

Incremental Maintenance of Materialized XQuery Views

Maged El-Sayed, Elke A. Rundensteiner, and Murali Mani
Department of Computer Science
Worcester Polytechnic Institute Worcester, MA 01609
(maged | rundenst | mmani)@cs.wpi.edu

1 Introduction

Materializing the contents of views has important applications including providing fast access to derived database repositories, optimizing query processing based on cached results, and increasing availability. Maintaining the consistency between materialized views and their base data in the presence of source updates is important to ensure that the materialized views are up-to-date. The straightforward solution for this problem is to recompute the view from scratch over the updated sources. This is not practical considering that source updates are typically small relative to the original source size and that this process is expensive and might take a long time (hours or even days for large data sources). To address this problem, incremental maintenance has been advocated as a cheaper solution over full recomputation [6]. Such view maintenance work has been largely done in the context of traditional databases [2, 6, 9, 10]. However, this no longer fulfills current needs with XML emerging as an important medium for representing and exchanging data over the Internet. Modern data sources, including structured and semi-structured sources, often export XML views, typically defined using XQuery over base data.

Incremental maintenance of XQuery views poses unique challenges compared to the incremental maintenance of relational or even object-oriented views as we illustrate in Section 2. Little work has been done on the incremental maintenance of XML and semi-structured views. Early solutions for maintaining semi-structured views [1,12] consider only a small class of views. Further, they require materialization of large auxiliary data structures. [8] propose a solution for maintaining views defined using a restricted subset of the XML-QL language. It does not support explicit union operations and complex nested queries and places restrictions on updating some source values. In all the above work, XML order was not supported. [3] proposes a solution that supports order-sensitive view maintenance for XQuery views based on special propagation rules that require special purpose processing for propagating updates. This does not follow the standard framework where propagation of updates is done using the regular query processing engine, like in [7]. [3] also requires materialization of intermediate data structures.

2 Challenges in Maintaining XQuery Views

Motivating Example. Consider the two XML documents shown in Fig. 1. Assume that the order among nodes in each of these sources represents a desired source document order. Consider the XQuery view shown in Fig. 2(a) defined over these source documents. The view extent resulting from executing the XQuery view is shown in Fig. 2(b). Now assume that the source XML document receives updates, shown in Fig. 3, defined using the XQuery update language [11]. The view maintenance solution must propagate those updates to refresh the materialized view. We highlight the following challenges:

- *Modeling and Validating Source XML Updates.* In the relational context source updates are simply represented as flat tuples that conform to the same schema as that of the source to be updated. Each tuple is self-contained in that it includes all necessary information for propagation. In the XML context an update might be provided in different shapes like an entire XML fragment as in Fig. 3(a), a root to an XML fragment as in Fig. 3(b), a leaf node value as in Fig. 3(c). We need to decide on how to (i) model XML source updates, (ii) handle different types of XML updates, (iii) batch source XML update, (iv) check for update relevancy, and (v) handle XML source updates with missing information.

```
<bib>
<book year = "1994">
<title>TCP/IP Illustrated</title>
<author>
<last>Stevens</last><first>W.</first>
</author>
</book>
<book year = "2000">
<title>Data on the Web</title>
<author>
<last>Abiteboul</last>
<first>Serge</first>
</author>
</book>
</bib>
bib.xml
```

```
<prices>
<entry>
<price>39.95</price>
<b-title>Data on the Web</b-title>
</entry>
<entry>
<price> 65.95</price>
<b-title>TCP/IP Illustrated</b-title>
</entry>
<entry>
<price> 69.99</price>
<b-title>Advanced Programming in
the Unix environment </b-title>
</entry>
</prices>
prices.xml
```

Figure 1. Two input XML documents.

- *Propagating Source XML Updates.* The query shown in Fig. 2(a) is a relatively simple XQuery expression, yet it contains many operations (navigation, unnesting, join, grouping, distinct, result construction, and a nested subquery). We need a propagation solution that is general

enough to support an expressive subset of the XQuery language. The propagation of XML updates is complex due to the hierarchical semi-structured nature of data with order semantics in contrast to the flat tuples with no order semantics processed by relational views. In addition XQuery is order sensitive, hence even relational like operations performed by XQuery are also order sensitive.

