Exporting Relational Data into a Native XML Store

Jaroslav Pokorny and Jakub Reschke

Department of Software Engineering, Faculty of Mathematics and Physics,
Charles University, 118 00 Praha, Czech Republic,
pokorny@ksi.ms.mff.cuni.cz

Introduction

XML, see Bray et al. (2004), has thoroughly established itself as the standard format for data interchange between heterogeneous systems. Its significant role is also in the vision of so-called Semantic web specified in W3C (2001) where it contributes to low-level representation of information. As most of the enterprise’s data is stored in relational database systems, conversion problems of relational data into XML should be studied in details. The publishing scenario can be twofold: relational data has to be visible as XML data independently from how the data is stored or, and it is more important today, relational data has to reside in a native XML store. The former scenario is simple because the resulting structure mirrors the original relational tables’ flat structure. The latter requires more advanced techniques, for example preserving at least a part of integrity constraints applied in the original relational database or ensuring non-redundant storing the database as XML data. Generally, it means to convert relational database schemes into schemes expressed in XML Schema language (Fallside and Walmsley 2004).

Commercial RDBMS partially support these facilities mainly through the XML features of the standard SQL:2003, see ISO (2003), particularly its part SQL/XML (XML-Related Specifications) given in ISO (2004). SQL/XML (hereinafter called “the standard”) has been embraced by most major relational database vendors. Anyhow, there are many previously released proprietal solutions in their RDBMSs as well.

The standard treats not only XML publishing functions, but also mapping rules for transformations from the extended relational data model (RDM) to XML Schema. By “extended” we mean mainly a possibility to nest relations, which distinguishes the model from the original flat relations. According to usual terminology we call a description of XML data expressed in XML Schema as *XSD* (*XML Schema Definition*). This paper offers an algorithm doing this task and respecting recommendations of the standard. It covers also some integrity constrains. A prototype implementation by Reschke (2005) enables to generate an XSD, a conversion of relational data into an XML document, and its validation against the XSD.

The paper starts with a brief introduction to XML Schema and a relevant part of the standard. After this we describe some existing algorithms, particularly NeReFT developed by Liu et al. (2004). Then we introduce a new algorithm, called XMLConversion here, which is based on NeReFT. XMLConversion provides a number of improvements and keeps the rules proposed in the standard. We also mention shortly its implementation and conclude the paper.

XML Schema

We mention only the features of XML Schema that are important for the transformation algorithms. Figure 1 shows the hierarchy of data types used in the language.

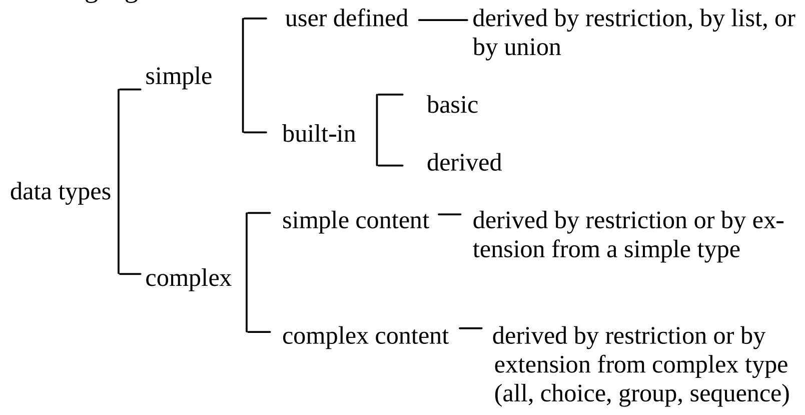
data types
simple
complex
user defined
derived by restriction, by list, or by union
built-in
basic
derived
simple content
derived by restriction or by extension from a simple type
complex content
derived by restriction or by extension from complex type (all, choice, group, sequence)

Fig. 1. Taxonomy of data types in XML Schema.

