Translate Graphical XML Query Language to SQLX

Wei Ni Tok Wang Ling

Department of Computer Science, National University of Singapore, Singapore
E-mail: {niwei, lingtw}@comp.nus.edu.sg

Abstract: Semi-structured data has become more and more attention-getting with the emergence of XML, and it has aroused much enthusiasm for integrating XML and SQL in database community. Due to the complexity of XQuery, graphical XML query languages have been developed to help users query XML data. In this paper, we propose a new XML-to-SQL solution on the base of ORA-SS, a rich semantic model for semi-structured data. We model the data by ORA-SS schema and store them in an ORDB. Based on ORA-SS, we developed a graphical XML query language GLASS that not only expresses the query constraints and reconstruction structure in XML view but also the relational semantic in the XML view. This paper focuses on the translation algorithm from GLASS to SQLX, an XML extension on traditional SQL.

1. Introduction

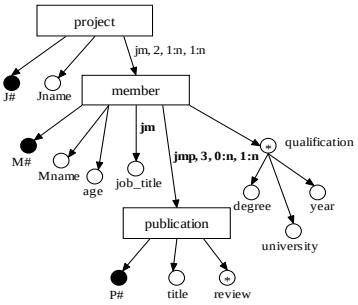
XML has been accepted as a potential standard for data publishing, exchanging and integration on the web; and there is much enthusiasm for integrating XML and traditional object-relational data model, which will benefit from the fruit of over 30 years research on object-relational technology. Meanwhile, since XQuery[12] and other text-based functional languages are complex and difficult to common users, researchers have proposed graphical languages and graphical user interfaces (GUIs), such as Equix[3]/BBQ[7, 9], XML-GL[1, 2, 4], XMLApe [8], QURSED[11], etc, to make the XML query easier to use.

In this paper, we use ORA-SS (Object-Relational-Attribute model for Semi-Structured data) [5], a rich semantic data model for semi-structured data, to describe the XML schema; and the XML data are stored in an Object-Relational Database (ORDB). To query the data, we have designed a graphical language GLASS (Graphical query Language for Semi-Structured data) [10] with full consideration of relational semantic information in ORA-SS, which has stronger expressive power than other graphical XML query languages. And we translate the GLASS query into SQLX[6], an expansion of SQL which is often used as a publishing tool from relational table to XML file.

The rest of this paper is organized as follows. Section 2 is a brief presentation of ORA-SS model and the mappings from ORA-SS schema to ORDB schema. In Section 3, we introduce the GLASS query with an example. Section 4 discusses the translation from GLASS to SQLX based on the query example in Section 3. And before the end, we conclude this paper and highlight the future works in Section 5.

2. ORA-SS model and the storage of XML data

Compared with DTD, XML Schema [14], OEM, Dataguide, XML Graph [1] and their equivalents, ORA-SS is a rich semantic data model for catching the relational information. The most significant feature of ORA-SS is that it not only represents the tree structure of the original schema in terms of object class, relationship types and attributes but also distinguishes relationship attributes from object attributes, etc. For example, the ORA-SS schema in Fig. 1(a) contains three object classes (*project*, *member* and *publication*) and two relationship types (binary relationship type *jm* between project and member; and ternary relationship type *jmp* among project, member and publication). The label "jm" on the arrow from member to job_title indicates that *job_title* is an attribute of the relationship type *jm*, i.e., the job_title attribute is determined by both *project and member* rather than member only. It should be emphasized that the object ID in ORA-SS (the attributes denoted as solid circles in the diagram such as *J#*) is different from the object identifier in OEM. In ORA-SS, the object ID identifies each unique object instance rather than each element instance (in OEM). For example, if one member attends two projects, the same member instance may appear as two element instances in the XML data. In ORA-SS, both use the same object ID (*M#*); but in OEM, they will have different object identifiers.



(a) An example ORA-SS schema

Object Relations
project (J#, Jname)
member (M#, Mname, age
qualification(degree,
university,
year)*)
publication (P#, title, (review)*)
Relationship Relations
jm (J#, M#, job_title)
jmp (J#, M#, P#)

(b) The ORDB storage schema of (a)

Fig. 1. An example of ORA-SS schema and its ORDB storage schema.

