XML-Enabled Relational Database for XML Document Update

Eric Pardede, J. Wenny Rahayu
Dept. of Comp. Science & Comp. Engineering
La Trobe University
Bundoora VIC 3086, Australia
{ekpardede, wenny}@cs.latrobe.edu.au

David Taniar
School of Business Systems
Monash University
Clayton VIC 3800, Australia
David.Taniar@infotech.monash.edu.au

Abstract

With increasing demands for a proper and efficient XML Data storage, XML-Enabled Database (XEnDB) has emerged as one of the popular answers. It claims to combine the strengths and limit the shortcomings of the traditional Database Management Systems and Native XML Database. The implication is more research need to be done for this database family.

This paper focuses on the XML update management in XEnDB. Our aim is to preserve the conceptual semantic constraints in XML data during update operations. The constraints are classified and represented in SQL/XML Schema. Then, we propose the update methodology that utilizes the proposed schema and implement the method in one of the current XEnDB products.

1. Introduction and Motivation

With the increasing importance of eXtensible Markup Language (XML) data interchange, repository choice has become a crucial decision. Many database users prefer to store their XML data in an established database system (DBMS), such as in Relational Database (RDB) [4,12], Object Oriented Database (OODB) [8], Object-Relational Database (ORDB) [13], etc. In these DBMSs, XML data will be restructured into a specific unit such as tables in RDB/ORDB and classes in OODB.

Another group of users claims that XML storage has to facilitate the tree-nature of XML. It introduces the Native XML Database (NXD) [6,7]. However, since it is considerably in its infancy stage, it cannot compete with DBMSs that are already in their mature establishment. Even though the NXD communities are continuously improving their products, there are still a lot of database users, who are not willing to leave years of investment and development in current DBMS such as RDB.

Like many other problems, people try to find the answer in moderation. It marks the emergence of XML-Enabled Database (XEnDB) [1]. It is a database that is built on top of a well-established database to utilize its mature technology with some adjustments to facilitate XML nature. This database contains extensions for transferring data between XML documents and the data structures of their underlying database [1].

Since XEnDB is based on different data models, we cannot find a general data model for XEnDB. Each XEnDB has its own design process and implementation, which is determined by the underlying DBMS.

Despite the non-existence of the data model, ISO/ANSI has been working on a standard for XEnDB; which is called SQL/XML [5]. It serves the same purpose as SQL, OQL or XQuery. As in these three languages, SQL/XML provides capabilities to store and to retrieve the data. However, like XQuery [14], SQL/XML does not yet provide a full capability for data update.

XML Update is a topic that is still in its infancy stage. Even in NXD, update has not been fully implemented [1]. NXD products only support very primitive updates. In addition, there is no guarantee that the updated document maintains its original conceptual constraints. In [9] we preserve XML document constraints during update in NXD. However, to the best of our knowledge, there is no work that has done the same purpose for XEnDB.

Like many other problems, people try to find the answer in moderation. It marks the emergence of XML-Enabled Database (XEnDB) [1]. It is a database that is built on top of a well-established database to utilize its mature technology with some adjustments to facilitate XML nature. This database contains extensions for transferring data between XML documents and the data structures of their underlying database [1].

Since XEnDB is based on different data models, we cannot find a general data model for XEnDB. Each XEnDB has its own design process and implementation, which is determined by the underlying DBMS.

Despite the non-existence of the data model, ISO/ANSI has been working on a standard for XEnDB; which is called SQL/XML [5]. It serves the same purpose as SQL, OQL or XQuery. As in these three languages, SQL/XML provides capabilities to store and to retrieve the data. However, like XQuery [14], SQL/XML does not yet provide a full capability for data update.

XML Update is a topic that is still in its infancy stage. Even in NXD, update has not been fully implemented [1]. NXD products only support very primitive updates. In addition, there is no guarantee that the updated document maintains its original conceptual constraints. In [9] we preserve XML document constraints during update in NXD. However, to the best of our knowledge, there is no work that has done the same purpose for XEnDB.

Like many other problems, people try to find the answer in moderation. It marks the emergence of XML-Enabled Database (XEnDB) [1]. It is a database that is built on top of a well-established database to utilize its mature technology with some adjustments to facilitate XML nature. This database contains extensions for transferring data between XML documents and the data structures of their underlying database [1].

Since XEnDB is based on different data models, we cannot find a general data model for XEnDB. Each XEnDB has its own design process and implementation, which is determined by the underlying DBMS.

Despite the non-existence of the data model, ISO/ANSI has been working on a standard for XEnDB; which is called SQL/XML [5]. It serves the same purpose as SQL, OQL or XQuery. As in these three languages, SQL/XML provides capabilities to store and to retrieve the data. However, like XQuery [14], SQL/XML does not yet provide a full capability for data update.

