Databases Normalization. Algorithm to Determinate the Closure of Attributes Set

Prof.dr. Nicolaie GIURGIȚEANU
Facultatea de Ştiinţe Economice, Universitatea din Craiova

Designing databases is one of the most laborious activities and so, one of the most boring parts in the life of an informatic system. Of course, the theorists have developed a system of axioms and rules in order to facilitate this activity. The main model of data representation, the relational model, has allowed the introduction into the databases theory, of some very efficient concepts. One of these concepts is that of functional dependency and multivalued dependency. These concepts are fundamental for the normalization of databases. In this paper, we present an algorithm, as a Prolog program, to determine the closure of a set of attributes, and also, another program that establishes whether a set of attributes is a primary key or not.
Keywords: database normalization, attributes set, system of axioms, system of rules.

Introducere

Proiectarea bazelor de date este una din activităţile cele mai plictisitoare din viaţa unui sistem informatic. De buna proiectare a bazelor de date depinde reuşita ulterioară a realizării sistemului informatică. Deşi este o activitate importantă cu care ar trebui să înceapă proiectarea oricărui sistem informatic, de multe ori ea este neglijată de marea majoritate a creatorilor de sisteme informatice şi acest lucru are ca efect o slabă productivitate a muncii.

Conform bibliografiei actuale, cel puţin din punct de vedre teoretic, proiectarea bazelor de date se face în patru paşi şi anume: *specificaţii de cerere*, *proiectare conceptuală*, *proiectare logică* şi *proiectare fizică*.

Specificaţiile de cerere constau în obţinerea tuturor cerinţelor de la utilizatori, iar *proiectarea conceptuală* constă din traducerea acestor cerinţe într-un model conceptual (de exemplu modelul Entităţi-Relaţii alias E-R). Rezultatele acestui pas se concretizează prin realizarea unei *scheme conceptuale*. Acest rezultat este preluat în etapa de *proiectare logică* şi tradus într-un model de reprezentare a datelor (de exemplu modelul relaţional) suportat de sistemul de gestiune a bazelor de date în care vom lucra. Rezultatul etapei de proiectare logică a bazelor de date se concretizează într-o schemă logică, ce va fi tradusă în etapa de *proiectare fizică*, într-o *schemă fizică* adecvată configuraţiei sistemului pe care se va exploata baza de date.

În lucrarea de faţă ne vom ocupa de proiectarea logică a bazelor de date relaţionale şi vom prezenta doi algoritmi, ambii în Visual Prolog, pentru determinarea închiderii unei mulţimi de atribute şi, respectiv pentru stabilirea faptului că o mulţime de atribute este, sau nu, cheie pentru o schemă relaţională dată.

Modelul relaţional este unul din cele mai studiate modele şi, în acelaşi timp, cel mai răspândit model. De fapt, vechile modele, ierarhic şi reţea au fost de mult abandonate, poate noile modele, precum cel orientat obiect să constituie un adversar redutabil pentru cel relaţional.

Una din cele mai importante ramuri ale teoriei bazelor relaţionale este cea care se ocupă cu realizarea schemei relaţionale. La rândul ei această ramură se bazează pe teoria dependenţelor funcţionale şi restricţiilor impuse schemelor relaţionale. Neformal, dependenţa funcţională se defineşte ca fiind proprietatea valorilor unor atribute de a permite reconstruirea, fără echivoc, a valorilor altor atribute.

Armstrong demonstrează că o familie de dependenţe funcţionale (sau echivalent, scheme relaţionale) poate fi descrisă de închiderea unei mulţimi de atribute.

Scopul principal al lucrării noastre este de a

prezenta o modalitate de construire a închiderii unei mulțimi de atribute şi de stabilire a unei chei primare pentru o schemă relațională dată şi pentru o mulțime de dependențe funcționale, de asemenea, dată.

În a doua secțiune a lucrării vom prezenta conceptele, legate de dependențele funcționale, necesare pentru atingerea scopului lucrării. În partea a treia vom definii închiderea unei mulțimi de atribute, iar în partea a patra vom prezenta algoritmii Visual Prolog, pentru determinare închiderii şi, respectiv al stabilirii unei chei primare.

