AUTOMATIC DATABASE NORMALIZATION AND PRIMARY KEY GENERATION

Amir Hassan Bahmani [1], *Mahmoud Naghibzadeh* [2], *Behnam Bahmani* [3]

[1] Young Researchers Club, Dept of Computer Engineering, Islamic Azad University of Mashhad, Mashhad, Iran
amirbahmani.h@gmail.com

[2] Dept of Computer Engineering, Ferdowsi University of Mashhad, Mashhad, Iran
naghib@ferdowsi.um.ac.ir

[3] Dept of Computer Engineering, Islamic Azad University of Firoozkooh
b.bahmani@iaufb.ac.ir

ABSTRACT

Normalization is the most exercised technique for the analysis 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. A normalized database does not show anomalies due to future updates. It is very much time consuming to employ an automated technique to do this data analysis, as opposed to doing it manually. At the same time, the process is tested to be reliable and correct. This paper presents a new complete automated relational database normalization method. It produces the dependency matrix and the directed graph matrix, first. It then proceeds with generating the 2NF, 3NF, and BCNF normal forms. All tables are also generated as the procedure proceeds. One more side product of this research is to automatically distinguish one primary key for every final table which is generated.

Keywords: Relational Database, Functional Dependency, Automatic Normalization, Primary Key

1. INTRODUCTION

Normalization as a method of producing good relational database designs is a well-understood topic in the relational database field [1]. 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. The main goal of any normalization technique is to design a database that avoids redundant information and update anomalies [2]. The process of normalization was first formalized by E.F.Codd. 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. An amendment was later added to the third normal form by R. Boyce and E.F. Codd called Boyce–Codd Normal Form (BCNF). The trend of defining other normal forms continued up to eighth normal form. In practice, however, databases are normalized up to and including BCNF. 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. Given a relation R, a set of attributes X, in R, is said to functionally determine another attribute Y, also in R, (written as $X \rightarrow Y$) if and only if each X value is associated with at most one Y value. That is, given a tuple and the values of the attributes in X, one can unequally determine the corresponding value of the Y attribute. 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 $X \rightarrow Y$

Augmentation (Axiom of Augmentation): If $X \rightarrow Y$, then $XZ \rightarrow YZ$

Transitivity (Axiom of Transitivity): If $X \rightarrow Y$ and $Y \rightarrow Z$, then $X \rightarrow Z$

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

Normalization is a major task in the design of relational databases [4]. Mechanization of the normalization process saves tremendous amount of time and money. Despite its importance, very few algorithms have been developed to be used in the design of commercial automatic normalization tools. Mathematical normalization algorithm is implemented in [5]. In [6] a comparison of related students' perceptions of different database normalization approaches and the effects on their performance is studied. In [7] a set of stereotypes and tagged values are used to extend the UML meta-mode. A graph rewrite rule is then obtained to transfer the data model from one normal form to a higher normal form.

In Section 2, we use dependency graph diagrams to represent functional dependencies of a database and we have generated the dependency matrix and the directed graph dependency matrix. In Section 3 a new algorithm is introduced to produce normal forms of the database. Section 4 is a short conclusion.

CCECE/CCGEI May 5-7 2008 Niagara Falls. Canada
978-1-4244-1643-1/08/$25.00 © 2008 IEEE

2. REPRESENTING DEPENDENCIES

We will use three structures, Dependency Graph (DG), Dependency Matrix (DM), and Directed Graph Matrix (DG), to represent and manipulate dependencies amongst attributes of a relation.

2.1. Dependency Graph Diagram

With functional dependency we can monitor all relations between different attributes of a table. 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 and all attributes of the table is drawn at the lowest level (i.e., bottom) of the graph.
2. A horizontal line is drawn on top of all attributes.
3. Each composite key (if any) is encircled and all composite keys are drowning on top of the horizontal line.
4. All functional dependency arrows are drawn.
5. All reflexivity rule dependencies are drawn using dotted arrows (for example AB-->A, AB-->B).

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

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

Figure 1 is the graphical representation of the dependencies.

