pasul 2. Se retează relația r_1 din O_1 , eliminând din r_1 acele tupluri ce nu se joncționează cu proiecția pe AB transmisă din orașul O_2 . (Să se observe că rezultatul obținut nu este altceva decât semijoncțiunea r_1 și r_2 , adică $r_1 \leftarrow r_1 | x r_2$).

Pasul 3. Din orașul O_1 relația obținută se transmite în orașul O_2 , unde se produce joncțiunea ei cu relația r_2 .

Fie $bd = \{r_1, \dots, r_m\}$ o bază de date distribuită și fie că către această bază de date este adresată o interpelare. E de dorit ca din fiecare relație să fie înlăturate tuplurile ce nu participă la joncțiunea $r_1 | x \dots | x | r_m$.

Definiția 6.3. Vom numi *reducere completă* a relației r_i mulțimea tuturor tuplurilor din r_i ce participă la joncțiunea $|x|(bd) = r_1 | x \dots | x | r_m$. Orice consecutivitate de atribuire $r_i \leftarrow r_i | x s_j$ se numește *program semijoncțiune*. Dacă pentru orice bază de date bd cu schema Bd și pentru orice relație r_i , programul semijoncțiune pentru relația r_i produce reducerea completă a r_i , atunci acest program semijoncțiune se numește *program semijoncțiune de reducere completă* a schemei Bd .

Deci, dacă schema Bd posedă programul de reducere completă, atunci pot fi complet reduse relațiile r_1, \dots, r_m , înainte de a fi transmise într-un punct comun de prelucrare și calculare a joncțiunii.

Însă într-un sistem distribuit concret, răspunsul la întrebarea, dacă e eficientă sau nu utilizarea unui sau altui program semijoncțiune, depinde de extensiile relațiilor aparte. Calculând $r(R) | x | s(R)$ într-un sistem distribuit, poate fi mai puțin costisitor de transmis toată relația în punctul unde este alocată relația s , decât la început de transmis

$\pi_{R \cap S}(s)$ în punctul de locație a relației r , apoi de transmis $r|Xs$ în punctul s . Pentru o bază de date concretă utilizarea unui program de reducere completă poate fi convenabilă, iar pentru alta - poate să nu fie eficientă.

Exemplul 6.5. Fie $Db = \{ABC, BCD, CD\}$ schema bazei de date reprezentată în fig.6.14. Pentru schema Db există programe de reducere completă. Unul din ele poate fi:

$r_2 \leftarrow r_2 \mid \times r_1;$

$r_3 \leftarrow r_3 \mid \times r_2;$

$r_2 \leftarrow r_2 \mid \times r_3;$

$r_1 \leftarrow r_1 \mid \times r_2.$

Rezultatul acestui program e reprezentat în fig.6.15.

r_1	A	B	C
	a ₇	b ₄	c ₆
	a ₈	b ₄	c ₆
	a ₇	b ₅	c ₆
	a ₉	b ₄	c ₁₁
	a ₈	b ₅	c ₁₁

r_2	B	C	D
	b ₄	c ₆	d ₇
	b ₅	c ₆	d ₇
	b ₄	c ₁₁	d ₉
	b ₅	c ₁₁	d ₉
	b ₄	c ₁₂	d ₉

r_3	C	D
	c ₆	d ₇
	c ₆	d ₉
	c ₈	d ₉
	c ₁₁	d ₉

Fig.6.14. Baza de date $db = \{r_1, r_2, r_3\}$

Exemplul 6.6. Considerăm schema $Db = \{ABC, BCD, CE, DE\}$ a bazei de date prezentate în fig.6.16. Pentru Db nu există un program de reducere completă

r_1^1	A	B	C
	a ₇	b ₄	c ₆
	a ₈	b ₄	c ₆
	a ₇	b ₅	c ₆
	a ₉	b ₄	c ₁₁
	a ₈	b ₅	c ₁₁

r_2^1	B	C	D
	b ₄	c ₆	d ₇
	b ₅	c ₆	d ₇
	b ₄	c ₁₁	d ₉
	b ₅	c ₁₁	d ₉

r_3^1	C	D
	c ₆	d ₇
	c ₁₁	d ₉

Fig.6.15. Reducerea completă $db^1 = \{r_1^1, r_2^1, r_3^1\}$ a bazei de date $db = \{r_1, r_2, r_3\}$ din fig.6.14.

r_1	A	B	C
-------	---	---	---

r_2	B	C	D
-------	---	---	---

a ₁	b ₂	c ₃
a ₇	b ₈	c ₉

b ₂	c ₃	d ₄
b ₈	c ₉	d ₁₀

r ₃	C	E
	c ₃	e ₅
	c ₉	e ₁₁

r ₄	D	E
	d ₄	e ₁₁
	d ₁₁	e ₅

Fig.6.16. Baza de date $db = \{r_1, r_2, r_3, r_4\}$

Teorema 6.4. Dacă schema bazei de date este aciclică, atunci există un program semijoncțiune ce reduce complet toate relațiile în baza de date.

Cu alte cuvinte, dacă schema este aciclică, atunci strategia semijoncțiunii este utilă. Însă, situația e completamente alta în cazul, dacă schema este ciclică.

Teorema 6.5. Dacă schema bazei de date este ciclică, atunci nu întotdeauna există un program semijoncțiune ce reduce complet toate relațiile în baza de date.

Remarcă. Deci, din teoremele 6.4 și 6.5, conchidem că schema bazei de date este aciclică, dacă și numai dacă există un program semijoncțiune de reducere completă a relațiilor în baza de date.

6.5. Expresii joncțiune monotone

Considerăm următorul scenariu. Utilizatorul a decis să joncționeze patru relații r_1 , r_2 , r_3 , și r_4 . Poate să se întâmple următoarele. El joncționează la început r_1 și r_2 , $r_1 \bowtie r_2$, și joncțiunea produce, spre exemplu, o relație cu o mie tupluri. Apoi el poate joncționa rezultatul cu r_3 , adică $r_1 \bowtie r_2 \bowtie r_3$, și obține o relație, să zicem, cu un milion tupluri. În final joncțiunea relațiilor r_1 , r_2 , r_3 , și r_4 , $r_1 \bowtie r_2 \bowtie r_3 \bowtie r_4$, produce zece tupluri. Prin urmare, chiar dacă rezultatul conține doar câteva tupluri, rezultatele intermediare pot fi excesiv de mari.

Trebuie menționat că și în cazul, când relațiile bazei de date sunt complet reductibile, o alegere nereușită a joncțiunilor poate genera dimensiuni foarte mari ale rezultatelor intermediare.

Exemplul 6.7. Considerăm calcularea joncțiunii relațiilor $r_1 \bowtie r_2 \bowtie r_3$, ale bazei de date asupra schemei $Db = \{ABC, BCD, CDE\}$ din fig.6.17. Dacă începem cu calcularea $r_1 \bowtie r_3$, atunci vom obține un rezultat intermediar cu 10 tupluri, în timp ce rezultatul final are 6 tupluri. Însă, dacă joncțiunea începe cu $r_1 \bowtie r_2$, atunci relația din rezultatul intermediar va conține numai șase tupluri.

r ₁	A	B	C
	a ₁	b ₃	c ₅
	a ₁	b ₄	c ₅
	a ₂	b ₃	c ₅
	a ₂	b ₄	c ₆

r ₂	B	C	D
	b ₃	c ₅	d ₇
	b ₄	c ₅	d ₈
	b ₃	c ₅	d ₉
	b ₄	c ₆	d ₈

r ₃	C	D	E
----------------	---	---	---

c ₅	d ₇	e ₁₀
c ₅	d ₈	e ₁₀
c ₅	d ₉	e ₁₁
c ₆	d ₈	e ₁₁

Fig.6.17. Baza de date cu schema Db={ABC, BCD, CDE}

Exemplul 6.8. Considerăm calcularea joncțiunii relațiilor $r_1 \bowtie r_2 \bowtie r_3 \bowtie r_4$, ale bazei de date asupra schemei Db={ABC, BCD, DE, CE}, reprezentată în fig.6.18. Joncțiunea oricăror perechi de relații produce un rezultat intermediar cu mai multe tupluri decât rezultatul final. Să menționăm, însă, că această bază de date este complet reductibilă.

Pe noi ne interesează schemele bazelor de date, ale căror relații complet reductibile pot fi joncționate într-o așa consecutivitate de joncțiuni perechi încât numărul de tupluri în rezultatele intermediare să nu depășească numărul de tupluri în rezultatul final. Mai mult decât atât, noi dorim să avem o astfel de consecutivitate, care ar avea această proprietate pentru toate bazele de date cu schema dată. În realitate noi vom cere satisfacerea unei condiții mai stricte: de fiecare dată când se calculează joncțiunea, relațiile ce participă în joncțiune trebuie să fie complet joncționabile.

r ₁	A	B	C
	a ₁	b ₂	c ₃
	a ₁	b ₂	c ₄
	a ₁	b ₂	c ₅
	a ₁	b ₂	c ₆
	a ₁	b ₂	c ₇

r ₂	B	C	D
	b ₂	c ₃	d ₈
	b ₂	c ₄	d ₈
	b ₂	c ₅	d ₁₁
	b ₂	c ₅	d ₁₂
	b ₂	c ₆	d ₁₆
	b ₂	c ₇	d ₁₇

r ₃	D	E
	d ₈	e ₉
	d ₈	e ₁₀
	d ₁₁	e ₁₃
	d ₁₂	e ₁₄
	d ₁₆	e ₁₅
	d ₁₇	e ₁₅

r ₄	C	E
	c ₃	e ₉
	c ₄	e ₁₀
	c ₅	e ₁₄
	c ₆	e ₁₅
	c ₇	e ₁₅

Fig.6.18. Baza de date db = {r₁, r₂, r₃, r₄}

Definiția 6.4. *Expresie joncțiune* este o expresie formată din scheme relaționale, simbolul " \bowtie " și paranteze, între care orice joncțiune este binară (participă numai două scheme).

Exemplul 6.9. Dacă R₁, R₂, R₃ și R₄ sunt scheme relaționale, atunci ((R₂ \bowtie R₃) \bowtie (R₁ \bowtie R₄)) este o expresie joncțiune ce presupune joncțiunea relațiilor cu schemele R₂ și R₃, joncțiunea relațiilor cu schemele R₁ și R₄, și apoi joncțiunea ambelor rezultate intermediare.

Fie θ o expresie joncțiune asupra tuturor schemelor din Db și db o bază de date cu schema Db . Prin $\theta(db)$ vom subînțelege relația ce rezultă prin substituirea oricărei scheme R_i din θ cu r_i , unde $r_i \in db$ și r_i are schema R_i . De exemplu, dacă $db = \{r_1, r_2, r_3, r_4\}$ și θ este expresia joncțiune $(R_2 | \times | (R_3 | \times | R_2))$, unde r_2 și r_3 au schemele respectiv R_2 și R_3 , atunci $\theta(db)$ este relația $(r_2 | \times | (r_3 | \times | r_2))$, adică relația $r_2 | \times | r_3$.

O subexpresie a expresiei joncțiune se definește în mod obișnuit.

Definiția 6.5. Fie θ o expresie joncțiune, conținând scheme relaționale din Db și fie db o bază de date cu schema Db . Vom spune că θ este *monotonă în raport cu db* , dacă pentru orice subexpresie $(\theta_1 | \times | \theta_2)$ din θ relațiile $\theta_1(r)$ și $\theta_2(r)$ sunt consistente.

