Preserving and Querying Histories of XML-Published Relational Databases

Fusheng Wang and Carlo Zaniolo

Department of Computer Science, University of California, Los Angeles
Los Angeles, CA 90095, USA
{wangfsh, zaniolo}@cs.ucla.edu

Abstract. There is much current interest in publishing and viewing database-resident data as XML documents. In fact, such XML views of the database can be easily visualized on web browsers and processed by web languages, including powerful query languages such as XQuery. As the database is updated, its external XML view also evolves. In this paper, we investigate the problem of representing the evolution history of such a view as yet another XML document, whereby the complete history of the database can also be visualized on web browsers, processed by web languages, and queried using powerful query languages such as XQuery. We investigate various approaches used for publishing relational data, and identify and select those which are best for representing and querying database histories. We show that the selected representations make it easy to formulate in XQuery temporal queries that are difficult to express using SQL on database relations. Finally, we discuss briefly the storage organization that can be used to support these queries efficiently.

1 Introduction

There is a much current interest in publishing database-resident data as (concrete or dynamic) XML documents, which can then be viewed on web browsers, and processed by various web-based applications, including queries written in languages such as XPath and XQuery [4]. As the underlying database is updated, its external XML view also changes (continuously for dynamic documents and at refresh time for concrete ones). Most users who are interested in viewing and querying the current database are also interested in viewing and querying its past snapshots and evolving history—preferably, using the same browsers and query languages. In fact, in many applications, (such as inventory control, supply chain management, surveillance, etc.) changes in the database being monitored are of critical interest. To address this need, web data warehouses have been proposed recently [25]; these detect changes in web sites of interest, preserve their past contents, and answer continuous queries for subscribing users [25]. As in the case of more traditional warehouses, changes can be monitored in two ways:

1. The site publishing the database sends to the web warehouse the log of its recent updates (either continuously or at regular intervals), or

A. Olivé et al. (Eds.): ER 2002 Ws, LNCS 2784, pp. 26–38, 2003.

2. The web warehouse downloads from the site frequent snapshots of the XML-published data, and then computes the delta between the new version and the previous one.

The second problem can be reduced to the first one, by computing the delta between the two versions and then deriving an edit script that shows how one version can be transformed into the other; algorithms to support this computation were proposed in [25,17]. Since we are dealing with XML-published relational data, the order of the tuples is immaterial and we can also use the change detection algorithm for semistructured information proposed in [9]. All these algorithms represent the deltas between the documents as edit scripts and return minimum deltas that will transform the old version into the new one. As discussed in [6], for elements that are logically identified by keys, it is semantically preferable to detect changes between elements denoted by the same key. The X-Diff algorithm proposed in [33] applies in this situation; this algorithm was in fact designed for detecting changes in unordered XML documents with keys, as in the case of our XML-published relational data. By utilizing node signatures and node XHash values, the algorithm tries to find the minimum-cost matching. The algorithm can reach a high matching accuracy, and has complexity $O(n^2)$ [33].

The additional step of computing the edit script is avoided when the publishing site communicates the changes directly to the web warehouse. Thus in the rest of the paper, we assume that the update log is given. Moreover, we will not go into details about the particular form in which the corresponding updates to the XML document are represented. While somewhat different representations have been used in the past, these differences are not significant in our study, and they are bound to disappear once a standard XML update language will emerge [30]. Moreover, the use of the database update log avoids the temporal indeterminacy problems that instead occur when the remote database is sampled at regular intervals and the edit script is reconstructed using various diff algorithms.

All the approaches previously discussed focus on the preservation and retrieval of past versions of web documents; in this paper, we instead focus on relational tables and discuss how to preserve their content and support complex historical queries via XML and XQuery. Thus, we examine alternative ways to represent the history of XML-published relational tables as XML documents, and show that some of these representations allow the expression of powerful historical queries in a natural fashion. The conceptual and practical interest of this conclusion is underscored by the fact that expressing temporal queries directly on relational databases had instead proven to be a difficult problem that required major extensions to SQL [16,35,36,26]. Thus viewing the history of relational tables as XML documents could provide an appealing venue for supporting historical queries on databases. Observe that the publication of the current database as an XML document is actually not required for representing the *database history* as an XML document, since this can be constructed directly from the update log of the database.

