Temporal Query Operators in XML Databases

Kjetil Nørvåg*
Department of Computer and Information Science
Norwegian University of Science and Technology
7491 Trondheim, Norway
Kjetil.Norvag@idi.ntnu.no

ABSTRACT

The contents of an XML database or an XML/Web data warehouse is seldom static. New documents are created, documents are deleted, and more important: documents are updated. In many cases, we want to be able to search in historical versions, retrieve documents valid at a certain time, query changes to documents, etc. This can be supported by extending the system with temporal database features. In this paper we describe the new query operators needed in order to support an XML query language which supports temporal operations.

Keywords

XML, temporal databases, query processing

1. INTRODUCTION

The amount of data available in XML is rapidly increasing. One of the advantages of XML is that a document itself contains information that is normally associated with a schema. This makes it possible to do more precise queries, compared to what has been previously possible with unstructured data. Queries against the XML data can either be directly to the database storing the XML data (for example an object-relational database system), or to an XML data warehouse, created from XML data collected from the Web (for example Xyleme [14]).

The contents of a database or data warehouse is seldom static. New documents are created, documents are deleted, and more important: documents are updated. In many cases, we want to be able to search in historical (old) versions, retrieve documents that was valid at a certain time, query changes to documents, etc. (Note that although this has some similarities to general document versioning maintenance, the aspect of time makes possibilities as well as appropriate solutions different.) The "easiest" way to realize this is to store all versions of all documents in the database, and use a middleware layer to convert temporal query language statements into conventional statements, executed by an underlying database system (also called a *stratum approach* [8]). Although this approach makes the introduction of temporal support easier, it can be difficult to achieve good performance: temporal query processing is in general costly, and the cost of storing the complete document versions can be too high. For this purpose, a *temporal XML database system* is necessary.

In order to realize an efficient temporal XML database system, several issues have to be solved. The most important issues are 1) efficient storage of versioned XML documents, 2) efficient indexing of temporal XML documents and 3) temporal XML query processing. As for the first two issues, some related work has already been done (see Section 2). In this paper, we concentrate on the issue of temporal XML query processing. We give a brief outline of what can be expected to be found in a temporal XML query language, and introduce the query operators necessary to execute such queries.

The organization of the rest of this paper is as follows. In Section 2 we give an overview of related work. In Section 3 we study time and identity in the context of temporal XML queries. In Section 4 we described the assumed data model. In Section 5 we give examples of temporal XML queries. In Section 6 we give an overview of the algebra operators and illustrate their use in some example queries. Finally, in Section 7, we conclude the paper and outline issues for further research.

2. RELATED WORK

A model for representing changes in semistructured data (DOEM) and a language for querying changes (Chorel) was presented by Chawathe et al. in [2, 3]. Chorel queries were translated to Lorel (a language for querying semistructured data), and can therefore be viewed as a stratum approach. The work by Chawathe et al. has later been extended by Oliboni et al. [11].

Storage of versioned documents are studied by Marian et al. [9] and Chien et al. [4, 5, 12]. Chien et al. also consider access to previous versions, but only snapshots retrievals.

An approach that is orthogonal, but related to the work pre-

*This work was done while the author was an ERCIM fellow at the VERSO group at INRIA, France.

SAC 2002, Madrid, Spain
© 2002 ACM 1-58113-445-2/02/03...$5.00

sented in this paper, is to introduce valid time features into XML documents, as presented by Grandi and Mandreoli [6, 7].

An extended version of this paper is available as [10].

3. TEMPORAL XML QUERIES

Two important issues that pose some additional difficulties in the context of XML, and in particular in the context of XML documents retrieved from the Web (in the case of an XML data warehouse), are time and identity. These issues will now be discussed in more detail.

