INTRODUCTION

The amount of data available in XML is rapidly increasing and at the same time the price of mass storage is rapidly decreasing, and this makes it possible to store larger amounts of data. The contents of a database or data warehouse are seldom static. New documents are created, documents are deleted and, more important, documents are updated. In many cases, one wants to be able to search in historical (old) versions, retrieve documents that were valid at a certain time, query changes to documents, and so forth. (Note that although this process is somewhat similar to general document versioning maintenance, the aspect of time makes possibilities and appropriate solutions different.) The “easiest” way to do 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 (an example of such a system is TeXOR; Nørvåg, Limstrand, & Myklebust, 2003). 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 high. Thus, a temporal XML database system is necessary.

In our group we have developed several database systems suitable for storing XML data, including V2 (Nørvåg, 2003b). An important issue is efficient query processing. To achieve this, appropriate query operators are needed. In this article, we describe the query operators necessary to execute such queries.

BACKGROUND

Related Work

A model for representing changes in semistructured data (i.e., DOEM) and a language for querying changes (i.e., Chorel) were presented by Chawathe, Abiteboul, and Widom (1998, 1999). Chorel queries are translated to Lorel (a language for querying semistructured data) and can therefore be viewed as a stratum approach.

An approach that is orthogonal, but still related to the work presented in this article, is to introduce valid time features into XML documents, as presented by Grandi and Mandreoli (1999, 2000).

Many algorithms for execution of temporal query operators are based on the availability of temporal full-text indexes (Nørvåg, 2002) that can support temporal text-containment queries. A number of such indexes have been proposed (e.g., see Nørvåg, 2004, 2003a).

Identity of Elements in Versioned XML Documents

An important issue that poses some additional difficulties in the context of XML and, in particular, in the context of XML documents retrieved from the Web (such as from an XML data warehouse), is identity of elements.

XML documents have a quasi-persistent identifier: the URL (quasi-persistent because documents on the Web frequently are moved). However, in general the elements of a document do not have an identity of their own that persists from one version of a document to the next, which implies that many queries can be difficult to express as well as expensive to execute. Two simple examples are (a) a query for the create-time of elements, and (b) a query asking for the previous version of a certain element. Thus, although elements seen from “the outside” do not have persistent identifiers, the storage system should support this feature to make it a part of the data model for the query language. One particular system that provides such functionality is Xyleme (Marian, Abiteboul, Cobena, & Mignet, 2001). 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. The element ID (EID) is the concatenation of document ID and XID. Thus, an EID uniquely identifies a particular element in a particular document.

In a temporal XML database there will, in general, be more than one version of each element (i.e., 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. The identifier of a particular version of an element is the temporal EID (TEID) (i.e., the concatenation of EID and timestamp).
Assumed Data Model

Documents in the database are often viewed as a forest of trees. Queries for certain versions of a document (or several documents) are similar to queries for a general set of XML documents, which can also be view as a forest of trees, or the forest of trees resulting from prefiltering (i.e., returning subtrees of documents, possibly more than one tree for each document).

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 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 Chawathe et al. (1999). 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. Instead, previous versions are stored as, for example, 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.

Examples of Temporal XML Queries

The main purpose of this section is to describe the kinds of queries that can be expected in a temporal XML database (note that the query operators and associated algorithms are in general independent of which query language is actually used). The query language is based on a mix of Lorel, the Xyleme query language (Aguilera, Cluet, Veltri, Vodislav, & Wattez, 2000), and elements of XPath and XQuery (World Wide Web Consortium, 2001). Note that even if Xyleme has support for historical versions or deltas, there is no special support for them in the query language. For example, a query returning all restaurants with prices less than $10 could be written as:

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

Assume that the results of an “outer query” are delivered as default in a document with enclosing tags named <results>. Each result 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 a // operator). However, when querying semistructured data such as XML, many queries can be expected to contain the // operator.

To retrieve documents that are valid at a particular time (i.e., 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
```

For more complex queries, or when more than one version to be selected is wanted, use the keyword EVERY instead of timestamp. For example, to retrieve the price history of the restaurant named Napoli:
Related Content

Information Search Patterns in E-Commerce Product Comparison Services
Fiona Fui-Hoon Nah, Weiyin Hong, Liqiang Chen and Hong-Hee Lee (2010). Journal of Database Management (pp. 26-40).
www.igi-global.com/article/information-search-patterns-commerce-product/42084?camid=4v1a

Rewriting and Efficient Computation of Bound Disjunctive Datalog Queries
www.igi-global.com/chapter/rewriting-efficient-computation-bound-disjunctive/11205?camid=4v1a

Visual Support for Use Case Modeling: An Experiment to Determine the Effectiveness of Use Case Diagrams
www.igi-global.com/chapter/visual-support-use-case-modeling/52949?camid=4v1a

Semantic Integration and Knowledge Discovery for Environmental Research
www.igi-global.com/article/semantic-integration-knowledge-discovery-environmental/3366?camid=4v1a