Constraint Preserving Mapping Algorithm for XML Storage

Kamsuriah Ahmad\textsuperscript{a}, Ali Mamat\textsuperscript{b}, Hamidah Ibrahim\textsuperscript{c}, Shahrul Azman Mohd Noah\textsuperscript{d}

\textsuperscript{a,d}Fakulti Teknologi dan Sains Maklumat
Universiti Kebangsaan Malaysia,
Tel: 03-89216730 Fax: 03-89296184,
E-mail: kam@ftsm.ukm.my; samn@ftsm.ukm.my

\textsuperscript{b,c}Fakulti Sains Komputer dan Teknologi Maklumat
Universiti Putra Malaysia,
Tel: 03-89466557 Fax: 03-89466576
E-mail: ali@fsktm.upm.edu.my; hamidah@fsktm.upm.edu.my

Abstract - The use of XML as the common format for representing, exchanging, storing, integrating and accessing data poses many new challenges to database systems. Most of application data are stored in relational databases due to its popularity and rich development experiences over it. Therefore, how to provide a proper mapping approach from XML model to relational model becomes the major research problem. The mapping from XML to relational is not an easy task because the data model of an XML document is fundamentally different from that of a relational database. Especially the structure of an XML document is hierarchy and the XML elements may be nested and repeated. So it is also useful in information exchange and data integration for preserving the semantics of data originating in XML to relational databases. Although several approaches exist, they are incomplete in the sense that they focus only on a part of constraints and ignore the constraints for XML as expressed in functional dependencies. In this paper, we propose an algorithm how to preserve these constraints while mapping from XML to relational schema. Compared with other methods, our approach can preserve more XML constraints, while reduced nesting and redundant data.

1. INTRODUCTION

Extensible Markup Language (XML) is fast emerging as the dominant standard for data interchange and data representation on the web. Its nested, self-describing structure provides a simple yet flexible means for application to model and exchange data. Data exchange involves transformations of data, and therefore the “transformed” data can be seen as a view of its source. Thus, the problem we investigate is how constraints are propagated to views. Even though XML can exist as a database but the capability is very limited when compared with sophisticated storage and query ability already provided by existing relational database systems. We expect that the needs to convert data formats between XML and relational models will grow substantially. But the problem with XML is that it is only syntax and does not carry the semantics of the data. Sometimes it is necessary to map XML data to relations considering the constraints and then relational data to publish in XML according to the constraints.

More recently, keys \cite{3}, foreign keys \cite{15} and functional dependencies \cite{2} have been proposed to capture semantic constraints, and various aspects of these proposals have found their way into XML-Data\cite{12} and XML Schema\cite{15}. Among these proposals, functional dependencies for XML \cite{2} are important to capture the semantics of XML data. However, in relational databases, the semantic constraints have been proved useful in recognizing keys, normalizing to make a good design, preventing update anomaly, reduced redundancy and etc. Functional dependencies (FDs) are critical part of its semantics and FDs for XML, called XFDs are the counterpart of those for relational data. They must be taken advantage of in the process of mapping. A natural question to ask, therefore, is how information about constraints in FDs can be used to generate a good database schema.

In this paper, we analyze constraints for XML as expressed in functional dependencies, and proposed an algorithm on how to preserve these constraints in relational schema. The rest of the paper is organized as follows. Section 2 gives the motivations of the studies. Section 3 gives the definitions and notations used in the paper. The algorithm for mapping from data type definition (DTD) in the presence of functional dependencies to
11. Motivations

Although an array of researches has addressed the particular issues lately [5],[7],[8],[9],[10],[11],[14], they are incomplete in the sense that they focus only on a part of constraints and ignore the constraints for XML. Besides that, the resulted schema still contains data redundancies. Data redundancies are usually due to some form of dependencies among the data, such as functional dependencies and multi-valued dependencies in relational databases. Traditional functional dependencies are not suited for XML data because of the structural difference between the two types of database. On the other hand, dependencies naturally exist among data, no matter what format the data is in.

