Journal of Computing and Information Technology - CIT 14, 2006, 3, 1–16
doi:10.2498/cit.2006.03.06

Storing Linked XML Documents in Object-Relational DBMS

Pensri Amornsinlaphachai*, Nick Rossiter and M. Akhtar Ali

School of Computing, Engineering & Information Sciences, Northumbria University, Newcastle upon Tyne, UK

Currently, several researchers have proposed mapping both structure and constraints of XML documents to an object-relational database (ORDB). However, these researches cannot be conducted because of the limited range of constraints in available object-relational DBMSs. We therefore propose mapping rules that are practicable in available technologies. Normally, an XML document is treated as a database, so much data redundancy occurs. To solve this problem, we keep non-redundant data in several separate XML documents, link the data dispersed in these documents together by a mechanism called 'rlink' and then map this mechanism to ORDB. Finally we perform a case study in Oracle9i to illustrate the mapping of XML to ORDB according to our rules. Our contribution is that we find that mapping linked XML documents to traditional databases such as (O)RDB makes it easier to join several documents and to update several documents in one update command.

Keywords: XML, semi-structured data, constraints, linked XML documents, ORDB

1. Introduction

When XML (Extensible Markup Language) became an effective standard for representation of structured and semi-structured data on the Web, researches were undertaken to store [10, 23], query [13, 10, 8, 6, 7] and publish [30, 4, 11] XML documents. A dominating approach for storing XML documents is applying traditional databases: relational database (RDB), object-oriented database (OODB) and object-relational database (ORDB) to manage XML documents. Mapping XML documents to RDB is a most popular research [39, 19]. However, the structure of XML documents is hierarchical whereas relational structure is flat; thus it is not a good fit. To fit the structure of XML documents to traditional databases, it seems that OODB [1] is suitable; nevertheless, some constraints such as not null and delete cascade are not supported by OODB [5]. In ORDB, constraints are inherited from RDB; nevertheless, not every feature in ORDB supports constraints. Our motivation comes from three reasons as follows.

- There is some research [26, 20] presented to map XML documents to ORDB, but it is not practicable though researchers [26] claim to use SQL4. One open problem is that available DBMSs have not supported all the features of SQL4.
- Usually, an XML document is treated as a database keeping all data in one document; thus data redundancy always occurs. The problem of data redundancy can lead to data inconsistency and low performance when updates are performed. Some work [9, 3] presented the technique of reducing data redundancy during mapping XML documents to RDB by using XML functional dependencies. However, with this method, data in the XML documents differ from the data in the database; thus it becomes difficult to maintain different data sets between the two storages.
- Updating several XML documents or performing joins between XML documents in one update command is not easy and thus, presently, no work proposes a methodology for this task. This will be discussed in Section 7.

In our approach, we will demonstrate how to map both structure and constraints of XML doc-

*This work was supported by the Royal Thai Government via Nakhonratchasima Rajabhat University.

uments to ORDB with awareness of practicability in available technologies whereby several object-relational features can be exploited. We store non-redundant data in separate XML documents and then propose a mechanism for linking these separate documents together and propose the rules for mapping these linked XML documents to ORDB. The result derived from a case study by mapping XML to ORDB is conducted through Oracle9i.

The rest of this paper is organized as follows. Related work is discussed in Section 2 and a mechanism for linking XML documents is proposed in Section 3. Section 4. describes the rules for mapping linked XML documents to ORDB and Section 5. describes the rules for mapping constraints to ORDB. We present the preserving of order of XML elements in Section 6. and a case study in Section 7. Finally, conclusion and further work are discussed in Section 8.

2. Related Work

There are a number of researches concentrating on mapping XML documents to traditional databases. These researches can be separated into two categories: automatic mapping and non-automatic mapping. The non-automatic mapping method [14] requires users to specify how to map the structure of XML to the schema of a database and this method is proposed by several commercial DBMSs such as IBM DB2 and Oracle database server [28]. Our work focuses on the automatic mapping of XML documents to traditional databases proposed by several previous workers as follows.

For RDB, [31] utilized a DTD graph to represent a DTD and to find a good mapping strategy. The researchers proposed three approaches: basic inlining, shared inlining and hybrid inlining techniques to map DTD to relational schema. They indicated that the hybrid inlining technique is superior to basic inlining and shared inlining techniques. [10] adapted a data mining algorithm to identify supported patterns for storage in relations and combined semi-structured and relational techniques to process semi-structured data by using OEM model and RDB to store and manage semi-structured data. [12] evaluated several mapping techniques and indicated that the best overall approach is the attribute approach. However, another research presented by [35] identified that the attribute approach has a poorer performance than the DTD approach. [18] use both DTD and XML documents for mapping. They keep elements and path of elements in one table and keep attributes and path of attributes in another table. The reference between elements is represented by path-IDs kept in the tables. [39] presented their system called X-Database system which uses the XML-Schema file to generate a RDB schema and then decomposes valid XML documents according to the Schema to store their information in the database.

From the above researches, only [39] performed mapping constraints to database. However, mapping XML to RDB can produce many unnecessary tables leading to unnecessary joins in querying since usually XML documents contain multi-value attributes while mapping these data to RDB is performed by putting the data into separate tables.

For OODB, [1] proposed an approach to map semi-structured data (SGML) to an object model. In this approach, each SGML element definition in DTD is interpreted as a class, choice connector (|) is modeled by a union type, element components marked by "+" or "*" occurrence indicator are represented by lists, attributes are represented by private property of the class. Nonetheless, mapping semantics (constraints) of semistructured documents is not proposed because of the limited constraints of OODB.

In the case of ORDB, [32] proposed the method that decomposes XML documents into the nodes and stores them in four tables: element, attribute, text and path tables while [27] used XML data type to store a fragment of an XML document. However, these researchers did not use any object-relational feature. [20] and [23] exploited set/list and nested tables which are features of ORDB, but no DBMS supports both set/list and nested tables in one ORDB (Informix [17] supports list/set, Oracle [24] supports nested tables while PostgreSQL [25] supports array). In addition, [23] defined foreign keys in nested tables whereas [20] used several constraints such as domain and default constraints in collection type and [26] defined primary keys in collection type. Defining the full

range of constraints in nested tables or collection type is restricted in available object-relational DBMSs; thus none can be conducted. [36] mapped XML to ORDB by using UniSQL, but the researchers did not employ constraints and UniSQL/X itself is based on OODB having limited constraints [38].

To summarize, none of the previous work mapping both structure and constraints of XML to ORDB can be conducted in existing technologies; furthermore, none of the previous work mapped several types of linked XML documents to traditional databases; thus all XML update languages such as Extended XQL [42] and XML update extension [33] including update languages [43, 2, 29] for native XML database were designed to update an XML document without joins between documents.

3. A Mechanism for Linking XML Documents

In the case of (O)RDB, foreign keys and reference type are employed to represent inter-table references; thus to model linked XML documents, the mechanism for linking XML documents can be translated into foreign keys. For XML documents, XLinks (XML Linking Language) [41] and XInclude [40] are mechanisms for linking the documents together. However, XLink and XInclude are not designed from a database viewpoint; thus they do not provide enough information for linking XML documents from a database point of view. Moreover, XInclude [15] does not allow circular reference (recursion). The major purpose of XLink is to link XML documents in the Web while the main purpose of XInclude is to build a large XML document out of smaller XML documents. In our research, we propose a mechanism called rlink whose purpose is to associate the relationships between elements from different XML documents so that this provides more convenience for updating data across XML documents. The rlink provides information to identify the document and/or element to which a link is made. Although this may be extended to XLink, the main purposes of XLink and rlink are different from each other and currently only Mozilla and its derivatives such as Netscape support XLink, but the support is incomplete [16]; moreover, no XML query language supports XLink so we do not wish to make any extension to it.