Definiții de bază

În această secțiune vom prezenta, foarte pe scurt, conceptele principale ale teoriei dependențelor funcționale, necesare atingerii scopului propus. Alte concepte pot fi găsite în ă01, 03, 04].

Fie U o mulțime de atribute (de exemplu: nume, adresa, studii etc. - universul discursului). Elementele lui U vor fi notate cu a, b, c, ... , x, y, z sau, dacă este necesară o ordonare pe U, cu a_1, a_2, ... , a_n. În unele cazuri, în locul literelor mici, se preferă literele mari. În teoria bazelor de date universul discursului se mai numeşte şi schemă şi atunci când vorbim de modelul relațional vom spune schemă relațională iar universul discursului va fi notat cu R.

Definiția 2.1. O aplicație *dom* asociază fiecărui atribut $a \in R$ o mulțime de valori, numită domeniu de valori, care este notată cu *dom(a)*.

Definiția 2.2. O relație r peste R este o submulțime a produsului cartesian $\prod_{a \in R} dom(a)$.

Notația 2.1. Dacă relația r are drept schemă relațională pe R, atunci, pentru a preciza acest lucru, vom folosi o notație de forma $r(R)$

Definiția 2.3. Numim tuplu, al unei relații r, un element al produsului Cartesian $\prod_{a \in R} dom(a)$.

Deci o relație poate fi privită ca o mulțime de tupluri, adică r=ăt$_1$(b),t$_2$(b), ..., t$_n$(b)}, unde: $t_i(b) \in \prod_{a \in R} dom(a)$, sau cu alte cuvinte $t_i(b) = (v_1, v_2, \ldots, v_n)$, cu $b=(a_1, a_2, \ldots , a_n)$ şi $t_i(a_j) = v_j \in dom(a_j)$.

Definiția 2.4. O dependendnță funcțională, sau pe scurt o FD, este o expresie de forma $X \rightarrow Y$, cu $X, Y \subseteq R$. Spunem că această FD are loc pe r dacă şi numai dacă oricare ar fi tuplurile t_1 şi t_2 din r, ori de câte ori vom avea $t_1(X)=t_2(X)$, vom avea şi $t_1(Y)=t_2(Y)$.

Cu alte cuvinte, o FD este o restricție, naturală sau impusă, unei scheme relaționale, care permite stabilirea valorilor unei mulțimi de atribute, plecând de la valorile altei mulțimi de atribute. În acest fel avem un criteriu de verificare (validare) a relației.

Observația 2.1. Să observăm că dacă $X \rightarrow R$, atunci, valorile diferite ale lui X, vor identifica în mod unic, valori diferite ale lui R – X, sau cu alte cuvinte X va identifica fără ambiguitate tuplurile lui r. Aceasta înseamnă că X va acționa ca o cheie pe R.

Observația 2.2. Dacă între mulțimile de atribute X şi Y există o dependență funcțională de forma $X \rightarrow Y$, atunci vom spune că X determină funcțional pe Y sau că Y este determinat funcțional de X.

Observația 2.3. Dacă $X \cup Y = R$, atunci dependențele funcționale pot fi folosite pentru eliminarea unuia din cele două tupluri care se repetă.

O altă aplicabilitate a acestei restricții o vom vedea după prezentarea câtorva proprietăți ale dependențelor funcționale.

Proprietăți ale dependențelor funcționale

Să observăm că dacă X, Y şi Z sunt trei mulțimi de atribute cu proprietatea că X determină funcțional pe Y şi Y determină funcțional pe Z, atunci X determină funcțional şi pe Z.

Demonstrarea acestei afirmații este banală şi nu o vom reproduce. Am folosit această afirmație pentru a vedea că dependențele funcționale au o serie de proprietăți. Printre proprietățile cele mai importante ale dependențelor funcționale enumerăm:

A1. Dacă $Y \subseteq X$, atunci $X \rightarrow Y$ (axioma reflexivității).