In this paper we illustrate how in the presence of functional dependencies, data redundancy can be detected in XML documents, and how to produce redundancy free relational schema which based on the information given. At the same time, the constraints, as well as the content and the structure of XML will be preserved. As an example, consider the XML tree representation of a faculty document shown in Fig 1. Given this document and our understanding of its semantics, we may wish to state the following constraints:

C1: In the context of the whole document, each course is uniquely identified by CNO

C2: If two students with the same STUDNO, then they must have the same sname

C3: Each student will get their grade for every course they enrolled.

The first constraint is an example of an absolute key, where the key is defined over the whole document and the third is an example of relative key, where the key is define within same context, in the terminology of [3],[4]. The second constraint is an example of a functional dependency [2] and cannot be expressed as a key constraint. Even though algorithm proposed in [7] tried to map XML to relational schema in the presence of functional dependencies by using redundancy reducing strategies, but they fail to detect redundancy that cause by the redundant nodes as in courses and students. Also the student node by the value “Siti” was stored twice in the tree, this will cause data redundancies in the resulted schema. It is important to check for this redundancy before the mapping took place, without the knowledge the same data will store twice and this is what we are trying to avoid. With the information about constraint in the schema, we can generate a good relational database, and this became the basis of our works.
The strategy adopted in this paper, is to produce a relational design, which preserves structural and semantic constraints of the XML data while reduced redundancies. First we capture the structural of XML data by the DTD and generate the DTD schema, which is the formal description of XML. By using the constraint preserving algorithm, we remove redundant path. Finally, by mapping paths in XFDs to relational attributes we get a set of relational functional dependencies and produce a relational storage for the XML data which preserves the content and the structure information of the original XML document, removes redundancy as indicated by the XFDs, and enforced efficiently using relational primary key constraints. How to detect redundancies in general path have been introduced in [17]. But the objective of the paper is towards normalizing the XML document, which is different from what we are doing here. The author also exploit the use of XPath[6], where their definition of XFD has the capability of upward and downward navigation. But not all the constraint as expressed in XFD can be expressed in relational form due to their different structures. This motivates us to limit our studies to the semantic constraints in a downward path.

In relational databases, the normalization process reduces or eliminates data redundancies for generating a good relational database design. Similar to relational databases, updates in an XML structure may cause anomalies if the XML data is redundant. A schema (or data definition) language is used to specify structures and constraints for a model. We study the publication of relational data in XML documents, the propagation of its constraints and the associated decision problems. Constraints are of fundamental importance in databases, and are also important to many forms of hierarchically structured data including XML documents, particularly in the data mapping. In our algorithm, semantic information in keys and functional dependencies were used to guide the schema design. We ignored the ordered features provided by XML in our mapping algorithm. If the features are so important we can simply add another parameter to our schema to capture the ordered structures. We omit them, as it is not the focus of our study.

III. DEFINITIONS AND NOTATIONS

Before we move further, we will define the notations that we been used in the paper, which are similar with the one in [2] but with minor modification to suit our mapping strategies.

A. XML Tree

As well known, an XML document can be represented by a tree. We call elements that have sub-elements and/or attribute as a complex element and denote it as $E_1$. And element that only have a single value as a simple element and denote as $E_2$. Let $E_1$ and $E_2$ be disjoint sets of element names, $A$ be a set of attribute names, $E = E_1 \cup E_2$, and $E$ and $A$ be disjoint. Element names and attribute names are called labels.