To associate the relationships between elements from different documents, we propose two additional attributes, rlink:relationship and rlink:href. The rlink:relationship indicates which document and/or which element are involved in the rlink mechanism whereas the rlink:href links to the document and the element specified by rlink:relationship. The 'rlink' is used as a namespace.

The rlink mechanism will only serve the function of linking XML documents; thus elements containing rlink mechanism must be EMPTY and have no other attributes except rlink:relationship and rlink:href. In DTDs, the format of

```
<!ELEMENT Publications(Publication*)>
<!ELEMENT Publication(Title, Year, Author+)>
<!ATTLIST Publication PubID ID #REQUIRED >
<!ELEMENT Title(#PCDATA)>
<!ELEMENT Year(#PCDATA)>
<!ELEMENT Author EMPTY>
<!ATTLIST Author rlink:href CDATA #REQUIRED
  rlink:relationship #FIXED "Authors.xml::Author" >

<Publications xmlns:rlink = "http://www.unn.ac.uk/rlink">
 <Publication PubID = "P111">
  <Author rlink:href = "//Author[@AuthorID='A222']">
  </Author>
 ...
 </Publication>
 ...
</Publications>
```

```
<!ELEMENT Authors(Author*)>
<!ELEMENT Authors(Name, Email?)>
<!ATTLIST Author AuthorID ID>
<!ELEMENT Name(FName, LName)>
<!ELEMENT FName(#PCDATA)>
<!ELEMENT LName(#PCDATA)>
<!ELEMENT Email(#PCDATA)>


<Authors>
 <Author AuthorID = "A222">
  <Email>... </Email>
  ...
 </Author>
 ...
</Authors >
```

Fig. 1. Publications.xml and Authors.xml.

value assigned to rlink:relationship is LinkedDocument::LinkedElement and its property is FIXED while the value type of rlink:href is CDATA. In XML documents, the format of the value assigned to rlink:href is an XPath clause linking to the document and the element specified by rlink:relationship in the DTD.

To illustrate this, we will give an example of using rlink:relationship and rlink:href in DTDs and XML documents as follows:

Example 1: Suppose that there are two XML documents: Publications.xml and Authors.xml linked together by rlink as shown in Figure 1.

From the XML documents in Figure 1, the Author element in Publications.xml uses rlink:href as an attribute to link information of Author having AuthorID = A222 from Authors.xml whereas rlink:relationship in part of DTD is used to indicate which document and which element rlink:href will link to.

The recommendation for separating XML documents is as follows. C_1 and C_2 are complex elements (elements consisting of sub-elements or attributes) and value(E) is the value of element where E is an element. If several C_1 can refer to the same value C_2, the C_2 should be separated into another document.

For example, Publication and Author are complex elements where several publications can have the same author. Therefore Author is separated into another document. Another example is that Author and Address (of author) are complex elements as each author has a different address; but here there is no need to separate Address into another document. This principle is only a recommendation. Users may or may not separate a XML document into several documents. However, our mapping-rules support both forms of documents.

Note: The attribute rlink:relationship used in the DTD is applied to IDREF(s) to indicate the involved elements.

4. Mapping Linked XML Documents to ORDB

In this section, firstly we discuss the type of recursion that can occur in DTD, secondly we propose rules for mapping an XML document to ORDB and finally we present additional rules for mapping the rlink mechanism to ORDB. A diagram for mapping XML structure and rlink to ORDB is shown in Figure 2.

4.1. Forms of Recursion in DTD

In our research, mapping XML documents to ORDB is based on DTD since DTD is more compact than XML Schema; nonetheless, mapping XML documents based on XML Schema will be our future work. Normally, there are two forms of recursion in DTD. The first form of recursion comes from a recursive structure: an element contains its ancestor elements as child elements. The second form of recursion stems from IDREF(s). As we propose the rlink mechanism, the third form of recursion results in the case that two elements in two XML documents refer to each other.

4.2. Rules for Mapping Structure of an XML Document to ORDB

In our mapping rules, three features of object-relational technology: abstract data type, object table and nested tables, will be used. In the rules, elements having type #PCDATA and without attributes are called simple elements whereas elements consisting of child-elements or attributes are called complex elements.

1. Complex elements which do not correspond to the rules 2-5 are converted to object tables.
2. Complex elements having only one complex child-element are converted to object tables and their complex child-elements are converted to abstract data type fields.
3. Complex elements which have occurrence ? or 1 (although there is no single symbol for occurrence meaning one, from now on we will use the symbol '1' as canonical short label), have sibling and all children as simple elements are converted to abstract data type fields.
4. Complex elements being the root element and having only several complex child-elements with occurrence * or + are converted to nothing in ORDB and their child elements are converted to object tables.

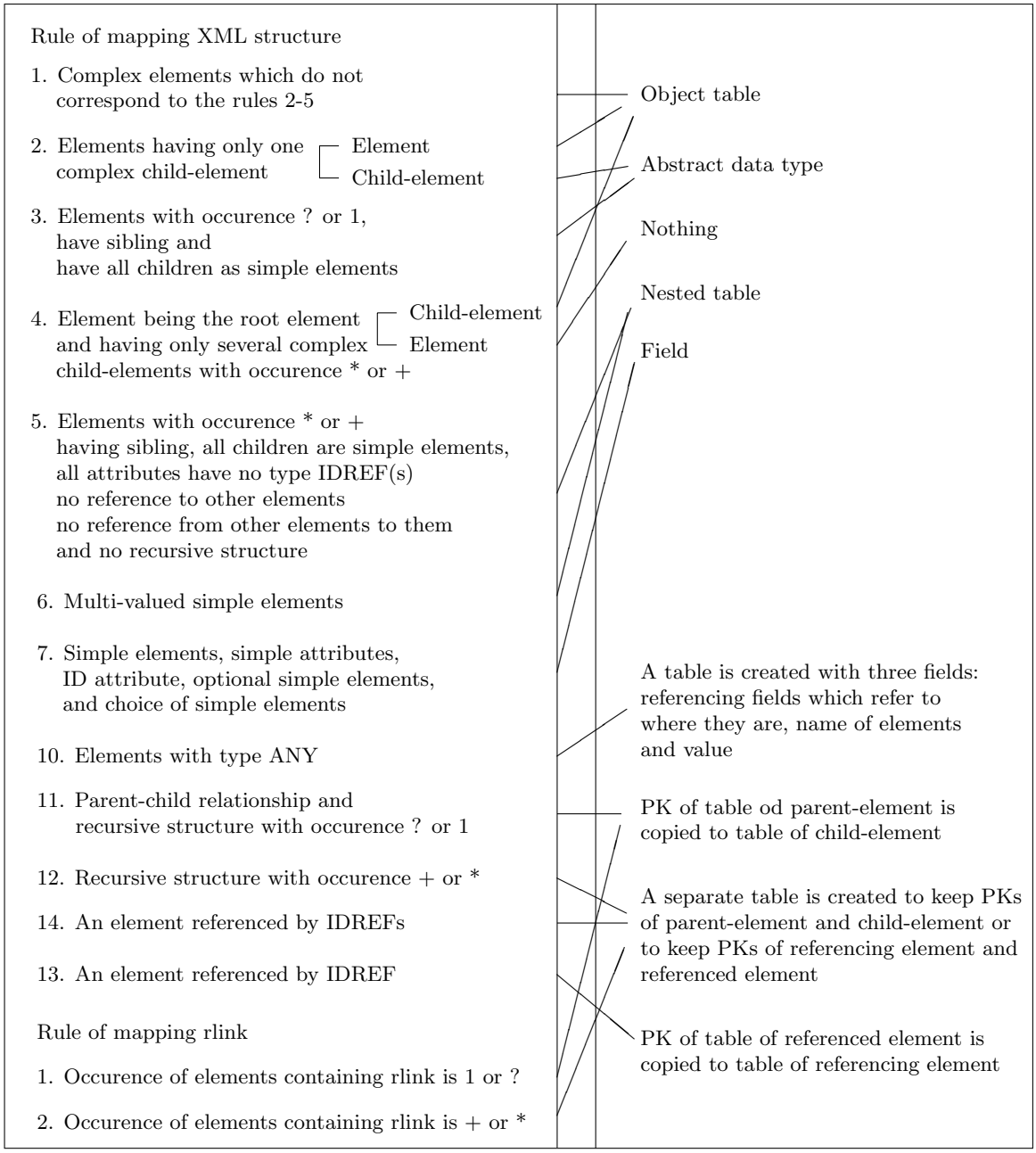

Fig. 2. Mapping XML structure and rlink to ORDB.