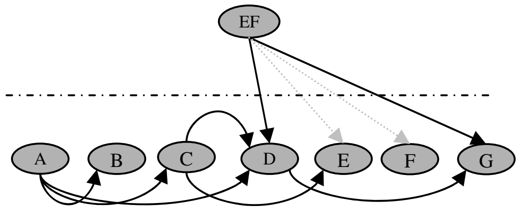

Figure 1: Graphical representation of dependencies

If we are able to obtain all dependencies between determinant keys we can produce all dependencies between all attributes of a relation. These dependencies are represented by using a Dependency Matrix (DM). Using path finding algorithms and Armstrong's transitivity rule new dependencies are discovered from the existing dependency set. This is the basis of the normalization algorithm which we will be discussing in the following.

2.2. Dependency Matrix

From a dependency graph, the corresponding Dependency Matrix (DM) is generated as follows:

i. Define matrix DM [n] [m], where
 n = number of *Determinant Keys.*
 m = number of *Simple Keys.*

ii. Suppose that $\beta \subseteq \alpha$, $\gamma \not\subset \alpha$ and $\beta,\gamma \in$ {Simple key set}
 $\alpha \in$ {Determinant key set}

iii. Establish DM elements as follows:

$$if\ \alpha \to \beta \Rightarrow \quad DM[\alpha][\beta] = 2\ ,$$

$$if\ \alpha \to \gamma \Rightarrow \quad DM[\alpha][\gamma] = 1\ ,$$

Otherwise $\Rightarrow DM[\alpha][\gamma] = 0$,

The DM for Example 1 is shown in Figure 2.

	A	B	C	D	E	F	G
A	2	1	1	1	0	0	0
C	0	0	2	1	1	0	0
D	0	0	0	2	0	0	1
EF	0	0	0	1	2	2	1

Figure 2: Initial dependency matrix

2.3. Directed Graph Matrix

The Directed Graph (DG) matrix for determinant keys is used to represent all possible direct dependencies between determinant keys. The DG is an n×n matrix where n is the number of determinant keys. The process of determining the elements of this matrix follows.

The elements of the DG matrix are initially set to zeros. Starting from the first row of the dependency matrix DM, this matrix is investigated in a row major approach. Suppose we are investigating the row corresponding to determinant key x. If all simple keys that x is composed of depend on a determinant key other than x then x also depends on that determinant key (Armstrong's augmentation rule). The dependency of a simple key to a determinant key is represented by a non-zero in the DM matrix.

For example, suppose that FDs = {AB→E, BC→A, DE→A}.The corresponding dependency matrix and the initial directed graph matrix are shown in Figure 3.

	A	B	C	D	E
AB	2	2	0	0	1
BC	1	2	2	0	0
DE	1	0	0	2	2

(a): Dependency Matrix

	AB	BC	DE
AB	0	0	0
BC	0	0	0
DE	0	0	0

(b): Directed Graph Matrix

Figure 3: Initializing DM and DG Matrices

In part (a) of Figure 4, we start with the first row of the DM matrix. The determinant key of this row is AB. A and B are subsets of AB which appear in columns one and two of the matrix. In Row one, columns one and two are both nonzero. Therefore AB depends on AB. Considering the second row, columns one and two are both nonzero, too. Hence, AB depends on BC. However, for the third row, it is not the case that both A and B depend on DE. Therefore, a -1 value is put in the intersection of row DE and column AB in the DG matrix of part (b) of Figure 4.

	A	B	C	D	E
AB	2	2	0	0	1
BC	1	2	2	0	0
DE	1	0	0	2	2

(a)

	AB	BC	DE
AB	1	-1	-1
BC	1	1	-1
DE	-1	-1	1

(b)

Figure 4: Dependency Matrix and Directed Graph Matrix

The algorithm for producing the DG graph follows.

```
Directed- Graph-Matrix()
{
  for (i=0; i<n;  i++)
  for (k= each attribute that composed determinant key i)
      for ( j=0; j<n ; j++) {
          if ( DM[j][k]!=0 && DG[j][i]!=-1)
               DG[j][i]=1;
          else DG[j][i]=-1; }
 }
```

After generating the DG matrix we turn our attention towards finding all possible paths between all pairs. This matrix will show all transitive dependencies between determinant keys. There is many such path finding algorithms like Prim, Kruskal, and Warshal algorithms. If there is a path from node x to node y it means y transitively depends on x. As an example, Figure 5 shows the complete determinant key transitive dependencies corresponding to the DM graph of Figure 3.

