SIMULTANEOUS QUERYING OF XML AND RELATIONAL CONTEXTS

Madani Kenab
IRIT, 118, Route de Narbonne 31062 Toulouse

Tayeb Ould Braham
MSI, 83, Rue d'Isle 87000 Limoges

Keywords: Relational schema, XML schema, XML view, XQuery language, SQL language

Abstract: The presentation of the results of relational queries is flat. The prime objective of this work is to query an XML view of relational data in order to have nesting results of data implemented in the form of flat data. The second objective is to combine, in query results, structured data of a relational database and semi-structured data of an XML database. A FLWR expression (For Let Where Return) of the XQuery language can be nested at various levels in another FLWR expression. In our work, we especially are interested in the nesting of a FLWR expression in the Return clause of another FLWR expression in order to imbricate data in the result. In this paper, we will describe all necessary stages in order to carry out these two objectives.

1 INTRODUCTION

Still today, much of data are stored in relational databases. The relational model is constrained by the normal forms that prohibit the existence of attributes having aggregate values. XML schemas offer a richer set of data type than that which is proposed in the relational model. Contrary to the relational model, XML schemas enable to define personal and eventually recursive types. The relational model use only structured data, on the other hand, the XML model can use structured or semi-structured data according to the utilization or not of an XML schema and according to the contents of the used XML schema. This possibility brings flexibility to the XML model. The XQuery language uses the Xpath language to identify the different elements and/or attributes from an XML document in order to handle them according to the needs. Contrary to the XQuery, the SQL language provides results in the form of tuples and does not allow nesting results. All these advantages of the XML model encourage to use it as a pivot model in the systems integrating heterogeneous databases.

In this paper, we present a system of integration of heterogeneous databases that we carried out in the form of a software layer on an XQuery engine and a relational engine contrary to the studied systems which use only one engine (CLIO (Hernandez, 2001), SilkRoute (Fernandez 2002, Fernandez 2000), XTABLES (Fan, 2002), AGORA (Manolescu, 2000) and LeSelect (INRIA, 1998)). Our system offers two graphic interfaces: one allowing to generate and visualize an XML schema describing a relational database according to the choice of the user (Kenab, 2004a) and the other allowing to query heterogeneous databases (relational database RDB and documentary database DDB) by using the XQuery language. The XML schema generated in the first interface is built from information concerning the relational schema of the database to integrate. The translation of the Xquery requests, intended to the relational engine, in SQL requests is based on XML schema without using intermediate representation such as it is made in the studied systems. The results coming from the query of heterogeneous data are converted and displayed in the form of a persistent XML document. In order to improve the presentation of these results, we enable to display these results in the form of an XHTML document. This last is obtained by using an XSLT processor and an XSL stylesheet. This XSL stylesheet is generated automatically from the XML document corresponding to the result and its XML schema (Kenab, 2004b). This XML schema is generated automatically from the metadata concerning these results. The XML document result

Kenab M. and Ould Braham T. (2005).
SIMULTANEOUS QUERYING OF XML AND RELATIONAL CONTEXTS.
In *Proceedings of the Seventh International Conference on Enterprise Information Systems*, pages 353-358
DOI: 10.5220/0002515403530358

with its XML schema become an XML entity that we can query.

This paper is structured in the following way. In the first section, we present a method allowing to obtain an XML schema describing an XML view on a relational database. In the second section, we model the translation of an Xquery request to an SQL request in a functional way. The transformation of the results of an SQL request, in the form of tuples, to an XML document is presented in the third section. In the fourth section, we present the possibility of querying heterogeneous databases (documentary and relational) in the same query.