2 Preserving the History of Documents

Traditional schemes for version management, such as RCS [32] and SCCS [28], are widely used in applications such as software configuration control and support for cooperative work; version-control techniques have also been proposed for databases, often in the context of O-O systems and CAD applications [24]. The emergence of web information systems and many new web-based applications has generated a flurry of interest and research activities, at first focusing on semistructured information [9], and now on XML [13,25,14,6]. This interest is due to the fact that (i) traditional version management applications are now migrating to a web-based environment [3], (ii) there is an increasing realization that e-permanence must be achieved and the broken link problem must be fixed [23], and (iii) very interesting queries can now be answered (using XQuery or XPath) on the preserved history of multiversion documents.

The e-permanence problem has motivated a significant amount of previous work. In particular the Wayback machine crawls the whole web [23], preserving the past content, but without much support for queries (temporal or otherwise). Transaction-time web servers were instead proposed in [18] to archive previous versions of web resources to support transaction timeslice requests by remote browsers. As further enhancement was proposed in [19], where it was shown that the XPath data model and query language can be naturally extended to support transaction time semantics.

The problem of efficiently storing and querying the history of versioned XML documents was discussed in [12,13,25,14]. The reference-based model proposed in [13] unifies the logical representation and the physical one, but can only handle simple queries; in fact, different storage representations are needed for more complex queries[14]. An extension of the SCCS scheme [28] was recently used for representing versions of hierarchically structured documents [6]. Here, we will use a similar version scheme to represent and query the history of XML-published databases at the logical level. Since many different XML-based representations can be used for publishing [29,30] the same database tables we will also study alternative representations and determines which are most suitable for supporting temporal queries. We will also show that still different representations are needed at the physical level.

3 Publishing Relational Data History as XML Documents

Table 1 and 2 describe the history of employees and departments. These transaction-time tables are shown here for illustration and *they are not stored in the actual database*. Instead, our database only contains the evolving snapshots of these relations—e.g., a single tuple for the `employee` in the example.

Therefore, we propose to represent and preserve the evolving history of these database relations by means of the XML documents shown in Figure 1 and Figure 2. We will call these *H-documents*. Each element in a H-document is assigned two attributes *tstart* and *tend*, which represent the inclusive time-interval of the

element. The value of *tend* can be set to *now*, to denote the ever-increasing current time.

Our H-documents use a temporally grouped data model [16]. Clifford, et al. [16] show that temporally-grouped models are more natural and powerful than temporarily-ungrouped ones. Temporal groups are however difficult to support in the framework of flat relations and SQL. Thus, many approaches proposed in the past instead timestamp the tuples of relational tables. These approaches incur into several problems, including the coalescing problem [35]. TSQL2's approach [35] attempts to achieve a compromise between these two [16], and is based on an implicit temporal model, which is not without its own problems [10].

Our model supports temporal grouping by taking advantage of the richer structure of XML documents, and the expressive power of XQuery. An advantage of our approach is that powerful temporal queries can be expressed in XQuery without requiring the introduction of new constructs in the language. We next show how to express temporal projections, snapshot queries, joins and historical queries on *employees* and *departments*. These queries were tested with Quip [2] (SoftwarAG's implementation of XQuery) and can be downloaded from `http://wis.cs.ucla.edu/~wangfsh/ecdm02/`.

