

Available from: http://dx.doi.org/10.1007/3-540-36901-5_4
On Transformation to Redundancy Free XML Schema from Relational Database Schema

Chengfei Liu\(^1\), Jixue Liu\(^1\), Minyi Guo\(^2\)

\(^1\) Advanced Computing Research Centre, School of Computer and Information Science, University of South Australia, SA 5095, Australia
\{chengfei.liu, jixue.liu\}@unisa.edu.au

\(^2\) Department of Computer Software, The University of Aizu, Aizu-Wakamatsu City, Fukushima, 965-8580, Japan
minyi@u-aizu.ac.jp

Abstract. While XML is emerging as the universal format for publishing and exchanging data on the Web, most business data is still stored and maintained in relational database management systems. As a result, there is an increasing need to efficiently publish relational data as XML documents for Internet-based applications. One way to publish relational data is to provide virtual XML documents for relational data via an XML schema which is transformed from the underlying relational database schema, then users can access the relational database through the XML schema. In this paper, we discuss issues in transforming a relational database schema into corresponding schema in XML Schema. We aim to achieve high level of nesting while introducing no data redundancy for the transformed XML schema. In the paper, we first propose a basic transformation algorithm which introduces no data redundancy, then we improve the algorithm by exploring further nesting of the transformed XML schema.

1 Introduction

While XML [4,1] is emerging as the universal format for publishing and exchanging data on the Web, most business data is still stored and maintained in relational database systems. As a result, there is an increasing need to efficiently publish relational data as XML documents for Internet-based applications. One approach to publish relational data is to create XML views of the underlying relational data. Through the XML views, users may access the relational databases as though they were accessing XML documents. Once XML views are created over a relational database, queries in an XML query language like XML-QL [6] or XQuery [3] can be issued against these XML views for the purpose of accessing relational databases. SilkRoute [8] is one of the systems taking this approach. In SilkRoute, XML views of a relational database are defined using a relational to XML transformation language called RXL, and then XML-QL queries are issued against these views. The queries and views are combined together by a
query composer and the combined RXL queries are then translated into the corresponding SQL queries. XPERANTO [12, 5, 11] takes a similar approach but uses XQuery [3] for user queries.

Another approach [10] to publish relational data is to provide virtual XML documents for relational data via an XML schema which is transformed from the underlying relational database schema, then users can access the relational database through the XML schema. In this approach, there is a need to transform a relational database schema to the corresponding XML schema.

Currently, there are two options for defining an XML schema. One is the Data Type Definition (DTD) and the other is the XML Schema [7]. We choose XML Schema because Data Type Definition (DTD) has a number of limitations.

XML Schema offers great flexibility in modeling documents. Therefore, there exist many ways to map a relational database schema into a schema in XML Schema. For example, XViews [2] constructs graph based on primary key/foreign key relationship and generate candidate views by choosing node with either maximum in-degree or zero in-degree as root element. The candidate XML views generated achieve a high level of nesting but suffer considerable level of data redundancy. NeT [9] derives nested structures from flat relations by repeatedly applying the nest operator on tuples of each relation. The resulting nested structures may be useless because the derivation is not at the type level.

In this paper, we discuss issues in transforming a relational database schema into the corresponding schema in XML Schema. We aim at achieving a high level of nesting while introducing no data redundancy for the transformed XML schema. In the paper, we first propose a basic transformation algorithm which is redundancy free, then we improve the algorithm by exploring further nested structures.

The rest of the paper is organized as follows. In Section 2, we give a brief introduction of the XML Schema, especially the features which will be used in the schema transformation. In Section 3, we present the mapping rules of a basic transformation algorithm which converts a relational schema to the corresponding schema in XML Schema. The improvement of the basic algorithm is discussed in Section 4 with more nested structure explored. Section 5 concludes the paper.