Intuitiv, θ este monotonă în raport cu db , dacă nici un tuplu nu este pierdut, când se execută careva joncțiune binară din $\theta(r)$.

Exemplul 6.10. Expresia $((R_2 | \times | R_3) | \times | (R_1 | \times | R_4))$ este monotonă în raport cu $db = \{r_1, r_2, r_3, r_4\}$, unde r_i are schema R_i , $1 \leq i \leq 4$, dacă

- (a) r_2 și r_3 sunt consistente;
- (b) r_1 și r_4 sunt consistente;
- (c) $(r_2 | \times | r_3)$ și $(r_1 | \times | r_4)$ sunt consistente.

Definiția 6.6. Vom spune că expresia θ este monotonă, dacă ea este monotonă în raport cu orice bază de date consistentă câte două relații cu scheme din Db . Dacă θ include exact schemele din Db , atunci spunem că Db are *expresie joncțiune monotonă*.

Expresiile monotone asigură utilizarea eficientă a spațiului de memorie în timpul joncțiunilor, fiindcă nici o joncțiune intermediară nu are mai multe tupluri decât joncțiunea finală.

Teorema 6.6. O schemă a bazei de date este aciclică, dacă și numai dacă există o expresie joncțiune monotonă.

6.6. Scheme α -aciclice

Noțiunea de aciclicitate, utilizată până aici, nu e altceva decât noțiunea de α -aciclicitate.

Definiția 6.7. Fie $H = (N, E)$ un hipegraf și n_1 și n_2 sunt două noduri din N . Vom numi cale din n_1 în n_2 (în hipegraful H) o consecutivitate de $k \geq 1$ muchii (E_1, \dots, E_k) , unde

- (i) $n_1 \in E_1$;
- (ii) $n_2 \in E_k$;
- (iii) $E_i \cap E_{i+1} \neq \emptyset$ dacă $1 \leq i < k$.

Vom spune, de asemenea, că consecutivitatea (E_1, \dots, E_k) este calea din E_1 în E_k .

Definiția 6.8. Două noduri ale hipegrafului H sunt *conexe*, dacă există o cale din unul în altul. O mulțime de noduri sau muchii sunt conexe, dacă orice două noduri sau muchii sunt conexe. *Componenta de conexiune* este mulțimea maximală de muchii conexe.

Exemplul 6.11. Considerăm hipergraful H din fig.6.19. Muchiile ABC , BCD , DE formează o cale din A în E și din ABC în DE , de aceea A și E , precum și ABC și DE sunt conexe. Mulțimile $\{ABC, BCD, DE\}$ și $\{IJ, JKL, IKL\}$ sunt componente de conexiune.

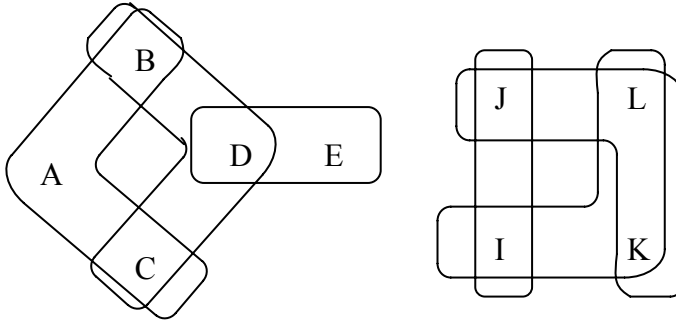


Fig.6.19. Un hipergraf cu două componente de conexiune

Noi vom examina hipergrafurile constituite dintr-o singură componentă de conexiune. Toate construcțiile și rezultatele ulterioare se generalizează și asupra hipergrafurilor cu mai multe componente.

Definiția 6.9. Fie hipergraful $H=(N, E)$. Hipergraful redus (N, E^1) se obține, eliminând din E toate muchiile ce se conțin în alte muchii. Hipergraful se numește *reduc*, dacă el este egal cu reducerea lui, adică nu poate fi eliminată nici o muchie.

Definiția 6.10. Fie E^1 o mulțime de muchii și fie N^1 o mulțime de noduri ce apar în una sau mai multe muchii din E^1 . Vom spune că E^1 este *închisă*, dacă pentru orice E_1 din hipergraf există o muchie E_2 în E^1 , încât $E_1 \cap N^1 \subseteq E_2$.

Exemplul 6.12. Mulțimea de muchii $\{G, H, I\}$ din fig.6.20 este o mulțime închisă. Așadar, intersecția muchiei K cu $G \cup H \cup I$ se conține în muchia H ; similar, intersecția muchiei J cu $G \cup H \cup I$ se conține în G . Însă, mulțimea $\{L, M\}$ nu este închisă, datorită nodurilor x și y . Intersecția muchiei I cu $L \cup M$ nu se conține nici în L nici în M .

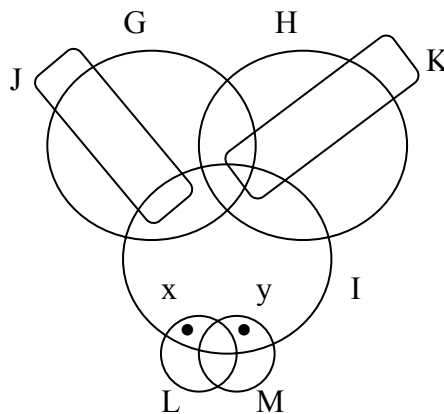


Fig.6.20

Să menționăm că orice mulțime de muchii într-un graf ordinar neorientat este închisă.

Definiția 6.11. Fie E^1 o mulțime conexă, redusă de muchii și fie E_1 și E_2 în E^1 . Fie $Q = E_1 \cap E_2$. Vom spune că Q este o *mulțime de articulație* pentru E^1 , dacă în rezultatul eliminării lui Q din orice muchie din E^1 mulțimea $\{E_i \setminus Q | E_i \in E^1\} \setminus \{\emptyset\}$ nu este conexă.

Definiția 6.12. Un hipergraf redus este α -*aciclic*, dacă orice mulțime închisă și conexă de cel puțin trei muchii are o mulțime de articulație. Un hipergraf se spune că este α -aciclic, dacă reducerea lui este α -aciclică.

Exemplul 6.13. E simplu de verificat că hipergraful din fig.6.6 este α -aciclic. Muchiile lui sunt ABC, CDE, EFA și ACE. O mulțime de articulație pentru toată mulțimea de muchii este $ABC \cap ACE = AC$, fiindcă în urma eliminării lui A și C din fiecare muchie obținem mulțimea de muchii B, DE, EF și E ce nu sunt conexe (B nu e conexă cu celelalte). Să menționăm că mulțimea de muchii $\{ABC, CDE, AFA\}$ nu are nici o mulțime de articulație. Însă, ea nu este nici închisă, deci nu este nici o contradicție cu presupunerea noastră că hipergraful din fig.6.6 este α -aciclic.

Aici putem face o legătură dintre noțiunile de dependențe joncțiune și aciclicitate.

Definiția 6.13. Vom spune că dependența joncțiune $|X|(R_1, \dots, R_m)$ este α -aciclică, dacă α -aciclic este hipergraful format din mulțimea de muchii $\{R_1, \dots, R_m\}$.

În calitate de concluzie, asupra celor spuse de la începutul acestui capitol, putem formula următoarea teoremă.

Teorema 6.7. Următoarele condiții asupra schemei bazei de date $Db = \{R_1, \dots, R_m\}$ sunt echivalente:

- (1) Schema Db este α -aciclică.
- (2) Algoritmul Graham produce o mulțime vidă de muchii.
- (3) Orice bază de date consistentă câte două asupra Db este consistentă.
- (4) Există un program semijoncțiune de reducere completă a bazei de date cu schema Db .
- (5) Schema Db are o expresie joncțiune monotonă.

6.7. Scheme β -aciclice

După cum se știe, un subhipergraf este o submulțime de muchii ale unui hipergraf. Similar, o subschemă a unei scheme a bazei de date este o submulțime de scheme relaționale din schema bazei de date. Am observat că subhipergraful unui hipergraf α -aciclic nu întotdeauna este α -aciclic. Drept exemplu pot servi hipergrafurile din fig.6.6 și fig.6.7.

Ne interesează clasa de hipergrafuri ale căror subhipergrafuri sunt α -aciclice. Astfel de hipergrafuri se numesc β -aciclice. Importanța acestui tip de aciclicitate e greu de subestimat, fiindcă de proprietățile enumerate în teorema 6.7 se va bucura orice subschemă a bazei de date. Deci și interpelările ce implică o parte din schemele relaționale vor fi procesate în mod eficient.

Definiția 6.14. Fie $(E_1, E_2, \dots, E_m, E_{m+1})$ o consecutivitate de muchii ale hipergrafului $H = (N, E)$. Presupunem că muchiile E_1, E_2, \dots, E_m sunt distincte și $E_{m+1} =$

E_1 . Consecutivitatea $(E_1, E_2, \dots, E_m, E_{m+1})$ reprezintă un ciclu simplu, unde $1 \leq i, j \leq m$, $m \geq 3$, dacă $E_i \cap E_j \neq \emptyset$ numai în cazul când $j = i + 1$.

În fig. 6.21, este reprezentat un ciclu simplu, ce constă din 6 muchii.

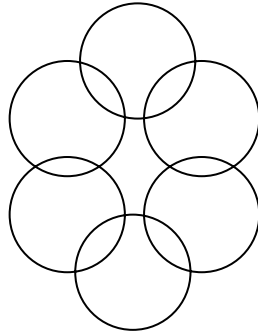


Fig.6.21. Un ciclu simplu

Definiția 6.15. Consecutivitatea de muchii $(E_1, E_2, \dots, E_m, E_{m+1})$ ale hipergrafului $H=(N, E)$ se numește β -ciclu, dacă consecutivitatea $(E_1^1, \dots, E_m^1, E_{m+1}^1)$ este un ciclu simplu, unde $E_i^1 = E_i \setminus X$, $1 \leq i \leq m$ și $X = E_1 \cap E_2 \cap \dots \cap E_m$.

Evident, orice ciclu simplu este un β -ciclu.

Definiția 6.16. Hipergraful $H=(N, E)$ este β -aciclic, dacă el nu conține un β -ciclu, în caz contrar el este β -ciclic. Schema bazei de date $Db=\{R_1, \dots, R_m\}$ este β -aciclică (β -ciclică), dacă hipergraful corespunzător este β -aciclic (β -ciclic).

Definiția 6.17. Consecutivitatea $(E_1, n_1, E_2, n_2, \dots, E_m, n_m, E_{m+1})$ de muchii și noduri ale hipergrafului $H=(N, E)$ se numește β -ciclu slab, dacă sunt satisfăcute condițiile:

- (i) n_1, n_2, \dots, n_m sunt noduri distincte ale lui H ;
- (ii) E_1, E_2, \dots, E_m sunt muchii distincte ale lui H ;
- (iii) $m \geq 3$;
- (iv) $n_i \in E_i \cap E_{i+1}$, și $n_i \notin E_j$ pentru orice $j \neq i, i+1$, $1 \leq i \leq m$.

