Journal of Computer Science 4 (9): 729-740, 2008
ISSN 1549-3636
© 2008 Science Publications

Extending the Concepts of Normalization from Relational Databases to Extensible-Markup-Language Databases Model

Hosam Farouk El-Sofany
Department of Computer Science and Engineering,
College of Engineering, Qatar University, Qatar

Abstract: In this study we have studied the problem of how to extend the concepts of Functional Dependency (FD) and normalization in relational databases to include the eXtensible Markup Language (XML) model. We shown that, like relational databases, XML documents may contain redundant information and this redundancy may cause update anomalies. Furthermore, such problems are caused by certain functional dependencies among paths in the document. Our goal is to find a way for converting an arbitrary XML Schema to a well-designed one, that avoids these problems. We introduced new definitions of FD and normal forms of XML Schema (X-1NF, X-2NF, X-3NF and X-BCNF). We shown that our normal forms are necessary and sufficient to ensure all conforming XML documents have no redundancies.

Key words: XML model, database design, functional dependencies, normal forms

INTRODUCTION

Although many XML documents are views of relational data, the number of applications using native XML documents is increasing rapidly. Such applications may use native XML storage facilities[2] and update XML data[3]. Updates, like in relational databases, may cause anomalies if data is redundant. In the relational world, anomalies are avoided by developing a well-designed database schema. XML has its version of schema too; such as DTD and XML Schema[1]. Our goal is to find the principles for good XML Schema design. We believe that it is important to do this research now, as a lot of data is being put on the web. Once massive web databases are created, it is very hard to change their organization; thus, there is a risk of having large amounts of widely accessible, but at the same time poorly organized legacy data.

Normalization is a process which eliminates redundancy, organizes data efficiently and improves data consistency. Whereas normalization in the relational world has been quite explored, it is a new research area in native XML databases. Even though native XML databases mainly work with document-centric XML documents and the structure of several XML document might differ from one to another, there is room for redundant information. This redundancy in data may impact on document updates, and efficiency of queries. Figure 1 show an overview of the XML normalization process that we propose.

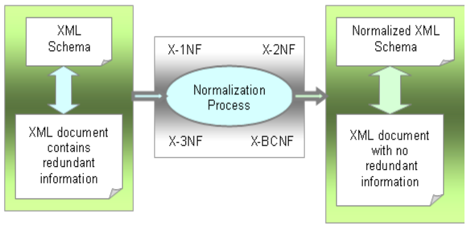

XML Schema
XML document contains redundant information
X-1NF
X-2NF
Normalization Process
X-3NF
X-BCNF
Normalized XML Schema
XML document with no redundant information

Fig. 1: An overview of the XML normalization process

This study is focus on dependency and normal form theory. This theory concerns with the well-designed databases and it connected with dependencies such as keys, functional dependencies (FDs), weak functional dependencies, multi-valued dependencies, inclusion dependencies, and join dependencies. All these classes of dependencies have been deeply investigated in the context of the relational data model[4,5]. The study now requires its generalization to XML (trees like) model.

Motivating example: Through an example, we show that, like relational databases, XML documents may contain redundant information and this redundancy may cause update anomalies.

Example 1: Consider the following XML Schema that describes a part of a "university" database. For every course, we store its number (cno), its title and the list of students taking the course. For each student taking a course, we store the student number (sno), name and the grade in the course.

```
<?xml version = “1.0” encoding = “IS0-8859-1”?>
<xs:schema    xmlns:xs    “http://www.w3.org/2001/
SMLSchema”>
<xs:element name = “courses”>
    <xs:complexType>
      <xs:sequence>
        <xs:element name = “course” type = “course”
        max0ccurs = “unbounded”/>
      </xs:sequence>
    </xs:complextType>
</xs:element>
<xs:element name = “course”>
    <xs:complextType>
      <xs:sequence>
        <xs: element name = “title” type = “xs:string”/>
        <xs:element   name   =   “taken_by”   type   =
“taken_by” max0ccurs = “unbounded”/>
    </xs:sequence>
      </xs:attribute name = “cno” type = “xs:string” use
      = “required”/>
  <xs:complexType>
</xs:element>
<xs:element name = “taken_by”>
    <xs:complesType>
      <xs:sequence>
        <xs:element name = “student” type = “student”
        max0ccurs = “unbounded:/>
      </xs:sequence>
    </xs:complexType>
<xs:element>
<xs:element name = “student”>
    <xs:complexType>
      <xs:sequence>
        <xs:element name = “name” type = “sx:string”/>
        <xs:element name = “grade” type = “sx:string”/>
    <xs:sequence>
        <xs:attribute name = “sno” type = “xs:string” use
= “required”/>
  </xs:complexType>
</xs:element>

</xs:schema>
```

An example of an XML document (tree) that conforms to this XML Schema is shown in Fig. 2[13]. This document satisfies the following constraint:
"any two student elements with the same sno value must have the same name"

This constraint (which looks very much like a FD), causes the document to store redundant information: for example, the name Deere for student st1 is stored twice, as in relational databases, such redundancies can lead to update anomalies: for example, updating the name of st1 for only one course results in an inconsistent document and removing the student from a course may result in removing that student from the document altogether.

In order to eliminate redundant information, we use a technique similar to the relational one and split the information about the name and the grade. Since we deal with just one XML document, we must do it by creating an extra element of complex Type, called info, for student information, as shown:

```
<?xml version = “1.0” encoding = “IS0-8859-1”?>
<xs:schema    xmlns:xs    “http://www.w3.org/2001/
SMLSchema”>
<xs:element name = “courses”>
    <xs:complexType>
      <xs:sequence>
        <xs:element name = “course” type = “course”
        max0ccurs = “unbounded”/>
        <xs:element  name  =  “info”  type  =  “info”
        max0ccurs = “unbounded”/>
      </xs:sequence>
    </xs:complextType>
</xs:element>
<xs:element name = “course”>
    <xs:complextType>
      <xs:sequence>
        <xs: element name = “title” type = “xs:string”/>
        <xs:element   name   =   “taken_by”   type   =
        “taken_by” max0ccurs = “unbounded”/>
    </xs:sequence>
      </xs:attribute name = “cno” type = “xs:string” use
= “required”/>
  <xs:complexType>
</xs:element>
<xs:element name = “taken_by”>
    <xs:complesType>
      <xs:sequence>
        <xs:element name = “student” type = “student”
        max0ccurs = “unbounded:/>
      </xs:sequence>
    </xs:complexType>
<xs:element>
<xs:element name = “student”>
    <xs:complexType>
      <xs:sequence>
        <xs:element name = “name” type = “sx:string”/>
        <xs:element name = “grade” type = “sx:string”/>
    <xs:sequence>
        <xs:attribute name = “sno” type = “xs:string” use
= “required”/>
  </xs:complexType>
</xs:element>
```

```
<xs:element name = “info”>
  <xs:complexType>
    <xs:sequence>
        <xs:element name = “number” type = “xs:string”
max0ccurs = “unbounded:/>
        <xs:element name = “name” type = “xs:string”>
    </xs:sequence>
        <xs:element name = “sno” type = “xs:string” use
= “required”/>
  </xs:complexType>
</xs:element>

</xs:schema>
```