XML Update is a topic that is still in its infancy stage. Even in NXD, update has not been fully implemented [1]. NXD products only support very primitive updates. In addition, there is no guarantee that the updated document maintains its original conceptual constraints. In [9] we preserve XML document constraints during update in NXD. However, to the best of our knowledge, there is no work that has done the same purpose for XEnDB.

Like many other problems, people try to find the answer in moderation. It marks the emergence of XML-Enabled Database (XEnDB) [1]. It is a database that is built on top of a well-established database to utilize its mature technology with some adjustments to facilitate XML nature. This database contains extensions for transferring data between XML documents and the data structures of their underlying database [1].

Since XEnDB is based on different data models, we cannot find a general data model for XEnDB. Each XEnDB has its own design process and implementation, which is determined by the underlying DBMS.

Despite the non-existence of the data model, ISO/ANSI has been working on a standard for XEnDB; which is called SQL/XML [5]. It serves the same purpose as SQL, OQL or XQuery. As in these three languages, SQL/XML provides capabilities to store and to retrieve the data. However, like XQuery [14], SQL/XML does not yet provide a full capability for data update.

XML Update is a topic that is still in its infancy stage. Even in NXD, update has not been fully implemented [1]. NXD products only support very primitive updates. In addition, there is no guarantee that the updated document maintains its original conceptual constraints. In [9] we preserve XML document constraints during update in NXD. However, to the best of our knowledge, there is no work that has done the same purpose for XEnDB.
Without constraints preservation mechanism, an updated XML data can become invalid and does not
confirm with its original schema. At the end, we may end
up with some classic database problems such as data
inconsistency and redundancy.

In this paper our proposed solution can be grouped
into two parts (see fig.1). The first part is the
transformation phase and the second part is the update
mechanism phase.

For the transformation phase, we represent different
constraints in XML data conceptual model into a logical
model, which takes form as a schema. The result of this
transformation will be adjusted with SQL/XML
components as the standard for XEnDB. We then
implement this schema into an XEnDB product. For the
update mechanism phase, we propose XML update
methodologies that preserve various data constraints.
These methodologies will be implemented as query or
PL/SQL in the selected XEnDB product.

2. XML Constraints

To have a safe update, we firstly need to recognize the
constraints of XML data and preserved them in the
model. We will discuss this issue in this section.

2.1. XML Constraint Classification

To the best of our knowledge there is no work that has
tried to classify XML semantic constraints. Some novel
works have tried to propose new data models for XML
and some are concerned with the transformation process.
[15] proposed transformation of XML conceptual model
to OR Schema. It has covered some relationship
structures that can be found in XML data. However, this
work is based on the relational implementation and thus,
only relational model constraints is covered.

[16] proposed a mapping of OO conceptual model into
XML Schema. While the conceptual model can be used
for XML data, it is based on the structure of OO concept.
Not all constraints in XML data are covered.

[3] proposed the conceptual model and transformation
of XML data into schema. This work has covered some
of the XML constraints represented in Semantic Network
Diagram. However, it concentrates on the transformation
level and not all XML data constraints are covered.

[2, 10] proposed the use of UML to represent XML
data. [2] focused on the use of UML for conceptual
model. However, it only covered the constraints of the
data instead of the semantic constraints. [10] focused on
the transformation of UML to XML Schema. Again, not
many constraints are covered in this work.

2.2. Representing Constraints in XML Schema

In this section we preserve the conceptual model of
XML documents using XML Schema. XML Schema will
be sufficient if we want to store the XML data inside a
NXD. If the storage does not keep the XML document in
a tree data format, additional steps are required. In [4], the
additional mapping is required to map the XML Schema
into Relational Schema. In [15], the XML Schema is
mapped into OR Schema with different approaches.

In this section we want to capture the constraints
described in [9] into XML Schema if the storage option is
XEnDB. For XEnDB, there are SQL/XML annotations
that can be added to XML Schema [5]. These annotations
transform the XML Schema components into SQL
components. The annotations are provided as namespaces
by different vendors. In this work we use the annotations
provided by Oracle XDB [11]. For case study, we use the
example of Faculty document depicted in figure 2.

Cardinality Constraint. Cardinality constraints in
XML Schema are determined by using “minoccurs” and
“maxoccurs”. It is a general practice that these constraints
are not used for attribute. A particular attribute will
appear, at most, once in an object/instance.

