Tatjana Welzer
Ivan Rozman
József Györkös
University of Maribor

UDK 681.3.06 PROLOG:519.24

After a brief presentation of base and some definitions neded, the aim of the article is to deal with the normalization of relational database.The implementation of normalization process will be shown by PROLOG, the fifth generation language. The ascendancy of this language over the languages of the fourth generation is that the program is not defined as a sequence of steps but as a discription of relations between objects. This enables a logical conclusion in questions that are put in various ways. The built-in database of PROLOG will be exploited as well.

Po kratki predstavitvi relacijske baze in definicij, ki jih potrebujemo v nadaljevanju, bomo v članku predstavili proces normalizacije relacijske baze podatkov. Izvedbo procesa normalizacije bomo predstavili s PROLOG-om, jezikom pete generacije. Prednost tega jezika pred jeziki četrte generacije je v tem, da program ne definiramo kot zaporedje korakov, temveč kot opis relacij med objekti, kar omogoča logično sklepanje ob različno zastavljenih vprašanjih. Prav tako pa bomo izkoristili tudi PROLOG-ovo vgrajeno bazo podatkov.

1.Ø INTRODUCTION

One of the most important modules for various computer applications is database. The database can be a hierarchical database, a network database or a relational database. The latter is more and more often used instead of the first two databases and is successfully gaining ground on all fields off application. Its advantage is above all in its simple and user friendly transfer from the paper to the computer. Successful work with the relational database can be ensured only by a thoughtful formulation of base. Because of its extensiveness this formulation should be aided by a corresponding tool.

2.Ø RELATIONAL DATABASE

Relational database is a temporally variable multitude of relations /ALAG84/. R(A1,A2,....,An) is the relation over multitudes D1,D2,...,Dn if the submultitude of Descartes product is as follows:

$$R \subseteq \{ D1 \times D2 \times \ldots \times Dn \}$$

D1,D2,....,Dn mean the domains of attributes. The attributes A1,A2,...,An denote the structure of the relation which is called **relational schema.**

2.1.Ø RELATIONAL ALGEBRA

Relational algebra is a simple formal language which enables data manipulation /CODD7Ø/. Relational algebra consists of operations over multitudes (union, intersection and difference) and special operation (projection,selection,join and division). For our further work, operation of projection is of main importance.

There is the following relation: R(A1,A2,...,An). X is a submultitude of the multitude of attributes X $\subseteq$ {A1,A2,...,An}, Y is the complement {A1,A2,...,An}/ X. The relation R(A1,A2,...,An) can be written as follows: R(X,Y). The operation of **projection** of the relation R according to attributes X is marked as follows: **R[X]** and defined:

$$R[X] = \{x \mid \exists y : (x,y) \in R(X,Y) \}$$

2.2.Ø LOGICAL DEPENDENCES

2.2.1 FUNCTIONAL DEPENDENCE

There is relation R(A1,A2,...,An) and submultitudes of the multitude of attributes X $\subseteq$ {A1,A2,...,An} and Y $\subseteq$ {A1,A2,...,An}. R[XY] denotes the projection of the relation R according to attributes from X and Y. The

functional dependence X --> Y exists if and only if it is valid for R[XY] in every moment that there is a functional dependence R[X] --> R[Y].
The functional dependence X --> Y is said to be **complete** if for every real submultitude X' (X' ⊆ X) it is valid that X' -/-> Y. If it is valid that X' --> Y then the functional dependence X --> Y becames a **partial functional dependence.**

2.2.2 KEY

R(A1,A2,...,An) forms a relation. X which is a submultitude of the multitude of attributes is said to be the **key** of the relation R if and only if the following two conditions are fulfilled:

(i) X determines functionally all attributes of the relation R, X --> Ai for i = 1,...n.

(ii) no real submultitude of the multitude X possesses this characteristic, for X' ⊆ X X' -/-> Aj is valid 1<j<n.

If such X ⊆ {A1,A2,...,An} does not exist, that is valid X --> Ai, for i = 1,...,n, then the **key** of the given relation is a complete multitude of attributes A1,A2,...,An.

