JMathNorm: A Database Normalization Tool Using Mathematica

Ali Yazici[1] and Ziya Karakaya[2]

[1] Computer Engineering Department, TOBB University of Economics & Technology, Ankara - Turkey
aliyazici@etu.edu.tr
[2] Computer Engineering Department, Atilim University, Ankara - Turkey
ziya@atilim.edu.tr

Abstract. This paper is about designing a complete interactive tool, named JMathNorm, for relational database (RDB) normalization using Mathematica. It is an extension of the prototype developed by the same authors [1] with the inclusion of Second Normal Form (2NF), and Boyce-Codd Normal Form (BCNF) in addition to the existing Third normal Form (3NF) module. The tool developed in this study is complete and can be used for real-time database design as well as an aid in teaching fundamental concepts of DB normalization to students with limited mathematical background. JMathNorm also supports interactive use of modules for experimenting the fundamental set operations such as closure, and full closure together with modules to obtain the minimal cover of the functional dependency set and testing an attribute for a candidate key. JMathNorm's GUI interface is written in Java and utilizes Mathematica's JLink facility to drive the Mathematica kernel.

1 Introduction

Design of a RDB system consists of four main phases, namely, (i) determination of user requirements, (ii) conceptual design, (ii) logical design, and finally, (iv) physical design [2]. During the conceptual design phase, set of business rules is transformed into a set of entities with a set attributes and relationships among them. Extended Entity Relationship (EER) modeling tool can be utilized for the graphical representation of this transformation. The entity set in the EER model is then mapped into a set of relation schemas $\{R_1, R_2, R_3, ..., R_n\}$ where each R_i represents one of the relations of the DB schema. A temporary primary key is designated, and a set of functional dependencies (FD's) among the attributes of each schema are established as an outcome of this phase.

As a side product of the logical design phase, each R_i is transformed into well-formed groupings such that one fact in one group is connected to other facts in other groups through relationships [3]. The ultimate aim of this article is to perform this rather mechanical transformation process, called normalization, efficiently in an automatic fashion.

Commercial DB design tools do not provide a complete solution for automatic normalization and existing normalization tools for the purpose require high level programming skills and complex data structures. Two such implementations in Prolog language are discussed in [4,5]. Another study on automatic transformation is given in [6] in which UML is used to access Object Constraint Language (OCL) to construct expressions that encode FD's using classes at a meta-level. An alternative approach to normalization is given in [3] that focus on addressing FD's to normalize DB schema in place of relying on the formal definitions of normal forms. The impact of this method on IS/IT students perceptions is also measured in the same study. It appears that this approach is only useful for small sets of FD's in a classroom environment and in particular not suited for automatic normalization. A web-based tool for automatic normalization is given in [7] which can normalize a DB schema up to 3NF for a maximum of 10 FD's only.

This article is an extension of the work [1] and discusses a complete normalization tool called JMathNorm which implements 2NF, 3NF, and BCNF using the abstract algorithms found in the literature [2,9,13]. JMathNorm's normalization modules are written in Mathematica [8] using the basic list/set operations, the user interface is designed using Java language, and finally, execution of Mathematica modules is accomplished by employing Mathematica's Java Link (JLink) utility. The design approach in this study is similar to the Micro tool given in [9]. However, JMathNorm provides additional aspects for educational purposes and is implemented efficiently without using any complex data structures such as pointers.

The remainder of this article is organized as follows. Section 2 briefly reviews the DB normalization and some of the basic functions used in normalization algorithms. In Section 3 Mathematica implementation of BCNF algorithm is given. JMathNorm tool is demonstrated in Section 4. Remarks about the tool and discussion for future work are provided in the final section.

2 A Discussion on Normalization Algorithms

A functional dependency (FD) is a constraint about sets of attributes of a relation R_i in the DB schema. A FD between two sets of attributes X and Y, denoted by, $X \rightarrow Y$ specifies that there exists at most one value of Y for every value of X (determinant)[2,10,11]. In this case, one asserts that X determines Y or Y is functionally dependent on X.

