Triggers over XML Views of Relational Data

Feng Shao Antal Novak* Jayavel Shanmugasundaram
Cornell University
{fshao, afn, jai}@cs.cornell.edu

1. Introduction

XML has emerged as a dominant standard for information exchange on the Internet. However, a large fraction of data continues to be stored in relational databases. Consequently, there has been a lot of interest in publishing relational data as XML. While there exist many systems that support XML views of relational databases [4, 5], these systems are *passive* in the sense that they can only support user-initiated queries over the views. By contrast, in this paper we propose an *active* system, which allows users to create triggers on XML views.

At a high level, there are two approaches to supporting triggers over XML views. The first is to materialize the entire view and store it in an XML database with support for XML triggers. However, this approach suffers from the overhead of replicating and incrementally maintaining the materialized XML on every relational update affecting the view, even though users may only be interested in relatively rare events. Another practical downside is that this approach requires a full-function XML DBMS supporting incremental updates and triggers, even though the underlying relational database supports all of this functionality. Therefore, in this paper, we propose the alternative approach of translating XML triggers into *SQL triggers*. The primary benefits of this approach are that it fully leverages sophisticated relational technology, does not require an XML database, and avoids having to materialize the XML view. There are some challenges involved in this approach, however, because triggers can be specified over complex XML views with nested predicates, while SQL triggers can only be specified over flat tables. Consequently, even *identifying* the parts of an XML view that could have changed due to a (possibly deeply nested) SQL update is a non-trivial task, as is the problem of *computing* the old and new values of an updated fragment of the view.

In this paper, we address the above challenges and propose a system architecture and an algorithm for supporting triggers over XML views of relational data. We implement and evaluate our system; the performance results indicate our techniques are a feasible approach to supporting triggers over XML views of relational data.

```
<db>
  <product>
    <row><pid>P1</pid><name>CRT 15</name></row>
    ...
  </product>
  <vendor>
    <row><vid>Amazon</vid><pid>P1</pid></row>
    ...
  </vendor>
</db>
```

Figure 1. Default view

```
<catalog>
  <product name="CRT 15">
    <vendor>
      <vid>Amazon</vid><price>100.00</price>
    </vendor>
    <vendor>
      <vid>Bestbuy</vid><price>120.00</price>
    </vendor>
    ...
  </product>
  <product name="LCD 19">
  ...
</catalog>
```

Figure 2. *MultiVendorProducts* view.

2. System Overview

We have developed our trigger processing techniques in the context of the Quark system, which is similar to XPERANTO [5] in its support for querying XML views. At a high level, XPERANTO allows users to define application-specific views over an automatically created *default view* of relational data; for example, a user can create the *MultiVendorProducts* view based on the *default view*, materialized in Figure 1 and Figure 2 respectively, to obtain products which are sold by *at least two* vendors. Note that although our techniques are implemented in Quark, they are applicable to any XML publishing system.

We use XQGM (the XML Query Graph Model) [5] to represent XQuery queries and views. XQGM consists of a set of operators and functions, and directed edges which connect the operators. Each operator produces zero or more output tuples (computed from its inputs), whose column values are XML nodes/values. In the case of views over re-

* Currently at Stanford University.

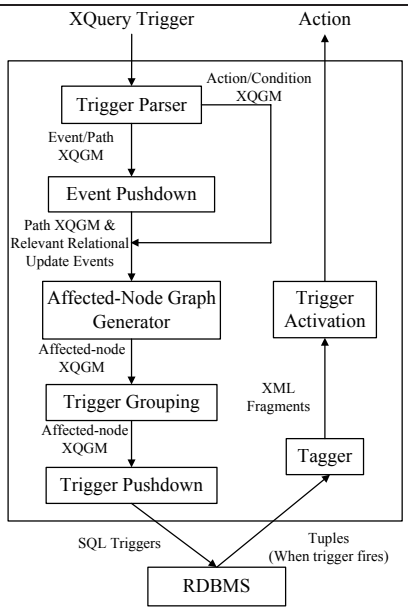

Figure 3. System architecture.

lational data, the default view is produced by a single *Table* operator, while a user-defined view has a more complex XQGM graph.

