Available Online at www.ijcsmc.com

International Journal of Computer Science and Mobile Computing

A Monthly Journal of Computer Science and Information Technology

ISSN 2320–088X

IJCSMC, Vol. 4, Issue. 10, October 2015, pg.286 – 294

RESEARCH ARTICLE

XML Database Schema Refinement through Functional Dependencies and Normalization

Zina Z. Al Shamaa, Prof. Musbah M. Aqel
G.D. of Educational Planning, Ministry of Education, Iraq
Middle East University, Amman
zina.shamaa@gmail.com, aqelm06@yahoo.com

Abstract-- EXtensible Markup Language (XML) has become the standard means for storing and exchanging data over the Web. It may contain redundant information due to the bad design of XML schemas which may lead to waste of storage and update anomalies. Our goal is to find a method for designing a good XML schema without redundancy. One approach to remove data redundancies in XML documents is based on normalization theory. This approach proposed XML Normal Forms (XNFs) to determine whether an XML schema is properly designed or not. Then the XML schema is redesigned or refined to satisfy some XNFs based on the supposed known XML Functional Dependencies (XFDs). The research about XFD and XML document normalization on the basis of XFD is still an open problem. In this paper, we improve the definition of XFD according to the hierarchical structure of XSD. We defined an XNF that generalizes BCNF. Finally, we develop a set of normalization rules for converting any XML schema into one in Normal Form.

Keywords: - XML FD, XML Normal Form, XML Normalization, XML schema, XML Path

1-**Introduction**

With the widespread use of the Web application and the accessibility of a huge amount of electronic data, XML has been used as the standard data model for storing and exchanging data over the Web. Similar to traditional databases, the XML document may contain redundant data due to the bad design of schema. The redundant stored data can lead to operation anomalies, increase the unnecessary storage space, inflates cost of storage and overmuch cost for transferring and manipulating data. Furthermore, once massive Web databases are created, it is challenging and hard to change their organization; hence, there is a risk of having huge amounts of widely accessible, but poorly organized data. Therefore, it is important to study such problems in XML research field.

XML documents have a nested structure. This gives a lot of flexibility when storing information. To specify the structure of a class of XML documents, we have to choose an appropriate schema. Schema languages for XML have been heavily researched with the DTD (Document Type Definition) [14] and XSD (XML Schema Definition) [15] being the most popular currently.

One strategy to avoid data redundancies in relational databases is to design schema without redundancy by applying Normalization theory approach [7]. The similar approach has been applied to XML schema design [2], [11]. However, it cannot be applied directly in XML documents due to the different in nature between the two models; relational model are flat and structured while XML documents are nested and have irregular hierarchical structure that makes XML items appear at different levels of XML tree ([9] ,[18])

Related Work:- One of the XML schema design approaches is XML Functional Dependency and Normalization theory. Provost, [12],[13] has been proposed to apply the theory of relational database on XML database schema design, many other researchers have worked on the same manner, but it is not a trivial task and cannot be applied directly on XML due to the significant differences in their structures [4]. Since there is still no standard way in defining XML functional dependencies, a lot of attempts have been made on defining XFD ([1]; [8]; [16]; [19]; [23]), the previous definitions of XFD differ in how to choose nodes of sub trees or how to specify equality between nodes. However, they are not powerful enough to specify constrains for every structure of XML. Depending on the definition of XFDs, and keys [3] several XML normal forms were proposed by ([2]; [9]; [11]; [17]; [20]; [21]; [23]) which are studied in the context of XML. They differ in Terms of schema and how to describe constraint, but are dependent on the same set of transformations.
Whatever, the above research has not solved the problems of XML functional dependency and XML Normalization perfectly, and obviously, the task of designing XML schema is becoming more complex than designing relational database due to the irregular hierarchical structure of XML schema.
The main contribution of this paper is as follows: We use XML Schema Definition (XSD) and introduce the formal definition of this schema, we improve the definition of XML functional dependency by using the XPath language and takes into consideration the hierarchical structure of XSD schema, then we introduce a set of dependencies depending on our improved functional dependencies definition, finally we define an XML normal form that generalizes BCNF.
Paper Outline: The structure of the paper is organized as follows. In section 2 we introduce a motivating example to demonstrate the anomalies that may occur in XML documents. In section 3 basic knowledge definitions are given. In section 4 we introduce our improvement definition of XFD and XML Normal Form then in section 5 we introduce Normalization Rules. Finally in section 6 the conclusion and future work are given.

