International Journal of Computer Applications (0975 – 8887)
Volume 20– No.8, April 2011

Minimization of Functional Dependencies

R N.Kulkarni
Dept. of Information Science & Engineering
Ballari Institute of technology & Management
Bellary - 583104.

Archana B.A
Dept. of Information Science & Engineering
Ballari Institute of Technology & Management
Bellary - 583104.

H Naga Sirisha
Dept. of Information Science & Engineering
Ballari Institute of technology & Management
Bellary - 583104.

B.S Vasundhara Takur
Dept. of Information Science & Engineering
Ballari Institute of technology & Management
Bellary - 583104.

ABSTRACT

Nowadays many organizations are maintaining computer based information systems. These information systems are valuable assets to the organization. Most of the business information or corporate decisions are buried across the systems in the organization and due to the need based modifications sometimes the attributes are scattered throughout the program and even there is a redundancy in the stored data. These business information and corporate decisions represents the business rules of the organization and they are in the form of functional dependencies. These functional dependencies are unevenly scattered and sometimes redundant too. In a database, the records containing these unevenly scattered functional dependencies may be distributed throughout the database, leading to anomalies.

This paper proposes a methodology for the minimization of the functional dependencies available either in a program code or in a database using the minimal cover process. By minimizing these functional dependencies, the redundant and irrelevant attributes are removed and the structure of the application program is kept intact in the maintenance phase.

Keywords

Functional dependencies, minimal cover, minimization, business rules.

1. INTRODUCTION

Computers today are playing a vital role in business processes. They are used to keep track of day to day transactions, perform business calculations, etc. Business forms are used as main input to the process of derivation of a set of functional dependencies [8] i.e. all the business rules are represented in the form of functional dependencies with in a program or database. Such rules are often invisible in a program, since they are distributed (buried) across hundreds or thousands of lines of code [5].

Due to the need based maintenance approach being followed in organisations today, perennial updation is done in order to cope with the advancement of technologies in the areas of storage, processing, Graphical User Interfaces [1] etc. However, perennial updation leads to the uneven scattering of functional dependencies and irrelevant documentation. Sometimes it also leads to redundancy of attributes, which in turn occupy more storage space.

The business rules which are inherent within the program have to be maintained without any loss of information or the structure of the program code. In case of systems that have a function-oriented design, the functionality has to be abstracted from modules. A module is a logically separable part of a program [2]. When the modules are loosely coupled, they are modifiable. For modules to be loosely coupled (good principle of software engineering) the interdependencies between the modules have to be minimized. Usually, the loosely coupled modules have greater cohesion. Cohesion of a module represents how tightly the internal elements are bound to one another [2]. A module can be made highly cohesive when the functional dependencies within it are minimized and the redundant attributes (elements) are eliminated. A module that is highly cohesive and also has low coupling with other modules is said to be functionally independent of other modules [7].

In case of data bases, the records might contain attributes that are unevenly scattered and moreover these records might also be distributed. This may lead to discrepancies in the storage and create anomalies. Normalization is done in order to minimize redundancy and also minimize the insertion, deletion and update anomalies [3].

The minimization of functional dependencies hence makes the program or database more structured, modifiable, less redundant and easy to abstract the functionality of the program.

2. TAXONOMY

Referenced Attribute: A variable is said to be referenced in a statement if the value of that variable is used during the execution of the statement without getting itself modified. For ex., A = B + C. The values of B & C are used or referenced in the statement [6].

Defined Attribute: A variable is said to be defined, in a statement if the execution of that statement can alter the value referenced or used in the statement, and A is said to be defined [6].

Functional Dependency (FD): A functional dependency, denoted by $X \rightarrow Y$, between two sets of attributes X and Y that are subsets of R (where R is a relation schema) specifies a constraint on the possible tuples that can form a relation state r of R. The constraint is that for any two tuples t_1 and t_2 in r that we have $t_1[X] = t_2[X]$, they must also have $t_1[Y] = t_2[Y]$.This means that the values of the Y component of a tuple in r depend on the values of X component [3].

Minimal Cover: Minimal cover of a set of functional dependencies E in the standard canonical form and without redundancy that is equivalent to E. We can find at least one minimal cover F for any set of dependencies E [1].

Canonical Form: Every dependency in a set of functional dependencies F has a single attribute for its right hand side [3].

Software Maintenance: Software maintenance refers to the post-delivery activities and involves modifying the code and associated documents in order to eliminate the effect of residual errors that come to surface during use [4].