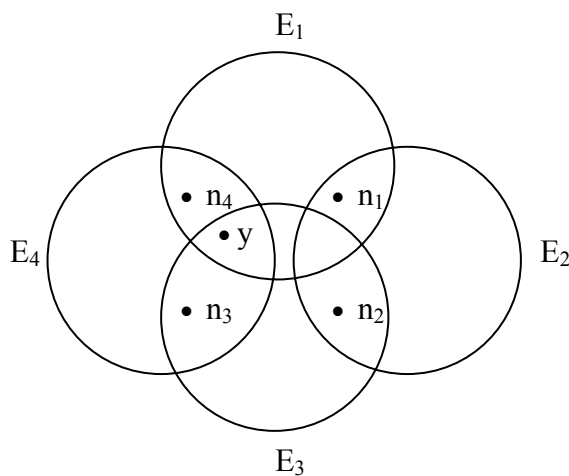


Fig.6.22. Un β -ciclu slab

Teorema 6.8. β -ciclu este un β -ciclu slab.

Demonstrație. Fie consecutivitatea $(E_1, \dots, E_m, E_{m+1})$ e un β -ciclu. Atunci E_1, \dots, E_m , sunt distincte, $E_i \neq E_j$, $m \geq 3$ și pentru orice două muchii vecine E_i și E_{i+1} , $1 \leq i \leq m$, există un nod n_i în $E_i \cap E_{i+1}$, și $n_i \notin E_j$ pentru orice $j \neq i, i+1$.

Afirmația inversă nu este corectă.

Exemplul 6.14. În fig.6.22 consecutivitatea $(E_1, n_1, \dots, E_4, n_4, E_1)$ este un β -ciclu slab, deoarece nodul $y \in E_1 \cap E_3 \cap E_4$ și $y \notin E_2$.

Definiția 6.18. Consecutivitatea de muchii $(E_1, \dots, E_m, E_{m+1})$ ale hipergrafului $H=(N, E)$ se numește Graham-ciclu, dacă sunt satisfăcute condițiile:

- (i) E_1, \dots, E_m sunt muchii distincte și $E_1 = E_{m+1}$;
- (ii) $m \geq 3$;
- (iii) $\Delta_i = E_i \cap E_{i+1} \neq \emptyset$, $1 \leq i \leq m$;
- (iv) pentru orice $1 \leq i, j \leq m$, unde $i \neq j$, mulțimile Δ_i și Δ_j sunt incomparabile, adică $\Delta_i \not\subset \Delta_j$ și $\Delta_j \not\subset \Delta_i$.

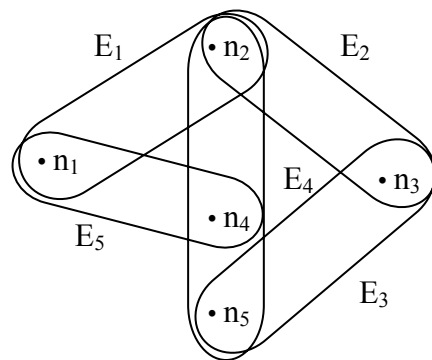


Fig.6.23. Un Graham-ciclu

Teorema 6.9. β -ciclu slab este un Graham-ciclu.

Demonstrație. Fie $(E_1, n_1, \dots, E_m, n_m, E_{m+1})$ un β -ciclu slab. Deoarece condițiile (i), (iv) din definiția β -ciclului slab, implică condițiile (iii), (iv) din definiția Graham-ciclului, iar condițiile (ii), (iii) din definiția 6.17 coincid cu condițiile (i), (ii) din definiția 6.18, atunci consecutivitatea $(E_1, \dots, E_m, E_{m+1})$ este Graham-ciclu.

Afirmația inversă nu este corectă.

Exemplul 6.15. În fig.6.23 consecutivitatea $(E_1, E_2, E_3, E_4, E_5, E_1)$ este un Graham ciclu, dar nu este β -ciclu slab. Unicele β -cicluri slabe sunt (E_2, E_3, E_4, E_2) și (E_1, E_4, E_5, E_1) .

Definiția 6.19. Două muchii ale hipergrafului $H=(N, E)$ se numesc *vecine*, dacă au noduri comune. Două muchii E_2 și E_3 sunt *independente* în raport cu muchia E_1 , dacă ele sunt vecine muchiei E_1 și mulțimile $E_1 \cap E_2$ și $E_1 \cap E_3$ sunt incomparabile.

Menționăm că noțiunea de independență are sens numai atunci, când este vorba de două sau mai multe muchii vecine muchiei date. În particular, muchiile care nu au deloc sau au numai un singur vecin, nu au muchii vecine independente.

Definiția 6.20. Fie hipergraful $H=(N, E)$ și $F \subseteq E$. Mulțimea de muchii F se numește *ciclu independent*, dacă ea este conexă și fiecare muchie $E_i \in F$ are cel puțin două muchii vecine independente, $E_j, E_k \in F$.

O particularitate a ciclului independent constă în faptul că el e conceput ca o mulțime de muchii, și nu ca o consecutivitate de muchii.

Teorema 6.10. Mulțimea de muchii ce formează un ciclu Graham este un ciclu independent.

Demonstrație. Fie consecutivitatea $(E_1, E_2, \dots, E_m, E_{m+1})$ un ciclu Graham. Mulțimea de muchii $\{E_1, \dots, E_m\}$ este conexă, deoarece $\Delta_j = E_j \cap E_{j+1} \neq \emptyset$, $1 \leq j \leq m$. Fiecare muchie E_k , $2 \leq k \leq m$, are în calitate de muchii vecine independente pe E_{k-1} și E_{k+1} , fiindcă $\Delta_k \not\subset \Delta_{k+1}$ și $\Delta_{k+1} \not\subset \Delta_k$. Pentru muchia E_1 , muchiile vecine independente sunt E_2 și E_m .

Afirmația inversă nu este corectă.

Exemplul 6.16. În fig.6.24, mulțimea de muchii $\{E_1, E_2, E_3, E_4, E_5\}$ este un ciclu independent, pe când Graham-cicluri în acest hipergraf sunt (E_1, E_2, E_3, E_1) și (E_3, E_4, E_5, E_3) .

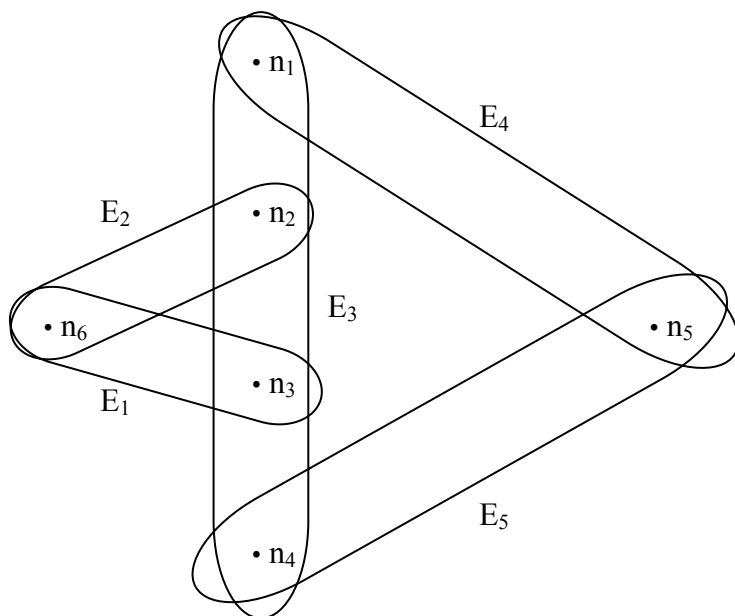


Fig.6.24. Un ciclu independent

Din teoremele 6.8, 6.9 și 6.10 și exemplele 6.14, 6.15, 6.16 urmează că corelația dintre noțiunile de β -ciclu, β -ciclu slab, Graham-ciclu și ciclu independent este cea din fig. 6.25.

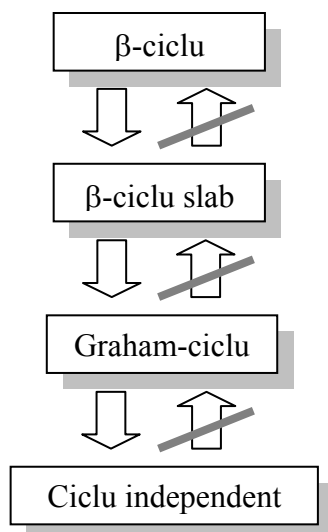


Fig.6.25. Interdependența dintre cicluri

Următoarea teoremă poate fi pusă la baza algoritmului de testare a β -aciclității. Trebuie menționat că cel mai eficient algoritm de determinare, dacă o schemă a bazei de date este β -aciclică, se bazează pe noțiunea de ciclu independent. Demonstrarea teoremei 6.11. și descrierea algoritmului nu se aduce.

Teorema 6.11. Hipergraful $H=(N, E)$ este β -ciclic, dacă și numai dacă el conține

- (a) un β -ciclu, sau
- (b) un β -ciclu slab, sau
- (c) un Graham ciclu, sau
- (d) un ciclu independent.

6.8. Scheme γ -aciclice

O schemă a bazei de date este γ -aciclică (γ -ciclică), dacă hipergraful corespunzător este γ -aciclic (γ -ciclic).

Definiția 6.21. Consecutivitatea $(E_1, n_1, E_2, n_2, \dots, E_m, n_m, E_{m+1})$ se numește γ -ciclu în hipergraful $H=(N, E)$, dacă

- (i) n_1, \dots, n_m sunt noduri distincte în H ;
- (ii) E_1, \dots, E_m , sunt muchii distincte în H și $E_{m+1}=E_1$;
- (iii) $m \geq 3$;
- (iv) $n_i \in E_i \cap E_{i+1}$, $1 \leq i \leq m$ și $n_i \notin E_j$ unde $j \neq i, i+1$ pentru $1 \leq i < m$.

Să observăm că singura diferență dintre definiția γ -ciclu și β -ciclu slab este că condiția " $1 \leq i \leq m$ ", din definiția 6.17(iv) este substituită de condiția " $1 \leq i < m$ " în definiția 6.21(iv). Deseori e comod să ne referim la γ -ciclu, luând în considerație numai consecutivitatea de muchii, făcând abstracție de noduri.

Definiția 6.22. Un hipergraf este γ -ciclic, dacă conține cel puțin un γ -ciclu.

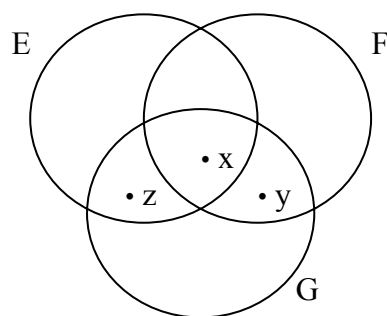


Fig.6.26. Un hipergraf γ -ciclic

Exemplul 6.17. În figura 6.26 este reprezentat un hipergraf γ -ciclic, dar β -aciclic. Într-adevăr, consecutivitatea (E, x, F, y, G, z, E) este un γ -ciclu. Pe de altă parte, $E \cap F \cap G = y$ și eliminând nodul y din fiecare muchie nu obținem un ciclu. Deci consecutivitatea (E, x, F, y, G, z, E) nu este un β -ciclu.

Cea mai elegantă caracteristică a unui hipergraf γ -ciclic este formulată de următoarea definiție.