For example, for a DB schema *PURCHASE-ITEM* = {***orderNo, partNo,*** *partDescription, quantity, price*}, with *PK* = {*orderNo, partNo*}, using a set of business rule one can specifiy the following FD's:

$FD1$: {*orderNo, partNo*} $\rightarrow$ {*partDescription, quantity, price*}
$FD2$: *partNo* $\rightarrow$ *partDescription*

For a given schema, other FD's among the attributes can be inferred from the Armstrong's inference rules [2]. Alternatively, for an attribute set X, one can deduce the others known as X closure, X^+, determined by X, based on the FD set F of the schema. Set closure is one of the fundamental functions for the normalization algorithms and will be referred to as ClosureX [1] in the sequel. FullClosureX, X^{++}, is yet another function similar to ClosureX which returns all attributes that are fully dependent on X with respect to FD set. This function is used to remove partial dependencies for transforming a relation into 2NF. An algorithm for full closure function is given below[9]:

Algorithm FullClosureX (*X: attribute set; F: FD set*)**:** *return closure in tempX*;

```
1. $tempX := X$;
2. repeat
       $oldX := tempX$;
       for each FD $Y \rightarrow Z$ in $F$ do
          if $Y \subset tempX$ then if $not(Y \subset X)$ then $tempX := Z \cup tempX$
          else if $Y = X$ then $tempX := Z \cup tempX$;
   until $(length(oldX) = length(tempX))$;
3. return tempX;
```

Given a set of FD's F, an attribute B is said to be extraneous [11] in $X \rightarrow A$ with respect to F if $X = ZB$, $X \neq Z$, and $A \in Z^+$. A set of FD's H is called a minimal cover[1,5] for a set F if each dependency in H as exactly one attribute on the right-hand side, if no attribute on the left-hand side is extraneous, and if no dependency in H can be derived from the other dependencies in H. Actually, the calculation of a minimal cover consists of "Elimination of Extraneous Attributes" followed by the "Elimination of Redundant Dependencies". Normalization algorithms considered in this study makes use of the minimal cover of a given set of FD's. Moreover, they are computationally efficient with at most $O(n^2)$ operations where n is the number of FD's in the schema.

Normalization is a step by step process to transform the DB schema into a set of subschemas. For the normal forms used in this study (2NF, 3NF and BCNF) this is achieved by decomposing each R_i into a set of relations by removing certain kind of redundancies in the relation. Lack of normalization in a DB schema causes update anomalies [13] which may destroy the integrity of the DB.

If a relation has no repeating groups, it is said to be in the first normal form (1NF). In this study, it is assumed that all relations do satisfy this condition. A relation is in the second normal form (2NF) if no part of a PK determines nonkey attributes of the relation. Note that, for the example above, because of $FD2$, the relation is not in 2NF. 3NF relations prohibit transitive dependencies among its attributes. And, finally, in Boyce-Codd Normal Form (BCNF) a nonkey attribute cannot determine a prime attribute (any part of PK).

A 2NF algorithm with the attribute preservation property is given in [9]. JMathNorm uses a slightly modified version of this algorithm to remove partial dependencies and hence transform the DB schema into 2NF. Bernstein's Synthesis algorithm [1,12] is implemented to provide 3NF relations directly for a given set of attributes and a set of FD's F. Original dependencies are preserved, however, lossless join property [1] is not guaranteed by this algorithm.

In certain 3NF DB schemas, a FD from a nonprime attribute into a prime one may exist. Boyce-Codd Normal Form (BCNF) of a 3NF relation is achieved by removing such dependencies. A sketch of the BCNF algorithm with the lossless join properties [2,12] is given below.

Algorithm BCNF (*R: attribute set in 3NF; F: FD set*)**:** *return Q in BCNF*;

```
1. $D := R$;
2. while there is a left-hand side X of a FD $X \rightarrow Y$ in F do
       if $X \rightarrow Y$ violatesBCNF then
           decompose R into two schemas $R_m := D - Y$; and $R_n := X \cup Y$;
3. return $Q := R_m \cup R_n$;
```

The function *violateBCNF* tests if a given FD violates the BCNF condition by calculating the X closure. If it includes all the attributes from R then R does not violate the BCNF constraint, otherwise R violates the constraint and needs to be decomposed into R_m and R_n as given above.

3 Mathematica Implementation

3.1 BCNF with Mathematica

In Fig.1, a use case diagram is given to demonstrate the functions and modules used in the tool.

Tasks in Fig.1 are effectively implemented as Mathematica modules by utilizing only the Mathematica's list structure and the well-known set operations [8]. These operations are *Union*[], *Complement*[], *Intersection*[], *MemberQ*[], *Extract*[], *Append*[], *Length*[], and *Sort*[].

