The Impact of XML Databases Normalization on Design and Usability of Internet Applications

doi:10.3991/ijac.v3i2.1265

Hosam F. El-Sofany[1], Fayed F. M. Ghaleb[2], and Samir A. El-Seoud[3]
[1] Qatar University, Doha, Qatar
[2] Ain Shams University, Cairo, Egypt
[3]Princess Sumaya University for Technology, Amman, Jordan

***Abstract*—Database normalization is a process which eliminates redundancy, organizes data efficiently and improves data consistency. Functional, multivalued, and join dependencies (FDs, MVDs, and JDs) play fundamental roles in relational databases where they provide semantics for the data and at the same time are the foundations for database design. In this study we investigate the issue of defining functional, multivalued and join dependencies and their normal forms in XML database model. We show that, like relational databases, XML documents may contain redundant information, and this redundancy may cause update anomalies. Furthermore, such problems are caused by certain 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 extend the notion of tuple for relational databases to the XML model. We show that an XML tree can be represented as a set of tree tuples. We introduce the definitions of FD, MVD, and JD and new Normal Forms of XML Schema that based on these dependencies (X-1NF, X-2NF, X-3NF, X-BCNF, X-4NF, and X-5NF). We show that our proposed normal forms are necessary and sufficient to ensure all conforming XML documents have no redundancies.**

***Index Terms*— Database design, Functional, Mulivalued, and Join dependencies, Normalization theory, XML**

I. INTRODUCTION

Recently, several researchers studied the issue of Web-based application distinguished three basic levels in every web-based application: *the Web character of the program*, *the pedagogical background*, and *the personalized management of the learning material* [23]. They defined a web-based program as an information system that contains a Web server, a network, a communication protocol like HTTP, and a browser in which data supplied by users act on the system's status and cause changes. The pedagogical background means the educational model that is used in combination with pedagogical goals set by the instructor. The personalized management of the learning materials means the set of rules and mechanisms that are used to select learning materials based on the student's characteristics, the educational objectives, the teaching model, and the available media.

Many works have combined and integrated these three factors in e-learning systems, leading to several standardization projects. Some projects have focused on determining the standard architecture and format for learning environments, such as IEEE Learning Technology Systems Architecture (LTSC), Instructional Management Systems (IMS), and Sharable Content Object Reference Model (SCORM). IMS and SCORM define and deliver XML-based interoperable specifications for exchanging and sequencing learning contents, i.e., learning objects, among many heterogeneous e-learning systems. They mainly focus on the standardization of learning and teaching methods as well as on the modeling of how the systems manage interoperating educational data relevant to the educational process.

The eXtensible Markup Language (XML) has recently emerged as a standard for data representation and interchange on the Internet. With the increase of data-intensive web applications, XML has conquered the field of databases. It is argued that XML can be used as a database language, which would not only support the data exchange on the web. This has led to significant research efforts including: 1) The storage of XML documents in relational databases, 2) Query languages for XML, which lead to the standard query language, XQuery 3) Schema languages for XML, which lead to the widely accepted XML Schema language, 4) Updates of XML documents and, 5) Dependency and normal form theory [1-7].

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 (Document Type Definition), and XML Schema [4]. 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, efficiency of queries, etc. Figure 1, shows an overview of the XML normalization process that we propose.

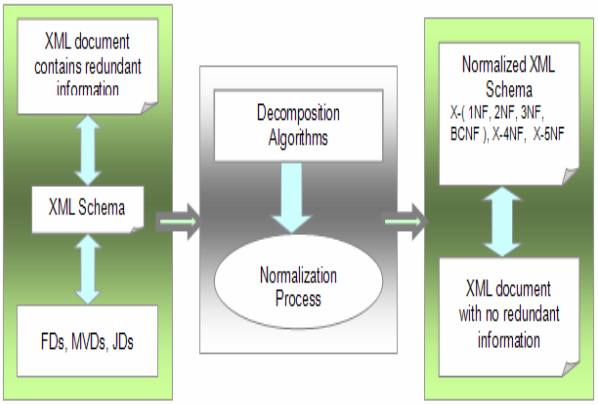


Figure 1. An overview of the XML normalization process

Functional dependency (FD) is one of the integrity constraints for any data model. In relational data model, FDs, MVDs, and JDs are well studied and are widely used in normalization theory and in key algorithms. In recent years, XML has emerged as a widely used data representation and storage format over the World Wide Web. The growing use of XML has necessitated the XML document semantically stronger. XML functional dependency has studied as one of the ways to make the XML data semantically richer [8, 13, 14, 21, 22].