As usually, simple data types include Boolean, string, float, etc. as well as various time and date types. Among types derived from built-in string types we can find e.g. `ID`, `IDREF`, and `IDREFS`. For expressing XML structures of relational data, the complex types are of great importance.

Clauses `unique`, `key,` a `keyref` are useful for expressing identities. Each identity constraint is expressed by an expression in XPath, see Clark and DeRose (1999). We can express referential integrity similarly to relational databases with these clauses. A `unique` element contains exactly one `selector` subelement and at lest one `field` subelement. The `se-`

`lector` determines a set of items (elements or attributes) inside of which the items determined by element(s) `field` must be unique. By `key` element we denote relational attributes or their combination whose resulted value must be in a given area unique and always defined. The value of `keyref` attribute must be a value of a `key` or `unique` element.

SQL:2003

Recently INCITS, ANSI, and ISO have added XML publishing functions to SQL:2003. We refer here to the version FCD (Final Committee Draft). It is expected that a movement from FCD to DIS (Draft Information Standard) should bring no significant changes influencing the approach used in our transformation algorithm.

New Data Types

Comparing to the version SQL:1999, focused mainly on the object-relational data model, the new standard contains the following extensions:

- data types `BIGINT`, `MULTISET`, and `XML`,
- functions for publishing XML data,
- mapping rules for description of XSD schemes and valid transformed relational data that conforms these schemes.

For our paper we will consider as relevant only the data structures of non-XML data, i.e. typed tables together with nesting via `ARRAY` and `MULTISET`, and omit the repertoire of associated predicates and operations usable in SQL queries.

XML Publishing Functions

The standard introduces a set of functions applicable directly in the `SELECT` statement which make it possible to generate data of `XML` type.

- `XMLELEMENT` – creates an XML element of given name with optional specification of namespaces (parameter `XMLNAMESPACES`) and attributes (parameter `ATTRIBUTES`).
- `XMLATTRIBUTES -` lists XML attributes to be placed in the XML element created by enclosing call of `XMLELEMENT`.

- `XMLFOREST` – is a shortcut function for generating a forest of elements with only columnar content. It takes as its arguments a set of column names or aliased column names.
- `XMLCONCAT` – based on a list of independently constructed XML expressions (for example via `XMLELEMENT`) constructs one value as a concatenation of values of these expressions.
- `XMLAGG` – aggregates a set of rows in the result set, emitting the XML that is specified as the `XMLAgg` function's argument for each row that is processed. It enables to express relationships with cardinality 1:N in XML.

For example, the statement

```
SELECT e.id,XMLELEMENT(NAME "Employee",
  XMLELEMENT(NAME "Name",e.first_n||''||e.last_n),
  XMLELEMENT(NAME "Subordinates",
    (SELECT COUNT (*) FROM Subordinates s
         WHERE s.chief = e.id ) ) AS Description
         FROM Employees e WHERE ...
```

can generate the table

```
ID                 Description
154                <Employee>
                      <Name>John Smith</Name>
                      <Subordinates>3</Subordinates>
                   </Employee>
```

Mapping Rules

The standard introduces mapping rules for tables, schemes, and catalogues to XML. Mapping rules include also coding data, `NULL` value representation, etc. They enable to express also simple integrity constraints allowing to describe better the value set for a given simple or derived simple type.

Mapping tables to XML documents. The standard defines how to map tables to an XML document. The source can be a single table, all tables, all tables in a schema, or all tables in a catalogue. As a result of the mapping we obtain two documents, the first one contains data from tables and the second one the associated XSD. The data document is valid against the XSD. Although there is more possibilities for a table representation in XML, the standard supports the one in which values of table columns are mapped to subelements of a `<row>` element. Notice that with subelements we fix an order of columns, which is not required in RDM. Modelling columns by XML attributes would determine no order.

A database with more than one table can be mapped into XML by two methods. Figure 2 represents two tables, `Reader(R1, R2)` and `Books(B1 ,B2)`, from a relational schema `Library`.

