Updating Recursive XML Views of Relations

Byron Choi1, 4 (蔡冠球), Gao Cong2, 4 (丛 高), Wenfei Fan3, 4 (樊文飞), and Stratis D. Viglas4

1 Division of Information System, School of Computer Engineering, Nanyang Technological University, 639798, Singapore
2 Microsoft Research Asia, Beijing 100080, China
3 Bell Laboratories, Murray Hill, NJ 07974-0636, U.S.A.
4 University of Edinburgh, Edinburgh EH8 9LE, Scotland, U.K.

E-mail: kkcchoi@ntu.edu.sg; gaocong@microsoft.com; {wenfei, sviglas}@inf.ed.ac.uk

Received September 3, 2007; revised March 7, 2008.

Abstract This paper investigates the view update problem for XML views published from relational data. We consider XML views defined in terms of mappings directed by possibly recursive DTDs compressed into DAGs and stored in relations. We provide new techniques to efficiently support XML view updates specified in terms of XPath expressions with recursion and complex filters. The interaction between XPath recursion and DAG compression of XML views makes the analysis of the XML view update problem rather intriguing. Furthermore, many issues are still open even for relational view updates, and need to be explored. In response to these, on the XML side, we revise the notion of side effects and update semantics based on the semantics of XML views, and present efficient algorithms to translate XML updates to relational view updates. On the relational side, we propose a mild condition on SPJ views, and show that under this condition the analysis of deletions on relational views becomes PTIME while the insertion analysis is NP-complete. We develop an efficient algorithm to process relational view deletions, and a heuristic algorithm to handle view insertions. Finally, we present an experimental study to verify the effectiveness of our techniques.

Keywords XML, XML publishing, XML views, view update

1 Introduction

As a classical technical problem, view updates have been studied for relational databases for decades (see, e.g., [1–4]), and the techniques developed in that area have been introduced into commercial DBMSs [5–7]. Recently, a number of systems have been developed for publishing relational data to XML [5–10]. The published XML documents can be seen as XML views of the relational data. For all the reasons that updating data through its relational views is needed, it is also important to update relational databases through their XML views.

In this paper we study the XML view update problem, which can be stated as follows. Given an XML view of a relational database, we want to propagate updates of the XML view to the relational tables, without compromising the integrity of neither the XML nor the relational data. Formally put, given an XML view defined as a mapping \( \sigma : R \rightarrow D \) from relations of a schema \( R \) to XML documents (trees) of a DTD \( D \), a relational instance \( I \) of \( R \), the XML view \( T = \sigma(I) \), and updates \( \Delta_X \) on the XML view \( T \), we want to compute relational updates \( \Delta_R \) such that \( \Delta_X(T) = \sigma(\Delta_R(I)) \). That is, the relational updates \( \Delta_R \), when propagated to XML via the mapping \( \sigma \), yield the desired XML updates \( \Delta_X \) on the view \( T \).

While several commercial systems [5–7] allow users to define XML views of relations, their support for XML view updates is either very restricted or not yet available. Previous work on XML view updates [11] has addressed the problem by translating XML view updates to relational view updates and delegating the problem to the relational DBMS; however, most commercial DBMSs only have limited view-update capability [5–7]. The state of the art in XML view update research [12, 13] solves the problem by explicitly focusing on non-recursively defined XML views and XML updates defined without recursive XPath queries. Though it is a complete solution, the restrictions posed in [13] are unfortunate since the recent proposals on XML update languages [14, 15] employ recursive XPath queries while DTDs (and thus XML view definitions) found in practice are often recursive [16].

Regular Paper

*Wenfei Fan is supported in part by EPSRC under Grants No. GR/S63205/01, No. GR/T27433/01, and No. EP/E029213/1.
In accordance with these requirements, we consider more general XML views and updates: possibly recursive XML view definitions and XML updates specified in terms of XPath expressions with recursion (descendant-or-self “//”) and complex filters, as illustrated by an example below.