The focus of this paper is on *functional*, *multivalued and join dependencies* and normal form theory. This theory concerns the old question of *well-designed* databases or in other words the syntactic characterization of semantically desirable properties. These properties are tightly connected with dependencies such as *keys*, *functional dependencies*, *weak functional dependencies*, *equality generating dependencies*, *multivalued dependencies*, *inclusion dependencies*, *join dependencies*, etc [9-12]. All these classes of dependencies have been deeply investigated in the context of the relational data model [5, 6]. The work now requires its generalization to XML (trees like) model.

The main contributions of this study are the new definitions of MVD and JD and the new normal forms of XML Schema (X-4NF and X-5NF). We extend our previous research works proposed in [21, 22], and show how to use MVDs and JDs to detect data redundancy in XML document, and then proposed normal forms of XML Schema with respect to the MVD and JD constraints.

II. PRIMARILY DEFINITIONS

To extend the notions of FDs, MVDs and JDs to the XML database model, we represent XML trees as sets of tuples [13, 14, 21, 22], and find the correspondence between documents and relations that leads to the definitions of functional and multivalued dependencies. We first describe the formal definitions of XML Schema (*XSchema*) and the conforming of XML tree to XSchema. 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, date, etc.)
- 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, and f = A default value of a or ε if not known
- $r \subseteq E$ is a finite set of root elements
- $\sum$ 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 $X = (E, A, M, P, r, \sum)$, 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 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: $EPaths(X) = \{ p \in paths(X) \mid last(p) \in E \}$
- 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
- $root \in V$ is called the root of T

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

Definition 5 (conformation and compatibility): 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 ⊨ X) if:

- lab: V → E
- For each v ∈ V, if M(lab(v)) = S, then ele(v) = [s], where s ∈ Str. Otherwise, ele(v) = [v_1, … , v_n] and the string lab(v_1) … lab(v_n) must be in the regular language defined by M(lab(v))
- att is a partial function, att: V × A → Str, s.t. for any v ∈ V and @l ∈ A, att(v, @l) is defined iff @l ∈ P(lab(v))
- lab(root) = r

We say that T is *compatible* with X (written T ▷X) iff paths(T) ⊆ paths(X). Clearly, T ⊨ X ∈ T ▷X

Definition 6 (subsumed): 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 $T_1 \leq T_2$ if:

- $V_1 \subseteq V_2$
- $root_1 = root_2$
- $lab_2|_{V1} = lab_1$
- $att_2|_{V1 \times Â} = att_1$
- $v \in V_1$, $ele_1(v)$ is a sub-list of a permutation of $ele_2(v)$

Definition 7 (equivalence): 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 $T_1 \leq T_2$ and $T_2 \leq T_1$ (i.e., $T_1 \equiv T_2$ iff T_1 and T_2 are equal as unordered trees):