2 INTEGRATION OF A RELATIONAL SCHEMA INTO AN XML CONTEXT

A relational database consists of a set of tables. Each table contains fields (attributes). This database can be described by the following relational schema:
Relational_schema=(table$_1$(Attribute$_{11}$:Attribute_Type$_{11}$,…,Attribute$_{1m_1}$:Attribute_Type$_{1m_1}$),…, table$_n$(Attribute$_{n1}$:Attribute_Type$_{n1}$, …, Attribute$_{nm_n}$:Attribute_Type$_{nm_n}$), Primary_Keys, Reference_Links)

We chose to offer a view of this relational database through an XML schema that is made up of three levels. In this XML schema, at a first level we declare a compound global element, which contains the various tables of the database. Then, at a second level for each table, we define a compound element, which contains a set of simple elements of third level which are attributes belonging to this table. This XML schema corresponds to the model without nesting in element oriented mode with key and keyref clauses to express keys and reference links. This XML schema enables to avoid the drawbacks of the update as well as the cases of associations with minimal cardinality equal to zero (Kenab, 2004a). The construction of this XML schema can be modelled by the following functional expression:

XML_Schema(
  Compound_global_element(
    XML_RDB,
    Compound_element(
      table$_1$,
      Simple_element(Attribute$_{11}$, Attribute_Type$_{11}$),
      …,
      Simple_element(Attribute$_{1m_1}$, Attribute_Type$_{1m_1}$)
    ),
    …,
    Compound_element(
      table$_i$,
      Simple_element(Attribute$_{i1}$, Attribute_Type$_{i1}$),
      …,
      Simple_element(Attribute$_{im_i}$, Attribute_Type$_{im_i}$)
    ),
    …,
    Compound_element(
      table$_n$,
      Simple_element(Attribute$_{n1}$, Attribute_Type$_{n1}$),
      …,
      Simple_element(Attribute$_{nm_n}$, Attribute_Type$_{nm_n}$)
      ),
      Key(table$_1$, Attribute$_{1c_1}$),
      …
      Key(table$_n$, Attribute$_{nc_n}$),
      Link(table$_1$, Attribute$_{1j_1}$, table$_{k_1}$),
      …
      Link(table$_n$, Attribute$_{nj_n}$, table$_{k_n}$)
)
)

with : i = 1..n, $c_i \in \{1, ..., m_i\}$, $j_i \in \{1, …, m_i\}$, $k_i \in \{1, …, n\}$ and n being the number of tables in the database and m_i the number of fields (attributes) of the table table$_i$.

XML_schema is a function which tags a parameter with <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"> and </xs:schema>. *Compound_global_element* is a function allowing to define the compound global element which contains the XML elements corresponding to the tables, the keys and the reference links of the relational database. *Compound_element* is a function allowing to define a compound XML element corresponding to a table of the relational database. *Simple_element* is a function allowing to define a simple XML element corresponding to a table attribute of the relational database. *Key* is a function allowing to define a key corresponding to a table key attribute of the relational database. *Link* is a function allowing to define a reference link corresponding to a table foreign key attribute of the relational database.

The realization of this integration of the relational schema into an XML schema (XML view) is made according to the schema of Figure1. The two principal stages of this integration are:

- The querying of the relational engine in order to obtain the relational schema (metadata).
- The transformation of the relational schema into an XML schema as view on the relational database thanks to the functional expression described previously.

Figure 1: Integration of a relational schema into an XML schema

3 TRANSFORMATION OF AN XML REQUEST (XQUERY) INTO A RELATIONAL REQUEST (SQL)

In this section, we use an Xquery request to query an XML view on a relational database (Figure2). Before being translated into an SQL request, the consistency of this request, with the XML schema describing an XML view, is checked. The result of the SQL request, in the form of table, is converted into an XML document as it will be described in next section.