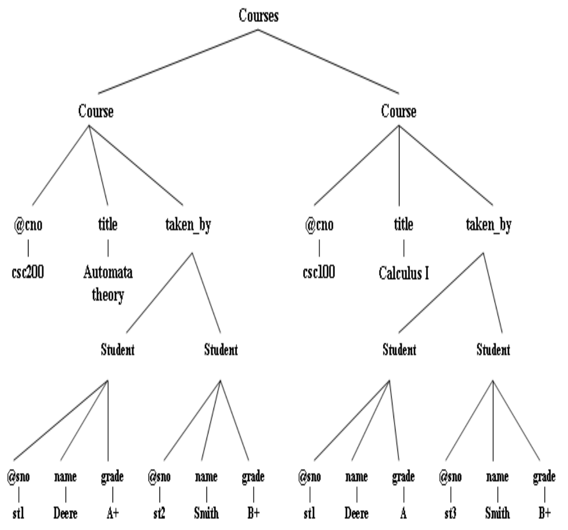


Fig. 2: A document containing redundant information

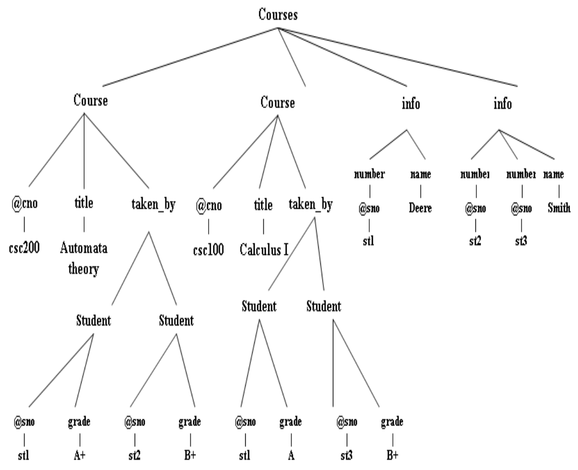


Fig. 3: A well-designed document

Each info element has (as children) one name and a sequence of number elements, with sno as an attribute. Different students can have the same name and we group all student numbers sno for each name under the same info element. A restructured document that conforms to this XML Schema is shown in Fig. 3, [13]. Note that st2 and st3 are put together because both students have the same name.

This example remembers us with the bad relational design caused by nonkey FDs and how the database designer solve this problem by modifying the schema.

MATERIALS AND METHODS

To extend the notions of FDs to the XML model, we represent XML trees as sets of tuples[13] and find the correspondence between documents and relations that leads to the definition of functional dependency.

We first describe the formal definitions of XML Schema (XSchema) and the conforming of XML tree to XSchema. The definition of XSchema is based on regular tree grammar theory that introduced in[22]. Assume that we have the following disjoint sets:

- $\hat{E}$: set of element names
- $\hat{A}$: set of attribute names
- DT: set of atomic data types (e.g., ID, IDREF IDREFS, string, integer and date, …)
- Str: set of possible values of string-valued attributes
- Vert: set of node identifiers

All attribute names start with the symbol @. The symbols φ and S represent element type declarations EMPTY (null) and #PCDATA, respectively.

Definition 1 (XSchema): An XSchema is denoted by 6-tuple: $X = (E, A, M, P, r, \sum)$, where:

- $E \subseteq \hat{E}$, is a finite set of element names.
- $A \subseteq \hat{A}$, is a finite set of attribute names.
- M is a function from E to its element type definitions: i.e., $M(e) = \alpha$, where $e \in E$ and α is a regular expression:
  $\alpha ::= \varepsilon \mid t \mid \alpha + \alpha \mid \alpha, \alpha \mid \alpha^* \mid \alpha^? \mid \alpha^+$

  where, ε denotes the empty element, $t \in DT$, "+" for the union, "," for the concatenation, α^* for the Kleene closure, $\alpha^?$ for $(\alpha + \varepsilon)$ and α^+ for (α, α^*)
- P is a function from an attribute name a to its attribute type definition: i.e., $P(a) = \beta$, where β is a 4-tuple (t, n, d, f), where:

$t \in DT$
n = Either "?" (nullable) or "¬?" (not nullable)
d = A finite set of valid domain values of a or ε if not known
f = A default value of a or ε if not known

- $r \subseteq E$ is a finite set of root elements
- ∑ is a finite set of integrity constraints for XML model. The integrity constraints we consider are keys (P.K, F.K,…) and dependencies (functional and inclusion)

Definition 2 (path in XSchema): Given an XSchema X = (E, A, M, P, r, ∑), a string $p = p_1 \ldots p_n$, is a path in X if, $p_1 = r$, p_i is in the alphabet of $M(p_{i-1})$, for each $i \in [2, n-1]$ and p_n is in the alphabet of $M(p_{n-1})$ or $p_n = @l$ for some $@l \in P(p_{n-1})$.

- We define length(p) as n and last(p) as p_n
- We let paths(X) stand for the set of all paths in X and EPaths(X) for the set of all paths that ends with an element type (rather than an attribute or S), that is: EPaths(X) = { p ∈ paths(X) | last(p) ∈ E }
- An XSchema is called recursive if paths(X) is infinite

Definition 3 (XML Tree): An XML tree T is defined to be a tree, T = (V, lab, ele, att, root)
Where:

- $V \subseteq Vert$ is a finite set of vertices (nodes)
- lab : $V \rightarrow \hat{E}$
- ele : $V \rightarrow Str \cup V^*$
- att is a partial function $V \times \hat{A} \rightarrow Str$. For each $v \in V$, the set $\{@l \in \hat{A} \mid att(v, @l) \text{ is defined}\}$ is required to be finite
- root ∈ V is called the root of T

The parent-child edge relation on V, $\{(v_1, v_2) \mid v_2$ occurs in $ele(v_1)\}$, is required to form a rooted tree. Note that, the children of an element node can be either zero or more element nodes or one string.

Definition 4 (path in XML tree): Given an XML tree T, a string: $p_1 \ldots p_n$ with $p_1, \ldots, p_{n-1} \in \hat{E}$ and $p_n \in \hat{E} \cup \hat{A} \cup \{S\}$ is a path in T if there are vertices $v_1 \ldots v_{n-1} \in V$ s.t.:

- v_1 = root, v_{i+1} is a child of v_i $(1 \le i \le n-2)$, $lab(v_i) = p_i$ $(1 \le i \le n-1)$
- If $p_n \in \hat{E}$, then there is a child v_n of v_{n-1} s.t. $lab(v_n) = p_n$. If $p_n = @l$, with $@l \in \hat{A}$, then $att(v_{n-1}, @l)$ is defined. If $p_n = S$, then v_{n-1} has a child in Str
- We let paths(T) stand for the set of paths in T

Now, we give a definition of a tree conforming to the XSchema ($T \models X$) and a tree compatible with X ($T \lhd X$).

