International Journal of Digital Information and Wireless Communications (IJDIWC) 3(2): 197-205
The Society of Digital Information and Wireless Communications, 2013 (ISSN: 2225-658X)

An Algorithmic Approach to Database Normalization

M. Demba
College of Computer Science and Information
Aljouf University, 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. Most of the recent works on database normalization use a restricted definition of normal forms where only the primary 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 minimize modification problems that could arise after modification of a table field. Edgar Codd, the inventor of the relational model, also introduced the concept of normalization and Normal Forms (NF). 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. Of course, 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 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 are given the schema relation R(A,B,C,D,E,F) together with the dependencies

F={A→BCDEF; BC→ADEF; B →D; E→F }.

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 considered here), 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.

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.

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 we conclude the paper.

2 BASIC DEFINITIONS

- A *super key* is a set of attributes which will uniquely identify each tuple in a relation.
- A *candidate key* is a minimal super key.
- A *primary key* is a chosen candidate key.
- An attribute *A* is a *key attribute* in a relation *R*, if *A* is part of some candidate key. Otherwise it is a *non-key attribute* in *R*, i.e., *A* is not a component of any candidate key.
- 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* → *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* → *Y*.
- A functional dependency X→Y is called *trivial* if *Y* is a subset of *X*.
- 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 and input schema relations are at least in first normal form (1NF).

3 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.

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 1* computes the closure of a given set of attributes w.r.t F:

***Algorithm 1*:** 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 $Y \rightarrow A \in F$
   If $Y \subseteq X^+$ and $A \not\subset X^+$ then
       $X^+ := X^+ \cup A$
   End if
 End while
```

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 $A \in (X\text{-}B)^+$. If $B \in (X\text{-}B)^+$ 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 nonimplied extraneous attribute is postponed until the time that redundant dependencies are eliminated, using the *Algorithm 3*. 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 3* removes any implied extraneous attributes.

Algorithm 2: Removes implied extraneous attributes.

```
Input: F, a set of FDs.
Output: F', a partially left-
     reduced set of FDs.
 Let F':=F
For each fd $X \rightarrow A \in F'$ do
    Let Y:=X
     For each attribute $B \in Y$
      If $A \in (Y\text{-}B)^+$ w.r.t F' then
         Y:=Y-B
      End if
     End for
  $F' := F' - (X \rightarrow A) \cup (Y \rightarrow A)$
End for
```

Remove all duplicated dependencies, if any.

Example 1:

Suppose we want to normalize the *1NF* relation *ClientRental* up to *3NF*. This example is taken from [12] with minor modifications.
This relation contains 9 attributes and 18 dependencies:

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

with the following dependencies F:

f_1:	*propertyNo,rentStart, ownerNo*	$\rightarrow$ *rentFinish*
f_2:	*clNo,propertyNo, clName*	$\rightarrow$ *rentFinish*
f_3:	*clNo,rentStart, clName*	$\rightarrow$ *rentFinish*
f_4:	*clNo*	$\rightarrow$ *clName*
f_5:	*propertyNo,rentStart*	$\rightarrow$ *clName*
f_6:	*clNo,propertyNo*	$\rightarrow$ *rentStart*
f_7:	*ownerNo*	$\rightarrow$ *oName*
f_8:	*propertyNo*	$\rightarrow$ *oName*
f_9:	*clNo,rentStart*	$\rightarrow$ *oName*
f_{10}:	*clNo,rentStart*	$\rightarrow$ *pAddress*
f_{11}:	*propertyNo*	$\rightarrow$ *pAddress*
f_{12}:	*propertyNo*	$\rightarrow$ *rent*
f_{13}:	*clNo,rentStart*	$\rightarrow$ *rent*
f_{14}:	*propertyNo*	$\rightarrow$ *ownerNo*
f_{15}:	*clNo,rentStart*	$\rightarrow$ *ownerNo*
f_{16}:	*clNo,rentStart*	$\rightarrow$ *propertyNo*
f_{17}:	*propertyNo,rentStart*	$\rightarrow$ *clNo*
f_{18}	*propertyNo, clNo*	$\rightarrow$ *ownerNo*

Initialization: F':=F

- For the fd f_1, we have $rentFinish \in (propertyNo,rentStart)^+$; then F' is changed by removing the attribute *ownerNo* at the rhs of f_1.

- For the fd f_2 and f_3, we have $rentFinish \in (clNo,propertyNo)^+$ and $rentFinish \in (clNo,rentStart)^+$ respectively; then F' is changed by removing the attribute *clName* at the rhs of f_2 and f_3.

The final partially left-reduced dependencies F' of F is then:

f_1:	*propertyNo,rentStart*	$\rightarrow$ *rentFinish*
f_2:	*clNo,propertyNo*	$\rightarrow$ *rentFinish*
f_3:	*clNo,rentStart*	$\rightarrow$ *rentFinish*
f_4:	*clNo*	$\rightarrow$ *clName*
f_5:	*propertyNo,rentStart*	$\rightarrow$ *clName*
f_6:	*clNo,propertyNo*	$\rightarrow$ *rentStart*
f_7:	*ownerNo*	$\rightarrow$ *oName*
f_8:	*propertyNo*	$\rightarrow$ *oName*
f_9:	*clNo,rentStart*	$\rightarrow$ *oName*
f_{10}:	*clNo,rentStart*	$\rightarrow$ *pAddress*
f_{11}:	*propertyNo*	$\rightarrow$ *pAddress*
f_{12}:	*propertyNo*	$\rightarrow$ *rent*
f_{13}:	*clNo,rentStart*	$\rightarrow$ *rent*
f_{14}:	*propertyNo*	$\rightarrow$ *ownerNo*
f_{15}:	*clNo,rentStart*	$\rightarrow$ *ownerNo*
f_{16}:	*clNo,rentStart*	$\rightarrow$ *propertyNo*
f_{17}:	*propertyNo,rentStart*	$\rightarrow$ *clNo*
f_{18}:	*propertyNo, clNo*	$\rightarrow$ *ownerNo*

As one can see there is no more extraneous implied attributes in F'. However, in f_{18} the attribute *clNo* is a *nonimplied* extraneous one because of the dependency f_{14} and *propertyNo* did not functionally determines *clNo*. *Nonimplied* extraneous attribute 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.

Definition. A dependency f_1: $Y \rightarrow A$ is redundant in a set of dependencies F if and only if there is a dependency f_2: $X \rightarrow A$ in F such that $Y \rightarrow X \in F^+$, i.e. $X \subseteq Y^+$. Algorithmically this can be expressed as follows:

Algorithm 3: Removes redundant dependencies.