When we store the XML data in an ORDB, each object class will be stored in an *object relation* with its object attributes and each relationship type will be stored in a *relationship relation* with its relationship attributes. Composite attributes (e.g. the *qualification* in the example in Fig. 1.) will be stored as a *nested relation* inside an object relation or relationship relation according to the ORA-SS schema. Fig. 1(b) presents the ORDB schema when we store the XML data conforming to the ORA-SS schema in Fig. 1(a).

3. GLASS query

GLASS is a graphical XML query language designed on the base of ORA-SS schema. The most significant features of GLASS from other graphical XML query languages (or GUIs) are:

(1) *GLASS separates the complex query logic from the query graph (the query graph is the graphical part of a GLASS query)*. This feature makes the GLASS query clear and concise even if it contains complex query logic with quantifiers and negation.
(2) *GLASS considers relationship types in querying XML data*. The relationship types could be those either defined in ORA-SS schema or derived from the schema. This feature enables the GLASS to define the query semantic precisely.

A typical GLASS query consists of four parts:

(1) **Left Hand Side Graph** (LHS graph) – denotes the basic conditions of a query, which presents the fundamental features that users interest in.
(2) **Right Hand Side Graph** (RHS graph) – defines the output structure of the query result, which is a compulsory part in the GLASS query.
(3) **Link Set** – specifies the bindings between the RHS graph and LHS graph. When two graph entities are *linked*, they are visually connected by a line, which means the data type and value of the entity in the RHS graph are from the corresponding linked entity in the LHS graph.
(4) **Condition Logic Window** (CLW) – It is an optional part where users write conditions and constructions that are difficult to draw, which includes Logic expressions, Mathematic expressions, Comparison expressions and IF-THEN statements.

Most notations in GLASS are borrowed from those in ORA-SS schema diagram, yet some new notations are introduced to represent the query condition and result reconstruction such as *Box of group entities* and *Condition Identifier*. The box of group entities is used to specify multi-field aggregations such as the query in Example 1. The condition identifier is defined by user, quoted by a pair of ":"s, which specifies a connected sub-graph in the LHS graph (e.g. the condition identifier "A", appears to be ":A:" on the arrow from member to age).

Example 1. (For the schema in Fig. 1.)

*Find the member whose age is less than 35, and he **either** has taken part in less than 5 projects **or** written more than 6 publications in some of the projects he attended; display the member id and name.*

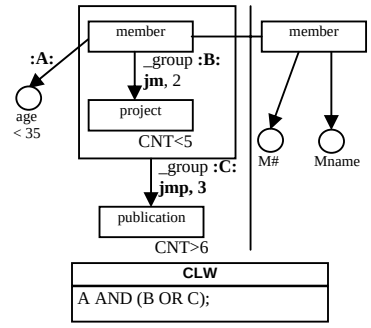


Fig. 2. The GLASS query of Example 1.

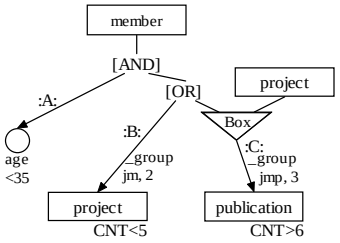


Fig. 3. The condition tree of the GLASS query in Fig. 2 from its LHS graph and CLW.

Fig. 2 shows the GLASS representation of Example 1. The result view structure is defined in the RHS graph. The object node with name "member" is linked with the object node with the same name in the LHS graph, which means the member in the RHS graph (the result view) is from the member in the LHS graph. In the LHS graph, there are three condition identifiers "A", "B" and "C" where "A" means *member* should have *age* attribute less than

35; "B" means to group *project* under *member* in the binary relationship type *jm* having count of *project* less than 5; and "C" means to group *publication* under each pair of *member* and *project* in the ternary relationship type *jmp* having count of *publication* more than 6. By default, the logic among the query conditions are "AND"; but it can be rewritten by the logic expressions specified in the CLW with the help of condition identifier. In the above example, the logic among the three conditions is defined as "A AND (B OR C)".

4. Translate GLASS into SQLX

In this section, we discuss the translation from GLASS to SQLX. SQLX (aka. SQL/XML) is an XML-related specification expanded on SQL. The syntax of SQLX combines the features in both XML document processing and the traditional SQL. Before introducing the translation algorithm, we shall preprocess the GLASS query as follows.

4.1 Preprocess

Observing the SQLX syntax [6], we find that, in translating from GLASS to SQLX query, the SELECT and FROM clauses can be easily generated by checking the GLASS query and the ORA-SS schema; and the major task in translation is *the generation of query conditions*. To generate the query conditions and the target SQLX expression, we need preprocess the GLASS query in the following 3 steps.