Definition 5: Given an XSchema X = (E, A, M, P, r, ∑) and an XML tree T = (V, lab, ele, att, root), we say that T is valid w.r.t. X (or T conforms to X) written as ($T \models X$) if:

- lab: $V \rightarrow E$
- For each $v \in V$, if M(lab(v)) = S, then ele(v) = [s], where $s \in Str$. Otherwise, $ele(v) = [v_1, \ldots, v_n]$ and the string $lab(v_1) \ldots lab(v_n)$ must be in the regular language defined by M(lab(v))
- att is a partial function, att: $V \times A \rightarrow Str$, s.t. for any $v \in V$ and $@l \in A$, att(v, @l) is defined iff $@l \in P(lab(v))$
- lab(root) = r
- We say that T is compatible with X (written $T \rhd X$) iff $paths(T) \subseteq paths(X)$
- Clearly, $T \models X \in T \rhd X$

Definition 6: Given two XML trees $T_1 = (V_1, lab_1, ele_1, att_1, root_1)$ and $T_2 = (V_2, lab_2, ele_2, att_2, root_2)$, we say that T_1 is subsumed by T_2, written as $T_1 \le T_2$ if:

- $V_1 \subseteq V_2$
- $root_1 = root_2$
- $lab_2|_{V_1} = lab_1$
- $att_2|_{V_1 \times \hat{A}} = att_1$
- $\forall\ v \in V_1$, $ele_1(v)$ is a sub-list of a permutation of $ele_2(v)$

Definition 7: Given two XML trees T_1 and T_2, we say that T_1 is equivalent to T_2 written $T_1 \equiv T_2$, iff $T_1 \le T_2$ and $T_2 \le T_1$ (i.e., $T_1 \equiv T_2$ iff T_1 and T_2 are equal as unordered trees):

- We define [T] to be the ≡-equivalence class of T
- We write: $[T] \models X$ if $T_i \models X$ for some $T_i \in [T]$
- It is easy to see that for any $T_1 \equiv T_2$, $paths(T_1) = paths(T_2)$, hence
- $T_1 \rhd X$ iff $T_2 \rhd X$
- We shall also write $T_1 < T_2$ when $T_1 \le T_2$ and $T_2 \not\le T_1$

In the following definition, we extend the notion of tuple for relational databases to the XML model. In a relational database, a tuple is a function that assigns to each attribute a value from the corresponding domain. In our setting, a tree tuple t in a XML Schema X is a function that assigns to each path in X a value in Vert∪Str∪{φ} in such a way that t represents a finite tree with paths from X containing at most one occurrence of each path. We show that an XML tree can be represented as a set of tree tuples.

Definition 8 (tree tuples): Given XML Schema X = (E, A, M, P, r, ∑), a tree tuple t ∈ X is a function, t: paths(X) → Vert∪Str∪{φ} such that:

- For p ∈ EPaths(X), t(p) ∈ Vert∪{φ} and t(r) ≠ φ
- For p ∈ paths(X) − EPaths(X), t(p) ∈ Str ∪ {φ}
- If $t(p_1) = t(p_2)$ and $t(p_1)$ ∈ Vert, then $p_1 = p_2$
- If $t(p_1)$ = φ and p_1 is a prefix of p_2, then $t(p_2)$ = φ
- {p ∈ paths(X) | t(p) ≠ φ} is finite

T(X) is defined to be the set of all tree tuples in X. For a tree tuple t and a path p, we write t.p for t(p).

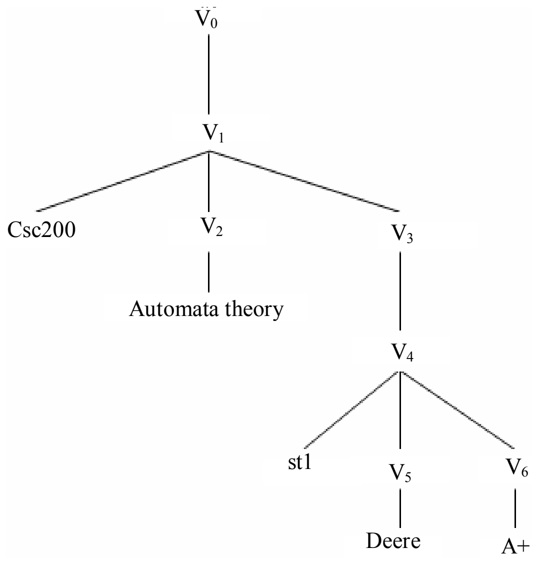
Example 2: Suppose that X is the XML Schema shown in example 1. Then a tree tuple in X assigns values to each path in paths(X) such as:

t(courses) = v_0
t(courses.course) = v_1
t(courses.course.@cno) = csc200
t(courses.course.title) = v_2
t(courses.course.title.S) = Automata Theory
t(courses.course.taken_by) = v_3
t(courses.course.taken_by.student) = v_4
t(courses.course.taken_by.student.@sno) = st1
t(courses.course.taken_by.student.name) = v_5
t(courses.course.taken_by.student.name.S) = Deere
t(courses.course.taken_by.student.grade) = v_6
t(courses.course.taken_by.student.grade.S) = A+

Definition 9 ($tree_X$): Given XML Schema X = (E, A, M, P, r, ∑) and a tree tuple t ∈ T(X), $tree_X(t)$ is defined to be an XML tree (V, lab, ele, att, root), where:

- root = t.r
- V = {v ∈ Vert | ∃ p ∈ paths(X) such that v = t.p}
- If v = t.p and v ∈ V, then lab(v) = last(p)
- If v = t.p and v ∈ V, then ele(v) is defined to be the list containing
- {t.p' | t.p' ≠ φ and p' = p.τ, τ ∈E, or p' = p.S, ordered lexicographically
- If v = t.p, @l ∈ A and t.p.@l ≠ φ , then att(v, @l) = t.p.@l

Example 3: Let X be the XML Schema from example 1 and t the tree tuple from Example 2. Then, t gives rise to the following XML tree:



Proposition 1: If t ∈ T (X), then $tree_X(t)$ ⊳X.

We would like to describe XML trees in terms of the tuples they contain. For this, we need to select tuples containing the maximal amount of information. This is done via the usual notion of ordering on tuples (relations).