	AB	BC	DE
AB	1	-1	-1
BC	1	1	-1
DE	-1	-1	1

Figure 5: Determinant key transitive dependencies

From Figure 5 we can deduct that AB depends on BC. On the other hand, E depends on AB. Therefore, E depends on BC. That is, BC→AB, AB→ E => BC→E. These dependencies are recognized through dependency closure procedure which is presented in Figure 6.

```
Dependency-closure ()
 {
  for (i=0; i<n ; i++)
      for( j=0; j<n ; j++)
          if(  i!=j  && Path[i][j]!=-1)  {
               for (k=0; k<m ; k++)
                    if( DM[j][k]!=0  && DM[j][k]!=2)
                       DM[i][k]=j;  }
 }
```

Figure 6: Recognition of dependency closure

DM of Figure 3 is updated as follows to reflect all dependencies including those that are obtained by *Dependency-closure* procedure.

	A	B	C	D	E
AB	2	2	0	0	0
BC	1	2	2	0	AB
DE	1	0	0	2	2

Figure 7: E depends on BC via AB

In Figure 7, E depends on BC via AB. It is possible that E might depend on BC through some other determinant key, too. In which case is will not matter which determinant key is used in Figure 7 to represent this dependency. One issue to be careful of is that by updating the DM matrix to reflect transitive dependencies some direct dependencies may fade away.

Consider FDs = {A→B, B→A and B→C}. The DM and DG matrices are shown in Figure 8.

	A	B	C
A	2	1	0
B	1	2	1

(a): Dependency Matrix

	A	B
A	1	1
B	1	1

(b): Directed Graph Matrix

Figure 8: The DM and DG matrices

By applying the path finding algorithm, the updated matrix is shown in part (a) of Figure 9. As it can be seen from part (a) of Figure 8, the direct dependency of C to B has faded away. To tackle this deficiency, the following *Circular-Dependency* algorithm is designed. This algorithm internally uses the FindOne recursive algorithm. The latter will find the direct dependency, if any, and replace the transitive one. This is reflected in part (b) Figure 9.

	A	B	C
A	2	1	B
B	1	2	A

(a)

	A	B	C
A	2	1	B
B	1	2	1

(b)

Figure 9: The original B→C is returned

In Figure 10, DM2 represents the initial dependency matrix.

```
Circular-Dependency ()
  {
   for ( i=0; i<n; i++)
     for(j=0; j<m; j++)
         if(DM[i][j]!= {0,1,2})
             if(FindOne (i, j, j, n)  && DM2[i][j]==1)
                DM[i][j]=1;
 }
int FindOne (int i, element  j, int k, int n)
 {
   if(DM[j][k]==1 && n>=1)    return 0;
   elseif (n<1)  return 1;
   else   return FindOne (i, DM[i][k], k, n-1);
}
```

Figure 10: Replacing transitive dependency with original direct dependency

Example 2: Consider the following case taken from [8]: Relation GH {A, B, C, D, E, F, G, H, I, J, K, L} with dependencies: FDs = {A→BC, E→AD, G→AEJK, GH→FI, K→AL, and J→K}. The corresponding dependency graph is given in Figure 11.

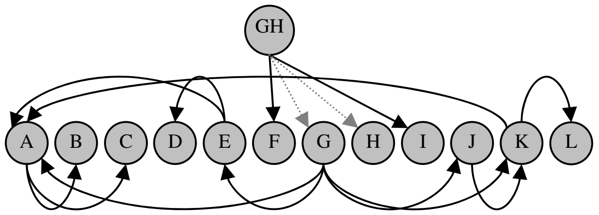

Figure 11: Dependency graph for Example 2

Figure 12 shows the original DM for Figure 11.