Example 1. Consider a registrar database $I_0$, which is specified by the relational schema $R_0$ (with keys underlined):

\[
\begin{align*}
\text{course} & (\text{cno}, \text{title}, \text{dept}) \\
\text{project} & (\text{cno}, \text{title}, \text{dept}) \\
\text{student} & (\text{ssn}, \text{name}) \\
\text{enroll} & (\text{ssn}, \text{cno}) \\
\text{prereq} & (\text{cno}_1, \text{cno}_2),
\end{align*}
\]

where a tuple $(c_1, c_2)$ in prereq indicates that $c_2$ is a prerequisite of $c_1$.

As depicted in Fig.1 (the dotted lines will be explained shortly), an XML view $T_0$ of the relational database is published for the CS department by extracting CS course-registration data from $I_0$. The view is required to conform to the DTD $D_0$ below (the definition of elements whose type is PCDATA is omitted):

\[
\begin{align*}
\langle \! \langle \text{ELEMENT} \ 	ext{db} \ \langle \! \langle \text{course}' \rangle \! \rangle \rangle \\
\langle \! \langle \text{ELEMENT} \ 	ext{course} \ \langle \! \langle \text{cno}, \text{title}, \text{prereq}, \text{takenBy} \rangle \! \rangle \\
\langle \! \langle \text{ELEMENT} \ 	ext{prereq} \ \langle \! \langle \text{cno}_1, \text{cno}_2 \rangle \! \rangle \\
\langle \! \langle \text{ELEMENT} \ 	ext{takenBy} \ \langle \! \langle \text{student} \rangle \! \rangle \\
\langle \! \langle \text{ELEMENT} \ 	ext{student} \ \langle \! \langle \text{ssn}, \text{name} \rangle \! \rangle
\end{align*}
\]

Note that the view is defined recursively since the DTD $D_0$ is recursive (course is defined indirectly in terms of itself via prereq). Now consider an XML update $\Delta_X = \text{insert } T'$ into $P_0$ posed on the XML view $T_0$, where $P_0$ is the (recursive) XPath query $\text{course}[\text{cno}=\text{CS650}]/\text{course}[\text{cno}=\text{CS320}]/\text{prereq}$, and $T'$ is the subtree representing the course CS240. It is to find all the CS320 nodes below CS650 in $T_0$ and for each CS320 node $v$, insert $T'$ as a prerequisite of $v$. To carry out $\Delta_X$, we need to find updates $\Delta_R$ on the underlying database $I_0$ such that $\Delta_X(T_0) = \sigma_0(\Delta_R(I_0))$.

Already a hard problem for relational views, the view update problem for XML views introduces several new challenges, which previous work [11–13] on XML view updates cannot handle.

First, update semantics should be revised in the context of XML views of relations. Referring to the example above, the operation asks for inserting CS240 as a prereq of only those CS320 nodes below CS650; however, CS320 nodes also occur elsewhere below the root. As the XML view is published from the same relational database, all the courses, and therefore CS320, have unique prereq hierarchies. Such an insertion on selected paths of the hierarchy will result in side effects that need to be detected. In such a case, the users need to be consulted and, if they insist on carrying on updating, the semantics of insertion is revised such that the insertion will be performed at every CS320 node. Thus the insertion can accommodate side effects while being consistent with the semantics of the XML view. Note that such side effects are orthogonal to both the publishing middleware used and the storage scheme of the XML views. The details of side effects on deletions are even more subtle and call for a new semantics (see Section 2).

Second, the XML view may be compressed by storing each subtree shared by multiple nodes in the tree only once, as indicated in Fig.1 (replacing the subtrees in the dotted triangles by dotted edges). The need for this is evident: the compressed view becomes a directed acyclic graph (DAG), which is often significantly (at times even exponentially) smaller than the original tree. Furthermore, one may want to store the view (DAG) in relations itself. This raises another question: how