Handling XML in Traditional Databases

Kwanjai Deejring and Pensri Amornsinlaphachai

Nakhon Ratchasima Rajabhat University, Thailand
E-mail: kdeejring@yahoo.com, kokkoy@hotmail.com

Abstract

Most research in the XML area has concentrated on storing, querying and publishing XML while not many researchers have paid attention to updating XML; thus, the XML update area is not fully developed. This work contributes a solution for the update of XML documents via ORDB (Object-Relational Database) to advance the techniques in this area through preserving constraints, maintaining performance in the presence of data redundancy, permitting joins of documents in updates and allowing the updates of documents whose structure is known partially or whose structure is recursive. Experimental study to evaluate the performance of XML update processing has been conducted. The experimental results show that updating multiple XML documents storing non-redundant data yields a better performance than updating a single XML document storing redundant data; an ORDB can take advantage of this by caching data to a greater extent than a native XML database.

Keywords: XML updates, ORDB, XML constraints

1. Introduction

XML has become an effective standard for representing semi-structured data on the Web since it provides a natural data structuring mechanism for hierarchical and recursive data; moreover it is flexible in that it allows the authors to define their own tags and structure for documents and can handle data whose occurrence is optional. Many researchers in the XML area have focused on storing, publishing, and querying XML documents. XML consequently provides most of the features normally expected for a database model. However, there is an omission in that most existing work does not pay much attention to modifying XML or does not mention it at all.

One possible reason behind the immaturity of the XML update area is as follows. XQuery has not provided update features because the W3C Consortium wanted to release the standard of XQuery as soon as possible [12]. Thus, only a few researchers have paid much attention to this area. Our work has identified five main problems as follows:

- The work published presently can update XML documents but only without checking constraints. Even commercial products cannot guarantee the integrity the database when XML data is updated [8].
- Normally, all XML data is kept in one document; thus data redundancy may occur. This can lead to data inconsistency and low performance when updates are performed.
- No XML update language supports joins of XML documents.
- Regular path expressions are used to query/update XML whose structure is unknown or only partially known. Using regular path expressions, especially a descendent path expression ('//'), can slow the process of querying/updating data [47] because the query engine must traverse all possible paths in XML.
- In XQuery, there is no specific facility to query data whose structure is recursive; however the effect can be achieved by creating a recursive user-defined function. Until now, no technique has been proposed to translate this recursive feature into SQL.

The rest of this paper is organized as follows. Section 2 presents the goal of our work, to devise a more effective solution for updating XML data and solve the open problems as mentioned previously. Section 3 shows the results of experimental study, including performance aspects. Related work is covered in Section 4. A conclusion is provided in Section 5.

2. Our Solution for Updating XML

Nowadays, there are two dominant approaches for managing XML repositories. The first approach is to use native XML databases to handle the data. The second approach maps XML onto a traditional database (e.g., relational database (RDB), ORDB and object-oriented database (OODB)).

XML updating has been relatively well-researched in the area of native XML databases, whereas in the area of applying traditional databases to manage XML, only one work [42] has presented an XML language for updating XML data. This work employed an RDB, but only the syntax and semantics of the language are presented. In our solution, the more advanced technology of ORDB is exploited to update XML documents.

The purpose of using a traditional database, ORDB, in this research is different from that of other work. The previous work uses OODB [1, 52], RDB [13, 23] and ORDB [24, 32] as DBMS of XML documents to store and query XML data but our approach uses ORDB to preserve constraints during updating and to indicate the target-elements in XML documents which should be updated. The updates are performed on XML documents; thus users can query data from XML documents instead of ORDB and it is not necessary to maintain the order of elements in ORDB. This reduces the cost of data conversion, since nowadays, the major expense of exchanging messages between Web Services comes from converting data such as between a database and XML format [48]. The overview of our approach is illustrated in Figure 1.

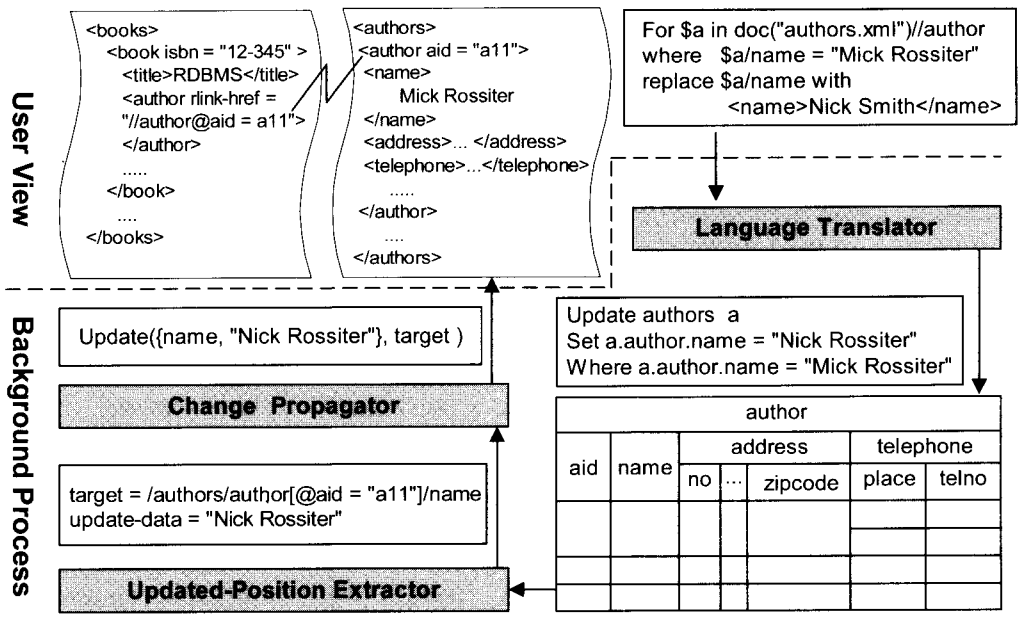

Figure 1. Overview of the solution to updating XML documents

In the solution, DTDs are used in our mapping since most XML documents still stick to DTDs [31]. Not only XML structure but also XML constraints are mapped to ORDB, since a DTD defines the constraints on the logical structure of XML documents [27].

Non-redundant data is kept in separate multiple XML documents, avoiding the storage of redundant data in one single XML document; then the separate documents are linked together. To update XML data, an XML update language, as an extension to XQuery, is proposed, and this language is translated into SQL to update XML data stored in ORDB. Then the changes in ORDB are propagated to the XML documents. More details of the overview of the solution can be found in [4]. In this section, an XML update language, translating the XML update language into SQL and propagating the change from ORDB to XML will be proposed, whereas mapping XML to ORDB is presented in [5].

2.1 Updating XML documents

We design an XML update language, an extension to XQuery, and then translate this language into SQL. Six major features of the XML update language inherited from XQuery will be translated: FLW(R|I|D) expression, conditional expression, quantifier, aggregate functions, non-recursive user-defined function and recursive function.

In this section, firstly the syntax of the update language is presented. Secondly, the technique for translating the first five features is presented. For the recursive function, its translation is presented in [6]. Our rules can be applied to RDB presented in [6] too.

