Accessing Relational Databases via XML Schema

Chengfei Liu[1], Minyi Guo[2], Jixue Liu[1]

[1] Advanced Computing Research Centre, School of Computer and Information Science, University of South Australia, SA 5095, Australia
[2] Department of Computer Software, The University of Aizu, Aizu-Wakamatsu City, Fukushima, 965-8580, Japan

Abstract. This paper proposes a new architecture for publishing relational data as XML documents. The architecture allows users to access relational databases via XML schema which is transformed from the underlying relational database schema. XML queries can be issued directly against the XML schema and then be translated into corresponding SQL queries. The SQL query results are converted into XML documents and returned to users. In the paper, the architecture and its components are presented.

1 Introduction

Recently, XML has emerged as the standard for data exchange and publication on the Web. However, most business data of enterprises has been stored and will continue to be stored in relational database management systems. Consequently, there have been a lot of research interests in publishing relational data as XML documents [5, 3]. One approach to publish relational data is to create XML views of the underlying relational data. SilkRoute [5] is one of the systems taking this approach. In SilkRoute, XML views of a relational database are defined using a relational to XML transformation language called RXL, and then XML-QL queries are issued against views. The queries and views are combined together by a query composer and the combined RXL queries are then translated into corresponding SQL queries. XPERANTO [3] takes a similar approach but uses XQuery [2] for user queries.

In this paper, we propose a new approach to publish relational data as XML documents by translating the underlying relational schema into equivalent XML schema. The translated XML schema preserves integrity constraints defined in the relational database schema, it is also normalized. After the XML schema is created, XQuery queries can be issued directly against the XML schema and then be translated into the corresponding SQL queries against the underlying relational schema. As such, the query translation gets simplified. Finally, the SQL query result relations are re-formatted and converted into XML documents. The remainder of the paper presents the architecture and its components.

2 The Proposed Architecture

There are three components in our proposed architecture: a schema translator, a query translator, and an XML document generator.

2.1 Schema Translator

The schema translator is responsible for translating a relational database schema into the corresponding XML schema. XML Schema [4] is chosen for defining XML schema because Data Type Definition (DTD) has some limitations, e.g., extremely limited data typing, insufficient support for identity and reference.

Integrity constraints are associated with a relational schema, e.g., primary keys (PKs), foreign keys (FKs), null/not-null, unique, etc. The null/not-null and unique constraints can be represented easily in XML Schema. Therefore, we focus on the mapping of PK/FK constraints. We set two goals in schema transformation: no data redundancy introduced, and high level of nesting.

XML Schema supports two mechanisms to represent identity and reference: one is ID/IDREF while the other is KEY/KEYREF. There are differences in using these two mechanisms. The former supports the dereference function in path expressions in most XML query languages, however, it only applies to a single element/attribute. The latter may apply to multiple elements/attributes but does not support the dereference function. For schema translation, we use ID/IDREF where possible because of the dereference function support. For this purpose, we differentiate the single attribute primary/foreign keys from multi-attribute primary/foreign keys. We also classify a relation into the following categories based on different types of primary keys: *regular* relation where the primary key contains no foreign keys, *component* relation where the primary key contains one foreign key, *supplementary* relation where the primary key as a whole is also a foreign key, and *association* relation where the primary key contains more than one foreign keys.

For a relational database schema, we create a root element in the target XML schema. For a regular or an association relation, we create an element with the same name as the relation schema and put it under the root element. The created element may be moved down later depending on some constraints. For a component or a supplementary relation, an element is created and placed as a child element of the element for its parent relation. For a non-key attribute, an element is created under the element for its relation.

For each single attribute primary key of a regular relation R, an attribute of the element for R is created with ID data type. For each multiple attribute primary key of a regular, a component or an association relation R, an attribute of the element for R is created for each primary key attribute with its corresponding data type, a *key* element is defined with a *selector* to select the element for R and several *fields* to identify all primary key attributes.

For each foreign key of a relation R except one which is contained in the primary key of a component or a supplementary relation, if it is a single attribute foreign key, an attribute of the element for R is created with IDREF data type; otherwise, an attribute is created for each foreign key attribute with its corresponding data type, a *keyref* element is defined with a *selector* to select the element for R and several *fields* to identify all the foreign key attributes.

To achieve high level of nesting, the position of the above created elements can be adjusted. If a relation R_1 has a NOT-NULL foreign key which references another relation R_2 and there is no loop between R_1 and R_2 in terms of reference relationship, we can move the element for R_1 to under the element for R_2.