Table 1. The snapshot history of employees

Name	Salary	Title	Dept	DOB	Start	Stop
Bob	60000	Engineer	QA	1945-04-09	1995-01-01	1995-05-31
Bob	70000	Engineer	QA	1945-04-09	1995-06-01	1995-09-30
Bob	70000	Sr Engineer	RD	1945-04-09	1995-10-01	1996-01-31
Bob	70000	Tech Leader	RD	1945-04-09	1996-02-01	1996-12-31

Table 2. The snapshot history of departments

Name	Manager	Start	End
QA	Johnson	1994-01-01	1998-12-31
RD	Joe	1992-01-01	1996-12-31
RD	Peter	1997-01-01	1998-12-31
Sales	Frank	1993-01-01	1997-12-31

3.1 Each Table as an XML Document: Columns as Elements

A natural way of publishing relational data is to publish each table as an XML document by converting relational columns into XML elements [29]. Figure 1 shows the history of the table `employee` and Figure 2 shows the history of the `dept` table. Thus the history of each relation is published as a separate H-document.

Based on the published documents, we can specify a variety of queries in XQuery:

```
<employees tstart="1995-01-01" tend="1996-12-31">
  <employee tstart="1995-01-01" tend="1996-12-31">
    <name tstart="1995-01-01" tend="1996-12-31">Bob</name>
    <salary tstart="1995-01-01" tend="1995-05-31">60000</salary>
    <salary tstart="1995-06-01" tend="1996-12-31">70000</salary>
    <title tstart="1995-01-01" tend="1995-09-30">Engineer</title>
    <title tstart="1995-10-01" tend="1996-01-31">Sr Engineer</title>
    <title tstart="1996-02-01" tend="1996-12-31">Tech Leader</title>
    <dept tstart="1995-01-01" tend="1995-09-30">QA</dept>
    <dept tstart="1995-10-01" tend="1996-12-31">RD</dept>
    <DOB tstart="1995-01-01" tend="1996-12-31">1945-04-09</DOB>
  </employee>
</employees>
```

Fig. 1. The history of the `employee` table is published as `employees.xml`

```
<depts tstart="1992-01-01" tend="1998-12-31">
  <dept  tstart="1994-01-01" tend="1998-12-31">
      <name  tstart="1994-01-01" tend="1998-12-31">QA</name>
      <manager tstart="1994-01-01" tend="1998-12-31">Johnson</manager>
  </dept>
  <dept tstart="1991-01-01" tend="1998-12-31">
      <name tstart="1991-01-01" tend="1998-12-31">RD</name>
      <manager tstart="1991-01-01" tend="1996-12-31">Joe</manager>
      <manager tstart="1997-01-01" tend="1998-12-31">Peter</manager>
  </dept>
  <dept tstart="1993-01-01" tend="1997-12-31">
      <name tstart="1993-01-01" tend="1997-12-31">Sales</name>
      <manager tstart="1993-01-01" tend="1997-12-31">Frank</manager>
  </dept>
</depts>
```

Fig. 2. The history of the `dept` table is published as `dept.xml`

QUERY 1: Temporal projection: retrieve the salary history of employee "Bob":

```
element salary_history{
  for $s in document("employees.xml")/employees/employee
      [name="Bob"]/salary
  return $s }
```

QUERY 2: Snapshot queries: retrieve the departments on 1996-01-31:

```
for $d in document("depts.xml")/depts/dept
    [@tstart <= "1996-01-31" and @tend >= "1996-01-31"]
let $n := $d/name[@tstart<="1996-01-31" and @tend>="1996-01-31"]
let $m := $d/manager[@tstart<="1996-01-31" and @tend>="1996-01-31"]
return( element dept{$n,$m } )
```

QUERY 3: Find employees history from 1995-05-01 to 1996-04-30:

```
for $e in document("employees.xml")/employees/employee
let $ol:= overlap($e/@tstart, $e/@tend, "1995-05-01","1996-4-30")
where not (empty($ol))
return ( $e/name, $ol )
```

