XBiT: An XML-based Bitemporal Data Model

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. Past research work on modeling and managing temporal information has, so far, failed to elicit support in commercial database systems. The increasing popularity of XML offers a unique opportunity to change this situation, inasmuch as XML and XQuery support temporal information much better than relational tables and SQL. This is the important conclusion claimed in this paper where we show that valid-time, transaction-time, and bitemporal databases can be naturally viewed in XML using temporally-grouped data models. Then, we show that complex historical queries, that would be very difficult to express in SQL on relational tables, can now be easily expressed in standard XQuery on such XML-based representations. We first discuss the management of transaction-time and valid-time histories and then extend our approach to bitemporal histories. The approach can be generalized naturally to support the temporal management of arbitrary XML documents and queries on their version history.

1 Introduction

While users' demand for temporal database applications is only increasing with time [1], database vendors are not moving forward in supporting the management and querying of temporal information. Given the remarkable research efforts that have been spent on these problems [2], the lack of viable solutions must be attributed, at least in part, to the technical difficulties of introducing temporal extensions into the relational data model and query languages.

In the meantime, database researchers, vendors and SQL standardization groups are working feverishly to extend SQL with XML publishing capabilities [4] and to support languages such as XQuery [5] on the XML-published views of the relational database [6]. In this context, XML and XQuery can respectively be viewed as a new powerful data model and query language, thus inviting the natural question on whether they can provide a better basis for representing and querying temporal database information. In this paper, we answer this critical question by showing that transaction-time, valid-time and bitemporal database histories can be effectively represented in XML and queried using XQuery without requiring extensions of current standards. This breakthrough over the relational data model and query languages is made possible by (i) the ability of XML to support a temporally grouped model, which is long-recognized as natural and expressive [7, 8] but could not be implemented well in the flat

structure of the relational data model [9], and (ii) the greater expressive power and native extensibility of XQuery (which is Turing-complete [10]) over SQL. Furthermore, these benefits are not restricted to XML-published databases; indeed these temporal representations and queries can be naturally extended to arbitrary XML documents, and used, e.g., to support temporal extensions for database systems featuring native support for XML and XQuery, and in preserving the version history of XML documents, in archives [11] and web warehouses [12].

In this paper, we build and extend techniques described in previous papers. In particular, support for transaction time was discussed in [13], and techniques for managing document versions were discussed in [12]. However, the focus of this paper is supporting valid-time and bitemporal databases, which pose new complexity and were not discussed in previous papers.

The paper is organized as follows. After a discussion of related work in the next section, we study an example of temporal relations modeled with a temporal ER model. In Section 4 we show that the valid time history of relational database history can be represented as XML, and queried with XQuery. Section 5 briefly reviews how to model transaction-time history with XML. In Section 6, we focus on an XML-based bitemporal data model to represent the bitemporal relational database history, and show that complex bitemporal queries can be expressed with XQuery based on this model, and database update can also be supported. Section 7 concludes the paper.

2 Related Work

Temporal ER Modeling There has been much interesting work on ER-based temporal modeling of information systems at the conceptual level. For instance, ER models have been supported in commercial products for database schema designs, and more than 10 temporal enhanced ER models have been proposed in the research community [14]. As discussed in the survey by Gregersen and Jensen [14], there are two major approaches of extensions to ER model for temporal support, devising new notational shorthands, or altering the semantics of the current ER model constructs. The recent TIMEER model [15] is based on an ontological foundation and supports an array of properties. Among the temporal ER models, the Temporal EER Model (TEER) [16] extends the temporal semantics into the existing EER modeling constructs.

Temporal Databases A body of previous work on temporal data models and query languages include [17–20]; thus the design space for the relational data model has been exhaustively explored [2, 21]. Clifford et al. [9] classified them as two main categories: *temporally ungrouped* and *temporally grouped* data models. Temporally grouped data model is also referred to as non-first-normal-form model or attribute time stamping, in which the domain of each attribute is extended to include the temporal dimension [8], e.g., Gadia's temporal data model [22]. It is shown that the temporally grouped representation has more expressive power and is more natural since it is history-oriented [9]. TSQL2 [23]

tries to reconcile the two approaches [9] within the severe limitations of the relational tables. Our approach is based on a temporally grouped data model, which dovetails perfectly with the hierarchical structure of XML documents.