An XML tree is defined to be $T = (V, lab, ele, att, val, root)$, where (1) $V$ is a set of nodes; (2) $Lab$ is a mapping $V \rightarrow E \cup A$ which assigns a label to each node in $V$; a node $v$ in $V$ is called a complex element node if $lab(v) \in E_1$, a simple element node if $lab(v) \in E_2$, and an attribute node of $lab(v) \in A$. (3) $Ele$ and $att$ are functions from the set of complex elements in $V$: for every $v \in V$, if $lab(v) \in E_1$, then ele($v$) is a set of element nodes, and att($v$) is a set of attribute nodes with distinct labels. (4) $val$ is a function that assigns a values to each attribute or simple element. (5) root is the unique root node labeled with complex element name r. (6) if $v' \in$ ele($v$) $\cup$ att($v$), then we call $v'$ a child of v. The parent-child relationships defined by ele and att will form a tree rooted at root.

B. XML DTD

DTD describes the structure of XML documents and are considered as the schemata for XML documents. A DTD schema is denoted by 6 tuple $= (E_1, E_2, A, P, R, r)$ where (1) $E_1 \subseteq E$ is a finite set of complex element names, (2) $E_2 \subseteq E$ is a finite set of simple element names, (3) $A \subseteq A$ is a finite set of attribute, disjoint from $E$, (4) $P$ is a mapping function from $E_1$ to element type definitions: $\forall \tau \in E_1$, $P(\tau)$ is a regular expression, $\alpha ::= \varepsilon | \tau' | \alpha \mid \alpha, \alpha | \alpha^*$, where $\varepsilon$ is the empty word, $\tau' \in E_1 \cup E_2$ and “|”, “,” “*”, denote union, concatenation, and the Kleene closure, respectively; (5) $R$ is a mapping function from $E_1$ to sets of attributes in $A$ (6) $r$ is the element type of the root, which is distinct from all other symbols. A path in $D$ is a string $l_1, \ldots, l_m$ where $l_i$ is in the alphabet of $P(r)$, $l_1$ is in the alphabet of $P(l_{i-1})$ for $i \in [2, m-1]$, $l_m$ is in the alphabet of $P(l_{m-1})$ or in $R(l_{m-1})$.

C. Paths in XML trees

The path language we adopt is a common fragment of XPath:
where ε is the empty path, l is a node label, “/” denotes concatenation of two path expressions (child in XPath), and “//” means descendant-or-self in XPath. A path P is a sequence of labels \( l_1/\ldots/l_n \). A path expression Q defines a set of paths, while “//” can match any path. We use \( p \in Q \) to denote that \( p \) is in the set of paths defined by Q. For example, \( //\text{course/students/student/name} \).

D. Value Equality And Node Identity

To reduce redundancy we need to compare the nodes in the tree. When comparing two nodes \( n_1 \) and \( n_2 \) in an XML tree \( T \), we need to define the equality between them. Obviously, if \( n_1 \) and \( n_2 \) are the same node (denoted \( n_1 = n_2 \)), they should be considered equal, but this kind of node equality is not sufficient because there are cases where two distinct nodes have equal values. So we need to define value equality between nodes. Since we consider the ordering of child elements insignificant, our definition of value equality is different from those published previously.

1. Let \( n_1 \) and \( n_2 \) be two nodes in \( T \). We say \( n_1 \) and \( n_2 \) are value equal, denoted \( n_1 = v n_2 \), if \( n_1 \) and \( n_2 \) are of the same label, and
2. \( n_1 \) and \( n_2 \) are both attribute nodes or simple element nodes, and the two nodes have the same value, or
3. \( n_1 \) and \( n_2 \) are both complex elements, and for every child node \( n_{1i} \) of \( n_1 \), there is a child node \( n_{2i} \) of \( n_2 \) such that \( n_{1i} = v n_{2i} \), and vice versa.