- If we have two tree tuples t_1, t_2, we write $t_1 \sqsubseteq t_2$ if whenever t_1.p is defined, then t_2.p is also defined and t_1.p ≠ φ ∈ t_1.p = t_2.p
- As usual, $t_1 \sqsubset t_2$ means $t_1 \sqsubseteq t_2$ and $t_1 \neq t_2$
- Given two sets of tree tuples, Y and Z, we write: Y $\sqsubseteq^b$ Z, if: ∀ t_1 ∈ Y ⇒ t_2 ∈ Z s.t. $t_1 \sqsubseteq t_2$

Definition 10 ($tuples_X$): Given XML Schema X and an XML tree T such that T ⊳X, $tuples_X(T)$ is defined to be the set of maximal tree tuples t (with respect to ⊑), s.t. $tree_X(t)$ is subsumed by T, that is:
$\max_{\sqsubseteq}$\{ t ∈T (X) | $tree_X(t)$ ≤ T \}
Note that:

- $T_1 \equiv T_2$ implies $tuples_X(T_1) = tuples_X(T_2)$

- Hence, $tuples_X$ applies to equivalence classes: $tuples_X([T]) = tuples_X(T)$
- The following proposition lists some simple properties of $tuples_X(\cdot)$

Proposition 2: If $T \rhd X$, then $tuples_X(T)$ is a finite subset of $\mathcal{T}(X)$. Furthermore, $tuples_X(\cdot)$ is monotone: $T_1 \le T_2$ implies $tuples_X(T_1) \subseteq^b tuples_X(T_2)$.

Proof: We prove only monotonicity. Suppose that $T_1 \le T_2$ and $t_1 \in tuples_X(T_1)$. We have to prove that $\exists\ t_2 \in tuples_X(T_2)$ such that $t_1 \subseteq t_2$. If $t_1 \in tuples_X(T_2)$, this is obvious, so assume that $t_1 \notin tuples_X(T_2)$. Given that $t_1 \in tuples_X(T_1)$, $tree_X(t_1) \le T_1$ and therefore, $tree_X(t_1) \le T_2$. Hence, by definition of $tuples_X(\cdot)$, there exists $t_2 \in tuples_X(T_2)$ such that $t_1 \subset t_2$, since $t_1 \notin tuples_X(T_2)$.

Example 4: In example 1, we saw the XML Schema X and a tree T conforming to X. In example 2, we saw one tree tuple t for that tree, with identifiers assigned to some of the element nodes of T. If we assign identifiers to the rest of the nodes, we can compute the set $tuples_X(T)$:

{(v_0, v_1, csc200, v_2, Automata Theory, v_3, v_4, st1, v_5, Deere, v_6, A+)
(v_0, v_1, csc200, v_2, Automata Theory, v_3, v_7, st2, v_8, Smith, v_9, B-)
(v_0, v_{10}, mat100, v_{11}, Calculus I, v_{12}, v_{13}, st1, v_{14}, Deere, v_{15}, A)
(v_0, v_{10}, mat100, v_{11}, Calculus I, v_{12}, v_{16}, st3, v_{17}, Smith, v_{18}, B+)}

Finally, we define the trees represented by a set of tuples Y as the minimal, with respect to $\le$, trees containing all tuples in Y.

Definition 11 ($trees_X$): Given XML Schema X and a set of tree tuples $Y \subseteq \mathcal{T}(X)$, $trees_X(Y)$ is defined to be:

$\min_{\le}\{T \mid T \rhd X \text{ and } \forall\ t \in Y, tree_X(t) \le T\}$.

Notice that, if $T \in trees_X(Y)$ and $T' \equiv T$, then T' is in $trees_X(Y)$. The following shows that every XML document can be represented as a set of tree tuples, if we consider it as an unordered tree. That is, a tree T can be reconstructed from $tuples_X(T)$, up to equivalence $\equiv$.

Theorem 1: Given XML Schema X and an XML tree T, if $T \rhd X$, then $trees(tuples_X([T])) = [T]$.

Proof: Every XML tree is finite and, therefore, $tuples_X([T]) = \{t_1, \dots, t_n\}$, for some n. Suppose that $T \notin trees_X(\{t_1, . . . , t_n\})$. Given that $tree_X(t_i) \le T$, for each $i \in [1, n]$, there is an XML tree T' s.t. $T' \le T$ and $tree_X(t_i) \le T'$, for each $i \in [1, n]$. If $T' < T$, then there is at least one node, string or attribute value contained in T which is not contained in T'. This value must be contained in some tree tuple t_j ($j \in [1, n]$), which contradicts $tree_X(t_j) \le T'$. Therefore, $T \in trees_X(tuples_X([T]))$.

Let $T' \in trees_X(tuples_X([T]))$. For each $i \in [1, n]$, $tree_X(t_i) \le T'$. Thus, given that, $tuples_X(T) = \{t_1, \dots, t_n\}$, we conclude that $T \le T'$ and, therefore, by definition of $trees_X$, $T' \equiv T$.

Example 5: It could be the case that for some set of tree tuples Y there is no an XML tree T such that for every $t \in Y$, $tree(t) \le T$. For example, let X be XML Schema, X = (E, A, M, P, r, ∑), where E = {r, a, b}, $A = \varphi$, $M(r) = (a|b)$, $M(a) = \varepsilon$ and $M(b) = \varepsilon$. Let $t_1, t_2 \in \mathcal{T}(X)$ be defined as:

$t_1.r\ = v_0$	$t_2.r\ = v_2$
$t_1.r.a = v_1$	$t_2.r.a = \varphi$
$t_1.r.b = \varphi$	$t_2.r.b = v_3$

Since $t_1.r \ne t_2.r$, there is no an XML tree T such that, $tree_X(t_1) \le T$ and $tree_X(t_2) \le T$:

- We say that $Y \subseteq \mathcal{T}(X)$ is X-compatible if there is an XML tree T: $T \rhd X$ and $Y \subseteq tuples_X(T)$
- For X-compatible set of tree tuples Y, there is always an XML tree T: for every $t \in Y$, $tree_X(t) \le T$

Proposition 3: If $Y \subseteq \mathcal{T}(X)$ is X-compatible, then:

- There is an XML tree T such that $T \rhd X$ and $trees_X(Y) = [T]$
- $Y \subseteq^b tuples_X(trees_X(Y))$

Proof:

- Suppose that X = (E, A, M, P, r, ∑). Since Y is X-compatible, $\exists$ an XML tree T' = (V', lab', ele', att', root') s.t. $T' \rhd X$ and $Y \subseteq tuples_X(T')$. We use T' to define an XML tree T = (V, lab, ele, att, root) s.t. $trees_X(Y) = [T]$.
For each $v \in V'$, if there is $t \in Y$ and $p \in paths(X)$ s.t. $t.p = v$, then v is included in V. Furthermore, for each $v \in V$, lab(v) is defined as lab'(v), ele(v) =

$[s_1, \ldots, s_n]$, where each $s_i = t'.p.S$ or $s_i = t'.p.\tau$ for some $t' \in Y$ and $\tau \in E$ s.t., $t'.p = v$. For each $@l \in A$ s.t., $t'.p.@l \neq \varphi$ and $t'.p = v$ for some $t' \in Y$, att(v, @l) is defined as t'.p.@l. Finally, root is defined as root'. It is easy to see that $\text{trees}_X(Y) = [T]$

- Let $t \in Y$ and T be an XML tree s.t. $\text{trees}_X(Y) = [T]$. If $t \in \text{tuples}_X([T])$, then the property holds trivially. Suppose that $t \notin \text{tuples}_X([T])$. Then, given that $\text{tree}_X(t) \leq T$, there is $t' \in \text{tuples}_X([T])$ s.t. $t \subset t'$. In either case, we conclude that there is $t' \in \text{tuples}_X(\text{trees}_X(Y))$ s.t. $t \subseteq t'$.