2 XML Schema

XML Schema [7] is the W3C XML language for describing and constraining the content of XML documents. It is replacing the Data Type Definition (DTD) because DTD has a number of limitations, e.g., it is written in a non-XML syntax it has no support of namespaces; it only offers extremely limited data typing. XML Schema is a more comprehensive and rigorous method for defining content model of an XML document. The schema itself is an XML document, and so can be processed by the same tools that read the XML documents it describes. XML Schema supports rich built-in types and allows complex types built based on built-in types. It also supports key constraint and unique constraint which
are important to map relational database schema to XML schema. XML Schema supports two mechanisms to represent identity and reference: one is ID/IDREF which is also supported in DTD, the other is KEY/KEYREF which is not supported by DTD. ID and IDREF only apply to a single element/attribute while KEY and KEYREF can apply to multiple elements/attributes.

Shown below is a schema Company_XML in XML Schema for a company.

```xml
<xs:element name="Company_XML">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="Employee" minoccurs="0" maxoccurs="unbounded">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="name" type="xs:string"/>
            <xs:element name="city" type="xs:string"/>
            <xs:element name="salary" type="xs:int"/>
          </xs:sequence>
          <xs:attribute name="eno" type="xs:ID"/>
          <xs:attribute name="dno" type="xs:IDREF"/>
        </xs:complexType>
      </xs:element>
      <xs:element name="Dept" minoccurs="0" maxoccurs="unbounded">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="dname" type="xs:string"/>
            <xs:element name="city" type="xs:string"/>
            <xs:element name="DeptLoc" minoccurs="0" maxoccurs="unbounded">
              <xs:complexType>
                <xs:attribute name="dno" type="xs:IDREF"/>
                <xs:attribute name="city" type="xs:string"/>
              </xs:complexType>
              <xs:key name="PK_DeptLoc"/>
              <xs:selector xpath="Dept/DeptLoc/"/>
              <xs:field xpath="/dno"/>
              <xs:field xpath="/city"/>
            </xs:element>
          </xs:sequence>
          <xs:attribute name="dno" type="xs:ID"/>
          <xs:attribute name="mgrEno" type="xs:IDREF"/>
        </xs:complexType>
      </xs:element>
      <xs:element name="Project" minoccurs="0" maxoccurs="unbounded">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="pname" type="xs:string"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```
<xs:element name="city" type="xs:string"/>
</xs:sequence>
<xs:attribute name="pno" type="xs:ID"/>
<xs:attribute name="dno" type="xs:IDREF"/>
</xs:complexType>
</xs:element>
<xs:element name="WorksOn" minOccurs="0" maxOccurs="unbounded">
<xs:complexType>
<xs:element name="hours" type="xs:int"/>
<xs:attribute name="eno" type="xs:IDREF"/>
<xs:attribute name="pno" type="xs:IDREF"/>
<xs:key name="FK_WorksOn"/>
<xs:selector xpath="WorksOn/*"/>
<xs:field xpath="@eno"/>
<xs:field xpath="@pno"/>
</xs:element>
</xs:complexType>
</xs:element>
</xs:sequence>
</xs:complexType>
</xs:element>

Under the root element Company.XML, there are four set elements: Employee, Dept, Project and WorksOn. Each of these four elements is used to hold a set of instances conformed to the complexType defined within the element. In element Employee, there are three elements name, city and salary, and two attributes eno and dno. eno serves as the identity of the instances of Employee, dno intends to reference an instance of element Dept. In Dept, there are three elements dname, city and DeptLoc, and two attributes dno and mgrEno. dno serves as the identity of the instances of Dept while mgrEno intends to reference an instance of Employee. DeptLoc itself is also a set element which has two attributes dno and city, these two attributes together serve as the identity of the instances of DeptLoc using KEY element definition rather than ID type. In Project, there are two elements pname, city, and two attributes pno and dno. pno serves as the identity of the instances of Project while dno intends to reference an instance of Dept. In WorksOn, there are one element hours, and two attributes eno and pno. eno and pno together serve as the identity of the instances of WorksOn using KEY element definition. Individually, eno and pno intend to reference an instance of Employee and Project respectively.

3 Schema Transformation

The schema transformation from a relational database schema into the corresponding XML schema must take into account the integrity constraints associated with a relational schema, e.g., primary keys (PKs), foreign keys (FKs),
null/not-null, unique, etc. The null/non-null constraint can be easily represented by properly setting minOccurs of the transformed XML element for the relation attribute. The unique constraint can also be represented by the unique mechanism in XML Schema straightforwardly. In this paper, we focus on the mapping of PK/FK constraints. In the target XML schema, we aim to generate possible nested XML structure, however, we also aim to avoid introducing redundancy.

As introduced in the previous section, both ID/IDREF and KEY/KEYREF are supported by XML Schema. KEY/KEYREF can be used to represent keys with multiple attributes but ID/IDREF cannot. However, ID/IDREF supports the dereference function in path expressions in most XML query languages. Therefore, we will differentiate the single attribute primary/foreign keys from multi-attribute primary/foreign keys while transforming the relational database schema to XML schema and try to use ID/IDREF where possible because of dereference function support. We also classify a relation into the four categories based on different types of primary keys.

**Definition 3.1 regular relation**
A regular relation is a relation where the primary key contains no foreign keys.

**Definition 3.2 component relation**
A component relation is a relation where the primary key contains one foreign key which references its parent relation. The other part of the primary key serves as a local identifier under the parent relation.

The component relation is used to represent a component or a multi-valued attribute of its parent relation.

**Definition 3.3 supplementary relation**
A supplementary relation is a relation where the primary key as a whole is also a foreign key which references another relation.

The supplementary relation is used to supplement another relation or to represent a subclass for transforming a generalization hierarchy from a conceptual schema.

**Definition 3.4 association relation**
An association relation is a relation where the primary key contains more than one foreign keys, each of which references a participant relation of the association.

Based on above discussion and definitions, we give the set of mapping rules.

### 3.1 The Basic Algorithm

**Input:** A relational database schema \( Sch \) with primary key and foreign key definitions.

**Output:** A corresponding XML schema \( Sch_{XML} \) which is redundancy free.
Rule 1 For a relational database schema Sch, a root element named Sch/XML is created in the corresponding XML schema as follows.

