

The Table and the Tree: On-Line Access to Relational Data through Virtual XML Documents

P. Bohannon

H. F. Korth

P.P.S. Narayan

Bell Laboratories
600 Mountain Avenue
Murray Hill, NJ 07974 USA
{bohannon,hfk,ppsnarayan}@lucent.com

Abstract

For speed and convenience, applications routinely cache XML data locally, and access it through standard parser (SAX) or tree (DOM) interfaces. When the source of this data is a relational database, the consistency and integrity guarantees of the database are sacrificed for speed. We present the ROLEX system (standing for Relational On-Line Exchange with XML) which provides applications “live” virtual XML views of relational data through standard interfaces. This technology, combined with a main-memory database platform, promises to provide data integrity, consistent interoperation with relational applications, and the performance required by busy web or e-commerce applications.

1 Introduction

XML has gained widespread popularity as a standard for information representation and exchange. Popular infrastructure software for business hubs, supply chain integration, and catalog management all use XML encodings, and standards bodies such as RosettaNet and Oasis-Open are extremely active.

In these applications, the *source* of the data published as XML is frequently a relational database engine. This has led to a so-called “shred-and-publish” approach where incoming XML data is parsed (shredded) into relational tables and outgoing data is extracted by SQL engines then formatted (published) as XML. Recent work addresses the techniques required to define the mapping between XML data and relations, format SQL output as XML [3], and support translation of queries posed in XML query languages into SQL [3, 7]. However, other research suggests that traditional databases do not perform well enough to support a busy web server directly, even when appropriate query results for web pages are materialized in the DBMS [8]. Such speed issues and a concern for independence from a particular DBMS has led to widespread *application caching* of business data. For e-business, an obvious format for this cached data is XML. While it may solve the performance problem, the application cache is undesirable for a number of reasons. First, multiple applications must each re-implement a portion of the functionality provided by the DBMS. Second, concurrency and data integrity must be managed by the application. This is particularly worrisome when the underlying relational data is being accessed and updated by previously-existing applications, while cached copies of this data are being used by e-business applications.

In this paper, we introduce the ROLEX system (Relational On-Line Exchange with XML), which is designed to demonstrate that relational database technology can adapt not only to XML as an encoding format, but also to the speed and standardization requirements of e-commerce applications. In particular, we are designing ROLEX to

- Provide XML data through a standard interface.
- Support the speed and throughput requirements of front-end web-servers or e-business clients.
- Maintain transaction consistency between the data seen by the web applications and the data seen by “legacy” DBMS applications.

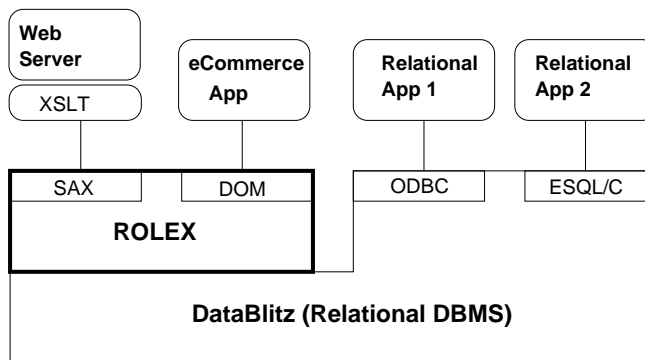


Figure 1: Role of ROLEX

To accomplish these goals, ROLEX provides applications with relational data in *virtual XML documents* through standard interfaces like DOM [14] or SAX [11], as shown in Figure 1. The virtual document interface has the potential to provide efficient, simultaneous and consistent access to relational data and XML views of that data. Furthermore, it provides an opportunity for a smooth transition to handling XML query languages on a relational DBMS, since these query language implementations can usually handle documents presented in a DOM or SAX interface. Thus query processing functionality for complicated features such as order, meta-data queries, recursive queries, etc., can continue to be handled by the XML query processor, yet it can push certain selections and/or projections down to the underlying relational engine.