Definiția 6.23. Un hipergraf este γ -ciclic, dacă el conține sau un γ -ciclu de lungimea 3, sau un ciclu simplu.

Exemplul 6.18. Un ciclu simplu și γ -ciclu de lungimea 3 sunt prezentate în fig.6.21 și 6.26 respectiv. Să menționăm, apelând la hipergraful din fig.2.26, că într-un γ -ciclu de lungimea 3 există cel puțin un nod în intersecția $E \cap F \cap G$, cel puțin un nod în $(E \cap G) \setminus F$ și cel puțin un nod în $(F \cap G) \setminus E$. Alte intersecții cu implicarea muchiilor E, F, G nu pot fi. Deci, dacă există un γ -ciclu de lungimea 3, atunci sau are forma ca în fig.6.26 sau ca în fig.6.21, numai cu trei muchii.

Definiția 6.24. Un hipergraf este γ -ciclic, dacă el conține o pereche E, F de muchii nondisjuncte incomparabile și în hipergraful ce se obține din excluderea intersecției $E \cap F$, din orice muchie restul muchiei E este conexă cu restul muchiei F .

Să arătăm că aceste trei definiții ale hipergrafurilor γ -ciclice sunt echivalente.

Teorema 6.12. Definițiile 6.22, 6.23 și 6.24 sunt echivalente.

Demonstrație. Vom arăta că $(6.22) \Rightarrow (6.23) \Rightarrow (6.24) \Rightarrow (6.22)$, unde prin “(i) \Rightarrow (j)” subînțelegem că, dacă un hipergraf este γ -ciclic conform definiției (i), atunci este γ -ciclic conform definiției (j).

(6.22) \Rightarrow (6.23). Fie H este γ -ciclic, conform definiției 6.22 și să arătăm că H e γ -ciclic și conform definiției 6.23. Fie că $(E_1, n_1, E_2, n_2, \dots, E_m, n_m, E_{m+1})$ este un γ -ciclic de lungime minimală.

Dacă $m=3$, atunci afirmația e demonstrată.

Presupunem că $m \geq 4$. Să arătăm că ciclul de mai sus e un ciclu simplu, adică muchiile vecine se intersectează și muchiile nonvecine din consecutivitate nu se intersectează. Muchiile vecine se intersectează conform definiției γ -ciclu. Să demonstrăm că muchiile ce nu sunt vecine nu se intersectează.

Să arătăm că E_1 nu intersectează un nonvecin. Presupunem că se intersectează. Găsim un k ($3 \leq k < m$) cel mai mic posibil pentru care $E_1 \cap E_k \neq \emptyset$. Fie $v \in E_1 \cap E_k$. Atunci $(E_1, n_1, \dots, E_{k-1}, n_{k-1}, E_{k-1}, E_k, v, E_1)$ e un γ -ciclu mai mic decât cel ipotetic. Aceasta e o contradicție.

Acum să arătăm că E_2 nu intersectează un nonvecin. Pentru aceasta presupunem că $v \in E_2 \cap E_k$ unde $1 \leq k \leq m$. Avem două cazuri.

Cazul 1. $v \in E_3$. Cunoaștem că $v \notin E_1$, deci E_1 nu intersectează nonvecinul E_3 . Găsim r cel mai mare posibil pentru care $v \in E_r$. E ușor de observat că $(E_1, n_1, E_2, n_2, v, E_r, \dots, E_m, n_m, E_1)$ este un γ -ciclu mai mic decât cel ipotetic. Deci e contradicție.

Cazul 2. $v \notin E_3$. Găsim r cel mai mic pentru care $v \in E_r$. E ușor de văzut că $(E_r, v, E_2, n_2, E_3, n_3, \dots, E_r)$ este un γ -ciclu mai scurt decât cel ipotetic. Contradicție.

Am arătat că nici E_1 și nici E_2 nu intersectează nonvecinii. Găsim j cel mai mic pentru care E_j intersectează un nonvecin E_k : fie $v \in E_j \cap E_k$. Atunci $3 \leq j$, și $j+2 \leq k \leq m$. E ușor de văzut că $(E_1, n_1, E_2, n_2, \dots, E_j, v, E_k, \dots, E_{m+1})$ este un γ -ciclu mai scurt decât cel ipotetic. Această contradicție implică $(6.22) \Rightarrow (6.23)$.

(6.23) \Rightarrow (6.24). Fie H e γ -ciclic conform definiției 6.23. Vom arăta că H e γ -ciclic, conform definiției 6.24. Fiindcă H e γ -ciclic conform definiției 6.23, H conține sau un ciclu de lungimea 3 sau un ciclu simplu.

Presupunem că H conține un γ -ciclu de lungimea 3 și fie acest ciclu $(E_1, n_1, E_2, n_2, E_3, n_3, E_1)$. E ușor de verificat că H este γ -ciclic, conform definiției 6.24, (unde E și F din fig.6.26 sunt E_1 și E_3 respectiv).

Acum presupunem că H conține un ciclu simplu. Presupunând că E și F sunt muchii nonvecine în ciclul simplu, vedem că H este γ -ciclic, conform definiției 6.24.

(6.24) \Rightarrow (6.21). Fie H e γ -ciclic conform definiției (6.24). Vom arăta că H e γ -ciclic conform definiției 6.21. Luăm E și F ca în definiția 6.24. și fie că $Q = E \cap F$. Există o consecutivitate (E_1, \dots, E_m) de muchii ce satisface condițiile:

- (i) $E_1 = E$,
- (ii) $E_m = F$,
- (iii) $(E_i \cap E_{i+1}) \setminus Q \neq \emptyset$, unde $1 \leq i \leq m-1$.

Presupunem că muchiile sunt selectate în așa fel că (i)-(iii) sunt satisfăcute pentru cel mai mic m . Dacă $m=2$, atunci conform (ii) $E_2 = F$. Atunci $E_1 \cap E_2 = Q$, ce contrazice condiției (iii) pentru $i=1$. Prin urmare $m \geq 3$. Conform (iii), pentru orice $1 \leq i \leq m-1$ în $(E_i \cap E_{i+1}) \setminus Q$ se găsește câte un nod. Punem E_{i+1} egal E ($=E_1$) și $n_m \in E \cap F$ (din presupunerea că $E \cap F \neq \emptyset$). Să arătăm că consecutivitatea $(E_1, n_1, E_2, n_2, \dots, E_m, n_m, E_1)$ este un γ -ciclu. Nodul n_1 nu se găsește nici într-o muchie E_3, \dots, E_{m-1} , conform minimalității lui m (deci, dacă $n_1 \in E_i$, unde $3 \leq i \leq m-1$, atunci consecutivitatea $E_1, E_i, E_{i+1}, \dots, E_m$ trebuie folosită în locul (E_1, E_2, \dots, E_m) . Mai departe, $n_1 \notin E_m = F$. Deci $n_1 \in E = E_1$; dar $n_1 \neq Q = E \cap F$. Deci n_1 se găsește în E_1 și E_2 , și nu în E_j , $j \neq 1, 2$. Similar, n_1 este în E_i și E_{i+1} dar nu în alt E_j , pentru $1 \leq i \leq m-1$. În particular toate nodurile n_1, \dots, n_{m-1} sunt distincte. În afară de aceasta n_m este distinct de toate nodurile n_1, \dots, n_{m-1} , fiindcă $n_m \in Q$ și $n_i \notin Q$, pentru $1 \leq i \leq m$. Prin urmare, toate nodurile n_1, \dots, n_m sunt distincte. Din condiția că m e minimal urmează că toate muchiile E_1, \dots, E_m sunt distincte. Aceasta e suficient pentru a spune că $(E_1, n_1, E_2, n_2, \dots, E_m, n_m, E_{m+1})$ este un γ -ciclu. Deci H e γ -ciclic, conform definiției 6.22.

6.9. Proprietăți ale schemelor γ -aciclice

De noțiunea de γ -aciclicitate sunt legate o serie de proprietăți ale schemelor, proprietăți ce sunt echivalente acestei noțiuni. Vom examina doar trei proprietăți. Pentru simplitate, vom considera numai schemele bazelor de date hipergrafurile cărora constau dintr-o singură componentă de conexiune.

- (1) *Orice expresie conexă de joncțiune asupra schemei bazei de date este monotonă.* Fie o schemă conexă Db. Echivalența acestei proprietăți cu γ -aciclicitate prezintă interes prin analogie cu teorema 6.6, care ne spune că schema bazei de date Db este α -aciclică atunci și numai atunci când există o expresie joncțiune monotonă asupra Db. Deci echivalența se formulează în felul următor. Schema bazei de date Db este γ -aciclică, atunci și numai atunci, dacă orice expresie joncțiune asupra Db este monotonă (cuvântul “conexă” a fost omis, fiindcă numai o joncțiune monotonă poate fi conexă). Să observăm diferența dintre aceste două afirmații: în cazul α -aciclicității există o asemenea expresie joncțiune monotonă, pe când în cazul γ -aciclicității orice expresie joncțiune este monotonă.

Proprietatea de față garantează o mare libertate în luarea joncțiunilor. Așadar, fie db este o bază de date asupra unei scheme ce se supune proprietății (1) și este consistentă câte două. Presupunem că utilizatorul dorește să facă joncțiunea a unei submulțimi de relații din baza de date db. Conform proprietății (1) el poate joncționa relațiile oricum dorește fără a “dăuna” și este sigur că a acționat în mod eficient. Prin “fără a dauna”, subînțelegem că nici o joncțiune nu va antrena relații cu scheme disjuncte, adică nu va calcula produsul cartezian. Prin “mod eficient”, subînțelegem că nici o joncțiune intermediară nu va avea mai multe tupluri decât joncțiunea finală.

- (2) *Fie schema bazei de date $Db = \{R_1, \dots, R_m\}$. Dependența joncțiune $|X|(R_1, \dots, R_m)$ presupune că orice submulțime conexă din Db posedă proprietatea joncțiunii fără pierderi.* Adică orice dependență joncțiune inclusă $|X|(S_1, \dots, S_k)$, unde $\{S_1, \dots, S_k\} \subseteq \{R_1, \dots, R_m\}$, are proprietatea joncțiunii fără pierderi.

Să menționăm că această proprietate nu e caracteristică schemelor α -aciclice, fiindcă nu orice hipergraf α -aciclic este γ -aciclic.

Exemplul 6.19. Hipergraful din fig.6.2 este α -aciclic și γ -aciclic.

- (3) *În orice bază de date consistentă există o singură asociere între atributele unei mulțimi de atribute.*

Fie $db = \{r_1, \dots, r_m\}$ o bază consistentă cu schema $DB = \{R_1, \dots, R_m\}$. Prin asocierea atributelor din X, unde $X \subseteq R_1 \cup \dots \cup R_m$, subînțelegem o relație $\pi_X(r_{i1} | X | \dots | X | r_{ik})$, unde $\{r_{i1}, \dots, r_{ik}\} \subseteq db$, $X \subseteq R_{i1} \cup \dots \cup R_{ik}$ și $\{R_{i1}, \dots, R_{ik}\}$ este conexă. Adică careva din relațiile bazei de date db sunt joncționate (unde nici o joncțiune nu formează produsul cartezian) și apoi rezultatul proiectat pe mulțimea de atribute X. Proprietatea (3) ne spune că relația rezultat este unică.