E. Functional Dependencies for XML

First, we should emphasize that semantic constraints [3],[13],[16] are not a part of XML specifications. They can be regarded as the extension of XML schema to make XML documents more significant. In this paper, we mainly discuss functional dependencies constraints. Functional dependencies (FDs) were introduced in the context of the relational data model by Codd in 1972. As in relational databases, functional dependencies for XML (XFDs) are used to describe the property that the values of some attributes of a tuple uniquely determine the values of other attributes of the tuple [1]. The difference lies in that attributes and tuples are basic units in relational databases, whereas in XML data, they must be defined using path expressions. We also show how to use this constraint to detect data redundancies in XML documents before mapping to relational. So the resulted relational schema is redundancy free and update anomaly can be avoided. Functional dependencies that we adopt is an expression of the form:

\[
(Q, [P_{x1}, P_{x2}, \ldots, P_{xn} \rightarrow P_y])
\]

where Q is the FD header path which is defined by an XPath expression from the root of the XML document, \( P_{xi} \) (\( 1 \leq i \leq n \)) is an LHS (Left-Hand-Side) entity type which consists of an element name with optional attributes(s), and \( P_y \) is an RHS (Right-Hand-Side) entity type which consists of an element name with an optional attribute name. An XML FD \( (Q, [P_{x1}, P_{x2}, \ldots, P_{xn} \rightarrow P_y]) \) specifies as follows: for any two subtrees identified by Q, if they agree on \( P_{xi} \), \( P_{x2}, \ldots, P_{xn} \), they must agree on \( P_y \), if it exists. From the first section, the constraints in the document can best be presented in our XML FD format and will become as input to our system.

FD1: \( //\text{course} (\text{CNO} \rightarrow \text{course}) \)
FD2: \( //\text{student} (\text{STUDNO} \rightarrow \text{student}) \)
FD3: \( //\text{course} (\text{CNO}, //\text{students/student}/\text{STUDNO} \rightarrow \text{grade}) \)

The constraint path can be achieved through definition of value equality and node equality. If the value of the path is equal then violation occurred.

IV. MAPPING XML DTD TO RELATIONAL IN THE PRESENCE OF FUNCTIONAL DEPENDENCIES

We now turn to the implication problem: Given a set of XFDs, what others can be inferred and how?

**Definition:** An XFD \( \varphi : X \rightarrow Y \) is logically implied by a set of functional dependencies \( F \), written \( F \models \varphi \), if and only if \( \varphi \) holds on every instance that satisfies all dependencies in \( F \), that is, \( \varphi \) hold whenever all XFDs in \( F \) hold.

This problem is typically addressed by finding a set of inference rules, e.g. Armstrong’s Axioms for functional dependencies in relational databases, and proved that they are sound and complete. Compared to the relational counterpart, however, the task of finding such a set of inference rules for XFDs is much more difficult. This is because XFDs are based on path expressions while relational FDs are defined on attribute names.

A. Constraint Preserving Mapping Algorithm

In this approach, we extend Armstrong’s Axioms (reflexivity, augmentation and transitivity) to
use path expressions instead of simple attributes. DTDs and XML Schema documents can be used to restrict the structure of XML documents. For simplicity we will focus on DTDs in this paper, but the ideas presented here also apply to any schema file including XML Schemas documents. Below is the DTD schema that conform to the diagram in Fig 1.

Figure 2. DTD file for faculty

The generated DTD schema defined in section 2 will be:

\[ E_1 = \{ \text{faculty, courses, course, students, student} \} \]
\[ E_2 = \{ \text{cname, sname, address, sname, address, grade} \} \]
\[ A = \{ \text{STUDNO, CNO} \} \]
\[ \text{P(faculty)} = \{ \text{courses} \} \]
\[ \text{P(courses)} = \{ \text{course}* \} \]
\[ \text{P(course)} = \{ \text{cname, students} \} \]
\[ \text{P(students)} = \{ \text{student}* \} \]
\[ \text{P(student)} = \{ \text{sname, grade} \} \]
\[ \text{P(sname)} = \text{P(grade)} = \text{P(grade)} = \text{P(cname)} = \text{S} \]
\[ \text{R(course)} = \{ \text{CNO} \} \]
\[ \text{R(student)} = \{ \text{STUDNO} \} \]
\[ \text{R(course)} = \{ \text{STUDNO} \} \]
\[ \text{R(courses)} = \text{R(students)} = \text{R(CNO)} = \text{R(STUDNO)} = \emptyset \]
\[ r = \{ \text{faculty} \} \]

The DTD structure that conforms to the XML document in Figure 1 is as below, where the symbol "*" denotes zero or many occurrence.

