ALGORITHM FOR RELATIONAL DATABASE NORMALIZATION UP TO 3NF

Moussa Demba

Department of Computer Science & Information, Aljouf University
Sakaka, Kingdom of Saudi Arabia
bah.demba@ju.edu.sa

ABSTRACT

When an attempt is made to modify tables that have not been sufficiently normalized undesirable side-effects may follow. This can be further specified as an update, insertion or deletion anomaly depending on whether the action that causes the error is a row update, insertion or deletion respectively. If a relation R has more than one key, each key is referred to as a candidate key of R. Most of the practical recent works on database normalization use a restricted definition of normal forms where only the primary key (an arbitrary chosen key) is taken into account and ignoring the rest of candidate keys.
In this paper, we propose an algorithmic approach for database normalization up to third normal form by taking into account all candidate keys, including the primary key. The effectiveness of the proposed approach is evaluated on many real world examples.

KEYWORDS

Relational database, Normalization, Normal forms, functional dependency, redundancy.

1. INTRODUCTION

Normalization is, in relational database design, the process of organizing data to minimize redundancy. It usually involves dividing a database into two or more tables and defining relationships between the tables. The objective is to isolate data so that additions, deletions, and modifications of a field can be made in just one table and then propagated through the rest of the database via the defined relationships. Edgar Codd, the inventor of the relational model, also introduced the concept of normalization and Normal Forms (NF). The normal forms of relational database theory provide criteria for determining a table's degree of vulnerability to logical inconsistencies and anomalies. The higher the normal form applicable to a table, the less vulnerable it is. In general, normalization requires additional tables and some designers find this first difficult and then cumbersome.

Violating one of the first three rules of normalization, make the application anticipates any problems that could occur, such as redundant data and inconsistent dependencies.

When using the general definitions of the second and third normal forms (2NF and 3NF for short) we must be aware of *partial* and *transitive dependencies* on all *candidate keys* and not just the primary key. This can make the process of normalization more complex; however, the general definitions place additional constraints on the relations and may identify hidden redundancy in relations that could be missed [1].

A functional dependency $X \rightarrow A$ is partial if some attribute $B \in X$ can be removed from X and the dependency still holds. Let A, B, and C be attributes of a relation R, $A \rightarrow B$ and $B \rightarrow C$ be two

DOI : 10.5121/ijdms.2013.5303

dependencies that hold in R. Then C is transitively dependent on A via B (provided that A is not functionally dependent on B or C) [1].

Most of the recent works on database normalization are web-based tools without presenting algorithms and define the two notions, 2NF and 3NF, with respect to primary keys only, ignoring the other candidate keys as in [2], [3, [4], [5]. However, the original definitions of the two notions as given in [6], [7] , [1], [8] consider all candidate keys. To see the difference between the two approaches, suppose we have a schema relation R(A,B,C,D,E,F) together with the dependencies F={A→BCDEF; BC→ADEF; B →D; E→F }

In this example, we have two candidate keys *A* and *BC*. If A is selected as the primary key of the relation, and ignoring BC then there is no partial dependencies on A, implying that the relation is in 2NF. But if all candidate keys are taken into account (that is the general definition is considered), although there are no partial dependencies on A, we have a partial dependency B→D on candidate key *BC*, implying that the relation is not in *2NF*. This make normalization process particularly confusing for many designers. In this paper, we propose an algorithmic approach for database normalization that uses the original and general definitions of normal forms. The general definitions take into account all candidate keys of a schema relation. For each synthesized relation, a primary key is also generated. The algorithms are presented step-by-step so designers can learn and implement them easily. Hereafter, all input schema relations are supposed at least in first normal form (1NF).

Throughout the paper, R represents a relational schema; *A, B, C*,... denote attributes; *X, Y* and *Z* denote set of attributes, and F a set of functional dependencies.

The rest of the paper is organized as follows: section 1 presents some basic concepts and notations, in section 2 we present a procedure for removing redundant attributes, in section 3 we present an algorithm for removing redundant attributes and another for redundant dependencies. Section 4 presents an algorithm for classifying dependencies into full and partial and in section 5 we present the algorithms for the decomposition into 2NF and 3NF respectively. In section 6 a complete and practice example is presented and in section 7 we conclude the paper.