Here *overlap($v1s, $v1e, $v2s, $v2e)* is a user-defined function that returns an element `overlap` with overlapped interval as attributes (tstart, tend). If there is no overlap, then no element is returned which satisfies the XQuery built-in function `empty()`. The next query is a containment query:

QUERY 4: Find employee(s) who worked in the "QA" department through the history of that department:

```
for $d in document("depts.xml")/depts/dept[name="QA"]
for $e in document("employees.xml")/employees/employee[dept="QA"]
where $e/@tstart = $d/name/@tstart and $e/@tend = $d/name/@tend
return $e/name
```

QUERY 5: Find the manager of each employee:

```
for $e in document("employees.xml")/employees/employee
for $d in document("depts.xml")/depts/dept[name=$e/dept]
for $m in $d/manager
let $ol :=overlap($m/@tstart,$m/@tend,$e/@tstart,$e/@tend)
where not (empty($ol))
return ( $e/name, $m, $ol )
```

This query will join employees.xml and depts.xml by *dept*, and the *overlap()* function will return only managers that overlap with the employee with the overlapped version timestamp intervals.

QUERY 6: Find the history of employees in each dept:

```
element depts{
  for $d in document("depts.xml")/depts/dept
  return
    element dept { $d/@*, $d/*,
     element employees {
       for $e in document("employees.xml")/employees/employee
       where  $e/dept = $d/name and
           not(empty(overlap($e/@tstart, $e/@tend, $d/@tstart,$d/@tend)))
       return ($e/name, $e/dept,
           overlap($e/@tstart, $e/@tend, $d/@tstart,$d/@tend) )
     }
    }
}
```

This query will join depts and employees document and generate a hierarchical XML document grouped by dept(Figure 5).

3.2 Multiple Tables as a Single XML Document: Flat Structure

Another way to publish relational data is to publish multiple relational tables into a single XML document(Figure 3), but still with the flat structure as shown in Figure 2. Essentially there is not much difference between this approach and the previous one.

Queries on this representation are similar to those described in the last section.

```
<company tstart="1995-01-01" tend="1996-12-31">
 <employees tstart="1995-01-01" tend="1996-12-31">
    <!--  <employee>... see Fig. 1 ...</employee> ...   -->
 </employees>
 <depts tstart="1992-01-01" tend="1998-12-31">
    <!--  <dept>... see Fig. 2 ... </dept> ...  -->
 </depts>
</company>
```

Fig. 3. The history of the `employee` and `dept` tables is published as `company.xml`

3.3 Multiple Tables as an XML Document: Flat Structure with IDs

To facilitate query processing, when multiple relational tables are published as XML document, tuples can be assigned IDs, which can be referred by IDREF from other elements. For example, in Figure 4, the IDs assigned to dept element, are referred to from employee.

```
<company tstart="1995-01-01" tend="1996-12-31">
 <employees tstart="1995-01-01" tend="1996-12-31">
   <employee ID="emp1" tstart="1995-01-01" tend="1996-12-31">
    <!-- name,salary,title,DOB ... -->
    <dept IDREF="dept1" tstart="1995-01-01" tend="1995-09-30">QA</dept>
    <dept IDREF="dept2" tstart="1995-10-01" tend="1996-12-31">RD</dept>
    <DOB tstart="1995-01-01" tend="1996-12-31">1945-04-09</DOB>
   </employee>
 </employees>
 <depts tstart="1992-01-01" tend="1998-12-31">
   <dept ID="dept1"  tstart="1994-01-01" tend="1998-12-31">
     <name  tstart="1994-01-01" tend="1998-12-31">QA</name>
     <manager tstart="1994-01-01" tend="1998-12-31">Johnson</manager>
   </dept>
   <!-- more dept... -->
 </depts>
</company>
```

Fig. 4. The history of the `employee` and `dept` tables is published as `company2.xml`

This representation supports queries similar to those discussed in the previous sections, but simplifies joins:

Query 7: Retrieve the dept Bob worked on 1995-10-15:

```
return document("company2.xml")/company/employees/
                 employee[name='Bob'])/dept/@ID=>dept/name
                 [@tstart<= "1995-10-15" and @tend >="1995-10-15"]
```

3.4 Multiple Tables as a Single XML Document: Hierarchical Structure

Another approach is to generate a hierarchical XML document from multiple relational tables(Figure 5). This approach is also taken by XPERANTO [8] through grouping in XML views and SQLX [21] through extended aggregate functions.

```
<depts tstart="1991-01-01" tend="1998-12-31">
  <dept  tstart="1994-01-01" tend="1998-12-31">
      <name  tstart="1994-01-01" tend="1998-12-31">QA</name>
      <manager tstart="1994-01-01" tend="1998-12-31">Johnson</manager>
      <employees tstart="1994-01-01" tend="1998-12-31">
       <employee tstart="1995-01-01" tend="1995-09-30">
        <name tstart="1995-01-01" tend="1995-09-30">Bob</name>
        <salary tstart="1995-01-01" tend="1995-05-31">60000</salary>
        <salary tstart="1995-06-01" tend="1995-09-30">70000</salary>
        <title tstart="1995-01-01" tend="1995-09-30">Engineer</title>
        <DOB tstart="1995-01-01" tend="1995-09-30">1945-04-09</DOB>
       </employee>
    </employees>
  </dept>
  <dept tstart="1991-01-01" tend="1998-12-31">
    <name tstart="1991-01-01" tend="1998-12-31">RD</name>
    <manager tstart="1991-01-01" tend="1996-12-31">Joe</manager>
    <manager tstart="1997-01-01" tend="1998-12-31">Peter</manager>
    <employees tstart="1991-01-01" tend="1998-12-31">
       <employee tstart="1995-10-01" tend="1996-12-31">
        <name tstart="1995-01-01" tend="1996-12-31">Bob</name>
        <salary tstart="1995-10-01" tend="1996-12-31">70000</salary>
        <title tstart="1995-10-01" tend="1996-01-31">Sr Engineer</title>
        <title tstart="1996-02-01" tend="1996-12-31">Tech Leader</title>
        <DOB tstart="1995-10-01" tend="1996-12-31">1945-04-09</DOB>
       </employee>
    </employees>
  </dept>
  <!-- ... -->
</depts>
```

Fig. 5. The history of `employee` and `dept` is published as `depts3.xml`

This approach simplifies some queries but complicates others. For example, if we want to retrieve employees in each department (containment query), we can simply write:

Query 8: Find employee(s) who worked in the QA department through the dept's history:

```
for $d in document("depts3.xml")/depts/dept[name='QA']
let $e := $d/employees/employee
```

```
let $e_all := document("depts3.xml")/depts/dept
              /employees/employee[name=$e/name]
where  count ($e_all) = 1
       and $e/@tstart = $d/name/@tstart and $e/@tend = $d/name/@tend
return $e/name
```

However, coalescing is needed for other queries in the hierarchical representation.
QUERY 9: Find the salary history of employee "Bob" in the company:

```
for $s in document("depts3.xml")/depts/dept/
           employees/employee[name='Bob']/salary
return coalesce($s)
```

Here we rely on a user-defined function *coalesce()* (Figure 6) to coalesce the employees. This function can also be defined in standard XQuery, as follows:

```
define function coalesce(xs:AnyType $e)  returns xs:AnyType {
if (count($e) =1) then $e
else
   if( $e[1]/text() != coalesce(subsequence($e,2)) [1]/text() )
   then ($e[1], coalesce(subsequence($e,2)) )
   else
     if( string($e[1]/@tend) <
         string(coalesce( subsequence($e,2) )[1]/@tstart)  )
     then $e
     else ( element {name($e[1]) }
            {$e[1]/@tstart, coalesce( subsequence($e,2)[1]/@tend ),
             $e[1]/text()},
             subsequence( coalesce( subsequence($e,2) ), 2) ) }
```

Fig. 6. A coalescing function defined in XQuery

3.5 Relational Tables as XML Document: Columns as Attributes

A relational table can also be published as XML document as attributes (Figure 7), e.g., the *FOR XML* statement in Microsoft SQL Server 2000 [7]. The published XML document is essentially a flat structure that corresponds to the tuple snapshots.