2.3.Ø NORMAL FORMS

Normal forms are rules on the joins of attributes into relations in which logical dependences are taken into account /DATE86/. Taking these rules into consideration the iregularities (anomalisms) of data input, deleting and updating are avoid.

2.3.1 THE FIRST NORMAL FORM

The relation R is in the **first normal form** if and only if the values in the domains are atomic for every attribute A in the relation R.

2.3.2 THE SECOND NORMAL FORM

Let X be the multitude of all attributes R(A1,A2,...,An), which are not a part of the key of the relation R. It is said that the relation R is in the **second normal form** if and only if each attribute from X is **completely functionally dependend** on each key of the relation R(A1,A2,...,An).

If the relation R(A1,A2,...,An) is not in the second normal form, then their exists a **decomposition** of the relation R(A1,A2,...,An) into a multitude of relations which are in the second normal form. The relations obtained in this way can be united again into a previous relation by means of the operation of natural join.
There is the relation R(A1,A2,...,An) which does not exist in the second normal form. The relation R can be recorded equivalently in the form R(X,Y,Z). In this case X means a multitude of key attributes and Y means a multitude of non-key attributes. X --> Y forms a partial functional dependence. The multitude Z covers all the remaining attributes of the relation R(A1,A2,...,An) which exist neither in the multitude X nor in the multitude Y.

The multitude X can be shown by X = X'X''. Then it is valid: X' --> Y. It is a complete functional dependence.

If the relation **R(X,Y,Z)** is supplanted by the projection **R[XZ]** and **R[X'Y]**, then the projection R[X'Y] is in the second normal form because X' --> Y forms **full functional dependence**. It should be found in which normal form the projection **R[XZ]** exists and, if necessary, this projection should be decomposed in the same way as **R(X,Y,Z)**. The procedure is final because after each decomposition of relation the relations with less number of attributes are obtained.

The previous statements can be confirmed by the following demonstration:

$$R[X'Y] *x' R[XZ] = R[X'Y] *x' R[X'X''Z] =$$

$$= R[X'YX''Z] = R[X'X''YZ] = R[XYZ] = R(X,Y,Z)$$

3.Ø APPLICATION OF PROLOG IN THE NORMALIZATION PROCES

3.1 NOTATION OF RELATION IN PROLOG

The description of structured of individual relations is presented by a relational schema. To model our relational schema the structured analaysis tools of DeMarco /DEMA79/ and Gane and Sarson /GANE79/ (data flowcharts, data dictionary, data store) were used. The result of modeling is the relational schemas wich are used for data storage by the relational database management system. The relation (the relational schema filled up with data) exits, depending on the choice of method in the first normal form or unnormalized.

The tables obtained were inrespective of their normal form, stored in the form of PROLOG structure. PROLOG's built-in database ensures a basic mechanism for data storage and data access. The notation of relation, that is of the whole relational base is a simple one. The data obtained from a relation are record in the form of PROLOG's facts, consisting of predicates and attributes. The name of relation is written in predicate; the values of attributes obtained from relational scheme are written in attributes. After the input of all data into the relational schema we can see that there is a record on the screen of the terminal which is equivalent to the record on the paper. This means that copying the relation from the paper to the screen is 1:1 (one-to-one).

3.2 IMPLEMENTATION OF PROJECTION BY MEANS OF PROLOG

The table into which the required columms were translated and in which the redundant lines were deleted is the result of operation **projection**.

The presented relations are described by means of PROLOG and in this way the program implementing a projection of suitable relation is designed.

```
* Program Projection *

projection(List_of_solutions,Solutions):-
find_equal(List_of_solutions,Solutions).

find_equal([],[]).

find_equal([H|T],Solutions):-
member(H,T),
find_equal(H,Solutions).

find_equal([H|T],Solutions):-
not_memeber(H,T),
write(T),
nl,
find_equal(T,[H|Solutions]).
member(X,[]):-!.
memeber(X,[_|Y]):- memeber(X,Y).
```

```
not_memeber(X,[]):-!.

not_memeber(X,[Y|Z]):- X \== Y,
                       not_memeber(X,Z).

findall(X,H,_):- asserta(found(mark)),
                 call(H),
                 asserta(found(X)),
                 fail.

findall(_,_,L):- collect_found([],M),
                 !,
                 L = M.

collect_found(S,L):- getnext(X),
                     !,
                     collect_found([X|S],L).
collect_found(L,L).

getnext(X):- retract(found(X)),
             !,
             X \== mark.
```

