International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:8, No:5, 2014

Applying Spanning Tree Graph Theory for Automatic Database Normalization

Chetneti Srisa-an

***Abstract*—**In Knowledge and Data Engineering field, relational database is the best repository to store data in a real world. It has been using around the world more than eight decades. Normalization is the most important process for the analysis and design of relational databases. It aims at creating a set of relational tables with minimum data redundancy that preserve consistency and facilitate correct insertion, deletion, and modification. Normalization is a major task in the design of relational databases. Despite its importance, very few algorithms have been developed to be used in the design of commercial automatic normalization tools. It is also rare technique to do it automatically rather manually. Moreover, for a large and complex database as of now, it make even harder to do it manually.

This paper presents a new complete automated relational database normalization method. It produces the directed graph and spanning tree, first. It then proceeds with generating the 2NF, 3NF and also BCNF normal forms. The benefit of this new algorithm is that it can cope with a large set of complex function dependencies.

***Keywords*—**Relational Database, Functional Dependency, Automatic Normalization, Primary Key, Spanning tree.

I. Introduction

NORMALIZATION as a method of producing good relational database designs is a well-understood topic in the relational database field [9]. The goal of normalization is to create a set of relational tables with minimum amount of redundant data that can be consistently and correctly modified [4]. The main goal of any normalization technique is to design a database that avoids redundant information and update anomalies [2], [7]. Normalization is often performed as a series of tests on a relation to determine whether it satisfies or violates the requirements of a given normal form. Three normal forms called first (1NF), second (2NF), and third (3NF) normal forms were initially proposed. In practice, however, databases are normalized up to and including 3NF. Therefore, higher order normalization is not addressed in this paper. The first normal form states that every attribute value must be atomic, in the sense that it should not be able to be broken into more than one singleton value. As a result, it is not allowed to have arrays, structures, and as such data structures for an attribute value. Each normal form is defined on top of the previous normal form. That is, a table is said to be in 2NF if and only if it is in 1NF and it satisfies further conditions. Except for the 1NF, the other normal forms of our interest rely on Functional Dependencies (FD) among the attributes of a relation. Functional Dependency is a fundamental notion of the Relational Model [3]. Functional dependency is a constraint between two sets of attributes in a relation of a database.

Chetneti Srisa-an is with the faculty of Information Technology, Rangsit University, Pathumthani Thailand (e-mail: chetneti@rsu.ac.th).

Given a relation R, a set of attributes X functionally determines another attribute Y, also in R, (written as X → Y) if and only if each X value is associated with at most one Y value. It is customarily to call X the determinant set and Y the dependent attribute. Given that X, Y, and Z are sets of attributes in a relation R, one can derive several properties of functional dependencies. Among the most important ones are Armstrong's axioms. These axioms are used in database normalization:

Subset Property (Axiom of Reflexivity)
: If Y is a subset of X, then X → Y
Augmentation (Axiom of Augmentation)
: If X → Y, then XZ → YZ
Transitivity (Axiom of Transitivity)
: If X → Y and Y → Z, then X →Z

By repeated application of Armstrong's rules all functional dependencies can be generated. These functional dependencies provide the bases for database normalization.

In Section II, Step1: we use dependency graph diagrams such as Fig. 1 to represent functional dependencies of a database and then Step2: we have generated the spanning tree graph using depth first search method. A few algorithms have been developed to be used in the design of commercial automatic normalization tools [5], [6]. In Section III, by applying Spanning Tree, a new algorithm is introduced to produce normal forms of the database. Section IV is conclusion.

II. Representing Dependencies

We will use two structures, Function Dependency Graph (DG), and Spanning tree (STG) Graph, to represent and manipulate dependencies among attributes of a relation.

A. Function Dependency Graph

With functional dependency such as Fig. 1, we can monitor all relations between different attributes of tables. We can graphically show these dependencies by using a set of simple symbols. In these graphs, arrow is the most important symbol used. Besides, in our way of representing the relationship graph, a (dotted) horizontal line separates simple keys (i.e. attributes) from composite keys (i.e., keys composed of more than one attribute).

A dependency graph is generated using the following rules.

1. Each attribute of the table is encircled.
2. Each composite key (if any) is encircled and all composite keys are drowning on top of the graph.

3. All functional dependency arrows are drawn.
4. All reflexivity rule dependencies are drawn using dotted arrows (for example AB→A, AB→B).