This approach is similar to that of timestamping the whole tuple in the relation. Temporal queries tend to be more complex and most queries require coalescing. Thus, in general, we recommend against this approach when publishing the history of relational tables.

In summary, XML representations that map columns as elements are preferable, and hierarchical representation can only be justified for special cases.

```
<employees>
<employee name="Bob" salary="60000" title="Engineer" dept="QA"
     DOB="1945-04-09" tstart="1995-01-01" tend="1995-05-31"/>
 <employee name="Bob" salary="70000" title="Engineer" dept="QA"
     DOB="1945-04-09" tstart="1995-06-01" tend="1995-09-30"/>
 <employee name="Bob" salary="70000" title=" Sr Engineer" dept="RD"
     DOB="1945-04-09" tstart="1995-10-01" tend="1996-01-31"/>
 <employee name="Bob" salary="70000" title="Tech Leader" dept="RD"
     DOB="1945-04-09" tstart="1996-02-01" tend="1996-12-31"/>
</employees>
```

Fig. 7. History of `employee` published as `employees2.xml` by mapping the table columns into attributes

4 Efficient Implementation

In the previous sections, we have shown how it is possible to preserve the history of XML-published data as XML documents, and to express complex queries on such documents using XQuery. However, the design of an efficient archival and querying system for such documents present many difficult challenges, due to the need to satisfy multiple competing performance requirements. In fact the design must achieve good performance on

- storage utilization,
- maintaining the archive (i.e. storing the latest changes),
- querying the archive (e.g., to reconstruct past snapshots of a database table, or the salary history of an employee).

For instance, the approach based on the SCSS [28] and recently used in [6] incurs in excessive costs when retrieving a snapshot of a database table—as needed to, e.g., support a query such as 'find the count of employees in each department on 1999-01-01'. In fact, in the SCCS storage scheme, the successive elements of the snapshot table tend to be scattered uniformly throughout the document history. Thus retrieval of a snapshot normally requires reading the whole document history. When the number of pages in the snapshot grows larger than the number of elements in the document, a temporal index can be used to identify which pages contain elements for a given snapshot. Even so, the number of page reads can be equal to the number of document elements, whereas page reads can be significantly reduced using temporal clustering schemes such as those proposed in [13,14].

In the archival scheme used in RCS [32], the changes to the document are appended at the end of the current history. Thus the cost of maintaining the archive is minimal with this scheme; the reconstruction of a snapshot, however, can require the traversal of the whole document history. This situation can be greatly improved (at the cost of some additional storage) with the usefulness based clustering approach discussed in [11,14], which is briefly discussed next.

Usefulness-Based Clustering. The usefulness-based clustering scheme (UBCC) [11,14] clusters the objects of a version into a new page if the percentage of valid objects in a page(i.e., its usefulness) falls below a threshold. When a page's usefulness falls below a minimum, all the valid records in that page are copied to a new page. Since the records for a given version are clustered, reconstructing the document at a version only requires to retrieve the pages that were useful at that version [15]. The usefulness-based clustering techniques can also play an important role in managing XML-published database histories.

Document Shredding. This technique is often used to manage efficiently XML documents stored in relational databases. Basically, the original XML document is decomposed into pieces that because of their more regular and simpler structure can be efficiently supported with current database engines. Each document piece is identified by an unique ID that then facilitates the reconstruction of the original document through various joins and outer-joins queries [31,14]. A natural way to shred XML published documents, is to decompose them along the attribute of the original relation—thus, e.g. the history of the employee relation might be shredded into a salary table, a position table, and a department table. No special new ID is here needed, since the relation key or the tuple ID can be used in this role.