2.1.1 The Syntax of the XML update language

The syntax of our XML update language is adapted from [42] and that of XQuery from [45]. The syntax is shown in Figure 2.

```
(ForClause | LetClause)+
WhereUpdateClause | IfUpdateClause
where each clause is:
ForClause          ::= For $var in XPath(,$var in XPath)
LetClause          ::= Let $var := XPath(,$var := XPath)
WhereUpdateClause ::= WhereClause? UpdateClause
WhereClause        ::= Where Condition
UpdateClause       ::= DeleteClause | ReplaceClause | InsertClause
DeleteClause       ::= Delete node WhereClause? (, Delete node WhereClause?)
ReplaceClause      ::= Replace node with content WhereClause?
                       (, Replace node with content WhereClause?)
InsertClause       ::= Insert content Into node WhereClause? (Before|After condition basedon_XPath)?
                       (,Insert content Into node WhereClause? (Before|After condition basedon_XPath)?
IfUpdateClause     ::= If Condition Then UpdateClause
                       (ElseIf Condition Then UpdateClause)  (Else UpdateClause)?
```

Figure 2. The Syntax of the XML update language

2.1.2 Four techniques for translating the XML update language

Our translation uses four main techniques: 1) update/delete join commands, 2) rewriting rules, 3) graph mapping and 4) optimization rules for translating the XML update language into SQL. The first three techniques are given below. The optimization rules are presented in [6].

Update/delete join commands

The translation of XML update commands can produce joins of several tables. However, in the SQL standard, update/delete join commands cannot be performed. Therefore, we translate the update commands into update/delete join commands and then rewrite them as SQL with sub-query commands. The syntax of update/delete join commands is as follows.

Syntax of joins in update command:

Update *table whose data will be updated*
From *all related tables*
Set *field1 = value1, field2 = value2, ...*
Where *Condition*;

Syntax of joins in delete command:

Delete *table whose data will be deleted*
From *all related tables*
Where *Condition*;

Rewriting Rules

There are seven categories of rewriting rules, one for each major feature of the XML update language as itemized above, one for SQL and one for the rlink. The categories are therefore: FLW(R|I|D) expression, aggregate function, quantifier, conditional expression, (non-recursive) user-defined function, SQL and rlink rewriting rules. In this section, we describe first the SQL functions followed by the seven categories of rewriting rules.

I) SQL Functions

In the translation of the language, all clauses of XML update commands must be rewritten as SQL functions which are conceptual

functions representing the operations of SQL commands. The SQL functions are used to group update clauses and their conditions together since one XML update command can consist of several update clauses and each update clause can have its own condition. These clauses are grouped by using function number (funcNo) which is a parameter of every SQL function. The funcNo 0 is assigned to ForClause, LetClause and WhereClause of the ForClause and LetClause. The clauses with funcNo 0 are shared clauses for UpdateClause. Each update clause has its own funcNo, a running number starting from 1. The update clause and its own condition will have the same funcNo. The SQL functions are as follows:

1. bindF(path, $var, funcNo)
2. bindL(path, $var, funcNo)
3. insert (node, value | :funcNo, funcNo)
4. delete(node, funcNo)
5. update(node, value | :funcNo, funcNo)
6. group_by(node, funcNo)
7. select(node, funcNo)
8. aggFunc(node, funcNo)
   where aggFunc ::=max|min|count|avg|sum
9. where|LogicalOper(node, CompareOper, value | :funcNo, funcNo)
10. having(aggFunc(node),CompareOper,value | :funcNo, funcNo)

Four SQL functions, insert(), update(), where|LogicalOper() and having(), have the parameter *value | :funcNo* since sometimes the value in inserting, in updating or in the predicate is not a constant value, but may come instead from selecting a value from other nodes. Hence in this case, funcNo has the same number as that for the funcNo of the select() function. The symbol ':' is used to differentiate the funcNo of an SQL function from funcNo, which is the value-parameter.

II) Rewriting rules for FLW(R|I|D)

For all rewritng rules, we use the symbol '- - >' stands for 'is rewritten as' and used the symbol '- - >>' stands for 'is translated into' The expression FLW(R|I|D) will be rewritten as SQL functions as follows:

1. For $var in XPath - - >
   bindF(XPath, $var, funcNo)
2. Let $var := XPath - - >
   bindL(XPath, $var, funcNo)
3. Where predicate - - >
   where(node, CompareOper, value|:funcNo, funcNo)
4. LogicalOper predicate - - >
   LogicalOper(node, CompareOper, value|:funcNo, funcNo)
5. For $var in $XPath_{predicate}$ - - >>
   For $var in XPath Where predicate

   Then this clause is rewritten as SQL functions according to rules 1, 3, 4.
6. Let $var := $XPath_{predicate}$ - - >>
   Let $var := XPath Where predicate

   Then this clause is rewritten as SQL functions according to rules 2-4.
7. Select *node* | Return *node* - - >
   select(node, funcNo)
8. Replace *node* with *simple content* - - >
   update(node, content's value, funcNo)
9. Delete *node* - - > delete(node, funcNo)
10. Insert *simple content* into *node* - - >
    insert(node, content value, funcNo)
11. Replace *node* with *complex content* and Insert *complex content* into *node*

    The *complex content* is shredded into many simple contents. Then the command is rewritten in the form of the commands based on the simple contents which are in turn rewritten as SQL functions.

III) Rewriting rules for aggregate functions

1. Define: For $var1 in XPath1
   Let $var2 := $var1/XPath2
   i): aggFunct($var2) - - >
   aggFunct($var2, funcNo)
   group_by($var1, funcNo)

   ii): Where aggFunc($var2)
   CompareOper value - - >
   group_by($var1, funcNo)
   having(aggFunct($var2), CompareOper, value, funcNo)

2. Define: Let $var := XPath
   i): aggFunct($var) - - >
   aggFunct($var, funcNo)

IV) Rewriting rules for quantifier

There are two quantifiers: existential quantifier (*some*) and universal quantifier (*every*). Both can be translated into a count() function since *some* is used to test whether at least one item in a sequence satisfies the condition while *every* is used to test whether every item in a sequence satisfies the condition; thus their meanings will first be translated and then rewritten as SQL functions as follows:

1. For $var1 in XPathExp1
   Where some $var2 in $var1/XPathExp2
   Satisfies (Condition) - - >>

   For $var1 in XPathExp1
   Let $var2 := $var1/XPathExp2
   Where count($var2) > 0
   And Condition - - >

   $var1 = bindF(XPathExp1, funcNo)
   $var2 = bindL($var1/XPathExp2, funcNo)
   where (node, CompareOper,
   value | :funcNo, funcNo)
   (LogicalOper(node,CompareOper,
   value | :funcNo, funcNo))
   group_by($var1, funcNo)
   having(count($var2), >, 0, funcNo)

2. For $var1 in XPathExp1
   Where every $var2 in $var1/XPathExp2
   Satisfies (Condition1)
   [And Condition2] - - >>

   For $var1 in XPathExp1
   Let $var2 := $var1/XPathExp2
   Where Condition1
   [And Condition2]
   And count($var2) =
   (For $var3 in XPathExp1
   Let $var4 := var3/XPathExp2
   Where $var3 = $var1
   [And Condition2]
   Return count($var4)
   - - >

   $var1 = bindF(XPathExp1, funcNo)
   $var2 = bindL(var1/XPathExp2, funcNo)
   where (node,CompareOper,
   value | :funcNo, funcNo)
   (LogicalOper (node, CompareOper,
   value | :funcNo, funcNo))
   [and(node,CompareOper,
   value|:funcNo, funcNo)
   (LogicalOper(node,CompareOper,
   value|:funcNo, funcNo))

   group_by($var1, funcNo)
   having(count($var2), =, :1, funcNo)
   $var3 = bindF(XPathExp1, :1)
   $var4 = bindL($var3/XPathExp2, :1)
   select(count($var4), :1)
   where ($var3, =, $var1, :1)
   [and(node, CompareOper, value, :1)
   (logical_operator(node, CompareOper,
   value, :1))

   group_by($var3, :1)

Besides 'some' and 'every' quantifiers, there are two functions: empty() and exists() which can be rewritten as count() functions. These functions and quantifiers can be used along with 'not'. To summarise, the meaning of these functions and quantifiers can be translated before rewriting as follows:

empty -->> $count_{predicate} = 0$

exists -->> $count_{predicate} > 0$

some -->> $count_{predicate} > 0$

not (empty) -->> $count_{predicate} > 0$

not (exists) -->> $count_{predicate} = 0$

not (some) -->> $count_{predicate} = 0$

every -->> $count_{predicate} = count_{without\ predicate}$

not (every) -->>
$count_{predicate} < count_{without\ predicate}$ and $count_{predicate} > 0$

V) Rewriting rules for conditional expression

(ForClause|LetClause)+
If (Condition $_1$) then
UpdateStm$_1$
Else If (Condition $_2$) then
UpdateStm$_2$
.....