2-Motivating Example:

To better understand the problem; let us consider the XML document shown in fig 1, which is part of a database for storing information about school management activity. This database describes departments, courses, students, offices and their relationship. Fig 2 shows a graphical representation for XSD schema. A database instance that conforms to this XSD is clarified in XML Document Tree in fig 3.
The XML document shown in fig 1 has the following information:
1) In the total document, the student number determines student name and age.
2) All in the school, course no determine course name.
3) In some departments, the room number determines the office name which valid in the entire document.
4) The department’s name determines the address of the office (supposed that every department locates in a certain building.
This constraint causes the document to store redundant information that can lead to update anomalies. For example in the first constraint if the same student takes many courses in the department, then his name and age repeated for each course.

```
<xs:element name="school">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="department" minOccurs="0" maxOccurs="unbounded" >
          <xs:complexType>
            <xs:sequence>
              <xs:element name="Dname" type="xs:string" />
              <xs:element name="course" minOccurs="0" maxOccurs="unbounded">
                <xs:complexType>
                  <xs:sequence>
                    <xs:element name="cno" type="xs:string" />
                    <xs:element name="cname" type="xs:string" />
                    <xs:element name="student" minOccurs="0" maxOccurs="unbounded">
                      <xs:complexType>
                        <xs:sequence>
                          <xs:element name="sno" type="xs:string" />
                          <xs:element name="sname" type="xs:string" />
                          <xs:element name="age" type="xs:string" />
                          <xs:element name="grade" type="xs:string" />
                        </xs:sequence>
                      </xs:complexType>
                    </xs:element>
                  </xs:sequence>
                </xs:complexType>
              </xs:element>
              <xs:element name="office" minOccurs="0" maxOccurs="unbounded">
                <xs:complexType>
                  <xs:sequence>
                    <xs:element name="room-no" type="xs:string" />
                    <xs:element name="Oname" type="xs:string" />
                    <xs:element name="address" type="xs:string" />
                  </xs:sequence>
                </xs:complexType>
              </xs:element>
            </xs:sequence>
          </xs:complexType>
          <xs:key name="departmentKey1">
            <xs:selector xpath="." />
            <xs:field xpath="mstns:Dname" />
          </xs:key>
          <xs:key name="departmentKey2">
            <xs:selector xpath=".//mstns:course" />
            <xs:field xpath="mstns:cno" />
          </xs:key>
          <xs:key name="departmentKey3">
            <xs:selector xpath=".//mstns:student" />
            <xs:field xpath="mstns:sno" />
          </xs:key>
          <xs:key name="departmentKey4">
            <xs:selector xpath=".//mstns:office" />
            <xs:field xpath="mstns:room-no" />
          </xs:key>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
```

