Algebra-Based Identification of Tree Patterns in XQuery

Andrei Arion[1,2], Véronique Benzaken[2], Ioana Manolescu[1],
Yannis Papakonstantinou[3], and Ravi Vijay[1,4]

[1] INRIA Futurs, Gemo group, France
firstname.lastname@inria.fr
[2] LRI, Univ. Paris 11, France
veronique.benzaken@lri.fr
[3] CSE Dept., UCSD, USA
yannis@cs.ucsd.edu
[4] IIT Bombay, India
ravivj@cse.iitb.ac.in

Abstract. Query processing performance in XML databases can be greatly enhanced by the usage of materialized views whose content has been stored in the database. This requires a method for identifying query subexpressions matching the views, a process known as view-based query rewriting. This process is quite complex for relational databases, and all the more daunting on XML databases.

Current XML materialized view proposals are based on tree patterns, since query navigation is conceptually close to such patterns. However, the existing algorithms for extracting tree patterns from XQuery do not detect patterns *across nested query blocks*. Thus, complex, useful tree pattern views may be missed by the rewriting algorithm. We present a novel tree pattern extraction algorithm from XQuery queries, able to identify larger patterns than previous methods. Our algorithm has been implemented in an XML database prototype [5].

1 Introduction

The XQuery language [23] is currently gaining adoption as the standard query language for XML. One performance-enhancing technique in XQuery processing is the usage of materialized views. The idea is to pre-compute and store in the database the result of some queries (commonly called *view definitions*), and when a user query arrives, to identify which parts of the query match one of the pre-computed views. The larger parts of the query one can match with a view, the more efficient query processing will be, since a bigger part of the query computation can be obtained directly from the materialized view.

Identifying useful views for a query requires reasoning about containment (e.g., is all the data in view v contained in the result of query q ?) and equivalence (e.g., is the join of views v_1 and v_2 equivalent to the query q ?). XML query ontainment and equivalence are well understood when views and queries are represented as *tree patterns*, containing tuples of elements satisfying specific

H. Legind Larsen et al. (Eds.): FQAS 2006, LNAI 4027, pp. 13–25, 2006.

structural relationships [18, 19]. Moreover, popular XML indexing and fragmentation strategies also materialize tree patterns [10, 11, 13, 14]. Therefore, tree patterns are an interesting model for XML materialized views [3, 5, 6, 11, 12].

Our work is placed in the context of XQuery processing based on a persistent store. We make some simple assumptions on this context, briefly presented next.

Most persistent XML stores assign some *persistent identifiers* to XML elements. Such identifiers are often *structural*, that is, by comparing the identifiers id_1 and id_2 of two elements e_1 and e_2, one can decide whether some structural relationship exists between e_1 and e_2: for instance, whether e_1 is a child, parent, or sibling of e_2. The interest of structural identifiers is that establishing such relationships directly is much more efficient than navigating from e_1 to e_2 in the database to verify it. Numerous structural ID proposals have been made so far, see e.g. [2, 20]. *We assume persistent IDs are available in the store.* The IDs may, but do not need to, have structural properties.

Our second assumption is that a materialized view may store: (*i*) *node IDs* [10, 12, 14], (*ii*) node *values* (i.e., the text nodes directly under an element, or the value of an attribute) [10], and/or (*iii*) node *content*, that is, the full subtree rooted at an XML element (or a pointer to that subtree) [6]. This assumption provides for flexible view granularity.

To take advantage of tree pattern-shaped materialized views, one has to understand which views can be used for a query q. This process can be seen as a translating q to some *query patterns* $p_{q1}, \ldots, p_{qn}$, followed by a rewriting of every query pattern p_{qi} using the view patterns $p_{v1}, \ldots, p_{vm}$. The first step (query-to-pattern translation) is crucial. Intuitively, the bigger the query patterns, the bigger the view(s) that can be used to rewrite them, thus the less computations remain to be applied on top of the views.

The contribution of this paper is a provably correct algorithm identifying tree patterns in queries expressed in a large XQuery subset. The advantage of this method over existing ones [6, 9, 21] is that the patterns we identify are strictly larger than in previous works, and in particular may span over nested XQuery blocks, which was not the case in previous approaches. We ground our algorithm on an algebra, since (as we will show) the translation is quite complex due to XQuery complexity, and straightforward translation methods may loose the subtle semantic relationships between a pattern and a query.

Materialized views: advantages and drawbacks. A legitimate question is whether the cost of materializing and maintaining materialized views is justified by the advantages they provide ? It turns out that in XML persistent store, a tree-based approach is rarely (if ever !) sufficient to support complex querying. We survey XML storage and indexing strategies in [15]. Shredding schemes (aiming at loading XML documents in a set of relational tables) also offer an example of materialized XML views, recognized as such in [11, 16]. XML view maintenance in the presence of updates is a direction we are currently working on.

The paper is organized as follows. Section 2 motivates the need for pattern recognition in XQuery queries. Section 3 sets the formal background for the translation algorithm presented in Section 4.