2.1. XML Trigger Specification Language

We use a subset of the trigger specification language proposed by Bonifati et al. [1], whose syntax is shown below:

```
CREATE TRIGGER Name AFTER Event
ON Path WHERE Condition DO Action
```

Briefly, each trigger has a unique *Name*; an *Event* type (UPDATE, INSERT, or DELETE); a *Path* (an XPath expression identifying a fragment of a view); a boolean *Condition* (which is an XQuery expression); and an *Action*, whose semantics are implementation-defined: in our system, it is a call to an external function, f_{action}, whose parameters are arbitrary XQuery expressions. Finally, two variables, `OLD_NODE` and `NEW_NODE`, are bound to the value of the node specified by *Path* before and after the *Event*; they may be referenced in the *Condition* and in the parameters to f_{action}. (When the *Event* is INSERT or DELETE, only the `NEW_NODE` or `OLD_NODE`, respectively, can be used.)

2.2. Architecture

Our system architecture is shown in Figure 3. When a user creates a trigger on a view, the Trigger Parser converts the *Path*, *Condition* and *Action* into their respective XQGM graphs. The trigger *Event* and the *Path* graph are then analyzed by the Event Pushdown module to determine the minimal set of base relations on which inserts, updates, or deletes could cause the trigger to be fired. This is done by adapting existing techniques for view and constraint maintenance [2, 3].

For each of these tables, the Affected-Node Graph Generator constructs a new XQGM graph which, when evaluated, will produce the parameters to pass to f_{action}. This algorithm, discussed in detail in [6], takes as input the *Path* XQGM graph, G, and recursively builds up a new graph, G_{key}, which identifies the *keys* of the XML nodes affected by the relational update. It then joins G_{key} with G on the key, to produce the entire affected node. The result is an XQGM graph which evaluates to a 2-tuple, (`OLD_NODE`, `NEW_NODE`), for each affected XML node. Next, it grafts on the graphs corresponding to the *Condition* (to filter out uninteresting updates) and *Action* (to compute f_{action}'s parameters from `OLD_NODE` and `NEW_NODE`). Finally, XQGM rewrite rules are applied to minimize unnecessary computation. The result is an XQGM graph, G_{params}, which, when evaluated for a given relational update, will produce one tuple for each call to f_{action}.

G_{params} is then fed into the Trigger Grouping module, which groups similar triggers together for improved scalability.

Finally, the Trigger Pushdown module takes this XQGM graph and, using the selection pushdown and tagger pull-up transformations described in [5], produces a set of SQL triggers, one for each relational event.

When activated, an SQL trigger issues a single SQL query to retrieve the relational data required for the actions of the XML triggers. The constant-space Tagger then converts these results to XML. Finally, the Trigger Activation module activates the appropriate XML triggers and passes in the XML results as parameters to their actions.

In our implementation, we support a powerful subset of XQuery. Specifically, we support arbitrarily complex nested views with FLWOR expressions, quantified expressions, XPath expressions with child/descendant axes, arithmetic operators, comparison operators, and element constructors.

3. Conclusion

We have presented a systematic way of translating triggers over XML views of relational data into SQL triggers, and for translating relational updates into their corresponding XML updates. We have implemented our techniques in the context of the Quark system built over IBM DB2. Our performance results [6] indicate that our proposed techniques are a scalable and feasible approach to supporting triggers over XML views of relational data.

References

[1] A. Bonifati, D. Braga, A. Campi, S. Ceri, "Active XQuery", ICDE 2002.

[2] S. Ceri, J. Widom, "Deriving Production Rules for Constraint Maintenance", VLDB 1990.

[3] S. Ceri, J. Widom, "Deriving Production Rules for Incremental View Maintenance", VLDB 1991.

[4] M. Fernandez, D. Suciu, W. Tan , "SilkRoute: Trading between Relations and XML", WWW 2000.

[5] J. Shanmugasundaram, J. Kiernan, E. Shekita, C. Fan, J. Funderburk. "Querying XML views of relational data", VLDB 2001

[6] F. Shao, A. Novak, J. Shanmugasundaram, "Triggers over XML Views of Relational Data", Cornell Technical Report available at http://www.cs.cornell.edu/database/quark/Triggers-full.pdf, 2004.