XML Publishing: Look at Siblings too!

Sihem Amer-Yahia
AT&T Labs–Research
sihem@research.att.com

Yannis Kotidis
AT&T Labs–Research
kotidis@research.att.com

Divesh Srivastava
AT&T Labs–Research
divesh@research.att.com

Abstract

In order to publish a nested XML document from flat relational data, multiple SQL queries are often needed. The efficiency of publishing relies on how fast these queries can be evaluated and their results shipped to the client. We illustrate novel optimization techniques that enable computation sharing between queries that construct sibling elements in the XML tree. Such queries typically share large common join expressions that can be exploited through appropriate rewritings. These rewritings are fundamental to XML publishing and provide considerable performance benefits without having to modify the relational engine.

1 Introduction

Publishing relational data in XML is a growing need for business applications where information is exchanged in XML while most of the legacy data is stored in relational back-ends. In addition, many users are now shredding XML documents in different ways to store them in relational systems. Once stored, retrieving this XML data is similar to publishing relational data.

Both applications need to construct an XML document "on the fly" and, thus, need do so as efficiently as possible. Due to the flat nature of relational data, as opposed to the nested structure of XML, generating an XML document from a relational store often involves evaluating multiple SQL queries (possibly as many as the number of elements in the DTD) containing join expressions and merging answers to queries corresponding to parent and children elements in order to reconstitute the XML tree structure. In addition, data has to be tagged to produce the final document. The cost of merging and tagging is negligible if the data is sorted in document order [13]. However, due to the presence of common join expressions in the SQL queries used to build an XML document, query performance can vary considerably between plans, necessitating a cost-based optimization of the plan for building an XML document.

In [6], the authors explore rewriting-based optimizations between the query for a parent element and the queries for its children elements, for this purpose. Intuitively, computation sharing that is possible between queries for sibling elements is guaranteed to be at least as much as, and often significantly more than, the computation sharing possible between a parent and its children elements. For this reason, in this paper, we argue that sharing computation between queries that construct sibling XML elements is the key to efficiently constructing XML documents from relational data, regardless of whether we are publishing legacy data or retrieving XML data that has been shredded into relations. We then exploit this observation to illustrate novel optimization techniques that enable computation sharing between queries for sibling elements in XML publishing.

The rest of this technical note is organized as follows. In Section 2, we illustrate using an example the technical challenges posed by the optimization of queries for constructing sibling elements in an XML document. We present related work in Section 3, and then summarize in Section 4.

2 Motivation: Publishing Legacy Data

Applications such as publishing legacy data in XML [4, 6, 12, 16, 17] and reconstructing XML documents that have been stored in relations [1, 3, 5, 7, 8, 10, 11] have gained a lot of interest recently. In both of these applications, XML queries are translated into SQL queries containing key/foreign key joins in order to construct XML documents. This process introduces common subexpressions between the SQL queries used to build a document. Here, we discuss optimization of such common subexpressions in the publishing application.

2.1 Legacy Data

We consider a simplified version of the relational schema of the TPC-H benchmark [15]. This schema describes parts ordered by customers and provided by suppliers.

```
CUSTOMER[C_CUSTKEY,C_NAME,C_ACCTBAL,C_NATIONKEY]
NATION[N_NATIONKEY,N_NAME,N_REGIONKEY]
REGION[R_REGIONKEY,R_NAME,R_COMMENT]
```

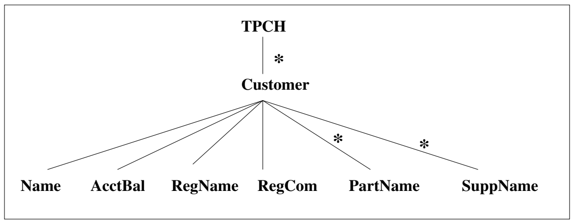

Figure 1. Publishing Legacy Data: DTD

```
PART[P_PARTKEY,P_NAME]
SUPPLIER[S_SUPPKEY,S_NAME,S_NATIONKEY]
ORDERS[O_ORDERKEY,O_CUSTKEY]
LINEITEM[L_ORDERKEY,L_PARTKEY,L_SUPPKEY]
```

2.2 Publishing Legacy Data as XML

We want to publish an XML document that conforms to the DTD (shown as a tree) given in Figure 1; edges labeled with a '$*$' are used for repeated sub-elements. In order to construct the XML document, an SQL query is generated for each element in the DTD, as follows (where $\mathtt{J1} = \mathtt{CUSTOMER} \bowtie_{\mathtt{NATIONKEY}} \mathtt{NATION} \bowtie_{\mathtt{REGIONKEY}} \mathtt{REGION}$ and $\mathtt{J2} = \mathtt{CUSTOMER} \bowtie_{\mathtt{CUSTKEY}} \mathtt{ORDERS} \bowtie_{\mathtt{ORDERKEY}} \mathtt{LINEITEM}$):