5. Complex elements having occurrence * or +, have siblings and comply with the following conditions:
   - All children are simple elements and all attributes have no type IDREF(s).
   - There is reference to other elements and no reference from other elements to them.
   - There is recursive structure: they must not refer back to their ancestors.

   are converted to nested-tables.

   This rule is to make sure that nested tables do not have any reference since the referential integrity constraint cannot be defined in nested tables; nevertheless, other constraints such as domain constraint and default constraint can be defined in nested tables.
6. Multi-valued simple elements are converted to nested tables having one field.
7. Simple elements, simple attributes, ID attribute, optional simple elements and choice of simple elements are converted to fields.

8. Optional complex elements and choice of complex elements are converted to tables or fields according to the rules 1-5.
9. For the choice of groups of elements where some of elements in each group are the same, duplicate elements are eliminated and then rules 1-7 are applied.
10. For elements with type ANY, a separate table is created with three fields: referencing fields which refer to their position, name of elements and value.
11. For parent-child relationship and recursive structure with occurrence ? or 1, the primary key of the table of parent-element is copied to the table of the child-element.
12. For recursive structure with occurrence + or *, a separate table will be created to hold the relationship of recursive structure by storing the primary keys of tables of a parent-element and a child-element.
13. For an element referenced by IDREF, the primary key of the table of a referenced element is copied to the table of a referencing element.
14. For an element referenced by IDREFs, a separate table will be created to keep the primary keys of tables of a referencing element and a referenced element.

4.3. Additional Rules for Mapping the Rlink Mechanism to ORDB

The relationship between XML documents is similar to the relationship specified by IDREF(s) in the same document; thus their mapping rules are similar too.

1. If the occurrence of elements containing rlink is 1 or ?, the primary key of the table of a referenced-element is added to the table of a referencing element.
2. If the occurrence of elements containing rlink is + or *, a separate table will be created to keep relationship between XML documents; thus the separate table consists of the primary keys of the table of a referencing-element and a referenced-element.
3. For recursive structure: rlink of elements in a referenced document refers to elements in a referencing document, it is considered in the same way as rules 1–2.

5. Mapping XML Constraints to ORDB Constraints

In this part, we firstly describe the types of constraints in (O)RDB. Secondly, we present the rules for mapping constraints in an XML document to constraints in ORDB. Thirdly, we propose supplementary rules for mapping constraints which stem from the rlink mechanism to constraints in ORDB. Finally, we determine how to preserve cardinality constraints when update operations are performed. A diagram for mapping XML constraints and rlink constraints to ORDB constraints is shown in Figure 3.

5.1. Type of Constraints in (O)RDB

The constraints in (O)RDB from a data-oriented viewpoint can be categorized into three types [37] as follows:

1. Row constraints: these constraints are related to exactly one table and can be evaluated independently for each row in that table. Constraints in this type include check (null value) constraint, domain constraint and default value constraint.
2. Table constraints: evaluating these constraints is associated with at least two rows in the same table. Examples of these constraints include primary key constraint, unique constraint and cardinality constraint.
3. Inter-table constraints: these constraints involve rows from at least two tables. An example of this constraint type is foreign key constraint (referential integrity constraint) including cascade rules.

Note: We have not found that the cardinality constraint is available in any (object-)relational products.

5.2. Rules for Mapping Constraints of an XML Document to ORDB

Since certain constraints in DTD are easily recognized, some of proposed rules are the same as some of the rules proposed in other work [22, 39, 21]. However, our work can extract more constraints in DTD than in previous work and some of our rules are different from the rules proposed in the previous work; in particular, no

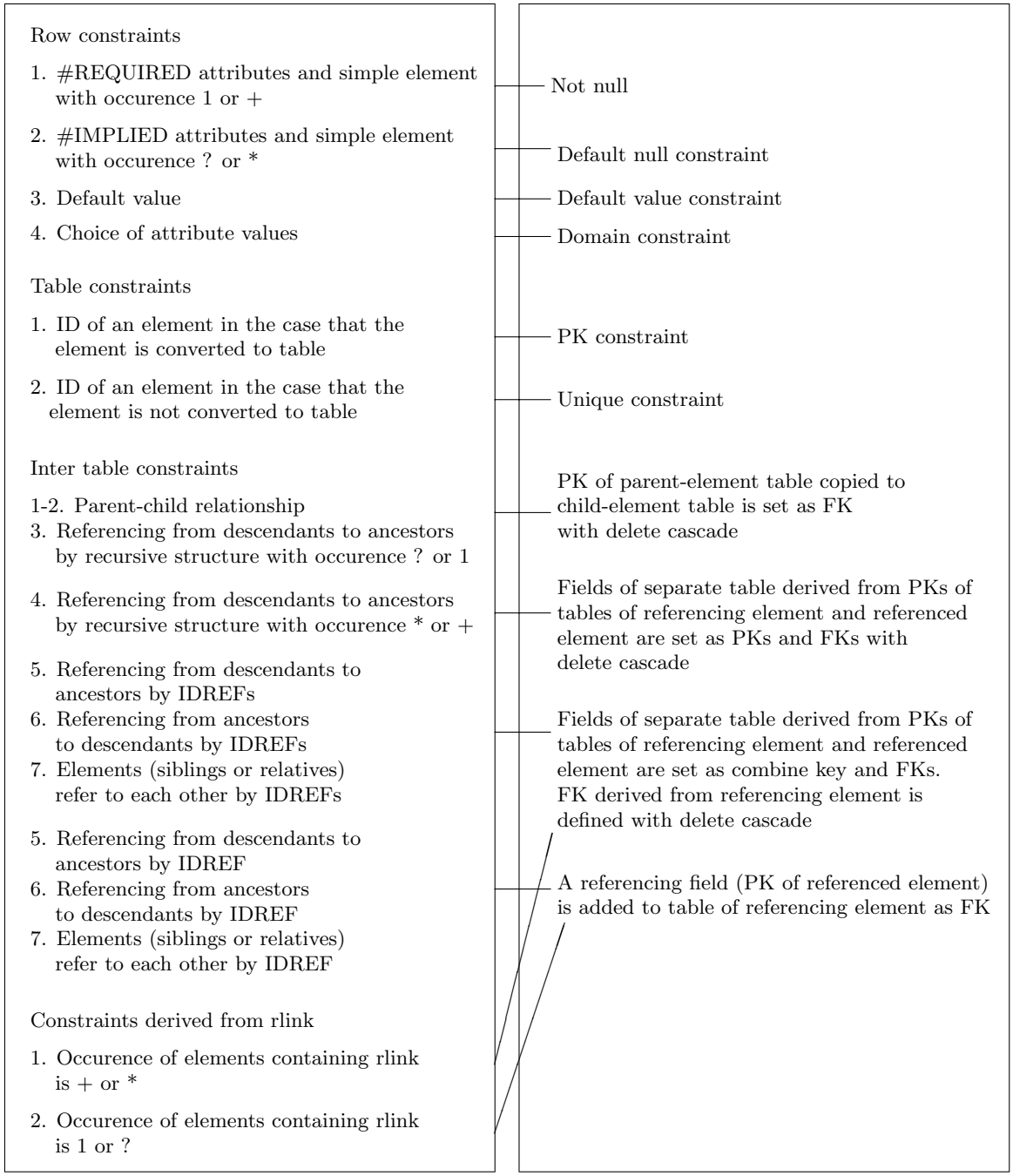

Fig. 3. Mapping XML constraints and rlink constraints to ORDB constraints.

work has proposed preserving the cardinality constraint when updates are performed. In this section, we will organize constraints in DTD according to constraint types found in (O)RDB as follows.