![Faculty XML Document](image-url)
For example, a faculty has to have one and only one name. If we use an element, we can determine the cardinality constraints as shown in schema below. Faculty also has to have one and only one ID. For an attribute, we do not use the same cardinality constraints. Instead, we just employ “use” constraint to “required”.

\[\text{<xs:element name = "Faculty">} \]
\[\text{<xs:complexType}>\]
\[\text{<xs:element name="FacName" type="xs:string" minoccurs="1" maxoccurs="1">}\]
\[\text{<xs:attribute name="FacID" type="xs:string" use="required"/>}\]
\[\text{<xs:complexType/>}\]

Now, we have to transform the XML Schema into XEnDB Schema. First of all, we have to add the namespace where the annotations are available. In this case the namespace is \( \text{xdb} \). All elements with “maxoccurs” > 1 will be implemented as collection in XEnDB. In the following schema we define \( \text{xdb:storeVarrayAsTable} \) to “true”. This global XML annotation tells the database to store all varray as nested tables. It can speed up the queries on XML elements [11].

\[\text{<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:xdb="http://www.oracle.com/xdb" xdb:storeVarrayAsTable="true">}\]

When XEnDB maps the XML schema to tables, the database gives default names to the tables and attributes. These names are not easy to remember and they do not represent the content of the data. Therefore, we can specify the name of the tables, types and attribute name like it is shown in the following schema.

\[\text{<xs:element name="Faculty" xdb:defaultTable="FacultyTable">} \]
\[\text{<xs:complexType name="FacultyType" xdb:SQLType="XDB_FacultyType">}\]
\[\text{<xs:sequence}>\]
\[\text{<xs:element name="FacID" type="xs:string" xdb:SQLName="FacID" xdb:SQLType="VARCHAR2"/>}\].
\[\text{<xs:complexType/>}\]

Note that in a table, there is no difference between attribute and elements. Thus, we need to convert all XML attributes into elements.

**Other Constraints.** We have to perform the mapping of each XML data constraint into XEnDB Schema. The mapping into XML Schema can be found in our previous work [9]. However, some additional SQL/XML adjustments are required in each constraint. Due to the page limitation, we do not provide each of them in this paper.

### 3. XML Update Methodology

In this section we propose the update methodology for each relationship to capture the conceptual constraints. The update can be classified into deletion, insertion and replacement. For each of the operations, we can select three maintenance operations, which are restricted, nullify and cascade. The target of the operations can be either an attribute or an element.

Notice that due to the limitation of space, we cannot demonstrate the update methodology for all constraints. We will only show the methods for maintaining the cardinality and the referential integrity constraints. In relation to referential integrity constraints, the update targets will hold functionality as key or keyref.

#### 3.1. Methodology for Insertion

If the target is a key we have to ensure that the value does not duplicate an existing key value in the database. It is done by two steps. First, we check the functionality of the target component. If it is a key, we take the second checking, which is to check the existing instance in the database. Algorithm 1 shows the methodology for insertion checking for key component as the target.

We can see that algorithm 1 only handle referential integrity constraint by avoiding duplication of a key component. The adhesion and cardinality constraint will not be affected. A key will automatically have a strong adhesion constraint and cardinality [1:1]. If the key value already exists, the adhesion and cardinality have been complied. Adding a duplicated key will violate the adhesion and cardinality constraints. Therefore, algorithm 1 has covered different constraints for this XML case.

For keyref insertion we need to check whether the new keyref refers to an existence key. We do not show the algorithm in this paper.

```
Algorithm 1. Insertion Update for Key Target
```

*Operation 1 (Read the Element)*
FOR ALL attribute and leaf element

*Operation 2 (Read the attribute and leaf element)*
FOR ALL attribute and leaf element

*Operation 3 (Check the XML Schema)*
IF target is Key
THEN

*Operation 4 (Check the XML Storage)*
IF it is a duplicate
THEN
Stop Insertion
ELSIF
ELSE Operation 5 (Move to the next att./leaf element)

For keyref insertion we need to check whether the new keyref refers to an existence key. We do not show the algorithm in this paper.
3.2. Methodology for Deletion

Deletion is the simplest update operation, especially if we want to delete an attribute or an element that contains simple data. We do not need to perform the checking on the instances’ values. However, if the target also has roles as key/keyref the checking on instances is required. This is to maintain the referential integrity constraint.

For different target functionality we can perform different update strategies. For example, we want to delete a target that holds functionality as a key. The deletion will be cancelled if the strategy is “restrict”. Otherwise, the deletion will be performed after some preliminary actions such as nullifying the keyref value for “nullify” strategy. We do not show the algorithms in this paper due to the page limitation.

3.3. Methodology for Replacement

For replacement, we need to check both key and keyref target. For key replacement, the process is simply the combination of key deletion and insertion. This process ensures that the new target is not a duplicate and the old target is not being referred by any keyref in the database.