A2. Dacă $X \rightarrow Y$ şi $Y \rightarrow Z$, atunci $X \rightarrow Z$ (axioma tranzitivității).

A3. Dacă $X \rightarrow Y$ şi V este o mulțime oarecare de atribute atunci $XV \rightarrow YV$ (axioma de dilatare).

A4. (axioma reuniunii) dacă $X \rightarrow Y$ şi $X \rightarrow Z$, atunci $X \rightarrow YZ$.

A5. (axioma descompunerii) Dacă $X \rightarrow YZ$ şi $Y \cap Z = \Phi$, atunci $X \rightarrow Y$ şi $X \rightarrow Y$.

A6. (axioma pseudo-tranzitivităţii) Dacă $X \rightarrow Y$ şi $WY \rightarrow Z$, atunci $XW \rightarrow Z$.

Observaţia 3.1. O mulţime de dependenţe funcţionale poate genera o altă mulţime de dependenţe funcţionale. În acest fel se poate spune că dependenţele funcţionale implică logic alte dependenţe funcţionale.

Definiţia 3.1. Mulţimea dependenţelor funcţionale, logic implicate de mulţimea de dependenţe funcţionale F, o vom numi închiderea lui F şi o vom nota cu F^+.

Acum putem reveni la proprietatea axiomelor lui Armstrong de a fi complete şi stabile. Completitudinea se traduce prin aceea că: cele trei axiome permit determinarea închiderii oricărei mulţimi de dependenţe funcţionale, iar stabilitatea se traduce prin aceea că: aplicarea lor nu duce la apariţia de dependenţe funcţionale inexistente sau parazite.

Observaţia 3.2. Dacă se dă o mulţime de atribute X şi o mulţime de dependenţe funcţionale F pe R, atunci dependenţele logic implicate de F, care folosesc atribute din X, vor duce la determinarea unei mulţimi de atribute logic implicate de X.

Definiţia 3.2. Mulţimea tuturor atributelor logic determinate de X, sub mulţimea de dependenţe funcţionale F, o vom numii închiderea lui X şi o vom nota cu X_F^+.

Definiţia 3.3. Dacă X_F^+ este chiar *R*, atunci vom spune despre X că este o cheie (un identificator) pentru R.

Observaţia 3.3. Dacă X este o cheie pentru *R* atunci orice tuplu al lui *r* va fi identificat fără ambiguităţi de o valoare a lui X.

Aplicaţii ale dependenţelor funcţionale

În secţiunea 2 am văzut una din aplicaţiile dependenţelor funcţionale. În cele ce urmează vom arăta şi alte aplicaţii ale acestora.

Pentru a vedea şi alte aplicaţii ale dependenţelor funcţionale să considerăm schema relaţională *Împrumut=(agenţie, avere, oraş, dosar, client, valoare_împrumut),* dintr-un sistem bancar. Semnificaţia atributelor acestei scheme relaţionale este: *agenţie* – sucursala unei bănci; *avere* - reprezintă capitalul sucursalei respective; *oraş* este oraşul în care-şi are sediul sucursala sau agenţia respectivă; *client* este clientul care face un împrumut la bancă prin agenţia respectivă; *dosar* este numărul dosarului cu care s-a acordat împrumutul, iar *valoare_împrumut* este valoarea împrumutului acordat.

Să vedem ce avantaje ar avea o relaţie construită pe baza acestei scheme relaţionale. În primul rând să observăm că dacă ne-ar interesa valarea împrumutului acordat unui client vom rezolva direct această problemă fără alte prelucrări. De asemenea, dacă ne-ar interesa agenţia care a acordat împrumutul unui client o putem rezolva fără alte prelucrări. Din punct de vedere al rapidităţii obţinerii informaţiei se pare că această schemă relaţională este cea mai bună.

Să analizăm însă şi alte aspecte legate de această schemă relaţională. Pentru aceasta să considerăm relaţia *împrumut* prezentată în figura 1.