```xml
<xs: element name = "Sch_XML">
    <xs: complexType>
        <xs: sequence>
            <!-- transformed relation schema of Sch -->
        </xs: sequence>
    </xs: complexType>
</xs: element>
```

Rule 2 For each regular or association relation R, the following element with the same name as the relation schema is created and then put under the root element.

```xml
<xs: element name = "R" minOccurs = "0" maxOccurs = "unbounded">
    <xs: complexType>
        <xs: sequence>
            <!-- the attributes of R -->
        </xs: sequence>
    </xs: complexType>
</xs: element>
```

Rule 3 For each component relation R_i, let its parent relation be R_2, then an element similar to Rule 2 and with the same name as the component relation is created and then placed as a child element of R_2.

Rule 4 For each supplementary relation R_i, let the relation which R_i references be R_2, then the following element with the same name as the supplementary relation schema is created and then placed as a child element of R_2. Notice, there is a difference between the transformed element of a component relation and the transformed element of a supplementary relation on maxOccurs.

```xml
<xs: element name = "R1" minOccurs = "0" maxOccurs = "1">
    <xs: complexType>
        <xs: sequence>
            <!-- the attributes of R1 -->
        </xs: sequence>
    </xs: complexType>
</xs: element>
```

Rule 5 For each single attribute primary key with the name PKA of regular relation R, an attribute of the element for R is created with ID data type as follows.

```xml
<xs: element name = "R" minOccurs = "0" maxOccurs = "unbounded">
    <xs: complexType>
```
<xs:attribute name = "PKA" type = "xs:ID"/> 
</xs:complexType>

Rule 6 For each multiple attribute primary key PK of a regular, a component or an association relation R, suppose the key attributes are PKA₁, ⋯, PKAₙ. An attribute of the element for R is created for each PKAᵢ (1 ≤ i ≤ n) with the corresponding data type. If R is a component relation and PKAᵢ is a single attribute foreign key contained in the primary key, then the data type of the created attribute is IDREF. After that a key element is defined with a selector to select the element for R and several fields to identify PKA₁, ⋯, PKAₙ. The key element can be defined inside or outside the element for R. The name of the element PK should be unique within the namespace.

<xref type = "xs" name = "R" minOccurs = "0" maxOccurs = "unbounded"/>

Rule 7 Ignore the mapping for primary key of each supplementary relation.

Rule 8 For each single attribute foreign key FKA of a relation R except one which is contained in the primary key of a component or supplementary relation, an attribute of the element for R is created with IDREF data type.

Rule 9 For each multiple attribute foreign key FK of a relation R except one which is contained in the primary key of a component or supplementary relation, suppose FK references PK of the referenced relation, and the foreign key attributes are FKA₁, ⋯, FKAₙ, an attribute of the element for R is created for each FKAᵢ (1 ≤ i ≤ n) with corresponding data type. Then a keyref element is defined with a selector to select the element for R and several fields to identify FKA₁, ⋯, FKAₙ. The keyref element can be defined either inside or outside
the element. The name of the element FK should be unique within the namespace and refer of the element is the name of the key element of the primary key which it references.