2. REMOVING REDUNDANT ATTRIBUTES AND DEPENDENCIES

Before determining partial dependencies, some redundant functional dependencies could be removed. To do that, many algorithms have been proposed for removing redundant dependencies, called *minimal cover* [9], [10]. To achieve this goal, one needs to compute the closure of a given set of attributes and then remove redundant attributes.

Definition 1.

Given a relation *R*, a set of attributes *X* in *R* is said to *functionally* determine another set of attributes *Y*, also in *R*, written $X \rightarrow Y$, if and only if each *X* value is associated with precisely one *Y* value; *R* is then said to *satisfy* the *functional dependency* $X \rightarrow Y$.

Definition 2.

A set of functional dependencies *F* is in *canonical form* if each functional dependency X→A in *F*, *A* is a singleton attribute.

Hereafter, all functional dependencies are supposed in canonical form.

Let X be a set attributes and F be a set of functional dependencies. Let X^+ be the set of all attributes that depend on a subset of X with respect to F, i.e., X^+ is the set of attributes Z, such that $X \rightarrow Z \in F$. X^+ is called the closure of X w.r.t F.

The *Algorithm A1* computes the closure of a given set of attributes w.r.t F:

***Algorithm A1*:** computes X^+.

```
Input:  A relation R, a set of functional dependencies F and a
        set X of attributes.
Output:  X+, the closure of X.
    X+:= X
   While there is a fd Y→ A∈ F
      if Y⊆X+  and A⊄ X+  then
            X+:=X+ ∪{A}
```

To check if a functional dependency $Y \rightarrow A$ holds in a set of dependencies F, we compute the closure of Y and check If $A \subseteq Y^+$. This test is particularly useful, as we will see later in the next algorithm.

We can use attribute closure to remove redundant dependencies from a set of functional dependencies. We can do this by examining each dependency in turn to see if it is redundant. A dependency is redundant if it can be inferred from the other dependencies, and can thus be removed.

Given a set of dependencies F, an attribute B is *extraneous* in $X \rightarrow A$ with respect to F if $A \in (X-B)^+$. If $B \in (X-B)^+$ then B is called *an implied extraneous attribute*. If B is extraneous, but not implied, then it is *nonimplied*. For example, suppose we have the functional dependencies $A,B \rightarrow C$ and $A \rightarrow C$ in F. Then B is a *nonimplied* extraneous attribute in $A,B \rightarrow C$. As another example, suppose we have the dependencies $A,B \rightarrow C$ and $A \rightarrow B$ in F. Then B is an *implied extraneous attribute* in $A,B \rightarrow C$.

F is called *partially left-reduced* if no attributes are implied extraneous attribute. The elimination of non-implied extraneous attribute is postponed until the time that redundant dependencies are eliminated, using the *Algorithm A3*. Many algorithms, sometimes difficult to reuse, have been proposed in [11], [12] for removing extraneous attributes.

In our approach, we propose the following algorithm to eliminate implied extraneous attributes by minimizing attribute closure computations. *Algorithm A3* removes any implied extraneous attributes. Our observation is that any implied extraneous attribute has to appear both on the left-hand side of at least one dependency and on the right-hand side of at least one dependency. Therefore, *categorizing* attributes according to their appearance on the left-hand side and right-hand side of dependencies could reduce the number of attributes closure computation.We define the sets:

ro={A/ A occurs only on the right hand side of dependencies in F}
lo={A/ A occurs only on the left hand side of dependencies in F }
lr={A/ A∉ ro and A∉ lo }

Note that $lo \cap ro = \phi$, $lo \cap lr = \phi$ and $ro \cap lr = \phi$. Moreover, the set *lo*, if not empty, contains key-attributes.

Lemma 1. If B is an implied extraneous attribute in a set of dependencies F, then B∈ *lr*.

Proof. It is easy to see that an implied extraneous attribute, as defined above, has to occur on the left-hand side of a dependency, then cannot be an element of the set ro.

Lemma 2. If B is an extraneous implied attribute, then B∉ *lo* and B∉ *ro*.

In contrast to the algorithm proposed in [12] that tests the redundancy of all attributes in (*lo*∪*lr*), we will test only for the attributes in *lr*. Let |X| denotes the cardinality (number of attributes) of X.