$$Q_{\mathtt{Customer}} = \pi_{\mathtt{C\_CUSTKEY}}(\mathtt{CUSTOMER})$$
$$Q_{\mathtt{Name}} = \pi_{\mathtt{C\_CUSTKEY,C\_NAME}}(\mathtt{CUSTOMER})$$
$$Q_{\mathtt{AcctBal}} = \pi_{\mathtt{C\_CUSTKEY,C\_ACCTBAL}}(\mathtt{CUSTOMER})$$
$$Q_{\mathtt{RegName}} = \pi_{\mathtt{C\_CUSTKEY,R\_NAME}}(\mathtt{J1})$$
$$Q_{\mathtt{RegCom}} = \pi_{\mathtt{C\_CUSTKEY,R\_COMMENT}}(\mathtt{J1})$$
$$Q_{\mathtt{PartName}} = \pi_{\mathtt{C\_CUSTKEY,P\_NAME}}(\mathtt{J2} \bowtie_{\mathtt{PARTKEY}} \mathtt{PART})$$
$$Q_{\mathtt{SuppName}} = \pi_{\mathtt{C\_CUSTKEY,S\_NAME}}(\mathtt{J2} \bowtie_{\mathtt{SUPPKEY}} \mathtt{SUPPLIER})$$

2.3 Common Expressions in Sibling Queries

Several queries for sibling elements share common expressions, raising the possibility of shared (optimized) computation. The simplest example is the case of $Q_{\mathtt{Name}}$ and $Q_{\mathtt{AcctBal}}$ that are both projections of the same CUSTOMER table. This is due to the fact that different fields of the CUSTOMER table generate different sub-elements. These sibling queries could be merged into a single query:

$$Q_{\mathtt{C}} = \pi_{\mathtt{C\_CUSTKEY,C\_NAME,C\_ACCTBAL}}(\mathtt{CUSTOMER})$$

Note that, in this case, the parent query, $Q_{\mathtt{Customer}}$, shares the same common expression as the two sibling queries.

The second example is that of two sibling queries $Q_{\mathtt{RegName}}$ and $Q_{\mathtt{RegCom}}$ that share a common join expression J1. This is due to the fact that nations and regions are normalized into tables and that recovering them requires performing joins with these (intermediate) tables. By merging $Q_{\mathtt{RegName}}$ and $Q_{\mathtt{RegCom}}$ into a single query, the common expression is evaluated only once:

$$Q_{\mathtt{RegNameCom}} = \pi_{\mathtt{C\_CUSTKEY,R\_NAME,R\_COMMENT}}(\mathtt{J1})$$

Note that, in this case, the common expression shared between the parent query and each of the children queries is a sub-expression of that shared between the sibling queries.

The last example is the case of two sibling queries $Q_{\mathtt{PartName}}$ and $Q_{\mathtt{SuppName}}$ that share a common join expression J2. However, due to the fact that a customer can have multiple parts and multiple suppliers, sharing computation is less straightforward. Merging $Q_{\mathtt{PartName}}$ and $Q_{\mathtt{SuppName}}$, using an (outer)join, could result in replicated data, which can slow down query processing as well as communication. There are two ways to avoid replicated data in the result of the merged query, without repeating computation. First, the query is rewritten using an *outer union*, where the common subexpression is factored out [4]:

$$\begin{aligned} Q_{\mathtt{PartSupp}} = (&\pi_{\mathtt{C\_CUSTKEY,P\_NAME,NULL}}(\mathtt{J2} \bowtie_{\mathtt{PARTKEY}} \mathtt{PART})) \\ &\cup(\pi_{\mathtt{C\_CUSTKEY,NULL,S\_NAME}}(\mathtt{J2} \bowtie_{\mathtt{SUPPKEY}} \mathtt{SUPPLIER})) \end{aligned}$$

This is a good option if the relational database optimizer is able to optimize outer unions with shared expressions. Second, the relational database engine is forced to compute and materialize J2, and then use it to evaluate each of the two queries $Q_{\mathtt{PartName}}$ and $Q_{\mathtt{SuppName}}$. As a final option, the relational database optimizer could take advantage of the availability of some indices, and choose different plans to evaluate the common join expression in the two queries, $Q_{\mathtt{PartName}}$ and $Q_{\mathtt{SuppName}}$, possibly resulting in a cheaper evaluation than any of the merged query alternatives.

2.4 Merging Parent-Child Queries

Merging of queries for a parent and its children elements is sometimes beneficial, as with the merging of the parent query $Q_{\mathtt{Customer}}$ with the children queries $Q_{\mathtt{Name}}$ and $Q_{\mathtt{AcctBal}}$, to obtain $Q_{\mathtt{C}}$, or the merging of $Q_{\mathtt{Customer}}$ with $Q_{\mathtt{PartName}}$ or $Q_{\mathtt{SuppName}}$.

Often, though, merging a parent query with a child query can result in replicated data, as with the (outer)join of $Q_{\mathtt{C}}$ with $Q_{\mathtt{PartName}}$ or $Q_{\mathtt{SuppName}}$; each customer tuple (along with its fields) will be replicated as many times as the number of parts or suppliers for this customer.