```
Input: F, a set of partially
    left-reduced dependencies.
Output: $F_m$, a minimal cover of F
1. $F_m$:= F
2. For each $X \rightarrow A \in F_m$ do
    while there exists a $Y \rightarrow A \in F$
        G:= $F_m$-($Y \rightarrow A$)
        If $X \subseteq Y_G^+$ then
            $F_m$:= G
        End if
    End while
    End for
3. $F_m$ is a minimal cover of F.
```

To continue with our previous example, 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. $lhs(f_i)$ functionally determines $lhs(f_j)$.

- 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 $lhs(f_{14}) \subseteq lhs(f_{15})^+$
- f_{18} is redundant because $lhs(f_{14}) \subseteq lhs(f_{18})^+$

After removing the redundant dependencies (10), we get a minimal cover F_m :

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

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 3*.

Theorem 1. Let F be a set of partially left-reduced dependencies, and F_m its *minimal cover* obtained by the Algorithm 3. Then we have $F^+ \equiv F_m^+$.

4 FULL and PARTIAL DEPENDENCIES

A functional dependency $X \rightarrow A$ 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 $B \in X$, (X - B) does not functionally determine A. A functional dependency $X \rightarrow A$ is a *partial dependency* if A is not a key-attribute and some attribute $B \in X$ can be removed from X and the dependency still holds; that is, for some $B \in X$, $(X - B) \rightarrow A$ [7]. Full and partial dependencies are generated as follows:

Algorithm 4: Determines full and partial dependencies.

```
Input: $F_m$, a minimal cover set of F.
Output: $F_p$, $F_f$, sets of partial and full dependencies in $F_m$ respectively.
Let $F_p := \phi$ and $F_f := F_m$.
For each dependency $X \rightarrow A \in F_f$
    If X is a proper subset of a candidate key and A is not a key attribute then
        $F_p := F_p \cup \{X \rightarrow A\}$
        $F_f := F_f - (X \rightarrow A)$
        While there is a fd $Z \rightarrow B \in F_f$ s.t. $Z \in A^+$ do
            $F_p := F_p \cup \{Z \rightarrow B\}$
            $F_f := F_f - (Z \rightarrow B)$
        End while
    End if
End for
```

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 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 full and partial dependencies.

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).

The set of full dependencies is:

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

Lemma 1. $F_m = F_f \cup F_P$.

Lemma 2. Let R be a relation schema and F_m its minimal cover dependency set. If there is no partial dependency in R then $F_p = \phi$.

5 NORMALIZATION

5.1 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 $X \rightarrow Y$ where X is a strict subset of a candidate key and Y is a non-key attribute.
The following algorithm is used to decompose a 1NF relation schema R into 2NF:
Algorithm 5: Decomposes into *2NF*.