(1) Expansion of the simple projection;
(2) Expansion of the abbreviated RHS graph;
(3) Construction of the condition tree from LHS graph and CLW.

The GLASS query appears with RHS graph only when expressing simple projections that do not contain any constraints in the LHS graph. For such kind of query, we regard the LHS graph as null in translation.

The GLASS query supports abbreviated representation in defining the result XML view in the RHS graph especially when all attributes of an object class are extracted; and it is necessary for us to expand the RHS graph to get a full-version ORA-SS view schema of the result before we translate the query.

The condition tree is a labeled graph containing all query constraints in both the LHS graph and the CLW. The role of the condition tree in GLASS is similar to the condition tree in TQL (Tree Query Language) defined in [11]. Nevertheless, the condition tree in GLASS contains more features than TQL by including quantifiers, relationship type information and aggregation. The purpose of the condition tree is to combine the query constraints into one graph so that we can generate WHERE clauses by traversing the condition tree. The condition tree is initially a copy of the LHS graph in a given GLASS query, which can be a forest if the LHS contains multi-graphs. When we copy the LHS graph, we record the aggregation information, add the relationship type information, mark the condition identifiers in the condition tree and insert the logic operators (and quantifiers) according to the expressions in CLW. Particularly, the box is represented as a composite node (a triangular node) in the condition tree. The condition tree of the GLASS query in Fig. 2 is shown in Fig. 3.

4.2 Translation Algorithm

The basic idea of the translation algorithm is to traverse the expanded RHS graph in depth-first order and generate the nested SQLX query blocks according to the tree structure. The XML construction functions in SQLX can be different due to different the data types (element or attribute) in the result XML view. Like the traditional XML-to-SQL method, parent-child/ancestor-descendant relations among different object classes are performed by a series of join operations, which can be obtained from the relationship type information in the ORA-SS schema. The query constraints in the LHS graph (and the CLW) are *only* for the nodes with links in the RHS graph. Checking the condition tree, we generate WHERE clauses (denoted as W(N)) in different forms by applying the following rule.

Rule (Generate WHERE clauses of Node N from the condition tree):

If N is an attribute node, then W(N) is the value comparison expressions on N.

If N is an object class, N has k child nodes (say C_1 to C_k); and it is associated in a relationship type of D degree where the D-1 ancestor nodes of N are P_1 … P_{D-1}; then W(N) is

CASE 1. there is a (negated) existential quantifier in front of NL, then we generate

WHERE [NOT] EXIST (SELECT N FROM …

$$W(C_1)\theta^1W(C_2)\theta^2\ldots\theta^{k-1}W(C_k)\theta^kW(P_1)\theta^{k+1}W(P_2)\theta^{k+2}\ldots\theta^{k+D-2}W(P_{D-1}))$$

CASE 2. there is a "_group" label follows N, then we generate

WHERE N IN (SELECT DISTINCT N FROM…

$$W(C_1)\theta^1W(C_2)\theta^2\ldots\theta^{k-1}W(C_k)\theta^kW(P_1)\theta^{k+1}W(P_2)\theta^{k+2}\ldots\theta^{k+D-2}W(P_{D-1}))$$

CASE 3. there is a "_group" label before N under object class M, then we generate

WHERE N IN (SELECT N, AGG(N) FROM …

$$W(C_1)\theta^1W(C_2)\theta^2\ldots\theta^{k-1}W(C_k)\theta^kW(P_1)\theta^{k+1}W(P_2)\theta^{k+2}\ldots\theta^{k+D-2}W(P_{D-1})$$

GROUP BY M

HAVING value comparison on AGG(N))

CASE 4. for all other cases, we generate

WHERE N IN (SELECT N FROM …

$$W(C_1)\theta^1W(C_2)\theta^2\ldots\theta^{k-1}W(C_k)\theta^kW(P_1)\theta^{k+1}W(P_2)\theta^{k+2}\ldots\theta^{k+D-2}W(P_{D-1}))$$