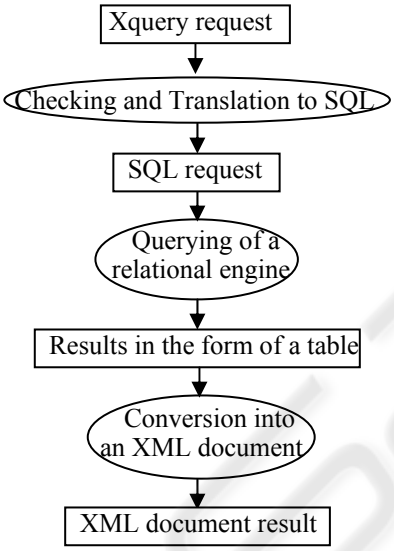

Figure 2: Querying of an XML view on a relational database

A general R Xquery request is a FLWR expression which we can model in the following form:

for $f_var_1 in f_exp_1 (: f_ for for :)
...
for $f_var_n in f_exp_n
let $l_var_1 := l_exp_1 (: l_ for let :)
...
let $l_var_r := l_exp_r
where *condition*(BFs, JFs)
return *XML_doc*(Results, SRs)

Where:

- f_exp_i and l_exp_i are simplified Xpath expressions (without condition between [ ] and without wildcard) used respectively in the for and let clauses.
- BFs are boolean expressions expressed thanks to Xpath expressions, each one containing f_var_i or l_var_i variable.
- JFs are boolean expressions of joint expressed thanks to Xpath expressions , each one containing f_var_i or l_var_i variable.
- *condition* is a boolean function returning the value of a logical expression made up of logical expressions built from the results of the BFs and JFs functions passed in parameters.
- *XML_doc* is a function which models an XML document from the tags provided by the user and parameters having various types according to the context. In this precise context, the parameters correspond to Results and nested FLWR expressions SRs.
- Results are Xpath expressions expressed thanks to f_var_i and l_var_i variables of the for and let clauses of the XQuery request R.
- SRs are nested FLWR expressions (XQuery Sub-Requests) of the return clause having the same format than R, they enable to formulate nested sub-requests of the same level. The other levels of nesting will be generated recursively.

The *Tran*(R) function of the above R FLWR expression will return an SQL request which we can model in several stages and in a recursive way. The contents of the select part (S function) and the from part (F function) of *Tran*(R) will be built from the return clause and from the for and let clauses of the R Xquery request. The contents of the where part (W function) of *Tran*(R) will be built from the where clause and from the for and let clauses of the R Xquery request. The transformation of the XML element into relational attributes or tables is related to the structure of the XML schema resulting from the integration of the relational schema. In our case, it is the XML schema without nesting, element oriented mode with keys and reference links expressed respectively thanks to *key* and *keyref* clauses. For that reason, an Xpath expression indicating an attribute will have the following form /XML_RDB/Named_tuple/Attribute.

The result of the function *Tran*(R) which will have the following shape: select *S*(*Tran*(R)) from *F*(*Tran*(R)) where *W*(*Tran*(R)) will be an SQL request where the *S*(*Tran*(R)) part will be a sequence of C_i attributes, prefixed by the variable names f_var_i and l_var_i which have been used to specify each attribute, followed by *S*(*Tran*(SRs)) parts of the sub-requests SRs. The C_i attribute is the third level element of the Xpath expression obtained by replacing the f_var_i or l_var_i occurrence of Results by the corresponding Xpath expression f_exp_i or l_exp_i.

The *F*(*Tran*(R)) part of this SQL request will be a sequence of f_var_i and l_var_i of the for and let clauses, which have prefixed the attributes of the *S*(*Tran*(R)) part, preceded by the relational table names followed by *F*(*Tran*(SRs)) parts of the sub-requests SRs. Each relational table name is the second level element of the Xpath expression obtained by replacing the f_var_i or l_var_i occurrence of Results by the corresponding Xpath expression f_exp_i or l_exp_i.

The *W*(*Tran*(R)) will be obtained by replacing, in the *condition* function of the R XQuery request, the Xpath expressions of the BFs and JFs boolean expressions by a sequence of B_i attributes, prefixed by the variable names f_var_i and l_var_i of these Xpath expressions, followed by *W*(*Tran*(SRs)) parts of the sub-requests SRs. The B_i attribute is the third level element of the Xpath expression obtained by replacing the f_var_i or l_var_i occurrence of the Xpath expression of the BFs and JFs boolean expression by the corresponding Xpath expression f_exp_i or l_exp_i.