The lack of temporal support in commercial DBMS can be attributed to the limitations of SQL, the engineering complexity, and the difficulty to implement it incrementally [24].

Publishing Relational Databases in XML There is much current interest in publishing relational databases in XML. A middleware-based approach is used in SilkRoute [25] and XPERANTO [6]. For instance, XPERANTO can build a default view on the whole relational database, and new XML views and queries upon XML views can then be defined using XQuery. XQuery statements are then translated into SQL and executed on the RDBMS engine. SQL/XML is emerging as a new SQL standard supported by several DBMS vendors [4, 26], to extend RDBMS with XML support.

Time in XML Some interesting research work has recently focused on the problem of representing historical information in XML. In [27] an annotation-based object model is proposed to manage historical semistructured data, and a special Chorel language is used to query changes. In [28] a new <**valid**> markup tag for XML/HTML documents is proposed to support valid time on the Web, thus temporal visualization can be implemented on web browsers with XSL. In [29], a dimension-based method is proposed to manage changes in XML documents, however how to support queries is not discussed.

In [30], a data model is proposed for temporal XML documents. However, since a valid interval is represented as a mixed string, queries have to be supported by extending DOM APIs or XPath. Similarly, in [31, 32], extensions of XPath is needed to support temporal semantics. (In our approach, we instead support XPath/XQuery without any extension to XML data models or query languages.) A τXQuery language is proposed in [33] to extend XQuery for temporal support, which has to provide new constructs for the language.

An archiving technique for scientific data using XML was presented in [34], but the issue of temporal queries was not discussed. Both the schema proposed in [34] and our schema are generalizations of SCCS [35].

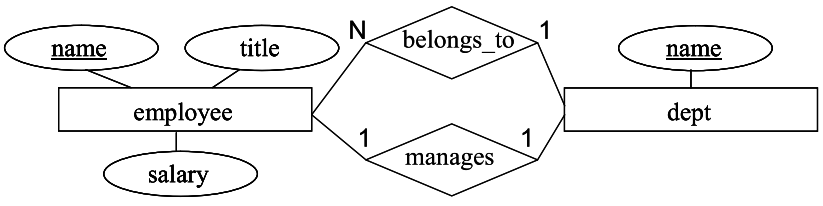

Fig. 1. TEER Schema of Employees and Departments (with Time Semantics Added)

3 An Example

The Temporal EER Model (TEER) [16] extends the temporal semantics into the existing EER modeling constructs, and works for both valid time and transaction time. TEER model associates each entity with a lifespan, and an attribute's value history is grouped together, and assigned with a temporal element (a union of valid temporal spans). Each relationship instance is also associated with a temporal element to represent the lifespan.

This temporal ER model is believed by the authors to be more natural to manage temporal aspects of data than in a tuple-oriented relational data model [16]. Suppose that we have two relations **employees** and **departments**, and each employee has a name, title, salary, and dept (name is the key), and each dept has a name and manager(name is the key). To model the history of the two relations, we use a TEER diagram as shown in Figure 1. (For simplicity, only valid time is considered, and transaction time can be modeled in a similar way.) Figure 1 looks exactly like a normal ER diagram except that the time semantics is added.

In this schema, the entity **employee** (or **e**) will have the following temporal attribute values:

```
SURROGATE(e) = {[1995-01-01,now] -> surrogate_id}
NAME(e)   = {[1995-01-01,now] -> Bob}
TITLE(e)  = {[1995-01-01,1997-12-31] -> Engineer,
             [1998-01-01,now] -> {Sr Engineer} }
SALARY(e) = {[1995-01-01,1997-12-31]-> 65000,
             [1998-01-01,1999-12-31]-> 70000,
             [2000-01-01,now] -> 85000}
```

Here each attribute value is associated with a valid time lifespan. **surrogate** is a system-defined identifier, which can be ignored if the key doesn't change.