```xml
<xs: element name = "R" minOccurs = "0" maxOccurs = "unbounded">
  <xs: complexType>
    <xs: attribute name = "FKA1" type = "xs:FKA1_type"/>
    ... ... 
    <xs: attribute name = "FKAn" type = "xs:FKAn_type"/>
  </xs: complexType>
  <xs: keyref name = "FK" refer = "PK"/>
  <xs: selector xpath = "R/"/>
  <xs: field xpath = "@FKA1"/>
    ... ... 
  <xs: field xpath = "@FKAn"/>
</xs: keyref>
</xs: element>
```

**Rule 10** For each non-key attribute of a relation R, an element is created as a child element of R. The name of the element is the same as the attribute name.

### 3.2 Example and Discussion

Let us have a look at a relational database schema *Company* for a company. Primary keys are underlined while foreign keys are in italic font.

- Employee(eno, name, city, salary, dno)
- Dept(dno, dname, mgrEno)
- DeptLoc(dno, city)
- Project(pno, pname, city, dno)
- WorksOn(eno, pno, hours)

If the above schema is given as an input to the basic schema transformation algorithm, the schema in XML Schema *Company_XML* shown in Section 2 will be generated.

As XML allows nested structure, redundancy may be brought in when transforming flat relation structure to nested XML structure. For example, if we put element Dept under element Project, same department will be repeated in all projects in the department. However, if we put elements Dept and Project at the same level or put element Project under element Dept, there is no data redundancy introduced.

From the algorithm and the transformation example, we can see that the mapping rules bring no data redundancy provided the underlying relational schema is redundancy free. So we have the following.

**Theorem 3.1** If the relational database schema Sch is redundancy free, the XML schema Sch_XML generated by the basic transformation algorithm is also redundancy free.
This theorem is easy to prove. For a regular or an association relation \( R \), an element with the same name \( R \) is created under the root element, so the relation \( R \) in \( Sch \) is isomorphically transformed to an element in \( Sch_{XML} \). For a component relation \( R \), a sub-element with the same name \( R \) is created under its parent \( R_p \). Because of the foreign key constraint, we have the functional dependency \( PK_R \to PK_{R_p} \), i.e., there is a many to one relationship from \( R \) to \( R_p \) therefore it is impossible that a tuple of \( R \) is placed more than one time under different element of \( R_p \). Similar to a component relation, there is no redundancy introduced for a supplementary relation.

4 Exploring Nested Structures

As we can see, the basic transformation algorithm introduced above fails to explore all possible nested structures. For example, the \textit{Project} element can be moved under the \textit{Dept} element if every project belongs to a department. Nesting is important in XML schema because it allows navigation of path expressions to be processed efficiently. If we use IDREF instead, we may use system supported dereference function to get the referenced elements. In XML, the dereference function is expensive because ID and IDREF types are value based. If we use KEYREF, we have to put an explicit \textit{join} condition in an XML query to get the referenced elements. Therefore, we need to explore all possible nested structure by investigating the referential integrity constraints in the relational schema. For this purpose, we introduce a reference graph as follows:

\textbf{Definition 4.1 :} Given a relational database schema \( Sch = \{ R_1, \ldots, R_n \} \), a reference graph of the schema \( Sch \) is defined as a labeled directed graph \( RG = (V, E, L) \) where \( V \) is a finite set of vertices representing relation schema \( R_1, \ldots, R_n \) in \( Sch \); \( E \) is a finite set of arcs, if there is a foreign key defined in \( R_i \) which references \( R_j \), an arc \( e = \langle R_i, R_j \rangle \in E \); \( L \) is a set of labels for edges by applying a labeling function from \( E \) to the set of foreign keys.

The reference graph of the relational schema \textit{Company} is shown as in Figure 1. In the graph, we can see that the element of node \textit{DeptLoc} has been put under the element of node \textit{Dept} by the basic algorithm. The element of node \textit{Project} could be put under the element of node \textit{Dept} if the foreign key \textit{dn} is defined to NOT-NULL. This is because that node \textit{Project} only references node \textit{Dept} and a many to one relationship from \textit{Project} to \textit{Dept} can be derived from the foreign key constraint. In addition, the NOT-NULL foreign key means every project has to belong one department. As a result, one project can be put under one department and cannot be put twice under different departments in the XML document. In the graph, we also see a loop between \textit{Employee} and \textit{Dept}, what we can get from this is a many to many relationship between \textit{Employee} and \textit{Dept}. In fact, the foreign key \textit{mgrEno} of \textit{Dept} reflects a one to one relationship from \textit{Dept} to \textit{Employee}. Unfortunately, this semantic got lost when we map the conceptual model (e.g., E-R or UML) to the logical model. With the guide of a designer, the element of the node \textit{Employee} can also put under the element
of the node `Dept` if the foreign key `dno` is defined to NOT-NULL. The node `Works On` references two nodes `Employee` and `Project`. The element of `Works On` can be put under either `Employee` and `Project` if the corresponding foreign key is NOT-NULL. However, which node to choose to put under all depends on which path will be used often in queries. We will also leave this decision to be chosen by a designer.

Based on the above discussion, we can at least improve the basic algorithm by the following.

**Theorem 4.1** If a relation $R_1$ has only one foreign key $FK$ which references to another relation $R_2$ and $FK$ is defined as NOT-NULL, then we can move the element for $R_1$ to under the element for $R_2$ without introducing data redundancy.

The proof of this theorem has already explained by the relationship between `Project` and `Dept` in Figure 1. The NOT-NULL $FK$ suggests a many to exact one relationship from $R_1$ to $R_2$. Therefore, for each instance of $R_1$, it is put only once under exactly one instance of $R_2$, no redundancy will be introduced.

If we apply this theorem to the transformed XML schema `Company.XML`, the element for `Project` will be moved to under `Dept` as follows, the attribute `dno` with IDREF type can be removed from the `Project` element.