<i>serv_funct</i>	FUNCT	DEPT	SAL	
	Ionescu	Cs	150 lei	

<i>info_dept</i>	DEPT	ORAȘ	MGR	
	Cs	Chișinău	Popa	

<i>dom_funct</i>	FUNCT	STRADĂ	ORAȘ	COPII
	Ionescu	V.Lupu	Orhei	Sandu

Fig.6.27.

Exemplul 6.20. Fie schema bazei de date constă din trei scheme relaționale: *serv_funct* cu atributele FUNCT (pentru ”funcționar”), DEPT (pentru “departament”) și SAL (pentru “salariu”); schema relațională *info_dept* cu atributele DEPT, ORAȘ și MGR (pentru “manager”); și schema *dom_funct* cu atributele FUNCT, STRADĂ, ORAȘ, COPII. În fig.6.27 este reprezentată baza de date cu un tuplu în fiecare relație.

În acest exemplu sunt două asocieri {FUNCT, ORAȘ} distincte. Una, care presupune un tuplu <Ionescu, Chișinău> ce ne arată orașul unde funcționarul își are serviciul și alta, cu tuplul <Ionescu, Orhei> ce indică locul de trai al funcționarului. Să menționăm că schema bazei de date este γ -ciclică, chiar α -ciclică.

Să renumim atributul ORAȘ din schema *info_dept* cu OR_SERV și atributul ORAȘ din schema *dom_funct* cu OR_DOM (vezi fig.6.28). Acum avem o singură asociere {FUNCT, OR_SERV} cu un tuplu <Ionescu, Chișinău>, și o singură asociere {FUNCT, OR_DOM} cu un tuplu <Ionescu, Orhei>.

Schema bazei de date din fig. 6.28 este γ -aciclică.

Dat fiind faptul că asocierea dintre atribute într-o schemă γ -aciclică e unică se simplifică esențial forma interpelărilor. Cea mai simplă interpelare în limbajul SQL de găsire a tuturor funcționarilor ce își au serviciul în orașul Chișinău este

```

SELECT FUNCT
FROM serv_funct, info_dept
WHERE serv_funct.DEPT = info_dept.DEPT
AND info_dept.OR_SERV="Chișinău"

```

Dar, ținând cont de proprietatea (3), interpelarea poate fi formulată astfel:

```

SELECT FUNCT
WHERE OR_SERV="Chișinău"

```

Trebuie menționat că avantajele nu constau numai într-o sintaxă mai simplă a interpelărilor, dar și în posibilitatea SGBD-ului de a optimiza procesul de căutare a răspunsurilor cu o flexibilitate mai mare. Sistemul poate exploata faptul că oricare nu ar fi relațiile în joncțiune, joncțiunea va fi monotonă și deci eficientă.

Unele sisteme relaționale folosesc un fișier special care să determine ce relații trebuie să participe la joncțiune pentru a răspunde unei interpelări. Dacă schema bazei de date este γ -aciclică, adică posedă proprietatea (3), nu e nevoie de un așa fișier.

<i>serv funcț</i>	FUNCT	DEPT	SAL
	Ionescu	Cs	150 lei

<i>info dept</i>	DEPT	OR SERV	MGR
	Cs	Chișinău	Popa

<i>dom funcț</i>	FUNCT	STRADĂ	OR DOM	COPII
	Ionescu	V.Lupu	Orhei	Sandu

Fig.6.28.

6.10. Algoritm de testare a γ -aciclicității

Următorul algoritm poate fi utilizat pentru verificarea dacă o schemă este sau nu γ -aciclică. El este similar celui de testare a schemelor α -aciclice.

Algoritmul constă în aplicarea în orice ordine a regulilor (a) – (e) până nu mai poate fi aplicată nici o regulă.

- (a) Nodul izolat (ce aparține unei singure muchii) se elimină din muchie.
- (b) Muchia unitară (ce constă dintr-un singur nod) se elimină.
- (c) Muchia vidă se elimină.
- (d) Dacă două muchii conțin aceleași noduri, se elimină una din muchii.
- (e) Dacă două noduri aparțin acelorași muchii, atunci un nod din cele două se elimină din toate muchiile ce le conțin.

Bineînțeles că aplicarea are un număr finit de pași. Dacă rezultatul final este o mulțime vidă de muchii, atunci hipergraful este γ -aciclic.

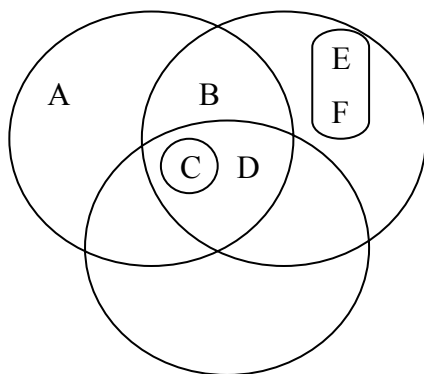


Fig.6.29.

Exemplul 6.20. Să aplicăm algoritmul asupra hipergrafului din fig.6.29. Pentru a simplifica lucrul algoritmului vom considera implicit utilizarea regulilor (d) și (c).

La început hipergraful din fig.6.29 se prezintă:

B C D E F

A	B	C	D		
		C			
		C	D		
				E	F

Nodul A e izolat și muchia {C} e unitară, deci conform regulilor (a) și (b) sunt eliminate. Au rămas muchiile:

B	C	D	E	F
B	C	D		
	C			
	C	D		
			E	F

Nodurile E și F aparțin împreună acelorași muchii (BCDEF și EF) și conform regulii (e) nodul F se elimină din ambele muchii. Similar, nodurile C și D aparțin acelorași muchii. Se elimină nodul D din cele trei muchii. Au rămas muchiile:

B	C	E
B	C	
	C	
		E

Se elimină a treia și a patra muchie, fiindcă sunt unitare:

B	C	E
B	C	

Nodul E e izolat. După eliminarea nodului E au rămas muchiile:

B	C
B	C

Aceste două muchii sunt identice. Conform regulii (d) o muchie se elimină:

B	C
---	---

Întrucât ambele noduri sunt izolate, ele sunt eliminate. În rezultat s-a obținut o mulțime vidă de muchii, deci hipergraful din fig.6.29 este γ -aciclic. Deci și schema asociată lui este γ -aciclică.

6.11. Scheme Berge-aciclice

Vom considera în această secțiune unul din cele mai puternice tipuri de aciclicitate ale schemelor bazei de date și anume Berge-aciclicitatea.

Definiția 6.25. Fie hipergraful $H=(N, E)$. Consecutivitatea $(E_1, n_1, E_2, \dots, E_m, n_m, E_{m+1})$ se numește *Berge-ciclu*, dacă sunt satisfăcute condițiile:

- (i) n_1, \dots, n_m sunt noduri distincte în N ;
- (ii) E_1, \dots, E_m sunt distincte în E și $E_1=E_{m+1}$;
- (iii) $m>1$;
- (iv) $n_i, n_{i+1} \in E_i, 0<i<m+1$.

Definiția 6.26. Hipergraful $H=(N, E)$ este Berge-ciclic, dacă conține cel puțin un Berge-ciclu, în caz contrar e Berge-aciclic.

Exemplul 6.21. Hipergraful din fig.6.30 este Berge-ciclic, dar γ -aciclic. Un Berge-ciclu este consecutivitatea (ABC, B, BCD, C, ABC). Consecutivitatea dată nu e γ -ciclu, fiindcă sunt implicate în ea numai două muchii.

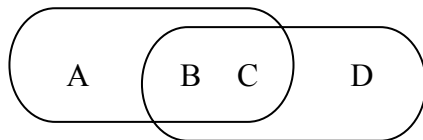


Fig.6.30. Un hipergraf Berge-ciclic

Exemplul 6.21 ne sugerează următoarea teoremă.

Teoremă 6.13. Dacă o pereche de muchii ale unui hipergraf au două sau mai multe noduri comune, atunci hipergraful este Berge-ciclic.

Demonstrație. Afirmatia rezultă imediat din definiția 6.25.

O procedură de testare, dacă o schemă este Berge-aciclică, constă în aplicarea următoarelor două reguli:

- (a) Nodul izolat (ce aparține unei singure muchii) este eliminat din muchie;
- (b) Muchia unitară (ce constă dintr-un singur nod) sau vidă se elimină.

Algoritmul sfârșește, dacă nu mai poate fi aplicată nici una din reguli. Dacă algoritmul produce o mulțime vidă de muchii, atunci hipergraful inițial este Berge-aciclic, în caz contrar e Berge-ciclic.

6.12. Exerciții

- 6.1. Fie schema bazei de date $Db = \{ABC, BCE, CDE\}$. Să se aducă un exemplu de bază de date $db = \{r_1(ABC), r_2(BCE), r_3(CDE)\}$ și a două programe semijoncțiune de reducere completă, astfel că aplicarea unui program reduce costul calculării $r_1 \times r_2 \times r_3$, iar aplicarea altui program nu e eficientă. Să se considere că costul transferării unei valori atomice e 1 și costul calculării $r_i \times r_j$ este egal cu costul transferării proiecției relației r_j asupra schemei relației r_i , iar joncțiunea trebuie să se efectueze în punctul de păstrare a relației r_1 .
- 6.2. Poate să se schimbe relația r în urma atribuirii de forma $r \leftarrow r \times s$, dacă schemele relațiilor r și s sunt disjuncte?
- 6.3. Să se arate că pentru schema bazei de date $Db = \{ABC, BCD, DE\}$ nu există o expresie de joncțiune monotonă.
- 6.4. Să se determine care din următoarele scheme ale bazei de date sunt α -aciclice și care α -ciclice.
 - (a) $Db_1 = \{ABC, CDE, AIE, ACE\}$;
 - (b) $Db_2 = \{ABC, BCD, ACD, ABD\}$;
 - (c) $Db_3 = \{AB, BD, CD, CE, DE\}$.

- 6.5. Să se arate că schema bazei de date $Db = \{ABCD, CDE, DEF, EFG, GK, BCDIL, MIL, EN, NO\}$ este β -aciclică.
- 6.6. Să se determine cel mai înalt graf de aciclicitate a schemelor de mai jos:
- (a) $Db1 = \{ABCD, BCE, CDF\};$
 - (b) $Db2 = \{ABC, CDE, DF, EGH, B\};$
 - (c) $Db3 = \{ABCD, CDEG, BDEF\};$
 - (d) $Db4 = \{CH, ACD, CDF, BDE, DEG, BI\};$
 - (e) $Db5 = \{BCE, ABC, CEG, BEF\};$

CALCULUL RELAȚIONAL

Cu toate că algebra relațională servește drept fundament al unor limbaje de interpelări, majoritatea limbajelor relaționale sunt bazate pe calculul relațional sau tablouri. Cauza principală constă în faptul că algebra relațională este un sistem procedural de operații, adică expresia algebrei relaționale determină o serie de operații asupra relațiilor și ordinea lor de executare (cu exactitatea unor reguli asociative prestabilite).

Calculul relațional reprezintă o adaptare a calculului cu predicate în domeniul bazelor de date relaționale. Ideea de bază a calculului relațional este de a identifica o relație ca un predicat. Deci el eliberează utilizatorul de obligația de a defini cum să obțină rezultatul. Pe baza unor predicate (relații) inițiale, prin aplicarea unor operatori ai calculului predicatelor se pot defini noi predicate (relații).