Row constraints:

1. #REQUIRED attributes and simple element with occurrence 1 or + are converted to the not null constraint.
2. #IMPLIED attributes and simple element with occurrence ? or * have default null constraint
3. Default value is translated into default value constraint.
4. Choice of attribute values is converted to domain constraints.
5. For a choice of elements such as <!ELEMENT e $(s_1 \mid s_2)$>, meaning can have either

s_1 or s_2 but not both simultaneously, then the constraint will be:

Check ((s1 is not null AND s2 is null) OR (s1 is null AND s2 is not null))

In translating <!ELEMENT e (s_1 |...| s_n)>, the constraint will become:

Check((s1 is not null AND s2 is null AND ... AND sn is null) OR ... OR (s1 is null AND s2 is null AND ... AND sn is not null))

6. For a choice of groups of elements, that some the of elements in each group are the same but their constraints may be different, are converted with the following rules.

   Firstly, every element+ or element in each group is converted to "AND element is not null". Secondly, every element* or element? in each group is converted to nothing. Thirdly, the OR operation is performed on every group. Finally, if there are some elements in one group which do not appear in other groups, the "AND these elements are null" is added to the groups for which these elements do not appear. For example:

   (name, telephone+)|(name, telephone*, email)

   Constraints will be:

   Check ((name is not null AND telephone is not null AND email is null) OR (name is not null AND email is not null))

Table constraints:

1. ID of an element is converted to primary key constraint in the case that the element is converted to an object table or a nested table; otherwise ID is defined with unique constraint.
2. Occurrence for complex elements converted to tables should be converted to the cardinality constraint; however no (object-)relational technology provides this constraint; hence this constraint will be checked with rules for preserving the cardinality constraint in Section 5.4.

Inter-table constraints:

1. Parent-child relationship (1 to many relationship)
2. Parent-child relationship (1 to 1 relationship)
3. Referencing from descendants to ancestors by recursive structure with occurrence ? or 1
4. Referencing from descendants to ancestors by recursive structure with occurrence * or +
5. Referencing from descendants to ancestors by IDREFs or IDREF (Recursive)
6. Referencing from ancestors to descendants by IDREFs or IDREF
7. Elements (siblings or relatives) refer to each other by IDREFs or IDREF

From parent-child relationship and recursive structure with occurrence ? or 1 (1-3), the primary key of parent-element table is copied to the child-element table as a foreign key constraint and a delete cascade is defined on this constraint.

From referencing by IDREFs and recursive structure with occurrence * or + (4-7), a new separate table will be created to hold relationships of references. This separate table consists of fields derived from the primary keys of tables of a referencing element and a referenced element. These fields are set as a combined key and foreign keys for this table. The foreign key derived from a referencing element is defined with the delete cascade. In the case of referencing by IDREFs (5-7), the foreign key derived from a referenced element is defined without a delete cascade whereas in the case of recursive structure (4), the foreign key derived from a referenced element is defined with a delete cascade.

From referencing (5-7) by IDREF, a referencing field (same as primary key of the table of referenced element) is added to the table of a referencing element as a foreign key without a delete cascade to point to the primary key of a referenced element.

Not null is defined on a foreign key in the case that occurrence is 1 or IDREF is declared with #REQUIRED.

Note: For a table without a primary key, the RowID automatically created in an object table will be used as the primary key.

5.3. Additional Rules for Mapping Constraints Derived from Rlink Mechanism

The supplementary constraint rules for mapping rlink mechanism are similar to the constraint rules of IDREF(s). The rules are as follows.

1. In the case that the occurrence of elements containing rlink is 1 or ?, the primary key of the table of a referenced element will be held in the table of a referencing element as a foreign key without a delete cascade.
2. In the case that the occurrence of elements containing rlink is * or +, a separate table is created consisting of two fields derived from keys of tables of referencing element and referenced elements. These two fields are set as a combined key and foreign keys but only the foreign key derived from the table of a referencing element is defined with a delete cascade whereas the foreign key derived from the table of a referenced element is defined without a delete cascade.
3. For a recursive structure, constraints are considered in the same way as rules 1, 2 and 4.
4. Not null is defined on a foreign key in the case that the occurrence of elements containing rlink is 1.

5.4. Rules for Preserving Cardinality Constraints when Updates are Performed

In this section, we will describe how the cardinality constraint is preserved when updates are performed since, nowadays, no (object-)relational DBMS can handle the cardinality constraint; hence a particular method is needed to manage it. Updating affects the relationship between elements; thus we preserve this constraint according to the type of relationship in the XML documents as follows:

1. Parent-child relationship (1 to many relationship)

   In the case that the child is a complex element converted to a nested table or the child is a complex element and has no sibling; then the child is converted to an abstract data type field and the parent element is converted to an object table containing only one abstract data type field.

   Delete child elements converted to abstract data type fields in case of occurrence +

   Cardinality constraint will be checked as follows:

   ```
   Select count (*) as count1
   From parent-element table;
   Select count (*) as count2
   From parent-element table
   Where delete-conditions;

   If (count2 − count1) >= 0 then
   Do not allow deletion
   End If
   ```

   Delete child elements converted to a nested tables in case of occurrence +

   Cardinality constraint will be checked as follows:

   ```
   Select count (*) as count1
   From child-element table
   Where PK(of parent-element table) = $PK;

   Select count (*) as count2
   From child-element table
   Where PK(of parent-element table) = $PK
   And delete-conditions;

   If (count2- count1) >= 0 then
   Do not allow deletion
   End If
   ```

   *Delete child elements in the case of occurrence **

   No need to be checked.

   *Insert children in case of occurrence + or **

   Cardinality constraint is not needed to be checked since any number of children is allowed when inserting

2. Parent-child relationship (1 to 1 relationship)

   In the case that the child is a complex element and has no sibling; then the child is converted to an abstract data type field (and the parent element is converted to an object table containing only one abstract data type field).

   Delete child elements in the case of occurrence 1

   Do not allow deletion.

   Delete child elements in the case of occurrence ?

   Insert children in case of occurrence 1 or ?

   Cardinality constraint is checked as follows:

   ```
   Select count(*) as count1
   From parent-element table;

   IF count1 > 0 then
   Do not allow insertion
   End If
   ```

3. Parent-child relationship (1 to many relationship) including recursive structure with occurrence * or +

   In the case that the parent and child are complex elements, the child is converted to an object table. In the case of recursive structure, a separate table is created to hold the relationship between the referencing (parent-element) and referenced elements (child-element).

   Delete child elements in case of parent-child relationship with occurrence +

   Cardinality constraint will be checked as follows:

   ```
   Select count (*) as count1
   From child-element table
   Where FK (PK of parent-element table) = $FK;

   Select count (*) as count2
   From child-element table
   Where FK = $FK
   And delete-conditions;

   If (count2- count1) >= 0 then
   Do not allow deletion
   End If
   ```

   Delete child elements in case of recursive structure with occurrence +

   Cardinality constraint will be checked as follows:

   ```
   Select count (*) as count1
   From separate table
   Where PK1 (PK of parent-element table) = $PK;

   Select count (*) as count2
   From separate table S, child-element table C
   Where S.PK1 = $PK And S.PK2 = C.PK
   And delete-conditions;

   If (count2- count1) >= 0 then
   Do not allow deletion
   End If
   ```

   *Delete child elements in case of occurrence **

   No need to be checked.