We shall also write $T_1 < T_2$ when $T_1 \leq T_2$ and $T_2 \nleq T_1$

In [21, 22] we extended 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 have shown 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 $t(p_1) \in$ 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 1: Suppose that X is the XML Schema shown below.

```
<?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 Figure 2, [13]. 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) = 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

- $V = \{v \in Vert \mid \exists\ p \in paths(X)$ such that $v = t.p\}$
- If $v = t.p$ and $v \in V$, then $lab(v) = last(p)$
- If $v = t.p$ and $v \in V$, then $ele(v)$ is defined to be the list containing $\{t.p' \mid t.p' \neq \varphi$ and $p' = p.\tau, \tau \in E$, or $p' = p.S$, ordered lexicographically
- If $v = t.p$, $@l \in A$ and $t.p.@l \neq \varphi$, then $att(v, @l) = t.p.@l$

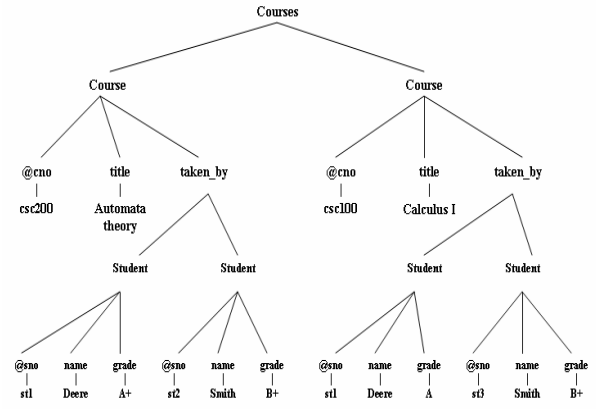
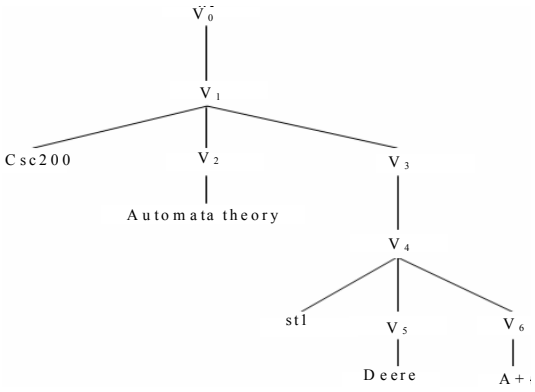


Figure 2. A document containing redundant information

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



Proposition 1: If $t \in T(X)$, then $tree_X(t) \rhd X$.

- If we have two tree tuples t_1, t_2, we write $t_1 \subseteq t_2$ if whenever $t_1.p$ is defined, then $t_2.p$ is also defined and $t_1.p \neq \varphi \in t_1.p = t_2.p$
- As usual, $t_1 \subset t_2$ means $t_1 \subseteq t_2$ and $t_1 \neq t_2$
- Given two sets of tree tuples, Y and Z, we write: $Y \subseteq^b Z$, if: $\forall\ t_1 \in Y \Rightarrow t_2 \in Z$ s.t. $t_1 \subseteq t_2$

Definition 10 ($tuples_X$): Given XML Schema X and an XML tree T such that $T \rhd X$, $tuples_X(T)$ is defined to be the set of maximal tree tuples t (with respect to $\subseteq$), s.t. $tree_X(t)$ is subsumed by T, that is:

$$\max_{\subseteq}\{\ t \in T(X) \mid tree_X(t) \leq T\ \}$$

Note that:

- $T_1 \equiv T_2$ implies $tuples_X(T_1) = tuples_X(T_2)$
- We have proved the following proposition [21, 22].

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

Example 3: In example 1, we saw the XML Schema X and a tree T conforming to X, and 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 $\leq$, trees containing all tuples in Y.

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

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

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 $\equiv$. We have proved the following theorem [21, 22].

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

Note that:

- We say that $Y \subseteq T(X)$ 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 $t \in Y$, $tree_X(t) \leq T$.
- We have proved the following proposition, and corollary [21, 22]:

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

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

Corollary: For a X-compatible set of tree tuples Y: $trees_X(tuples_X(trees_X(Y))) = trees_X(Y)$.

III. NORMAL FORMS BASED ON FUNCTIONAL DEPENDENCIES

A. 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 $S_1, S_2 \subseteq$ paths(X), $S_1, S_2 \neq \varphi$. The set of all FDs over X is denoted by FD(X).

For $S \subseteq$ 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 $\triangleright$X and $S_1 \cup S_2 \subseteq$ paths(T), we say that T satisfies $S_1 \rightarrow S_2$ (written T $\models S_1 \rightarrow S_2$), if $\forall$ $t_1, t_2 \in tuples_X(T)$, $t_1.S_1 = t_2.S_1$ and $t_1.S_1 \neq \varphi \in t_1.S_2 = t_2.S_2$.

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 $\rightarrow$ courses.course (FD1)

Another FD says that two distinct student sub-elements of the same course cannot have the same sno:

{courses.course,courses.course.taken_by.student.@sno} $\rightarrow$ 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 $\rightarrow$ 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, φ) $\vdash$ f.

B. 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, $\sum$) be XML Schema, a constraint $\sum$ over X has one of the following forms:

Key: e(l) $\rightarrow$ e, where e$\in$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 $e_1, e_2 \in$ 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 $\subseteq$ 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

C. 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:

D. 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 $\in S_1$, $(S_1\text{-}\{p\})$ does not functional determine S_2).

Definition 19: A FD $S_1 \rightarrow S_2$ is called *partial dependency* if, for some p $\in S_1$, $(S_1\text{-}\{p\}) \rightarrow S_2$ 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>
```

E. 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 → {Emp_Dept.Ename, Emp_Dept.Bdate, Emp_Dept.Address, Emp_Dept.Dnumber }

FD2: Emp_Dept.Dnumber → {Emp_Dept.Dname, Emp_Dept.DmgrSsn}

Note that:

The dependency:

Emp_Dept.Ssn→ Emp_Dept.DmgrSsn is transitive through Emp_Dept.Dnumber in Emp_Dept, because both the FDs:

Emp_Dept.Ssn → Emp_Dept.Dnumber and

Emp_Dept.Dnumber → 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, ∑) is in *third normal form* (X-3NF) if it satisfies X-2NF and no (elements e ∈ E or l ⊆ 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>
```

F. Boyce-codd normal form of XML schema (X-BCNF)

X-BCNF is 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, ∑) 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 ⊆ FD(X), (X, F) is in X-BCNF iff for every nontrivial FD f ∈ $(X, F)^+$ of the form S → p.@l or S → p.S, it is the case that, S → p ∈ $(X, F)^+$.

The intuition is as follows: Suppose that S → p.@l ∈ $(X, F)^+$. 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 → p must be implied by (X, F).

In definition 24, we suppose that, f is a nontrivial FD. Indeed, the trivial FD p.@l → p.@l is always in $(X, F)^+$, but often p.@l → p ∉ $(X, F)^+$, 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 →
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 → courses.info, can be easily verified to be in X-BCNF.

IV. NORMAL FORMS BASED ON MULTIVALUED DEPENDENCIES

We have discussed only FD, which is by far the most important type of dependency in XML database design theory. However, in many cases XML documents have constraints that cannot be specified as FD. In this part of the article, we discuss the concept of multivalued

dependency and define fourth normal form of XML Schema (X-4NF), based on this dependency.

Definition 25 (multivalued dependency): Given an XML Schema X, a *multivalued dependency* (MVD) over X is an expression of the form: $S_1 \twoheadrightarrow S_2$ where S_1, $S_2 \subseteq$ paths(X), S_1, $S_2 \neq \varphi$, specifies the following constraint on any path state S of paths(X): If two paths t_1, $t_2 \in$ T(X) exist in paths(X) such that $t_1.S_1 = t_2.S_1$, then two paths t_3, $t_4 \in$ T(X) should also exist in paths(X) with the following properties, where we use S_3 to denote (X-($S_1 \cup S_2$)):

$t_3.S_1 = t_4.S_1 = t_1.S_1 = t_2.S_1$

$t_3.S_2 = t_1.S_2$ and $t_4.S_2 = t_2.S_2$

$t_3.S_3 = t_2.S_3$ and $t_4.S_3 = t_1.S_3$

Whenever $S_1 \twoheadrightarrow S_2$ holds, we say that S_1 *multi-determines* S_1. Because of the symmetry in the definition, whenever $S_1 \twoheadrightarrow S_2$ holds in X, so does $S_1 \twoheadrightarrow S_3$. Hence $S_1 \twoheadrightarrow S_2$ implies $S_1 \twoheadrightarrow S_3$, therefore we can write it as: $S_1 \twoheadrightarrow S_2|S_3$.

Note that: an MVD $S_1 \twoheadrightarrow S_2$ in XML Schema X is called a **trivial MVD** if: $S_2 \subseteq S_1$ or ($S_1 \cup S_2$) = paths(X).

A. Fourth normal form of XML schema (X-4NF)

Definition 26 (fourth normal form): An XML Schema X is in *fourth normal form* (X-4NF) with respect to a set of dependencies *F* (that includes functional dependencies and multivalued dependencies) if, for every *nontrivial* multivalued dependency $S_1 \twoheadrightarrow S_2 \in F^+$, S_1 is a superkey for X.

Note: F^+ is the (complete) set of all dependencies (functional or multivalued) that will hold in every path paths $t \in$ T(X) that satisfies *F*. It is also called the **closure** of *F*.

Example 12: Consider the following XML Schema called "EMP", with the following MVDs:

EMP.Ename $\twoheadrightarrow$ EMP.Pname, and

EMP. Ename $\twoheadrightarrow$ EMP.Dname