Else [If (Condition $_n$)]
UpdateStm$_n$
-->>

(ForClause|LetClause)+
Where Condition $_1$
UpdateStm$_1$
(ForClause|LetClause)+

Where Condition $_2$
And ¬(Condition $_1$)
UpdateStm$_2$
.....

(ForClause|LetClause)+
[Where condition $_n$]
Where|And ¬ (condition $_1$)
And ¬ (condition $_2$)
.....
And ¬ (condition $_{n-1}$)
UpdateStm$_n$

The series of commands is then rewritten as SQL functions according to the category of clauses in the commands. The number of commands in the series corresponds to the number of conditions in if-then-else. The symbol ¬ stands for 'not'.

VI) Rewriting rules for non-recursive user-defined function

Calls to non-recursive functions are replaced with the body of such functions and parameters of the function are replaced with the values of arguments. After such replacements, the update command is rewritten as SQL functions according to the category of clauses in the command.

VII) Rewriting rules for the rlink

Define: rlinkEle is the rlink-element containing *@rlink-href* whose value is XPath

parentEle is the parent of rlinkEle

linkedEle is the element referenced by XPath which is the value of *@rlink-href*

1. *Where* rlink-element

If there is only one predicate in XPath based on PK of linkedEle as follows:
PK of linkedEle = value

Then Where parentEle/rlink(rlinkEle, XPath)
--> where(parentEle/rlinkEle#linkedEle, =, value, funcNo)

Else Where parentEle/rlink(rlinkEle, XPath)
--> where(parentEle/rlinkEle#linkedEle, =, :funcNo2, funcNo1)
select(PK of linkedEle, funcNo2)
where(condition in XPath, funcNo2)

2. *Insert* rlink-element

If there is only one predicate in XPath based on PK of linkedEle as follows:
PK of linkedEle = value

Then Insert rlink(rlinkEle, XPath)
Into parentEle -->
insert(parentEle/rlinkEle#linkedEle, value, funcNo)

Else Insert rlink(rlinkEle, XPath)
Into parentEle -->
insert(parentEle/rlinkEle#linkedEle, :funcNo2, funcNo1)
select(PK of linkedEle, funcNo2)
where(condition in XPath, funcNo2)

3. *Replace* rlink-element is translated into a sequence of *delete* and *insert* rlink-element with the sequence rewritten as SQL functions according to rules 1-2:

Replace parentEle/rlinkEle with rlink(rlinkEle, XPath1)
[Where parentEle/rlink(rlinkEle, XPath2)]
->>
Delete parentEle/rlinkEle
[Where parentEle/rlink(rlinkEle, XPath2)],
Insert rlink(rlinkEle, XPath1) Into parentEle

Then *Where* and *Insert* clauses are rewritten according to rules 1-2.

VIII) SQL rewriting rules

Rules are used to rewrite joins in update commands and joins in delete commands as SQL with sub-query commands. The rewriting rules for joins in update commands and in delete commands are shown in Figures 3 and 4 respectively.

Graph Mapping

The purpose of graph mapping is to indicate the SQL functions performed on tables, (column of) nested table, (column of) abstract data type or simple columns of the table.
The steps for graph mapping start from creating a graph whose paths correspond to paths in the SQL functions. Then the graph is mapped to the ORDB schema to identify the ORDB structure on the graph. The keys for joins of tables and join symbols are then added to the graph and the SQL functions are mapped to the graph. Next, the actions, pushing the function down to proper nodes of the graph and changing the meaning of update operations, are performed according to the following rules.

Update-rule 1: If a *delete* or *insert* function is performed on nodes converted to ADTs having siblings, fields of an ADT, fields of a table or fields of a nested table, without either a *delete* or *insert* function on an ancestor-node converted to a table or an ADT without a sibling, then the function will be converted to an *update* function.
Update-rule 2: If an *insert* function is performed on a node converted to the primary key of a table, then this *insert* function must also be applied to the foreign key of the child-tables to maintain parent-child relationships.
Update-rule 3: If a *select*, *where* or *group_by* function is performed on a node converted to a table or an ADT without a sibling, then the function will be pushed down to the primary key of the table.

The graph may then be split into several sub-graphs. The number of sub-graphs is equal to the number of update operations performed on the tables, which have a distinct function number. Finally, optimization rules [6] are applied to the (sub-)graphs and SQL commands or update/delete join commands are generated from the (sub-)graphs. If the generated commands are in the form of update/delete join commands, the commands are rewritten according to the SQL rewriting rules.