   *Insert children in the case of occurrence + or **

   No need to be checked.

4. Parent-child relationship (1 to 1 relationship)

   In the case that the parent and child are complex elements, the child is converted to an object table.

   Delete child elements in case of occurrence 1

   Do not allow deletion

   Delete child elements in case of occurrence ?

   No need to be checked.

   Insert children in case of occurrence 1 or ?

   Cardinality constraint is checked as follows:

   ```
   Select count(*) as count1
   From child-element table
   Where FK (PK of parent-element table) = $FK;

   IF count1 > 0 then
   Do not allow insertion
   End If
   ```

5. Referencing from descendants to ancestors by IDREFs (Recursive)
6. Referencing from ancestors to descendants by IDREFs
7. Elements (siblings or relatives) referring to each other by IDREFs
8. Referencing between XML documents by using rlink mechanism where the occurrence of an element containing rlink is + or *

   For cases 5-8 above, in the case of referencing by IDREFs or by the rlink mechanism where the occurrence of an element containing rlink is + or *, a separate table is created to hold the relationship between the referencing and referenced elements.

   Delete values in #REQUIRED IDREFs or delete elements containing rlink where occurrence of the elements is +

   Cardinality constraint will be checked as follows:

   ```
   Select count (*) as count1
   From separate table
   Where PK1 = $PK1;

   Select count (*) as count2
   From separate table
   Where PK1 = $PK1
   And PK2 = ($PK derived from delete-conditions);

   If (count2-count1) >= 0 then
   Do not allow deletion
   End If
   ```

   *Delete values in #IMPLIED IDREFs or delete elements containing rlink where occurrence of the elements is **

   No need to be checked.

   Insert values to IDREFs or insert rlink

   No need to be checked.

6. Preserving Order of XML Elements

When ordered XML documents are shredded into tables, ordering in tables has two dimen-

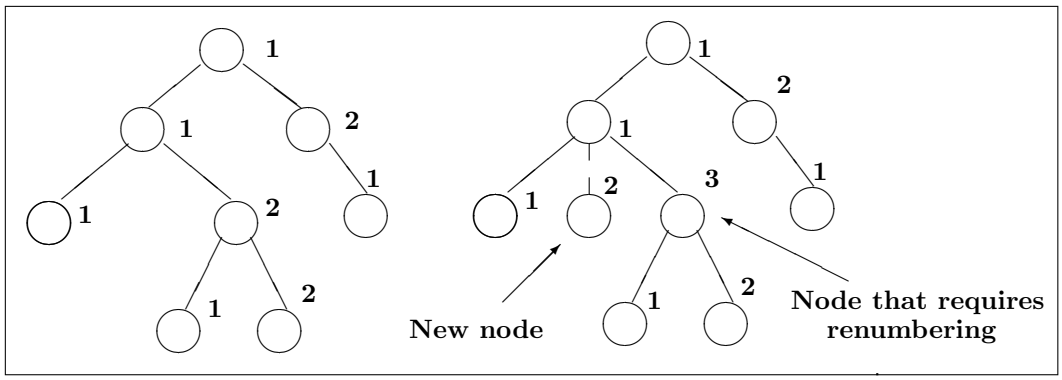

Fig. 4. Local Order Method.

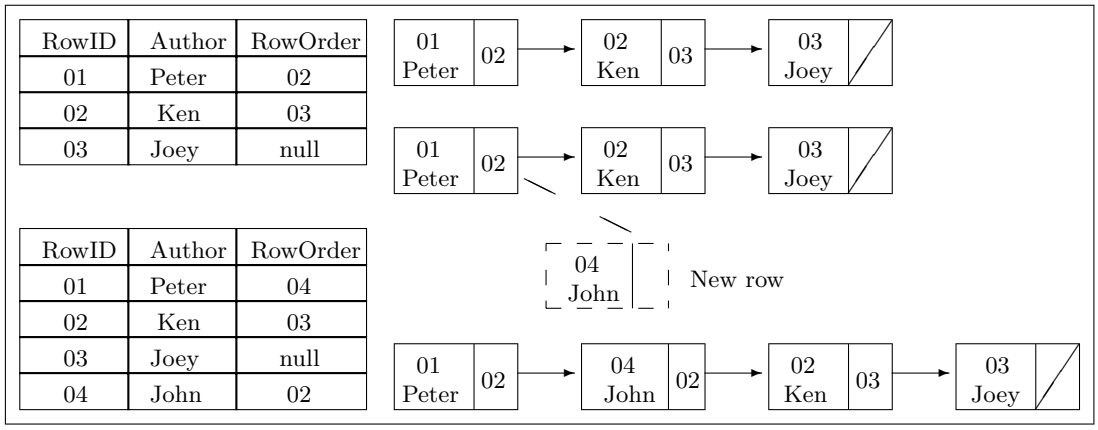

Fig. 5. Linked List Order Method.

sions: column ordering and row ordering. For columns, they are automatically ordered when tables are created and this order will never be changed. For row ordering, at the first time of loading data into tables, data can be loaded in sequence but when delete or insert operation is performed, it is necessary to reorder the data.

There is a proposal [34] for reordering XML data held in relational database. The researchers of this proposal demonstrated that Local Order method performs best on updates since only siblings following the new node need to be renumbered as shown in Figure 4.

However, this method can yield low performance if the new node has numerous siblings as many siblings must be renumbered. For our method, rows are considered as a linked list. This means that each row will hold the RowID (automatically generated) of the next row in sequence; thus this method can guarantee that when the insert or delete operation is performed, no more than two rows are affected, as shown in Figure 5. From the figure, when we insert a new author 'John' after author 'Peter', this can be performed by copying RowOrder of 'Peter' to RowOrder of 'John' and copying RowID of 'John' to RowOrder of 'Peter'.

7. A Case Study for Mapping Linked XML Documents

To elucidate, we will demonstrate how to map linked XML documents to ORDB. We suppose that three XML documents, Publications.xml, Authors.xml and References.xml, are linked together by two attributes: rlink:relationship and rlink:href, as shown in Figure 6. To gain more understanding, Schema Graph of DTDs in Figure 7 and the result tables in Figure 8 should be considered along with the description below.

Firstly, Publications has only one complex child element: Publication; thus Publications is converted to an object table and Publication is con-

```
<!ELEMENT Publications(Publication*)>
<!ELEMENT Publication(Title, Author+, Year, Reference?)>
<!ATTLIST Publication PubID ID #REQUIRED >
<!ATTLIST Publication PubType (book|article|journal) "book">
<!ELEMENT Title (#PCDATA)>
<!ELEMENT Author EMPTY>
<!ATTLIST Author rlink:href CDATA #REQUIRED
  rlink:relationship CDATA #FIXED "Authors.xml::Author">
<!ELEMENT Year (#PCDATA)>
<!ELEMENT Reference EMPTY>
<!ATTLIST Reference rlink:href CDATA #REQUIRED
  rlink:relationship CDATA #FIXED "References.xml::Reference">

<Publications xmlns:rlink="http://www.unn.ac.uk/rlink">
<Publication PubID = "P111">
  ...
  <Author rlink:href = "//Author[@AuthorID = 'A111']">
  </Author>
  <Reference rlink:href = "//Reference[@RefID = 'R111']">
  </Reference>
</Publication>
...
</Publications>
```

```
<!ELEMENT Authors(Author*)>
<!ELEMENT Author(Name, Email?,Telephone*)>
<!ATTLIST Author AuthorID ID #REQUIRED>
<!ELEMENT Name (FName, MName?, LName)>
<!ELEMENT FName (#PCDATA)>
<!ELEMENT MName (#PCDATA)>
<!ELEMENT LName (#PCDATA)>
<!ELEMENT Email (#PCDATA)>
<!ELEMENT Telephone (Location, TelNo)>
<!ELEMENT Location (#PCDATA)>
<!ELEMENT TelNo (#PCDATA)>


<Authors>
<Author AuthorID = "A111">
  <Name>
  <FName>John</FNAme>
  <LName>Smith</LName>
  </Name>
  ...
</Author>
...
</Authors>
```

```
<!ELEMENT References (Reference*)>
<!ELEMENT Reference(Publication+)>
<!ATTLIST Reference RefID ID #REQUIRED>
<!ATTLIST Reference RefType (References |Bibliography |Miscelleneuos) "References">
<!ELEMENT Publication EMPTY>
<!ATTLIST Publication rlink:href CDATA #REQUIRED
                      rlink:relationship CDATA #FIXED "Publicastions.xml::Publication">

<References xmlns:rlink="http://www.unn.ac.uk/rlink">
<Reference RefID = "R111" RefType = "Miscelleneous">
<Publication rlink:href = "//Publication[@PubID = 'P222']"></Publication>
<Publication rlink:href = "//Publication[@PubID = 'P333']"></Publication>
</Reference>
...
</References>
```