A FD set F of a schema is represented by two lists, one for the left hand sides (FL), and the other for the right hand sides of F (FR). Obviously, the order of attributes in such a list is important and should be maintained by care throughout the normalization process.

For the example above, the FD set is represented in Mathematica as follows:

$FL =$ {{*orderNo, partNo*}, {*orderNo, partNo*},{*orderNo, partNo*},*partNo*}
$FR =$ {*partDescription, quantity, price, partDescription*}

Accordingly, $FL[[i]] \rightarrow FR[[i]]$, for $i = 1, 2, 3, 4$ as specified by the FD set F.

As an illustration, the Mathematica code for the BCNF algorithm is given below. Given a FD set and a 3NF relation R, BCNF algorithm first looks for a BCNF violation using the function *violatesBCNF*. When found, it returns in Q two sub relations satisfying the BCNF constraint.

```
BCNF[FL_, FR_, R_] := Module[{i, X, D, Q, DIF, REL}, D = R; Q = {};
   For[i = 1, i <= Length[FL], i++,
      If[Length[FL[[i]]] > 1, X = Sort[FL[[i]]], X = {FL[[i]]}];
         flag = violatesBCNF[FL, FR, X, FR[[i]], U];
      If[flag == 1, REL = Union[X, {FR[[i]]}];
      Q = Union[Q, {REL}]; RC = Complement[R, {FR[[i]]}];
      DIF = Intersection[R, RC]; Q = Union[Q, {DIF}];];];Return[Q];];
violatesBCNF[FL_,FR_,X_,Y_,R_]:=Module[{XP, flag},
   XP=Sort[ClosureX[FL,FR,X]];
   If[XP==Sort[U], flag=0,flag=1];Return[flag];];
```

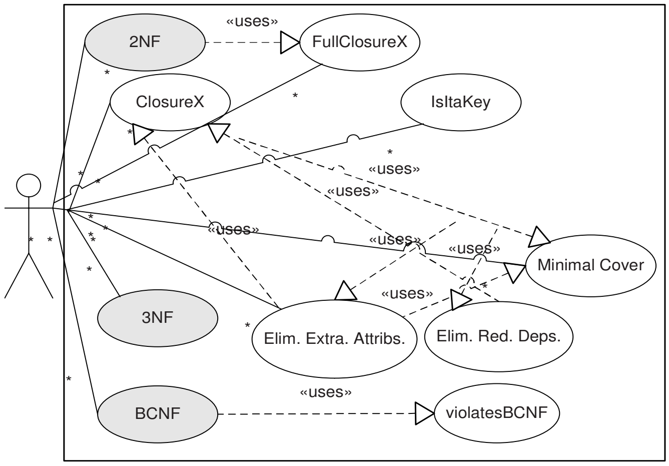


Fig. 1. Use Case Diagram for Normalization Modules

An example of a relation with BCNF violation and it's decomposition by the code above is given below. Consider a relation CLIENT-INTERVIEW={**clientno, interviewdate**, interviewtime, staffno, roomno} with the following FD's:

$FD1$: {*clientNo, interviewdate*} $\rightarrow$ {*interviewtime,staffno,roomno*}
$FD2$: {*staffno,interviewdate,interviewtime*} $\rightarrow$ *clientno*
$FD3$: {*staffno,interviewdate*} $\rightarrow$ *roomNo*

In this relation {clientno, interviewdate}, and {staffno, interviewdate} are both candidate keys and share a common attribute. And, BCNF constraint is violated by FD2. The result of running the BCNF module by providing the required parameters produces the following decomposition Q.

$$Q = \{\{interviewdate, roomno, staffno\},\\ \{clientno, interviewdate, interviewtime, staffno\}\}$$

4 JMathNorm User Interface

An interactive tool, JMathNorm, with a GUI written in Java is designed to implement the system given in Fig.1. Each algorithm is implemented as a Mathematica module. JLink (Java Link) facility of Mathematica is utilized to load the