The example below shows that it could be the case that $\text{tuples}_X(\text{trees}_X(Y))$ properly dominates Y, that is, $Y \subseteq^b \text{tuples}_X(\text{trees}_X(Y))$ and $\text{tuples}_X(\text{trees}_X(Y)) \not\subseteq^b Y$. In particular, this example shows that the inverse of Theorem 1 does not hold, that is, $\text{tuples}_X(\text{trees}_X(Y))$ is not necessarily equal to Y for every set of tree tuples Y , even if this set is X-compatible. Let X be as in example 5 and $t_1, t_2 \in T(X)$ be defined as:

$t_1.r = v_0$	$t_2.r = v_0$
$t_1.r.a = v_1$	$t_2.r.a = \varphi$
$t_1.r.b = \varphi$	$t_2.r.b = v_2$

Let t_3 be a tree tuple defined as:

$t_3.r = v_0$, $t_3.r.a = v_1$ and $t_3.r.b = v_2$

Then, $\text{tuples}_X(\text{trees}_X(\{t_1, t_2\})) = \{t_3\}$ since $t_1 \subset t_3$ and $t_2 \subset t_3$ and, therefore, $\{t_1, t_2\} \subseteq^{\mathbf{b}} \text{tuples}_X(\text{trees}_X(\{t_1, t_2\}))$ and $\text{tuples}_X(\text{trees}_X(\{t_1, t_2\})) \not\subseteq^b \{t_1, t_2\}$.

From Theorem 1 and Proposition 3, it is straightforward to prove the following Corollary.

Corollary: For a X-compatible set of tree tuples Y: $\text{trees}_X(\text{tuples}_X(\text{trees}_X(Y))) = \text{trees}_X(Y)$.

Functional dependencies of XML schema: We define the functional dependencies for XML Schema by using the tree tuples representation that discussed previously.

Definition 12 (functional dependencies): Given an XML Schema X, a functional dependency (FD) over X is an expression of the form: $S_1 \rightarrow S_2$ where $S_1, S_2 \subseteq \text{paths}(X)$, $S_1, S_2 \neq \varphi$. The set of all FDs over X is denoted by FD(X).

- For $S \subseteq \text{paths}(X)$ and $t, t' \in T(X)$, $t.S = t'.S$ means $t.p = t'.p \ \forall \ p \in S$. Furthermore, $t.S \neq \varphi$ means $t.p \neq \varphi \ \forall \ p \in S$

Definition 13: If $S_1 \rightarrow S_2 \in FD(X)$ and T is an XML tree s.t. $T \rhd X$ and $S_1 \cup S_2 \subseteq \text{paths}(T)$, we say that T satisfies $S_1 \rightarrow S_2$ (written $T \models S_1 \rightarrow S_2$), if $\forall \ t_1, t_2 \in \text{tuples}_X(T)$, $t_1.S_1 = t_2.S_1$ and $t_1.S_1 \neq \varphi \in t_1.S_2 = t_2.S_2$.

- Note that: if tree tuples t_1, t_2 satisfy an FD $S_1 \rightarrow S_2$, then for every path $p \in S_2$, $t_1.p$ and $t_2.p$ are either both null or both not null

Definition 14: If for every pair of tree tuples t_1, t_2 in an XML tree T, $t_1.S_1 = t_2.S_1$ implies they have a null value on some $p \in S_1$, then the FD is trivially satisfied by T.

- The previous definitions extends to the equivalence classes, since, for any FD f and $T \equiv T'$, $T \models f$ iff $T' \models f$
- We write $T \models F$, for $F \subseteq FD(X)$, if $T \models f$ for each $f \in F$ and we write $T \models (X, F)$, if $T \models X$ and $T \models F$

Example 6: Consider the XML Schema in example 1, we have the following FDs. Note that, cno is a key of course:

- courses.course.@cno → courses.course (FD1) Another FD says that two distinct student subelements of the same course cannot have the same sno:
- {courses.course,courses.course.taken_by.student.@sno} → courses.course.taken_by.student (FD2)

Finally, to say that two student elements with the same sno value must have the same name, we use:

- courses.course.taken_by.student.@sno →
- courses.course.taken_by.student.name.S (FD3)

Definition 15: Given XML Schema X, a set $F \subseteq FD(X)$ and $f \in FD(X)$, we say that (X, F) implies f, written $(X, F) \vdash f$, if for any tree T with $T \models X$ and $T \models F$, it is the case that $T \models f$. The set of all FDs implied by (X, F) will be denoted by $(X, F)^+$.

Definition 16: an FD f is trivial if $(X, \varphi) \vdash f$.

Primary and Foreign Keys of XML Schema: We present the definitions of the primary and foreign keys of the XML Schema. We'll use these definitions to introduce the normal forms of XML Schema. Also, we observe that while there are important differences between the XML and relational models, much of the thinking that commonly goes into relational database design can be applied to XML Schema design as well.

Definition 17 (key, foreign key and superkey): Let X = (E, A, M, P, r, ∑) be XML Schema, a constraint ∑ over X has one of the following forms:

- key: e(l) → e, where e∈E and l is a set of attributes in P(e). It indicates that the set l of attributes is a key of e elements
- foreign key: $e_1(l_1) \subseteq e_2(l_2)$ and $e_2(l_2) \rightarrow e_2$ where e_1, e_2 ∈ E and l_1, l_2 are non-empty sequences of attributes in $P(e_1)$, $P(e_2)$, respectively and moreover l_1 and l_2 have the same length. This constraint indicates that l_1 is a foreign key of e_1 elements referencing key l_2 of e_2 elements
- A constraint of the form $e_1(l_1) \subseteq e_2(l_2)$ is called an inclusion constraint
- Observe that a foreign key is actually a pair of constraint, namely an inclusion constraint $e_1(l_1) \subseteq e_2(l_2)$ and a key $e_2(l_2) \rightarrow e_2$
- superkey: suppose that, e ⊆ E and for any two distinct paths p_1 and p_2 in the XML Schema X, we have the constraint that: $p_1(e) \neq p_2(e)$. The subset e is called a superkey of X
- Every XML Schema has at least one default superkey - the set of all its elements

Normal forms of XML Schema: We will introduce the normal forms of XML documents. Our goal is to see what relational concepts we can usefully apply to XML. Can the normal forms that guide database design be applied meaningfully to XML document design?

