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

Hosam F. El-Sofany¹, Fayed F. M. Ghaleb², and Samir A. El-Seoud³
¹Qatar University, Doha, Qatar
²Ain Shams University, Cairo, Egypt
³Princess Sumaya University for Technology, Amman, Jordan

doi:10.3991/ijac.v3i2.1265

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, Multivalued, 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.
Deeply investigated in the context of the relational data etc [9-12]. All these classes of dependencies have been, equality generating dependencies, functional dependencies, multivalued dependencies, and join dependencies, tightly connected with dependencies such as weak functional 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 focus of this paper is on functional, multivalued and join dependencies and normal form theory. This theory concerns the old question of functional and multivalued dependencies, inclusion dependencies, weak functional and multivalued dependencies, and join dependencies, equality generating dependencies, multivalued dependencies, inclusion dependencies, 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 representation and storage format over the World Wide Web. The growing use of XML has necessitated the XML representation and storage format over the World Wide Web.

The main contributions of this study are the new definitions of MVD and JD and the new normal forms of XML (trees like) model. The integrity constraints we consider are keys (P.K, F.K, …) and dependencies (functional and inclusion). In particular, we extend the definitions of functional and multivalued dependencies. 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 representation and storage format over the World Wide Web.

### Definition 1 (XSchema): An XSchema is denoted by 6-tuple: X = (E, A, M, P, r, S), where:
- E is a finite set of element names.
- A is a finite set of attribute names.
- M is a function from E to its element type definitions: i.e., M(e) = α, where e ∈ E and α is a regular expression:
  \[ α := ε | t | α + α | α | α^* | α^+ \]
  where, ε denotes the empty element, t ∈ DT, "+" for the union, "+" for the concatenation, \( α^* \) for (α + ε) and \( α^+ \) for (α + ε)\.
- P is a function from an attribute name a to its attribute type definition: i.e., P(a) = β, where β is a 4-tuple (t, n, d, f), where: t ∈ 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 is a finite set of root elements.
- S 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, S), a string p = p₁…pₙ, is a path in X if, pᵢ = r, pᵢ is in the alphabet of M(pᵢ₋₁), for each i ∈ [2, n − 1] and pᵢ is in the alphabet of M(pᵢ₋₁) or pᵢ = @l for some @l ∈ P(pᵢ₋₁).

- We let paths(X) stand for the set of all paths in X and 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 is a finite set of vertices (nodes).
- lab : V → E
- ele : V → Str ∪ V∗
- att is a partial function V × A → Str. For each v ∈ V, the set \{v ∈ V | att(v, @l) is defined\} is required to be finite.
- root ∈ V is called the root of T.

### Definition 4 (path in XML tree): Given an XML tree T, a string: p₁…pₙ with p₁, …, pₙ ∈ E and pᵢ ∈ (E ∪ S), is a path in T if there are vertices vᵢ, …, vᵢ₋₁ ∈ V such that:
- \( v₁ = \text{root}, \ vᵢ₋₁ \) is a child of \( vᵢ \) (1 ≤ i ≤ n − 2), \( \text{lab}(vᵢ) = pᵢ \) (1 ≤ i ≤ n − 1).
- If pᵢ ∈ E, then there is a child vᵢ of vᵢ₋₁, s.t. \( \text{lab}(vᵢ) = pᵢ \).
- If pᵢ = @l, with @l ∈ A, then att(vᵢ₋₁, @l) is defined.
- We let paths(T) stand for the set of paths in T.
Definition 5 (conformation and compatibility): Given an XSchema \( X = (E, A, M, P, r, \Sigma) \) and an XML tree \( T = (V, \text{lab}, \text{ele}, \text{att}, \text{root}) \), we say that \( T \) is valid w.r.t. \( X \) or \( T \) conforms to \( X \) written as \( (T \models X) \) if:

- \( \text{lab} : V \rightarrow E \)
- For each \( v \in V \), if \( M(\text{lab}(v)) = S \), then \( \text{ele}(v) = [s] \), where \( s \in \text{Str} \). Otherwise, \( \text{ele}(v) = [v_1, \ldots, v_n] \) and the string \( \text{lab}(v_i) \ldots \text{lab}(v_n) \) must be in the regular language defined by \( M(\text{lab}(v)) \)
- \( \text{att} \) is a partial function, \( \text{att} : V \times A \rightarrow \text{Str} \) such that for any \( v \in V \) and \( @l \in A \), \( \text{att}(v, @l) \) is defined iff \( @l \in P(\text{lab}(v)) \)
- \( \text{lab}(\text{root}) = r \)

We say that \( T \) is compatible with \( X \) (written \( T \parallel X \)) if:

- \( \text{paths}(T) \subseteq \text{paths}(X) \). Clearly, \( T \models X \iff T \parallel X \)

Definition 6 (subsumed): Given two XML trees \( T_1 = (V_1, \text{lab}_1, \text{ele}_1, \text{att}_1, \text{root}_1) \) and \( T_2 = (V_2, \text{lab}_2, \text{ele}_2, \text{att}_2, \text{root}_2) \), we say that \( T_1 \) is subsumed by \( T_2 \), written as \( T_1 \subseteq T_2 \) if:

- \( V_1 \subseteq V_2 \)
- \( \text{root}_1 = \text{root}_2 \)
- \( \text{lab}_{2|V_1} = \text{lab}_1 \)
- \( \text{att}_{2|V_1 \times A} = \text{att}_1 \)
- \( v \in V_1 \), \( \text{ele}_2(v) \) is a sub-list of a permutation of \( \text{ele}_1(v) \)

Definition 7 (equivalence): Given two XML trees \( T_1 \) and \( T_2 \), we say that \( T_1 \) is equivalent to \( T_2 \) written \( T_1 = T_2 \) if:

- \( T_1 \subseteq T_2 \) and \( T_2 \subseteq T_1 \) (i.e., \( T_1 = T_2 \) 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 \( T_2 \neq T_1 \)

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 database, a tuple is a function that assigns to each attribute a value from the corresponding domain. In a relational database, a tuple is a function that assigns to each attribute a value from the corresponding domain. In a relational database, a tuple is a function that assigns to each attribute a value from the corresponding domain.

Given XML Schema \( X = (E, A, M, P, r, \Sigma) \) and 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 \( \text{paths}(X) \) such as:

\[
\begin{align*}
&\text{t}(\text{courses}) = v_0 \\
&\text{t}(\text{courses.course}) = v_1 \\
&\text{t}(\text{courses.course.cno}) = \text{csc200} \\
&\text{t}(\text{courses.course.grade}) = v_3 \\
&\text{t}(\text{courses.course.taken_by}) = v_4 \\
&\text{t}(\text{courses.course.take_by_student}) = \text{Automata Theory} \\
&\text{t}(\text{courses.course.take_by_student.cno}) = \text{csc200} \\
&\text{t}(\text{courses.course.take_by_student.grade}) = v_5 \\
&\text{t}(\text{courses.course.take_by_student.student}) = \text{Deere} \\
&\text{t}(\text{courses.course.take_by_student.student.grade}) = A^+ \\
\end{align*}
\]

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 \( \text{paths}(X) \) such as:

- \( \text{t}(\text{courses}) = v_0 \)
- \( \text{t}(\text{courses.course}) = v_1 \)
- \( \text{t}(\text{courses.course.cno}) = \text{csc200} \)
- \( \text{t}(\text{courses.course.grade}) = v_3 \)
- \( \text{t}(\text{courses.course.taken_by}) = v_4 \)
- \( \text{t}(\text{courses.course.take_by_student}) = \text{Automata Theory} \)
- \( \text{t}(\text{courses.course.take_by_student.cno}) = \text{csc200} \)
- \( \text{t}(\text{courses.course.take_by_student.grade}) = v_5 \)
- \( \text{t}(\text{courses.course.take_by_student.student}) = \text{Deere} \)
- \( \text{t}(\text{courses.course.take_by_student.student.grade}) = A^+ \)

Definition 8 (tree tuples): Given XML Schema \( X = (E, A, M, P, r, \Sigma) \), a tree tuple \( t \in X \) is a function, \( t \) assigns values to each path in \( \text{paths}(X) \)

\[
\begin{align*}
&\text{t}(\text{courses}) = v_0 \\
&\text{t}(\text{courses.course}) = v_1 \\
&\text{t}(\text{courses.course.cno}) = \text{csc200} \\
&\text{t}(\text{courses.course.grade}) = v_3 \\
&\text{t}(\text{courses.course.taken_by}) = v_4 \\
&\text{t}(\text{courses.course.take_by_student}) = \text{Automata Theory} \\
&\text{t}(\text{courses.course.take_by_student.cno}) = \text{csc200} \\
&\text{t}(\text{courses.course.take_by_student.grade}) = v_5 \\
&\text{t}(\text{courses.course.take_by_student.student}) = \text{Deere} \\
&\text{t}(\text{courses.course.take_by_student.student.grade}) = A^+ \\
\end{align*}
\]

Definition 9 (treeX): Given XML Schema \( X = (E, A, M, P, r, \Sigma) \) and a tree tuple \( t \in T(X) \), \( \text{treeX}(t) \) is defined to be an XML tree \( (V, \text{lab}, \text{ele}, \text{att}, \text{root}) \), where:

- \( \text{root} = t.r \)

Example 1: Suppose that \( X \) is the XML Schema shown below.

```xml
<?xml version = "1.0" encoding = "ISO-8859-1"?>
<xs:schema xmlns:xs = "http://www.w3.org/2001/XMLSchema">
  <xs:element name = "courses">
    <xs:complexType>
      <xs:sequence>
        <xs:element name = "name" type = "xs:string"/>
        <xs:element name = "grade" type = "xs:string"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```
THE IMPACT OF XML DATABASES NORMALIZATION ON DESIGN AND USABILITY OF INTERNET APPLICATIONS

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

Proposition 1: If \( t \in T(X) \), then \( \text{tree}_X(t) \mid X \).

- We have proved the following proposition [21, 22].

Proposition 2: If \( T \mid X \), then \( \text{tuples}_X(T) \) is a finite subset of \( T(X) \). Furthermore, \( \text{tuples}_X(t) \) is monotone: \( T_1 \leq T_2 \) implies \( \text{tuples}_X(T_1) \subseteq \text{tuples}_X(T_2) \).

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 \( \text{tuples}_X(T) \): \[
\{(v_0, v_1, csc200, v_2, Automata Theory, v_3, v_4, st1, v_5, Deere, v_6, A+)\}
\{v_0, v_1, v_3, v_4, st2, v_5, Smith, v_6, B-\}
\{(v_0, v_10, mat100, v_11, Calculus I, v_12, v_13, st1, v_14, Deere, v_15, A)\}
\{v_0, v_10, mat100, v_11, Calculus I, v_12, v_16, st3, v_17, Smith, v_18, B+\}\]

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 \( Y \subseteq T(X) \), \( \text{trees}_X(Y) \) is defined to be:
\[
\text{min}_{\leq} \left\{ T \mid T \mid X \text{ and } \forall t \in Y, \text{tree}_X(t) \leq T \right\}
\]

Note that:
- We say that \( Y \subseteq T(X) \) is \( X \)-compatible if there is an XML tree \( T \mid X \) and \( Y \subseteq \text{tuples}_X(T) \).
- For \( X \)-compatible set of tree tuples \( Y \), there is always an XML tree \( T \): for every \( t \in Y \), \( \text{tree}_X(t) \leq T \).
- 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 \( T \mid X \) and \( \text{trees}_X(Y) = [T] \)
- \( Y \subseteq^b \text{tuples}_X(\text{trees}_X(Y)) \)

Corollary: For a \( X \)-compatible set of tree tuples \( Y \): \( \text{trees}_X(\text{tuples}_X(\text{trees}_X(Y))) = \text{trees}_X(Y) \).
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 \( S_1, S_2 \subseteq \text{paths}(X) \), \( S_1, S_2 \neq \emptyset \). 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, \( t.S \neq \emptyset \) means \( t.p \neq \emptyset \) \( \forall p \in S \).

Definition 13: If \( S_1 \rightarrow S_2 \in \text{FD}(X) \) and T is an XML tree s.t. \( T \supseteq X\) 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 \( t_1.S_2 \neq \emptyset \) \( \in t_1.S_2 = t_2.S_2 \).

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 extend to the equivalence classes, since, for any FD f and \( T = T' \), \( T \models f \) iff \( T' \models f \).

We write \( T \models F \), for \( F \in \text{FD}(X) \), 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:

\[
\text{courses.course.@cno} \rightarrow \text{courses.course} \quad \text{(FD1)}
\]

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

\[
\{\text{courses.course,courses.course.taken_by.student.@sno}\} \rightarrow \text{courses.course.taken_by.student} \quad \text{(FD2)}
\]

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

\[
\text{courses.course.taken_by.student.@sno} \rightarrow \text{courses.course.taken_by.student.name.S} \quad \text{(FD3)}
\]

Definition 15: Given XML Schema X, a set F \( \subseteq \text{FD}(X) \) and f \( \in \text{FD}(X) \), we say that (X, F) implies f, written \( (X, F) \models f \), 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 \( (X, \emptyset) \models f \).

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, \( \Sigma \)) be XML Schema, a constraint \( \Sigma \) 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 \( e_1, e_2 \subseteq E \) 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 \( e_1(l_1) \subseteq e_2(l_1) \) 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 \( e_2(l_1) \rightarrow e_2 \).

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.

Definition 18: A FD \( S_1 \rightarrow S_2 \), where \( S_1, S_2 \subseteq \text{paths}(X) \) 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-\{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-\{p\}) \rightarrow S_2 is hold.

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

\[
<\!\text{xs:complexType name = "Emp_Proj"}>
<\!\text{xs:sequence}>
