Positioning-Based Query Translation between SQL and XQL with Location Counter

Joseph Fong*, Wilfred Ng+, San Kuen Cheung*, Ivan Au*

*Computer Science Department, City University of Hong Kong, Hong Kong
Email:csjfong@cityu.edu.hk
+Computer Science Department, Hong Kong University of Science and Technology
Email:Wilfred@cs.ust.hk

The need for interoperation and data exchange through the Internet has made Extensible Markup Language (XML) a dominant standard language. Much work has already been done on translating relational data into XML documents and vice versa. However, there is not an integrated method to combine them together as a unifying technology for database interoperability on the Internet. Users may not be familiar with various query language syntax. We propose database gateways built on the top of a Relational Database (RDB) and an XML Database (XMLDB). Users can access both databases at the same time through the query language SQL or XQL (an XML query language) to access data stored in either RDB or XMLDB. The translation process adopts query graph translation between a RDB and an XMLDB. Thus, a stepwise procedure of query translation is devised and amenable to implementation. The procedure also provides an XML interface to a RDB as well as a relational interface to XMLDB. A location counter sequence number is used to position tuples in a RDB for subsequent transforming the tuples into the corresponding positioning element instances in the XML documents. As a result, both XMLDB and RDB can co-exist, and be accessible by the users.

1 Introduction

This paper proposes a stepwise approach to query translation that constructs a gateway between relational and XML database systems. One of the keys to success in migration strategy is the ability that copes with the changes imposed by business requirements. We address the issue of such changes by partitioning the process of database migration and query translation into three phases as shown in Figure 1. Phase I is the translation of the source relational schema into a target XML schema. Phase II is an inverse mapping that is represented as augmented views. These augmented views are similar to relational views but it is more flexible for users to select their root-based XML documents. Phase III transforms a query from relational SQL into an equivalent XQL query over the target XML database. The rationale for defining phases I and II is that a relational schema is not wholly compatible with an XML schema. As a result, we need to partition a relational schema into an augmented view of XML tree structure in order to make them compatible. A methodology is developed to allow users to interoperate RDB and XMLDB through query translation. The database gateways receive the input queries before they are sent to the underlying databases. Query translation will be done through the gateways. The translated query will be sent to the appropriate database. Users can rely on one data model and his or her familiar query language to access data in both the RDB and the XMLDB.

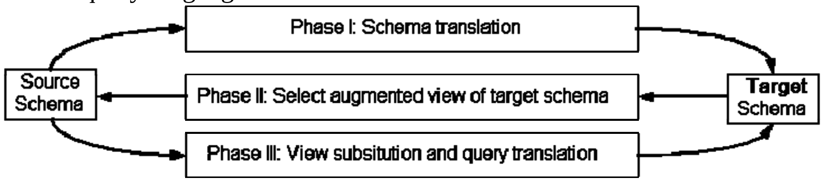

Phase I: Schema translation
Source Schema
Phase II: Select augmented view of target schema
Target Schema
Phase III: View subsitution and query translation

Figure 1 The three steps for database reengineering query translation

The query translation process consists of two main steps of schema translation and query translation. The system allows users to input SQL query which is translated into XPath for selecting the data on XMLDB. The architecture composes of two gateways, the XML Gateway and the Relational Gateway, as shown in Figure 2. There is a common interface between these gateways, which connect to a XMLDB server and a RDB server.