```xml
<xs:element name="Dept" minOccurs="0" maxOccurs="unbounded">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="dno" type="xs:string"/>
      <xs:element name="ename" type="xs:string"/>
      <xs:element name="deptno" type="xs:string"/>
      <xs:element name="EmpLoc" minOccurs="0" maxOccurs="unbounded">
        <xs:complexType>
          <xs:attribute name="dno" type="xs:IDREF"/>
          <xs:attribute name="ename" type="xs:string"/>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
  <xs:key name="FK_DeptLoc"/>
</xs:element>
```
<xs:selector xpath="Dept/DeptLoc/">
 <xs:field xpath="$dno"/>
 <xs:field xpath="$city"/>
 </xs:key>
</xs:element>

<xs:element name="Project" minOccurs="0" maxOccurs="unbounded">
 <xs:complexType>
 <xs:sequence>
 <xs:element name="pname" type="xs:string"/>
 <xs:element name="city" type="xs:string"/>
 </xs:sequence>
 <xs:attribute name="pno" type="xs:ID"/>
 </xs:complexType>
</xs:element>
<xs:sequence>
 <xs:attribute name="dno" type="xs:ID"/>
 <xs:attribute name="mgrEno" type="xs:IDREF"/>
</xs:complexType>
</xs:element>

5 Conclusion and Future Work

This paper addressed the issues in mapping relational database schema to XML schema. To generate high quality XML schema from relational schema, a schema transformation algorithm should have two features: exploring all possible nested structures and preserving data redundancy free property of a relational schema. The work in XViews and NeT satisfy the former only. The algorithm presented in this paper satisfies both features. It brings no data redundancy while achieving high level of nesting.

We believe that the proposed algorithm is effective and practical. In the future, we will investigate schema transformation from non-normalized relational database schema to redundancy free XML schema.

Acknowledgements

This work was supported in part by the visiting research program award R-5-2 from the University of Aizu, Japan.

References