However, if the interaction with the DBMS is itself too slow for a chosen application [8], this approach by itself will not be sufficient. To complete our solution, we have chosen to base our implementation of ROLEX on the DataBlitzTM Main-Memory Database System [2], allowing us to capitalize on the extremely low-latency access to data provided by a main-memory architecture. Obviously, many challenges arise in such an approach. Replacing application-specific memory caches with a shared main-memory database raises several issues, including efficient implementation of views (including partial or complete materialization as DOM structures), adapting the view implementation to user navigation, and efficiently ensuring consistency between the XML view and the underlying relational data. These are the issues being addressed by ROLEX.

The outline of the remainder of the paper is as follows. After discussing related work in Section 2, we give some background and a motivating example in Section 3. We introduce the ROLEX system architecture in Section 4. We briefly describe an implementation of view traversal and its performance in Section 5. Our conclusions are presented in Section 6.

2 Related Work

ROLEX addresses issues required to support high-performance *navigation* of XML views of relational data. It is most closely-related to previous work on publishing relational data as XML [3, 7] and on storing XML in object databases [6].

In [3], the authors describe the XPERANTO project, which addresses relational storage of XML documents, query translation between XML query languages and SQL, and combined querying of XML and relational data through XML query languages. SILKROUTE [7] defines a similar query mapping between XML and relational data, using a new language RXL which combines features of SQL and XML-QL [4], and gives algorithms for query translation between XML-QL and SQL. Both XPERANTO and SILKROUTE are related to ROLEX in the concern for efficient retrieval. However, we focus on interaction with XML interface standards, efficient navigation, and efficient evaluation of specific parameterized queries. For example, ROLEX may skip the evaluation of one or more joins on a particular traversal of the data, and may allow the processing to be handled partially by the relational query processor and partially by the XML query processor.

Work which deals primarily with the mapping of XML data to relational data models for efficient storage (see, for example, [5, 13]) is less related to ROLEX, as we assume such a mapping exists and focus on efficient retrieval. We note that our techniques can be applied to navigation of XML documents stored in a relational system as well as for relational data exported to XML applications, though the latter is emphasized in this paper. One commercial system [6]

HOTELS							
ID	Name	Kind	StreetAddr	City	SC	Pool	Gym
1	Hyatt	2Star	South St.	Murray Hill	NJ	Y	N
2	Grand	4Star	Central Av.	Springfield	NJ	N	Y

STATES	
SC	Name
NJ	New Jersey
NY	New York
PA	Pennsylvania

PHONES	
HotelID	Number
1	(908)5551234
1	(908)5551258
2	(908)5551264

ROOMS			
HotelID	Number	Type	Price
1	101	S	125.00
2	1001	D	500.00
1	102	D	200.00

AVAILABILITY			
HotelID	Number	Start Date	EndDate
1	101	10/30/2001	11/04/2001
2	1001	07/03/2001	07/07/2001

```

<accommodations>
  <hotel>
    <name> Hyatt </name>
    <amenities pool="Y" gym="N"> </amenities>
    <phone number="(908) 555-1234"> </phone>
    <phone number="(908) 555-1258"> </phone>
    <address sc="NJ">
      <street> South St. </street>
      <city> Murray Hill </city>
      <state> New Jersey </state>
    </address>
    <available>
      <room type="S" price="125.00">
        <number> 101 </number>
        <dates>
          <from> 10/30/2001 </from>
          <to> 11/04/2001 </to>
        </dates>
      </room>
      <room type=...>
        ...
      </room>
      ...
    </available>
  </hotel>
  <hotel>
    ...
  </hotel>
  ...
</accommodations>

```

(a)

(b)

Figure 2: A hotel reservation example: relational data and XML view

and one research system [1] have mapped XML to an object-oriented data model. This is in some sense more related to our work, since the DOM interface is directly supported by the database, which provides persistence, transactions, indexing, etc. Further, [6] provides caching for data extracted from relational databases. However, this interaction is through ODBC, unlike ROLEX, in which the relational and XML data models simultaneously access the same data. Finally, work addressing the caching of dynamic content for web publication, for example [9, 12] is related to ROLEX. While these systems assume that the view or query results will be materialized outside the DBMS, some of the same issues may be faced by us when we begin partially materializing document trees to further improve access speed, a topic of future work.