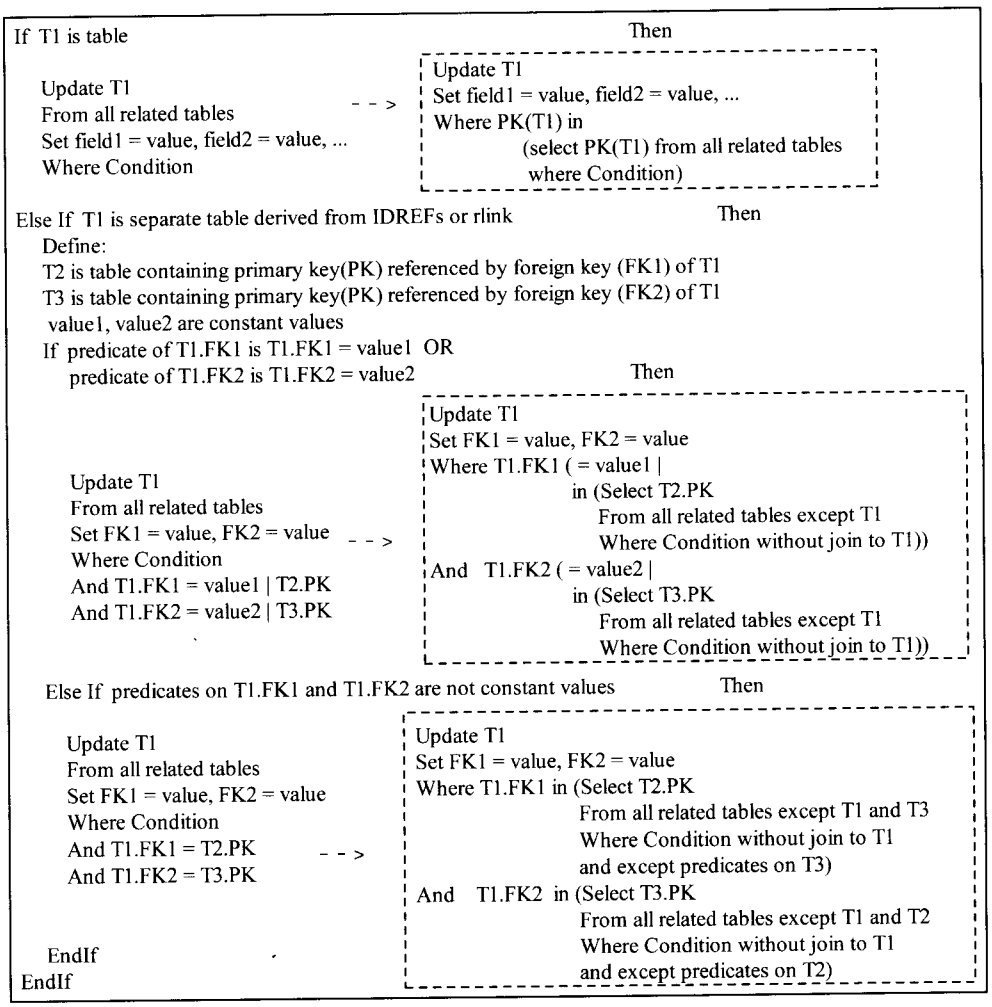

Figure 3. Rewriting rules for joins in update commands

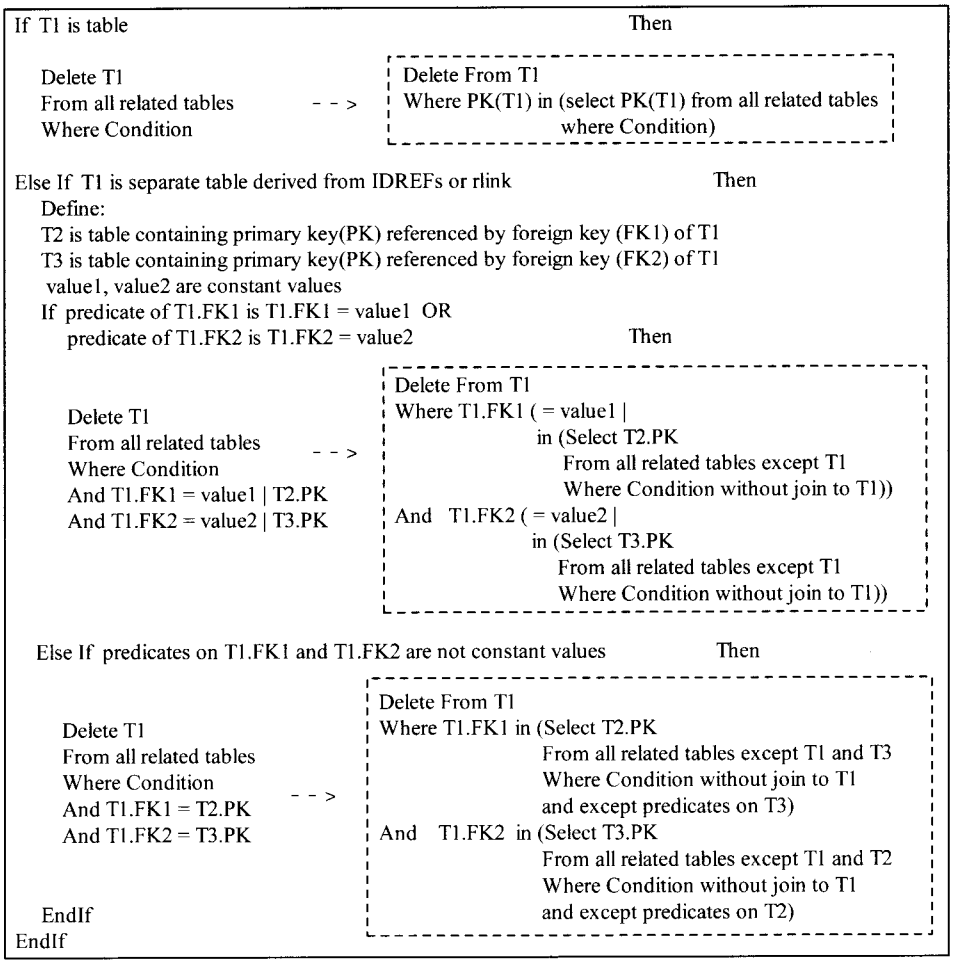

Figure 4. Rewriting rules for joins in delete commands

2.2 Propagating the changes to XML

Propagation of the changes from ORDB to XML documents is performed only on the parts affected by updating. We use values of a primary or foreign key of updated data in the database to indicate the elements in XML documents which should be updated. When data in the ORDB is updated, the values of a primary or foreign key of updated data will be returned. Subsequently paths in the XML update command are used to create XPath expressions whose conditions are based on these returned values to indicate the targets or reference positions in XML documents for updating. XPath has no capability for updating. Hence we propose five propagate functions, shown in Figure 5, to serve as operators for updating XML documents.

Propagate Functions	Description
Insert (nodeLst, targetLst)	Inserting nodeLst into nodes in targetLst
InsertBefore(nodeLst, targetLst)	Inserting nodetLst before nodes in targetLst
InsertAfter(nodeLst, targetLst)	Inserting nodeLst after nodes in targetLst
Delete (targetLst)	Deleting nodes in targetLst
Update (nodeLst, targetLst)	Replacing values of nodes in targetLst with values of nodes in nodeLst

Figure 5. Propagate functions

From the functions in Figure 5, the parameter *nodeLst* can be derived directly from the XML update command. The parameter *targetLst* is an XPath expression. The path in the XPath is derived from a path in the XML update command.

Locating position for updating

The nodes in XML documents which should be updated can be identified by an XPath expression whose condition is based on the values for keys returned from ORDB and some predicates derived directly from the XML update command. Such predicates are those of *before* and *after* clauses and on IDREFs, rlink-elements and elements/attributes converted to fields of nested tables. To determine which values for a key should be returned, the types of ORDB structure and types of update operations performed on that ORDB structure are examined. A summary of operations performed on ORDB and values for the key returned from ORDB, when an XML update command is performed on each XML structure type, is shown in Table 1.

The details of the eight columns in Table 1 are as follows.

- Column 1 shows the type of update operation performed on XML documents.
- Column 2 shows the type of XML structure on which the update operation in column 1 is performed.
- Column 3 shows the type of ORDB structure converted from the XML structure in column 2.
- Column 4 shows the type of update operation performed on ORDB.
- Column 5 shows the keys for the values which are returned from the ORDB.
- Column 6 shows which ID in XML documents is the receiver for the returned values for the keys in column 5. The receiver and the returned values are used to compose a key-condition for the XPath expression.
- Column 7 shows the conditions used along with the key-condition for the XPath expression.
- Column 8 shows the element or attribute which is the target or reference position for updating. This position is retrieved by the XPath expression.

3. Experimental study

It is important to verify the method for updating XML, developed in section 2. This was done through an experimental study in which a diverse range of 17 update queries were executed and the results carefully inspected to check that they were as expected. In addition, to gain an insight into the performance of the update techniques, runs were repeated with variable database size, cache state, degree of redundancy and methods for linking XML structures.

3.1 Experiment platform and methodology

In the experiments, three types of databases are used. The first is X-Hive, a commercial native XML database (nxd), used to keep redundant data of a single XML document. The second is Oracle ORDB employed to keep redundant data of a single XML document (sxd). The third is Oracle ORDB utilised to keep non-redundant data of linked XML documents (lxd). All experiments are conducted on 1.3 GHz Pentium M machine with 768 MB main memory and 20 GB disk running Windows XP. The experiments are designed to evaluate the performance for updating: (a) native XML database storing redundant data, (b) ORDB storing redundant XML data and (c) ORDB keeping non-redundant XML data. Varying sizes of the databases ranging from 5, 10, 20 to 40 MB are used. The number of redundant records for 5 MB data size varies from 10, 20, 40 to 80 records while the number of redundant records for 10, 20 and 40 MB data size is two, four and eight times, respectively (the number of redundant records for the 5 MB data size). In updating the linked XML documents, each update command affects 10 records. Thus the number of records in a single document affected by a command varies according to the proportion of redundant records.

To study the effect of data caching on the performance of updating XML, the experiments are conducted in cold cache, warm cache and hot cache. In cold cache, the database is restarted for each individual update command. In warm cache, the database is restarted for each individual command as well; however before running the command, five unrelated commands will be run first. In hot cache, the same command is run twice in succession and the performance measured for the second run.