4 TRANSFORMATION OF THE RESULTS OF A RELATIONAL REQUEST INTO AN XML DOCUMENT

The result of an SQL request corresponding to *Tran*(R) consists of a set of tuples. These tuples contain redundant information due to the absence of nesting in the relational model. The results of each sub-request are repeated as often as the number of existing values in the next sub-request result. We assume that the SRs sub-requests are ordered according to their position in the return clause of the R XQuery request. The construction of the XML document corresponding to the tuple of the Tran(R) SQL request result, is done in two stages. The first one consists at first to extract, for each sub-request of SRs, the corresponding results by eliminating the redundancy due to the flat nature of the relational tuples, then to extract the result corresponding to the Xpath expressions of Results of the return clause of R XQuery request. In the second stage, we built the XML document by tagging, thanks to the *XML_doc* tagging function of the return clause of the R XQuery request, each result corresponding to Xpath expressions of Results followed by the SRs sub-request results linked to this result. The SRs sub-request results are tagged, at their turn, thanks to thanks to the *XML_doc* tagging function of their return clauses. The final XML document is obtained by tagging the previously obtained XML document by <Global_result> and </Global_result>.

The XML schema associated with the XML document result corresponding to the result provided by the relational engine will be generated automatically. The construction of this XML schema will be made, thanks to the *XML_schema* function, from the return clause of the R Xquery request and the table schemas used by the relational engine. At the first level, this XML schema will contain the opening tag <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"> and the closing tag </xs:schema>. These tags will cover a second level which will contain the definition of a compound global element named Global_result generated by the *Compound_global_element* function (first section). The building of the XML schema continues by replacing each opening tag <tag> of the return clause of the R XQuery request, except the tags of the last level elements, by the following declaration tags of the XML schema language:

```
<xs:element name="tag">
        <xs:complexType>
                <xs:sequence>
```

Then each closing tag </tag> of the return clause of the R XQuery request will be replaced by the following corresponding closing tags of the XML schema language:

```
                </xs:sequence>
        </xs:complexType>
</xs:element>
```

The last level tags of the return clause and their contents will be replaced by description tags of simple elements generated by the *Simple_element* function (first section). The building of the XML schema will continue recursively in the same way for the sub-requests SRs.

Note: The information on keys and reference links do not appear in the XML schema of the results. Their meaning depends on the original relational database from which the results are extracted.

5 QUERYING OF XML AND RELATIONAL CONTEXTS

In this section, we present an extension of the Xquery request use such as it was formulated in the second section in order to query two heterogeneous databases: a documentary database and a relational database as it is shown in Figure3. The $\$f\_var_i$ and $\$l\_var_i$ variables will be used to reach either the documentary database by specifying the name of the XML document (*document.xml*) or the relational database by specifying the name of the XML schema (*schema.xsd*) which describes an XML view on this relational database. In this case, the Xquery request will be broken up into two sub-requests: one

concerning the relational database and the other the documentary database.

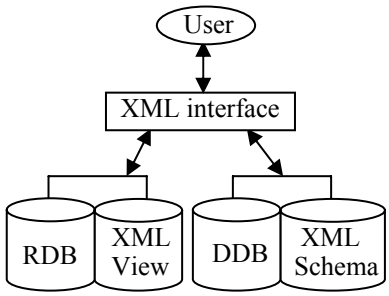

Figure 3: Querying of two heterogeneous databases (documentary and relational)