Algorithm A2: Removes implied extraneous attributes.

```
Input: F, a set of functional dependencies.
Output: G, a partially left-reduced set of dependencies.
Initialize G:=F
1.for each fd X→A∈G  do
   if |X|>1 then set Y:=X
      for any attribute B∈Y do
         if B∈lr then
           set H:=G-(X→A)∪{(Y-B)→A}
           if B∈(Y-B)+ under H then
               set Y:=Y-{B}
      end
      if X≠Y then G:=H
   end
  end
2.Remove all duplicated dependencies in G, if any.
```

This algorithm is substantially better than the one proposed in [12].

Example 1 [12]: Let F be the set of dependencies:

AB→D; B→A; ABC→D; D→A; AB→E; B→H; ABC→J; AB→F;
B→G; ABC→K; BN→H; AB→G

G:=F, lr={A, D}, lo={B,C} and ro={E,F,G,H.J,K}

According to lemma 2, there is no need to check for the attributes in *lo* or in *ro*.

-For AB→D.
B cannot be an implied extraneous attribute as B∉lr, then we check for A:
we have A∈(Y-A)+; Y=Y-A
G is changed by replacing AB→D by B→D :

B→D; B→A; BC→D; D→A; B→E; B→H; BC→J; B→F; B→G;
BC→K; BN→H; B→G

-For BC→D.
G will not change as B,C∉lr

-For BN→H.

G will not change as B,N∉lr

B→G is duplicated, then has to be removed we get the partially left-reduced dependencies F' of F:

B→D; B→A; BC→D; D→A; B→E; B→H; BC→J; B→F; B→G; BC→K;
BN→H

There is no more extraneous implied attributes in F', while the *nonimplied* extraneous attribute N, it will be removed by the next procedure.

The next step consists to remove redundant dependencies. The main idea is to compare the lhs of dependencies that have the same rhs: if a dependency f_1: X→A is redundant in F, then there is at least a dependency f_2: Y→A in F where $X \rightarrow Y \in F^+$. We write X_F^+ to mean that the closure of *X* w.r.t a set of dependencies F. Algorithmically this can be expressed as follows:

Algorithm A3: Removes redundant dependencies.

```
Input: F, a set of partially left-reduced dependencies.
Output: Fm, a minimal cover of F
1. Fm:= F
2. for each X→A∈Fm
     while there exists a Y→A∈Fm
       G:= Fm-(Y→A)
       if X⊆Y_G^+ then
           Fm:= G
       end
      end
     end
3. Fm is the  minimal cover of F.
```

To illustrate the algorithm, let F be the dependencies given in example 1.

1. Initialization of F_m to:

B→D; B→A; BC→D; D→A; B→E; B→H; BC→J; B→F; B→G; BC→K;
BN→H

2. Summarized in table 1.

3. F_m= { B→D; B→E; B→F; B→H; B→G; BC→J; BC→K; D→A }

Table 1. Sketch elimination of redundant dependencies.

Pass	X→A	Y→A	F_m
1	B→D	BC→D, B∈ $(BC)^+$	$F_m := F_m - (BC \rightarrow D)$
2	B→E		
3	B→F		
4	B→A		
5	B→H	BN→H, B∈ $(BN)^+$	$F_m := F_m - (BN \rightarrow H)$
6	B→G		
7	BC→J		
8	BC→K		
9	D→A	B→A , D∈ B^+	$F_m := F_m - (B \rightarrow A)$

In [12] they claimed that their (fast) algorithm computes 5 closures and the standard one 19 closures to remove redundant dependencies. As we can see in the table 1, only 3 computations are needed using our *Algorithm A3*, i.e., passes 1, 5 and 9, that is a substantial improvement. Moreover, for the other non-redundant dependencies, no extra effort is needed.

Unfortunately, there can be several minimal covers for a set of functional dependencies. We can always find at least one minimal cover F_m for any set of dependencies F using *Algorithm A3*.

Theorem 1. Let F be a set of partially left-reduced dependencies, and F_m its *minimal cover* obtained by the Algorithm A3. Then we have $F^+ \equiv F_m^+$.