Which mapping is chosen depends fully on a user. Consequently, any tool doing these transformations should be interactive or at least parameterizable.

```
<Library>                   <Library>
<Readers>                   <Readers>
<row>                         <R1>1</R1>
  <R1>1</R1>                  <R2>Kate</R2>
  <R2>Kate</R2>             </Readers>
</row>                      <Readers>
                       ...    <R1>2</R1>
</Readers>                    <R2>John</R2>
<Books>                     </Readers>
<row>                       ...
  <B1>1</B1>                <Books>
  <B2>Wings</B2>              <B1>1</B1>
</row>                        <B2>Wings</B2>
         ...                </Books>
</Books>                    ...
</Library>                    </Library>
```

Fig. 2. Two possibilities how to represent relations in XML Schema.

Mapping NULL values. A user has two flavours in the standard how to map `NULL` values. In the first one, the attribute `xsi:nil="true"` indicates that the column value is `NULL`. The person with `NULL` value of `Birth_date` looks in XML as

```
<row>
  <Id>1</Id>
  <First_n>John</First_n>
  <Last_n>Smith</Last_n>
  <Birth_date xsi:nil="true"></Birth_date>
  <Degree xsi:nil="true"/>
</row>
```

In the second case, the relational representation of columns with `NULL` value is omitted.

Mapping data types. The standard provides rules for transformation of particular types. For example, SQL types based on strings are mapped on XML Schema type `xsd:string` with subelements `xsd:length` or `xsd:maxLength` specifying the string length and maximal length, respectively. The “xsd” namespace prefix is used to indicate the XML Schema namespace.

For example, the SQL type CHAR restricted to 25 symbols has the following representation:

```
<xsd:simpleType name="CHAR_25">
  <xsd:restriction base="xsd:string">
  <xsd:length value="25"/>
  </xsd:restriction>
</xsd:simpleType>
```

Unfortunately, SMALLINT, INTEGER, and BIGINT are mapped in the standard to types with the same name. This leads to inconsistencies in situations when we have various constraints on values, e.g. of SMALLINT. We correctly resolve this problem by renaming new types.

SQL ARRAY a MULTISET types are mapped to complex types. The basic data type, whose values are stored into an array/multiset, is mapped to a simple type. For example,

```
<xsd:complexType name="Array_5.Array_5.VARCH_10">
  <xsd:sequence>
    <xsd:element name="element" minOccurs="0"
         maxOccurs="25" nillable="true"
         type="VARCHAR_10"/>
    </xsd:element>
  </xsd:sequence>
</xsd:complexType>
```

maps a two-dimensional array of 5×5 strings with maximum length equal to 10.

SQL XML type is mapped to a complex type. The structure of values stored in the XML type is not processed. An XML value is considered as a plain text without a meaning. To forbid processing such value during a validation, it is necessary to use the attribute processContents with the value "skip".

```
<xsd:complexType name="XML" mixed="true">
  <xsd:sequence>
    <xsd:any name="element" minOccurs="0"
      maxOccurs="unbounded"
      processContents="skip"/>
    </xsd:sequence>
</xsd:complexType>
```

Generating schema in the language XML Schema. There are many possibilities how to generate XSD describing relational database schemes composed from particular table schemes. The standard considers generating for each type and a table, own global type. These types are then used in more complex definitions.

Related Works

We will overview shortly existing algorithms for conversion of the relational database schema to the languages DTD and XML Schema. We suppose a relational database schema $\boldsymbol{R} = (R_1,\ldots,R_K, K\geq 1; \text{IC})$, where R_i are relation schemes and IC is a set of integrity constraints. By R^* we mean a relation associated to R.

Target Language: DTD. In FT (Flat Translation) Lee et al. (2001) transform relation schemes from $\boldsymbol{R}$ to elements of an XSD and attributes of relation schemes to attributes or elements of the XSD. A usage of attributes or elements depends on a user, since the algorithm can work in both modes. IC is not considered in the algorithm. The approach also does not exploit non-flat features of XML model, e.g. regular expressions specifying a number of element occurrences, and hierarchical nesting of elements.