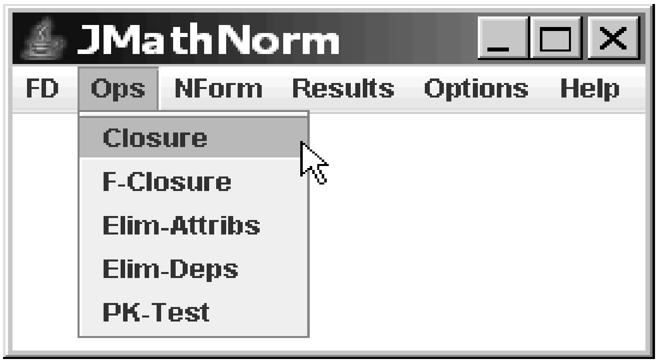
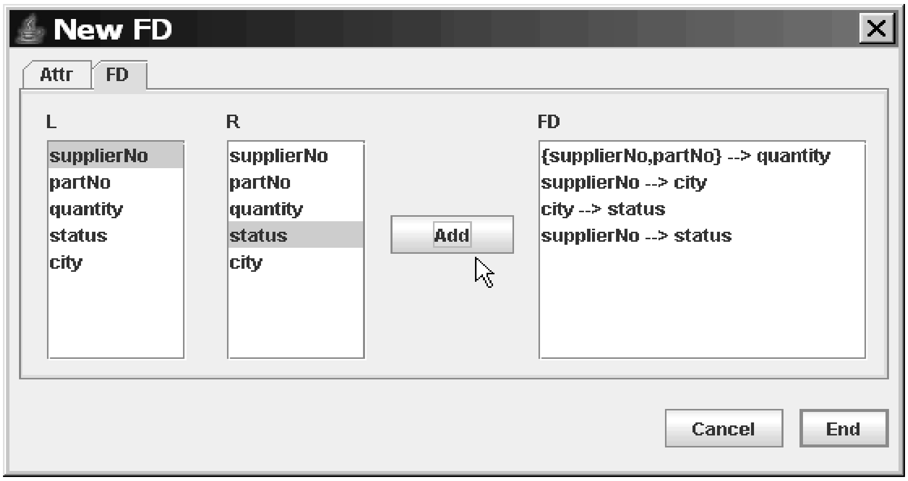


Fig. 2. JMathNorm's menu options

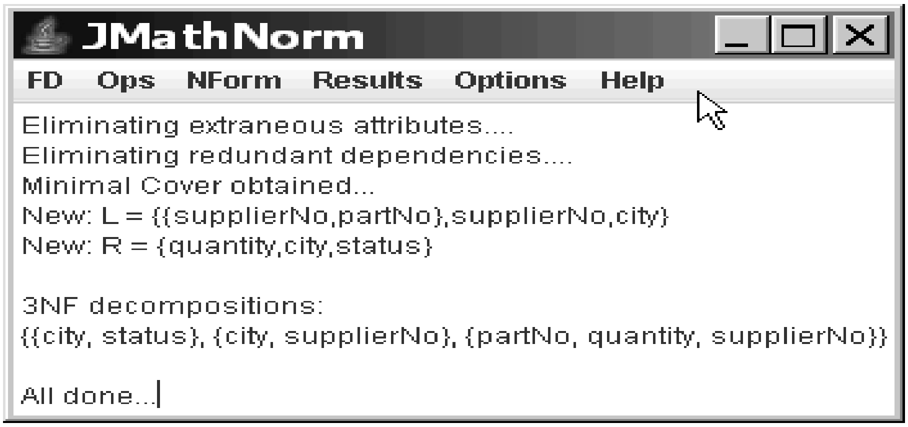


L	R	FD
supplierNo	supplierNo	{supplierNo,partNo} --> quantity
partNo	partNo	supplierNo --> city
quantity	quantity	city --> status
status	status	supplierNo --> status
city	city	

Fig. 3. Dialog box to define FD's

Mathematica kernel and execute these modules as required. JMathNorm starts with a dialogue box asking for the relevant path of Mathematica kernel. Mathematica functions for the normalization are loaded afterwards. Consequently, only the calling statement of those modules is passed to Mathematica to receive a result string. The result returned is just the set representation in a string. This string is to be parsed into the desired data structure. In JMathNorm, results are stored into java's Vector data structure.