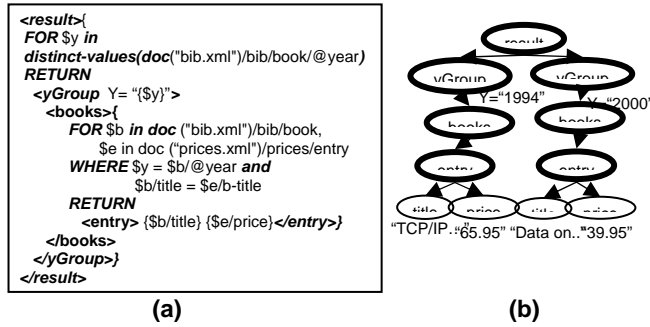


Figure 2. (a) XQuery expression on the two XML documents in Fig. 1 and (b) resulting XML view extent.

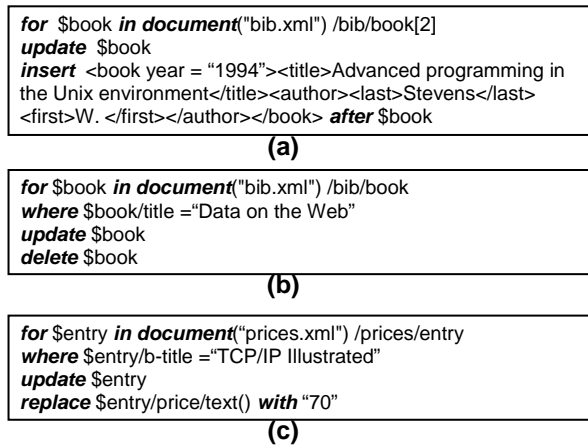


Figure 3. Three Source XML updates.

- Applying Propagated Updates to Materialized Views. Consider that we wish to refresh the view extent Fig. 2(b) on the source update in Fig. 3(a) that inserts a new “book” to “bib.xml”. The propagation of this source update should create a delta update that inserts a new “entry” element into the view extent. This “entry” element should be inserted into a particular location (as a child of the “books” element that has a parent “yGroup” element with attribute “Y” equal to “1994”) and in a particular order (after the existing “entry” element), based on the query order semantics. These issues are not encountered in the relational context since applying propagated updates to relational materialized views simply involves inserting or

deleting tuples from the materialized flat tuple set (a table) where order is not an issue.

3 Our Approach

We now propose a comprehensive framework for solving the problem of maintaining XML views defined in XQuery. Our solution supports an expressive subset of XQuery views including XPath expressions, FLWOR expressions, and Element Constructors. We support XML order and avoid intermediate result materialization. We take an algebraic approach for propagating updates. Our proposed approach generates incremental maintenance plans in the same language used to define the view, just like the hallmark of view maintenance strategies used in commercial database systems, like DB2 [7]. This makes implementing and integrating the view maintenance solution with the current XQuery processing engines an easy task. We also provide support for bulk update processing of possibly different update operation types.

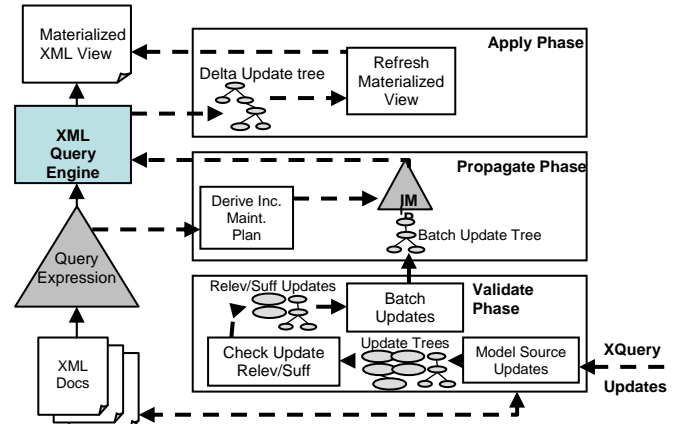


Figure 4. Our view maintenance framework.

As shown in Fig. 4, our solution is composed of three main phases.

- The Validate Phase.** In this stage XQuery source updates [11] are transformed into a set of well defined update primitives, called *update trees*. The relevancy of each update tree to the view is verified. Irrelevant updates are discarded. Updates that are potentially relevant to the query are annotated with needed information for propagation. This includes any other nodes needed by the query and appropriate counts. Lastly, update trees of even different update types are batched into one tree structure called *batch update tree*.
- The Propagate phase.** The most important task in this phase is to derive *Incremental Maintenance Plans (IMPs)* from the view query. Batch update trees are

processed using the Incremental Maintenance Plans to generate propagated updates, called *delta update trees*. Delta update trees are to be used in the next phase to incrementally refresh the view extent. IMPs are expressed in the same algebraic language used in computing the materialized view extents.