3.1 Time in XML databases

In temporal databases we have different aspects of time, where the two most common aspects are transaction time and valid time. In our context, we have two cases which from a query point of view are similar to transaction time:

- Local storage of documents (e.g., in a database system storing XML documents), where we have full information on time of creation/storage of an XML document. In this case, time is exactly transaction-time equivalent.
- XML warehouse or other non-synchronized storage of copies of XML documents. Although similar to transaction time, this is not exactly the same. Important differences are:
  - In this case we in general do not know the time of creation/storage of an XML document, only the time when the document was retrieved from the Web ("crawled").
  - The documents in the warehouse are not retrieved at the same point in time, the result is an inconsistent view of the documents. For example, a document can have references to a document not yet retrieved, or a document that has already been deleted.
  - There might have been updates between the versions we have retrieved, i.e., we do not necessarily have all the versions of a particular document.

A third case, which has similarities to valid time, is document time. Many documents include a timestamp in the document itself. This can for example be the time the document was written, or when it was posted. Examples are news notices from the news agencies. The documents can also be indexed and queried based on this document time. Although it could be difficult to extract this time from a document automatically, we can expect many documents to include this metadata in a "standardized" way, based on RDF. One example is XMLNews-Meta (based in part on RDF), which if used can provide meta-information such as publication time and expire time.

In this paper, we concentrate on a transaction-time support. It should be noted that the techniques presented here are equally applicable to a valid-time context, but that additional operators should be introduced in that case, for example coalescing.

3.2 Identity of elements in versioned XML documents

XML documents have a quasi-persistent[1] identifier, the URL. However, in general the elements of a document *do not* have any identity of their own that persist from one version of a document to the next. This implies that many queries can be difficult to express, as well as expensive to execute. Two simple examples are 1) a query for the create time of elements, and 2) a query asking for the previous version of a certain element. Thus, although elements seen from "the outside" do not have persistent identifiers, we believe that the storage system should support this feature, in order to make it a part of the data model for the query language. One particular system that provides this functionality is Xyleme [9]. The persistent identifiers, in Xyleme called XIDs, identify an element in a particular document in a time independent manner, and will not be reused when an element is deleted. For convenience we will in this paper also use the acronym EID (Element ID), which is the concatenation of document ID and XID. Thus, an EID identifies uniquely a particular element in a particular document.

In a temporal XML database there will in general be more than one version of each element (different versions of an element have the same EID). In order to uniquely identify a particular version of an element, the timestamp can be used together with the EID. We denote the identifier of a particular version of an element TEID (temporal EID), i.e., the concatenation of EID and timestamp.

4. ASSUMED DATA MODEL

A document in the database is viewed as a forest of trees. One of the advantages of this approach is that queries on versions of a document (or several documents) is similar to the query on a general set of XML documents which can also be view as a forest of trees, or the forest of trees resulting from pre-filtering (i.e., returning subtrees of documents, possibly more than one tree for each document).

We assume that:

- Every element has a timestamp.
- Every update of an element also implies update of the element it is contained in. Note that even if this logically has to be applied recursively up to the document to the root, *it does not have to be implemented in this way.*

Note that the distinction between document timestamp and element timestamp is not significant for snapshot queries, only for change-oriented queries. An example of document versions can be seen in Figure 1. The document versions are versions of a restaurant guide database, as described in [3]. The restaurant guide will also be used in the examples below.

Note that in the *physical* storage model, it is unlikely that all versions of all documents are stored as complete versions.

[1] *Quasi*-persistent based on the observation that documents on the Web frequently are moved.