Fig. 6. Three linked XML documents.

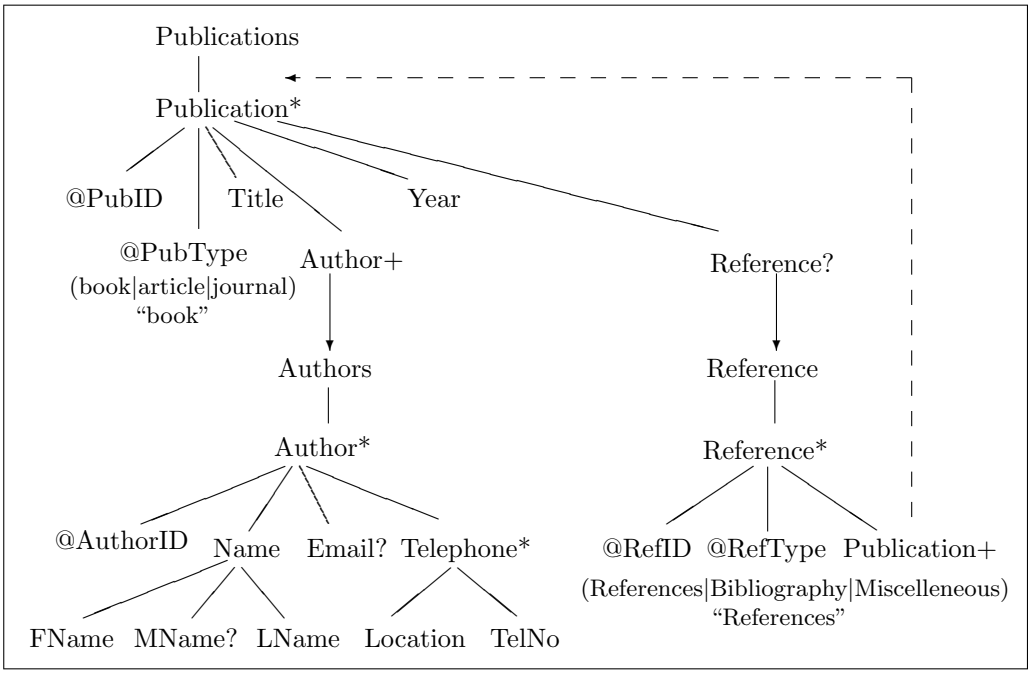

Fig. 7. Schema Graph of DTDs.

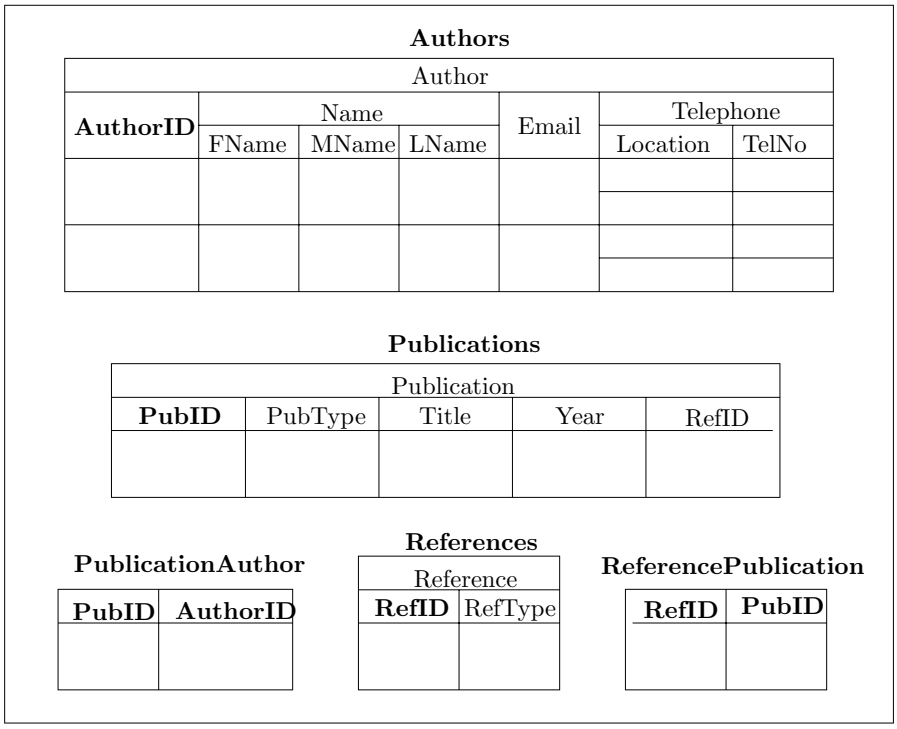

Fig. 8. Tables derived from mapping rules.

verted to an abstract data type field (Structure mapping Rule 1). Publication has attributes: PubID, PubType and simple elements: Title, Year; these four are converted to simple fileds (Structure mapping Rule 5). For the constraints, Title and Year have occurrence 1; thus not null is applied (row constraint mapping rule 1) and table constraint mapping rule 1 is applied to PubID as a primary key. PubType has a choice of attribute values and a default value; so domain and default value constraints are applied (row constraint mapping rules 3–4). For Publication*, rules for preserving cardinality constraint are applied when updates are performed.

Secondly, the complex child elements of Publication: Author and Reference whose attributes are rlink are handled as follow:

- Since the occurrence of Author is +, a separate table is created. This table consists of the primary keys of Publication and Author set as a combined key and foreign keys. PubID is defined with a delete cascade (structure mapping rule 2 and constraint mapping rule 2 of rlink are applied).
- Since the occurrence of Reference is ?, the primary key of Reference is added to Publication and is set as a foreign key (structure mapping rule 1 and constraint mapping rule 1 of rlink are applied).

Thirdly, Authors has only one complex child element: Author; thus Authors is converted to an object table and Author is converted to an abstract data type field (Structure mapping Rule 1). Author consists of attribute: AuthorID and simple element: Email; so these two are converted to simple fields (Structure mapping Rule 5). Name has a sibling, occurrence 1, and all children are simple elements; thus Name is converted to an abstract data type field (Structure mapping Rule 3). Telephone has a sibling, occurrence *, and all children are simple elements and there is no reference; hence Telephone is converted to a nested table (Structure mapping Rule 2).

For the constraints, AuthorID is set as a primary key (table constraint mapping rule 1), Name, FName, LName, Location and TelNo have occurrence 1; thus not null is applied (row constraint mapping rule 1). For Author* and Telephone*, rules for preserving the cardinality constraints are applied.