NeT (*Nesting-based Translation*) algorithm presented in the same paper tries to overcome drawbacks of FT using an operator *Nest*. The main idea is to describe a structure of nested elements by Kleene operators. Unlike NeT the CoT algorithm (*Constraints-based Translation*) by Lee et al. (2002) considers also a referential integrity.

Target Language: XML Schema. The ConvRel (*Relationship Conversion*) algorithm by Duta (2003) considers referential integrity and constraints expressed by `UNIQUE` and `NULL`. The author classifies cardinalities 1:1, 1:N and M:N between rows of associated relations to determine a nesting of elements. A key problem is to determine which relation will create the outer element and which one its subelement.

ConvRel considers only simple links between two relations, while the relations are actually connected by more complex links. Each table can refer to or is referenced from more other tables. In the algorithm Conv2XML Duta considers links among three tables.

NeReFT Algorithm. Li et al. (2003) approach the problem with rules that are applicable for relation schemes of various types. These rules are driven by referential integrity associations between schemes. NeReFT (Nested Redundancy Free Translation) works also simply with `NULL/NOT NULL` constraints by properly setting `minOccurs` attribute in XML elements. `UNIQUE` constraints have a straightforward representation with `unique` mechanism in XML Schema. The strategy of NeReFT is to reach nested XML structures and minimum redundancy in XML data. By redundancy we mean here repeating data in the resulting XML data document[1].

Thus, IC include primary keys, referential integrities, `NULL/NOT NULL`, and `UNIQUE` constraints. As an output we obtain an XSD describing non-redundant XML documents.

Suppose a schema $R(K_1,\ldots,K_n,A_{n+1},\ldots,A_{n+m})$, where $K_1,\ldots,K_n$ compose the primary key (PK_R) of R. For a referential integrity between relations R and P, where a foreign key (FK) from R references to P, we denote this fact as $FK_R \rightarrow P$. Referential integrity naturally induces a digraph $G_{\mathbf{R}}$. Schemes R and P are called *child* and *parent*, respectively. Each R from $\boldsymbol{R}$ is classified into one of four categories dependent of PK_R:

- *regular* – no FK occurs among $K_1,\ldots,K_n$.
- *component* – there is K_i which references to P. The rest of PK_R serves to local identification of rows under one K_i value.
- *supplementary* – the PK_R is also an FK_R, $FK_R \rightarrow P$, for a P.
- *association* – the PK_R contains more FK_Rs.

In practice, regular and component relations correspond to entity and weak entity types, respectively. Supplementary relations correspond often to members of an ISA hierarchy or a vertical decomposition of relation. Finally, association relations are transformed relationship types.

The algorithm core:

(1) For a schema $\boldsymbol{R}$ the algorithm creates a root element in the target XSD.

(2) For a regular or an association relation R, it creates an element with the name R and puts it under the root element. The created element may be moved down later depending on some constraints.

(3) For a component or a supplementary R, an element is created and placed as a child element of the element for its parent relation. The representations of both relation types differ only in the value of `maxOccurs` attribute.

(4) For each single attribute PK of a regular R, an attribute of the element for R is created with `ID` data type. For each multiple attribute PK of a regular, a component or an association R, an attribute of the element for R is created for each PK attribute with its corresponding data type; a key element is defined with a `selector` to select the element for R and several `fields` to identify all PK attributes.

(5) For each FK of a relation R, where FK $\not\subset$ PK of a component or a supplementary relation, if it is a single attribute FK, an attribute of the element for R is created with `IDREF` data type; otherwise, an attribute

[1] Problems of redundancy are discussed in details by Vincent et al. (2004).

is created for each FK attribute with its corresponding data type, a `keyref` element is defined with a `selector` to select the element for *R* and several `field`s to identify FK attributes.