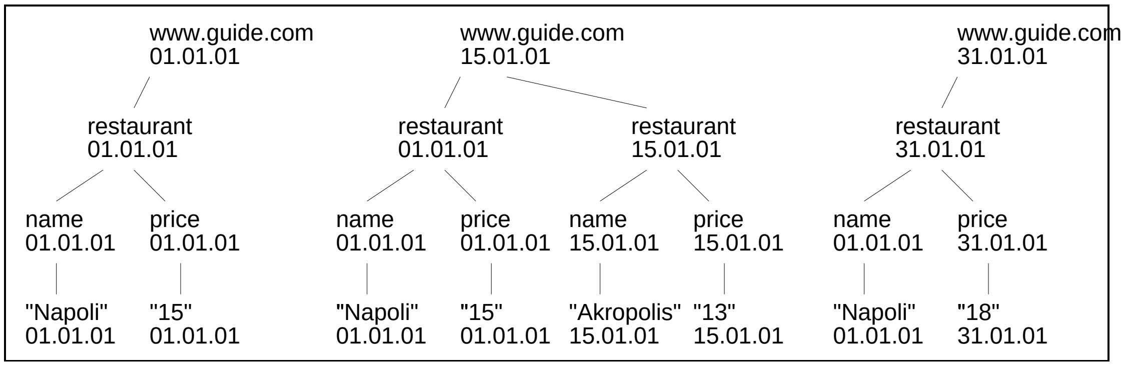

Figure 1: The restaurant list at guide.com as retrieved on January 1st, January 15th, and January 31st. The timestamps on each element is the time of update of the element or one of its children.

Instead, previous versions are stored as, e.g., delta versions. In order to reconstruct these previous versions, we might have to retrieve and process the complete last versions as well as a number of delta documents.

5. EXAMPLES OF TEMPORAL XML QUERIES

The main purpose of this section is not to create a new query language, but to describe what kind of queries can be expected in a temporal XML database. The query language is based on a mix of Lorel, the Xyleme query language[2] [1], and elements of XPath and XQuery [13] (note that the query operators and associated algorithms are independent of which query language is actually used). For example, a query returning all restaurants with price less than $10 could be written as:

```
SELECT R
FROM doc("http://guide.com/")/restaurant R
WHERE R/price < 10
```

It is assumed that the results of an "outer query" is delivered as default in a document with enclosing tags named <results>. Each result from the from the SELECT expression is delivered in one element with tags named <result>. In most of the following example queries, complete paths are used (i.e., not containing an // operator). However, when querying semistructured data like XML, many queries can be expected to contain the // operator.

In order to retrieve documents valid at a particular time (snapshot query), a timestamp is given for the path in the `FROM` clause, filtering out only those element versions valid at the particular time:

```
SELECT R
FROM doc("http://guide.com/")/restaurant[26/01/2001] R
```

For more complex queries, or when we want more than one version to be selected, we use the keyword EVERY instead of timestamp. For example, in order to retrieve the price history of the restaurant named Napoli:

```
SELECT TIME(R), R/price
FROM doc("http://guide.com/")/restaurant[EVERY] R
WHERE R/name="Napoli"
```

where `TIME(R)` returns the timestamp of the element `R`. Further predicates on time can be included in the SELECT clause, including delete and create time of elements. In order to query time relative to another time, for example `[NOW]` (which denotes the "current time") or a certain time `DD/MM/YYYY`, expressions like `NOW - 14 DAYS` or `26/01/2001 - 2 WEEKS` can be used.

6. ALGEBRA OPERATORS

Here we consider operators that will be needed in temporal XML query processing, and describe them in terms of input, output, and operation. In addition to the operators described here, we also assume the availability of traditional operators, for example projection and join, but we will not discuss them any further.

Two of the operators are extensions of the PatternScan operator described in [1]. The PatternScan operator takes as input a forest of trees (which can be a set of EIDs), which are XML documents or filtered elements (subtrees) from XML document, and a pattern tree which each tree shall be matched against. The pattern tree includes information on projection as well as *isParentOf* and *isAscendantOf* relationships. Informally, we can define the operator as `PatternScan(`F`, pattern)` where F is a forest of trees and `pattern` is the pattern tree. For more details on the PatternScan operator we refer to [1].

6.1 Overview of the operators

The temporal query operators to be added are as follows:

- `TPatternScan(`F`, pattern, `T`)`

  This is a temporal snapshot PatternScan operator. `TPatternScan` is similar to the PatternScan operator

[2]Note that even if Xyleme has support for historical versions/deltas, there is no special support for these in the query language.

in [1], except that it operates on the snapshot of documents valid at time T. The output of the operator is a set of TEIDs (see Section 3.2).

- `TPatternScanAll(`C`, pattern)`

  TPatternScanAll returns all matches for *pattern* for all versions of the documents in a collection C. The output of the operator is a set of TEIDs.