Fig 1. Example of an XSD

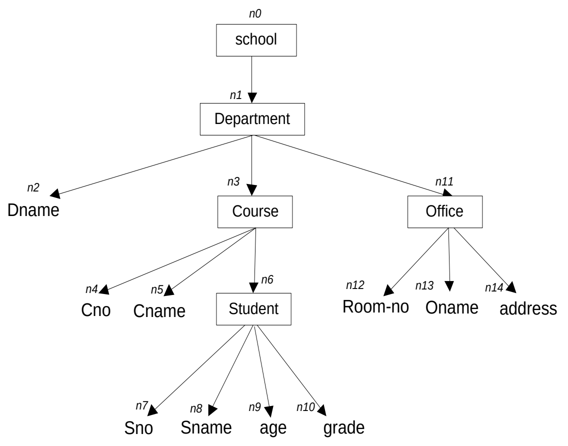

Fig 2. A graphical representation for XSD schema

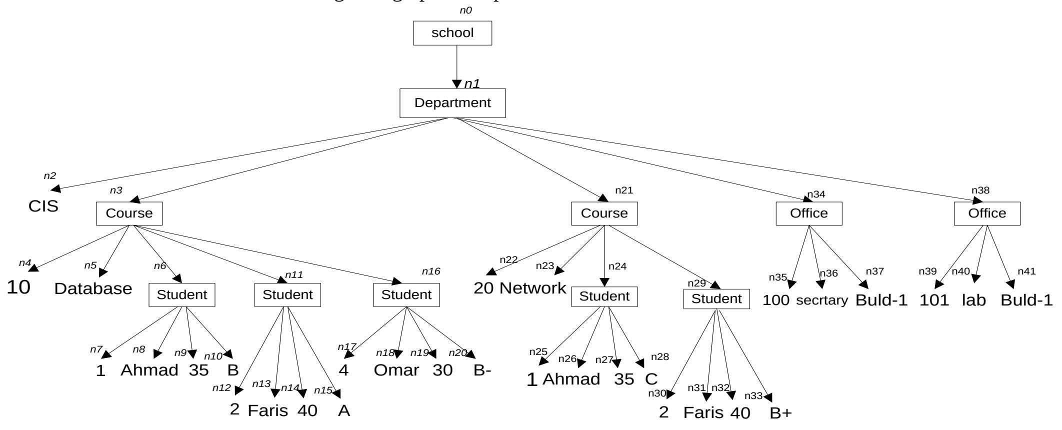

Fig 3. XML Document Tree

3-Basic Knowledge Definitions

In this section, we present the formal model for XSDs XML documents. Also, we introduce the notion of paths in XML documents and in XSDs.

3.1 XSDs and XML Documents

Assume that we have the following disjoint sets:

El is any set of complex element names and simple element names,

A is any set of attribute names (to refer to attribute, all attribute names start with the symbol @ to distinguish them from labels),

L is any set of labels,

S String values of attributes

N is any set of Nodes, represented as $\{n_0, n_1, ..., n_i\}$

Definition.1: XML Schema Definition (XSD)
The XML schema is a node labeled tree used to express the content model of an XML document structure. The XSD X is defined to be $X = (CE, SE, A, F, root, C)$, where: (1) CE is a set of complex elements which has another complex element, simple elements and attribute. The children of complex element can be described by three types of model groups (all, choice and sequence). For simplicity we consider on sequence model group whose defines the appearance of sub-elements items in specified order; (2) SE is a set of simple element nodes that have no sub-elements or attributes, they associated with a data types such as string, date, and decimal; (3) *A* is a set of attribute names that used to identify the properties of a complex element node; (4) *F* is a function from each CE to the children of the element set and attribute ($CE \to El \cup A \cup S$), and from each SE to its element type definitions; (5) *root* is the root element of the schema, it is of complex type and it is in level zero; (6) *C* is any set of identity constraint such as Keys, and key references.

Definition.2: XML Document as XML Tree
XML document is represented as XML tree T [4], which is defined to be $T = (N, root, label, comp, val)$, where:(1) *N* is a finite set of nodes in the tree that represent $CE \cup SE \cup A$; (2) *root* is the first complex node in tree; (3) *Label* is a function that assigns a label name to each node in tree, such that for each node $n \in N$, if $Label(n) \in CE$ then *n* is called complex element, if $Label(n) \in SE$ then *n* is called simple element, and if $lab(n) \in A$ then *n* is called attribute; (4) *comp* is a function from complex element node to a sequence node of $CE \cup SE \cup A$ such as when $n \in N$,if $u \in comp(n)$ then we call *u* a child of *n* and *n* the parent of *u*, so the parent-child relationships represent the structure of tree; (5) *val* is a function that assigns a value to each *SE* or *A*, the *CE* is null.