3. FULL AND PARTIAL DEPENDENCIES

A functional dependency X → A is a *full functional dependency* if removal of any attribute B from X means that the dependency does not hold any more; that is, for any attribute B∈X, (X - B) does not functionally determine A. A functional dependency X→A is a *partial dependency* if A is not a key-attribute and some attribute B∈X can be removed from X and the dependency still holds; that is, for some B∈X, (X - B) → A [7]. Full and partial dependencies are generated as follows:

Algorithm A4: Determines full and partial dependencies.

```
 Input: Fm, a minimal cover set of F.
Output: Fp, Ff, sets of  partial and full dependencies in
        Fm respectively.
Let Fp:=ϕ and  Ff:=Fm.
 For each dependency X→A∈Ff
   if  X is a proper subset of a candidate key and A is not a
    key attribute then
    Fp:=Fp∪{X→A}
    Ff:=Ff-(X→A)
   while there is a fd Z→B∈Ff s.t. Z∈A+ do
    Fp:=Fp∪{Z→B}
    Ff:=Ff-(Z→ B)
   end
  end
end
```

In the If-statement, we make condition on the candidate keys and not the primary key, because if a relation has many keys, all have to be considered (see the definition of 2NF in the next section).

We can continue our running example by initializing F_f to:

{B→ D; B→ F; B→ G; D→ A; B→ E; B→ H; BC→ J; BC→ K}

we have one candidate key BC, that is the primary key. $F_p=\phi$. For the lines 1-5, B⊆(BC), then we have partial dependencies.

Table 2. Sketch determination of F_f and F_P.

X→ A	**F_f**	**F_P**
B→ D	$F_f := F_f$ –{B→ D; D→ A}	$F_P := F_P \cup${B→ D; D→ A}
B→ E	$F_f := F_f$ – (B→ E)	$F_P := F_P \cup${B→ E }
B→ F	$F_f := F_f$ – (B→ F)	$F_P := F_P \cup${B→ F}
B→ H	$F_f := F_f$ – (B→ H)	$F_P := F_P \cup${B→ H }
B→ G	$F_f := F_f$ – (B→ G)	$F_P := F_P \cup${B→ G }
BC→ J		
BC→ K		

The final result is:

F_f ={BC→ J; BC→ K}
F_P ={ B→ D; D→ A; B→ E; B→ F; B→ G; B→ H }

Lemma 3. $F_m = F_f \cup F_P$.

Lemma 4. Let R be a relation schema and F_m its minimal cover dependency set. If there is no partial dependency in R then $F_p=\phi$.

4. NORMALIZATION

4.1. The Second Normal Form

A relation is in second normal form (abbreviated 2NF) if it is in 1NF and no non-key attribute is partially dependent on any candidate key [7], [1]. In other words, no X→Y where X is a strict subset of a candidate key and Y is a non-key attribute.

Simply, a table is in 2NF if and only if it is in 1NF and every non-key attribute of the relation is either dependent on the whole of any candidate key, or on another non-key attribute.

The following algorithm is used to decompose a 1NF relation schema R into 2NF:

Algorithm A5: Decomposes into *2NF*.

```
Input: R, Ff and Fp.
Output: a set R^2NF of relations into 2NF.
Let G:=Fp and Xf the set of attributes in Ff.
R^2NF:=ϕ
 for each Y→A∈G do
   if Y is a proper subset of a candidate key then
```

```
      - create R_Y(Y_G+).
      - choose Y as the primary key of R_Y.
      - set X_f:=X_f-{B} for any non-key attribute B∈Y_G+.
      - remove from G any dependency whose lhs is in Y_G+.
      - R^2NF:= R^2NF ∪ R_Y(Y_G+)
    end
  end
R^2NF:= R^2NF ∪ R_K(X_f), where K∈X_f is the (chosen) primary key of R.
```

If Y is a key-attribute that means that Y is causing a partial dependency. The statement $X_f:=X_f-\{B\}$ is justified by the fact that *Y* is already in X_f. Moreover, the resulting decomposition preserves the functional dependencies of F_m, because $Y \rightarrow A$ belongs to R_Y and $Y \subseteq X_f$.

Theorem 2. Let *R* be a *1NF* relation. We have the following results:

a. If $F_p=\phi$ then *R* is automatically in *2NF*.
b. If $F_p \neq \phi$ then *R* is not in *2NF*.

Proof.

a. If $F_p=\phi$, i.e., there is no partial dependency in R, that means also that the first *if-condition* of the algorithm will never hold. Then R_K is the only relation created by the Algorithm A5. As R_K is partial dependency free (no partial dependency exists in F_f), then $R= R_K$ is in *2NF*.
b. If $F_p \neq \phi$, i.e., the first *if-condition* of the algorithm holds at least once. Therefore, R is not in *2NF* as $R \neq R_K$.