(6) For a non-key attribute of *R*, an element is created under the element for *R*.

(7) To achieve higher level of nesting, if a relation *R* has a `NOT NULL` FK_R, $FK_R \rightarrow P$, and there is no loop between *R* and *P* in $G_{\mathbf{R}}$, we can move the element for *R* under the *P* element. This rule reflects N:1 cardinality among rows of *R* and *P*.

XML Conversion Algorithm

SQL data types are categorized into built-in and user defined types (UDT). Built-in data types are further differentiated into simple and complex data types. Simple (e.g. numeric or string) data types are straightforward translated into simple types in XSD.

Complex data types (as ARRAY, MULTISET, ROW) are processed in a different way. ARRAY or MULTISET is a collection of the same basic type. This basic type can be repeatedly complex data type, so the translation recursively generates all necessary definitions according mapping rules given by SQL:2003. Data type ROW is translated into complex type containing record for every simple item, which this data type ROW contains. Simple items can be of complex data type, so they must be recursively processed as well. These definitions result in a nested structure.

Let *N* be a type ARRAY, MULTISET, or ROW used in ***R***. For purposes of this paper we denote the nearest supertype of *N* in ***R*** as *owner* of *N*. Clearly, there can be more such owners. These additional definitions are used in more complex XSD definitions.

Processing of UDTs depends on their complexity. In the case of UDT founded on a simple data type, a simple type is created in XSD. Otherwise, a complex data type is created, as well as by complex data types.

As the standard uses as key constructs `key`, `keyref` clauses and does not prefer combination `ID` and `IDREF` in the case of single-attribute keys, we use in our algorithm `key`, `keyref` for all keys. Relation attributes are transformed to elements in all cases.

Phase I. – Preparation. For nested types (tables) (see MULTISET, ARRAY, and ROW possibilities) their owner types (tables) are determined. In this case, the definition of a new complex type describing such a nested table has to be introduced first in the resulted XSD. Then the definition of its owner relation can be introduced. It will contain the definition of the nested table as an element.

Suppose that all information about ***R*** were analyzed and stored. The following steps are performed:

1. Each schema *R* from ***R*** receives a type according to the NeReFT classification. This type depends on the number of FK_Rs in PK_R and whether the FK_R is entire the PK_R. In the case, if *R* is a component or a supplementary relation, its parent relation is determined.
2. An order is assigned to all schemes of ***R***. According to the order the R_i will be processed. This order is implied by mapping rules (1) – (6).
    a. First, regular relations are processed. For each such *R* the component and supplementary relations dependent of *R* are preferred. They obtain lower order and will be processed earlier. This process is done recursively because these dependent relations can be parents for other dependent relations. After processing all dependent component and supplementary relations the *R* is incorporated into the ordered list of relations.
    b. For each remaining (association) relation *R* an order is set. It follows the order in which the metadata concerning *R* is stored in the XMLConversion implementation.
    c. There are created records about the explicit nested relations.
3. Based of the rule (7) the order of relations to be processed is changed. For each *R* is tested if the condition in (7) is fulfilled. If yes, then (7) can be applied. The order is modified in this way that the parent relation of *R* will be processed later than the child relation.

Phase II. - *Generating XSD*

1. XML declaration is generated and information about namespaces is put into tags `<xsd:schema>` and `<xsd:import>`.
2. All relations are processed in the given order according to the mapping rules included in the standard. If a relation depends on a currently processed relation *R* (or other relations are nested in *R*), in the definition of complex type describing *R* a new element is included. The type of this element was defined earlier in Phase II. This means, that explicit nested relations are processed too and definitions of their types are used for new elements included in *R.*
3. After creating the types defining structures of all relations, a new type $T_{\mathbf{R}}$ describing the entire schema ***R*** is created. $T_{\mathbf{R}}$ contains elements whose types are the types describing particular R_i. Only the types are used that are not nested and do not occur in other types.
4. A new element is created, whose type is $T_{\mathbf{R}}$. This element will contain all definitions of keys and references to keys via elements

`<xsd:key>`, `<xsd:unique>` and `<xsd:keyref>`. The XSD is closed with `</xsd:schema>`.