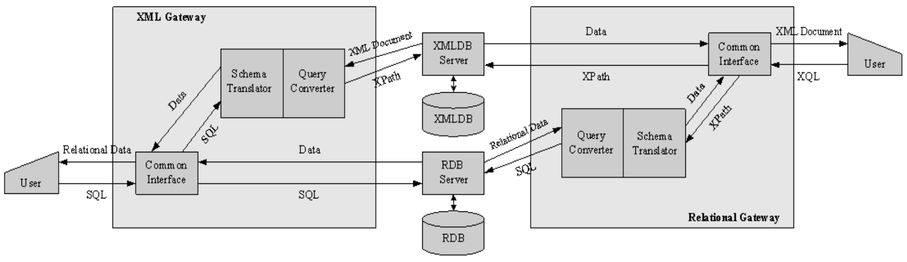


Figure 2 XML and Relational Gateways

1.1 Related Work

Shanmugasundaram [2] presents three inlining algorithms that focus on the table level of the schema while Florescu and Kossmann [3] investigate various performance issues among five algorithms that focus on the attribute and value level of the schema. They all transform the given XML DTD to a relational schema. Collins [4] describes two algorithms for mapping relational and network models schemas into XML schema using the relational mapping algorithm. Such an approach allows the data in the relational and network database system. Tatarinov [5] studies how XML's ordered data model can be supported using an unordered RDB system. They propose three order encoding methods (Global Order, Local Order and Dewey Order) for representing XML in the relational model. Tseng and Hwung [6] developed a system called XML meta-generator (XMG) that is an extraction scheme to store and retrieve XML documents through object-relational databases. WebReader [7] is a middleware for automating the search and collecting information and manipulation in XSL, WebReader also provides the users with a centralized, structured, and categorized means to specify for querying web information.

2 Methodology of Query Translation between SQL and XQL

As shown in Figure 1, we need to abstract an augmented view of the target XML tree structure into a relational schema in phases I and II. We have a compatible tree structure in a partitioned relational schema and a mapped target XML schema. Then in phase III, we translate an SQL to XQL according to the mapped XML schema. Similarly, we can translate an XQL to SQL according to the partitioned relational schema. These three phases are further detailed in the subsequent subsections.

2.1 Phase I: Schema Translation between Relational and XML Data

We add a sequence number into a relational table for data position in XML document. For any table that is used for query translation, an extra column - *seqno* is required. This column is used by the XML gateway described as follows:

For each table, the last column is *seqno*. This *seqno* column is used to ensure that the records returned from database are in the right order. The column is also used for translation of XQL location index functions (e.g. *position()*). The *seqno* column is incremented by one for each new record of the same key value and maintained by using the insert trigger. The records in the repository table *node_tablecolumn_mapping* is used for mapping the column of the table that is used for maintaining the *seqno* value.

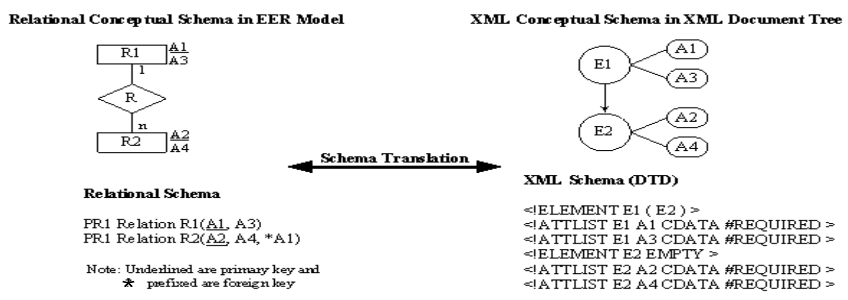
node_tablecolumn_mapping Table

Table name	Node key column
CLIENT	ClientID
CLIENTACCOUNTEXECUTIVE	ClientID
ACCOUNTEXECUTIVE	AEID
BALANCE	ClientID

CLIENTACCOUNTEXECUTIVE Table

ClientID	AEID	*Seqno*
600001	AE0001	1
600001	AE0002	2
600002	AE0001	1
600003	AE0003	1

On inserting a new record into a table, the insert trigger first locates that the column is used for counting *seqno* from the *node_tablecolumn_mapping* table. Then, the trigger selects the maximum *seqno* value for the new record. The maximum *seqno* value plus one is assigned as the *seqno* value of the new record. There is no need to update the *seqno* value in case the record is deleted. In XQL, the location index function (e.g. *position()*) counts the order of the record relative to the parent node. Given receiver's relations R1(A1, A3) and R2(A2, A4, *A1) with an FD (functional dependency): R2.A1 → R1.A1. R1 and R2 are classified and joined into a relation R(A1, A2, A3, A4), which is then translated into a single sub-element topological XML document by mapping parent relation R1 into element E1, and child relation R2 into sub-element E2 as shown in Figure 3.