	A	B	C	D	E	F	G	H	I	J	K	L
A	2	1	1	0	0	0	0	0	0	0	0	0
E	1	0	0	1	2	0	0	0	0	0	0	0
G	1	0	0	0	1	0	2	0	0	1	1	0
GH	0	0	0	0	0	1	2	2	1	0	0	0
K	1	0	0	0	0	0	0	0	0	0	2	1
J	0	0	0	0	0	0	0	0	0	2	1	0

Figure 12 : Initial Dependency Matrix for Figure 11

Figure 13 is the corresponding DG matrix.

	A	E	G	GH	K	J
A	1	-1	-1	-1	-1	-1
E	1	1	-1	-1	-1	-1
G	1	1	1	-1	1	1
GH	-1	-1	1	1	-1	-1
K	1	-1	-1	-1	1	-1
J	-1	-1	-1	-1	1	1

Figure 13 : The DG matrix for Example 2

The path matrix is shown in Figure14.

	A	E	G	GH	K	J
A	1	-1	-1	-1	-1	-1
E	1	1	-1	-1	-1	-1
G	1	1	1	-1	1	1
GH	1	1	1	1	1	1
K	1	-1	-1	-1	1	-1
J	1	-1	-1	-1	1	1

Figure 14: Determinant key transitive dependencies

New dependencies are applied to the DM and Figure 15 is the semi-final result.

	A	B	C	D	E	F	G	H	I	J	K	L
A	2	1	1	0	0	0	0	0	0	0	0	0
E	1	A	A	1	2	0	0	0	0	0	0	0
G	K	E	E	E	1	0	2	0	0	1	J	K
GH	K	G	G	G	G	1	2	2	1	G	J	K
K	1	A	A	0	0	0	0	0	0	0	2	1
J	K	K	K	0	0	0	0	0	0	2	1	0

Figure 15 : Dependency closure matrix

It is now the time to replace direct dependencies which might have disappeared by applying transitive dependencies. However, the FindOne algorithm does not discover any fade away dependency. Therefore, Figure 15 shows the optimal dependency set. Entries with value 1 are identify components of this set.

We are now in a position to obtain candidate keys. A candidate key is a set of attributes to which all other attributes depend on. From the final DM we notice that GH has this property.

There are other sets of attributes which can be considered as candidate keys. For example, the set of {G, F, H, I} could be considered as a candidate key. However, the set with the least number of attributes amongst the determinant keys will be considered the primary key in the following discussions.

3. THE PROPOSED NORMALIZATION PROCESS

It is assumed that the reader is familiar with the definitions of different normal forms. On the other hand, tables of a relational database are assumed to be in 1NF form to begin with. Our proposed 2NF and 3NF normalization process makes use of both dependency and determinant key transitive dependencies.

3.1 Second Normal Form (2NF)

To proceed with the 2NF, it is assumed that the table is already in 1NF form. The resulting 1NF relation is:

GH_Relation :{ <u>GH</u>, A, B, C, D, E, F, I, J, K, L}

The goal is to discover all partial dependencies. To produce the 2NF form, we should find all partial dependencies. To do this, the DM is scanned row by row (ignoring the primary key row), starting from the first row. If all values of the simple keys that make up the determinant key of the row being scanned are equal to 2 and the values of the corresponding columns of the candidate key are equal to 2, then a partial dependency is found.

In Figure 15, the dependency of G to GH is partial. Therefore, we have to create a new table. From the DM matrix, we notice that E and J are directly dependent to G. The new table will be composed of G, E, J, and all simple keys which are transitively dependent on G. The transitive dependencies are obtained from the determinant key transitive dependencies matrix. G is the primary key of this table. There is no other partial dependency. In Figure 16, the DM matrix is partitioned into two new DMs corresponding to new tables.