Sunt cunoscute două variante ale calculului relațional. O variantă utilizează în calitate de valori ale variabilelor asupra relațiilor tupluri de relație. Din această cauză variabilele au fost denumite *variabile tuplu*, iar calculul relațional a primit denumirea de *calcul relațional orientat pe tuplu*. Altă variantă presupune că variabilele sunt definite asupra domeniilor. Aceste variabile se numesc *variabile domeniu*, iar calculul relațional bazat pe acest tip de variabile e cunoscut sub numele de *calcul relațional orientat pe domeniu*.

7.1. Calculul relațional orientat pe tuplu

La început vom considera calculul relațional ce permite definirea relațiilor infinite. Apoi vom introduce modificările necesare ce vor garanta că orice formulă în calculul relațional notează o relație finită.

Formulele în calculul relațional au forma $\{t \mid f(t)\}$, unde t este variabila tuplu, adică variabila ce denotă un tuplu de o lungime fixată, iar f este formula construită din atomi și operatori.

Definiția 7.1. Fie mulțimea universală U de atribute și pentru orice $A \in U$, $\text{dom}(A)$ e mulțimea de valori. Fie mulțimea de operatori de comparație $\Theta = \{=, \neq, <, \leq, >, \geq\}$, schema bazei de date $Db = \{R_1, \dots, R_m\}$, unde $R_i \subseteq U$, $1 \leq i \leq m$ și baza de date $db = \{r_1, \dots, r_m\}$.

Atomii în formula f sunt definiți astfel:

- (1) Valorile de veridicitate, notate cu *true* sau *false* sunt atomi;
- (2) Variabila tuplu t , ce reprezintă un tuplu al relației $r_j(R_j)$, notat $r_j(t)$, este atom, unde r_j este o relație cu schema R_j în baza de date db ;
- (3) $t(A_j)\theta s(A_k)$ este atom, unde t și s sunt variabile tuplu (nu numaidecât distincte), A_j și A_k sunt atribute (nu numaidecât distincte) compatibile din U , θ este operația de comparație, și $t(A_j)$ și $s(A_k)$ sunt respectiv A_j -componenta lui t și A_k -componenta lui s .
- (4) $c\theta t(A)$ și $t(A)\theta c$ sunt atomi, unde c este o constantă în $\text{dom}(A)$, $t(A)$ este A -componenta a variabilei tuplu t , și θ este operator de comparație din Θ .

Exemplul 7.1. Fie schema bazei de date constă din schemele relaționale
funcționari(NUME SALARIU MANAGER DEPARTAMENT),
vânzări(DEPARTAMENT ARTICOL),
furnizări(ARTICOL FURNIZOR),
culori(ARTICOL CULOARE CANTITATE)

și fie expresiile

$$furnizări(t), \quad (7.1)$$

$$vânzări(s), \quad (7.2)$$

$$t(ARTICOL) = s(ARTICOL), \quad (7.3)$$

$$u(DEPARTAMENT) = t(DEPARTAMENT) \quad (7.4)$$

$$s(FURNIZOR) = \text{"Microsoft"}. \quad (7.5)$$

Expresiile (7.1), (7.2), (7.3), (7.4), (7.5) sunt atomi de tipul (2), (2), (3), (3), (4), respectiv. Iar $t(ARTICOL) \neq s(CULOARE)$ nu este atom fiindcă attributele ARTICOL și CULOARE nu sunt compatibile.

Pentru definirea operațiilor calculului relațional sunt utile noțiunile de variabile tuplu libere și legate. Noțiunile acestea au același sens ca și în calculul predicatelor. Neformal vom spune că *variabila tuplu* este *legată* într-o formulă, dacă este calificată existențial sau universal. *Variabila tuplu* se numește *liberă*, dacă nu e calificată.

Noțiunea de variabilă liberă e analogică noțiunii de variabilă globală din limbajele de programare, adică variabilă definită în afara procedurii curente. Variabila legată e similară valorii locale, ce e definită în procedura curentă.

Definiția 7.2. Formulele și variabilele libere și legate în formule se definesc recursiv.

- (1) Orice atom este formulă. Orice variabilă tuplu în cadrul unui atom trebuie să fie liberă.
- (2) Dacă f este formulă, atunci negația lui f , notată $\neg f$, este formulă. Orice variabilă tuplu este liberă sau legată în $\neg f$, dacă este liberă sau legată în f .
- (3) Dacă f_1 și f_2 sunt formule, atunci conjuncția și disjuncția formulelor f_1 și f_2 , notate corespunzător $f_1 \wedge f_2$ și $f_1 \vee f_2$, sunt formule. Orice variabilă tuplu liberă (legată) apărută în f_1 și f_2 sau în ambele formule va rămâne la fel în $f_1 \wedge f_2$ sau $f_1 \vee f_2$. Orice variabilă tuplu, liberă într-o formulă și legată în alta, este liberă sau legată în $f_1 \wedge f_2$ sau $f_1 \vee f_2$ în dependență unde ele apar.
- (4) Dacă variabila tuplu t cu schema R este liberă în formula f , atunci $\forall t(R)f(t)$ și $\exists t(R)f(t)$ sunt formule, unde t este calificată universal și existențial, respectiv. Variabila tuplu t ce e liberă în f devine legată în $\forall t(R)f(t)$ și $\exists t(R)f(t)$. Orice altă variabilă tuplu s , unde $s \neq t$, este liberă sau legată în $\forall t(R)f(t)$ sau $\exists t(R)f(t)$ în dependență cum este în f .
- (5) Dacă f e formulă, atunci (f) e formulă.
- (6) Nimic altceva nu e formulă.

Se presupune următoarea ordine descrescătoare de precedență: operatorii de comparație, calificativele existențial și universal și în sfârșit \neg , \wedge și \vee .

Definiția 7.3. Se numește *expresie al calculului relațional orientat pe tuplu* o construcție de forma $\{t(R) \mid f(t)\}$, unde f este o formulă și t este singura variabilă tuplu liberă cu schema R .

Exemplul 7.2. În calculul relațional orientat pe tuplu uniunea relațiilor $r(R)$ și $s(R)$ se exprimă prin $\{t(R) \mid r(t) \vee s(t)\}$. Această expresie se citește: “mulțimea de tupluri t , ce aparțin relației r sau relației s ”. Să ne amintim că uniunea poate fi realizată, dacă r și s au aceeași aritate. Similar formula $r(t) \vee s(t)$ are sens în aceleași condiții, fiindcă variabila t are aceeași lungime.

Diferența $r(R) \setminus s(R)$ poate fi prezentată de expresia $\{t(R) \mid r(t) \wedge \neg s(t)\}$.

Dacă $r(R)$ și $s(S)$ sunt relații, unde $R = A_1 \dots A_k$, și $S = B_1 \dots B_m$, atunci produsul cartezian $r \times s$ în calculul relațional orientat pe tuplu se scrie

$$\{t(A_1 \dots A_k B_1 \dots B_m) \mid \exists t_r(R) \exists t_s(S) (r(t_r) \wedge s(t_s) \wedge (t(A_1)=t_r(A_1)) \wedge \dots \wedge (t(A_k)=t_r(A_k)) \wedge (t(B_1)=t_s(B_1)) \wedge \dots \wedge (t(B_m)=t_s(B_m)))\}.$$

Expresia de mai sus ne spune că $r \times s$ este o mulțime de tupluri t de aritatea $|R|+|S|$, pentru care există t_r , ce aparține relației r și t_s , ce aparține relației s , și primele componente ale lui t formează t_r , iar celelalte componente formează t_s .

Fie relația $r(R)$ și $X \subseteq R$, unde $X = B_1 \dots B_k$, atunci proiecția $\pi_{B_1 \dots B_k}(r)$ se exprimă astfel:

$$\{t(B_1 \dots B_k) \mid \exists t_r(R) (r(t_r) \wedge (t[B_1]=t_r[B_1]) \wedge \dots \wedge (t[B_k]=t_r[B_k]))\}.$$

Selecția $\sigma_F(r)$ este expresia de forma $\{t(R) \mid r(t) \wedge F^1\}$, unde F^1 este formula F , în care fiecare operând i , ce denotă componenta i e substituită de $t(i)$.

Fie relația $r(AB)$ de aritatea 2. Atunci

$$\{t(AB) \mid \exists t_r(AB) (r(t) \wedge r(t_r) \wedge (t(A) \neq t_r(A) \vee t(B) \neq t_r(B)))\}$$

este o expresie a calculului relațional, ce definește relația r , dacă r are două sau mai multe tupluri, și o relație vidă, dacă r e vidă sau constă dintr-un singur tuplu.

7.2. Expresii bine formate

Ca și expresiile algebrei relaționale, expresiile din calculul relațional orientat pe tuplu reprezintă definiții ale unor relații. În forma prezentată mai sus, aceste expresii permit definirea unor relații cu un număr infinit de tupluri. De exemplu, expresia $\{t(R) \mid \neg r(t)\}$ este mulțimea de tupluri de lungime egală cu aritatea relației r ce nu aparțin r .

Deoarece e greu de precizat “toate tuplurile posibile” se impune excludere unor expresii absurde de tipul celei de mai sus. Aceasta se poate atinge, dacă ne vom limita doar la expresiile bine formate. În astfel de expresii $\{t(R) \mid f(t)\}$, fiecare componentă a lui t ce satisface f trebuie să fie element al lui $\text{dom}(f)$. Mulțimea $\text{dom}(f)$ se definește ca o mulțime de simboluri, care, fie apar explicit în f , fie sunt componentele unor tupluri din relația r , citată în f . Astfel definit $\text{dom}(f)$ este finit întotdeauna.

Exemplul 7.3. Fie $f(t)$ este $t(a) = a \vee r(t)$, unde $r(AB)$ e o relație. Atunci $\text{dom}(f)$ poate fi definită de formula algebrei relaționale $\{a\} \cup \pi_A(r) \cup \pi_B(r)$.

Definiția 7.4. Expresia $\{t(R) \mid f(t)\}$ a calculului relațional orientat pe tuplu este *bine formată*, dacă sunt satisfăcute condițiile:

- (1) Fiecare componentă a lui t , ce satisface f , aparține $\text{dom}(f)$.
- (2) În orice subexpresie de forma $\exists t_1(R) f_1(t_1)$, dacă orice componentă a lui t_1 aparține $\text{dom}(f_1)$, atunci t_1 satisface f_1 .

- (3) În orice subexpresie de forma $\forall t_1(R)f_1(t_1)$, dacă orice componentă a lui t_1 nu aparține lui $\text{dom}(f_1)$, atunci t_1 satisface f_1 .

Condițiile (2) și (3) stabilesc veridicitatea formulelor calificate $\exists t_1(R)f_1(t_1)$ și $\forall t_1(R)f_1(t_1)$, considerând numai t_1 , constituit din simboluri ce aparțin lui $\text{dom}(f_1)$. De exemplu, orice formulă $\exists t_1(R)(r(t_1) \vee \dots)$ satisface condiția (2) și orice formulă $\forall t_1(R)(\neg(r(t_1) \vee \dots))$ satisface condiția (3).