Support for Complex Queries. Efficient indexing schemes [15] and query processing algorithms [14] can be used to support complex queries on multiversion documents. For instance, multiversion B-Trees (MVBT) [5] indexing is used to support complex queries. The MVBT is a directed graph with multiple roots, and each root is associated with a consecutive version interval.

Finally, while complex operators such as coalesce can be expressed directly in XQuery, much faster execution can be achieved by their direct implementation as built-in primitives.

5 Conclusions

In this paper, we have shown that XML-based representations and query languages provide effective ways for representing and querying the database history. In particular, we have concentrated on a situation where relational data is published using XML: we have shown that the history of the database can be represented as an XML document and queried using XQuery. The resulting XML representation is quite natural, and similar to the temporally grouped data models that were proposed in the past as the most natural approach to dealing with historical data [16]—but one that is difficult to realize in the context of the flat relational data model. In this paper, we studied alternative XML representations and identify those that best support temporal queries. We have shown that XQuery without any modification can express complex temporal queries on such representations. We have briefly discussed the physical representations and indices that are needed to ensure a more efficient execution of these queries.

The historical representations and queries discussed here find applications in data warehouses that archive and collect data from sites of interest to assure the

e-permanence [1] of critical information and support complex queries on changes [25]. Efficient support for archiving warehouse data is already supported in some commercial systems, and various techniques have been proposed for supporting complex queries on such historical data warehouses [34,27]. Many of the problems that considered in this paper are similar to those that occur in the context of transaction-time web servers and XPath extensions along the transaction time axis [18,19]. An integration of the web-server and web-warehouse functions on the historical axis is possible and desirable and represents an interesting topic for future investigations.

In this paper, we have focused on how to preserve through XML the change history of the database. But similar representations and queries could, respectively, be used to capture valid-time information in XML documents, and to support temporal queries on such documents. This is a very interesting problem [22] that can be expected to become the focus of much future research. Various techniques developed in the valid-time context [20] can also be effective for dealing with the temporal indeterminacy problems that occurs in warehouses that periodically crawl remote web sites.

References

1. National Archives of Australias policy statement Archiving Web Resources: A Policy for Keeping Records of Web-based Activity in the Commonwealth Government. `http://www.naa.gov.au/recordkeeping`.
2. Software AG's XQuery prototype Quip. `http://www.softwareag.com/tamino`.
3. WebDAV, WWW Distributed Authoring and Versioning. `http://www.ietf.org/html.charters/webdav-charter.html`.
4. XQuery 1.0: An XML Query Language. `http://www.w3.org/TR/xquery/`.
5. Becker, B., Gschwind, S., Ohler, T., Seeger, B., Widmayer, P.: On Optimal Multiversion Access Structures. Proc. of Symposium on Large Spatial Databases, Vol 692 (1993) 123–141.
6. Buneman, P., Khanna, S., Ajima, K., Tan, W.: Archiving Scientific Data. Proc. ACM SIGMOD (2002).
7. Burke, P.J., et. al.: Professional Microsoft SQL Server 2000 XML. Wrox Press (2001).
8. Carey, M., Kiernan, J., Shanmugasundaram, J., et al.: XPERANTO: A Middleware for Publishing Object-Relational Data as XML Documents. VLDB (2000).
9. Chawathe, S., Rajaraman, A., Garcia-Molina, H., Widom, J.: Change Detection in Hierarchically Structured Information. Proc. ACM SIGMOD (1996).
10. Chen, C.X., Zaniolo, C.: Universal Temporal Extensions for Database Languages. ICDE (1999) 428-437.
11. Chien, S.Y., Tsotras, V.J., and Zaniolo, C.: Version Management of XML Documents. WebDB 2000 Workshop, Dallas, TX (2000) 75-80.
12. Chien, S.Y., Tsotras, V.J., and Zaniolo,C.: Copy-Based versus Edit-Base Version Management Schemes for Structured Documents. 11th RIDE Workshop (2001)
13. Chien, S.Y., Tsotras, V.J., and Zaniolo,C.: Efficient Management of Multiversion Documents by Object Referencing. Proc. VLDB (2001).
14. Chien, S.Y., Tsotras, V.J., Zaniolo, C., and Zhang, D.: Efficient Complex Query Support for Multiversion XML Documents. EDBT (2002).