	A	B	C	D	E	G	J	K	L
A	2	1	1	0	0	0	0	0	0
E	1	A	A	1	2	0	0	0	0
G	K	E	E	E	1	2	1	J	K
K	1	A	A	0	0	0	0	2	1
J	K	K	K	0	0	0	2	1	0

(a): **G_Relation :**{ <u>G</u>, E, J, K, A, B, C, D, L}

	F	G	H	I
GH	1	2	2	1

(b): **GH_Relation :**{ <u>GH</u>, F, I}

Figure 16 : Database normalized up to 2NF

3.2 Third Normal Form (3NF)

In order to transform the relations into 3NF, each DM is scanned row by row starting from the first row. If a determinant key is encountered whose dependency is neither partial (from Figure 16) nor it is wholly dependent on part of the primary key [9] a separate table has to be formed. Of course, if a table is previously formed a duplicate is not generated. This new table will include the determinant key and all other attributes which are transitively depend on this key. As it can be seen, there is no transitive dependency in part (b) of Figure 16. However, dependencies of A, E, K, and J in part (a) are of transitive form. Each of these dependencies led to production of a new table.

	F	G	H	I
GH	1	2	2	1

(a)

	J	K
J	2	1

(b)

	A	K	L
K	1	2	1

(c)

	E	G	J
G	1	2	1

(d)

	A	D	E
E	1	1	2

(e)

	A	B	C
A	2	1	1

(f)

Figure 17: Database Normalized up to 3NF

3.3 The BCNF Normal Form

For a relation with only one candidate key, 3NF and BCNF are equivalent. To transform the relations to BCNF requires the creation of new relation for each transitivity dependency. The resulting BCNF relations are:

GH_Relation :{ <u>GH</u>, F, I},**J_Relation :**{ <u>J</u>, K}
K_Relation :{ <u>K</u>, A, L},**G_Relation :**{ <u>G</u>, E, J}
E_Relation :{ <u>E</u>, A, D} and **A_Relation :**{ <u>A</u>, B, C}.

To develop the process of generating BCNF form, consider the case where there is more than one candidate key for the table being normalized.

3.4 A Complete Normalization Example

The following is a complete example with multiple candidate keys.

Example 3: Consider the following case taken from [9]: Relation AB:{A, B, C, D, E, F, G, H} with dependencies: FDs = {AB→CEFGH, A→D, F→G, BF→H, BCH→ADEFG and BCF→ADE}.

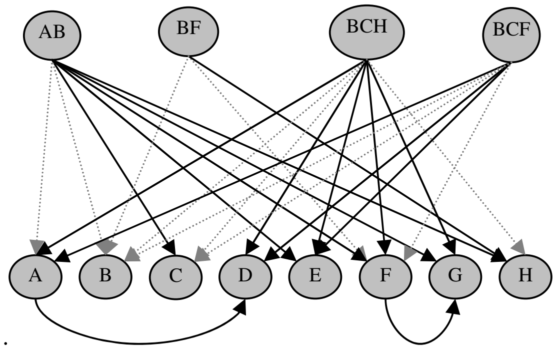

Figure 18: Dependency graph for Example 3

Figure 19 shows the original DM for Figure 18.

	A	B	C	D	E	F	G	H
AB	2	2	1	0	1	1	1	1
A	2	0	0	1	0	0	0	0
F	0	0	0	0	0	2	1	0
BF	0	2	0	0	0	2	0	1
BCH	1	2	2	1	1	1	1	2
BCF	1	2	2	1	1	2	0	0

Figure 19: Initial Dependency Matrix for Figure 18

Figure 20 is the corresponding DG matrix.

	AB	A	F	BF	BCH	BCF
AB	1	1	1	1	1	1
A	-1	1	-1	-1	-1	-1
F	-1	-1	1	-1	-1	-1
BF	-1	-1	1	1	-1	-1
BCH	1	1	1	1	1	1
BCF	1	1	1	1	-1	1

Figure 20: he DG matrix for Example 3

The path matrix is shown in Figure 21.