```
<xs: complexType name “EMP”>
<xs:sequence>
        <xs: element name = “Ename” type = “string”/>
        <xs: element name = “Pname” type = “string”/>
        <xs: element name = “Dname” type = “string”/>
<xs:sequence>
<xs:complexType>
<xs: key name = “empEnamKey”>
        <xs: selector xpath = “EMP”/>
        <xs: field xpath = “Ename”/>
</xs:key>
<xs: key name = “empPnamKey”>
        <xs: selector xpath = “EMP”/>
        <xs: field xpath = “Pname”/>
</xs:key>
<xs: key name = “empDnamKey”>
        <xs: selector xpath = “EMP”/>
        <xs: field xpath = “Dname”/>
</xs:key>
```

The XML Schema "EMP" has no FD since it is an all-key XML Schema. Because BCNF constraints are stated in terms of FD only, an all-key Schema is always in BCNF by default. Hence EMP is in X-BCNF. However, EMP is not in X-4NF because in the nontrivial MVDs EMP.Ename $\twoheadrightarrow$ EMP.Pname and EMP.Ename $\twoheadrightarrow$ EMP.Dname, and Ename is not a superkey of EMP. We decompose EMP into EMP-PROJECTS and EMP-DEPENDENTS:

```
<xs: complexType name “EMP-PROJECTS”>
<xs:sequence>
        <xs: element name = “Ename” type = “string”/>
        <xs: element name = “Pname” type = “string”/>
        <xs:sequence>
<xs:complexType>
<xs: key name = “empEnamKey”>
        <xs: selector xpath = “EMP-PROJECTS”/>
        <xs: field xpath = “Ename”/>
</xs:key>
<xs: key name = “empPnamKey”>
        <xs: selector xpath = “EMP-PROJECTS”/>
        <xs: field xpath = “Pname”/>
</xs:key>

<xs: complexType name “EMP-DEPENDENTS”>
<xs:sequence>
        <xs: element name = “Ename” type = “string”/>
        <xs: element name = “Dname” type = “string”/>
<xs:sequence>
<xs:complexType>
<xs: key name = “empEnamKey”>
        <xs: selector xpath = “EMP-DEPENDENTS”/>
        <xs: field xpath = “Ename”/>
</xs:key>
<xs: key name = “empDnamKey”>
        <xs: selector xpath = “EMP-DEPENDENTS”/>
        <xs: field xpath = “Dname”/>
</xs:key>
```

Both EMP-PROJECTS and EMP-DEPENDENTS are in X-4NF, because the MVDs:

EMP-PROJECTS.Ename $\twoheadrightarrow$ EMP-PROJECTS.Pname in EMP-PROJECTS and EMP-DEPENDENTS.Ename $\twoheadrightarrow$ EMP-DEPENDENTS.Dname, in EMP-DEPENDENTS are trivial MVDs. No other FDs and nontrivial MVDs hold in either EMP-PROJECTS or EMP-DEPENDENTS.

Note that: The relational view of the XML Schemas "EMP", "EMP-PROJECTS", "EMP-DEPENDENTS" and the corresponding relation states can be illustrated in the following Figure: EMP (Ename, Pname, Dname)

(a) **EMP**

ENAME	PNAME	DNAME
Smith	X	John
Smith	Y	Anna
Smith	X	Anna
Smith	Y	John

(b) **EMP_PROJECTS**

ENAME	PNAME
Smith	X
Smith	Y

EMP_DEPENDENTS

ENAME	DNAME
Smith	John
Smith	Anna

V. NORMAL FORMS BASED ON JOIN DEPENDENCIES

Definition 27 (join dependency) : A join dependency (JD), denoted by JD(X_1, X_2, ..., X_n), specified on XML Schema X, specifies a constraint on the XML trees T of X. The constraint states that every complete XML tree T of X should have a non-additive join decomposition into X_1, X_2, ..., X_n, that is, for every such XML tree T we have:

$$* (\pi_{X1}(T), \pi_{X2}(T), ..., \pi_{Xn}(T)) = T$$

Note that:

- An MVD is a special case of a JD where n = 2.
- A join dependency JD(X_1, X_2, ..., X_n), specified on XML Schema X, is a **trivial JD** if one of the XML Schema X_i in JD(X_1, X_2, ..., X_n) is equal to X.

A. *Fifth normal form of XML schema:*

Definition 28 (fifth normal form): An XML Schema X is in fifth normal form (X-5NF) or (Project-Join Normal Form (X-PJNF)) with respect to a set of dependencies *F* of functional, multivalued, and join dependencies if, for every nontrivial join dependency JD(X_1, X_2, ..., X_n) $\in$ *F*+ (that is, implied by *F*), every X_i is a superkey of X.

Example 13: Consider the following XML Schema called "SUPPLY", with no MVDs is in X-4NF but not in X-5NF:

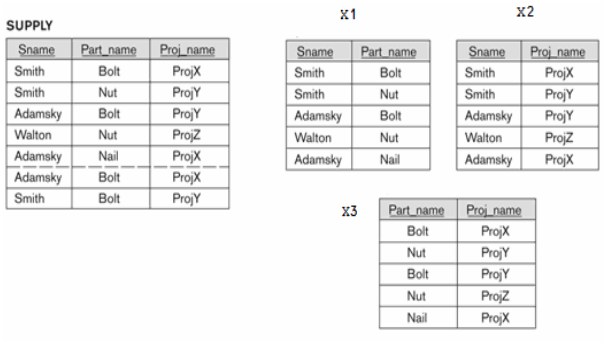
```
<xs: complexType name “SUPPLY”>
<xs:sequence>
        <xs: element name = “Sname” type = “string”/>
        <xs: element name = “Part_name” type = “string”/>
        <xs: element name = “Proj_name” type = “string”/>
<xs:sequence>
<xs:complexType>
<xs: key name = “supSnamKey”>
        <xs: selector xpath = “SUPPLY”/>
        <xs: field xpath = “Sname”/>
</xs:key>
<xs: key name = “supPart_namKey”>
        <xs: selector xpath = “SUPPLY”/>
        <xs: field xpath = “Part_name”/>
</xs:key>
<xs: key name = “supProj_namKey”>
        <xs: selector xpath = “SUPPLY”/>
        <xs: field xpath = “Proj_name”/>
</xs:key>
```

Suppose that the following additional constraint always holds: "*whenever a supplier s supplies part p, and a project j uses part p, and the supplier s supplies at least one part to project j, then supplier s will also be supplying part p to project j*". This constraint can be restated in other ways and specifies a join dependency JD(X_1, X_2, X_3) among the three projections X_1, X_2, and X_3 defined as following:

```
<xs: complexType name “X1”>
<xs:sequence>
        <xs: element name = “Sname” type = “string”/>
        <xs: element name = “Part_name” type = “string”/>
<xs:sequence>
<xs:complexType>
<xs: complexType name “X2”>
<xs:sequence>
        <xs: element name = “Sname” type = “string”/>
        <xs: element name = “Proj_name” type = “string”/>
<xs:sequence>
<xs:complexType>
<xs: complexType name “X3”>
<xs:sequence>
        <xs: element name = “Part_name” type = “string”/>
        <xs: element name = “Proj_name” type = “string”/>
<xs:sequence>
<xs:complexType>
```

This shows how the SUPPLY, XML Schema with the *join dependency* is decomposed into three XML Schemas "X1", "X2", and "X3" that are each in X-5NF.

Note that: The relational view of the XML Schemas " SUPPLY", "X1", "X2", and "X3" and the corresponding relation states can be illustrated in the following Figure:



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, ISBN:1-58113-348-0, pp: 201-210.

[10] Buneman, P., S. Davidson, W. Fan, C. Hara and W.C. Tan, 2003. Reasoning about keys for XML. Proceedings of the 8th International Workshop on Database Programming Languages. ISSN:0306-4379, pp: 1037 – 1063.

[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. doi:10.1016/0304-3975(91)90266-5

[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. doi:10.1016/S0304-3975(98)80029-7

[20] Ramez Elmasri and Shamkant B. Navathe, 2007. Fundamentals of Database System. 5th. Edn.,

[21] Hosam F. El-Sofany, and Samir A. El-Seoud, "Schema Design and Normalization Algorithm for XML Databases Model", *International Journal of Emerging Technologies in Learning* – iJET, Volume 4, Issue 2, Pages 11-21, doi:10.3991/ijet.v4i2.768, June 2009.

[22] Hosam Farouk El-Sofany, "Extending the Concepts of Normalization from Relational Databases to Extensible-Markup-Language Databases Model", *Journal of Computer Science* 4(9): P: 729-740, 2008, ISSN 1549-3636- Science Publications U.S.A, 2008.

[23] F. P. Rokou et al., "Modeling web-based educational systems: process design teaching model," *Educational Technology and Society*, Vol. 7, pp. 42-50, 2004.

AUTHORS

Hosam F. El-Sofany is with the Department of Computer Science and Engineering, College of Engineering, Qatar University.

Fayed F. M. Ghaleb is with the Department of Mathematics, Faculty of Science, Ain Shams University.

Samir A. El-Seoud is with Princess Sumaya University for Technology, Amman, Jordan.

Manuscript received March 10, 2010. Published as resubmitted by the authors May 22nd, 2010.