3.2 Path Expression
The path expression is an important concept in XSD used for navigating and specifying the sequence of elements in XML document. To express about the path logically, the XPath language is used, which is defined as a standardized path description language [6]. It is used to move in all directions in the document tree such as down to children and descendents, or upwards to parents and ancestors, or may be sideways to siblings [3]. The path that begins with a slash (/) is starting from the top of the document, while the path that start with double slash (//) means that the node start from anywhere in the document.

Definition.3: Path in XSD
Given the XSD $X = (CE, SE, A, F, root, C)$ then the path is defined as a sequence of elements $P(x) = e_1/...../e_k$ [16] where:

- e_1 = *root*, *Length (P)* = *k* and *last(P)*= e_k
- $e_i \in CE \cup SE \cup A \cup \{S\}$ where $1 \le i \le k$ and e_i is in the alphabet of $F(e_{i-1})$ for $2 \le i \le k-1$,
- e_k is in the alphabet of $F(e_{k-1})$, or e_k =*S*, or e_k =*A*

Definition.4: Path Instance in an XML Tree Document
A path instance in an XML tree $T = (N, root, lab, comp, val)$ is defined as a sequence of nodes [16], where:

$n_1 = root$,, *Length (P)* = *k* and *last(P)*= n_k

n_i is in the alphabet of $comp(n_{i-1})$

n_k is in the alphabet of $comp(n_{k-1})$, or $n_k = SE$, or $n_k = A$

3.3 Tuples for XML
In order to extend the concept of relational database to the XML data model, the XML data tree is considered as tree tuple [2]. The tree tuples are used to assign to each path in X schema a value of stored data. It is defined as a finite XML tree constructed with at most one occurrence of each path in a schema X. The following definition regarding tree tuples are adapted from [22].

Definition.5: Tuple for XML

Given a schema $X = (CE, SE, A, F, root, C)$, a tuple t in X is a function from paths(X) to $N \cup S \cup \{\perp\}$ such that:

- $t(r) \neq \perp$
- If $p \in Epaths(X)$ then $t(p) \in N \cup \{\perp\}$
- If $p \in paths(X) - Epaths(X)$, then $t(p) \in S \cup \{\perp\}$
- If $t(p1) = t(p2)$ and $t(p1), t(p2) \in N$ then *p1=p2*
- If $t(p1) = \perp$ and *p1* is a prefix of *p2*, then $t(p2) = \perp$
- $\{p \in paths(X) \mid t(p) \neq \perp\}$ is finite.

The set of all tuples in X is defined as *T(X)*

3.4 Equality

Definition.6: Node Equal and Value Equal

It is important to compare the nodes in the tree and detect value equality between them which is fundamental to the semantics of XFD and data redundancy. Two nodes n_1, n_2 are called value equal denoted by $n_1 =_v n_2$, if n_1, n_2 is a simple element or attribute then:

1) $label(n_1) = label(n_2)$
2) $val(n_1) = val(n_2)$

Otherwise, if n_1, n_2 is complex elements called node equal if

1) $label(n_1) = label(n_2)$, or
2) If the attribute $a \in$ *n1* there is an attribute $b \in$ *n2* such that $a = b$ and vise versa.
3) The sequence of their children elements is equal in pairs, which is mean $Comp(n_1) = comp(n_2)$ [19]; [23]

3.5 Key Constraints

Keys are an important part of any data model, as well as in XML database. It is considered as one of those integrity constraints which specify the way that the elements are associated to each other [1], and identifies the scope of uniqueness over XSD level [3]. The XSD supports the definition of key and foreign key concepts and has precise way for specifying them through the use of XPath language [5]. The formal definition of key in XSD is as follows:

Definition.7: XML Key (XK)

To define a key constraint, we specify a unique element that can determine uniquely other simple elements or attributes in the whole XML document [21]. Like in relational database key, it can be defined to include one or more fields which are called a composite key [12]. The formal definition of key in XSD is as follows:

Given an XSD $X = (CE, SE, A, F, root, C)$, the XK is a key of X schema that defined as $K(P_S, \{P_{f,1}, ..., P_{f,n}\})$ represented primary keys (which are to be unique and cannot be null) [10], where *n>0*, P_S is a selector path on XSD that specifies the complex element that hold fields with uniqueness constraint and $\{P_{f,1}, ..., P_{f,i}, ..., P_{f,n}\}$ are a set of field paths that represent the nodes to be checked for their value equality or uniqueness.