Chiar dacă condiția (3) pare neobișnuită, trebuie să observăm că formula $\forall t_1(R)f_1(t_1)$ este echivalentă formulei $\neg \exists t_1(R) \neg f_1(t_1)$. Ultima expresie nu e bine formată, dacă și numai dacă există un t_1^0 pentru care e adevărată $\neg f_1(t_1^0)$ și t_1^0 nu aparține domeniului formulei $\neg f_1$. Întrucât domeniile lui f_1 și $\neg f_1$ sunt aceleași, condiția (3) afirmă că formula $\forall t_1(R)f_1(t_1)$ e bine formată, dacă e bine formată formula $\neg \exists t_1(R) \neg f_1(t_1)$.

Exemplul 7.4. Fie $f(t)$ e o formulă și fie că orice subformulă a ei de forma $\exists t_1(R)f_1(t_1)$ sau $\forall t_1(R)f_1(t_1)$ este bine formată. Atunci sunt bine formate expresiile de forma $\{t(R) \mid r(t) \wedge f(t)\}$, fiindcă orice tuplu ce satisface $r(t) \wedge f(t)$ aparține relației r , prin urmare, orice componentă a lui t aparține lui $\text{dom}(r(t) \wedge f(t))$. În calitate de exemplu al acestui tip de formule poate fi formula diferenței a două relații $\{t(R) \mid r(t) \wedge \neg s(t)\}$, unde $f(t) = \neg s(t)$. Dacă $f(t) = F^1$, atunci formula exprimă selecția.

Generalizând cele spuse, observăm că este bine formată și formula de forma

$$\{t \mid r_1(t) \vee r_2(t) \vee \dots \vee r_k(t) \wedge f(t)\}.$$

Aici componenta $t(i)$ trebuie să fie un simbol ce apare în componenta i al unui tuplu dintr-o relație r_j . Această formă are, de exemplu, formula ce exprimă uniunea a două relații, adică $\{t(R) \mid r(t) \vee s(t)\}$. În ea lipsește f , fiindcă aici f este identic adevărată cum ar fi, spre exemplu, $t(1) = t(1)$.

O altă expresie bine formată este

$$\{t \mid \exists t_1(R_1) \dots \exists t_k(R_k) (r_1(t_1) \wedge \dots \wedge r_k(t_k) \wedge (t(1) = t_{i1}(j_1)) \wedge \dots \wedge (t(m) = t_{im}(m)) \wedge f(t, t_1, \dots, t_{k3}))\}.$$

Asupra componentei $t(p)$ se aplică restricția că ea trebuie să fie un simbol ce apare în componenta j_p al unui tuplu din R_{ip} . Formula produsului cartezian din exemplul 7.2 are această formă.

7.3. Reducerea algebrei relaționale la calculul relațional orientat pe tuplu

Vom arăta că orice expresie a algebrei relaționale se poate reduce la o expresie a calculului relațional orientat pe tuplu.

Teorema 7.1. Dacă E_a este o expresie a algebrei relaționale, atunci există o expresie, E_t , a calculului orientat pe tuplu echivalentă expresiei E_a .

Demonstrare. Teorema se demonstrează prin inducție după numărul operațiilor în E_a .

Baza inducției: Considerăm zero operatori în E_a . În acest caz, E_a sau este o relație constantă asupra R , adică $\{t_1, \dots, t_n\}$, sau o variabilă r , ce denotă o relație.

În primul caz expresia algebrică E_a e echivalentă expresiei $E_t = \{t(R) \mid t = t_1 \vee \dots \vee t = t_n\}$, unde $t = t_i$ este notația prescurtată pentru $t(A_1) = t_i(A_1) \wedge \dots \wedge t(A_k) = t_i(A_k)$. Este clar că $t(A_i)$ este un simbol ce se prezintă explicit în calitate de componentă a unui tuplu constantă t_j .

În al doilea caz, Ea e echivalentă expresiei $Et = \{t(R) \mid r(t)\}$, care, precum e arătat în exemplul 7.4, este bine formată.

Inducția: Presupunem că teorema a validă pentru expresii algebrice cu mai puțin de k operatori. Fie Ea are k operatori. Atunci avem de considerat cinci cazuri (fiindcă sunt cinci operații de bază ale algebrei relaționale, celelalte deducându-se din ele).

- (1) Uniunea: $Ea = Ea_1 \cup Ea_2$. Din ipoteza inductivă, există $Et_1 = \{t_1(R) \mid f_1(t_1)\}$ și $Et_2 = \{t_2(R) \mid f_2(t_2)\}$ echivalente cu Ea_1 și Ea_2 , respectiv, unde Ea_1 și Ea_2 au mai puțin de k operatori. Atunci Ea este echivalentă expresiei $Et = \{t(R) \mid f_1(t) \vee f_2(t)\}$. Dacă t satisface $f_1(t) \vee f_2(t)$, atunci orice componentă a lui t aparține $\text{dom}(f_1)$ sau $\text{dom}(f_2)$. Întrucât $\text{dom}(f_1(t) \vee f_2(t)) = \text{dom}(f_1) \cup \text{dom}(f_2)$ rezultă că Et e expresie bine formată.
- (2) Diferența: $Ea = Ea_1 \setminus Ea_2$. Fie Ea_1 și Ea_2 au mai puțin de k operatori. Atunci Ea este echivalentă expresiei $Et = \{t(R) \mid f_1(t) \wedge \neg f_2(t)\}$, unde $f_1(t)$ și $f_2(t)$ sunt cele din cazul (1), iar $\neg f_2(t)$ este negația lui $f_2(t)$. Întrucât $\text{dom}(f_1(t) \wedge \neg f_2(t)) = \text{dom}(f_1) \cap \text{dom}(f_2)$ expresia Et este expresie bine formată.
- (3) Produsul cartezian: $Ea = Ea_1 \times Ea_2$. Fie Ea_1 și Ea_2 sunt expresii ale algebrei relaționale cu mai puțin de k operatori. Conform ipotezei inducției există expresii ale calculului orientat pe tuplu $Et_1 = \{t_1(R) \mid f_1(t_1)\}$ și $Et_2 = \{t_2(S) \mid f_2(t_2)\}$ echivalente cu Ea_1 și Ea_2 , respectiv. Atunci Ea este echivalentă expresiei

$$Et = \{t(RS) \mid \exists t_1(R) \exists t_2(S) (f_1(t_1) \wedge f_2(t_2) \wedge (t(R)=t_1(R)) \wedge (t(S)=t_2(S)))\},$$
unde $t(R)=t_1(R)$ este scrierea scurtă pentru $t(A_1)=t_1(A_1) \wedge \dots \wedge t(A_n)=t_1(A_n)$, $R=A_1 \dots A_n$. Este evident că Et este o expresie bine formată.
- (4) Proiecția: $Ea = \pi_{A_{i1} A_{i2} \dots A_{ij}}(Ea_1)$. Fie Ea_1 reprezintă o relație cu schema R , iar A_{i1}, \dots, A_{ij} , sunt attribute din R . Atunci Ea este echivalentă expresiei

$$Et = \{t(A_{i1} \dots A_{ij}) \mid \exists t_1(R) (f_1(t_1) \wedge (t(A_{i1})=t_1(A_{i1})) \wedge \dots \wedge (t(A_{ij})=t_1(A_{ij})))\}.$$
Expresia este bine formată din aceleași considerente că și expresia din cazul (3).
- (5) Selecția: $Ea = \sigma_F(Ea_1)$. Fie Ea_1 reprezintă o relație cu schema R și formula F este aplicabilă. Atunci Ea e echivalentă expresiei $Et = \{t(R) \mid f_1(t) \wedge F^1\}$, unde F^1 este obținută din F substituind orice atribut A_i ce apare în F cu $A_i -$ componenta variabilei tuplu t , $t(A_i)$.

Celelalte operații ale algebrei relaționale se deduc din aceste cinci operații de bază.

Exemplul 7.5. Fie expresia algebrei relaționale

$$Ea = r(AB) \setminus (s(A) \times \pi_B(r(AB))).$$

Expresia calculului orientat pe tuplu echivalentă ei este

$$Et = \{t(AB) \mid r(t) \wedge \neg (\exists t_1(A) \exists t_2(B) s(t_1) \wedge \exists t_3(AB) (r(t_3) \wedge (t_2(B)=t_3(B) \wedge (t(A)=t_1(A)) \wedge (t(B)=t_2(B))))\}.$$

Exemplul 7.6. Fie $r(AB)$ și $s(CD)$ două relații. Să se găsească expresia calculului orientat pe tuplu echivalentă expresiei $Ea = \pi_{AD}(\sigma_{B=C}(r \times s))$ a algebrei relaționale.

Folosind teorema 7.1, expresia calculului orientat pe tuplu echivalentă expresiei $r \times s$ este

$$\{t_3(ABCD) \mid \exists t_1(AB) \exists t_2(CD) (r(t_1) \wedge s(t_2) \wedge (t_3(A)=t_1(A)) \wedge (t_3(B)=t_1(B)) \wedge (t_3(C)=t_2(C)) \wedge (t_3(D)=t_2(D)))\}.$$

Pentru expresia $\sigma_{B=C}(r \times s)$ la formula de mai sus se mai adaugă $t_3(B)=t_3(C)$. Atunci expresia algebrică $\pi_{AD}(\sigma_{B=C}(r \times s))$ este echivalenta următoarei expresii a calculului relațional orientat pe tuplu

$$Et = \{t(AD) \mid \exists t_3(ABCD) \exists t_1(AB) \exists t_2(CD) (r(t_1) \wedge s(t_2) \wedge (t_3(A)=t_1(A)) \wedge (t_3(B)=t_1(B)) \wedge (t_3(C)=t_2(C)) \wedge (t_3(D)=t_2(D)) \wedge (t_3(B)=t_3(C)) \wedge (t(A)=t_3(A)) \wedge (t(D)=t_3(D))))\}.$$

Această expresie nu e cea mai scurtă. Ea poate fi simplificată, dacă se înlătură t_3 și se substituie toate componentele lui t_3 cu componentele lui t_1 și t_2 . Atunci obținem

$$Et = \{t(AD) \mid \exists t_1(AB) \exists t_2(CD) (r(t_1) \wedge s(t_2) \wedge (t_1(B)=t_2(C)) \wedge (t(A)=t_1(A)) \wedge (t(D)=t_2(D))))\}.$$

7.4. Calculul relațional orientat pe domeniu

Calculul relațional orientat pe domeniu utilizează în construcțiile sale aceiași operatori ca și în calculul relațional orientat pe tuplu, dar variabilele, care apar în aceste construcții, sunt definite asupra domeniilor.

Atomul, ca construcție elementară din calculul orientat pe domeniu, se definește recursiv în felul următor.

Definiția 7.5. Atomul în calculul orientat pe domeniu poate avea una din formele:

- (1) Dacă $r_j(A_{i1} A_{i2} \dots A_{ik})$ este o relație în baza de date db, atunci $r_j(a_1 a_2 \dots a_k)$ este formulă atomică, unde orice a_p , $1 \leq p \leq k$, este sau variabilă domeniu cu schema A_{ip} , sau constantă în $\text{dom}(A_{ip})$.
- (2) Dacă a și b sunt variabile domeniu cu aceeași schemă și θ este operator de comparație și c este o constantă cu schema identică lui a , atunci $a\theta b$, $a\theta c$ și $c\theta a$ sunt formule atomice.
- (3) Valorile de veridicitate *true* și *false* sunt atomi.
- (4) Nici o altă formulă nu e atom

Noțiunile de variabile legate și libere în calculul orientat pe domeniu se definesc ca și în calculul orientat pe tuplu.

