XML 文档及其函数依赖到关系的映射*

王 庆[1], 周俊梅[2], 吴红伟[1], 萧建昌[1], 周傲英[1]

[1](复旦大学 计算机科学与工程系,上海 200433)

[2](中兴通讯 上海研究所,上海 200233)

Mapping XML Documents to Relations in the Presence of Functional Dependencies

WANG Qing[1], ZHOU Jun-Mei[2], WU Hong-Wei[1], XIAO Jian-Chang[1], ZHOU Ao-Ying[1]

[1](Department of Computer Science and Engineering, Fudan University, Shanghai 200433, China)

[2](Shanghai R&D Institute, ZTE Corporation, Shanghai 200233, China)

+ Corresponding author: Phn: 86-21-65643790 ext 802, Fax: 86-21-65643503, E-mail: qingwang@fudan.edu.cn; semantics@163.com

http://www.fudan.edu.cn

Received 2002-05-10; Accepted 2002-11-22

Wang Q, Zhou JM, Wu HW, Xiao JC, Zhou AY. Mapping XML documents to relations in the presence of functional dependencies. *Journal of Software*, 2003,14(7):1275~1281.

http://www.jos.org.cn/1000-9825/14/1275.htm

Abstract: Many methods for mapping XML to relations have been proposed without considering the semantics of XML data before. But the semantics is very important to design schemas for storage, optimize queries, and check update anomalies, etc. In the presence of XML FDs which are specified over DTDs, this paper presents a method based on the hybrid inlining to map XML to relations for storage. The constraints which are represented by FDs, as well as the content and the structure, are preserved at the same time. Much storage redundancy can be reduced through using this method. Furthermore, this paper proves that those relations mapped from XML are all in Third Normal Form (3NF).

Key words: semantic; constraint; mapping XML; XML FD

摘 要: 有许多文章提出了根据DTD将XML映射成关系的方法,但都没有考虑XML的语义,而语义信息对数据存储模式设计、查询优化、更新异常检查等来说是十分重要的,如果在DTD上指定了XML的函数依赖,在映

* Supported by the National Grand Fundamental Research 973 Program of China under Grant No.G1998030414 (国家重点基础研究发展规划(973))

WANG Qing was born in 1977. He is a Ph.D. candidate at the Fudan University. His research interests are XML data management and semistructural data. **ZHOU Jun-Mei** was born in 1976. She is a system engineer at the Shanghai R&D Institute, ZTE Corporation. Her research interests are wireless access network, wireless LAN and CDMA. **WU Hong-Wei** was born in 1979. He is a master student at the Fudan University. His research interests are XML data management and semistructural data. **XIAO Jian-Chang** was born in 1978. He is a master student at the Fudan University. His research interests are XML data management and semistructural data. **ZHOU Ao-Ying** was born in 1965. He is a professor and doctoral supervisor at the Fudan University. His current research areas are XML data management, data mining and P2P computing.

射到关系数据库中就需要将其考虑进去.基于 Hybrid Inlining 方法并考虑 XML 函数依赖,提出了一种既能保持 XML 文档的内容和结构信息,又能保持函数依赖信息的映射方法.通过这种方法可以减少存储冗余,同时证明了映射后的关系都满足第三范式.

关键词: 语义;约束;映射 XML;XML 的函数依赖

中图法分类号: TP311 文献标识码: A

Over the past six years, XML[1] has been popular as a format for data exchange on the Internet. Holding many advantages, it has been being used in many applications. Usually we exchange data in XML documents on the web but store and query[2] in relational databases in local disks. As we know, XML data is hierarchical while relational data is flat. Hence, to store XML data into relational databases is not trivial. Many papers[3,4] have presented solutions. They focus their attention on the content and the structure. But in some cases users are concerned with not only the content and the structure but the constraints. They need to map XML data to relations considering the constraints and then export relational data to publish in XML according to the constraints when necessary.