Relational Conceptual Schema in EER Model
R1
A1
A3
1
R
n
R2
A2
A4
Schema Translation
XML Conceptual Schema in XML Document Tree
E1
A1
A3
E2
A2
A4
Relational Schema
XML Schema (DTD)
Note: Underlined are primary key and
* prefixed are foreign key

```
PR1 Relation R1(A1, A3)
PR1 Relation R2(A2, A4, *A1)
```

```
<!ELEMENT E1 ( E2 ) >
<!ATTLIST E1 A1 CDATA #REQUIRED >
<!ATTLIST E1 A3 CDATA #REQUIRED >
<!ELEMENT E2 EMPTY >
<!ATTLIST E2 A2 CDATA #REQUIRED >
<!ATTLIST E2 A4 CDATA #REQUIRED >
```

Figure 3 Translation of Functional Dependency between Relational and XML Schemas

2.2 Phase II: Select Augmented View of Target XML Schema in Mapping RDB to XML Schema

To convert a relational database into an XML document tree, we integrate the translated XML document trees into an XML document tree, select an element as root and put its relevant information into a document. We load the relational database into the object instances of the XML documents. Each XML document focuses on a root class upon user requirements. The selection is driven by some business requirements. Relevance concerns elements that are related to a root selected by the users among the integrated XML document tree (DOM tree). Figure 4(a) shows an integrated XML document tree. The user can select root A1 with its relevant classes to form a partitioned XML document tree.

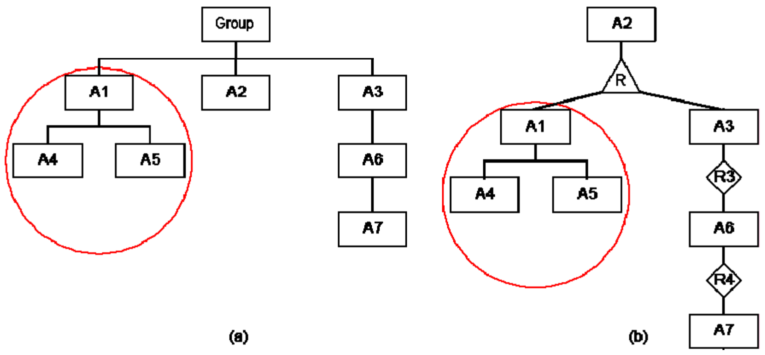


Figure 4 Select augment view of target (a) XML schema and (b) relational schema in cycle

Select augmented view of target RDB schema in mapping XML to RDB schema. Similar to convert an XML schema to RDB schema, we allow user select an augmented view of the EER model of target RDB schema as shown in Figure 4(b).

2.3 Phase III: Query Translation between XQL and SQL

2.3.1 Query Translation from SQL to XQL

In query transformation, a syntax-directed parser converts the SQL into multi-way trees. The transformation process is performed, based on the subtree matching and replacement technique. The process of SQL query transformation is given in Figure 5.