Let's assess our example:

$F_f=\{ BC \rightarrow J, BC \rightarrow K \}$
$F_p=\{ B \rightarrow D; D \rightarrow A; B \rightarrow E; B \rightarrow F; B \rightarrow G; B \rightarrow H \}$

and $F_m=F_f \cup F_p$.

As we can see all attributes in F_f depend fully on the only candidate key {BC} and all the attributes in F_p depend directly/indirectly on the key-attribute B, i.e., B is causing a partial key. Therefore, the relations R_{BC} and R_B are created:

$R_{BC}(\underline{B,C},J,K)$
$R_B(\underline{B},D,A,E,F,G,H)$

Note that when a 1NF relation has no composite candidate keys (candidate keys consisting of more than one attribute), the relation is automatically in 2NF. If there is no composite candidate keys, then $F_p=\phi$ and by theorem 2 the relation is in 2NF.

4.2 Third Normal Form

A relation is in third normal form (abbreviated 3NF) if it is in 2NF and none of its non-key attributes are transitively dependent upon any candidate key [6], [1], [7]. An alternative (simpler) definition is a relation is in 3NF if in every non-trivial dependency $X \rightarrow A$ either X is a super key or A is a key attribute (i.e., A is contained within a candidate key).

The transitive dependencies set on 2NF relations is defined by:

F_t={ $X \rightarrow A \in F_m$} such that

i. *X* is not a candidate key, and
ii. *A* is not a key attribute.

Note that partial dependencies violate the 3NF. We have the following characterization.

Theorem : A relation schema R is in third normal form if and only if for every key K of R and every nonkey-attribute A depends directly on K.

The following procedure is used to decompose a 1NF relation into 3NF.

Algorithm A6. Decomposes into 3NF.

```
Input: R(X) a 1NF relation, and the set of dependencies F_t.
       Where X is the set of attributes in R.
Output: a set R^3NF of relations into 3NF.
Let G:=F_t and R^3NF:=ϕ
For each dependency Y→A∈G do
      - create a new relation R_Y, if not already created,
      - add Y and A in R_Y, if not already added,
      - consider Y as the primary key of R_Y,
      - R^3NF:= R^3NF ∪ R_Y
      - remove Y→A from G, and
      - if A∈X then
           set R(X):=R(X-{B}) for any B∉Y and B∈A_G+.
        end
end
  R^3NF:= R^3NF ∪R(X)
end
```

Lemma 5. If $F_t \neq \phi$, then the input relation R is not in 3NF.

Now, we can complete our example given the initial 1NF relation with BC the only candidate key BC:

R_{BC}(A, <u>B, C</u>, D, E, F, G, H, J, K)

together with the following dependencies:

F_m={BC→J; BC→K; B→D; D→A; B→E; B→F; B→G; B→H }
F_t={B →D; D→A; B→E; B→F; B→G; B→H }

According to the lemma 5, R_{BC}(X), where *X* is the set of attributes, is not in 3NF. Therefore new relations R_B(<u>B</u>, D, E, F, G, H) and R_D(<u>D</u>, A) are then created and the attributes D, E, F, G, H are removed from *X* because they are in X^+. We get the final relations into 3NF:

R_{BC}(<u>B, C</u>, J, K)
R_B(<u>B</u>, D, E, F, G, H)
R_D(<u>D</u>, A)

Example 2: Consider the schema relation
R(isbn, invno, title, authors, publ, place, year)

together with the set of dependencies:

F_m={(title, authors, publ, place, year)→ isbn; isbn→ title; isbn→ authors;
isbn→ publ; isbn→place; isbn→ year; invno → isbn }

The attribute *invno* is the unique key of R. The set of partial dependencies $F_p=\phi$ and the set of transitive dependencies :
F_t= {(title, authors, publ, place, year)→ isbn; isbn→ title; isbn→ authors;
isbn→ publ; isbn→place; isbn→ year}

Therefore the relation R is in 2NF but not in 3NF because $F_t\neq\phi$. R can be easily normalized by decomposition using the algorithm A6 as follows:

R_1(title, authors, publ, place, year, isbn)
R_2(invno, title, authors, publ, place, year)

5. A COMPLETE EXAMPLE

To demonstrate the applicability of the above algorithms, consider the more realistic example presented in [1].