Agenţie	Avere	Oraş	Dosar	Client	Val_împ
BCR Iaşi 1	9.000.000	Iaşi	17	Aioanei	2.000
BCR Craiova	8.700.000	Craiova	21	Georgescu	3.000
BCR Iaşi 2	7.500.000	Iaşi	23	Apetrei	2.500
BCR Cluj	9.600.000	Cluj-Napoca	45	Pătru	3.000
BCR Craiova	8.700.000	Craiova	37	Ionescu	2.500
BCR Iaşi 1	9.000.000	Iaşi	32	Sandu	3.500
BCR Oradea	7.500.000	Oradea	45	Manta	2.500

Fig. 1. Relaţia *împrumut*

Se constată că această relaţie suferă de următoarele anomalii.

1. ***repeteţia informaţiei***. Se constată că pentru orice împrumut acordat de o agenţie, trebuie să repetăm averea şi oraşul în care-şi are sediul agenţia respectivă. O redundanţă de informaţie inacceptabilă.

2. ***anomalii de actualizare***. Repetarea informaţiei nu are repercursiuni doar asupra dimensionilor memoriei în care se păstrează informaţia ci are repercursiuni şi asupra actualizărilor. De exemplu să presupunem că se măreşte capitalul agenţiei BCR Iaşi 1. Toate tuplurile care se referă la această agenţie tre-

buie să conţină acelaşi capital. Dacă dintr-un motiv oarecare nu toate tuplurile vor fi aduse la acelaşi capital, atunci am avea o bază de date care oferă capitaluri diferite ale aceleiaşi agenţii. Acest lucru mai este cunoscut şi sub denumirea de inconsistenţă a datelor. Acest lucru vrea să spună că din aceleaşi date nu obţinem aceeaşi informaţie prin prelucrări şi interpretări identice.

3. ***anomalii de inserare - inabilitatea reprezentării informaţiei***. Să presupunem că o cheie primară pentru această relaţie ar fi perechea (agenţie, dosar). Orice nou tuplu inserat în relaţie trebuie să conţină o valoare pentru aceste atribute, deoarece integritatea existenţială cere ca nici o cheie să nu fie parţial sau total NULL. Cu toate acestea dacă cineva vrea să insereze un nou client şi un nou împrumut în baza de date, el nu va putea face acest lucru fără ca acest client să facă un împrumut de la o agenţie oarecare, astfel încât atributele cheii primare să primească o valoare. Deci informaţiile despre un nou client nu pot fi inserate decât dacă clientul face un împrumut. Acest tip de problemă este cunoscut ca anomalii de inserare.

4. anomalii de ştergere – pierdere de informaţii importante. În anumite circumstanţe se poate pierde informaţie utilă prin efectuarea unei ştergeri. De exemplu, dacă dintr-un motiv oarecare clientul care face un împrumut la agenţia BCR Oradea renunţă la acesta, atunci prin ştergerea tupulului corespunzător se va pierde informaţia despre agenţia BCR din Oradea.

Probleme ca cele de mai sus apar în primul rând datorită faptului că relaţia *Împrumut* conţine atât informaţii despre agenţii, cât şi informaţii despre împrumuturile acordate clenţilor. Una din soluţiile folosite pentru eliminarea acestor probleme o constituie descompunerea relaţii şi deci a schemei relaţionale, în două sau mai multe sub-relaţii (subscheme). Descompunerea poate produce beneficii importante mai ales în sistemele distribuite, informaţii distincte pot fi memorate în locuri distincte. Desigur că descompunerea relaţiilor are şi efecte negative asupra timpului de obţinere a informaţiei deoarece relaţiile distincte trebuie legate, în vederea obţinerii unui răspuns corect şi complet.