The interface offers a menu driven interaction with the system. The main and *Operations* submenus are displayed in Fig.2. JMathNorm's *FD* pull-down menu can be used to set up a new set of FD's, open an existing one, save or edit FD's using a data entry dialog box. One can experiment with basic normalization tasks, namely, *set closure*, *set full closure*, *elimination of redundant attributes*,



```
Eliminating extraneous attributes....
Eliminating redundant dependencies....
Minimal Cover obtained...
New: L = {{supplierNo,partNo},supplierNo,city}
New: R = {quantity,city,status}

3NF decompositions:
{{city, status}, {city, supplierNo}, {partNo, quantity, supplierNo}}

All done...
```

Fig. 4. A sample run for 3NF decomposition

elimination of redundant dependencies, *testing for primary key* and *obtaining minimal cover* by utilizing the *Basic Operations* submenu. These set theoretic operations form the basis of all of the normalization algorithms discussed in the preceding sections. Moreover, because of their symbolic nature, verification of the result returned from each manually is rather cumbersome. JMathNorm overcomes this problem, by providing a verification mechanism as a background for teaching normalization theory effectively in a classroom environment. Database schemas can be transformed into the required normal form directly from the *NForm* submenu. As a result of normalization, the original relations of the DB schema are decomposed into sub relations which is displayed systematically by the *Results* submenu. In Fig.3, the dialog box for defining FD's are shown. A sample run to decompose the relation into 3NF for is displayed in Fig.4.

5 Tests and Discussions

Several benchmark tests found in the literature are successfully applied with varying number of FD's having different initial normal forms. Normalization algorithms used in the tool possess at most quadratic time complexity providing in the number of FD's and are computationally effective.

JMathNorm was also used in a classroom environment during a Database Systems course offered to about 25 third year computer engineering majors during the Spring semester of 2006-07 academic year. Students are requested to form project teams and design a medium size database system involving 8-10 relations. During the design process, they ended up with normalizing the relational schema. Students usually preferred using JMathNorm to support or validate the normalization process. It was reported that each team used JMathNorm on average of four times. In addition to the use in the project, students utilized the tool to understand the normalization process and the underlying theory based on the set theoretic operations discussed earlier.

In the course evaluation forms, majority of the students have indicated that the tool was quite useful to check their manual work in studying the normalization algorithms and to normalize schemas for the database design project of the course.

Modules of JMathNorm was written in Mathematica utilizing only basic list/set operations as the fundamental data structure. These operations empowered by the symbolic nature of Mathematica resulted in an effective normalization tool. Currently, it does not have the ability to create SQL statements for the normalized schema. A table creation facility geared towards a specific DBMS is to be included to JMathNorm.

References

1. Yazici, A. and Karakaya, Z.: Normalizing Relational Database Schemas Using Mathematica, LNCS, Springer-Verlag, Vol.3992 (2006) 375-382.
2. Elmasri, R. and Navathe, S.B.: Fundamentals of Database Systems, 5th Ed., Addison Wesley (2007).
3. Kung, H. and Case, T.: 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 (2004) 53-76.
4. Ceri, S. and Gottlob, G.: Normalization of Relations and Prolog, Communications of the ACM, Vol.29, No.6 (1986)
5. Welzer, W., Rozman, I. and Gyrks, J.G.: Automated Normalization Tool, Microprocessing and Microprogramming, Vol.25 (1989) 375-380.
6. Akehurst, D.H., Bordbar, B., Rodgers, P.J., and Dalgliesh, N.T.G.: Automatic Normalization via Metamodelling, Proc. of the ASE 2002 Workshop on Declarative Meta Programming to Support Software Development (2002)
7. Kung, H-J. and Tung, H-L.: A Web-based Tool to Enhance Teaching/Learning Database Normalization, Proc. of the 2006 Southern Association for Information Systems Conference (2006) 251-258.
8. Wolfram, S.: The Mathematica Book, 4th Ed., Cambridge University Press (1999).
9. Du, H. and Wery, L.: Micro: A Normalization Tool for Relational Database Designers, Journal of Network and Computer Applications, Vol.22 (1999) 215-232.
10. Manning, M.V.: Database Design, Application Development and Administration, 2nd. Ed., McGraw-Hill (2004).
11. Diederich, J. and Milton, J.: New Methods and Fast Algorithms for Database Normalization, ACM Trans. on Database Systems, Vol.13, No.3(1988) 339-365.
12. Ozharahan, E.: Database Management: Concepts, Design and Practice, Prentice Hall (1990).
13. Bernstein, P.A.: Synthesizing Third Norm Relations from Functional Dependencies, ACM Trans. on Database Systems, Vol.1, No.4 (1976) 277-298.