Table 1: Summary of update operations on XML and ORDB sides, keys for returned values, receivers, conditions and target of updating

On XML side		On ORDB side		Key-condition to locate target/reference position		Other conditions used along with key-condition	Target or reference position
Operation	XML structure	ORDB structure	Operation	Keys for values returned from ORDB	receiver for returned value		
replace	E or A	field, ADTF, ADT with sibling (E)*	update	PK of updated row	ID of nearest ancestor		E or A
	E or A	NT (E)* or NTF	update	PK of row containing NT	ID of nearest ancestor	conditions on fields of NT	
	E	ADT no sibling, table	update	PK of updated row	ID of E or nearest desc.		
	IDREFs	Separate table	update	1st PK of updated row	ID of nearest ancestor	condition on IDREFs	
	rlinkE	Separate table	del, ins	1st PK of deleted row	ID of nearest ancestor	condition on E@rlink-href	E@rlink-href
	rlinkE	FK	update	PK of updated row	ID of nearest ancestor		
delete	E or A	field, ADTF, ADT with sibling (E)*	update	PK of updated row	ID of nearest ancestor		E or A
	E	NT	delete	PK of row containing NT	ID of nearest ancestor	conditions on fields of NT	
	E or A	NTF	update	PK of row containing NT	ID of nearest ancestor	conditions on fields of NT	
	E	ADT no sibling, table	delete	PK of deleted row	ID of E or nearest desc.		
	IDREFs	Separate table	delete	1st PK of deleted row	ID of nearest ancestor	condition on IDREFs	
	rlinkE	Separate table	delete	1st PK of deleted row	ID of nearest ancestor	condition on E@rlink-href	
	rlinkE	FK	update	PK of updated row	ID of nearest ancestor		
Insert...into	E or A	field, ADTF, ADT with sibling (E)*	update	PK of updated row	ID of nearest ancestor		Parent of E/A
	E	NT	insert	PK of row containing NT	ID of nearest ancestor		
	E or A	NTF	update	PK of row containing NT	ID of nearest ancestor	conditions on fields of NT	
	E	ADT no sibling, table (not root)	insert	FK of inserted row	ID of nearest ancestor		
	E	ADT no sibling (root)	insert	none	none		
	IDREFs, rlinkE	Separate table	insert	1st PK of inserted row	ID of nearest ancestor		
	rlinkE	FK	update	PK of updated row	ID of nearest ancestor		
insert...into before/after	E	field, ADTF, ADT with sibling (E)*	update	PK of updated row	ID of nearest ancestor	condition of before/after	Sibling in before/after condition
	E	NT	insert	PK of row containing NT	ID of nearest ancestor	condition of before/after	
	E	NTF	update	PK of row containing NT	ID of nearest ancestor	conditions on fields of NT condition of before/after	
	E	ADT no sibling, table (not root)	insert	FK of inserted row	ID of nearest ancestor	condition of before/after	
	E	ADT no sibling (root)	insert	none	none	condition of before/after	
	IDREFs, rlinkE	Separate table	insert	1st PK of inserted row	ID of nearest ancestor	condition of before/after	
	rlinkE	FK	update	PK of updated row	ID of nearest ancestor	condition of before/after	

E: element, A: attribute, NT: nested table, NTF: nested table field, ADT: abstract data type, ADTF: field of ADT, rlinkE: element containing rlink mechanism, PK: primary key, FK: foreign key

(E)* : The ORDB structure can be converted from E only (not from A), (root): table of ADT is root element, (not root): table (of ADT) is not root element

We design commands covering the 17 features shown in Figure 6. Each experiment is executed five times and the longest and the shortest elapsed times are ignored; thus only an average of three elapsed times is reported.

C1 Exact Match	C10 Join documents in update
C2 Update without join of documents	C11 Navigation by reference traversal
C3 Change selectivity	C12 Handling missing elements
C4 Allow condition on text	C13 Element ordering
C5 Support aggregation	C14 Using regular path expression
C6 Support quantifiers	C15 Mix between data-centric and document-centric
C7 Joins based on values	C16 Hierarchical and sequential update
C8 Joins based on pointer	C17 Recursion and reference traversal
C9 Casting	

Figure 6. The 17 update features for the experimental study

3.2 Discussion of the experimental results

This section discusses the performance with different data redundancy and data caching. Figure 7 contains four graphs, one for each size of the database. Each graph plots average elapsed time for replace, delete and insert operations against the number of redundant records in the three possible cache states: cold, warm and hot.

Note that the update time in the graphs and the tables excludes serialization time since, usually, serialization can be performed only once after all updates finish.

The commands C14 (regular path expression) and C17 (recursion) are excluded from the average time of nxd since X-Hive does not support C17 while the elapsed time of the C14 run on X-Hive is so long that it can affect the overall performance of the systems.

From the graphs in Figure 7 in cold cache, sxd has the worst performance for every data size, whereas nxd has the best performance. However when the data size is 40 MB and there is considerable data redundancy, lxd can outperform nxd. This is because lxd does not contain redundant data; thus although the number of updated records will be constant for every degree of redundancy, the data size will be smaller when nxd has more redundant data. Therefore the performance of lxd is better when the degree of redundancy in nxd is greater.

For warm and hot caches, lxd has the best performance in all cases while nxd has the worst performance when the data size is smaller than 20 MB. When the data size is 20 and 40 MB, then nxd can outperform sxd. This is because the update time in sxd consists of SQL-time and DOM-time (time for updating DOM of XML) and when the data size is doubled, DOM-time is also approximately doubled whereas the update time of nxd is increased to a lesser extent. When the data size is increased, there are clearly more records to be updated. As sxd has a rollback mechanism, this means that more data has to be preserved for recovery purposes. nxd does not have such a mechanism so there is no penalty in performance here. So the performance for sxd is affected by redundant data more than that for nxd as can be seen clearly when the data size is bigger than 5 MB.

The time in cold cache for lxd is about four or five times that in warm cache, and the time in cold cache for sxd is double that in the warm cache. By contrast, the times in cold and warm caches for nxd are similar, showing that lxd and sxd gain more benefit from caching data than nxd. The difference in time between the cold and warm caches for lxd is more than that for sxd. This is because caching data has only a little effect on DOM-time but a much greater effect on SQL-time. For lxd, the DOM-time is small when compared to the SQL-time; thus, when the cache state is changed from cold to warm, most of the difference in time is attributed to the change in SQL-time, causing significant difference in the elapsed times for cold and warm caches. On the other hand for sxd, the DOM-time is nearly equal to the SQL-time in cold cache and the DOM-time is changed by only a little when the cache state is changed from cold to warm. Thus, the difference in elapsed time between cold and warm caches in sxd, is less than of the same case in lxd.