Definition.8: XML Foreign Key (XFK)

The foreign key in XSD is defined by the use of keyref function. It specifies association between nodes of XSD and asserts similar constraints on the value of referencing nodes [13].

The formal definition of the XFK is as follows:

$XFK = (XK, P_{SR}, \{P_{fR,1}, ..., P_{fR,m}\})$, [6] where:

XFK is defined as foreign key referred to be constrained to *XK* key, P_{SR} is the selector reference path that specifies the complex element that hold reference fields, and $P_{fR,1},........,P_{fR,m}$ is the set of fields reference paths that consider as a foreign key referred to the key in field of *XK*.

4. Extended Functional Dependencies

Just like in traditional databases, a functional Dependency is considered as one of the most popular data dependencies, it has played a centric role in providing richer data semantics [4].

4.1 The Definition of XFD

Definition.9: Given XSD $X = (CE, SE, A, F, root, C)$ then the functional dependency defined as:
$XFD = (P_h, [./x1, l_{1...i}, ..., ./xn, l_{1...i}] \rightarrow [./y1, l_{1...j}, ..., ./ym, l_{1...j}])$, where:

(1) P_h is the header path of XFD which defines the longest common repeatable path that is a prefix of both left hand side and right hand side and it is starting with the root node. The header path specifies the scope of XFD in which the constraint holds, and defines the node set in which the functional dependency holds, by considering the last element of header path such that $last(P_h) \in CE$.

If $P_h \neq \phi$ and $P_h \neq root$, then the scope of functional dependency is the sub-tree rooted $last(P_h)$ which is called a local functional dependency; otherwise, when $P_h = root$ then it is hold the scope overall the schema X which is called a global functional dependency.

(2) $./x1, l_{1...i}, ..., ./xn, l_{1...i}$ are the set of last elements of paths for the left hand side of the XFD which determine the right hand side, and represent $last(P_{x1})$,…. , $last(P_{xn})$ respectively.

$l_{1...i}$ is to specify the level of the last element of the paths, where bounded from 1 to i.

(3) $./y1, l_{1...j}, ..., ./ym, l_{1...j}$ are the set of last elements of paths for the right hand side of the XFD which functionally depend on the left hand side, and represent $last(P_{y1})$,…., $last(P_{ym})$ respectively.

$l_{1...j}$ is to specify the level of the last element of the paths, where bounded from 1 to j.

In general it means, for any two instance of tree tuples *t1, t2* identified by the XFD header P_h, if all LHS tree tuples agree on their values, then they must also agree on the value of the RHS tree tuples such as: $t_1.P_{xi} = t_2.P_{xi}$ imply $t_1.P_{yi} = t_2.P_{yi}$.

By applying the definition of XFD, the data constraints of XML document database in Fig 3 can be represented the functional dependency in the following forms:
XFD(1) $(school/department/course/student, [./Sno, l4 \rightarrow ./Sname, l4, ./age, l4])$
XFD(2) $(school/department/course, [./Cno, l3 \rightarrow ./Cname, l3])$
XFD(3) $(school/department/office, [./room\_no, l3 \rightarrow ./Oname, l3])$
XFD(4) $(school/department, [./dname, l2 \rightarrow .//office/address, l3])$

4.2 Types of Dependencies

The Functional Dependencies are classified by [23] into two types: absolute and relative. They proposed the functional dependency that has the element of both sides in the same level and in the bottom of schema as absolute functional dependency. While if the element of both sides in different level is relative dependency. In our research we classified relative dependency into two types: Relative Transitive XFD and Relative Full/Partial XFD.