In response to the absence of XML semantics (e.g., keys, FDs, etc.), many proposals[5~9] besides XML-Schema[10] have proposed some notations about XML constraints. Among these proposals, FDs for XML[9] are important for us to capture the semantics of XML data. In relational data, FDs are a critical part of its semantics. They are useful in recognizing keys, normalizing to make a good design, preventing update anomaly, etc. FDs for XML are the counterpart of those for relational data. They must be taken advantage of in the process of mapping.

On this premise, to map XML to relations according to XML FDs is feasible. At the same time, the constraints, as well as the content and the structure, can be preserved.

To the best of our knowledge, work related to the relationship between the semantics of XML data and the semantics of relational data includes Refs.[11~13]. Reference [11] presents a method to map XML to relations preserving the constraints defined by keys and keyrefs. This method is based on the hybrid inlining method of Ref.[4]. The proposal of Ref.[12] presents a PTIME algorithm to determine the problem of deriving a FD from predefined XML keys. That is, the relation schemas are specified in a group of transformation rules which are written by human beings not automatically. On the basis of those rules, the algorithm can be used to decide whether a given FD on some schema can be derived from XML keys or not. Reference [13] presents an algorithm CPI to derive constraints from DTDs. The method used to map is also based on the hybrid inlining method.

In this paper, we present a method to map XML to relations according to XML FDs. The method preserves the constraints, the content, and the structure as mentioned above.

The rest of the paper is organized as follows: we introduce some notations in section 2 at first. And then we describe the method for mapping in detail and make some conclusions in section 3. In order to illustrate the method clearly, an example is taken in section 4. We summarize the entire paper in section 5.

1 Notations

Before we move to the description of the method for mapping, we introduce some notations described in Ref.[9] first in this section.

Definition 1. Let D be a DTD. *An XML functional dependency* φ *over the DTD* D is an expression of the form: $(Q,[P_{x1},P_{x2},...,P_{xn}\rightarrow P_y])$, where Q is the FD header path which is defined by a simple XPath[14] expression (i.e. without wild card and Kleene closure) from the root of the XML document, $P_{xi}(1\le i\le n)$ is an LHS (Left-Hand-Side) entity type which consists of an element name with optional attribute(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 $(Q,[P_{x1},P_{x2},...,P_{xn}\rightarrow P_y])$ specifies as follows: for any two subtrees identified by Q, if they agree on

$P_{x1}, P_{x2}, ..., P_{xn}$, they must agree on P_y, if it exists. If it is the case to an XML document T, we say that the document satisfies the FD, denoted by $T \models \varphi$. Moreover, for any FD φ in a set Σ of FDs, if $T \models \varphi$, we say the document satisfies the set Σ, denoted by $T \models \Sigma$.

From the definition of a FD, the value of any LHS entity (or any RHS entity) is of a simple type. If an LHS entity is an element name without any attribute, then the value of the entity is equivalent to that of the text child (PCDATA) of the element. Otherwise, if the entity consists of an element followed by attribute(s), the value of the entity is the value of the attribute(s). And the LHS entities must exist. On the other hand, if an RHS entity is an element name without an attribute, its value is equivalent to that of the text child of the element, and there must be no other subtree under the element. Otherwise, its value is equivalent to that of the attribute. For example, in the following FD FD1, the value of *project.name* is equal to the value of the attribute *name* while the value of *contractid* is equal to the PCDATA under element *contractid*.

As we know, ID can identify a node in an XML document globally and uniquely. If we regard ID (or IDREF) in DTDs as an attribute, FDs for XML are more general.