First normal form for XML schema (X-1NF): First normal form (1NF) is now considered to be a part of the formal definition of a relation in the basic relational database model. Historically, it was defined as: "The domain of an attribute in a tuple must be a single value from the domain of that attribute"[20].
Of course, XML is hierarchical by nature. An XML "tuple" can vary from first normal form in several ways, all of them are valid by means of data modeling:

- It can have varying numbers of fields and default values for attributes
- It can have multiple values for a field, through the maxOccurs attribute for particles
- It can have choices of field types instead of a straight sequence or conjunction
- Fields can be of complex type
- The last feature is the most apparent when looking at an XML document. An XML tuple is a tree, not a table
- The second feature affects the relational database normal forms. It may at first seem as a good way to model multiple children (of simple or complex type) directly under a parent, without having to resort to multiple tables and foreign keys just to express a simple one-to-many relationship

Second normal form of XML schema (X-2NF): X-2NF is based on the concept of full functional dependency.

Definition 18: A FD $S_1 \rightarrow S_2$, where $S_1, S_2 \subseteq$ paths(X) is called full FD, if removal of any element's path p from S_1, means that the dependency does not hold any more, (i.e., for any p ∈ S_1, (S_1-{p}) does not functional determine S_2).

Definition 19: A FD $S_1 \rightarrow S_2$ is called partial dependency if, for some p ∈ S_1, (S_1-{p}) → S_2 is hold.

Example 7: Consider the following part of XML Schema called "Emp_Proj"

```
<xs:complexType name “Emp_Proj”>
    <xs:sequence>
      <xs: element name = “Sss” type = “string”/>
      <xs: element name = “Pnumber” type =
      “string”/>
      <xs: element name = “Hours” type = “string”/>
      <xs: element name = “Ename” type = “string”/>
      <xs: element name = “Pname” type = “string”/>
      <xs: element name = “Plocation” type =
      “string”/>
    <xs:sequence>
<xs: complexType>
<xs: key name = “emSssKey”>
    <xs: selector xpath = “Emp_Proj”/>
    <xs: field xpath = “Sss”/>
<xs: key>
<xs: key name = “ProfectNoKey”>
    <xs: selector cpath = “Emp_Proj”/>
    <xs: field xpath = “Pnumber”/>
</xs:key>
```

With the following FDs:

FD1: {Emp_Proj.Sss, Emp_Proj.Pnumber} → Emp_Proj.Hours
FD2: Emp_Proj.Sss → Emp_Proj.Ename

FD3: Emp_Proj.Pnumber → {Emp_Proj.Pname, Emp_Proj.Plocation}

Note that:

- FD1 is a full FD (neither Emp_Proj.Sss → Emp_Proj.Hours nor Emp_Proj.Pnumber → Emp_Proj.Hours holds).
- The FD: {Emp_Proj.Sss, Emp_Proj.Pnumber} → Emp_Proj.Ename is partial because Emp_Proj.Sss → Emp_Proj.Ename holds.

Definition 20 (X-2NF): An XML Schema X = (E, A, M, P, r, ∑) is in second normal form (X-2NF) if every elements e∈E and attributes l ⊆ P(e) are fully functionally dependent on the key elements of X.

- The test for X-2NF involves testing for FDs whose left-hand side are part of the primary key. If the primary key contain a single element's path, the test need not be applied at all

Example 8: The XML Schema Emp_Proj in the above example is in X-1NF but is not in X-2NF. Because the FDs FD2 and FD3 make Emp_Proj.Ename, Emp_Proj.Pname and Emp_Proj.Plocation partially dependent on the primary key {Emp_Proj.Sss, Emp_Proj.Pnumber} of Emp_Proj, thus violating the X-2NF test.

- Hence, the FDs FD1, FD2 and FD3 lead to the decomposition of XML Schema Emp_Proj to the following XML Schemas EP1, EP2 and EP3:

```
<xs: complexType name “EP1”>
      <xs:sequence>
        <xs: element name = “Sss” type = “string”/>
        <xs: element name = “Pnumber” type =
        “string”/>
        <xs: element name = “Hours” type = “string”/>
      </xs element>
      </xs:sequence>
<xs:cmplexType>
<xs:cmplexType name “EP2”>
      </xs:sequence>
        <xs: element name = “Sss” type = “string”/>
        <xs: element name = “Pname” type = “string”/>
      </xs:sequence>
<xs:cmplexType>
<xs:cmplexType name “EP3”>
      </xs:sequence>
        <xs: element name = “Pnumber” type =
        “string”/>
        <xs: element name = “Pname” type = “string”/>
        <xs: element name = “Plocation” type =
“string”/>
        </xs element>
      </xs:sequence>
<xs:cmplexType>
<xs: key name = “empSssKey”>
      <xs: selector xpath = “EP1”/>
      <xs: field xpath = “Sss”/>
</xs:key>
<xs: key name = “ProjectNoKey”>
      <xs: selector xpath = “EP1”/>
      <xs: field xpath = “Pnumber”/>
</xs:key>
<xs: key name = “emp2SssKey”>
      <xs: selector xpath = “EP2”/>
      <xs: field xpath = “Sss”/>
</xs:key>
<xs: key name = “Project3NoKey”>
      <xs: selector xpath = “EP3”/>
      <xs: field xpath = “Pnumber”/>
<xs:key>
```

Third Normal Form of XML Schema (X-3NF): X-3NF is based on the concept of transitive dependency.

Definition 21: A FD $S_1 \rightarrow S_2$, where $S_1, S_2 \subseteq$ paths(X) is transitive dependency if there is a set of paths Z (that is neither a key nor a subset of any key of X) and both $S_1 \rightarrow Z$ and $Z \rightarrow S_2$ hold.

Example 9: Consider the following XML Schema called "Emp_Dept":

Emp_Dept(Ssn, Ename, Bdate, Address, Dnumber, Dname, DmgrSsn)

```
<xs: complexType name “Emp_Dept”>
      <xs:sequence>
        <xs: element name = “Sss” type = “string”/>
        <xs: element name = “Ename” type = “string”/>
        <xs: element name = “Bdate” type = “string”/>
        <xs: element name = “Address” type = “string”/>
        <xs: element name = “Dnumber” type =
        “string”/>
        <xs: element name = “Dname” type = “string”/>
        <xs: element name = “DmgrSsn” type =
        “string”/>
      <xs:sequence>
<xs:complexType>
<xs: key name = “empSssKey”>
      <xs: selector xpath = “Emp_Dept”/>
      <xs: field xpath = “Sss”/>
```

```
</xs:key>
```

With the following FDs:

FD1: Emp_Dept.Ssn $\rightarrow$ {Emp_Dept.Ename, Emp_Dept.Bdate, Emp_Dept.Address, Emp_Dept.Dnumber }
FD2: Emp_Dept.Dnumber $\rightarrow$ {Emp_Dept.Dname, Emp_Dept.DmgrSsn}

Note that:
The dependency:
Emp_Dept.Ssn$\rightarrow$ Emp_Dept.DmgrSsn is transitive through Emp_Dept.Dnumber in Emp_Dept, because both the FDs:
Emp_Dept.Ssn $\rightarrow$ Emp_Dept.Dnumber and
Emp_Dept.Dnumber $\rightarrow$ Emp_Dept.DmgrSsn
hold and Emp_Dept.Dnumber is neither a key itself nor a subset of the key of Emp_Dept.