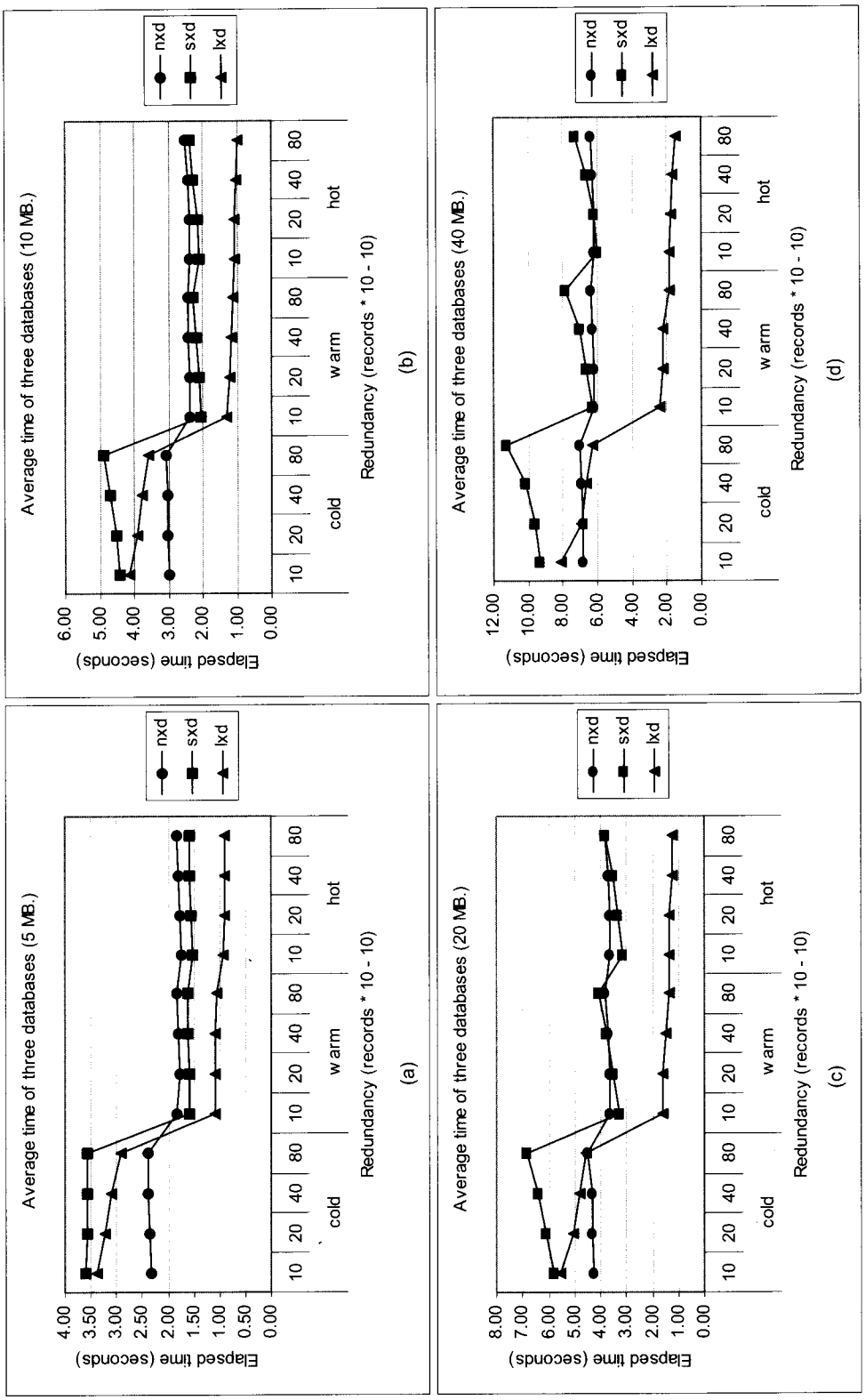

Figure 7. Average time of nxd, sxd and lxd in cold, warm and hot caches

4. Related work

The general steps for updating XML via the traditional databases are: (i) mapping XML schema to the database schema and (ii) translating an XML update language into SQL executed on the database. Our work updates XML via ORDB; hence related literature includes mapping XML to traditional databases, the languages for updating XML and the translation of the languages into SQL. Because of limited space, the detail of each work is omitted and we just summarize the comparison of mapping XML to the database in Table 2, comparison of XML Update Languages in Table 3, and Comparison of techniques for translating XML query into SQL in Table 4.

Table 2: Comparison of mapping XML documents to traditional database models

Research/Proposal	Model	Type of schema	Mapping method	Query Language	XML Update	Constraint Handling	Order kept	Recursion
Khan et al. [22]	Relational	DTD	Node	XPath	No	No	No	No
Psaila and Milano [36]	Relational	DTD	Shred	None	No	No	No	No
Shanmugasundaram et al. [40]	Relational	DTD	Shred	XML-QL Lorel	No	No	No	Yes
Zhou et al. [51]	Relational	DTD	Shred	None	No	No	No	Yes
Lv and Yan [28]	Relational	DTD	Shred	None	No	Partial	No	No
Bohannon et al. [10]	Relational	XML Schema	Shred	XQuery	No	No	No	Yes
Varlamis and Vazirgiannis [44]	Relational	XML Schema	Shred	DB. Command	No	Partial	No	No
Atay et al. [7]	Relational	DTD	Shred	None	No	No	Yes	Yes
Amer-Yahia et al. [3]	Relational	XML Schema	Shred Edge	XPath	No	No	Yes	Yes
Florescu and Kossmann [17]	Relational	None	Edge	None	No	No	Yes	Yes
Jiang et al. [21]	Relational	None	Edge	XPath	No	No	Yes	Yes
Manolescu et al. [30]	Relational	None	Edge	QUILT	No	No	No	Yes
Yoshikawa et al.[50]	Relational	None	Node	XPath	No	No	Yes	Yes
Deutsch et al. [13]	Relational	None	Data mining	User defined language	Yes	No	Yes	Yes
Jensen and Beitzel [9]	Relational	None	Node	XML-QL	No	No	No	Yes
Shimura et al. [41]	OR	None	Node	XQL	No	No	Yes	Yes
Klettke and Meyer [24]	OR	DTD	Shred	None	No	Partial	No	Yes
Runapongsa and Patel[37]	OR	DTD	Shred	SQL	No	No	Yes	Yes
Tseng and Hwung [43]	OR	DTD	Shred	None	No	No	No	No
Mo and Ling [32]	OR	DTD, XML Schema	Shred	None	No	Partial	No	No
Pardede et al. [34]	OR	XML Schema	Shred	None	No	Partial	No	No
Han et al. [19]	OO/OR	XML Schema	Shred	None	No	No	No	No
Abiteboul et al. [1]	OO	DTD	Shred	OQL extension	No	No	No	No
Fegaras and Elmasri [14]	OO	DTD	Shred	XML-OQL	No	No	No	No
Zwol et al. [52]	OO	DTD	Shred	Algebra	No	No	No	No

Table 3: Comparison of XML Update Languages

Features \ Language	XUpdate [49]	SixDML [33]	Lorel [2]	XML-GL [11]	XML-RL Update Language [26]	XML Update Extension [42]
Updating	elements	elements	nodes	nodes	elements	elements
Order Preserving	Y	Y	Y	N	Y	N
Recursion	N	N	N/A	N	Y	N
Joins of documents based on values	N	N	N	N	N	N
Joins of documents based on pointers	N	N	N	N	N	N
Update multiple linked documents	N	N	N	N	N	N
A sequence of updates	Y	Y	N	N	Y	Y
Implementaton/prototype	Y	N	Y	N	N	N
Origin of Language	XPath	XQuery	Lorel	XML-GL	XML-RL	XQuery

Table 4: Comparison of techniques for translating XML query into SQL

Researchers	Recursion	Optimization	Represent XML to Database	Language / expession	Database
Fong and Dillon [18]	N	N	shredding	XQL	Relational
Shanmugasundaram et al. [40]	N	N	shredding	Lorel	Relational
Jain et al. [20]	N	Y	shredding	XSLT	Relational
Krishnamurthy et al. [25]	Y	N	shredding	Path expressions	Relational
Fernandez et al. [15]	N	Y	XML V.	XQuery	Relational
Fernandez et al. [16]	N	Y	XML V.	XML-QL	Relational
Shanmugasundaram et al. [39]	N	Y	XML V.	XQuery	Relational
Jensen and Beitzel [9]	N	N	Node	XML-QL	Relational
Khan and Rao [23]	N	N	Node	XPath	Relational
Prakash et al. [35]	Y	N	Node	Path expressions	Relational
Shimura et al. [41]	N	N	Node	XQL	OR
Yoshikawa et al. [50]	N	N	Node	XPath	Relational
Florescu and Kossmann [17]	N	N	Edge	XML-QL	Relational
Jiang et al. [21]	N	N	Edge	XPath	Relational
Manolescu et al. [30]	N	N	Edge	Quilt	Relational
Manolescu et al. [29]	N	N	Edge	XQuery	Relational

Three major points from the literature review are summarized as follows.