Consider the functional dependency set of Example 1 [2] for a relation r.

Example 1. FDs = {A → BCD, C → D, EF → DG,D → G}

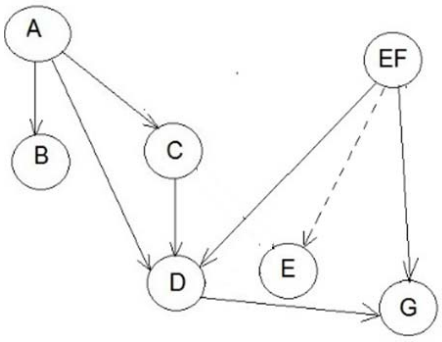

Fig. 1 Dependency graph diagrams of Example 1

If we are able to obtain all dependencies between determinant keys we can produce all dependencies between all attributes of a relation.

Depth-first search (DFS) is an algorithm for traversing or searching tree or graph data structures. One starts at the root (selecting some arbitrary node as the root in the case of a graph) and explores as far as possible along each branch before backtracking. Since a spanning tree of the graph is a connected subgraph in which there are no cycles, all nodes will be visited by using depth first search algorithm. Using Depth-first search (DFS) Algorithm and Armstrong's transitivity rule, a new Spanning Tree Graph (STG) is constructed. From both graphs will be used for a new normalization algorithm which we will be discussing in the following.

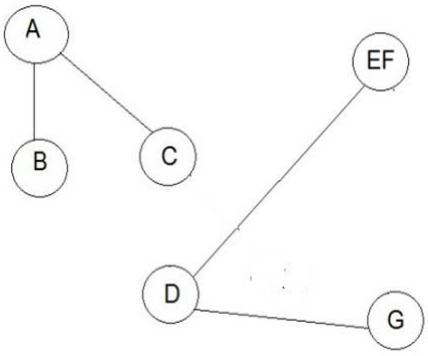

Fig. 2 Spanning Tree Graph (STG)

III. The Proposed Normalization Process

It is assumed that the reader is familiar with the definitions of different normal forms. Our proposed 2NF and 3NF normalization process makes use of both dependency and determinant key transitive dependencies.

A. Second Normal Form (2NF)

The tables of a relational database are assumed to be in 1NF form to begin with. The resulting 1NF relation is: R2(<u>AB</u>, C, D, E, F) FDs = {AB →CDEF, A→EF}

The goal is to discover all partial dependencies. To produce the 2NF form, we should find all partial dependencies. To do this, we construct dependency graph and function dependency

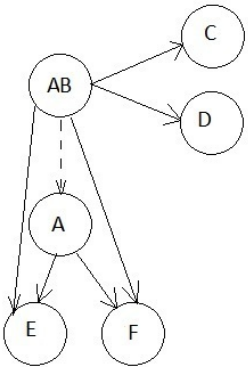

Fig. 3 Function Dependency Diagram Graph

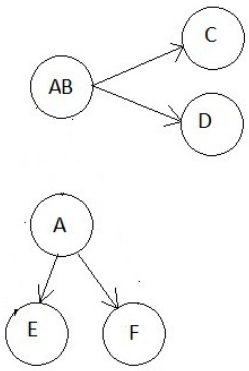

Fig. 4 Spanning Tree

From Fig. 4, we can simply construct a new normalize as following relation.

R21(A,E,F)

F21={A→E, F} R22(AB,C,D) F22={AB→C,D}

B. Third Normal Form (3NF)

Every non-prime attribute is non-transitively dependent on every candidate key in the table. The attributes that do not contribute to the description of the primary key are removed from the table [1].

R3(<u>A</u>, B,C, D, E, F) FDs = {A → B,C,D,E,F, B→C,D}

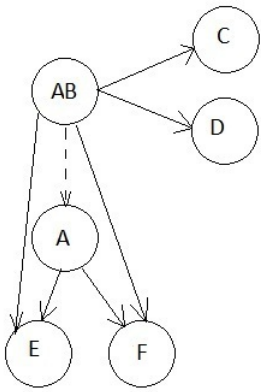

Fig. 5 Function Dependency Diagram Graph

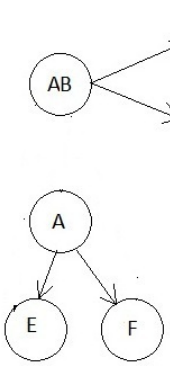

Fig. 6 Spanning Tree

C. The BCNF Normal Form

A relation is in BCNF, if and only if, every determinant is a candidate key. For a relation with only one candidate key like example 1, 3NF and BCNF are equivalent [8]. To develop the process of generating BCNF form, consider the case where there is more than one candidate key for the table being normalized.