For example, suppose the DTD is shown in Fig.1 and the DTD graph representing it is shown in Fig.2, where the graph is constructed by using the method in Ref.[4]. We can specify the FDs are:

- FD1=(/*PSJ*,[*project.name*,*supplier.name*->*contractid*])
  for any two *PSJ*s, if any two *project*s agree on the *name* and any two *supplier*s of the *project*s also agree on the *name*, then the *contractid*s are the same.

```
<!ELEMENT  PSJ (project*)>
<!ELEMENT  project (date, supplier*)>
<!ATTLIST     project name CDATA #REQUIRED>
<!ELEMENT  supplier (state, contractid, city, part*)>
<!ATTLIST     supplier name CDATA #REQUIRED>
<!ELEMENT  part (name*, color?, weight?, price, quantity)>
<!ATTLIST     part pno CDATA #REQUIRED>

(PCDATA is omitted here.)
```

Fig.1 A DTD

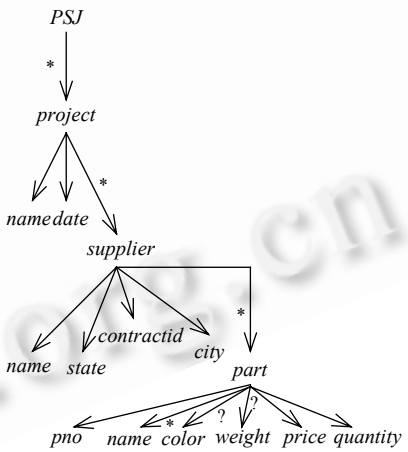

Fig.2 A DTD graph

- FD2=(/*PSJ*/*project*,[*supplier.name*,*part.pno*->*price*])
  for any two *project*s, if any two *supplier*s of theirs agree on the *name* and any two *part*s that the *supplier*s supply agree on the *pno*, the *price*s must be the same.
- FD3=(/*PSJ*,[*project.name*,*supplier.name*,*part.pno*->*quantity*])
  for any two *PSJ*s, if any two *project*s in them agree on the *name*, any two *supplier*s of the *project*s agree on the *name*, and any two *part*s that the *supplier*s supply agree on the *pno*, then the *quantitie*s are the same.
- FD4=(/*PSJ*/*project*/*supplier*,[*city*->*state*])
  for any two *supplier*s, if they agree on the *city* element, they must agree on the *state* element.

As far as relational data is concerned, the above FDs actually correspond to

(FD1')*PSJ.project.name*,*PSJ.project.supplier.name*->*PSJ.project.supplier.contractid*,

(FD2')*PSJ.project.supplier.name*,*PSJ.project.supplier.part.pno->PSJ.project.supplier.part.price*,
(FD3')*PSJ.project.name*,*PSJ.project.supplier.name*,*PSJ.project.supplier.part.pno->PSJ.project.supplier.part.quantity*,
(FD4')*PSJ.project.supplier.city->PSJ.project.supplier.state*,
respectively. Hence, if the set of XML FDs is denoted by Σ, the corresponding set of FDs on relational data can be denoted by Γ.

In addition, it is worth mentioning that keys can be specified by FDs. For example, each ID can be regarded as a key of some node although it is too restricted. We have FDs just like: (/*PSJ*, [*project.#id->date*]), (/*PSJ*/*project*, [*supplier.#id ->city*]), etc. (*#id* denotes the ID attribute.)

There are some types of FDs for XML classified by the relationships between the head path, the LHS entities, and the RHS entity. Two important and meaningful types are flat FDs and well-structured FDs.

Definition 2. Consider the DTD:

<!ELEMENT $H_1(H_2*)$>
…
<!ELEMENT $H_m(E_1*)$>
<!ELEMENT $E_1(E_2*)$>
…
<!ELEMENT $E_x(E_y)$>.

An XML FD $(Q,[P_{x1},P_{x2},...,P_{xn}\rightarrow P_y])$ is called *well-structured* where $Q=/H_1/.../H_m$, $P_{xi}(1\le i\le n)$ consists of an element in the set $\{E_1,...,E_x\}$ ($E_1,...,E_x$ are distinct.) and its optional attribute(s), and P_y consists of E_y and an optional attribute, if the following conditions are satisfied:

(1) the ordered XML elements in Q, $P_{x1},P_{x2},...,P_{xn}$, and P_y form a lineage; (A set L is called a *lineage* iff there exists a node N in the set L such that all other nodes are ancestors of N, and for each node M in L, if one of ancestors of M is in L, the parent of M is also in L.)