where θ^m (m = 1, …, k, k+1, …, k+D-2) are the logic operators ("AND" or "OR"). To avoid repeating generation the where clauses of the same node, we exclude node N when we generate each $W(P_i)$ (i = 1, …, D-1); and we ignore the parent nodes when generating each $W(C_j)$ (j = 1, …, k).

```
SELECT XMLELEMENT(NAME "member",
          XMLATTRIBUTES (M1.M# AS "member_id")
          XMLELEMENT (NAME "Mname", M1.Mname)
)FROM member M1
       WHERE M1.age <35
       AND (M1.M# IN (SELECT DISTINCT M# FROM member
                  WHERE (SELECT COUNT(J#) FROM jm
                             WHERE M# = jm.M#)<5)
       OR    M1.M# IN (SELECT DISTINCT M# FROM member
                  WHERE (SELECT DISTINCT J# FROM project
                             WHERE(SELECT COUNT(P#) FROM jmp
                                        WHERE jmp.J# = project.J#
                                          AND jmp.M# = member.M#)>6)))
```

Fig. 4. The translated SQLX expression of the Example in Fig. 2.

It should be emphasized that $W(P_i)$ is indispensable when

(1) *N is not the root in the condition tree, and*

(2) *In the RHS graph, N does not have any parent/ancestor nodes that are linked*

with their counterparts in the LHS graph.

Otherwise, $W(P_i)$ can be omitted.

Applying the above method and rules to the example in Fig. 2, we can get the SQLX expressions of the query as shown in Fig. 4.

5. Conclusion and future work

In this paper, we have introduced a new XML-to-SQL query solution based on ORA-SS and discussed the translation from GLASS, the graphical query language in our project, to SQLX. Compared with other graphical XML query languages (and GUIs), GLASS define both the structure and the relational semantic in the XML view; and the GLASS query is seamlessly cohered with the ORDB and SQL/SQLX on ORA-SS schema. Compared with traditional XML-to-SQL solutions that use XQuery or XPath [13], GLASS has stronger expressive power and provides an easy-to-use interface. So far, the case tool of GLASS query has been partially implemented. As to the future work, it may include the query optimization of GLASS queries and the translated SQLX expressions as well as the improvement of the case tool.

References

1. S. Ceri, S.Comai, E. Damiani, P.Fraternali, S. Paraboschi, and L.Tanca. XML-GL: a graphical language of querying and restructuring XML documents. In Proc. WWW8, Toronto, Canada, May 1999.
2. S. Ceri, S. Comai, E. Damiani, P. Fraternali, and L. Tanca. Complex Queries in XML-GL. SAC(2) 2000:888-893.
3. S. Cohen, Y. Kanza, Y. Kogan, W. Nutt, Y. Sagiv and A. Serebrenik. Equix – Easy Querying in XML Databases. In proceedings of Webdb'98 – The Web and Database Workshop, 1998.
4. S. Comai, E. Damiani, P. Fraternali. Computing Graphical Queries over XML Data. ACM Transactions on Information Systems, Vol. 19, No. 4, October 2001, Pages 371-430.
5. G. Dobbie, X. Y. Wu, T. W. Ling, M. L. Lee. ORA-SS: An Object-Relationship-Attribute Model for Semistructured Data. TR21/00, Technical Report, Department of Computer Science, National University of Singapore, December 2000.
6. Information technology -- Database languages -- SQL -- Part 14: XML-Related Specifications. ISO/IEC 9075-14:2003
7. B. Ludaescher, Y. Papakonstantinou, and P. Velikhov. Navigation-driven evaluation of virtual mediated views. In *Proceedings of the sixth International Conference on Extending Database Technology* (*EDBT*)(Konstanz, Germany, March), Lecture Notes in Computer Science, vol. 1777, Springer-Verlage, New York, 2000.
8. L. Mark, etc. XMLApe. College of Computing, Georgia Institue of Technology. http://www.cc.gatech.edu/projects/XMLApe/
9. K. D. Munroe, B. Ludaescher and Y. Papakonstantinou. Blended Browsing and Querying of XML in Lazy Mediator System. Konstanz, Germany, March 2000.
10. W. Ni, T. W. Ling. GLASS: A Graphical Query Language for Semi-Structured Data. DASFAA 2003.
11. Y. Papakonstantinou, M. Petropoulos and V.Vassalos. QURSED: Querying and Reporting Semistructured Data. ACM SIGMOD 2002, Jun 4-6, Madison, Wisconsin, USA.
12. XQuery 1.0: An XML Query Language. W3C Working Draft 22 August 2003 http://www.w3.org/TR/xquery/
13. XML Path Language (XPath) 2.0. W3C Working Draft 22 August 2003 http://www.w3.org/TR/xpath20/
14. XML Schema. http://www.w3.org/XML/Schema