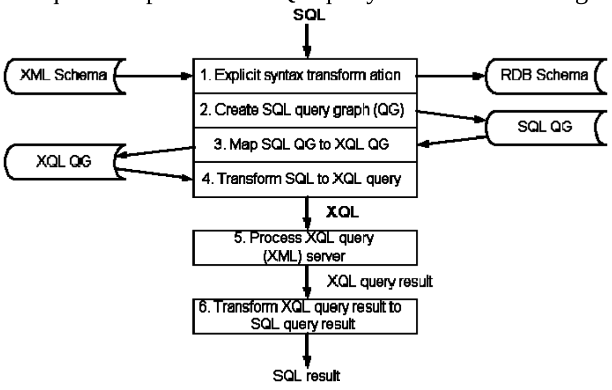

SQL
XML Schema
1. Explicit syntax transform ation
RDB Schema
2. Create SQL query graph (QG)
SQL QG
3. Map SQL QG to XQL QG
XQL QG
4. Transform SQL to XQL query
XQL
5. Process XQL query (XML) server
XQL query result
6. Transform XQL query result to SQL query result
SQL result

Figure 5 Process for SQL to XQL Transformation

Translation of SQL Query to XPath Query

After the schema is done, SQL query can be translated to XPath query by the following steps:

Step 1 Decompose SQL Query Transaction: The basic syntax SQL SELECT statement is given as follows:

```
SELECT {attribute-list-1} FROM {relation-list}
WHERE {join-condition} AND / OR {search-condition-1}
ORDER BY {attribute-list-2} GROUP BY {attribute-list-3}
HAVING {search-condition-2}
```

Step 2 Create the SQL Query Graph: Based on the relation-list and the join-condition in the SQL query transaction, the SQL query graph is created. The join condition is based on the natural join or based on the search condition specified in the SQL query [1].

Step 3 Map the SQL Query Graph to XPath Query Graph: The SQL query graph is mapped to the XPath query graph. The table joins from the SQL query graph forms the XPath location path, which are the steps for navigating down the document tree from root node.

Step 4 Transform SQL to XPath Query: In this step, the SQL query is transformed into XPath syntax as follows:

```
/root/node1[@attribute1=condition]/…/node2[@attribute2=condition]/@attribute3
```

The attribute-list in the SQL query is mapped to the leaf attribute node at the bottom of the document tree. If all the attributes of the element node are selected, “@*” is mapped to select all the attributes from the leaf element node. If more than one attributes are selected, the union operator is used to get the result. For example:

/root/node1/@attribute1 | /root/node1/@attribute2

Step 5 Transform XPath Query Data into SQL Query Data: The XML document returned from XMLDB is formatted into tables before the document returning to user. The format of the result is based on the data stored in the table *table_column_seq* (prepared in pre-processed schema translation).

2.3.2. Query Translation from XQL to SQL

To translate query from XQL to SQL, document tree nodes in XQL query are replaced by the relational JOIN in SQL. The XQL allows data retrieval using path expressions, and data manipulation using methods. A syntax-directed parser converts the XQL into multi-way trees. The transformation process is performed, based on the subtree matching and replacement technique. The process of XQL query transformation is given in Figure 6.

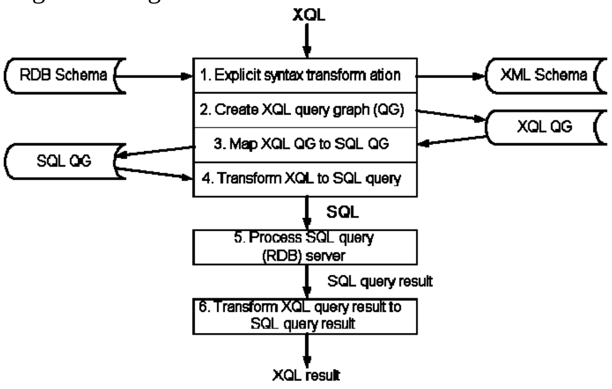

XQL
RDB Schema
1. Explicit syntax transform ation
XML Schema
2. Create XQL query graph (QG)
XQL QG
3. Map XQL QG to SQL QG
SQL QG
4. Transform XQL to SQL query
SQL
5. Process SQL query (RDB) server
SQL query result
6. Transform XQL query result to SQL query result
XQL result