<\!\text{xs:element name = "Sss" type = "string"}>
<\!\text{xs:element name = "Pnumber" type = "string"}>
<\!\text{xs:element name = "Plocation" type = "string"}>
<\!\text{xs:complexType name = "Emp_Proj"}>

http://www.i-jac.org
With the following FDs:
FD1: \{Emp\_Proj.Sss, Emp\_Proj.Pnumber\} → Emp\_Proj.Hours
FD2: Emp\_Proj.Sss → Emp\_Proj.Ename

Note that:
FD1 is a full FD (neither Emp\_Proj.Sss → Emp\_Proj.Hours nor Emp\_Proj.Pnumber → Emp\_Proj.Hours holds).

Definition 20 (X-2NF): An XML Schema X = (E, A, M, P, r, Σ) 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:

E. Third Normal Form of XML Schema (X-3NF)

X-3NF is based on the concept of transitive dependency.

Definition 21: A FD S1 → S2, where S1, S2 ⊆ 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 S1 → Z and Z → S2 hold.

Example 9: Consider the following XML Schema called "Emp\_Dept":

Emp\_Dept(Ssn, Ename, Bdate, Address, Dnumber, Dname, DmgrSsn)
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
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, Õ) 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)

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, Õ) is in Boyce-Codd Normal Form (X-BCNF) if whenever a nontrivial FD S1 → S2 holds in X, where S1, S2 ⊆ paths(X), then S1 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: $S_1 \rightarrow S_2$ where $S_1, S_2 \subseteq \text{paths}(X)$. $S_1, S_2 \neq \phi$, specifies the following constraint on any path state $S$ of $\text{paths}(X)$: If two paths $t_1, t_2 \in T(X)$ exist in $\text{paths}(X)$ such that $t_1.S_1 = t_2.S_1$, then two paths $t_1, t_2 \in T(X)$ should also exist in $\text{paths}(X)$ with the following properties, where we use $S_1$ to denote $(X - (S_1 \cup S_2))$:

- $t_1.S_2 = t_2.S_2 = t_1.S_1 = t_2.S_1$
- $t_1.S_2 = t_1.S_2$ and $t_4.S_2 = t_3.S_2$
- $t_1.S_3 = t_2.S_3$ and $t_4.S_3 = t_1.S_3$

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

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

**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 $S_1 \rightarrow S_2 \in F^+$, $S_1$ is a superkey for X.

Note: $F^+$ is the (complete) set of all dependencies (functional or multivalued) that will hold in every path $t \in T(X)$ 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 $\rightarrow$ EMP.Pname, and
- EMP. Ename $\rightarrow$ EMP.Dname

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 $\rightarrow$ EMP.Pname and EMP.Ename $\rightarrow$ EMP.Dname, and Ename is not a superkey of EMP. We decompose EMP into EMP-PROJECTS and EMP-DEPENDENTS:

- EMP.EType $\rightarrow$ EMP-PROJECTS.Pname
- EMP.EType $\rightarrow$ EMP-DEPENDENTS.Dname

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

- EMP-PROJECTS.Ename $\rightarrow$ EMP-PROJECTS.Pname in EMP-PROJECTS and EMP-DEPENDENTS, and
- 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)
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:

* (π_{X_1}(T), π_{X_2}(T), ..., π_{X_n}(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 Xi 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) ∈ 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:

\[
\begin{align*}
\text{SUPPLY} & : \text{complexType} \\
& (: \text{sequence} \\
& \quad (: \text{element} \text{name = "Sname" type = "string"}) \\
& \quad (: \text{element} \text{name = "Part_name" type = "string"}) \\
& \quad (: \text{element} \text{name = "Proj_name" type = "string"}) \\
& \quad (: \text{complexType}) \\
& \quad (: \text{complexType} \text{name = "X1"}) \\
& \quad (: \text{complexType} \text{name = "X2"}) \\
& \quad (: \text{complexType} \text{name = "X3"})
\end{align*}
\]

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:

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:

\[
\begin{align*}
\text{supply} & : \text{complexType} \text{name = "SUPPLY"} \\
& (: \text{sequence} \\
& \quad (: \text{element} \text{name = "Sname" type = "string"}) \\
& \quad (: \text{element} \text{name = "Part_name" type = "string"}) \\
& \quad (: \text{element} \text{name = "Proj_name" type = "string"}) \\
& \quad (: \text{complexType} \text{name = "X1"}) \\
& \quad (: \text{complexType} \text{name = "X2"}) \\
& \quad (: \text{complexType} \text{name = "X3"}) \\
& \quad (: \text{complexType})
\end{align*}
\]

REFERENCES


AUTHORS

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

Fayed 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.

iJAC – Volume 3, Issue 2, May 2010