Automatic Generation of SQLX View Definitions from ORA-SS Views

Ya Bing Chen, Tok Wang Ling, Mong Li Lee

School of Computing, National University of Singapore
{chenyabi, lingtw, leeml}@comp.nus.edu.sg

Abstract. Although XML is the dominant standard for publishing and exchanging data for Internet-based business applications, data is typically stored in relational or object-relational databases. Thus, it is necessary to define XML views over these traditional databases. Unfortunately, it is not easy for users to manually write SQLX queries to define the XML views. This paper describes a method to automatically generate SQLX view definitions from object-relational databases. We utilize the semantically rich ORA-SS data model to capture the schematic structure and semantics of the underlying data. Valid ORA-SS views are first designed on the ORA-SS schema, before they are mapped to XML views. The generated view definitions are SQL queries with XML extension (SQLX) that can be directly evaluated on object-relational databases to materialize the views. This approach removes the need to manually write executable view definitions for the XML views, and provides a user-friendly interface to retrieve XML data via views.

1 Introduction

In this work, we consider XML views in a particular case where the XML data are stored in an object-relational database. The conceptual schema for XML views are first extracted using the semantically rich ORA-SS data model [6]. The semantics captured in the ORA-SS data model are then used to map the XML data into a set of nested tables in the object-relational database [9]. This storage method avoids the unnecessary redundancies that exist in the case where XML data are stored in XML files. Next, valid XML views are designed based on the ORA-SS source schema with the query operators defined in [3]. The designed XML views are also expressed in ORA-SS schema, which is not executable on the underlying database. In order to generate the result of the XML views, we use SQLX [10] to express the ORA-SS views. SQLX are SQL queries with XML extension (SQLX) which can be directly evaluated in the object-relational database to produce XML documents.

However, it is difficult to manually write SQLX view definitions for the XML views. Further, such definitions are not easy to understand. Thus, we propose an approach to automatically translate the designed ORA-SS views into SQLX query expressions. The approach removes the need for users to manually write complex SQLX view definitions. It can be used to materialize the views and map queries issued on ORA-SS views into the equivalent queries in SQLX syntax on the underlying database.

The rest of the paper is organized as follows. Section 2 presents the proposed approach to generate SQLX query definitions from valid ORA-SS views. Section 3 briefly reviews related works and concludes.

2 Generating SQLX Query Definitions

There are two steps to generate SQLX query expressions from ORA-SS views. First, we must discover how the relationship types in ORA-SS source schema are changed into new relationship types in the view schema so that we can identify the necessary mapping information when generating SQLX view definitions. Second, we generate the appropriate query block for different object classes and attributes in ORA-SS view schema. The following sub sections give the details of these two steps.

2.1 Relationship Types in ORA-SS Views

There are two relationship types in ORA-SS views. The first is the original relationship types that exist in the source schema, while the second is the new relationship types that are derived from original relationship types in the source schema. The new relationship types can be obtained in two ways during the design of ORA-SS views.

A. Project Existing Relationship Types.
New relationship types can be derived by projecting existing relationship types in the source schema.

Rule Proj: If an object class O_i ($1 \le i \le n$) is dropped in a view, then for each relationship type R involving O_i, we create

$$R' = \Pi_{key1,\ldots,keyi-1,\ keyi+1,\ \ldots,\ keyn}\ R$$

in the view and the attributes of R (say $attr_j$, j=1, 2,…,m) are dropped in default by the drop operator. R has the following storage relation schema:

$$R(\underline{key_1, key_2, \ldots, key_n}, attr_1, attr_2, \ldots, attr_m),$$

where key_1, key_2, …, key_n are the keys of O_1, O_2, …, O_n respectively, which also form the key of R, and $attr_1$, $attr_2$, …, $attr_m$ are the attributes of R.

B. Join Existing Relationship Types

New relationship types can also be derived by joining existing relationship types in source schema.

Rule Join: If all the object classes of relationship types R_1 and R_2 are in a continual path in the source schema, and R_1 and R_2 have common object classes and all the common object classes are dropped in a view, then R_1 and R_2 are joined based on their common object classes in the view to create

$$R' = \Pi_{keyi, \ldots key\,j} (R_1 \bowtie_{keyl, \ldots keym} R_2)$$

where key_l, … key_m are the keys of the common object classes of R_1 and R_2, while key_i, …, key_j are the keys of the rest object classes of R_1 and R_2 in the view. Thus, R' has the following relation schema

$$R'(\underline{key_i, \ldots, key_j}) .$$

Based on the above two rules, we can determine how relationship types are derived in ORA-SS views. This enables us to map the derived relationship types back to their corresponding original ones in source schema when generating condition constraints for an object class in view schema.