For replacement of a keyref, we just need to check whether the new keyref content refers to a valid instance. No cardinality or adhesion checking is required since to accommodate a new keyref, we must have to delete another keyref beforehand. This process has a similarity to the methodology for keyref insertion. We do not show the algorithms in this paper due to the page limitation.

4. Implementation of XML Update

We have proposed the methodologies to preserve XML constraints during the update. Different XEnDB products will implement these methods differently. In this section we implement our methods using Oracle, one of the XEnDB products that are based on OR Model.

Oracle claims it has supported XML storage since its Oracle 8i version. However, not until Oracle 9i Release 2 does it support the use of XML Type for the XML data. For this implementation we use Oracle 10g Release 1. The proposed methods are implemented as PL/SQL triggers. Outside these triggers we will use some built-in functions specifically for XML Type [11]. We use the case study of Faculty document as it is shown in figure 2.

Before we implement the update methodologies, we create the tables to store the XML data. The tables will be based on the XEnDB schema. Once the schema is registered, we can use all or part of the schema to create a table or a column. In the implementation we use the registered schema to create tables Subject and Staff in Faculty document.

Instead of having one table for the whole document, we create a separate table for each complex type. By doing this, we can utilize more update facilities provided by Oracle. One example, we cannot perform a subject insertion if we store the faculty document in a single table. Another example, if we store the XML as a large tree, we cannot insert built-in constraints to elements in low tree hierarchy. Now the storages are ready, we can start implement the update methodologies.

Since Oracle 10g is based on OR data model, we can implement algorithm 1 by using primary and foreign key mechanism. The primary and foreign keys are the physical representation of logical key/keyref in XML Schema. Unfortunately, at the time of writing the key/keyref in XEnDB Schema is not supported by Oracle 10g. When we map the schema into object-relational tables, the information regarding the key and keyref is abandoned.

Table 1. XML Data Constraint Preservation Methodologies and Implementation in XEnDB

<table>
<thead>
<tr>
<th>Constraints</th>
<th>Proposed Methodology</th>
<th>Implementation in Oracle10g</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Built-in Function</td>
<td>Trigger PL/SQL</td>
</tr>
<tr>
<td>Association</td>
<td>Ref.Integrity</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>No.Participants</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>Adhesion</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>Cardinality</td>
<td>✓</td>
</tr>
<tr>
<td>Aggregation</td>
<td>Adhesion</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>Cardinality</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>Ordering</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>Homogeneity</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>Share-Ability</td>
<td>✓</td>
</tr>
<tr>
<td>Inheritance</td>
<td>Ex.Disjoint</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>No.Ancestors</td>
<td>✓</td>
</tr>
<tr>
<td>Collection</td>
<td>Set</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>List</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>Multiset</td>
<td>✓</td>
</tr>
</tbody>
</table>
Fortunately, we can preserve the key/keyref constraints after the table is created by table alteration. However, they are only applicable for table attribute of simple data type. For some cases we have to implement triggers to preserve the constraints.

Due to the page limitation, we cannot show the full implementation for the update methodologies. Table 1 summarises the implementation of different constraints in XEnDB.

Most of the constraints identified in the conceptual model can be captured using Oracle 10g. The only limitation is that it does not differentiate an attribute with an element. Therefore, during the mapping we map the attribute as an element with simple type. The built-in function that checks the XML instance with a registered schema also simplifies many of the checking mechanisms. In some other products more user-defined function might be needed.

5. Conclusion

XEnDB, which has emerged as a combination between traditional DBMS and NXD, is a popular solution for XML data repository. However, not many works have been done to research this database family.

This work investigates the aspect of updating XML Document stored in XEnDB. Our goal is proposing methodologies that can maintain the semantic constraints of the documents after some update operations. To achieve our target, we transform different XML conceptual constraints into the logical model. For XEnDB we use the XML Schema with SQL/XML Standard adjustments.

We use the schema to propose the generic update methodologies. We differentiate the methods based on the operations type: (i) insertion, (ii) deletion and (iii) replacement. For each we consider three different maintenance strategies: (i) restrict, (ii) nullify and (iii) cascade. We also consider different updated target functionality, either as a key, a keyref or just a simple content. These generic methods are the basic functions that need to be followed by any XML storage regardless of the underlying data model.

We have implemented the update methodologies using Oracle built-in constraints and some user-defined triggers written in PL/SQL. For some triggers, we might need a duplicate table to enable the checking. This is one shortcoming for this current work, since it is certainly costly to have duplicate value and duplicate operations. Nevertheless, to maintain the integrity of the data, some overhead cost is inevitable. In addition, this is only applicable to tables that contain critical information and require high data integrity.

6. References