Finally, References has only one complex child element: Reference; so References is converted to an object table and Reference is converted

to an abstract data type field (Structure mapping Rule 1). Reference has attributes: RefID, RefType and an element: Publication with occurrence + whose attribute is rlink referencing back to the Publication element in the document which cites it; thus RefID and RefType are converted to simple fields (Structure mapping Rule 5) and not null is applied (row constraint mapping rule 1). RefType has a default value and a choice of attribute values; so the row constraint mapping rules 3-4 are applied and RefID is set as a primary key (table constraint mapping rule 1). Publication contains a recursive rlink; hence a separate table is created and this separate table consists of the primary keys of Reference and Publication set as a combined key and foreign keys. The foreign key derived from Reference is defined with a delete cascade (structure mapping rule 3 and constraint mapping rule 3 of rlink are applied). For Reference*, rules for preserving cardinality constraint are applied.

We create a schema derived from our rules in Oracle9i [24] and assume that the length of fields which are primary keys and foreign keys is 15 characters whereas the length of other fields is 30 characters. The schema generated in Oracle9i by using our rules is shown in Figure 9.

Mapping linked XML documents makes it easier to perform joins between XML documents and to update several linked XML documents in an update command; for example, from Figure 6, if we want to update title and author's email of publication having PubID = 'P111' and author having FName = 'John', an update command may be

```
Create Type TPublication as object
( PubID varchar2(15),
PubType varchar2(30),
Title varchar2(30),
Year varchar2(30),
RefID varchar2(15)
);

Create Type TPublications as object(
Publication TPublication
);

Create Table Publications of TPublications(
Primary key (Publication.PubID),
Publication DEFAULT
TPublication(null, 'book', null, null, null),
CHECK (Publication.PubType IN
('book', 'article', 'journal')),
CHECK (Publication.Title is not null and
Publication.Year is not null),
Foreign Key(Publication.RefID) references
References(Reference.RefID)
);

Create Type TName as object(
FName varchar2(30),
MName varchar2(30),
LName varchar2(30)
);

Create Type TTelephone as object(
Location varchar2(30),
TelNo varchar2 (30)
);

Create Type NTTelephone as table of TTelephone;

Create Type TAuthor as object(
AuthorID varchar2(15),
Name TName,
Email varchar2(30),
Telephone NTTelephone
);

Create Type TAuthors as object (
Author TAuthor
);
```

```
Create Table Authors of TAuthors(
Primary key (Author.AuthorID),
Check (Author.Name is not null),
Check (Author.Name.FName is not null
and Author.Name.LName is not null)
)
nested table Author.Telephone STORE AS
Telephone_TAB(
(CHECK (Location is not null),
CHECK (TelNo is not null))
);

Create Table PublicationAuthor(
PubID varchar2(15) references
Publications(Publication.PubID) on delete cascade,
AuthorID varchar2(15) references
Authors(Author.AuthorID),
Primary key(PubID, AuthorID)
);

Create Type TReference as object(
RefID varchar2(15),
RefType varchar2(30)
);

Create Type TReferences as object(
Reference TReference
);

Create Table References of TReferences (
Primary key(Reference.RefID),
Check(Reference.RefType is not null),
Reference DEFAULT TReference
(null, 'References'),
CHECK(Reference.RefType IN
('References','Bibliography','Miscelleneous'))
);

/*To keep relationship of rlink recursion*/
Create Table ReferencePublication (
RefID varchar2(15) references References
(Reference.RefID) on delete cascade,
PubID varchar2(15) references
Publications (Publication.PubID),
Primary key (RefID, PubID)
);
```

Fig. 9. Schema generated in Oracle 9i.

For $p in doc("Publications.xml")//Publication,
$a in $p/Author → doc("Authors.xml")//Author
Where $p@PubID = "P111"
Replace $p/Title with <Title>Java2</Title>,
Replace $a/Email with <Email>au@Hill.com </Email>
Where $a/Name/FName="John"

The above update language can be translated into SQL as follows:

```
Update Authors A
Set A.Author.Email = 'au@Hill.com'
Where A.Author.AuthorID in
(Select A.Author.AuthorID
From Authors A, PublicationAuthor PA
Where PA.PubID = 'P111'
And PA.AuthorID = A.Author.AuthorID
And A.Author.Name.FName = 'John');

Update Publications P
Set P.Publication.Title = 'Java2'
Where P.Publication.PubID = 'P111';
```

8. Conclusion and Further Work

For the time being, work converting both structure and constraints of XML to ORDB cannot be conducted readily because of limited constraints in available object-relational DBMSs. Oracle supports nested tables but the referential integrity constraint cannot be defined on them. Informix supports Set, List and Row types but some constraints such as default constraint and domain constraint cannot be defined on Set, List and fields of Row type while PostgreSQL only supports arrays and constraints cannot be defined on individual elements of an array.

In our work, we map both structure and constraints of XML to ORDB with awareness of practicability in available technologies. However we use nested tables instead of set/list since most constraints can be defined on nested-tables except the referential integrity constraint. Usually, data in XML documents are stored redundantly. We therefore propose an alternative way for keeping non-redundant data in several separate documents. This involves a mechanism called 'rlink' to link data in the separate documents together and additional rules for mapping the 'rlink' mechanism to ORDB. Finally, we create the object-relational schema with constraints derived from our mapping rules in Oracle9i. Our contribution is that we find that mapping linked XML documents makes it easier to perform joins between XML documents and to update several linked XML documents in one update command as discussed in Section 7.

In further work, we will make an extension to XQuery for updating (linked) XML documents and then we will translate it into SQL. The translation will include linear and non-linear recursive update commands and a mechanism will be proposed for propagating the change in ORDB to the XML documents. We will also conduct a performance comparison between updating one XML document containing redundant data and updating linked XML documents containing non-redundant data. This work is currently under development.

References

[1] S. ABITEBOUL, Querying Semi-Structured Data. *The International Conference on Database Theory*, Delphi, Greece, (1997), pp. 1–18.

[2] S. ABITEBOUL, D. QUASS, J. MCHUGE, J. WIDOM AND J.L. WINER, The Lorel query language for semistructured data. *Proceedings of International Journal on Digital Libraries*, (1997), pp. 68–88.

[3] *M. Arenas and L. Libkin*, A Normal Form for XML Documents. *Proceedings of the 21th Symposium on Principles of Database Systems (PODS)*, (2002), pp. 85–96.

[4] M. CAREY, D. FLORESCU, Z. IVES, Y. LU, J. SHANMUGASUNDARAM, E. SHEKITA AND S. SUBRAMANIAN, XPERANTO: *Publishing Object-relational Data as XML*. WebDB, Dallas, Texas, USA, (2000), pp. 105–110.

[5] R.G.G. CATTELL AND D.K. BARRY, The Object Data Standard: *ODMG 3.0. Morgan Kaufmann Publishers*, (2000).

[6] S. CERI, S. COMAI, E. DAMIANI, P. FRATERNALI, S. PARABOSCHI AND L. TANCA, XML-GL: a Graphical Language for Querying and Restructuring WWW Data. Computer Networks: *The International Journal of Computer and Telecommunications Networking*, 31 (1999), pp. 1171–1187.

[7] D. CHAMBERLIN, XQuery: An XML query language. *IBM SYSTEMS JOURNAL*, 41 (2002), pp. 597–615.

[8] D. CHAMBERLIN, J. ROBIE AND D. FLORESCU, Quilt: An XML Query Language for Heterogeneous Data Sources. *Int'l Workshop on the Web and Databases (WebDB)*, Dallas, TX, (2000).

[9] Y. CHEN, S. DAVIDSON, C. HARA AND Y. ZHENG, RRXS: Redundancy reducing XML storage in relations. *Proceedings of the 29th VLDB Conference*, Berlin, Germany, (2003).