2.2 Generation Rules

Without the semantics of relationship types among object classes in the view, we have to consider all the ancestors of an object class in the view to generate the condition constraints. Fortunately, we have identified the various relationship types that can occur in an ORA-SS view. This allows us to identify which particular ancestors' values determine the value of an object class through the relationship types involved. In other words, based on the ancestors and the relationship types, we can generate the condition constraints for the object class.

Definition 1. Given an object class O in an ORA-SS view V, if an ancestor of O participates in a relationship type R with O in V, then the ancestor is called a *Determining Object Class* (DOC) of O in the view, and the relationship type R is called a *Determining Relationship Type* (DRT) of O in the view.

We employ the information of DRTs and DOCs of each object class to construct two rules to generate the condition constraints for an object class in an ORA-SS view. Rule Gen 1 generates condition constraints for the query expression of object class O in the case where the set of DRTs of O is not null in the view. Since the attributes of O are stored together with O in the underlying database, we do not need to consider the attributes of O separately when we generate the condition constraints for O. Rule Gen 2 generates appropriate query block for attributes of relationship types in the view.

Suppose the DRTs of O in the view is $\{R_1, R_2, \ldots, R_k\}$. There are three cases for R_i (i=1, …, k) in the DRTs which cover all the possible cases in which a relationship type in an ORA-SS view can be. For each case, we will give a sub-rule for generating condition constraints for O.

Rule Gen 1: If the set of DRTs of O in the view is $\{R_1, R_2, \ldots, R_k\}$, then

Case 1. If R_i ($1 \le i \le k$) is an original relationship type from the source schema and contains the DOCs of O (say $\{O_1, \ldots, O_i\}$), then we generate the following condition constraints in the Where clause of the query block of O:

$$O.key=R_i.O.key \text{ and } R_i.O_1.key=O_1.key \text{ and } \ldots \text{ and } R_i.\ O_i.key=O_i.key$$

The condition constraints indicate that only those instances of O are chosen which exist in an record of R_i whose values of $O_1, \ldots, O_i$ are equal to the current values of $O_1, \ldots, O_i$.

Case 2. If R_i ($1 \le i \le k$) is a derived relationship type generated by applying projection operators to an original relationship type R_i' in source schema and R_i contains the DOCs of O (say $\{O_1, \ldots, O_i\}$), then we use R_i' to generate the following condition constraints in the Where clause of the query block of O:

$$O.key=R_i'.O.key \text{ and } R_i'.O_1.key=O_1.key \text{ and } \ldots \text{ and } R_i'.\ O_i.key=O_i.key$$

The difference between Case 1 and Case 2 is that we replace R_i with the original one R_i' in Case 2. Without such rewriting, the condition constraints will not be executable because R_i is a virtual one only shown in the view.

Case 3. If R_i ($1 \le i \le k$) is a derived relationship type generated by applying join operators to several original relationship types (say $R_{i1}, R_{i2}, \ldots, R_{ij}$) in source schema and involves the DOCs of O (say $O_1, \ldots, O_i$) in the view, then we first generate the following condition constraints using R_i in the Where clause of the query block of O:

$$O.key = R_i.O.key \text{ and } O_1.key = R_i.O_1.key \text{ and } \ldots \text{ and } O_i.key = R_i.O_i.key$$

This condition constraints express the influence of DOCs ($O_1, \ldots, O_i$) on O through R_i in the view. However, R_i is a virtual relationship type and does not exist in the source schema. The following steps rewrite the condition constraints to involve the actual relations.

Step 1. If O participates in the original relationship type R_{ip} ($1 \le p \le j$), then $O.key = Ri.O.key$ in the condition constraints is rewritten into $O.key = R_{ip}.O.key$.

Step 2. If O_q ($1 \le q \le i$) participates in the original relationship type R_{ip} ($1 \le p \le j$), then $O_q.key = R_i.O_q.key$ in the condition constraints is rewritten into $O_q.key = R_{ip}.O_q.key$.

Step 3. If O, $O_1, O_2, \ldots, O_i$ participate in the original relationship types R_{i1}, $R_{i2}, \ldots, R_{im}$ ($1 \le m \le j$) respectively, then we append the following condition constraints in the rewritten condition constraints: $R_{i1}.O_{c1}.key = R_{i2}.O_{c1}.key \text{ and } \ldots \text{ and } R_{im-1}.O_{cr}.key = R_{im}.O_{cr}.key$, where $O_{c1}, \ldots, O_{cr}$ are the common object classes of $R_{i1}, \ldots, R_{im}$, based on which they are joined.