15. Chien, S.Y., Tsotras, V.J., Zaniolo, C., and Zhang, D.: Storing and Querying Multiversion XML Documents using Durable Node Numbers. WISE (2001).
16. Clifford, J., Croker, A., Grandi, F., and Tuzhilin, A.: On Temporal Grouping. Proc. of the Intl. Workshop on Temporal Databases (1995).
17. Cobena, G., Abiteboul, S., Marian, A.: Detecting Changes in XML Documents. Proc. ICDE (2002).
18. Dyreson, C.: Towards a Temporal World-Wide Web: A Transaction Time Web Server Proc. of the Australian Database Conf. (2001).
19. Dyreson, C.: Observing Transaction-time Semantics with TTXPath. WISE (2001).
20. Dyreson, C.E., Snodgrass, R.T.: Supporting Valid-Time Indeterminacy. TODS 23(1) (1998) 1–57
21. Eisenberg, A., Melton, J.: SQL/XML and the SQLX Informal Group of Companies. `http://www.sqlx.org`.
22. Grandi, F., Mandreoli, F.: The Valid Web: an XML/XSL Infrastructure for Temporal Management of Web Documents. Proc. of ADVIS (2000).
23. Kahle, B., Alexa et al.:The Internet Archive–The Wayback Machine–Surf the Web as it was. `http://www.archive.org/index.html`.
24. Katz, R.H., Chang, E.: Managing Change in Computer-Aided Design Databases. Proc. of VLDB (1987).
25. Marian, A., et al.: Change-centric management of versions in an XML warehouse. Proc. of VLDB (2001).
26. Ozsoyoglu, G., and Snodgrass, R.T.: Temporal and Real-Time Databases: A Survey. IEEE Trans. on Knowledge and Data Engineering, 7(4) (1995) 513–532.
27. Papadias, D., Tao, Y., Kalnis, P., Zhang, J.: Indexing Spatio-Temporal Data Warehouses. ICDE (2002).
28. Rochkind, M.J.: The Source Code Control System. IEEE Trans. on Software Engineering, SE-1, 4 (1975) 364–370.
29. Shanmugasundaram, J., et al.: Efficiently Publishing Relational Data as XML Documents. Proc. of VLDB (2000) 65–76.
30. Tatarinov, I., Ives, Z.G., Halevy, A.Y., Weld, D.S.: Updating XML. Proc. of SIGMOD (2001).
31. Tian, F., DeWitt, D. J., Chen, J., and Zhang, C.: The Design and Performance Evaluation of Various XML Storage Strategies. `http://www.cs.wisc.edu/niagara/Publications.html`.
32. Tichy, W.F: RCS–A System for Version Control. Software–Practice&Experience 15, 7 (1985) 637–654.
33. Wang, Y., DeWitt, D.J., and Cai, J.: X-Diff: A Fast Change Detection Algorithm for XML Documents. ICDE (2003).
34. Yang, J.: Temporal Data Warehousing. Ph.D. Dissertation, Stanford University (2001)
35. Zaniolo, C., Ceri, S., Faloutsos, C., Snodgrass, R.T., Subrahmanian, V.S., and Zicari, R.: Advanced Database Systems. Morgan Kaufmann Publishers, (1997) 97–160.
36. The TSQL2 Language Design Committee: TSQL2 Language Specification, ACM SIGMOD Record, 23(1), (1994) 65–86.