(2) the LHS entity types do not contain any redundant LHS entity type.

From the definition we can see that FD1, FD2 and FD3 are all well-structured.

Definition 3. Consider the DTD:

<!ELEMENT $H_1(H_2*)$>
…
<!ELEMENT $H_{m-1}(H_m*)$>
<!ELEMENT $H_m(E_1,\ldots,E_x,E_y)$>.

An XML FD $(Q,[P_{x1},P_{x2},...,P_{xn}\rightarrow P_y])$ is called *flat* where $Q=/H_1/.../H_m$, $P_{xi}(1\le i\le n)$ consists of an element in the set $\{E_1,...,E_x\}$ ($E_1,...,E_x$ are distinct.) and its optional attribute(s), and P_y consists of E_y and an optional attribute, if $P_{x1},P_{x2},...,P_{xn}$ do not contain any redundant LHS entity type.

FD4 is flat according to the definition. Because well-structured FDs and flat FDs are meaningful and others may make things ambiguous and unclear, we consider these two types of FDs only in this paper.

2 Mapping XML to Relations

Like that of relational data, the semantics of XML data is represented by XML FDs. We describe the method for mapping XML to relations with FDs in this section. That is, if all FDs are specified by users, the FDs, the content, and the structure are preserved in the process of mapping. This method is also based on the hybrid inlining method. Certainly, it can be generalized to cooperate with the shared inlining method, the basic inlining method, and

others. The method for mapping is given as follows:

(1) construct a DTD graph to represent the DTD using the method in the paper of Ref.[4];

(2) for each $(Q,[P_{x1},P_{x2},...,P_{xn}\rightarrow P_y])$, create a relation whose attributes are extracted from $P_{x1},P_{x2},...,P_{xn},P_y$, and specify the attributes corresponding to $P_{x1},P_{x2},...,P_{xn}$ as the key. Then, mark the last edge of the path from the root to P_y;

(3) for some edges unmarked in the DTD graph, if they form a recursion, then select one (if one is via "*", it has a high priority), break the cycle, create a relation for the end element, and mark the edge;

(4) for each edge unmarked in the DTD graph, if it is via a "*", then create a relation for the end element. Then mark the edge;

(5) create a relation for the root of the DTD;

(6) for each edge unmarked, inline the end element into the start element repeatedly until there has already existed a relation for the start element;

(7) ensure each parent-child relationship has been recorded by a parent id (sometimes a parent code is needed) in the relation for the child.

Distinguished from the hybrid inlining method, it preserves the constraints by step 2. Through making a relation for each FD, the method is dependency-preserving. Secondly, there always exists an attribute in some relation to extract the value from attributes or text nodes in the XML document. The content of the XML document is preserved. Thirdly, step 7 ensures the structure has been stored by *#id* and *parentid*. That is, a large relation can be obtained by joining those relations with *#id* and *parentid*. And step 2 records the relationship between the large relations obtained after joining and the relations created in this step by using keys and foreign keys. To sum up, the whole document can be reconstructed by using the operator "join" losslessly. We explain the method by an example in the next section. Investigating the method, we can make two conclusions.

Proposition 1. Given a mapping σ, an XML document T conforming to DTD D, and a set Σ of XML FDs over D, $\sigma(T)\models\Gamma$ iff $T\models\Sigma$.