	AB	A	F	BF	BCH	BCF
AB	1	1	1	1	1	1
A	-1	1	-1	-1	-1	-1
F	-1	-1	1	-1	-1	-1
BF	-1	-1	1	1	-1	-1
BCH	1	1	1	1	1	1
BCF	1	1	1	1	1	1

Figure 21: Determinant key transitive dependencies

New dependencies are applied to the DM and Figure 22 is the semi-final result.

	A	B	C	D	E	F	G	H
AB	2	2	1	A	BCH	BCH	F	BF
A	2	0	0	1	0	0	0	0
F	0	0	0	0	0	2	1	0
BF	0	2	0	0	0	2	0	1
BCH	1	2	2	AB	AB	AB	AB	2
BCF	1	2	2	AB	AB	2	AB	AB

Figure 22: Dependency closure matrix

It is now the time to restore direct dependencies which might have been replaced by transitive dependencies. The FindOne algorithm discovers all fade away dependencies. One such dependency is shown in Figure 23 in the intersection of AB and F and AB and E.

	A	B	C	D	E	F	G	H
AB	2	2	1	A	1	1	F	BF
A	2	0	0	1	0	0	0	0
F	0	0	0	0	0	2	1	0
BF	0	2	0	0	0	2	0	1
BCH	1	2	2	AB	AB	AB	AB	2
BCF	1	2	2	AB	AB	2	AB	AB

Figure 23: After Circular Dependency

3.4.1 Candidate Keys of Example 3

A candidate key is a set of attributes to which all other attributes completely depend on. AB is a candidate key because all simple keys either directly depend on AB or via a determinant key which is not qualified to be a candidate key. For other potential candidate keys BCH and BCF there are some dependencies of simple keys which are via a potential candidate key. For example, C depends on BCH via AB which is a potential candidate key. These kinds of dependencies have to be reexamined to make sure whether the dependency persists if the dependency through the potential candidate key is ignored. This can be done by applying the Dependency-closure routine on the initial DM of the relation. However, a modification has to be considered for the Dependency-closure routine which is to be used here. We would like to ignore dependencies of a potential candidate key to another potential candidate key. To do so, the statement

$$if\,(i != j \;\;\&\&\;\; Path[i][j] != -1)$$

is replaced by

$$if\,(i != j \;\;\&\&\;\; Path[i][j] != -1 \;\;\&\&\;\; j \notin \{Potential\;\; Candidate\;\; key\;\; set\})$$

The result is depicted in Figure 24. This figure shows the set of optimal dependencies and real candidate keys.

	A	B	C	D	E	F	G	H
AB	2	2	1	A	1	1	F	BF
A	2	0	0	1	0	0	0	0
F	0	0	0	0	0	2	1	0
BF	0	2	0	0	0	2	F	1
BCH	1	2	2	A	1	1	F	2
BCF	1	2	2	A	1	2	F	BF

Figure 24: The set of optimal dependencies

In the following we will act on 2NF and 3NF. From now on we assume AB is the primary key.

3.4.2 2NF of Example 3

The normalization of 1NF relations to 2NF involves the removal of partial dependencies on the primary key. If a partial dependency exists, we remove the functionally dependent attributes from the relation by placing them in a new relation with a copy of their determinant. On identifying the functional dependencies, we continue the process of normalization the relation. We begin by testing whether the relation is in 2NF by identifying the presence of any partial dependencies on the primary key. We see that the attribute D is partially dependent on part of the primary key, namely A. On the other hand, the attributes C, E, and F are fully dependent on the whole primary key. We note that H is not wholly dependent on part of the primary key AB and therefore does not violate 2NF. Hence, we need to create a new relation called A_relation. As a result, DM is also partitioned as follows.