[10] A. DEUTSCH, M. FERNANDEZ AND D. SUCIU, *Storing Semistructured Data with STORED*. SIGMOD Conference, Pennsylvania, United States, (1999), pp. 431–442.

[11] M. FERNANDEZ, Y. KADIYSKA, D. SUCIU, A. MORISHIMA AND W. TAN, SilkRoute: *A Framework for Publishing Relational Data in XML*. ACM Transactions on Database Systems (2002), pp. 1–55.

[12] D. FLORESCU AND D. KOSSMANN, (1999) A Performance Evaluation of Alternative Mapping Schemes for Storing XML Data in a Relational Database. *Rapport de Recherche* No. 3684.

[13] D. FLORESCU AND D. KOSSMANN, Storing and Querying XML Data using an RDBMS. *IEEE Data Engineering Bullentin*, 22 (1999), pp. 27–34.

[14] W. HAN, K. LEE AND B.S. LEE, An XML Storage System for Object-Oriented/Object-Relational DBMSs. *Journal of Object Technology*, 2 (2003), pp. 113–126.

[15] E.R. HAROLD, XInclude. XML 1.1 Bible. *Wiley Publishing*, Inc., (2004), pp. 657.

[16] E.R. HAROLD, XLinks. XML 1.1 Bible. *Wiley Publishing*, Inc., (2004), pp. 580.

[17] IBM. and Informix.: `http://www-306.ibm.com/software/data/informix/pubs/ //library/datablade/dbdk/start.htm`. 2004.

[18] L. KHAN, Q. CHEN AND Y. RAO, A Comparative Study of Storing XML Data in Relational and Object-Relational Database Management Systems. *Proc. of International Conference on Internet Computing*, Las Vegas, Nevada, (2002), pp. 277–282.

[19] L. KHAN AND Y. RAO, *A Performance Evaluation of Storing XML Data in Relational Database Management Systems*. ACM (2001).

[20] M. KLETTKE AND H. MEYER, *Managing XML Documents in object-relational databases*. Computer Science Department, University of Rostock, Rostock, Germany, (1999).

[21] D. LEE AND W.W. CHU, Constraints-Preserving Transformation from XML Document Type Definition to Relational Schema. *19th International Conference on Conceptual Modeling*, Salt Lake City, Utah, USA., (2000), pp. 323–338.

[22] D. LEE AND W.W. CHU, CPI: Constraints-Preserving Inlining Algorithm for Mapping XML DTD to Relational Schema. *Data & Knowledge Engineering*, 39 (2001), pp. 3–25.

[23] *Y. Mo and L.T. Wang*, Storing and Maintaining Semistructured Data Efficiently in an Object-Relational Database. *The Third International Conference on Web Information Systems Engineering*, Singapore, (2002), pp. 247–256.

[24] Oracle: `http://otn.oracle.com/documentation/index.html`. 2004.

[25] PostgreSQL: `http://www.postgresql.org`. 2005.

[26] J.W. RAHAYU, E. PARDEDE AND D. TANIAR, On Using Collection for Aggregation and Association Relationships in XML Object-Relational Storage. *ACM Symposium on Applied Computing*, Nicosia, Cyprus, (2004).

[27] K. RUNAPONGSA AND J.M. PATEL, Storing and Querying XML Data in Object-Relational DBMSs. *EDBT Workshops*. Publisher: Springer-Verlag Heidelberg, 2490 / 2002 (2002), pp. 266–285.

[28] J. SENG, Y. LIN, J. WANG AND J. YU, An analytic study of XML database techniques. *Industrial Management & Data Systems*, 103 (2003), pp. 111–120.

[29] B. SHAMKANTE AND S. NAVATHE, A Proposal for an XML Data Definition and Manipulation Language. *VLDB Conference*, Hongkong, (2002).

[30] J. SHANMUGASUNDARAM, E. SCHEKITA, R. BARR, M. CAREY, B.G. LINDSAY, H. PIRAHESH AND B. REINWALD, Efficiently publishing relational data as XML documents. Proceedings of the Conference on Very Large Data Bases, (2000).

[31] J. SHANMUGASUNDARAM, K. TUFTE, G. HE, C. ZHANG, D. DEWITT AND J. NAUGHTON, Relational Databases for Querying XML Documents: Limitations and Opportunities. *Proceedings of the 25th VLDB Conference*, Edinburgh, Scotland, (1999), pp. 302–314.

[32] T. SHIMURA, M. YOSHIKAWA AND S. UEMURA, Storage and Retrieval of XML Documents Using Object-Relational Databases. *IPSJ Transactions on Databases Abstract*, 40 (2001).

[33] I. TATARINOV, Z. IVES, A.Y. HALEVY AND D.S. WELD, Updating XML. *Proceedings of 2001 SIGMOD Conference*, Santa Barbara, CA, USA., (2001), p. 413-424.

[34] I. TATARINOV, S.D. VIGLAS, K. BEYER, J. SHANMUGASUNDARAM, E. SHEKITA AND C. ZHANG, Storing and Querying Ordered XML Using a Relational Database System. *Proceedings of the 2002 ACM SIGMOD international conference on Management of data*, Madison, Wisconsin, (2002), pp. 204–215.

[35] F. TIAN, D. DEWITT, J. CHEN AND C. ZHANG, *The Design and Performance Evaluation of Alternative XML Storage Strategies*. SIGMOD Record, 31 (2002).

[36] F.S.C. TSENG AND W. HWUNG, An automatic load/extract scheme for XML documents through object-relational ropositories. *The Journal of Systems and Software*, 64 (2002), pp. 207–218.

[37] C. TURKER AND M. GERTZ, Semantic integrity support in SQL:1999 and commercial (object-)relational management systems. *The VLDB Journal*, 10 (2001), pp. 241–269.

[38] UniSQL/X: UniSQL/X User's Manual Vol I. 2004: `http://dev.unisql.com/dev/manuals/manuals.htm`.

[39] I. VARLAMIS AND M. VAZIRGIANNIS, Bridging XML-Schema and relational databases. A system for generating and manipulating relational databases using valid documents. *ACM Symposium on Document Engineering* (2001), pp. 105–114.

[40] W3C: XML Inclusions (XInclude) Version 1.0. Candidate Recommendation. 2004: `http://www.w3.org/TR/2004/CR-xinclude-20040413/`.

[41] W3C: XML Linking Language (XLink) Version 1.0. Recommendation. 2001: `http://www.w3.org/TR/xlink`.

[42] R.K. WONG, The Extended XQL for Querying and Updating Large XML Databases. *ACM Symposium on Document Engineering*, (2001), pp. 95–104.

[43] XMLDB: XUpdate. 2002: `http://www.xmldb.org/xupdate/xupdate-wd.html`.

Received: October, 2004
Revised: August, 2005
Accepted: September, 2005

Contact address:
Pensri Amornsinlaphachai
School of Computing, Engineering & Information Sciences
Northumbria University
Pandon Building (Room 113), Camden Street,
Newcastle upon tyne, NE2 1XE, UK.
e-mail: `pensri.amornsinlaphachai@unn.ac.uk`

PENSRI AMORNSINLAPHACHAI is a Ph.D. student at School of Computing, Engineering & Information Sciences, Northumbria University, Newcastle, UK. She received her MSc. with Distinction in 2001 and the reward Sun Certified Programmer For THE JAVA 2 in 2002.

DR. NICK ROSSITER is a reader at School of Computing, Engineering and Information Sciences, Northumbria University, Newcastle, UK. He is interested in interoperability of information systems.

DR. M. AKHTAR ALI is a senior lecturer at School of Computing, Engineering and Information Sciences, Northumbria University, Newcastle, UK. In 2003 he received his Ph.D. from Manchester University.