Definition 22 (X-3NF): An XML Schema X = (E, A, M, P, r, $\sum$) is in third normal form (X-3NF) if it satisfies X-2NF and no (elements $e \in E$ or $l \subseteq P(e)$) is transitively dependent on the key elements of X.

Example 10: The XML Schema Emp_Dept in the above example is in X-2NF (since no partial dependencies on a key element exist), but Emp_Dept is not in X-3NF. Because of the transitive dependency of Emp_Dept.DmgrSsn (and also Emp_Dept.Dname) on Emp_Dept.Ssn via Emp_Dept.Dnumber.

- We can normalize Emp_Dept by decomposing it into the following two XML Schemas ED1 and ED2:
  ED1(Ssn, Ename, Bdate, Address, Dnumber)
  ED2(Dnumber, Dname, DmgrSsn)

```
<xs:complexType name "ED1">
    <xs:sequence>
      <xs: element name = "Sss" type = "string"/>
      <xs: element name = "Ename" type = "string"/>
      <xs: element name = "Bdate" type = "string"/>
      <xs: element name = "Address" type = "string"/>
      <xs: element name = "Dnumber" type =
"string"/>
    <xs:sequence>
<xs:complexType>
<xs:complexType name "ED2">
      <xs: element name = "Dnumber" type =
      "string"/>
      <xs: element name = "Dname" type = "string"/>
      <xs: element name = "DmgrSsn" type =
      "string"/>
    <xs:sequence>
<xs:complexType>

<xs: key name = "empSssKey">
    <xs: selector xpath = "ED1"/>
    <xs: field xpath = "Sss"/>
</xs:key>

<xs: key name = "deptNoKey">
    <xs: selector xpath = "ED2"/>
    <xs: field xpath = "Dnumber"/>
</xs:key>
```

Boyce-codd normal form of XML schema (X-BCNF): Boyce-Codd Normal form of XML Schema (X-BCNF), proposed as a similar form as X-3NF, but it was found to stricter than X-3NF, because every XML Schema in X-BCNF is also in X-3NF, however, an XML Schema in X-3NF is not necessarily in X-BCNF. The formal definitions of BCNF differs slightly from the definition of X-3NF

Definition 23 (X-BCNF): An XML Schema X = (E, A, M, P, r, $\sum$) is in Boyce-Codd Normal Form (X-BCNF) if whenever a nontrivial FD $S_1 \rightarrow S_2$ holds in X, where $S_1, S_2 \subseteq$ paths(X), then S_1 is a superkey of X.

Also, we can consider the following definition of X-BCNF:

Definition 24: Given XML Schema X and $F \subseteq FD(X)$, (X, F) is in X-BCNF iff for every nontrivial FD $f \in (X, F)^+$ of the form $S \rightarrow p.@l$ or $S \rightarrow p.S$, it is the case that, $S \rightarrow p \in (X, F)^+$.

- The intuition is as follows: Suppose that $S \rightarrow p.@l \in (X, F)^+$. If T is an XML tree conforming to X and satisfying F, then in T for every set of values of the elements in S, we can find only one value of p.@l. Thus, for every set of values of S, we need to store the value of p.@l only once, in other words, $S \rightarrow p$ must be implied by (X, F)
- In definition 24, we suppose that, f is a nontrivial FD. Indeed, the trivial FD $p.@l \rightarrow p.@l$ is always in $(X, F)^+$, but often $p.@l \rightarrow p \notin (X, F)^+$, which does not necessarily represent a bad design

To show how X-BCNF distinguishes good XML design from bad design, we consider example 1 again, when only functional dependencies are provided.

Example 11: Consider the XML Schema from example 1 whose FDs are FD1, FD2 and FD3, shown in example 6. FD3 associates a unique name with each student number, which is therefore redundant. The design is not in X-BCNF, since it contains FD3 but does not imply the functional dependency:

courses.course.taken_by.student.@sno $\rightarrow$
courses.course.taken_by.student.name

To solve this problem, we gave a revised XML Schema in example 1. The idea was to create a new element info for storing information about students. That design satisfies FDs, FD1, FD2, as well as

courses.info.number.@sno $\rightarrow$ courses.info

and can be easily verified to be in X-BCNF.

RESULTS AND DISCUSSION

It was introduced in[6] an XML normal form called XNF, that defined in terms of functional dependencies, multi-valued dependencies and inclusion constraints. The normal form of [6] was defined in terms of two conditions: XML specifications must not contain redundant information with respect to a set of constraints and the number of schema trees must be minimal. Further, Embley, D. and Mok, W.Y. [6] presented a conceptual-model-based methodology that automatically generates XNF satisfied the DTDs and proved that the algorithms, which are part of the methodology, produce DTDs to ensure that the XML documents satisfy the properties of XNF.

It was proved in[7] that an XML specification given by a DTD, D and a set $\sum$ of XML functional dependencies is in XNF if and only if no XML tree conforming to D and satisfying $\sum$ contains redundant information. Thus, for the class of FDs defined in this article, the XML normal form introduced in[6] is more restrictive than our XML normal forms. The FD language used in[6] is based on a language for nested relations and it does not consider relative constraints.

In[8], a language for expressing FDs for XML was introduced. In that language, a FD is defined as an expression of the form:

$(p, [q_1, \ldots, q_n \rightarrow q_m])$

where, p is a fully qualified path expression (i.e., a path starting from the XML document root), every q_i ($i \in [1, n]$) is a LHS (Left-Hand-Side) entity type, a LHS entity type consists of an element name in the XML document and the optional key attribute(s); and q_m is a RHS (Right-Hand-Side) entity type, a LHS entity type consists of an element name in the XML document and an optional attribute name. An XML tree T satisfies this constraint if for any two subtrees T_1, T_2 of T whose roots are nodes reachable from the root of T by following path p, if T_1 and T_2 agree on the value of q_i, for every $i \in [1, n]$, then they agree on the value of q_m. This language does not consider relative constraints and its semantics only works properly if some syntactic restrictions are imposed on the FDs. Note that, the normalization problem is not considered in[8].

In[14] the author presented a formal model for relational trees focusing on constructors for lists and disjoint unions. These leads to new definition and derivation rules for FDs.