4.2.1 Absolute XML Functional Dependency (AXFD)

Definition 10:AXFD

Let's consider the $XFD = (P_h, [./x1, l_{1..i}, ..., ./xn, l_{1..i}] \rightarrow [./y1, l_{1..i}, ..., ./ym, l_{1..i}])$

According to [23], it is an absolute functional dependency when
1) The last element of the header path is the element that represents the root of sub tree that contains elements of both sides of dependency.
2) The level of the last element of the paths in both sides of dependency is the same and refers to the terminal level in the schema
The XFD(1) $(school/department/course/student,[./Sno,l4 \rightarrow ./Sname,l4,./age,l4])$ for the XML document in Fig 3 which is represent an absolute functional dependency.
The header path ends with the element student, which is the sub tree root to the elements Sno, Sname, and age. This dependency specifies that each student must have student number as identifier key that determines the name of student and the age. While the same student takes many courses in the department, then his name and age repeated for each course. That means redundancy in the name and age.

4.2.2 Relative XML Functional Dependency (RXFD)
4.2.2.1 Relative Transitive XFD (RTXFD)
Definition 11: (RTXFD)
Let the paths Px ends with x element in level i, Py ends with y element in level i+1, and Pz ends with z element in level i+1,
Let's $XFD = (P_h,[./X,l_i \rightarrow ./Y,l_{i+1}])$ & $XFD = (P_h,[./Y,l_{i+1} \rightarrow ./Z,l_{i+1}])$ then $XFD = (P_h,[./X,l_i \rightarrow ./Z,l_{i+1}])$ is transitive XFD
This definition means the element (x) in level i transitively determines the element (z) in level i+1.
Consider the XML document in Fig 1 has the following functional dependencies:
$(school/department,[./dname,l2 \rightarrow ./office/room-no,l3])$,
$(school/department,[./office/room-no,l3 \rightarrow ./office/address,l3])$
These two functional dependencies represent the
XFD(4) $(school/department,[./dname,l2 \rightarrow .//office/address,l3])$,
which assert the transitive dependency. Where the key (dname) in level 2 under the complex element department is relatively determines the simple element (address) in level 3 under complex element office. It is obvious from the corresponding level that the element dname is not in the same level of the RHS element address.
The relative dependency can cause redundancy when violated. In Fig 3 the tree representation of XML document for school management database, when the department name (CIS) has many offices nodes, the room number (100) which is the room name (secrtary) is located in building (buld-1), and the same department (CIS) has office with room number (101) which is the room named (lab) located in building (buld-1). So in the department (CIS), the address of offices will repeat in each office node, which is mean redundancy in repeating the address.
4.2.2.2 Relative Full and Partial XFD (RFXFD and RPXFD)
Definition 12: (RFXFD)
Lets $./X,l_i,./Y,l_{i+1},./Z,l_{i+2},./S,l_{i+3}$ be last elements of paths Px, Py, Pz, and Ps in levels i, i+1, i+2,and i+3 respectively, and the elements x, y, z are keys for their parent complex elements. The following XFD is definition of relative full functional dependency (RFXFD) $XFD = (P_h,[./X,l_i,./Y,l_{i+1},./Z,l_{i+2} \rightarrow ./S,l_{i+3}])$, since the element S in the right hand side depend on all the key elements of the left hand side.
Definition 13: (RPXFD)
$XFD = (P_h,[./Y,l_{i+1},./Z,l_{i+2} \rightarrow ./S,l_{i+3}])$ is Relative Partial XFD.
In the first case of relative partial XFD we have one key element in the LHS, such case is considered as special case of relative transitive, hence we determine the XFD that has more than one key element part of composite key in left hand side and they are in deferent level.

4.3 Normal Form for XML
We now give the definition of XML Normal Form (XNF) based on defined dependencies.
Given XSD, and a set of XFD over the schema X.
The schema X is in normal form if and only if:
1) X has at least one key.
2) There is no non-trivial Absolute XFD.
3) There is no non-trivial Relative Transitive XFD.
4) There is no non-trivial Relative Partial XFD.

5) For any trivial XFD of the form satisfied by schema X, where X and Y are last elements of LHS paths and RHS paths respectively, then either X is a key or Y is part of the key in schema X.