In the working version of the program Projection the implementation of the operation is released by the predicate **findall** and **projection**.

```
?-findall([chosen_atributes*],
  name_of_relation*(all_atributes*),Solutions)),
  projection(Solutions,M).
```

In the relation chosen all possible solutions are looked for, first. Then in the list of solutions (List_of_solutions) all redundant lines are excluded so that equal records (find_equal) are looked for which are not placed (not_member) on the list of final solutions (Solutions).

3.3.Ø CHECKING THE RELATION IN THE LIGHT OF ITS NORMAL FORM

3.3.1 UNNORMALIZED RELATIONS

According to the definition from Chapter 2.3.1, the relation exists in the first normal form if and only if the values in domains are atomic. That is the values in the domain are not lists or sets of values or composite values. In practice such records are found very often. If there is an unnormalized relation over wich we want to perform certain operations, then the relation should be translated into the first normal form.

For the normalization of such an unnormalized table the following is required:

- to check the existence of redundant lines and to exclude them,
- to find out whether the values of individual attributes in the domain are recorded as multitudes or lists. If such records exist, they should be translated into a corresponding form.

The program of record in PROLOG is based on checking the elements in the structure of list into which the previously written relation has been translated. When the number of elements in individual lists is found out, the paralel lying elements are joined into lines which are then transmited to the screen of the terminal in their normalized form.

---

* In the operation chosen_atributes, name_of_relation and all_atributes are substituted with the real names of atributes and relation.

```
* Program Normalization *

normalization(A):- A == tab,!.
normalization(A):- nonvar(A),
                   A \==  tab,
                   A =.. X,
                   write(A),
                   write('.'),
                   nl,
                   change(X),
                   read(A1),
                   normalization(.A1).

normalization(A):- read(A1),
                   normalization(A1).


change([H|Tail]):- P = H,
                   split(P,Tail,List_of_goals).

split(P,Tail,List_of_goals):-
count_arg(Tail,N),
count_el_arg(Tail,K),
[H|T]=Tail,
join_rest([H|T],N,1,S,K,1,List_of_goals),
form_goals(P,G,List_of_goals).
count_arg([],-1).
count_arg([G|T],N) :- count_arg(T,N1),
                      N is N1 + 1.

count_el_arg([H,[H1|[]]|T],1).
count_el_arg([H,[H1|X]|T],K):-
count_el_arg([H,X|T],K1),
K is K1 + 1.

join_rest(_,_,_,S,K,M,S1) :- M2 is K + 1,
                             M2 = M,
                             reverse_list(S,O),
                             S1=O,!.

join_rest(Tail,N,N1,List_of_goals,K,M,S1) :-
join_el(Tail,[],N,N1,M,S),
M1 is M + 1,
join_rest(Tail,N,1,[S|List_of_goals],K,M1,S1).

form_goals(_,_,[]) :- !.
form_goals(P,H,[H1|T1]) :-
reverse_list(H1,H2),
X =..[P,H,H2],
tab(3Ø),
write(P),
write('('),write(H),copy(G2),write(').'),
nl,
form_goals(P,G,R1).

copy([]) :- !
copy([G|R]) :- write(','),
               write(H),
               copy(T).

join_el(L1,L2,N,N1,M,List) :- N2 is N + 1,
                              N2 = N1,
                              List = L2,!.

join_el([H|T],T1,N,N1,M,S)  :-
append_link(T,N1,M,T2),
N2 is N1 + 1,
join_el([H|T],[T2|T1],N,N2,M,S).

append_link(T,N1,M,X) :- n_link(N1,T,Arg),
                         n_link(M,Arg,X).

n_link(1,[H|T],C) :- C = H, !.
n_link(N,[H|T],C) :- N1 is N - 1,
                     n_link(N1,T,C).

reverse_list([],[]).
reverse_list([H|T],L) :- reverse_list(T,L1),
                         append(L1,[G],L).

append([],L,L).
append([H|RT,L,[H|T1]) :- append(T,L,T1).
```

3.3.2.Ø STATING THE SECOND NORMAL FORM

In Chapter 2.3.Ø we pointed out that in inputting, deleting and updating irregularities can occur if normal forms are not taken into account. An irregular input can prevent to add new lines if all the values for attributes are not known. In deleting the lines the information on certain attributes can be lost while the updating of one attribute value requires a change in all relations where this attribute appears.