![DTD Structure](image)

To achieve optimization, a redundant node, which satisfies the following condition, needs to be removed:

- Indegree = 1
- Node cardinality = 1, which is a singleton element
- Node child cardinality = 1 and
- A complex element

Thru this step, nodes COURSES under node FACULTY and node STUDENTS under node COURSE will be removed. The mapping algorithm relies on an input set of XFDs and DTD file. We therefore proposed an infer function, which given an XDF \( \emptyset : X \rightarrow Y \) and a schema \( D \), determines whether or not \( \emptyset \) can be inferred from \( D \). The algorithm works as follows: Traverses D top-down starting from the root of D, \( P(e) = r \), and generates a set \( F \) of FDs that is a cover of \( F' \), i.e. a superset of \( F_m \). More specifically, at each \( e \in P(e) \) encountered, it expands \( F \) by including certain FDs propagated from \( \Sigma \). It then removes redundant FDs from \( F \) to produce a minimum cover \( F_m \). But in the presence of DTD information to find minimum covers should be much easier. First, we need to consider the relationship between elements. The relationships that may appear between one element \( e \) and its sub-element \( e_i \) in DTD are:

- "1:1" - one element \( e \) has one and only one sub-element \( e_i \).
- "1:N" - meaning that one element \( e \) has one or more sub-elements \( e_i \).
“N:M” – meaning that one element \( e \) can have at least one sub-element \( e_i \) and these sub-elements are likely to belong to one or more different parent elements.

“1: 0” – meaning that the sub-element possess an optional operator

“1:0 … N” – meaning that the sub-element is an element with star operator.

The XFDs will be read according to the syntax, the header, the determinant (LHS) and the determine (RHS). The rules of implication applied here, i.e., given certain XFD what other XFDs can be implied. We extend the standard Armstrong rules (Reflexivity, Augmentation and Transitivity). But in the existence of DTD, the process can be simplified. For every singleton element (we treat simple elements and attributes are the same), it is true to say that

\[
\text{Student/sno} \rightarrow \text{student/sno/S} \\
\text{Student/sname} \rightarrow \text{student/sname/S} \\
\text{Course/cno} \rightarrow \text{course/cno/S} \\
\text{Course/cname} \rightarrow \text{course/cname/S}
\]

The element S is to indicate the values that contain in every element. If two elements have the same values then, we consider them identical. The concept of value equality applied here. We are not using the forms of key provided in DTD, because the known limitation, the key in the form of XFDs will be input instead. And we make the following proposition.

Proposition: Every element has at least one key.

We assume that in every relation there exists a unique key, so that every relation can be distinguished. If two elements have exactly the same DTD expression and values, then we consider them as identical, and denote a key for on element e as e.key. Then the basic functional dependencies exist:

If the XFD is in the form of e.key -> e, where key is a unique sub-element of e then e.key is a key for the element.

Normally the LHS of the XFDs will become the key for the relation. Referring to this form, CNO and SNO are keyed for course and student nodes respectively. When considered e.key is a key for the relation, then this rule can be deduced:

**Proposition 1:**

If e.key -> e, then e.key will determine every \( e_i \in P(E_i) \cup R(E_i) \) by using Implication procedure.

The constraint preserving mapping algorithm works as follows:

1. Every complex element in \( E_i \) will be the root of the relations.
2. Map every e.key to the attribute of the elements
3. Consider the relationship between the elements, if exist M:N relationship then create a new elements.
4. To maintain the parent-child relationship, every child element node needs to refer to the parent node.