Phase III. - Generating XML document.

1. XML declaration is generated. In the start tag of the root element, with name corresponding to ***R***, attributes with information about namespaces are stored.
2. According to the predefined order of the relations, processing all unnested tables are consecutively taken. For each relation R^* all its nested relations are determined. The relation R^* is then processed in the following way:
   a. For each row of R^*, a part of XML document containing row's data is generated. Then the associated nested data follows. Values of FK attributes of rows of nested data match values of PK attributes in this part of XML document.
   b. When input of nested data is finished, the part of XML document associated to one row from R^* is closed. The closing depends on a way how the tables are mapped to the document.
3. According to the given order all yet non-processed *R* are consecutively chosen. Data of each R^* is processed according to the mapping rules. The entire XML document is closed by the end tag of the root element.

Implementation

For implementation it is necessary to gain all important about ***R***. This information is stored in the system tables in a way which differs in various RDBMSs. A lot of special parameterized SQL queries have to be constructed for this purpose. For our implementation the Interbase RDBMS has been used and application development environment Delphi 6. Any transfer of the system to another RDBMS would require a change of this part.

Generating schemes is completely independent on RDBMS. As a parser and validator of XML documents we used Altova XMLSpy 2005 tool.

Conclusions

The problem addressed in this paper is related to exporting relational data in a native XML store. Our algorithm is designed with respect to the rules recommended by the specification SQL/XML. In implementation we had to change some details of this specification, as it not followed through. In opposite case, the generated XML documents could be not valid against the generated XSD.

An open problem is how to preserve more integrity constraints in XSD, i.e. these ones contained in the CHECK clause of CREAT TABLE statement of SQL. As yet this case is not tackled in a satisfactory way.

Acknowledgement

This research was supported in part by the National programme of research (Information society project 1ET100300419).

References

Bray T, Paoli J, Sperberg-McQueen CM, Maler E (2004) Extensible Markup Language (XML) 1.0 (Second Edition). W3C, www.w3.org/TR/REC-xml.

Clark J, DeRose S (1999) XML Path Language (XPath) Version 1.0. W3C, November www.w3.org/TR/xpath.

Duta C (2003) Conversion from Relational Schema to XML Nested-Based Schema. MSc. Thesis, Calgary, Canada.

Fallside DC, Walmsley P (2004) XML Schema Part 0: Primer. W3C, www.w3.org/TR/xmlschema-0/.

ISO (2004) Information technology – Database languages – SQL – Part 14: XML-Related Specifications (SQL/XML). ISO/IEC 9075-14:2004.

ISO 2003 Information technology – Database languages – SQL – Part 2: Foundation (SQL/Foundation). ISO/IEC 9075-2:2003.

Lee D, Mani M, Chiu F, Chu WW (2001) Nesting-based Relational to XML Schema Translation. Int'l Workshop on the Web and Databases (WebDB), Santa Barbara, CA.

Lee D, Mani M, Chiu F, Chu WW (2002) Effective Schema Conversions between XML and Relational Models. Proc. of European Conf. on Artificial Intelligence (ECAI), Knowledge Transformation Workshop (ECAI-OT), Lyon, France.

Liu Ch, Liu J, Guo M (2003) On Transformation to Redundancy Free XML Schema from Relational Database Schema. Proc. of APWeb 2003, LNCS 2642, pp 35-46.

Reschke J (2005) Transformations of relational database schemes to XML schemes. MSc. Thesis, Charles University, Prague, (in Czech).

Vincent MW, Liu J, Liu Ch (2004) Redundancy Free Mappings from Relations to XML. Proc. of WAIM 2004, LNCS 3129, pp 346-356.

W3C (2001) Semantic web. http://www.w3.org/2001/sw/.