The various stages which enable us to obtain the final result of the Xquery request querying the relational database and the documentary database are illustrated in Figure4. The querying of the relational engine is done as it is described in Figure2 (second section). In the synthesis stage, the results of the two sub-requests will be transformed into an XML document corresponding to the return clause of the initial R Xquery request. When it is necessary, a rewriting stage precedes the decomposition of the Xquery request in order to facilitate the latter. The R Xquery request, which was modeled in the second section, will be rewritten into an Rt Xquery request by modifying only the contents of the where clause. The rewriting of the R Xquery request is done in two phases. The first one consists in gathering boolean expressions BFs into BF_Es on the same variable in the where clause. The contents of the where clause of the Rt Xquery request, after the first phase, will have the following form:

condition_t1(BF_Es, JFs)

The second phase of the rewriting stage consists in gathering the FB_Es boolean expressions and the JFs joint boolean expressions on the variables which query the same database (relational or documentary). The rewriting stage will apply recursively in the same way on the where clauses of the SRs sub-requests. After the second phase, the contents of the where clause of the Rt Xquery request will have the following form:

condition_t2(RDB_Es, DDB_Es, JF_SYNs)

Where:

- RDB_Es is the gathering of the BF_Es boolean expressions and the JFs joint boolean expressions relating to the variables which refer to the relational database.
- DDB_Es is the gathering of the BF_Es boolean expressions and the JFs joint boolean expressions relating to the variables which refer to the documentary database.
- The JF_SYNs expressions are joint boolean expressions of synthesis relating to two variables: one which refers to the relational database and the other which refers to the documentary database. For synthesis reasons of the results coming from the relational database and the documentary database, there must be at least one joint boolean expression of this last type.

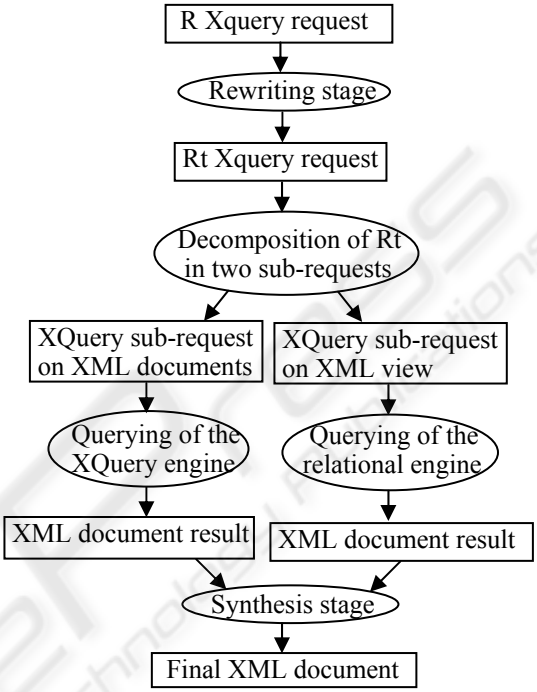

Figure 4: Query plan of a documentary database and a relational database

The stage of decomposition consists in breaking up the Rt Xquery request obtained at the rewriting stage into two sub-requests: the first one intended for the relational database and the second one intended for the documentary database. The for and let parts of the XQuery sub-requests RDB_R and DDB_R will contain the for and let parts of the XQuery request Rt concerning the variables referencing respectively the RDB and DDB. This distinction will be done according to the name of the XML document (.xml) or the name of the XML schema (.xsd) referenced in the f_exp_i or l_exp_i Xpath expressions. The where parts of the XQuery sub-requests RDB_R and DDB_R will be constituted respectively by RDB_Es and DDB_Es expressions obtained after the rewriting stage. The return parts the XQuery sub-requests RDB_R and DDB_R will be the result of the decomposition of the *XML_doc* tagging function into two parts which are respectively: the tags concerning variables which refer the RDB and the tags concerning variables which refer the DDB. This decomposition will apply recursively on the SRs sub-requests.