Din analiza relaţiei de mai sus şi schemei sale relaţionale se constată că există următoarele restricţii de tip dependenţe funcţionale:

agenţie $\rightarrow$ *avere ;*
agenţie $\rightarrow$ *oraş ;*
dosar $\rightarrow$ *val_imp ;*
dosar $\rightarrow$ *agenţie.*

Toate aceste dependenţe sunt naturale, intrinseci problemei respective. Pentru a evita marea majoritate a anomaliilor menţionate mai sus am putea descompune schema relaţională *Împrumut* în următoarele scheme relaţionale:
Agenţii=(agenţie, avere, oraş) şi
Credite=(agenţie, dosar, client, val_imp).
Se poate observa că aceste noi scheme relaţionale elimină o parte din anomalii şi că ele se obţin pe baza dependenţelor funcţionale. În teoria bazelor de date se arată că există o serie de scheme relaţionale cu anumite proprietăţi standard, numite forme normale şi aceste forme normale se obţin pe baza dependenţelor funcţionale.

În secţiunea următoare vom prezenta doi algoritmi descrişi în Prolog, unul pentru determinarea închiderii unei mulţimi de atribute iar al doilea pentru determinarea unei chei de identificare.

Algoritmi Visual Prolog pentru lucru cu FD

Una din formele normale cele mai cunoscute şi mai practice, din teoria bazelor de date, este forma normală Boyce-Codd (BCNF). Pentru descompunerea unei scheme relaţionale non BCNF în scheme relaţionale BCNF vom folosi un proedeu interactiv combinat cu algoritmul descris în Prolog pentru determinarea închiderii unei mulţimi de atribute şi prezentat în figura 2.

```
domains
        lista=symbol*
database
        fd(lista,lista)
predicates
        incarcaBICD
        nondeterm insert(symbol,lista,lista)
        nondeterm concaten(lista,lista,lista)
        nondeterm closure(lista)
        nondeterm incl(lista,lista)
        nondeterm apartine(symbol,lista)
```

```
        nondeterm aflista(lista)
        nondeterm go
        nondeterm append(lista,lista,lista)
        nondeterm qs(lista,lista)
        nondeterm partition(symbol,lista,lista,lista)
clauses
incarcaBICD:-
        /* assertz(fd([agentie],[avere])),
        assertz(fd([agentie],[oras])), */
        assertz(fd([dosar],[val imp])),
        assertz(fd([dosar],[agentie])),
        assertz(fd([dosar],[client])).
closure(X):-
        R=X,    retract(fd(U,V)), incl(U,R),
        concaten(V,R,RR),
        qs(R,R1),       qs(RR,R2),      not(R1=R2),
        consult("BICD"),        closure(R2).
closure(X):-write("["),aflista(X),write("]"),nl.
aflista([]).
aflista([X|R]):-
        write(X,","),  aflista(R).
incl([], ).
incl([X|R],Y):-
        apartine(X,Y), incl(R,Y).
apartine(X,[X| ]).
apartine(X,[ |R]):-apartine(X,R).
concaten([],L,L):-!.
concaten([X|R],L2,L3):-
        insert(X,L2,L4),concaten(R,L4,L3).
insert(X,[],[X]).
insert(X,L,L):-L=[X| ],!.
        insert(X,[Y|R],[Y|L]):-insert(X,R,L).
append([],L2,L2):-!.
append([X|L1],L2,[X|L3]):-
        append(L1,L2,L3).
qs([],[]).
qs([A],[A]).
qs([A, B | R], Ans) :-
        partition(A, [B | R], Left, Right),
        qs(Left, Ls), qs(Right, Rs),
        append(Ls,[A],L1), append(L1,Rs,Ans).
partition( ,[],[],[]).
partition(A,[B | Rest], [B | L], R) :-
        A > B, partition(A, Rest, L, R).
partition(A,[B | Rest], L, [B | R]) :-
        A <= B, partition(A, Rest, L, R).
go:-
        incarcaBICD,save("BICD"),write("Inchiderea lui [",aflista([dosar]),
        write("] este "),closure([dosar]).
goal go.
```