Figure 6 Process for XQL to SQL Transformation

2.3.3. Query Translation from XPath to SQL

XPath views a document as a tree of nodes consisting of elements and attributes. Based on the generated XML schema, the XPath query graph of node navigation is converted to SQL query graph of table joins. Below is a stepwise procedure of how XPath query is translated to SQL query:

Step 1 Decompose the XPath Transaction: Each slash-separated (/) path component of XPath query is a step. The following nodes and predicates are identified from the descendent axis:

1. * - selects all element children of the context node
2. @name – selects the name attribute of the context node
3. @* - selects all the attributes of the context node
4. [method] – built-in functions to create more expressive queries. They are text(), *position()*=n, or last() where n is index of the location of element instance starting from 0.
5. [@name= “attributeValue”] – the value of the name attribute is equal to “attributeValue”

Step 2 Create XPath Query Graph: The query graph of the XPath expression or query is created in this step. Based on the translated XML schema, a navigation path is created to indicate the relationship between nodes by stepping down the XML document tree hierarchy. From the node_sequence table, all the nodes down from the root element are identified.

Step 3 Map the XPath Query Graph to SQL Query Graph: From the XPath query graph, the root node and its descendant child node are located. For each node, the elements are mapped to their corresponding relation in the RDB.

Step 4 Translate XPath Query to SQL Queries: From the mapped XML schema, each XML document has a key for retrieving the data for each element node. A key cursor is created for first retrieving the keys for the XML documents. This key is stored in the table xml_document_key and each value fetched from this cursor is used subsequently for each translated SQL query. It is constructed by the following replacements:

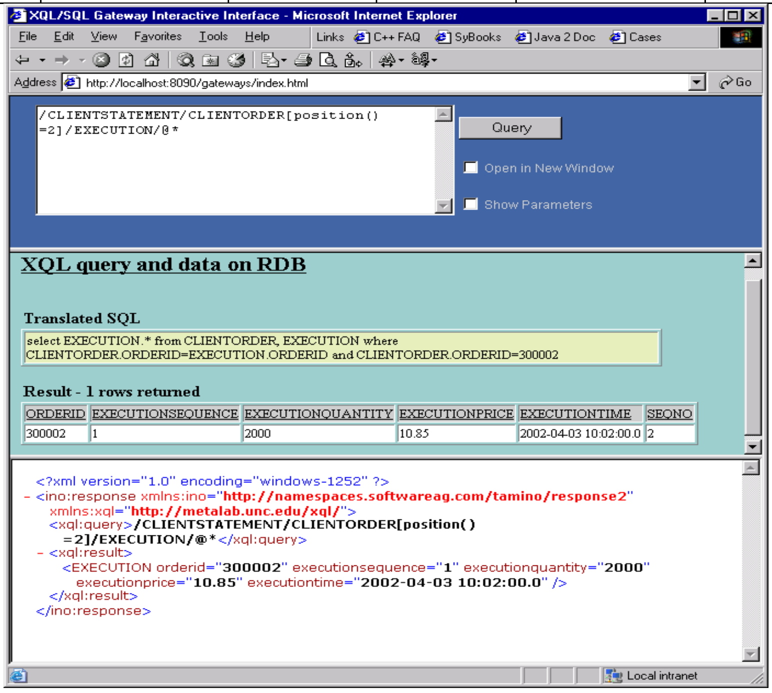
- Replacing XML document tree node navigation path by SQL join relations path.
- Replacing the XML tree elements by their corresponding SQL relations.
- Replacing XML document tree node filtering by SQL WHERE clause.
- Replacing the XML document instance location index function by embedded SQL query cursor. By counting the number of time result set is fetched from cursor, the location index of XML document is emulated.