Step 1 rewrites the condition constraint involving object class O. Step 2 rewrites the condition constraints involving all DOCs of O (O_1, …, O_i) in R_i in the view. Step 3 constructs new condition constraints involving the common object classes of R_{i1}, R_{i2}, …, R_{im} ($1 \leq m \leq j$) and append them to the rewritten one, which actually link the condition constraints rewritten in Steps 1 and 2. In this way, the rewritten condition constraints still express the influence of DOCs (O_1, …, O_i) on O through R_i.

Next, we need to process the relationship attributes in the view in Rule Gen 2. There are two cases in which an attribute of a relationship type can be in an ORA-SS view.

Rule Gen 2: If an attribute A belongs to a relationship type R and is located below object class O in the ORA-SS view, then

Case 1 If R is an original relationship type in source schema and involves the DOCs of O (say {O_1, …, O_i}) in the view, then we generate an *xmlelement* function for the attribute A as a sub-element of O within the Select clause of the query block of O: *xmlelement("A", R.A)*

In this case, we generate the attribute A as a sub element of O instead of an attribute of O since A is a relationship attribute.

Case 2 If R is a derived relationship type generated by projecting a original relationship type R' in source schema, and R involves DOCs of O in the view: O_1, O_2, …, O_j and O_{j+1}, O_{j+2}, …, O_p are the dropped object classes from R', and the original attribute for A is A' in the source, then we generate a subquery for the attribute A as a sub-element of O within the Select clause of the query block of O:

Select xmlelement(agg(A'))
From R'
Where O_1.key=R'.O_1.key and … and O_j.key=R'.O_j.key
Group by O_{j+1}, O_{j+2}, …, O_p

The attribute A must be an aggregate attribute by applying some aggregate function such as sum, avg, or max/min to the original attribute A'. Thus, we have to use a sub query to express the correct occurrences of A in the XML view.

We have developed an algorithm based on Rule Gen 1 & 2 to automatically generate the SQLX query definition for an ORA-SS view schema. The algorithm takes as input an ORA-SS view, the underlying ORA-SS source schema, and the corresponding storage schema in the object-relational database. The output is the SQLX query definition for the view.

3 Conclusion

SilkRoute [7] adopts a declarative language RXL to define XML views over relational data and the other language XML-QL to query views. XPERANTO [2] uses a canonical mapping to create a default XML view from relational data. Then it utilizes XQuery to define other views based on the default view. Xyleme [5] defines XML views on XML source data by connecting one abstract DTD to a large collection of concrete DTDs. XML views are also supported as a middleware in integration systems, such as MIX [1], YAT [4] and Agora [8] in order to fulfill the potential of XML.

Our work differs from the related work in the following aspects. First, we take into consideration semantic information when designing XML views, which are thus guaranteed to be valid. Second, we adopt a user-friendly approach to retrieve XML data via views by automatically generating SQLX query expressions for the ORA-SS views.

The proposed approach not only materializes XML views, but also maps queries issued on the XML views into the equivalent queries in SQLX syntax on the underlying database. To the best of our knowledge, this is the first work to employ a semantic data model to design and query XML views.

References

1. C. Baru, A. Gupta, B. Ludaescher, et. al. XML-Based Information Mediation with MIX, ACM SIGMOD (Demo), 1999.
2. M. Carey, J. Kiernan, J. Shanmugasundaram, et. al. XPERANTO: A Middleware for Publishing Object-Relational Data as XML Documents, VLDB, 2000.
3. Y.B. Chen, T.W. Ling, M.L. Lee. Designing Valid XML Views, ER Conference, 2002.
4. V. Christophides, S. Cluet, J. Simeon. On Wrapping Query Languages and Efficient XML Integration, ACM SIGMOD, 2000.
5. S. Cluet, P. Veltri, D. Vodislav. Views in a Large Scale XML Repository, VLDB, 2001.
6. G. Dobbie, X.Y Wu, T.W Ling, M.L Lee. ORA-SS: An Object-Relationship-Attribute Model for SemiStructured Data, Technical Report TR21/00, School of Computing, National University of Singapore, 2000.
7. M. Fernandez, W. Tan, D. Suciu. SilkRoute: Trading Between Relations and XML, World Wide Web Conference, 1999.
8. I. Manolescu, D. Florescu, D. Kossmann. Answering XML Queries over Heterogeneous Data Sources, VLDB, 2001.
9. Y.Y.Mo, T.W.Ling. Storing and Maintaining Semistructured Data Efficiently in an Object-Relational Database, WISE Conference, 2002.
10. SQLX. http://www.sqlx.org