3. The *Apply Phase*. In this final phase, delta update trees that had been computed in the Propagate Phase are applied to the materialized view to refresh it. This involves merging nodes in the delta update tree with nodes in the materialized view and performing any necessary insertion, deletion, or modification to the materialized view.

One major advantage of our solution is that incremental maintenance plans used to compute delta updates to view extents are in the same algebraic language used in computing the materialized view extents. This is similar to what commercial relational database systems do [7]. Hence, our solution can be easily integrated with any XQuery query engine. See [5] for details on how to derive incremental maintenance plans.

Distributive Property of Views. Our solution relies on the distributive property of views. The distributive property of a view in the relational context is typically defined over the union operator (\cup) [2]. For example a select view defined over a source R is distributive because for a source update ΔR the equation $\sigma_p(R \cup \Delta R) = \sigma_p(R) \cup \sigma_p(\Delta R)$ holds. That is, by processing the view over the update and combining the result $\sigma_p(\Delta R)$ with the initial materialized view $\sigma_p(R)$ we get the same final result that we would get if we fully recompute the view over the updated source $\sigma_p(R \cup \Delta R)$. XML views are not typically considered distributive. Operations like nesting, grouping, and element construction may require information about previously processed data in order to be able to process updates. Another difficulty associated with XQuery views in particular is the issue of order. Operations that are typically considered distributive in a non-order aware processing environment are no longer distributive when we consider XQuery views. We solve these issues through the use of a special semantic identifier solution [4]. Our semantic id solution (1) enables distributive XML query execution even for queries involving operators like group by, nesting, distinct, and sorting, typically known to be non-distributive and (2) enables operators to process data without having to worry about the order of that data, since the order is encoded as part of the semantic ids. Our solution also removes the need to perform intermediate sorting of data, even for operators that enforce new order like *Order By*.

4 Conclusions

In this paper we have proposed a framework for maintaining a large class of XQuery views. Our solution relies on special semantic identifier solution and on an algebraic update propagation strategy to provide an efficient propagation of source updates. Our solution generates incremental maintenance plans that can be processed using regular query processing techniques to generate delta updates used to refresh materialized view extents.

References

- [1] S. Abiteboul and et al. Incremental Maintenance for Materialized Views over Semistructured Data. In *VLDB*, pages 38–49, 1998.
- [2] J. A. Blakeley, P. Larson, and F.W. Tompa. Efficiently Updating Materialized Views. In *SIGMOD*, pages 61–71, 1986.
- [3] K. Dimitrova, M. El-Sayed, and E. A. Rundensteiner. Order sensitive view maintenance of materialized xquery views. In *ER*, pages 144–157, Oct. 2003.
- [4] M. El-Sayed, E. A. Rundensteiner, and M. Mani. Incremental Fusion of XML Fragments through Semantic Identifiers. In *IDEAS*, 2005.
- [5] M. El-Sayed, E. A. Rundensteiner, and M. Mani. Incremental Maintenance of Materialized XQuery Views. Technical Report WPI- CS-TR-05-12, Worcester Polytechnic Institute, June 2005.
- [6] A. Gupta and I. S. Mumick. Maintenance of Materialized Views: Problems, Techniques, and Applications. In *IEEE Bulletin of the Technical Committee on Data Engineering*, 18(2), pages 3–18, June 1995.
- [7] W. Lehner, R. Sidle, H. Pirahesh, and R. W. Cochrane. Maintenance of cube automatic summary tables. In *SIGMOD*, pages 512–513, 2000.
- [8] H. Liefke and S. B. Davidson. View maintenance for hierarchical semistructured data. In *DWKD*, pages 114–125, 2000.
- [9] J. Liu, M. W. Vincent, and M. K. Mohania. Maintaining views in object-relational databases. In *CIKM*, pages 102–109, 2000.
- [10] M. K. Mohania and Y. Kambayashi. Making aggregate views self-maintainable. *DKE*, 32(1):87–109, 2000.
- [11] I. Tatarinov, Z. G. Ives, A. Y. Halevy, and D. S. Weld. UpdatingXML. In *SIGMOD*, pages 413–424, May 2001.
- [12] Y. Zhuge and H. Garcia-Molina. Graph Structured Views and Their Incremental Maintenance. In *ICDE*, pages 116–125, February 1998.