The recent article[13] took a first step towards the design and normalization theory for XML documents. The authors introduced the concept of a FD for XML, over a DTD, defined an XML normal form called XNF and then show that XNF is a generalized form of BCNF. Other proposals for XML constraints (mostly keys) have been studied in[9,11] these constraints do not use DTDs. XML constraints that takes DTDs into account are studied in[12].

The main contributions in this study were, the new definitions of FDs and normal forms of XML Schema. We have extended the definitions introduced by Marcelo Arenas and Leonid Libkin[13], that is based on XML DTD, to include the XML Schema instead. We shown how to use FDs to detect data redundancy in XML document and then proposed normal forms of XML Schema with respect to the FD constraints.

CONCLUSION

We have studied the problem of schema design and normalization in XML databases model. We introduced new definitions of FD and normal forms of XML Schema (X-1NF, X-2NF, X-3NF and X-BCNF) with respect to the FD constraints. We have illustrated that our normal forms are necessary and sufficient to ensure all conforming XML documents have no redundancies. In the future work, we plan to introduce the decomposition algorithm for converting any XML Schema into normalized one, that satisfies X-BCNF. Also we intent to work on extending XML Schema normal forms to more powerful normal forms, in particular by taking into account multi-valued dependencies, so we can express the other normal forms of XML Schema such as X-4NF and X-5NF.

ACKNOWLEDGMENT

The author acknowledge the financial support (Qatar National Research Fund- UREP 4-2-3) received from Qatar Foundation, via Qatar University, Qatar.

REFERENCES

1. W3C, 2001. XML Schema. http://www.w3.org/XML/Schema.
2. Kanne, C.C. and G. Moerkotte, 2000. Efficient storage of XML data. Proceedings of the 16th International Conference on Data Engineering, Feb. 28-Mar. 03, IEEE Computer Society, Washington, DC., USA., pp: 198-198. http://portal.acm.org/citation.cfm?id=847347.
3. Tatarinov, I., Z. Ives, A. Halevy and D. Weld, 2001. Updating XML. Proceedings of the ACM SIGMOD International Conference on Management of Data, May 21-24, ACM Press, New York, USA., pp: 413-424. http://portal.acm.org/citation.cfm?id=375663.375720.
4. Paredaens, J., P. DE Bra, M. Gyssens and D. Van Gucht, 1989. The Structure of the Relational Database Model. 1st Edn., Springer-Verlag, USA., ISBN: 10: 0387137149, pp: 231.
5. Thalheim, B., 1991. Dependencies in Relational Databases. Teubner-Verlag, ISBN: 3-8154-2020-2.
6. Embley, D. and W.Y. Mok, 2001. Developing XML documents with guaranteed "good" properties. Proceedings of the 20th International Conference on Conceptual Modeling, Nov. 27-30, Springer-Verlag, London, UK., pp: 426-441. http://portal.acm.org/citation.cfm?id=725895.
7. Arenas, M. and L. Libkin, 2003. An information-theoretic approach to normal forms for relational. Proceedings of the twenty-second ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems, June 09-11, ACM Press, New York, USA., pp: 15-26. http://portal.acm.org/citation.cfm?id=645340.650226.
8. Lee, L., T.W. Ling and W.L. Low, 2002. Designing functional dependencies for XML. Proceedings of the 8th International Conference on Extending Database Technology, Mar. 25-27, Springer-Verlag London, UK., pp: 124-141. http://portal.acm.org/citation.cfm?id=645340.650226.
9. Buneman, P., S. Davidson, W. Fan, C. Hara and W.C. Tan, 2001. Keys for XML. Proceedings of the 10th International World Wide Web Conference, pp: 201-210. http://serv1.ist.psu.edu:8080/showciting;jsessionid=D7A85AB8266D292E9CDD29EC6F46CC66?cid=4543043.
10. Buneman, P., S. Davidson, W. Fan, C. Hara and W.C. Tan, 2003. Reasoning about keys for XML. Inform. Syst., 28: 1037-1063. DOI: 10.1016/S0306-4379(03)00028-0.
11. Fan, W. and J. Sim´eon, 2000. Integrity constraints for XML. Proceedings of the 19th ACM Symposium on Principles of Database Systems, May 15-18, ACM Press, New York, USA., pp: 23-34. http://portal.acm.org/citation.cfm?id=335172.
12. Fan, W. and L. Libkin, 2001. On XML integrity constraints in the presence of DTDs. Proceedings of the 20th ACM Symposium on Principles of Database Systems, 2001, ACM Press, New York, USA., pp: 114-125. http://portal.acm.org/citation.cfm?id=375568.
13. Marcelo Arenas and Leonid Libkin, 2004. A normal form for XML documents. ACM Trans. Database Syst., 29: 195-232. http://portal.acm.org/citation.cfm?doid=974750.974757.
14. Klaus-Dieter Schewe, 2005. Redundancy, dependencies and normal forms for XML databases. Proceeding of the 6th Conference on Australasian Database, 2005, Australian Computer Society, Inc., Darlinghurst, Australia, pp: 7-16. http://portal.acm.org/citation.cfm?id=1082224.
15. Florescu, D. and D. Kossman, 1999. Storing and querying XML data using an RDBMS. IEEE Data Eng. Bull., 22: 27-34. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.21.3245.
16. Buneman, P., A. Jung and A. Ohori, 1991. Using power domains to generalize relational databases. Theoret. Comput. Sci., 91: 23-55. http://portal.acm.org/citation.cfm?id=123758.
17. Grahne, G., 1991. The Problem of Incomplete Information in Relational Databases. 1st Edn., Springer-Verlag, New York, USA., ISBN: 3540549196, pp: 156.
18. Gunter, C., 1992. Semantics of Programming Languages: Structures and Techniques. 1st Edn., MIT Press, Cambridge, Mass, ISBN: 10: 0262071436, pp: 441.
19. Levene, M. and G. Loizou, 1998. Axiomatisation of functional dependencies in incomplete relations. Theoret. Comput. Sci., 206: 283-300. http://portal.acm.org/citation.cfm?id=297270.297291.
20. Ramez Elmasri and Shamkant B. Navathe, 2007. Fundamentals of Database System. 5th Edn., Addison-Wesley, ISBN: 0-321-41506-X, pp: 337-360.
21. Tufte, J.S.K., G. He, C. Zhang, D. DeWitt and J. Naughton, 1999. Relational databases for querying XML documents: Limitations and opportunities. Proceedings of the 25th International Conference on Very Large Data Bases, Sep. 07-10, Morgan Kaufmann Publishers Inc., San Francisco, CA., USA., pp: 302-314. http://portal.acm.org/citation.cfm?id=671499
22. Murali, M. and L. Dongwon, 2002. XML to relational conversion using theory of regular tree grammers. Proceeding of the 28th VLDB Conference, Aug. 20-23, Hong Kong, China, pp: 1-12. http://www.cobase.cs.ucla.edu/tech-docs/dongwon/eextt02.pdf.