	A	B	C	E	F	G	H
AB	2	2	1	1	1	F	BF
F	0	0	0	0	2	1	0
BF	0	2	0	0	2	F	1
BCH	1	2	2	1	1	F	2
BCF	1	2	2	1	2	F	BF

(a): **AB_Relation :**{ AB, C, E, F, G, H}

	A	D
A	2	1

(b): **A_Relation :**{ A, D}

Figure 25: Database normalized up to 2NF

3.4.4 3NF of Example 3

The normalization of a 2NF table to 3NF involves the removal of transitivity dependencies. If a transitivity dependency exists, we remove the transitivity dependency attributes from the relation by placing them in a new relation along with a copy of their determinant. First, we examine the functional dependencies within the A and AB relations, which are as figure 25.

The A_relaton does not have transitive dependencies on the primary key. However, although all the non-primary-key attributes within the AB_Relation are functionally dependent on primary key, G is also dependent on F. This is an example of a transitive dependency, which occurs when a non-primary-key attribute is dependent on another non-primary-key attribute. Although BF → H, BF is not a non-primary key (as B is part of the primary key). Therefore, we do not remove the dependency at this stage. In other words, this dependency is not wholly transitivity dependent on non-primary-key attribute and therefore does not violate 3NF. To transform the AB_Relation into third normal form, we must first remove transitive dependency. It is done by creating two new relations called F_Relation and AB_Relation [9]. The resulting 3NF relations have in figure 26.

	A	B	C	E	F	H
AB	2	2	1	1	1	BF
BF	0	2	0	0	2	1
BCH	1	2	2	1	1	2
BCF	1	2	2	1	2	BF

(a): **AB_Relation :**{ AB, C, E, F, H}

	A	D
A	2	1

(b): **A_Relation :**{ A, D}

	F	G
F	2	1

(c): **F_Relation :**{ F, G}

Figure 26: Third normal form of Example 3

3.4.4 Boyce-Codd Normal Form (BCNF)

We now examine A, AB and F relations to determine whether they are in BCNF. A relation is in BCNF if every determinant of a relation is a candidate key. Therefore, to test for BCNF, we simply identify all the determinants and make sure they are candidate keys. We can see that from DMs in Figure 26 DM (b) and DM(c) their relations are already in BCNF. To transform AB_Relation into BCNF, we must remove the dependency that violates BCNF by creating one new relation for BF→H. The resulting BCNF relations have in figure 27.

In this example, the decomposition of the original AB_Relation to BCNF relations has resulted in 'loss' of the functional dependency BCH→AEF. On the other hand, we recognize that if the functional dependency BF→H is not removed, the AB_relation will have data redundancy [9]. In practice, some designers stop at 3NF and do not proceed to BCNF. In which case, there may exist some redundancies in the database designed.

	A	B	C	E	F
AB	2	2	1	1	1
BCF	1	2	2	1	2

(a): **AB_Relation :**{ AB, C, E, F}

	F	G
F	2	1

(d): **F_Relation :**{ F, G}

	B	F	H
BF	2	2	1

(b): **BF_Relation :**{ BF, H}

	A	D
A	2	1

(c): **A_Relation :**{ A, D}

Figure 27: Database normalized up to BCNF

4. CONCLUSION

A new complete automated relational database normalization method is presented. The process is based on the generation of dependency matrix, directed graph matrix, and determinant key transitive dependency matrix. The details of the methods for 2NF, 3NF, and BCNF are discussed. Two examples, 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. A nine thing about the developed algorithms is the automatic distinguishing of one primary key for every final table that is generated. We believe that the algorithms are very efficient. However, we will compare our algorithms with other similar algorithms, in the future.

5. REFERENCES

[1] 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.

[2] Kolahi, S., Dependency-Preserving Normalization of Relational and XML Data, Journal of Computer System Science, Vol. 73(4): pp. 636-647, 2007.

[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] Akehurst, D.H., B. Bordbar, P.J. Rodgers, and N.T.G. Dalgliesh, Automatic Normalization via Metamodelling, ASE 2002 Workshop on Declarative Meta Programming to Support Software Development, 2002.

[8] Date, C.J., An Introduction to Database Systems, Addison-Wesley, Seventh Edition 2000.

[9] Connoly, Thomas, Carolyn Begg: Database Systems. A Practical Approach to Design, Implementation, and Management , Pearson Education, Third edition, 2005.