Mapping XML to conventional databases: No piece of research handles constraints fully during the mapping of XML to the database. A number [31, 35, 39, 40, 51] do provide partial support for constraints as shown in Table 4. However, for these, constraints may be defined which conflict with each other, for example [51], or which are impracticable in the current technology, for example [31,39], or which are incomplete, for example [35,40]. In addition,

there is just one work, [19], supporting simple update in the query language.

XML update languages: XQuery, a standard from W3C, is the most powerful of the existing XML query languages; thus some works make extensions to it while others propose new query languages supporting update features. Nonetheless, none can update multiple linked documents and join documents in updates. From Table 6 there is only one work XML-RL [101], which can update data whose structure is recursive but this work has not been implemented.

Translating XML query language into SQL: No existing work translates the full recursive function of XQuery into SQL, although, from Table 7, [25, 35] makes some attempt, but the researchers just translate a path expression containing '//' into SQL. Several other features of XQuery such as aggregate functions and quantifier have not been translated into SQL.

Overall: It is evident that no existing approach comes close to providing an effective update language meeting our requirements. In particular, constraints are at best partially handled with severe limitations in some cases on their application. Approaches that can handle constraints to a limited extent have no capability for recursion. In updating, no approach can update multiple linked documents and documents connected through joins.

5. Conclusion and further work

In our work, we have presented an XML update language as an extension to XQuery, translated it into SQL, which is executed on an ORDB storing XML data. The rules propagating the change from ORDB to XML are independent of any XML update language. Therefore, we feel that we have enhanced the XML data model with a general update facility. (We can anticipate that before too long update features for XQuery will be presented by W3C.) We have shown that rules employed in updating XML data can be based on the features of XQuery and the future version of XQuery supporting updates will be based on XQuery also as required by W3C [46].

Meanwhile a standard for updating XML documents has not been proposed. The existing XML databases and XML update languages have limitations in their capability for updating data. In our technique, the technology of ORDB is exploited to increase the capability of existing XML update approaches in the aspect of controlling constraints during updating of XML data, making it easier to join XML documents in updating, allowing the updates of documents whose structure is known partially or whose structure is recursive, and improving the performance of the updates by using regular path expressions. With this approach, there is no need to maintain the order of elements in ORDB, and the cost of converting ORDB data back to XML format is eliminated, since the change in ORDB is propagated to XML already. Our approach makes it possible to query XML data from XML documents, instead of just ORDB. For example, using Kweelt [38] which is an implementation of XQuery for querying XML documents directly, the result from querying is returned in XML format without any conversion. Although DOM has to be serialized back to XML document, serialization is performed only once after all updates finish.

We have conducted an experimental study to verify the method for updating XML and to gain an insight into the performance of the update techniques. Overall, the experimental results show that updating non-redundant data in linked XML documents outperforms updating redundant data. Caching data significantly improves the performance of updating ORDB data. The native XML database has a weakness in handling regular path expressions.

There are many interesting avenues for further work in the XML update area, since XML updating is still in its infancy. The possible extensions to our research include updating XML structure, transaction processing, concurrency control, security for accessing XML data and query optimization through using the results obtained in Section 3 as parameters for a cost model.

6. References

[1] Abiteboul, S., S. Cluet, V. Christophides, T. Milo, G. Moerkotte, and J. Simeon, *Querying documents in object databases.* International Journal on Digital Libraries, Vol. 1(1): pp. 5-19, 1997.

[2] Abiteboul, S., D. Quass, J. McHuge, J. Widom, and J.L. Winer, *The Lorel query language for semistructured data.* International Journal on Digital Libraries, Vol. 1: pp. 68-88, 1997.

[3] Amer-Yahia, S., F. Du, and J. Freire. *A Comprehensive Solution to the XML-to-Relational Mapping Problem.* in *Sixth ACM CIKM International Workshop on Web Information and Data Management (WIDM 2004).* Washington, DC, USA. pp. 31-38, 2004.

[4] Amornsinlaphachai, P., N. Rossiter, and M.A. Ali. *Updating XML using Object-Relational Database.* in *British National Conference.* Sunderland University, UK.: Springer-Verlag. pp. 155-160, 2005.

[5] Amornsinlaphachai, P., N. Rossiter, and M.A. Ali, *Storing Linked XML documents in Object-Relational DBMS.* Journal of Computing and Information Technology, Vol. 14(3): pp. 225-241, 2006.

[6] Amornsinlaphachai, P., N. Rossiter, and M.A. Ali, *Translating XML update language into SQL.* Journal of computing and Information Technology, Vol. 14(2): pp. 81-100, 2006.

[7] Atay, M., A. Chebotko, D. Liu, S. Lu, and F. Fotouhi, *Efficient schema-based XML-to-Relational data mapping.* Information Systems, 2006.

[8] Babcock, C., *Internet Insight: XML Users Consider Nonstandard.* 2002, appears in Ziff Davis' eWeek 11 Feb. 2002. http://www.charlesbabcock.com/xquery.htm

[9] Beitzel, S.M., E.C. Jensen, and D.A. Grossman. *Using a Relational Database Management System to Implement XML-QL.* in *Proceedings of the 17th International Conference on Advanced Science and Technology (ICAST'2001).* Chicago, 2001.

[10] Bohannon, P., J. Freire, P. Roy, and J. Simeon. *From XML schema to Relations: A Cost-Based Approach to XML Storage.* in *Proceedings of the International Conference on Data Engineering.* pp. 64-75, 2002.

[11] Ceri, S., S. Comai, E. Damiani, P. Fraternali, S. Paraboschi, and L. Tanca, *XML-GL: a Graphical Language for Querying and Restructuring WWW Data.* The International Journal of Computer and Telecommunications Networking, Vol. 36: pp. 1171-1187, 1999.

[12] Chamberlin, D., *XQuery from the experts: Aguide to the W3C XML Query Language.* Influences on the Design of XQuery, p. 143, ed. H. Katz. XQuery from the experts: Aguide to the W3C XML Query Language: Addison-Wesley, 2003.

[13] Deutsch, A., M.F. Fernandez, and D. Suciu:. *Storing Semistructured Data with STORED.* in *SIGMOD Conference.* Philadelphia, Pennsylvania, United States. pp. 431-442, 1999.

[14] Fegaras, L. and R. Elmasri, *Query Engine for Web-Accessible XML Data.* The VLDB Journal, pp. 251-260, 2001.

[15] Fernandez, M., Y. Kadiyska, D. Suciu, A. Morishima, and W.-C. Tan, *SilkRoute: A Framework for Publishing Relational Data in XML.* ACM Transactions on Database Systems, pp. 438-493, 2002.

[16] Fernandez, M., W.C. Tan, and D. Suciu, *SilkRoute : Trading between Relations and XML.* Computer Networks, Vol. 33: pp. 723-745, 2000.

[17] Florescu, D. and D. Kossmann, *Storing and querying XML data using an RDBMS.* IEEE Data Engineering Bulletin, Vol. 22(3): pp. 27-34, 1999.

[18] Fong, J. and T. Dillon. *Towards Query Translation From XQL to SQL.* in *Proceedings of the 9th IFIP 2.6 working conference on database semantics (DS9).* Hong Kong. p. 113-129, 2001.

[19] Han, W.-S., K.-H. Lee, and B.S. Lee, *An XML Storage System for Object-Oriented/Object-Relational DBMSs.* Journal of Object Technology, Vol. 2(3): pp. 113-126, 2003.