Fig. 2. Algoritm pentru determinarea închiderii unei mulțimi de atribute

Rulând programul prezentat în figura 2 vom obține închiderea mulțimii de atribute formată din atributul *dosar* şi se constată că aceasta este chiar schema relațională *Credite*. Am adăugat, în acest program încă o dependendență funcțională pe care o impunem noi şi anume *dosar* → *client*. Modificând programul de mai sus prin renunțarea la comentarii şi calculând închiderea atributului *agenție* se constată că aceasta este chiar schema relațională *Agenții*. Algoritmul pentru determinarea unei chei de identificare este prezentat în figura 32.

```
domains
        lista=symbol*
database
        fd(lista,lista)
        schema(lista)
predicates
        incarcaBICD
        nondeterm go
        incarcaschema
        nondeterm insert(symbol,lista,lista)
        nondeterm concaten(lista,lista,lista)
        nondeterm closure(lista)
        nondeterm incl(lista,lista)
        nondeterm apartine(symbol,lista)
        nondeterm aflista(lista)
        nondeterm append(lista,lista,lista)
        nondeterm qs(lista,lista)
        nondeterm partition(symbol,lista,lista,lista)
        nondeterm verificaKey(lista)
clauses
incarcaschema:-
        assert(schema([a,b,c,g,h,i])).
incarcaBICD:-
```

```
        assertz(fd([c,g],[h])),       assertz(fd([a],[b])), assertz(fd([a],[c])),
        assertz(fd([b],[h])), assertz(fd([c,g],[i])).
closure(X):-
        R=X,
        retract(fd(U,V)),     incl(U,R),     concaten(V,R,RR),
        qs(R,R1),      qs(RR,R2),     not(R1=R2),
        consult("BICD"),      closure(R2).
closure(X):-verificaKey(X).
aflista([]).
aflista([X|R]):-
        write(X,","),
        aflista(R).
incl([], ).
incl([X|R],Y):-
        apartine(X,Y),
        incl(R,Y).
apartine(X,[X| ]).
apartine(X,[ |R]):-apartine(X,R).
concaten([],L,L):-!.
concaten([X|R],L2,L3):-
        insert(X,L2,L4),concaten(R,L4,L3).
insert(X,[],[X]).
insert(X,L,L):-L=[X| ],!.
        insert(X,[Y|R],[Y|L]):-insert(X,R,L).
verificaKey(L1):-
        schema(L),     qs(L1,L2),     qs(L,L3),      L2=L3,
        write(" este identificator al schemei relationale n ["),
        aflista(L),write("]"),nl.
verificaKey( ):-
        schema(L),
        write(" nu este identificator al schemei relationale •n ["),
        aflista(L),write("]"),nl.
go:-
        incarcaschema,incarcaBICD,save("BICD"),write("["),aflista([dosar]),
        write("]"),closure([dosar]).
append([],L2,L2):-!.
        append([X|L1],L2,[X|L3]):-
                append(L1,L2,L3).
        qs([],[]).
        qs([A],[A]).
        qs([A, B | R], Ans) :-
                partition(A, [B | R], Left, Right),
                qs(Left, Ls), qs(Right, Rs),
                append(Ls,[A],L1), append(L1,Rs,Ans).
        partition( ,[],[],[]).
        partition(A,[B | Rest], [B | L], R) :-
                        A > B, partition(A, Rest, L, R).
        partition(A,[B | Rest], L, [B | R]) :-
                        A <= B, partition(A, Rest, L, R).
goal go.
```

Fig. 3. Algoritm pentru determinarea unei chei de identificare

Se poate constata, rulând acest algoritm, că *agenție* este o chei de identificare pentru schema relațională *Agenții*, iar *dosar* este chei de identificare pentru schema *Credite*.

Bibliografie

- Giurgițeanu N., *Proiectarea bazelor de date relaționale.* SITECH, Craiova 1997
- Giurgițeanu, N., Popa, S., *Using Prolog to verify the Functional and Multivalued Dependencies*, International Workshop IE&SI, Timişoara, 23-24 Mai 2003, 157-162
- Elmasri R., Navathe S., *Fundamentals of database Systems*. Benjamin/Cummings, 1994
- Codd E.F. A relational model for large shared data banks. Comm. ACM 13 (1970), 377-387.