Example 2. Relation R3(AB, C, D) with dependencies:
F3= {AB→CD, C→B}

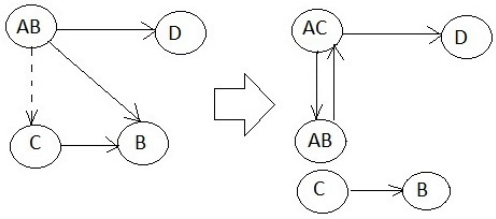

Fig. 7 Equivalent Candidate Key Mapping

To transform the relations to BCNF requires the creation of new relation for each transitivity dependency. The resulting BCNF relations are: R31(AC, B,D) and R32(C,B).

Example 3. R4(A,B,C,D,E,F,G,H,I,J,K) and
F4 = {A→BCEH, B →CDE, CD→EFG, DH→IJ
I→D, J→K, K→H, IJ→DKHL, HI→M}

In example 3, we will show how to cope with a large and complex set of function dependencies that contain a multiple candidate keys [2].

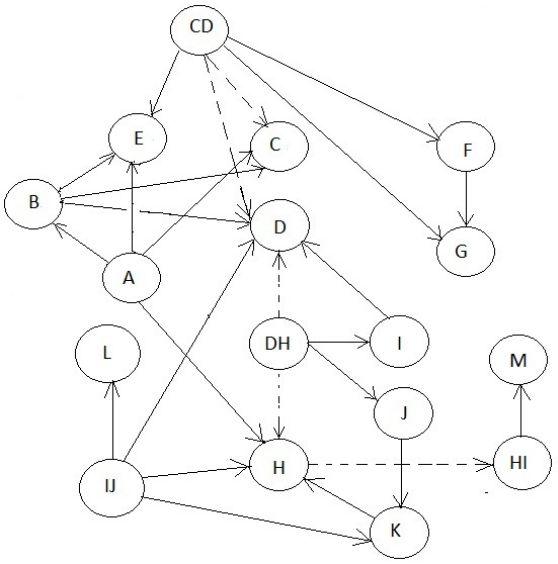

Fig. 8 Dependency graph diagram of Example 3

It is noticeable that the candidate key will be DH,IL. Firstly, the spanning tree is generated in Fig. 9. Secondly, we will identify the equivalent key mapping in Fig. 10.

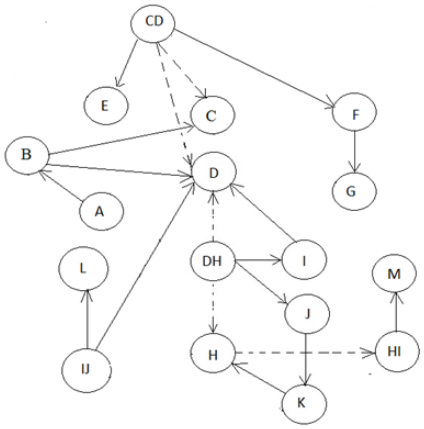

Fig. 9 Spanning Tree Graph (STG) of Example 3

From Fig. 9, the candidate key is DH and IJ; therefore, we can group into DHIJ→KL as show below. We redraw into Fig.10.

To transform the relations to BCNF requires the creation of new relation for each transitivity dependency. The resulting BCNF relations are:

DH→I,J
I →D
J →K DHIJ→KL
K-->H

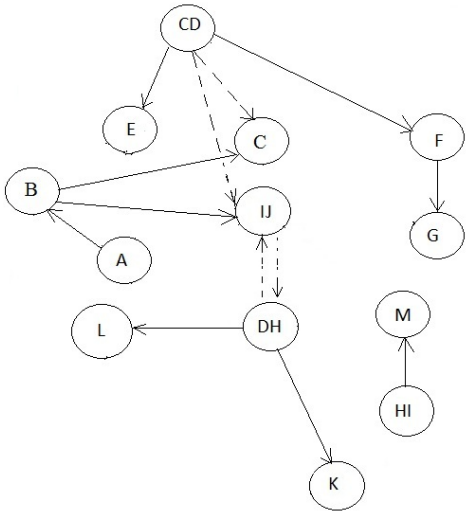

Fig. 10 Equivalent Candidate Key Mapping

In Fig. 10, Sub Trees represent all normalized tables.