**Example**

If given \( /\text{student/sno/S} \rightarrow \text{student} \) can we implied that \( \text{student/sno/S} \rightarrow \text{student/sname/S} \)?

Since each STUDENT has exactly one SNAME element as a child (1:1 relationship) and nodes have unique identifiers, then it is true to say that

\[
/\text{student} \rightarrow \text{student/sname} \text{ then using singleton element, this XFDs is trivially satisfied} \\
/\text{student/sname} \rightarrow \text{student/sname/S}
\]

Finally based on transitivity, it follows

\[
/\text{student/sno/S} \rightarrow /\text{student/sname/S} \\
/\text{student} \rightarrow /\text{student/sname}, \text{ then} \\
/\text{student/sno/S} \rightarrow /\text{student/sname/S} \text{ but,} \\
/\text{student/sname} \rightarrow /\text{student/sname/S} , \text{ therefore by transitivity,} \\
/\text{student/sno/S} \rightarrow /\text{student/sname/S}
\]

This will generate the minimum covers for the relation rules. The elements that are in the same set or rules will group into the same classes; we called this as Equivalence class. At the end of this step we will get (Rule (R_1), … , Rule (R_n)) that will form the schema relation. From the example above we called \( E_{new} \) as course-student node. Since the exist other element (course-student node) associate with GRADE then drop this node from STUDENT. Then we have

\[
/\text{course/cno/S} , /\text{student/sno/S} \rightarrow \text{course-student course-student} \rightarrow \text{course-student/grade/S} \\
\text{by transitivity} \\
/\text{course/cno/S} , /\text{student/sno/S} \rightarrow \text{course-student/grade/S}
\]

Based on the basic Armstrong inference rules, the following can be deduced.

i) Two complex elements \( E_i, E_j \) where \( E_i, E_j \in E_1 \), \( E_i \) has a 1:N relationship and \( E_i \) is a prefix of \( E_j \), then 

\[
E_i,\text{key} \rightarrow E_i, \text{ then } E_i,\text{key} \rightarrow E_j
\]
2) Two complex elements \(E_i, E_j\) where \(E_i, E_j \in E_1, E_i, E_j\) has a M:N relationship and \(E_i\) is a prefix of \(E_j\), then 
\(E_i\).key, \(E_j\).key \(\rightarrow\) \(E_{new}\), where \(E_{new}\) is a new node

Then a transformation \(\sigma\) from the above XML data to \(R\) could be specified as:
\[
\sigma = (\text{Rule(student)}, \text{Rule(course)}, \text{Rule(CS)})
\]

The set of equivalence classes according to the rules are
\[
\text{Rule(student)} = \{\text{SNO, sname, grade}\}
\]
\[
\text{Rule(course)} = \{\text{CNO, cname}\}
\]
\[
\text{Rule(CS)} = \{\text{CNO, SNO}\}
\]

Where the minimum covers for the relations are
\[
\text{student/sno/S} \rightarrow \text{student}
\]
\[
\text{student/sno/S} \rightarrow /\text{student/sname/S}
\]
\[
\text{course/cno/S} \rightarrow /\text{course/cnameS}
\]
\[
\text{CS/cno/S,course-student/sno/S} \rightarrow \text{CS/grade/S}
\]

At the end the following schema will be generated

### Student
- PRIMARY KEY SNO

### Course
- PRIMARY KEY CNO, SNO
- REFERENCES KEY SNO REFER TO student(SNO)
- REFERENCES KEY CNO REFER TO course(CNO)

### CS

<table>
<thead>
<tr>
<th>STUDNO</th>
<th>CNO</th>
<th>Grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>A100</td>
<td>TS2923</td>
<td>A</td>
</tr>
<tr>
<td>A200</td>
<td>TS2923</td>
<td>B</td>
</tr>
<tr>
<td>A100</td>
<td>TS1913</td>
<td>C</td>
</tr>
<tr>
<td>A300</td>
<td>TS1913</td>
<td>B</td>
</tr>
</tbody>
</table>