3.3.2.1 STATING THE LOCAL AND PARCIAL FUNCTIONAL DEPENDENCE

According to the definition in chapter 2.2.1, the existence of functional dependence is stated by checking the values of attributes that form the multitudes X and Y, between which we would like to shate the existence of functional dependence (X --> Y). It is valid that for each value x, x ¢ X there is exactly one y, y ¢ Y.

To state the functional existence by means of PROLOG the program Projection is used first to design the relation **R[XY]**. Then in this relation the condition **R[X] --> R[Y]**, according to chapter 2.2.1 is fulfilled. By means of a modified predicate find_equal from the program Projection equal x's are searched. If they exists then the value of belonging y's we can conclude to state the existence of functional dependence.

According to Chapter 2.2.1 the functional dependence can be a full or a partial one. To state this characterictic the multitude atribute X should be decomposed into real submultitudes (X', X'', X''',....). Then the existence of partial dependence between the real submultitudes (X', X'', X''',...) and the multitude of attributes Y should be checked according to the procedure described.

A rule to decompose the list (multitude X) into sub-list or elements (real submultitudes of the multitude X) should be added to the program in PROLOG.

3.3.3 NORMALIZATION OF RELATION INTO THE SECOND NORMAL FORM

The relation which was found out that it was not in the secon normal form should be decomposed into several tables that would meet the condition on second normal form.

According to the definition in Chapter 2.3.2, the relation R(A1,A2,...,An) is decomposed into two tables **R[XZ]** and **R[X'Y]** which form the result of the projection. The relation **R[X'Y]** exists in the second normal form because the following condition should be met: X' --> Y forms full functional dependence.
To normalize the relation into the second normal form given programs for projection and full functional dependence are used.

To check the relation according to its third normal form similar stepts (required for the third normal form) should be carried out.

4.Ø CONCLUSION

The described process of normalization in PROLOG is performed by IJS PROLOG on the main frame computer VAX 88ØØ. The local information system of research group was chosen as a model. The data on research group members, their activities, working results and the tools used by the member are recorded.

The fifth generation language **PROLOG** was chosen because of its simple copying of relation into the structure of PROLOG's facts and definition of rules (records of program) which implemented certain actions within the normalization process.

To bring the application nearer to the circle of potential users (also to those possessing no knowledge how to program) the application will be transfered to a Personal Computer by means of operation system DOS (TURBO PROLOG Borland, Inc. will be used). In this development phase, the application offers above all an aid for good design of relational database which provides the existence of successfull information system.

5.Ø LITKRATURK

COOD7Ø E.F.Codd: A Relational Model of Data for Large Shared Data Banks, Communication of the ACM, Volume 13, No.:6, June 197Ø

CLOC84 W.F.Clocshin, C.S.Mellish: Programming in Prolog, Springer-Verlag, Berlin 1984

DEYI84 Deyi Li: A Prolog Database System, Research Studies Press LTD,1984

DATE83 C.J.Date: Database: A Primer, Addison-Wesley Publishing Company, 1983

DATE86 C.J.Date: An Introduction to Database Systems, Volume 1, Addison - Wesley Publishing Company, 1986

KELL87 R.Keller: Expert System Technology Development and Appplication, Yourdon Press, Prentice-Hall Company Englewood Cliffs, NJ 1987

MARC86 C.Marcus: Prolog Programming, Addison-Wesley Publishing Company, 1986

WAH86 B.W.Wah, Guo-Jie Li: Tutorial: Computers for Artifical Intelligence Applications, IEEE 1986