5. XML Schema Normalization

The goal of normalization process of XML database schema is to convert an initial poorly designed schema into one of normal forms which eliminate redundancies and update anomalies [18]. Reference [2] was proposed an XNF decomposition algorithm that combines two basic ideas: creating a new element, and moving an attribute. These two ideas are the basic for the following elimination rules. In our research we integrated normalization rules, using reference [2] proposed algorithm, to perform the process of normalization of XML schema design.

5.1 Normalization Rules

Rule-1: Eliminate Absolute XFD (EAXFD)

This type of elimination is used when there is a redundancy caused by a non-trivial AXFD which holds between the elements under the same sub tree node.

Suppose X is a simple element key, and Y is a simple element, they are both under a complex element CE and in the same level.

To eliminate the redundancy caused by $AXFD = (P_h, [./X, li \rightarrow ./Y, li])$, we do the following:

1- Create a new complex element name (new-CE), under the root.
2- Replicate the simple element key in LHS, and simple element node in RHS.
3- Make them as children to node (new-CE).
4- Make the copy of the LHS as key under the (new-CE).
5- Delete the simple element of RHS from original location.

Rule-2: Eliminate Relative Transitive XFD (ERTXFD)

This type of elimination works with the non-trivial Relative Transitive XFD that has one element key in LHS.

Let's $XFD = (P_h, [./X, l_i \rightarrow ./Y, l_{i+n}])$ & $XFD = (P_h, [./Y, l_{i+n} \rightarrow ./Z, l_{i+n}])$ then $XFD = (P_h, [./X, l_i \rightarrow ./Z, l_{i+n}])$ is RTXFD

To eliminate redundancy caused by non-trivial RTXFD/H, we do the following:

1- Replicate the Right hand side of XFD which is in level i+n.
2- Put it in the same level of the LHS element which is i.
3- Delete it from original location.

Rule-3: Eliminate Relative Partial XFD (ERPXFD) This type of elimination works with the non-trivial Relative Partial XFD.

If $XFD = (P_h, [./X, l_i, ./Y, l_{i+1}, ./Z, l_{i+2} \rightarrow ./S, l_{i+2}])$ then $XFD = (P_h, [./Y, l_{i+1}, ./Z, l_{i+2} \rightarrow ./S, l_{i+2}])$ is Relative Partial XML functional dependency (RPXFD).

To eliminate redundancy caused by non-trivial RPXFD/H, we do the following:

1- Create new element-1 under the root.
2- Replicate the elements $./y, l_{i+1}$, $./Z, l_{i+2}$, and $./S, l_{i+2}$ with its sequence level of hierarchy.
3- Put the element $./y, l_{i+1}$ under the new element-1.
4- Create new element-2 under the new element-1.
5- Put the elements $./Z, l_{i+2}$ and $./S, l_{i+2}$ under new element-2.
6- Make the elements $./y, l_{i+1}$ and $./Z, l_{i+2}$ in the new location as keys.
7- Delete the RHS element $./S, l_{i+2}$ from original location.

6. Conclusions and Future Work

In this paper, we used XSD as a schema for XML database as it is improvement over DTD and has more capability than DTD. We improved the definition of XML functional dependency (XFD) according to the hierarchical structure of XSD schema through using XPath language and considering the level of the elements in order to detect the possible redundancy, and then according to our defined XFD we propose the XML Normal Form to determine the redundancy issue. Our proposed normal form are generalizes to BCNF in relational database, it is also preserves the hierarchical structure for both the XSD schema and XML document, and satisfies user requirement. Finally, we define set of normalization rules that eliminate redundancies.

The future works in this area worth to investigate of other types of redundancy such as those caused by multi-valued dependencies. Also, it is more interesting to evaluate the normal form on real XML dataset.

References

[1] K.Ahmad and H.Ibrahim**,** "Functional Dependencies and Inference Rules for XML,*"* IEEE *Information Technology, ITSim. International Symposium on*, 2008, pp.1-6.

[2] M. Arenas and L. Libkin, "A Normal Form for XML Documents," ***ACM, Transactions on Database Systems***, Vol. 29, No 1, PP 195-232, 2004.