Essentially, when a parent element has both unique and repeated children elements, parent-child merges result in conflicting optimizations: the first creates "fat" nodes, the second works well with "slim" nodes. Sibling merges, via outer unions, on the other hand, avoids such conflicts. Hence, depending on the amount of replicated data, it might be desirable to optimize queries at siblings separately from their parent query.

3 Related Work

The use of an optimization algorithm to find the SQL queries that offer the best compromise between communication and processing costs when publishing relational data in XML has been first explored in the Silkroute system [6]. The authors also provide a language to express XML views of relational data and a query composition algorithm that composes an XML query with a view definition to generate an initial set of SQL queries that are given to the optimizer.

Another related work in the research community is the one described in [13]. The authors provide an extension to SQL to express XML views of relations and perform an experimental study of publishing relational data in XML. This work has not adopted an optimization approach to this problem. Rather, it derived a set of heuristics that can be useful when implementing a system where the choice of which SQL queries to generate for a given view specification is fixed and hard-coded depending on the nature of the underlying application.

The recent work presented in [2] addresses issues related to DTD-directed publishing. It introduces a new data structure called ATG (Attribute Translation Grammar), an extension of DTDs that associates attributes and semantic rules with elements. A dynamic programming algorithm has been designed to generate XML documents using ATGs.

There are three main commercial XML publishing systems: Microsoft SQL Server 2000 [14], Oracle XML SQL Utility [16] and IBM DB2 XML Extender [17]. These systems provide a language interface for expressing XML views of relations, and rely on the user to specify SQL queries. Our approach could be used with each of them.

4 Conclusion

We discussed the problem of efficiently publishing relational data in XML and showed that exploring common computation between sibling queries is fundamental to this problem. In particular, in the case where an element has both unique and repeated children elements, sibling merging combined with common subexpression elimination guarantees computation sharing without replicating data (which is not the case for parent-child merging), reducing both processing and communication times.

In the full version of this technical note, we describe several rewriting-based optimization techniques that exploit shared computation between queries used to build an XML document, design an optimization algorithm that applies our rewritings to find the set of SQL queries that achieves a good compromise between processing and communication times, and provide an experimental study that compares several possible ways of sharing common computation between sibling queries used to build an XML document.

References

[1] S. Banerjee, V. Krishnamurthy, M. Krishnaprasad, R. Murthy. Oracle8i - The XML enabled data management system. ICDE 2000.

[2] M. Benedikt, C. Y. Chan, W. Fan, R. Rastogi, S. Zheng, A. Zhou. DTD-directed publishing with attribute translation grammars. VLDB 2002.

[3] P. Bohannon, J. Freire, P. Roy, J. Simeon. From XML schema to relations: A cost-based approach to XML storage. ICDE 2002.

[4] M. J. Carey, J. Kiernan, J. Shanmugasundaram, E. J. Shekita, S. N. Subramanian. XPERANTO: Middleware for publishing object-relational data as XML documents. VLDB 2000.

[5] J. M. Cheng, J. Xu. XML and DB2. ICDE 2000.

[6] M. Fernandez, A. Morishima, D. Suciu. Efficient evaluation of XML middle-ware queries. SIGMOD 2001.

[7] D. Florescu, D. Kossmann. A performance evaluation of alternative mapping schemes for storing XML in a relational database. IEEE Data Eng. Bulletin 1999.

[8] C. C. Kanne, G. Moerkotte. Efficient storage of XML data. ICDE 2000.

[9] P. Roy, S. Seshadri, S. Sudarshan, S. Bhobe. Efficient and extensible algorithms for multi query optimization. SIGMOD 2000.

[10] M. Rys. Bringing the Internet to your database: Using SQLServer 2000 and XML to build loosely-coupled systems. ICDE 2001.

[11] J. Shanmugasundaram, H. Gang, K. Tufte, C. Zhang, D. J. DeWitt, J. F. Naughton. Relational databases for querying XML documents: Limitations and opportunities. VLDB 1999.

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

[13] J. Shanmugasundaram, E.J. Shekita, R. Barr, M.J. Carey, B. G. Lindsay, H. Pirahesh, B. Reinwald. Efficiently publishing relational data as XML documents. VLDB Journal 10(2-3): 133-154 (2001).

[14] Support WebCast: Microsoft SQL Server 2000. New XML Features. http://support.microsoft.com/servicedesks/ Webcasts/wc042800/wcblurb042800.asp. April 2000.

[15] Transaction Processing Performance Council. TPC-H benchmark: Decision support for ad-hoc queries. http://www.tpc.org/.

[16] B. Wait. Using XML in Oracle Database Application. http://technet.oracle.com/tech/xml/info/ htdocs/otnwp/about_xml.htm. Nov. 1999.

[17] XML Extender Administration and Programming. IBM DB2 Universal Database XML Extender. http://www-4.ibm.com/software/data/db2/extender/xmlext/ docs/v71wrk/english/index.htm.