```
Input: R, $F_f$ and $F_p$.
Output: relations into 2NF.
  Let G:=$F_p$ and $X_f$ the set of
  attributes in $F_f$.
 For each $Y \rightarrow A \in G$ do
   If Y is a key attribute then
     - create $R_Y$, and add any
       attribute in $Y_G+$.
     - remove from G any
       dependency whose lhs is in $Y_G^+$
     - choose Y as the
       primary key of $R_Y$.
     - if $A \in X_f$, $X_f := X_f - \{A\}$
   End if
 End for
Let $K \in X_f$ be the (chosen)
primary key of R. Create a new
relation schema $R_K(X_f)$.
```

If Y is a key-attribute that means that Y is causing a partial dependency.

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 5. 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={ *clNo,propertyNo* → *rentStart;*
clNo,propertyNo → *rentFinish;*
clNo,rentStart → *propertyNo;*
propertyNo,rentStart → *clNo* }

F_p={ *clNo* → *clName;*
propertyNo → *pAddress;*
propertyNo → *rent;*
propertyNo → *ownerNo;*
ownerNo → *oName* }

and $F_m=F_f \cup F_p$.

As we can see all attributes in F_f depend fully on the candidate keys (clNo, propertyNo), (clNo, rentStart) and (propertyNo, rentStart) and all the attributes in F_p depend directly/indirectly on the key-attributes clNo or propertyNo, i.e., the two key-attributes are causing partial dependencies.

- For *clNo* → *clName*

clNo is a key-attribute, a new relation R_{clNo}(<u>clNo</u>, clName) is then created

- For *propertyNo* → *pAddress*

propertyNo is a key-attribute, a new relation
$R_{propertyNo}$(<u>propertyNo</u>, pAddress, rent,ownerNo, oName)

is then created with all attributes in *propertyNo*$^+$.

- If we choose arbitrarily (clNo, propertyNo) as the primary key, but no special consideration will be given to this key over the other candidate keys, then a new relation is needed:

$R_{(clNo,propertyNo)}$(<u>clNo, propertyNo</u>, rentStar, rentFinish)

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.

5.2 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={ $X\rightarrow A \in F_m$/ X is not a subset of any candidate key and A is not a key-attribute }

Note that partial dependencies, even if violate the 3NF, are not considered in F_t, because the input relations are supposed already in 2NF using the previous algorithm. The following procedure is used to decompose 2NF relations into 3NF.

Algorithm 6: Decomposes into 3NF.

```
Input: R_1,...,R_n relations in 2NF, and the
       set F_t.
Output: a set of relations into 3NF.
Let G:=F_t
For each dependency X→A∈G do
    - create a new relation R_X, if
      not already created,
    - add X and A in R_X, if not
      already added,
    - consider X as the primary key
      of R_X,
    - remove X→A from G, and
    - remove the attribute A from
      the relations R_1,...,R_n.
Fnd for
```

Now, we can complete our example given the relations:

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

F_m={

f₂:	*clNo,propertyNo*	*→rentFinish*
f₄:	*clNo*	*→clName*
f₆:	*clNo,propertyNo*	*→rentStart*
f₇:	*ownerNo*	*→oName*
f₁₁:	*propertyNo*	*→pAddress*
f₁₂:	*propertyNo*	*→rent*
f₁₄:	*propertyNo*	*→ownerNo*
f₁₆:	*clNo,rentStart*	*→propertyNo*
f₁₇:	*propertyNo,rentStart*	*→clNo*

}

We can say that the set transitive dependencies is
F_t={ *ownerNo*$\rightarrow$*oName* } as *ownerNo* is not a subset of a candidate key and *oName* is not a key-attribute.

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:

$R_{(clNo,propertyNo)}$(<u>clNo,propertyNo</u>, rentStar, rentFinish)
R_{clNo}(<u>clNo</u>,clName)
$R_{propertyNo}$(<u>propertyNo</u>, pAddress, rent, ownerNo)
$R_{ownerNo}$(<u>ownerNo</u>, 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.

8 REFERENCES

1. Thomas, C., Carolyn, B.: Database Systems, A Practical Approach to Design, Implementation, and Management, Pearson Fourth edition (2005).
2. Bahmani A., Naghibzadeh, M. and Bahmani, B.: Automatic database normalization and primary key generation, Niagara Falls Canada IEEE (2008).
3. Beynon-Davies, P.: Database systems, Palgrave Macmillan, Third edition, ISBN 1–4039--1601–2 (2004).
4. Dongare,Y. V., Dhabe,P. S. and Deshmukh, S. V.: RDBNorma: A semi-automated tool for relational database schema normalization up to third normal form, Interrnational Journal of Database Management Systems, Vol.3, No.1 (2011).
5. Vangipuram, R., Velputa, R., Sravya, V.: A Web Based Relational database design Tool to Perform Normalization, International Journal of Wisdom Based Computing, Vol.1(3) (2011).
6. Codd, E.F.: Further normalization of the data base relational model. In Data Base Systems, Courant Inst. Comptr. Sci. Symp. 6, R. Rustin, Ed., Prentice-Hall, Englewood Cliffs, pp. 33--64 (1972).
7. Elmasri, R., Navathe, S.B.,: Fundamentals of Database Systems, Addison Wesley, fourth Edition (2003).
8. Date, C.J.: Introduction to Database Systems (8th ed.). Boston: Addison-Wesley. ISBN 978-0-321-19784-9 (2004).
9. Ullman, J.D.: Principe of Database Systems. Computer Science Press, Rockville, Md. (1982).
10. Maier, D.: The Theory of relational Databases. Computer Science Press, Rockville, Md. (1983).
11. Bernstein, P.A.: Synthesizing third normal form relations from functional dependencies. ACM Transactions on database Systems, vol.1, No.4, pp.277--298 (1976).
12. Diederich, J., Milton, J.: New Methods and Fast Algorithms for Database Normalization, ACM Transactions on database Systems, Vol.13, No.3, pp. 339--365 (1988).