From [14] the resulted schema will be

### Course

<table>
<thead>
<tr>
<th>CID</th>
<th>CNO</th>
<th>Cname</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>TS2923</td>
<td>Networking</td>
</tr>
<tr>
<td>2</td>
<td>TS1913</td>
<td>Database</td>
</tr>
</tbody>
</table>

### Student

<table>
<thead>
<tr>
<th>SID</th>
<th>ParentID</th>
<th>ParentCode</th>
<th>STUDNO</th>
<th>Sname</th>
<th>Grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>TS2923</td>
<td>course</td>
<td>A100</td>
<td>Siti</td>
<td>A</td>
</tr>
<tr>
<td>2</td>
<td>TS2923</td>
<td>Course</td>
<td>A200</td>
<td>Amin</td>
<td>B</td>
</tr>
<tr>
<td>3</td>
<td>TS1913</td>
<td>Course</td>
<td>A100</td>
<td>Siti</td>
<td>C</td>
</tr>
<tr>
<td>4</td>
<td>TS1913</td>
<td>Course</td>
<td>A300</td>
<td>Mary</td>
<td>B</td>
</tr>
</tbody>
</table>

Even tough our algorithm will produce more tables if compared with Inlining [14] but we reduced the number of attributes in the relation. Some node ids (ID, parentID and parentCODE) generated by Hybrid Inlining are removed, this is possible as each instance node can be uniquely identified using key-based value information. In our method, which is based on traditional database theory, efficiently uses keys to join relations between parent and child. As a result, our method can produce resulting tables with less data redundancies. The generated schema in our algorithm is correct with respect to keys and functional dependencies. In fact our schema is in 3NF, as proved by the following proposition.

**Proposition.** Given a mapping \(\sigma\), an XML document \(T\) conforming to DTD \(D\), and a set \(\Sigma\) of XML FDs that generated from keys over \(D\), if \(T |\Sigma\), then each relation in \(\sigma(T)\) is in Third Normal Form (3NF).

**Proof.** To satisfy the Third Normal Form, we need to prove that each relation is in First and Second Normal Form. Since attributes of all relations in \(\sigma(T)\) are extracted from attributes or text nodes in a document, attributes of all relations are atomic. That is, all relations are in First Normal Form (1NF). Because of XML FDs are all in the set of \(\Sigma\), the semantics is in \(\Sigma\). All FDs on relational data are in the correspondence \(\Gamma\) of \(\Sigma\). We can conclude that each non key attribute in each relation is functionally dependent upon the primary key of the relation. That is, all relations are in Second Normal Form (2NF). According to the process of mapping, a relation is
created for each FD. Therefore, the relations created in step 2 are in 3NF. Additionally, the relations created in other steps used FD to describe the property that the values of some attributes of a tuple (keys) uniquely determine the values of other attributes of the tuple and the attributes that are not dependent upon the primary key have been eliminated. That is, these relations are in 3NF. So, all the relations $\sigma(T)$ are in 3NF.

VI. CONCLUSION

We have investigated the problem of how to design a normalized relational schema for XML data and how to automate the instance mapping.

REFERENCE


We have developed a new approach which given functional dependencies and DTD, we can detect redundancy in XML document and used this information for mapping to relational which can reduced redundancy and at the same time preserve the constraints as expressed in functional dependencies. It can be efficiently operated, automated and eliminates unnecessary ID caused by Hybrid Inlining algorithm[11]. We also define a new definition of functional dependencies in multiple relations for the case of unordered features of XML. As an immediate task, we would like to find efficient algorithm for the implication problem for the functional dependencies defined above. Through this study we hope that it will give some contributions to the database community.