3. PROPOSED METHODOLOGY

The functional dependencies present in a program or a data base is taken as an input. The input given must be in the canonical form, i.e. each functional dependency must define one attribute. For example, if there is a dependency say, A→BC, then the canonical form of this dependency will be A→B and A→C. All the functional dependencies that are identified have two parts the LHS and the RHS part. The RHS contains the attribute that is being defined and the LHS has the set of attributes that define the RHS.

The LHS is written in binary form. All the attributes are listed and the value of each attribute is either 1 if it participates in a particular functional dependencies or its value is 0.

The functional dependencies that define the same RHS are grouped. And each group is further subjected to minimization of the functional dependencies by using the basic logical operators on them. Only two FDs are considered per iteration. The outcome of minimization will either retain both the FDs or will retain only one FD. The minimized functional dependencies from each such group are combined to obtain a final set of minimized functional dependencies.

3.1 Algorithm to minimize the functional dependencies

// to find the minimized set of functional dependencies from a given set of functional dependencies (FDs).

Input: A set of functional dependencies from a program or a database.

Output: A set 'min' consisting of all the minimized functional dependencies.

```
Step 1: Transform the FDs into canonical form.

Step2: Form m group of FDs having the same attribute on the
        RHS.

Step3: k=1
        i=1
        j=1

Step4: ∀ group_k up to group_m
        do
Step5:     ∀ fdi and fdj in group_k
           do
               if $fd_j$ ← NULL
                 min ← min U {$fd_i$}
             res← fdi.LHS && fdj.LHS
             if res = $\begin{cases} 1 & \text{min} \leftarrow \text{min} \cup \{fd_i\} \\ 0 & \text{min} \leftarrow \text{min} \cup \{fd_i,\ fd_j\} \end{cases}$
               goto step5

               if fdi.LHS ⊆ fdj.LHS
               if   res = $\begin{cases} fd_i.LHS & \text{min} \leftarrow \text{min} \cup \{fd_i\} \\ fd_j.LHS & \text{min} \leftarrow \text{min} \cup \{fd_j\} \end{cases}$
               goto step5
             if fdi.LHS ⊂ fdj.LHS
               min← min U {fdi.LHS || fdj.LHS}
             goto step5
```

Since the algorithm divides the initial set of functional dependencies into many groups and then performs the basic operations on each group, and later combines the minimized functional dependencies obtained from each group into a single minimized set **min**, it can be categorized as the divide and conquer algorithm design technique. Hence, the basic asymptotic notation for the proposed algorithm can be **Θ (nlogn)** [9].

4. CASE STUDY

Let us assume that the set of attributes {A, B, C, D, G, H, K} is present either in a database.

These set of attributes define the following functionality:

ACD→K
ABC→B
AC→K
AC→D
AB →K
ABD→G
AC→K
A→H
AD→H
A→K
G→K

We represent the LHS of the FDs in binary values. If the attribute is participating in the dependency it is represented as 1, if the attribute is not present, it is represented as 0.The table shows the above functional dependencies in binary form:

Table 1. Functional dependencies with the binary representation of LHS

FD No.	**A**	**B**	**C**	**D**	**G**	**H**	**J**	**K**	**Dependency**
1	1	0	1	1	0	0	0	0	K
2	1	1	1	0	0	0	0	0	B
3	1	0	1	0	0	0	0	0	K
4	1	0	1	0	0	0	0	0	D
5	1	1	0	0	0	0	0	0	K
6	1	1	0	1	0	0	0	0	G
7	1	0	1	0	0	0	0	0	K
8	1	0	0	0	0	0	0	0	H
9	1	0	0	1	0	0	0	0	H
10	1	0	0	0	0	0	0	0	K
11	0	0	0	0	1	0	0	0	K

We first group the FDs that are having the same RHS. Hence we obtain 5 different groups for the set of RHS {K, H, D, G, B}. Let us consider the group of FDs determining the RHS ‘K’.