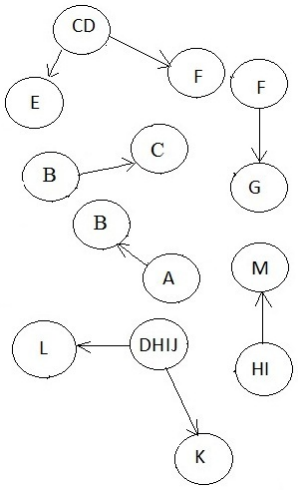

Fig. 11 Sub Tree of Example2

R41 (A,B,H) F41={A→ B,H}
R42 (B,C,D) F42={B →C, D}
R43 (CD,E,F) F43={CD→E,F}
R44 (DHIJ,L,K) F44={DHIL → J,K}
R45 (HI,M) F45={HI→M}

IV. Algorithm

There are three steps in our algorithm. Our algorithm starts with comparing two graphs that are generated in example 1. There are four links are eliminated by our algorithm as shown in Figs. 1 and 2. Step2: the determinants in each original function dependency will be a primary key. This example will be A,C, EF, and D. Step3: locating a candidate key is to form BCNF. Finally, all sub trees will be isolated and redrawn as shown in Fig. 12.

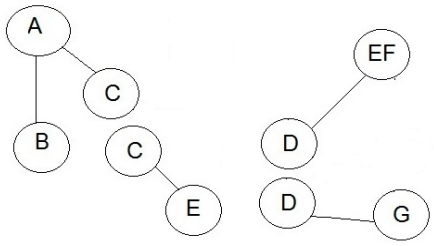

Fig. 12 Sub Trees that represent all normalized tables

R1(A,B,C,D,E) F1= {A → BC}
R2(C,E) F2={C → E}
R3(E,F,D) F3={EF → D}
R4(D,G) F4={D→G}

In summary, 3NF/BCNF Decomposition algorithms are as following steps.

Step1. Construct dependency graph and spanning tree
Step2. For each determinant in the left side, gather all the non key attributes in the same group of FD.
Step3. Locating a candidate key is to form BCNF.
Step4. Construct all sub trees to form all normalized tables.

V. Conclusion

A new complete automated relational database normalization method is presented. The process is based on the generation of dependency graph and spanning tree, and algorithm. In an example 1 and 2, one without multiple candidate keys and one with multiple candidate keys are considered and the defined algorithms are applied to produce the desired final tables. The developed algorithm is a new technique that will formulate 3NF/BCNF normal form and distinguish a primary key for every final table that is generated. In example 2, the benefit of this algorithm is that it can cope with a large set of complex function dependencies.

References

[1] Connoly, Thomas, Carolyn Begg: Database Systems. A Practical Approach to Design, Implementation, and Management , Pearson Education, Third edition, 2005.Relational and XML Data, Journal of Computer System Science, Vol. 73(4): pp. 636-647, 2007..
[2] Date, C.J., An Introduction to Database Systems, Addison-Wesley, Seventh Edition 2000.
[3] Mora, A., M. Enciso, P. Cordero, IP de Guzman, An Efficient Preprocessing Transformation for Functional Dependencies Sets Based on the Substitution Paradigm, CAEPIA2003, pp.136-146, 2003.
[4] Du H., and L. Wery, A Normalization Tool for Relational Database Designers, Journal of Network and Computer Applications, Volume 22, No. 4, pp. 215-232, October 1999.
[5] Yazici, A., and Z. Karakaya, Normalizing Relational Database Schemas Using Mathematica, LNCS, Springer-Verlag, Vol.3992, pp. 375-382, 2006.
[6] Kung, H. and T. Case, Traditional and Alternative Database Normalization Techniques: Their Impacts on IS/IT Students' Perceptions and Performance, International Journal of Information Technology Education, Vol.1, No.1 pp. 53-76, 2004.
[7] Kolahi, S., Dependency-Preserving Normalization of Relational and XML Data, Journal of Computer System Science, Vol. 73(4): pp. 636-647, 2007.

[8] M Arenas, L Libkin, An Information-Theoretic Approach to Normal Forms for Relational and XML Data, Journal of the ACM (JACM), Vol. 52(2), pp. 246-283, 2005.