Expresiile în calculul relațional orientat pe domeniu au forma $\{d_1(A_1) \dots d_k(A_k) \mid f(d_1, \dots, d_k)\}$

Definiția 7.6. Expresia calculului orientat pe domeniu este bine formată, dacă:

- (1) din veridicitatea funcției $f(d_1, \dots, d_k)$ urmează că $d_i \in \text{dom}(f)$;
- (2) $\exists a(A) f_1(a)$ e o subformulă a lui f și din veridicitatea $f_1(a)$ urmează că $a \in \text{dom}(f_1)$;
- (3) $\forall a(A) f_1(a)$ e o subformulă a lui f și din veridicitatea $\neg f_1(a)$ urmează că $a \in \text{dom}(f_1)$.

r	A	B
	a ₁	b ₂
	a ₁	b ₃

Fig.7.1(a).

r	A	B
	a ₁	b ₂

Fig.7.1(b).

Exemplul 7.7. Apelăm din nou la ultima parte a exemplului 7.2, unde trebuia scrisă expresia în termenii calculului orientat pe tuplu, care ar exprima relația $r(AB)$, dacă are două sau mai multe tupluri și o relație vidă în caz contrar. În calculul orientat pe domeniu expresia ar avea forma

$$Ed = \{w(A)x(B) \mid \exists y(A) \exists z(B) (r(wx) \wedge r(yz) \wedge (w \neq y \wedge x \neq z))\}.$$

Într-adevăr, fie $f(w, x, y, z) = r(wx) \wedge r(yz) \wedge (w \neq y \wedge x \neq z)$ și r e relația din fig.7.1(a). Dacă presupunem că $w=a_1$ și $x=b_2$, atunci $\exists y(A) \exists z(B) (f(a_1 b_2, y_1 z))$ este validă, fiindcă prin alegerea lui $y=a_1$ și $z=b_3$ va fi validă formula f . În același mod această formulă e validă pentru $w=a_1$ și $x=b_3$, fiindcă putem selecta $y=a_1$ și $z=b_2$. Prin urmare, ambele tupluri $\langle a_1 b_2 \rangle$ și $\langle a_1 b_3 \rangle$ aparțin relației reprezentată de expresia calculului orientat pe domeniul de mai sus. Dacă, însă, se iau alte valori pentru w și x , atunci subformula $r(w, x)$ din f este falsă. Prin urmare, și formula $\exists y(A) \exists z(B) (f(w, x, y, z))$ e falsă. Deci, dacă în r sunt două sau mai multe tupluri expresia Ed reprezintă relația r .

Fie, acum, r este relația cu un singur tuplu din fig.7.1(b). În acest caz, nici o valoare a lui w și x nu satisface formulele $\exists y(A) \exists z(B) (f(w, x, y, z))$.

Într-adevăr, prima subformulă $r(w x)$ a formulei f e satisfăcută numai pentru $w=a_1$ și $x=b_2$. A doua subformulă $r(yz)$ e satisfăcută numai pentru $y=a_1$ și $z=b_2$, dar, în acest caz, nu e satisfăcută subformula $(w \neq y \wedge x \neq z)$.

Exemplul 7.8. Expresia calculului relațional pe domeniu pentru interpelarea “Care sunt funcționarii care-și au serviciul în departamentul “Soft”?” are forma:

$$Ed = \{n(NUME) \mid \exists s(SALARIU) \exists m(MANAGER) \exists d(DEPARTAMENT) (funcționari(n s m d) \wedge d = \text{”Soft”})\}.$$

Exemplul 7.9. Fie interpelarea: “Care sunt departamentele ce vând articole “imprimantă EPSON” și “imprimantă HP”?”

Expresia în termenii calculului orientat pe domeniu ce corespunde acestei interpelări este

$$Ed = \{d(DEPARTAMENT) \mid \exists a(DEPARTAMENT) \exists b(DEPARTAMENT) (vânzări(a \text{ “imprimantă EPSON”}) \wedge d=a \wedge vânzări(b \text{ “imprimantă HP”}) \wedge d=b)\}.$$

Exemplul 7.10. Fie interpelarea ”Care sunt departamentele ce vând articole “imprimantă EPSON” sau “imprimantă HP” ?” Atunci

$$Ed = \{d(DEPARTAMENT) \mid \exists a(DEPARTAMENT) \exists b(DEPARTAMENT) ((vânzări(a \text{ “imprimanta EPSON”}) \wedge d=a) \vee vânzări(b \text{ “imprimanta HP”}) \wedge d=b))\}.$$

7.5. Reducerea calculului orientat pe tuplu la calculul orientat pe domeniu

Expresia Et a calculului orientat pe tuplu se transformă destul de ușor într-o expresie Ed a calculului orientat pe domeniu.

Fie $\{t(R) \mid f(t)\}$ expresia calculului orientat pe tuplu, unde $R=A_1 \dots A_k$. Atunci:

- (1) orice atom $r(t)$ din f este substituit de $r(d_1 \dots d_k)$, unde d_i este variabila domeniu cu schema A_i .
- (2) orice atom $t(A_i)\theta c$ sau $c\theta t(A_j)$, unde c este componentă a altei variabile tuplu sau o constantă, se substituie de $d_i\theta c$ sau $c\theta d_i$, respectiv, unde d_i este variabila domeniu ce denotă componenta A_i a variabilei tuplu t ;
- (3) orice atom $t_1(A_i)\theta t_2(B_j)$ se substituie de $d_{1i}\theta d_{2j}$, unde d_{1i} și d_{2j} sunt variabile domeniu ce denotă componentele A_i și B_j ale variabilelor tuplu t_1 și t_2 , corespunzător;
- (4) subformula calificată $\exists t(R)f$ este substituită de $\exists d_1(A_1) \exists d_2(A_2) \dots \exists d_k(A_k)f$;

- (5) subformula calificată $\forall t(R)f$ este substituită de $\forall d_1(A_1) \forall d_2(A_2) \dots \forall d_k(A_k)f$;
- (6) $t(R)$ este substituită de $d_1(A_1) d_2(A_2) \dots d_k(A_k)$. Deci d_i primește acele valori pe care le primea $t(i)$.

Este evident că dacă expresia $\{t(R) \mid f(t)\}$ este bine formată, atunci e bine formată și expresia calculului pe domeniu. Vom formula următoarea teoremă fără a o demonstra.

Teorema 7.2. Orice expresie bine formată din calculul relațional orientat pe tuplu are o expresie bine formată echivalentă în cadrul calculului relațional orientat pe domeniu.

Exemplul 7.11. Să examinăm expresia

$$Et = \{t(AD) \mid \exists t_1(AB) \exists t_2(CD) (r(t_1) \wedge s(t_2) \wedge (t_1(B)=t_2(C)) \wedge (t(A)=t_1(A)) \wedge (t(D)=t_2(D)))\}$$

în termenii calculului relațional orientat pe tuplu din exemplul 7.6 și să construim expresia echivalentă în termenii calculului relațional orientat pe tuplu.

Substituim $t(AD)$ cu $d^1(A)$ și $d^2(D)$; $t_1(AB)$ cu $d_1^1(A)$ și $d_1^2(B)$; $t_2(CD)$ cu $d_2^1(C)$ și $d_2^2(D)$. Atunci

$$Ed = \{d^1(A)d^2(D) \mid \exists d_1^1(A) \exists d_1^2(B) \exists d_2^1(C) \exists d_2^2(D) (r(d_1^1 d_1^2) \wedge s(d_2^1 d_2^2) \wedge d_1^2 = d_2^1 \wedge d^1 = d_1^1 \wedge d^2 = d_2^2)\}.$$

Pentru a demonstra echivalența dintre algebra relațională și calculul relațional, este necesar să se demonstreze faptul că, pentru orice expresie bine formată din calculul relațional (expresie pe tuplu sau pe domeniu), există o expresie echivalentă din algebra relațională.

Mai jos vom prezenta doar formularea teoremei fără a aduce demonstrarea ei.

Teorema 7.3. Orice expresie bine formată din calculul relațional orientat pe domeniu are o expresie echivalentă în cadrul algebrei relaționale.

Exemplul 7.12. Fie schema bazei de date constă din schemele relaționale *domiciliu*(PERS_NUME STRADĂ ORAȘ), *serviciu*(PERS_NUME COMPANIE SALARIU), *sediu*(COMPANIE ORAȘ), *manageri*(PERS_NUME MNG_NUME).

Să se scrie expresiile în termenii algebrei relaționale, calculului relațional orientat pe domeniu, calculului orientat pe tuplu pentru următoarea interpelare: “Să se găsească numele de persoane care-și au serviciul la compania MSG”. Expresiile sunt notate cu Ea , Ed și Et , corespunzător.

$$Ea = \pi_{PERS_NUME}(\sigma_{COMPANIE="MSG"}(serviciu))$$

$$Ed = \{n(PERS_NUME) \mid \exists c(COMPANIE) \exists s(SALARIU) (serviciu(n, c, s) \wedge (c="MSG"))\}$$

$$Et = \{t(PERS_NUME) \mid \exists t_1(PERS_NUME COMPANIE SALARIU) serviciu(t_1) \wedge (t_1(PERS_NUME) = t(PERS_NUME)) \wedge (t_1(COMPANIE)="MSG")\}.$$

Exemplul 7.13. Fie schema bazei de date din exemplul 7.12. Să se aducă expresiile Ea , Ed și Et pentru interpelarea “Să se aducă numele și orașele unde locuiesc persoanele care-și au serviciul la compania MSG”. Expresiile sunt următoarele:

$$Ea = \pi_{PERS_NUME, ORAȘ}(\sigma_{COMPANIE="MSG"}(domiciliu \mid x \mid serviciu)).$$

$Ed = \{n(PERS_NUME), o(ORAȘ) \mid \exists st(STRADĂ) \exists s(SALARIU) (domiciliu(n, st, o) \wedge serviciu(n, "MSG", s))\}.$

$Et = \{t(PERS_NUME \ ORAȘ) \mid \exists t_1(PERS_NUME \ STRADĂ \ ORAȘ) \exists t_2(PERS_NUME \ COMPANIE \ SALARIU) (t(PERS_NUME) = (t_1(PERS_NUME)) \wedge (t(ORAȘ)=t_1(ORAȘ)) \wedge (t_1(PERS_NUME)=t_2(PERS_NUMA)) \wedge (t_2(COMPANIE)="MSG"))\}.$

7.6. Exerciții

- 7.1. Fie schema bazei de date din exemplul 7.12. Să se scrie expresiile algebrei relaționale, calculului relațional orientat pe tuplu pentru interpelările:
 - (a) Să se afișeze lista numelor de persoane care nu își au serviciul la compania MSG.
 - (b) Să se afișeze lista numelor de persoane care nu au serviciu.
- 7.2. Să se găsească expresiile calculului orientat pe tuplu echivalente pentru operațiile algebrei relaționale:
 - (a) uniunea;
 - (b) diferența;
 - (c) produsul cartezian;
 - (d) proiecția;
 - (e) selecția;
 - (f) intersecția;
 - (g) θ - joncțiunea;
 - (h) diviziunea.
- 7.3. Să se găsească expresiile calculului relațional orientat pe domeniu echivalente expresiilor calculului relațional orientat pe tuplu, obținute în exercițiul 7.2.