Table 2. Grouping the FDs that determine the attribute ‘K’

FD No.	**A**	**B**	**C**	**D**	**G**	**H**	**J**	**K**	**Dependency**
1	**1**	**0**	**1**	**1**	**0**	**0**	**0**	**0**	**K**
2	1	1	1	0	0	0	0	0	B
3	**1**	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**K**
4	1	0	1	0	0	0	0	0	D
5	**1**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**K**
6	1	1	0	1	0	0	0	0	G
7	**1**	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**K**
8	1	0	0	0	0	0	0	0	H
9	1	0	0	1	0	0	0	0	H
10	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**K**
11	**0**	**0**	**0**	**0**	**1**	**0**	**0**	**0**	**K**

Hence, the group of FDs determining the attribute ‘K’ is:

group $_1$ = {ACD→K, AC→K, AB→K, AC→K, A→K, G→K}

min= { }

For every 2 FDs in the group$_1$, step5 is carried out for five iterations. They are as follows:

Iteration 1:
For the first two FDs of the group, ACD→ K and AC→ K, i.e. 10110000→K and 10100000→K
res= 10100000
min= min U {AC→ K}

Here the LHS of the second FD is a proper subset of the LHS of the FD in ‘min’ and hence the second FD is retained into ‘min’ and the first FD is discarded.

Iteration 2:
The next FD in the group is AB→ K. Now, the two FDs to be compared are AB→ K and AC→ K, i.e. 11000000→K and 10100000→K
res= 10000000.

min= {~~AC→K~~} U {ABC→ K; (i.e., 110 || 101)}

Here LHS of second FD is a partial subset of the LHS of the FD in ‘min’ and hence only the result of the OR operation is taken and both the input FDs are discarded.

Iteration 3:
The next FD in the group is AC→K. Now, the two FDs to be compared are AC→ K and ABC→ K, i.e. 10100000→K and 11100000→K.
res= 10100000.

min= {~~ABC→K~~} U {AC→ K}.

Here the LHS of first FD is a proper subset of the second FD and hence the first FD is retained and second FD is discarded from ‘min’.

Iteration 4:
The next FD in the group is A→ K. Now, the two FDs to be compared are A→K and ABC→ K, i.e. 10000000→K and 10100000→K
res= 10000000→K.
min= {~~ABC→K~~} U {A→ K}

Here the LHS of first FD is a proper subset of the second FD and hence first FD is retained and the second FD is discarded from ‘min’.

Iteration 5:
The next FD in the group is G→ K. Now the two FDs are G→ K and A→ K.
res=00000000

Here both the FDs are different and hence both the FDs are retained.
Therefore, min= {A→K} U {G→ K}.

Iteration 1 of group$_2$:
There is only one FD ABC→B in the group$_2$, the other FDs do not exist i.e. they are NULL. Hence, there is one iteration that adds the FD ABC→B to the set 'min'.
min= {A→K, G→K, ABC→B}

Again, the step 4 is repeated and the group of FDs that determine the attribute B are obtained i.e.
Group$_2$ = {ABC→B}
min= {A→K, G→K}

Similar process is carried out for all the other groups of FDs determining the attributes D, G, H present in the RHS. The final minimized set of functional dependencies for the given input is:
min = {A→K, G→K, ABC→B, AC→D, ABD→G, A→K}.

5. CONCLUSION

This paper presents an automatic tool that takes the functional dependencies which are abstracted from database or program as input and then applies the algorithm on the abstracted functional dependencies. The functional dependencies are linked to form the attributes closure. Repeating the algorithm on functional dependencies sets, a minimized set of functional dependencies are obtained.

6. REFERENCES

[1] Dr. Shivanand M. Handigund, Rajkumar N. Kulkarni, "An Ameliorated Methodology for the Abstraction and Minimization of Functional Dependencies of Legacy 'C' program Elements", International Journal of Computer Applications, Volume 16- No 3, February 2011.

[2] Pankaj Jalote, "An Integrated Approach to Software Engineering", Third Edition, Narosa Publishing House.

[3] Ramez Elmasri, Shamkant B. Navathe, "Fundamentals of Database Systems", Fifth Edition, Pearson Education.

[4] Pratap K. J. Mohapatra, "Software Engineering (A Lifecycle Approach)", New Age International (P) Limited, New Delhi.

[5] Philip A. Hausier, Mark G. Pleszkoch, Richard C. Linger, Alan R. Hevner, "Using Function Abstraction to Understand Program Behaviour", IEEE-1990.

[6] K K Aggarwal and Yogesh Singh, "Software Engineering", Revised Second Edition, New Age International (P) Limited, New Delhi.

[7] Rajib Mall, "Fundamentals of Software Engineering", Third Edition, PHI Learning Private Limited, New Delhi 2010.

[8] Joobin Choobineh, Santosh S. Venkatraman, "A methodology and Tool for Automated Derivation of Functional Dependencies", IEEE-1989.

[9] Anyny Levitin, "Introduction to The Analysis & Design of Algorithms, Pearson Education.