3 Background and Motivation

Our motivating example is drawn from a hotel reservation application, and Figure 2(a) shows some relational tables used to support the making of reservations. For this example, we assume further that in addition to supporting one or more call-centers and hotels with a traditional SQL-based application, the hotel chain conducts business electronically in two ways:

1. Through a web-site for browsing and reserving rooms which supports dynamic access to the reservations database. This web-site has been using direct access to the relational engine on each request to generate HTML, but would like to move to a new system that generates XML, so that reservations can be made more easily with mobile clients.
2. Through a contract with an external on-line reservation “hub”, say TRAVELEDIA.COM. This contract requires that their interactions with TRAVELEDIA take place using an XML representation following an XML standard for the travel industry.

An example XML representation of the data in Figure 2(a) is shown in Figure 2(b).

Clearly, the service can simply ship each user, web or hub request to the relational system for processing. However, this multi-tiered approach leads to slow response times, particularly for on-line users who are browsing rates and availability at several hotels prior to deciding on a booking. Such users may generate queries that access data on several hotels at once (e.g. “Find hotels within 10 miles of New Providence that have non-smoking rooms on May 25, 2001 and that offer either an indoor pool or an exercise room”), and generate a heavy workload on the hotel chain’s relational database.

3.1 SAX and DOM: Application Access to XML

When the demands of the workload lead to excessive hardware cost, the hotel chain may instead choose to cache information about available rooms as XML documents in application servers outside the DBMS. To access the cached XML data, either of two standard interfaces may be employed, SAX or DOM. SAX [11], is a parser-based interface which returns elements of the document in order, while DOM [14] exports the model of an untyped object tree. Obviously, a parser-based interface requires little storage of the document being processed and is generally much lighter-weight than a tree-based interface, and for this reason the SAX interface is popular with many applications. By contrast, the DOM interface stores an internal representation of an XML document, and allows the user to navigate and update this representation after the original parse. This interface is organized around subclasses of the `DOMNode` class, and navigation is accomplished with `firstChild/nextSibling` calls.

3.2 Access Patterns

While complicated access patterns are certainly possible within the DOM framework, and arbitrary navigation will be supported in ROLEX, we focus in this paper on two access patterns which we expect will be common in performance-critical workloads, document traversal and evaluation of parameterized queries.

The first access pattern we consider is a traversal of a virtual XML document. As an example of document traversal, consider traversing the document shown in Figure 2(b) to generate a web page which shows all the hotels and their phone numbers. Note that this does not require traversal of the `<available>` subtree, and thus access to the `ROOMS` and `AVAILABILITY` relations, and the associated join processing should be avoided. For this example, the cost could also be avoided by producing a separate virtual document without the subtree, but this is not the case if the application uses the more complete virtual document, but only occasionally accesses the subtree. Avoiding computation not needed by a particular navigation of the tree, as in this example, is a key feature of ROLEX.

The second access pattern we consider is that of a parameterized query. In the context of our hotel-reservation example, a typical parameterized query asks for the availability of certain rooms in a certain city on a certain date. These queries must be executed efficiently, and transactional consistency must be maintained with room reservation applications executing through the SQL interface. Furthermore, when these queries are evaluated against a DOM interface, further navigation is possible from the resulting values. For example, the rooms returned from the room availability query are in fact `DOMNode`s from which arbitrary further navigation will be possible - for example, it will be possible to navigate up from a returned room node and determine what hotel it is in.

4 Design of ROLEX

In this section we present the design of ROLEX and the issues faced in its implementation. Views defined in ROLEX may be *parameterized*. For example, in the room availability query, the start-date, end-date and city may be parameters to the query. For each such view, a *mapping* defines the XML structure of the resulting document, and its relation to the underlying table or tables. Based on this mapping and usage patterns, some portion of the data (or structure) of the resulting document may be *materialized*. Further, for each view, one or more *execution plans* are available, to support some number of *navigation patterns*. For example, “sequential scan” and “DOM breadth-first traversal” are two possible navigation patterns.