Suppose we want to normalize the *1NF* relation *ClientRental* up to *3NF*. This relation contains 9 attributes and 17 dependencies:

Clientrental (clientNo, propertyNo, clientName, pAddress, rentStart, rentFinish, rent,
ownerNo, ownerName)

with the following dependencies F:

f_1:	*propertyNo,rentStart*	→ *rentFinish*
f_2:	*clNo,propertyNo*	→ *rentFinish*
f_3:	*clNo,rentStart*	→ *rentFinish*
f_4:	*clNo*	→ *clName*
f_5:	*propertyNo,rentStart*	→ *clName*
f_6:	*clNo,propertyNo*	→ *rentStart*
f_7:	*ownerNo*	→ *oName*
f_8:	*propertyNo*	→ *oName*
f_9:	*clNo,rentStart*	→ *oName*
f_{10}:	*clNo,rentStart*	→ *pAddress*
f_{11}:	*propertyNo*	→ *pAddress*
f_{12}:	*propertyNo*	→ *rent*
f_{13}:	*clNo,rentStart*	→ *rent*
f_{14}:	*propertyNo*	→ *ownerNo*
f_{15}:	*clNo,rentStart*	→ *ownerNo*
f_{16}:	*clNo,rentStart*	→ *propertyNo*
f_{17}:	*propertyNo,rentStart*	→ *clNo*

Step 1: First of all, we call the *Algorithm A3* to generate a minimal cover F_m of F, as there is no implied extraneous attributes. Let $lhs(f_j) \subseteq lhs(f_i)^+$ means that the left-hand-side of the dependency f_j is in the closure of the left-hand-side of the dependency f_i, i.e. $lhs(f_i) \rightarrow lhs(f_j)$.

- f_1 is redundant because $lhs(f_2) \subseteq lhs(f_1)^+$
- f_3 is redundant because $lhs(f_2) \subseteq lhs(f_3)^+$
- f_5 is redundant because $lhs(f_4) \subseteq lhs(f_5)^+$
- f_8 is redundant because $lhs(f_7) \subseteq lhs(f_8)^+$
- f_9 is redundant because $lhs(f_7) \subseteq lhs(f_9)^+$
- f_{10} is redundant because $lhs(f_{11}) \subseteq lhs(f_{10})^+$
- f_{13} is redundant because $lhs(f_{12}) \subseteq lhs(f_{13})^+$
- f_{15} is redundant because $lhs(f_{14}) \subseteq lhs(f_{15})^+$

After removing the redundant dependencies (9), we get the minimal cover F_m :

f_2:	*clNo,propertyNo*	$\rightarrow$*rentFinish*
f_4:	*clNo*	$\rightarrow$*clName*
f_6:	*clNo,propertyNo*	$\rightarrow$*rentStart*
f_7:	*ownerNo*	$\rightarrow$*oName*
f_{11}:	*propertyNo*	$\rightarrow$*pAddress*
f_{12}:	*propertyNo*	$\rightarrow$*rent*
f_{14}:	*propertyNo*	$\rightarrow$*ownerNo*
f_{16}:	*clNo,rentStart*	$\rightarrow$*propertyNo*
f_{17}:	*propertyNo,rentStart*	$\rightarrow$*clNo*

We have three candidate keys (clNo, propertyNo), (clNo, rentStart) and (propertyNo, rentStart). No matter is the choice of the primary key, we can determine partial and transitive dependencies.

Step 2: We call the Algorithm A4 to generate F_f and F_p from F_m. The set of partial dependencies is:

F_p={ *clNo*$\rightarrow$*clName;*
propertyNo$\rightarrow$*pAddress;*
propertyNo$\rightarrow$*rent;*
propertyNo$\rightarrow$*ownerNo;*
ownerNo$\rightarrow$*oName*
}

As $F_p \neq \phi$ the relation *ClentRental* is not in 2NF. In fact, the attribute *clName* is partially dependent on the candidate keys (clNo, propertyNo) and (clNo, rentStart). Also, the attributes *pAddress, ownerNo* and *oName* are partially dependent on the candidate keys (clNo, propertyNo) and (rentStart, propertyNo).