- `DocHistory(document, `T_S`, `T_E`)`

  Returns all the versions of a certain document valid in the interval $[T_S, T_E>$, where $[T_S, T_E>$ is short for the time interval from T_S to T_E, including T_S but not T_E (open-ended upper bound). The output of the operator is a set of TEIDs, where the TEIDs are roots of documents.

- `ElementHistory(EID, `T_S`, `T_E`)`

  Returns all versions of an element valid in the interval $[T_S, T_E>$. The output of the operator is a set of TEIDs (with the EID in the TEID equal to the EID in the input parameters).

- `CreTime(TEID)`

  Returns the create time of an element. This is useful for retrieving elements created before or after a particular time, e.g., as in:

  ```
  SELECT R
  FROM ...
  WHERE CREATE_TIME(R)>=11/01/2001
  ```

  Note that EIDs are unique, so that when an element is deleted the EID will not be reused for a new element. Thus, we do not strictly need timestamps in the element identification for the CreTime and DelTime operators. However, as shown in the extended version of this paper [10], the availability of timestamp can improve performance. The timestamps will in general be available in any case, so that no extra cost is involved by assuming its availability.

- `DelTime(TEID)`

  DelTime returns the delete time of an element.

- `PreviousTS(TEID)`
  `NextTS(TEID)`
  `CurrentTS(EID)`

  Returns the timestamp of the previous/next/current version of a given element version (note that timestamp is not needed for the current version, as this is given implicitly). The timestamp, together with the EID (i.e., the TEID), can be used for retrieving the version itself. These operators can be used in constructions like:

  ```
  SELECT DISTINCT CURRENT(R)/name
  FROM ...
  WHERE ...
  ```

  which retrieves the current versions of elements (possibly generated from a temporal snapshot), and as in

  ```
  SELECT PREVIOUS(R)
  FROM ...
  WHERE ...
  ```

  which retrieve the previous versions of elements.

- `Reconstruct(TEID)`

  Reconstructs the tree rooted at EID in TEID for a particular version. The timestamp T_S in the TEID can, e.g., be the result of `NextTS/PreviousTS/CurrentTS` operations. If previous versions of documents are stored as deltas this can imply accessing and processing a potentially large number of deltas in addition to one complete version (the exception is the current version which will normally be stored as a complete version). If previous versions of documents are stored as complete documents, only the actual version itself needs to be read.

- `Diff(E1, E2)`

  In some cases we want to query for the changes between different versions of elements. These changes can conveniently be returned (and eventually post-processed by the application or in a separate query) as `edit scripts`. Edit scripts describe changes between two versions, similar to for example the information in RCS files. In our context, the edit scripts are XML trees themselves. Note that as long as an edit script is represented in XML this operator does not break closure properties of queries. E1 and E2 can be versions of the same element, but can also represent different documents or subtrees of elements. `Diff` is useful in constructions like:

  ```
  SELECT DIFF(R1,R2)
  FROM ...
  WHERE ...
  ```

It is possible to support string equality and string contain queries with different operators and access structures. However, as described in [10], the access methods for containment queries already exist, so that there is little to gain from providing additional access methods for string equality. Therefore we expect that there are no separate operators and access structures for equality queries, and that the general containment operators/access methods are used, followed by equality testing.

6.2 Example queries

In order to illustrate the use of the operators, we now give three example queries based on the restaurant database example, together with the corresponding query operators.

Q1: List all restaurants in the list as of 26/01/2001:

```
SELECT R
FROM doc("http://guide.com/")/restaurant[26/01/2001] R
```

This is a snapshot query, listing the name in all versions of restaurant elements valid at time 26/01/2001 (that is, versions created before or on 26/01/2001 that is not further updated or deleted).

Operators: `TPatternScan`, followed by `Reconstruct`.

Q2: Retrieve the number of restaurants at 26/01/2001:

```
SELECT SUM(R)
FROM doc("http://guide.com/")/restaurant[26/01/2001] R
```

Operators: `TPatternScan` followed by the traditional aggregate operator `Sum`. Note that reconstruction of the documents is *not* needed. This is important, and shows that in many cases the storage of only deltas of previous document versions does not create performance problems.