In [3, 7], a mapping between an XML document and relational tables is established by either an extended SQL syntax [3], or a modified semi-structured query language [7]. ROLEX does not presume a particular syntax for establishing a mapping. Note that a mapping is closely related to a query execution plan for a semistructured query (see, e.g., [10]).

The design of ROLEX allows partial materialization of virtual XML views *directly* into objects in the target API. For example, in the case of DOM, it is possible to keep portions of frequently used XML trees materialized as DOM nodes. Trigger mechanisms of DataBlitz will be used to either keep the materialized nodes up-to-date, or to invalidate sections of the materialized tree. Note that the structure of DOM along with the in-memory architecture of DataBlitz allows for fine-grained selection of data to materialize. For example, one reasonable strategy is to materialize the more stable parent-child relationships, while for other parent-child relationships, attribute and text values, the result is looked up on demand from the underlying data manager.

To support key application queries like the room search discussed in Section 3, we allow views to be parameterized, thus pushing certain selections down into the query plan being used by ROLEX. Specific values for query parameters are passed when the virtual document is opened.

Finally, the central concern of ROLEX is to support efficient navigation of the resulting virtual document. Two technologies are required to accomplish this: First, the query optimizer must be modified to produce optimal execution plans based on a *probabilistic model* of navigation behavior. Second, efficient techniques must be developed to avoid unnecessary work based on specific navigation choices made by the user. One obviously common access pattern is sequential navigation of the document tree, for example to export the document (or part of it) as text. Another important access pattern might be a top-down evaluation of a path expression. If no single query plan can adapt to give good performance to both navigation patterns, then it may be important to store two query plans. ROLEX is designed to recognize the access pattern based on the first few accesses and use the appropriate query plan, and within that query plan, to adapt to fine-grained navigation choices by not producing the data required for subtrees that are not explored. None of these optimizations are available when XML documents are exported *in toto* from the execution of a single SQL query from which a nested document is to be extracted.

5 Implementation

We are in the process of implementing ROLEX on top of the DataBlitz Main-Memory Database [2]. Our current prototype supports virtual document mappings based on a restricted set of query execution plans. Nesting of data from one or more tables is allowed, and data from the same table can be used in different parts of the tree. Data in a subtree is always joined to data in a parent node by an outer join, so that the parent node always appears regardless of the child node. Joins are evaluated using nested loop scans with index lookups into the inner relation.

Based on this prototype, we have conducted experimental scans of a larger instance of the virtual document shown in Figure 2(b). The idea of the experiment is to observe the advantage afforded by adapting to varying navigation patterns. To this end, for each <hotel> node encountered, the “scan user” either scans the entire subtree, or scans the subtree but avoids the subtree under the <available> tag, depending on some probability, *res*. (This probability represents a selection computed in the application). The relational tables contains 100 randomly generated hotels, each with 100 rooms. Each room has a single record in the AVAILABILITY table.

The experiments were conducted on an 296Mhz UltraSPARC-II with 256MB of RAM. Data was fully present in RAM. While the test was run with a single active process, sufficient locking was done to ensure serializability. We measured the total time required to scan the virtual XML document of the database. We measured the *events* that represented: 1) starting and ending of elements in the document, 2) text nodes within the elements, and 3) attributes of elements. We also measured the *data node visits*, corresponding to the text and the attribute nodes.

The results of the experiment, shown in Figure 3, allowed us to observe two features. First, the system could support a high volume of scans, with a scan of the entire database accomplished in 5.76 seconds, and an *event* generated every 37.7 microseconds. Second, we observed that this performance was improved by taking advantage of variations in the user’s access patterns. When the <available> subtree was never traversed (*res* = 0), the cost of two joins (HOTELS \bowtie AVAILABILITY \bowtie ROOMS) was avoided leading to faster access of the traversed subtrees. Similarly, when the <available> subtree was traversed with some probability, the two joins were evaluated only when needed, thus giving us faster scans than when all joins are always computed. This is further validated by varying *res* as shown in the table.