[20] Jain, S., R. Mahajan, and D. Suciu. *Translating XSLT Programs to Efficient SQL Queries.* in *Proceedings of the Eleventh International World Wide Web Conference, WWW2002.* New York, NY, USA., Honolulu, Hawaii, USA.: ACM Press. pp. 616-626, 2002.

[21] Jiang, H., H. Lu, W. Wang, and J.X. Yu. *XParent: An efficient RDBMS based XML database system.* in *Proceedings of the 18th International Conference on Data Engineering.* San Jose, California. pp. 335-336, 2002.

[22] Khan, L., Q. Chen, and Y. Rao. *A Comparative Study of Storing XML Data in Relational Database Management Systems.* in *Proceedings of the International Conference on Internet Computing,*

IC'2002. Las Vegas, Nevada. pp. 277-282, 2002.

[23] Khan, L. and Y. Rao. *A Performance Evaluation of Storing XML Data in Relational Database Management Systems*. in *3rd International Workshop on Web Information and Data Management (WIDM 2001)*. Atlanta, Georgia, USA. pp. 31-38, 2001.

[24] Klettke, M. and H. Meyer, *Managing XML Documents in object-relational databases, PhD. Thesis*, in *Computer Science Department*., University of Rostock: Rostock, Germany, 1999.

[25] Krishnamurthy, R., V.T. Chakaravarthy, R. Kaushik, and J.F. Naughton. *Recursive XML Schema, Recursive XML Queries, and Relational Storage: XML-to-SQL Query Translation*. in *Proceedings of the 20th International Conference on Data Engineering, ICDE 2004*. Boston, MA, USA. pp. 42-53, 2004.

[26] Lu, L., M. Liu, and G. Wang. *A Declarative XML-RL Update Language*. in *Proceedings of 22nd International Conference on Conceptual Modeling (ER 2003)*. Chicago, Illinois, USA: Springer-Verlag. pp. 506-519, 2003.

[27] Lu, S., Y. Sun, M. Atay, and F. Fotouhi, *On the consistency of XML DTDs*. Data & Knowledge Engineering, Vol. 52: pp. 231-247, 2005.

[28] Lv, T. and P. Yan, *Mapping DTDs to relational schema with semantic constraints*. Information and Software Technology, Vol. 48: pp. 245-252, 2006.

[29] Manolescu, I., D. Florescu, and D. Kossma. *Answering XML Queries over heterogeneous Data Sources*. in *Proceedings of the 27th VLDB Conference*. Roma, Italy, 2001.

[30] Manolescu, I., D. Florescu, and D. Kossma, *Pushing XML Queries inside Relational Databases*, in., INRIA Technical Report No. 4112 Report no. 4112, 2001.

[31] Mignet, L., D. Barbosa, and P. Veltri. *The XML Web: a First Study*. in *The Twelfth International World Wide Web Conference (WWW2003)*. Budapest, Hungary. pp. 500-510, 2003.

[32] Mo, Y. and T.W. Ling. *Storing and Maintaining Semistructured Data Efficiently in an Object-Relational Database*. in *The Third International Conference on Web Information Systems Engineering (WISE'00)*. Singapore. pp. 247-256, 2002.

[33] Obasanjo, D. and S.B. Navathe. *A Proposal for an XML Data Definition and Manipulation Language*. in *VLDB 2002 Workshop EEXTT and CAiSE 2002 Workshop DTWeb*. Hongkong Chaina. pp. 1-21, 2002.

[34] Pardede, E., J.W. Rahayu, and D. Taniar, *Object-relational complex structures for XML storage*. Information and Software Technology, Vol. 48: pp. 370-384, 2006.

[35] Prakash, S., S.S. Bhowmick, and S. Madria, *Efficient recursive XML query processing using relational database systems*. Data & Knowledge Engineering, 2006.

[36] Psaila, G. *ERX: A Conceptual Model for XML Documents*. in *Proceedings of the 2000 ACM Symposium on Applied Computing*. Como, Italy. pp. 898-903, 2000.

[37] Runapongsa, K. and J.M. Patel. *Storing and Querying XML data in Object-Relational DBMSs*. in *XML-Based Data Management and Multimedia Engineering - EDBT 2002 Workshops*. Prague, Czech Republic. pp. 266-285, 2002.

[38] Sahuguet, A. *Kweelt: More than Just "Yet Another Framework to Query XML!* in *SIGMOD 2001 Electronic Proceedings*. Santa Barbara, CA. pp. 602, 2001.

[39] Shanmugasundaram, J., J. Kiernan, E. Shekita, C. Fan, and J. Funderburk. *Querying XML Views of Relational Data*. in *Proceedings of the 27th VLDB Conference*. Roma. Italy. pp. 261-270, 2001.

[40] Shanmugasundaram, J., K. Tufte, G. He, C. Zhang, D. DeWitt, and J. Naughton. *Relational Databases for Querying XML Documents: Limitations and Opportunities*. in *Proceedings of the 25th VLDB Conference*. Edinburgh, Scothland. pp. 302-314, 1999.

[41] Shimura, T., M. Yoshikawa, and S. Uemura. *Storage and Retrieval of XML Documents using Object-Relational Databases*. in *Database and Expert Systems Applications, 10th International*

Conference, DEXA '99. Florence, Italy. pp. 206-217, 1999.

[42] Tatarinov, I., Z. Ives, A.Y. Halevy, and Daniel S. Weld. *Updating XML*. in *Proceedings of 2001 SIGMOD Conference*. Santa Barbara, CA, USA. pp. 413-424, 2001.

[43] Tseng, F.S.-C. and W.-J. Hwung, *An automatic load/extract scheme for XML documents through object-relational repositories*. The Journal of Systems and Software, Vol. 64: pp. 207-218, 2002.

[44] Varlamis, I. and M. Vazirgiannis. *Bridging XML-Schema and relational databases. A system for generating and manipulating relational databases using valid documents*. in *ACM Symposium on Document Engineering 2001*. Atlanta, Georgia, USA. pp. 105-114, 2001.

[45] W3C, *XQuery 1.0: An XML Query Language*. 2003. http://www.w3.org/TR/xquery

[46] W3C, *XQuery Update Facility Requirements, W3C Working Draft*. 2005.http://www.w3.org/TR/xquery-update-requirements/

[47] Wang, G. and M. Liu. *Query Processing and Optimization for Regular Path Expressions*. in *Advanced Information Systems Engineering, 15th International Conference*. Klagenfurt, Austria. pp. 30-45, 2003.

[48] Watson, P. *Databases in Grid Applications: Locality and Distribution*. in *Proceedings of the Database: Enterprise, Skills and Innovation. 22nd British National Conference on Databases, BNCOD 22*. Sunderland, UK: Springer-Verlag. pp. 1-16, 2005.

[49] XMLDB, *XUpdate*. 2002. http://www.xmldb.org/xupdate/xupdate-wd.html

[50] Yoshikawa, M., T. Amagasa, T. Shimura, and S. Uemura, *XREL: A path based approach to storage and retrieval of XML documents using relational databases*. ACM Transactions on Internet Technology, Vol. 1(1): pp. 110-141, 2001.

[51] Zhou, A., H. Lu, S. Zheng, Y. Liang, L. Zhang, W. Ji, and Z. Tian. *VXMLR: A visual XML-relational database system*. in *Proc. of the 27th Int'l Conference on Very Large Data Base*. Roma, Italy. pp. 719-720, 2001.

[52] Zwol, R.V., P.M.G. Apers, and A.N. Wilschut. *Modelling and Querying Semistructured Data with MOA*. in *proceedings of Workshop on Query Processing for Semistructured Data and Non-standard Data Formats*. Jerusalem, Israel, 1999.