[3] P. Buneman, S. Davidson, W. Fan, C. Hara, and W. Tan, "'Keys for XML," ***ACM*** ,WWW10, Hong Kong, 2001.

[4] H. Chen and H. Liao, "An Overview of Functional Dependencies in XML," ***International Conference on Education Technology (ICEIT),*** **IEEE International Conference** , 2010, PP 174-178,.

[5] J. Clark and S. DeRose, (1999) XPath homepage on W3 . [On-Line]. Available: http://www.w3.org/TR/xpath/

[6] T. Connolly and C. Begg , ***Database Systems A Practical Approach to Design, Implementation, and Management***, 5th ed., Pearson Education, Inc., PP. 365-405, 2010.

[7] C. Date, ***An Introduction to Database Systems***, 8th Ed., Pearson Education, Inc.,pp.333-402, 2004.

[8] M. Lee, T. Ling and W. Low, "Designing Functional Dependencies for XML," ***Springer, Advances in Database Technology, Lecture Notes in Computer Science*** **LNCS**, Vol. 2287, PP.124-141, 2002.

[9] T. Lv and P. Yan, "XML Normal Forms Based on Constraint-Tree-Based Functional Dependencies***," Springer, Lecture Notes in Computer Science LNCS***, Vol. 4537, PP.348-357, 2007.

[10] M. Necasky and J. Pokorny,"'Extending E-R for Modelling XML Keys," ***IEEE***, 2007, PP 236-241.

[11] T. Pankowski and T. Pilka, "Transformation of XML Data into Normal Form," ***Informatica 33***, PP.417-430, 2009.

[12] W. Provost (2002) The XML website. [On-Line]. Available :http://www.xml.com/pub/a/2002/11/13/normalizing.html/

[13] W. Provost (2002) The XML website. [On-Line]. Available :http://www.xml.com/pub/a/2002/12/04/normalizing.html/

[14] J. Shipman . (2009) "***Constructing a Document Type Definition (DTD) for XML***," dtd.pdf homepage on infohost.nmt.edu, [On-Line]. Available : http://infohost.nmt.edu/tcc/help/pubs/dtd/dtd.pdf

[15] H. Thompson , D. Beech , M. Maloney and N. Mendelsohn (2004), xmlschema homepage on ***W3.*** [On-Line]. Available:http://www.w3.org/TR/2004/REC-xmlschema-1-20041028/

[16] M. Vincent , J. Liu and C. Liu, "Strong Functional Dependencies and Their Application to Normal Forms in XML," ***ACM, Transactions on Database Systems***, Vol. 29, No. 3, PP.445–462, 2004.

[17] X. Wu , T. Ling , S.Lee , M. Lee and G. Dobbie, "NF-SS:A Normal Form for Semistructured Schema," ***Springer, LNCS 2465***, 2002, PP.292-305.

[18] Y. Wu, "Normalization Design of XML Database Schema for Eliminating Redundant Schema and Satisfying Lossless Join," ***Proceedings of the IEEE/ ACM International Conference on Web Intelligence***, (2004), PP 660-663.

[19] P. Yan and T. Lv, "Functional Dependencies in XML Documents," ***APWeb Workshops, LNCS 3842, Springer-Verlag Berlin Heidelberg***, (2006), PP 29-37.

[20] C. Yu and H. Jagadish, "XML schema refinement through redundancy detection and normalization," ***VLDB Journal 'The International Journal on Very Large Data Bases'***, Vol. 17, no. 2, Springer, pp 203-223, 2008.

[21] Z. Zainol and B.Wang , "XML Document Design via GN-DTD," ***European Journal of Scientific Research,*** Vol.44, No.2, pp.314-336. , 2010.

[22] Y. Zhang , "Multivalued Dependencies and a Normal Form for XML," Master thesis, University of Toronto, 2004.

[23] X. Zhao , J. Xin and E. Zhang, "XML functional Dependency and Schema Normalization," ***IEEE, Hybrid Intelligent Systems, Ninth International Conference*** , 2009, PP.307-312.