Proof sketch. For any XML FD $\varphi=(Q,[P_{x1},P_{x2},...,P_{xn}\rightarrow P_y])$ in Σ, there exists its correspondence $\phi=Y\rightarrow l$ in Γ where each attribute in Y is extracted from $P_{x1},P_{x2},...,P_{xn}$ and l is extracted from P_y, if the XML document T satisfies φ (i.e. two subtrees which are identified by Q and agree on $P_{x1},P_{x2},...,P_{xn}$ must agree on P_y), in $\sigma(T)$ there exists only one relation which contains Y and l at the same time according to step 2 such that if two tuples in the relation agree on Y, then they must agree on l. That is, $\sigma(T)\models\phi$. So, $\sigma(T)\models\Gamma$ if $T\models\Sigma$. On the contrary, given an instance $\sigma(T)$ of a relational schema, the XML document can be constructed by the reverse of σ. Therefore, vice versa.

Proposition 2. Given a mapping σ, an XML document T conforming to DTD D, and a set Σ of XML FDs over D, if $T\models\Sigma$, then each relation in $\sigma(T)$ is in Third Normal Form (3NF).

Proof sketch. Firstly, 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). Secondly, because XML FDs are all in the set Σ, the semantics is in Σ. All FDs on relational data are in the correspondence Γ of Σ. We can conclude that each nonkey 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 have the FDs which have the form (the primary key->some attribute) and the attributes that are not dependent upon the primary key have been eliminated. That is, these relations are in 3NF. So, all relations in $\sigma(T)$ are in 3NF.

3 A Case Study

For example, suppose the DTD that XML documents should conform to be given in Fig.1. We map it as follows:

(1) construct the DTD graph shown in Fig.2;

(2) as for FD1, a relation *FD*1 (*PSJ.project.name*,*PSJ.project.supplier.name*,*PSJ.project.supplier.contractid*) is created; as for FD2, a relation *FD*2 (*PSJ.project.supplier.name*,*PSJ.project.supplier.part.pno*, *PSJ.project.supplier.part.price*) is created; as for FD3, a relation *FD*3 (*PSJ.project.name*, *PSJ.project.supplier.name*,*PSJ.project.supplier.part.pno*,*PSJ.project.supplier.part.quantity*) is created; as for FD4, a relation *FD*4 (*PSJ.project.supplier.city*,*PSJ.project.supplier.state*) is created. Then the edge from *supplier* to *contractid*, the edge from *part* to *price*, the edge from *part* to *quantity*, and the edge from *supplier* to *state* are marked;

(3) there is no recursion in the graph. Therefore, do nothing in step 3;

(4) for the edge from *PSJ* to *project*, create a relation *project* (*#id*); for the edge from *project* to *supplier*, create a relation *supplier* (*#id*); for the edge from *supplier* to *part*, create a relation *part*(*#id*); for the edge from *part* to *name*, create a relation *name* (*#id*,*name*). The four edges are marked;

(5) create a relation *PSJ* (*#id*) for the root;

(6) inline *project.name* and *project.date* into the relation *project*; inline *supplier.name* and *supplier.city* into the relation *supplier*; inline *part.pno*, *part.color*, and *part.weight* into the relation *part*. Then the relations are *project* (*#id*,*project.name*,*project.date*), *supplier* (*#id*,*supplier.name*,*supplier.city*), and *part* (*#id*,*part.pno*, *part.color*,*part.weight*). The related edges are marked;

(7) ensure the parent-child relationships: add *parentid* into the relations *project*, *supplier*, *part*, *name*. And the parent code is not needed here.

Finally, the relations are as follows:

- *PSJ*(*#id*);
- *project*(*#id*,*parentid*,*project.name*,*project.date*);
- *supplier*(*#id*,*parentid*,*supplier.name*,*supplier.city*);
- *part*(*#id*,*parentid*,*part.pno*,*part.color*,*part.weight*);
- *name*(*#id*,*parentid*,*name*);
- *FD*1(*PSJ.project.name*,*PSJ.project.supplier.name*,*PSJ.project.supplier.contractid*);
- *FD*2(*PSJ.project.supplier.name*,*PSJ.project.supplier.part.pno*,*PSJ.project.supplier.part.price*);
- *FD*3(*PSJ.project.name*,*PSJ.project.supplier.name*,*PSJ.project.supplier.part.pno*,*PSJ.project.supplier.part.quantity*);
- *FD4*(*PSJ.project.supplier.city*,*PSJ.project.supplier.state*).