And the set of full dependencies is:

F_f={ *clNo,propertyNo*$\rightarrow$*rentStart;*
clNo,propertyNo$\rightarrow$*rentFinish;*
clNo,rentStart$\rightarrow$*propertyNo;*
propertyNo,rentStart$\rightarrow$*clNo*
}

and the set of transitive dependencies is:

F_t={ *ownerNo→oName* }

Step 3: Decomposition of the relation *ClientRental* into 2NF:

We can choose (clNo,propertyNo) as the primary key, but no special consideration will be given to this key over the other candidate keys. Then the call to the algorithm A5 creates new relations. From F_f, we get the relation

$R_{(clNo,propertyNo)}$(clNo, propertyNo, rentStar, rentFinish)

R_{clNo}(clNo,clName)

$R_{propertyNo}$(propertyNo, pAddress, rent,ownerNo, oName)

The key-attributes *clNo* and *propertyNo* cause the violation of 2NF. The three relations are in Second Normal Form as every non-key attribute is fully functionally dependent on the primary key of the relation.

Step 4: Decomposition into 3NF:

As the input of the *Algorithm A6* we have $R_{(clNo,propertyNo)}$, R_{clNo}, $R_{propertyNo}$ and F_t. The only relation that contains the transitive dependency F_t is $R_{propertyNo}$. We then decompose $R_{propertyNo}$ into two relations and remove the attribute *oName* from that relation to get all relations into 3NF:

$R_{(clNo,propertyNo)}$(clNo,propertyNo, rentStar, rentFinish)
R_{clNo}(clNo,clName)
$R_{propertyNo}$(propertyNo, pAddress, rent, ownerNo)
$R_{ownerNo}$(ownerNo, oName)

The algorithms are dependency preserving as the original *ClientRental* relation can be recreated by joining the 3NF relations $R_{(clNo,propertyNo)}$, R_{clNo}, $R_{propertyNo}$ and $R_{ownerNo}$ through the primary key/foreign key mechanism.

6. CONCLUSION

In this paper we have presented algorithms for relational database normalization into 2NF and 3NF using their general definitions in a step-by-step feature. The first step before performing the procedure is to make a preprocessing on the set of dependencies to remove redundant dependencies. We have tested our algorithms on many realistic examples with multiple candidate keys taken from different sources.

This work has mainly the following major advantages: *(i)* the general and original definition of normal forms is used, *(ii)* the removal of redundant dependencies, *(iii)* in all phases, the computation of attributes closure are minimized compared to other algorithms although using a restricted definition of normal forms, and *(iv)* a primary key is determined for any generated relation.

REFERENCES

[1] Thomas, C., Carolyn, B. (2005) Database Systems, A Practical Approach to Design, Implementation, and Management, Pearson Fourth edition .

[2] Bahmani A., Naghibzadeh, M. and Bahmani, B. (2008) "Automatic database normalization and primary key generation", Niagara Falls Canada IEEE.

[3] Beynon-Davies, P. (2004) Database systems, Palgrave Macmillan, Third edition, ISBN 1–4039—1601–2.

[4] Dongare,Y. V., Dhabe,P. S. and Deshmukh, S. V. (2011) RDBNorma: "A semi-automated tool for relational database schema normalization up to third normal form", International Journal of Database Management Systems, Vol.3, No.1.

[5] Vangipuram, R., Velputa, R., Sravya, V. (2011) "A Web Based Relational database design Tool to Perform Normalization", International Journal of Wisdom Based Computing, Vol.1(3).

[6] Codd, E.F. (1972) "Further normalization of the data base relational model", In Database Systems, Courant Inst. Comptr. Sci. Symp. 6, R. Rustin, Ed., Prentice-Hall, Englewood Cliffs, pp. 33—64.

[7] Elmasri, R., Navathe, S.B. (2003) Fundamentals of Database Systems, Addison Wesley, fourth Edition.

[8] Date, C.J. (2004) Introduction to Database Systems (8th ed.). Boston: Addison-Wesley. ISBN 978-0-321-19784-9.

[9] Ullman, J.D. (1982) Principe of Database Systems. Computer Science Press, Rockville, Md.

[10] Maier, D. (1983) The Theory of relational Databases. Computer Science Press, Rockville, Md.

[11] Bernstein, P.A. (1976) "Synthesizing third normal form relations from functional dependencies", ACM Transactions on database Systems, vol.1, No.4, pp.277—298.

[12] Diederich, J., Milton, J. (1988) "New Methods and Fast Algorithms for Database Normalization", ACM Transactions on database Systems, Vol.13, No.3, pp. 339—365.