If the attributes of the synthesis joint are not mentioned in the return clause of an Rt Xquery

request, these attributes must be added in the return clauses of the two sub-requests by tagging them in order to be able to identify them. In the same way if after the decomposition, each one of DDB_R and/or RDB_R has at least one nested sub-request with a joint condition between one of its attributes and one of the enclosing request and if this attribute does not appear in the return clause then we must to add it in the return clauses of the request and the sub-request. The results of the two DDB_R and RDB_R sub-requests will be stored respectively in the two XML documents: res1.xml and res2.xml which has respectively as compound global element: DDB_global_result and RDB_global_result. The synthesis joints will be treated after obtaining the results of the two DDB_R and RDB_R sub-requests by using the SYN_R synthesis sub-request which is generated automatically.

The SYN_R synthesis sub-request will be constituted by two for clauses with two variables $\$r_1$ and $\$r_2$ in order to query respectively the two result XML documents res1.xml and res2.xml. These two variables $\$r_1$ and $\$r_2$ specify the XML element corresponding respectively to the first tags of the return clauses of the DDB_R and RDB_R sub-request. The where part of SYN_R sub-request will contain the synthesis joint expressions SYN_JFs. The return parts the XQuery sub-requests SYN_R will be constituted by the *XML_doc* tagging function of Rt applied, for each Results Xpath expression of the Rt XQuery request return clause, to the equivalent Xpath expression in the DDB_R or RDB_R sub-request. The building of SYN_R will continue recursively in the same way on the SRs sub-requests.

The XML schema describing the structure of the XML document result of the SYN_R synthesis sub-request will be built from the XML schema describing the XML view on the relational database, the XML schema of the documentary database and the return clause of the XQuery request R. The types of the simple elements making up the XML document result are deduced from the original schema. In the case of a simple element coming from the XML view of the relational database, the type of this element is deduced form the XML schema describing this view. If this element is coming from an XML document of the documentary database, the type of this element depends on the existence of the document XML schema. If the document does not possess an XML schema, the type of the simple element will be *anyType*. The building of the XML schema of the synthesis sub-request SYN_R result will be done according to a similar method than that used for the building of the XML schema of the document created from the tuples returned by the relational engine thanks to the *XML_Schema* function described in the third section.

6 CONCLUSION

Our system takes advantage of relational systems which are powerful for the management of structured data and it takes advantage of XML systems for the management of non-structured or semi-structured data. It is in this meaning where our work is different from the systems which store XML documents in the shape of tables. In these systems, it is often non-optimal to reconstitute the document. Our system enables us to build results made up of non-structured data, coming from documentary database, and structured data, coming from relational database, in the shape of an XML document. These results could then be consulted as persistent documents thanks to an Xquery request.

This work is intended to be extended to a distributed system where relational and documentary data could be on geographically distributed sites. In this distributed system, the XML schema will be the pivot schema and the Xquery language will be the pivot language.

REFERENCES

Fan, C., Funderburk, J., Lam, H.I., Kiernan, J., Shekita, E., Shanmugasundaram J., 2002. XTABLES : Bridging Relational Technology and XML. IBM Corporation.

Fernandez, M., Kadiyska, Y., Morishima, A., Suciu, D., Tan W.C., 2002. SilkRoute : a framework for publishing relational data in XML. *ACM Transactions on Database Technology*.

Fernandez, M., Tan, W.C., Suciu D., 2000. SilkRoute : Trading between relations and XML. *International Conference WWW'00*.

Hernandez, M., Miller, R. J., Haas L. M., 2001. Clio: A Semi-Automatic Tool For Schema Mapping. *International conference SIGMOD'01*. ACM Press.

INRIA., 1998. LeSelect (CARAVEL project). http://www-caravel.inria.fr/~leselect/

Kenab, M., Ould Braham, T., Bazex P., 2004. Evaluation of a document database description by different XML schemas. *International conference IASTED: DBA'04*. Acta Press.

Kenab, M., Ould Braham, T., Bazex P., 2004. Parameterized Formatting of an XML document by XSL rules. *International conference ADVIS'04*. Springer-Verlag.

Manolescu, I., Florescu, D., Kossmann, D., Olteanu, D., Xhumari F., 2000. Agora: Living with XML and relational. *International conference VLDB*. Morgan-Kaufmann.