6 Conclusion

In this paper we have presented the design of ROLEX, a system intended to allow standard relational applications and XML-based e-commerce applications to be *tightly coupled* for performance and data consistency, while still interacting

res	Total Scan Time (seconds)	Total Number of		Avg Time Per (microseconds)	
		Events	Data Nodes	Event	Data Node
1.0	5.76	153002	51000	37.68	113.04
0.75	4.32	115452	38500	37.48	112.39
0.5	2.77	77902	26000	35.67	106.88
0.25	1.40	40352	13500	34.69	103.70
0.0	0.038	2802	1000	13.61	38.15

Figure 3: **Adapting to Navigation Patterns**

through *standard interfaces* like SQL, SAX and DOM. We believe ROLEX has the potential to replace application caches with suitably high-performance database technology, thus bringing the relational DBMS to the front-line of electronic commerce implementations.

Our future work includes addressing a number of issues in optimizing queries for navigation and fine-grained view materialization. Also, we plan to build on our initial implementation to demonstrate simultaneous access to data from both relational and DOM-based applications.

References

- [1] S. Abiteboul, V. Aguilera, S. Ailleret, B. Amann, S. Cluet, B. Hills, F. Hubert, J.-C. Mamou, A. Marian, L. Mignet, T. Milo, C. Souza dos Santos, B. Tessier, and A.-M. Vercoustre. Xml repository and active views demonstration. In *VLDB'99, Proceedings of 25th International Conference on Very Large Data Bases*, 1999.
- [2] J. Baulier, P. Bohannon, S. Gogate, C. Gupta, S. Haldar, S. Joshi, A. Khivesera, H. F. Korth, P. McIlroy, J. Miller, P.P.S. Narayan, M. Nemeth, R. Rastogi, S. Seshadri, A. Silberschatz, S. Sudarshan, M. Wilder, and C. Wei. Datablitz storage manager: Main memory database performance for critical applications. In *Proceedings of the ACM SIGMOD International Conference on the Management of Data*, 1999.
- [3] M. Carey, D. Florescu, Z. Ives, Y. Lu, J. Shanmugasundaram, E. Shekita, and S. Subramanian. XPERANTO: Publishing object-relational data as XML. In *Proceedings of the Third International Workshop on the Web and Databases*, May 2000.
- [4] A. Deutsch, M. Fernández, D. Florescu, A. Levy, and D. Suciu. A query language for XML. In *Proceedings of the Eighth World-Wide Web Conference*, 1999.
- [5] A. Deutsch, M. Fernández, and D. Suciu. Storing semistructured data with STORED. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 1999.
- [6] eXcelon Corporation. An XML data server for enterprise web applications. (White Paper) www.exceloncorp.com/products/whitepapers.html.
- [7] M. Fernández, D. Suciu, and W.C. Tan. SilkRoute: Trading between relations and XML. In *Proceedings of the WWW9*, 2000.
- [8] A. Labrinidis and N. Roussopoulos. WebView materialization. In *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data: May 16–18, 2000, Dallas, Texas*, 2000.
- [9] Q. Luo, J. Nuaghton, R. Krishnamurthy, P. Cao, and Y. Li. Active query caching for database web servers. In *Proceedings of WebDB 2000*, pages 29–34, 2000.
- [10] J. McHugh and J. Widom. Query optimization for XML. In *Proceedings of the Twenty-Fifth International Conference on Very Large Data Bases*, September 1999.
- [11] David Megginson. SAX 2.0: The simple API for XML. <http://www.megginson.com/SAX/>.
- [12] L. Quan, L. Chen, and E. Rudensteiner. Agros: Efficient refresh of an xql-based web-caching system. In *Proceedings of WebDB 2000*, 2000.
- [13] J. Shanmugasundaram, H. Gang, K. Tufte, C. Zhang, D. J. DeWitt, and J. Naughton. Relational databases for querying XML documents: Limitations and opportunities. In Malcolm Atkinson et al., editors, *Proceedings of the Twenty-fifth International Conference on Very Large Databases, Edinburgh, Scotland, UK, 7–10 September, 1999*, pages 302–314, Los Altos, CA 94022, USA, 1999. Morgan Kaufmann Publishers.
- [14] W3C. Document object model(DOM). <http://www.w3.org/DOM/>.