Step 5 Map the Retrieved Relation Data into XML Document Format: The result set returned from SQL query is mapped into the translated XML schema. The tags for the data returned from SQL query are identified from the table xml_document_node. The result is formatted into XML document returned to user[1],

3 Conclusions

This paper describes a methodology that translates XQL query to SQL query and vice versa. Sequence numbers are applied to indicate the position of the relational tuples that are involved in the schema translation. The approach provides flexibility for users to query on a selected (focused on root based) XML view of a relational database when translating SQL to XQL, or to query a set of selected relational tables in translating XQL to SQL. The benefit of our approach is that the sequential processing of both the relational database and the XML database are compatible due to the added positioning *SEQNO* in the relational side. Our approach has a distinct feature that XDB and RDB are able to co-exist, and XQL and SQL can be employed to access both systems. As shown in Figure 7, a prototype of the database gateways is developed. It shows that query translation between SQL and XQL with the proposed methodology is feasible. For example, we want to fetch its second tuple of the following Execution table:

Orderid	Exe_sequence	Quantity	Price	Time	*Seqno*
300001	1	10000	21.6	200204031001	1
300002	1	2000	10.85	200204031002	1


XQL/SQL Gateway Interactive Interface - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Links C++ FAQ SyBooks Java 2 Doc Cases
Address http://localhost:8090/gateways/index.html
Go
Query
Open in New Window
Show Parameters
Local intranet

```
/CLIENTSTATEMENT/CLIENTORDER[position()
=2]/EXECUTION/@*
```

XQL query and data on RDB

Translated SQL

```
select EXECUTION.* from CLIENTORDER, EXECUTION where
CLIENTORDER.ORDERID=EXECUTION.ORDERID and CLIENTORDER.ORDERID=300002
```

Result - 1 rows returned

ORDERID	EXECUTIONSEQUENCE	EXECUTIONQUANTITY	EXECUTIONPRICE	EXECUTIONTIME	SEQNO
300002	1	2000	10.85	2002-04-03 10:02:00.0	2

```
  <?xml version="1.0" encoding="windows-1252" ?>
- <ino:response xmlns:ino="http://namespaces.softwareag.com/tamino/response2"
    xmlns:xql="http://metalab.unc.edu/xql/">
    <xql:query>/CLIENTSTATEMENT/CLIENTORDER[position()
      =2]/EXECUTION/@*</xql:query>
  - <xql:result>
      <EXECUTION orderid="300002" executionsequence="1" executionquantity="2000"
        executionprice="10.85" executiontime="2002-04-03 10:02:00.0" />
    </xql:result>
  </ino:response>
```

Figure 7 XQL Query with Position Value Translated into SQL

References

[1] Ivan Au, Feasibility Study of Query Translation between SQL and XQL, M.Sc. dissertation of C.S. Department at City University of Hong Kong, 2002

[2] Jayavel Shanmugasundaram et al, RDBs for Querying XML Documents: Limitations and Opportunities, 25th VLDB Conference, 1999, Page(s): 302-314.

[3] D. Florescu, and D. Kossman, Storing and Querying XML Data Using an RDBMS, IEEE Data Engineering Bulletin, 22(3), 1999, Page(s): 27-34

[4] Samuel Robert Collins et al, XML Schema Mappings for Heterogeneous Database Access, Information and Software Technology, Volume 44, Issue 4, March 2002, Page(s): 251-257

[5] Igor Tatarinov et al, Storing and Querying Ordered XML Using a RDB System, 2002 ACM SIGMOD int'l conference on Management of data, June 2002

[6] Frank S. C. Tseng and Wen-Jong Hwung, An Automatic Load/Extract Scheme for XML Documents through Object-Relational Repositories, Journal of Systems and Software, Volume 64, Issue 3, December 2002, Page(s): 207-218

[7] J. Chan and Q. Li, WebReader: A Mechanism for Automating the Search and Collecting Information from the World Wide Web, 1st International Conference on Web Information Systems Engineering, Volume 2, 2000, Page(s): 47-56