Chapter 2.8
Semantic Integrity Constraint Checking for Multiple XML Databases

Praveen Madiraju
Marquette University, USA

Rajshekhar Sunderraman
Georgia State University, USA

Shamkant B. Navathe
Georgia Institute of Technology, USA

Haibin Wang
Emory University, USA

ABSTRACT

Global semantic integrity constraints ensure integrity and consistency of data spanning multiple databases. In this paper, we take initial steps towards representing global semantic integrity constraints for XML databases. We also provide a general framework for checking global semantic integrity constraints for XML databases. Furthermore, we set forth an efficient algorithm for checking global semantic integrity constraints across multiple XML databases. Our algorithm is efficient for three reasons: (1) the algorithm does not require the update statement to be executed before the constraint check is carried out; hence, we avoid any potential problems associated with rollbacks, (2) sub constraint checks are executed in parallel, and (3) most of the processing of algorithm could happen at compile time; hence, we save time spent at run-time. As a proof of concept, we present a prototype of the system implementing the ideas discussed in this paper.

INTRODUCTION

Consider a scenario wherein two or three different companies host XML data (native XML database management system) at different and independent sites. Data at these sites are not necessarily independent, but may participate in a relationship with data from other sites. A single update (XUpdate [Tatarinov, Ives, Halevy, & Daniel, 2001; Laux & Martin, 2000]) on one site might cause a global constraint (*global XConstraint*) to be violated. By global XConstraints, we mean global semantic integrity constraints affecting multiple XML databases. Hence we need an approach to check for such constraint violations. In the XML database setting, the majority of the times, users are interested in generating (updating), integrating and exchanging data. So, frequent updates on XML data may cause frequent global constraint violations. *Hence we need a plan that will efficiently and speedily check for such global constraint violations.*

Plan *A* would be to translate the XML document into relational data using methods such as those found in Shanmugasundaram, Tufte, He, Zhang, DeWitt, and Naughton (1999), Chen, Davidson, Hara, and Zheng (2003), and Fong and Wong (2004). And then, map the updates and constraints on the XML data to corresponding updates and constraints on the relational data (Chen, Davidson, & Zheng, 2002a). Now the problem of constraint checking on XML data is pushed to the problem of constraint checking on relational data. There are well established models for constraint checking in the relational world. However, this approach suffers from the overhead cost involved in transforming XML data into relational data (Kane, Su, & Rundensteiner, 2002). Plan *B* would be to check for constraint violations on the XML data without transforming to relational data. It should be noted that using plan *A* versus plan *B* depends on the application being considered. If the application contains millions of records and if it benefits to use relational database features such as querying, fast indexing, and so forth, it is worthwhile to consider plan A; otherwise, plan B suffices for a normal-sized application. In this paper, we consider the plan B route.

A brute force approach would first update an XML document and then check for constraint violations. If a constraint is violated, we can rollback. However, such a brute force approach suffers from the overhead of time and resources spent on rollback. Hence, we need an approach that would check for constraint violations before updating the database and therefore obviates the need for rollback situations.

In our constraint checking procedure, constraint violations are checked at compile time, *before* updating the database. Our approach centers on the design of the *XConstraint Checker*. Given an XUpdate (Tatarinov et al., 2001; Laux & Martin, 2000) statement and a list of global XConstraints, we generate *sub XConstraint* checks corresponding to local sites. Sub XConstraint is an XML constraint, expressed as an XQuery, local to a single site (more details in the fourth section). The results gathered from these sub XConstraints determine if the XUpdate statement violates any global XConstraints. Our approach is *efficient*; since we do not require the update statement to be executed before the constraint check is carried out and hence, we avoid any rollback situations. Our approach achieves *speed* as the sub constraint checks can be executed in parallel.

Overview of the System

Figure 1 gives the overview of the system. We propose a three-tier architecture. The server side consists of two or more sites hosting native XML databases. In Figure 1, we show three sites — S_1, S_2, and S_3. The client makes an XUpdate request through the middleware. The middleware consists of the *XConstraint Checker* and the XML/DBC (Gardarin, Mensch, Tuyet, & Smit, 2002) API. In previous work (Madiraju, Sunderraman, & Navathe, 2004), we have introduced our notations