Q3: List the price history of the restaurant "Napoli":

```
SELECT TIME(R), R/price
FROM doc("http://guide.com/")/restaurant[EVERY] R
WHERE R/name="Napoli"
```

The use of `EVERY` instead of a particular timestamp retrieves all versions of restaurant. Note that the predicate in the WHERE clause acts on all versions, not only the current version of the elements. As a result, the price history of all restaurants through the history with the name Napoli will be listed.

Operator: `TPatternScanAll`.

7. SUMMARY

We have in this paper described how temporal queries can be executed in an XML database system. We have described issues related to time and identity in this context, and described an appropriate data model as the basis of a temporal XML query language. Temporal support in an XML query language implies that new operators are needed in the query processing, and we have identified operators that will be useful in order to support typical queries in temporal XML databases. In order to achieve the desired performance in such databases, efficient execution of query operators are needed. For a more detailed discussion on these issues, we refer to [10], which also describes algorithms for execution of the operators presented in this paper.

Future work includes developing techniques for further reducing the cost of executing the query operators. The main goal in this context would be to develop techniques that can reduce the number of delta versions that have to be retrieved. Two important strategies for achieving this goal is to develop new types of indexes and algebraic rewriting techniques.

Acknowledgments

I would like to thank Serge Abiteboul for suggesting this topic of research, and Vincent Aguilera and Benjamin Nguyen for useful discussions and constructive comments.

8. REFERENCES

[1] V. Aguilera, S. Cluet, P. Veltri, D. Vodislav, and F. Wattez. Querying XML documents in Xyleme. Technical Report 182, Verso/INRIA, 2000.

[2] S. S. Chawathe, S. Abiteboul, and J. Widom. Representing and querying changes in semistructured data. In *Proceedings of the Fourteenth International Conference on Data Engineering*. IEEE Computer Society, 1998.

[3] S. S. Chawathe, S. Abiteboul, and J. Widom. Managing historical semistructured data. *TAPOS*, 5(3), 1999.

[4] S.-Y. Chien, V. J. Tsotras, and C. Zaniolo. A comparative study of version management schemes for XML documents (short version published at WebDB 2000). Technical Report TR-51, TimeCenter, 2000.

[5] S.-Y. Chien, V. J. Tsotras, and C. Zaniolo. Version management of XML documents: Copy-based versus edit-based schemes. In *Proceedings of the 11th International Workshop on Research Issues on Data Engineering: Document management for data intensive business and scientific applications (RIDE-DM'2001)*, 2001.

[6] F. Grandi and F. Mandreoli. The valid web: it's time to go. Technical Report TR-46, TimeCenter, 1999.

[7] F. Grandi and F. Mandreoli. The valid web: An XML/XSL infrastructure for temporal management of web documents. In *Proceedings of Advances in Information Systems, First International Conference, ADVIS 2000*, 2000.

[8] C. S. Jensen and R. T. Snodgrass. Temporal data management. *IEEE Transactions on Knowledge and Data Engineering*, 11(1), 1999.

[9] A. Marian, S. Abiteboul, G. Cobena, and L. Mignet. Change-centric management of versions in an XML warehouse. In *Proceedings of VLDB 2001*, 2001.

[10] K. Nørvåg. Algorithms for temporal query operators in XML databases. Technical Report IDI X/2001, Norwegian University of Science and Technology, 2001. Available from http://www.idi.ntnu.no/grupper/DB-grp/.

[11] B. Oliboni, E. Quintarelli, and L. Tanca. Temporal aspects of semistructured data. In *Proceeding of TIME-01*, 2001.

[12] C. Z. Shu-Yao Chien, Vassilis J. Tsotras. Efficient management of multiversion documents by object referencing. In *Proceedings of VLDB 2001*, 2001.

[13] World-Wide Web Consortium. XQuery: A query language for XML, February 2001 (most recent version available at http://www.w3.org/TR/xquery/).

[14] L. Xyleme. A dynamic warehouse for XML data of the web. *IEEE Data Engineering Bulletin,*, 24(2), 2001.