We can see that the relations not only satisfy the FDs without losing the content and the structure but also are all in 3NF.

4 Conclusion

XML FDs are defined just like those on relational data. In relational databases, a critical task of FDs is to decrease redundancy and prevent update anomaly further. Something like normalization or decomposition in relation databases can be dealt with in mapping. In hierarchical data, redundancy is a phenomena that cannot be avoided. By defining XML FDs, the information of redundancy can be specified. The relations which are created to store XML data need to avoid the problem as much as possible. In this paper, we present a method based on the

hybrid inlining method to deal with the problem. It maps XML to relations preserving the constraints, as well as the content and the structure. Whether some other meaningful types of FDs that are specified over DTDs can be found or not is a part of the future research work. Given those FDs, the method can be extended and generalized to be more flexible and more compatible.

References:

[1] Bray T, Paoli J, Sperberg-McQueen CM. Extensible markup language (XML) 1.0. W3C Recommendation, 1998. http://www.w3.org/TR/REC-xml.

[2] Boag S, *et al*. XQuery 1.0: An XML query language. W3C Working Draft, 2002. http://www.w3.org/TR/xquery.

[3] Florescu D, Kossmann D. A performance evaluation of alternative mapping schemes for storing XML data in a relational database. Technical Report, No. 3684, INRIA, 1999.

[4] Shanmugasundaram J, *et al*. Relational databases for querying XML documents: Limitations and opportunities. In: Atkinson MP, *et al*., eds. Proceedings of the 25th International Conference on Very Large Data Bases. Edinburgh, Scotland: Morgan Kaufmann Publishers, 1999. 302~314.

[5] Buneman P, Davidson S, Fan W, Hara C, Tan W. Keys for XML. In: Proceedings of the 10th International Conference on World Wide Web. Hong Kong: ACM, 2001. 201~210.

[6] Buneman P, Fan W, Siméon J, Weinstein S. Constraints for semistructured data and XML. SIGMOD Record (ACM Special Interest Group on Management of Data), 2001,30(1):47~54.

[7] Fan W, Schwenzer P, Wu K. Keys with upward wildcards for XML. In: Mayr HC, *et al*., eds. Database and Expert Systems Applications, 12th International Conference. Lecture Notes in Computer Science 2113, Munich, Germany: Springer-Verlag, 2001. 657~667.

[8] Fan W, Siméon J. Integrity constraints for XML. In: Proceedings of the 19th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems. Dallas, Texas: ACM, 2000. 23~34.

[9] Lee ML, Ling TW, Low WL. Designing functional dependencies for XML. In: Jensen CS, *et al*., eds. Advances in Database Technology--EDBT 2002, 8th International Conference on Extending Database Technology. Lecture Notes in Computer Science 2287, Prague, Czech Republic:Springer-Verlag, 2002. 124~141.

[10] Thompson H, *et al*. XML schema. W3C Working Draft, 2001. http://www.w3.org/XML/Schema.

[11] Chen Y, Davidson S, Zheng Y. Constraint preserving XML storage in relations. Technical Report, MS-CIS-02-04, University of Pennsylvania, 2002.

[12] Davidson S, Fan W, Hara C. Propagating XML keys to relations. Technical Report, MS-CIS-01-33, University of Pennsylvania, 2001.

[13] Lee D, Chu WW. Constraints-Preserving transformation from XML document type definition to relational schema. In: Laender AHF, *et al*., eds. Conceptual Modeling--ER 2000, 19th International Conference on Conceptual Modeling. Lecture Notes in Computer Science 1920, Salt Lake City, Utah: Springer-Verlag, 2000. 323~338.

[14] Clark J, DeRose S. XML path language (XPath). W3C Working Draft, 1999. http://www.w3.org/TR/xpath.