Detailed mapping rules and examples can be found in [7].

XViews [1] constructs graph based on PK/FK relationship and generate candidate views by choosing node with either maximum in-degree or zero in-degree as root element. The candidate XML views are of high level of nesting but suffer considerable level of data redundancy. NeT [6] derives nested structures from flat relations by repeatedly applying *nest* operator on tuples of each relation. The result nested structures may be useless because the derivation is not at the type level. Compared with XViews and NeT, we introduce no data redundancy while achieve high level of nesting.

2.2 Query Translator

The query translator is responsible for translating XML queries issued against XML schema into the corresponding SQL queries against the underlying relational database schema. XQuery [2] is chosen as the XML query language since it is currently being standardized by W3C. In this paper, we only consider the translation of basic XQuery queries which do not include aggregate functions. The main structure of an XQuery query can be formulated by an FLWR expression with the help of XPath expressions. FOR and LET clauses serve to bind values to one or more variables using (path) expressions. The FOR clause is used for iteration, with each variable in FOR iterates over the nodes returned by its respective expression; while the optional LET clause binds a variable to an expression without iteration, resulting in a single binding for each variable. As the LET clause is usually used to process grouping and aggregate functions, the processing of the LET clause is not discussed here. The optional WHERE clause specifies conditions to restrict the binding-tuples generated by FOR and LET clauses. The RETURN clause is used to specify an element structure and to construct the result elements in the specified structure. As our XML schema is mapped from the underlying relational database schema, we know the reverse mapping which is crucial for query translation.

To translate an XQuery query Q_{xquery} to the corresponding SQL query Q_{sql}, we first make Q_{xquery} canonical, i.e., to move the test conditions in path expressions to the WHERE clause, iteration variables may be added in the FOR clause. Then we identify all relations which are normally indicated by the end labels of all path expressions in the FOR clause. After that, we identify all target attributes for each identified relation. All target attributes appear in the RETURN clause as leaf elements or attributes. Finally, we identify conditions. All conditions in the WHERE clause of Q_{xquery} are moved to the WHERE clause of

Q_{sql}. In addition, extra join conditions between identified relations representing an element and its parent element are added.

If a nested FLWR expression appears in the RETURN clause of Q_{xquery}, the above translation procedure needs to be done recursively.

2.3 XML Document Generator

After the execution of a translated SQL query, the result flat relation is passed to the XML document generator which is responsible for generating the result XML document. From the above query translation procedure, the SQL query takes all leaf elements or attributes defined in an XQuery query RETURN clause and output them in a flat relation. Therefore, we are able to apply a *nest* operator on the flat relation several times to generate the nested structure as specified in the original XQuery query.

3 Conclusion and Future Work

This paper proposed a new architecture for publishing relational databases as XML documents. The schema translator, query translator, and XML document generator of the architecture were presented. The architecture presents to user with a clear XML schema which is normalized and preserves integrity constraints defined in the underlying relational schema. It also simplifies query translation process as compared with SilkRoute and XPERANTO. We believe that the proposed architecture is an effective and practical architecture for accessing relational databases via XML. In the future, we will investigate the query translation of complex XQuery queries.

References

1. C. Baru. Xviews: Xml views of relational schemas. In *Proceedings of DEXA Workshop*, pages 700–705, 1999.
2. S. Boag, D. Chamberlin, M. Fernandez, D. Florescu, J. Robie, J. Simeon, and M. Stefanescu. *XQuery 1.0: An XML Query Language*, April 2002. W3C Working Draft, http://www.w3.org/TR/2002/WD-xquery-20020430/.
3. M. Carey, J. Kiernan, J. Shanmugasundaram, E. Shekita, and S. Subramanian. Xperanto: Middleware for publishing object-relational data as xml documents. In *Proceedings of VLDB*, pages 646–648, 2000.
4. D. Fallside. *XML Schema Part 0: Primer*, May 2001. W3C Recommendation, http://www.w3.org/TR/xmlschema-0/.
5. M. Fernandez, W. Tan, and D. Suciu. Silkroute: Trading between relations and xml. In *Proceedings of WWW*, pages 723–725, 2000.
6. D. Lee, M. Mani, F. Chiu, and W. Chu. Nesting-based relational-to-xml schema translation. In *Proceedings of the WebDB*, pages 61–66, 2001.
7. C. Liu, J. Liu, and M. Guo. On transformation to redundancy free xml schema from relational database schema. In *Proceedings of APWeb (to appear)*, 2003.