The following is the list of temporal attribute values of entity **dept** (or **d**) :

```
SURROGATE(d) = {[1995-01-01,now] -> surrogate_id}
NAME(d)      = {[1995-01-01,now] -> RD}
```

Similarly, for the instance **rb** of the relationship **belongs_to** between employee 'Bob' and dept 'RD', the lifespan is **T(rb)=[1995-01-01,now]**, and for the instance **rm** of the relationship **manages** between employee 'Mike' and dept 'RD', the lifespan is **T(r)=[1999-01-01,now]**.

In the next section, we show that such temporal ER model can be supported well with XML.

4 Valid Time History in XML

While transaction time identifies when data was recorded in the database, valid time concerns when a fact was true in reality. One major difference is that while transaction time is appended only and cannot be updated, valid time can be updated by users. We show that, with XML, we can model the valid time history naturally.

Name	Title	Dept	Salary	Start	End
Bob	Engineer	RD	65000	1995-01-01	1997-12-31
Bob	Sr Engineer	RD	70000	1998-01-01	1999-12-31
Bob	Sr Engineer	RD	85000	2000-01-01	now

Fig. 2. Valid Time History of Employees

Figure 2 shows a valid time history of employees, where each tuple is timestamped with a valid time interval. This representation assumes valid time homogeneity, and is temporally ungrouped [9]. It has several drawbacks: first, redundancy information is preserved between tuples, e.g., Bob's department appeared the same but was stored in all the tuples; second, temporal queries need to frequently coalesce tuples, which is a source of complications in temporal query languages.

These problems can be overcome using a representation where the timestamped history of each attribute is grouped under the attribute [9]. This produces a hierarchical organization that can be naturally represented by the hierarchical XML view shown in Figure 3 (VH-document). Observe that every element is timestamped using two XML attributes **`vstart`** and **`vend`**.

```
<employees vstart="1995-01-01" vend="now">
  <employee vstart="1995-01-01" vend="now">
    <name vstart="1995-01-01" vend="now">Bob</name>
    <title vstart="1995-01-01" vend="1997-12-31">Engineer</title>
    <title vstart="1998-01-01" vend="now">Sr Engineer</title>
    <dept vstart="1995-01-01" vend="now">RD</dept>
    <salary vstart="1995-01-01" vend="1997-12-31">65000</salary>
    <salary vstart="1998-01-01" vend="1999-12-31">70000</salary>
    <salary vstart="2000-01-01" vend="now">85000</salary>
  </employee>
</employees>
```

Fig. 3. XML Representation of the Valid-time History of Employees(VH-document)

In the VH-document, each element is timestamped with an inclusive valid time interval (vstart, vend). **`vend`** can be set to *now* to denote the ever-increasing current date, which is internally represented as "9999-12-31"(Section 4.2). Please note that an entity (e.g., employee 'Bob') always has a longer or equal lifespan than its children, thus there is a *valid time covering constraint* that the valid time interval of a parent node always covers that of its child nodes, which is preserved in the update process(Section 4.3).

Unlike the relational data model that is almost invariably depicted via tables, XML is not directly associated with a graphical representation. This creates the challenge and the opportunity of devising the graphical representation most conducive for the application at hand—and implementing it using standard XML

Name	Title	Dept	Salary
1995-01-01	1995-01-01	1995-01-01	1995-01-01
	Engineer		**65000**
	1997-12-31		1997-12-31
	1998-01-01		1998-01-01
Bob		**RD**	**70000**
	Sr Engineer		1999-12-31
			2000-01-01
			85000
now	now	now	now

Fig. 4. Temporally Grouped Valid Time History of Employees

tools such as XSL [36]. Figure 4 shows a representation of temporally grouped tables that we found effective as user interface (and even more so after contrasting colored backgrounds and other browser-supported embellishments).

4.1 Valid Time Temporal Queries

The data shown in Figure 4 is the actual data stored in the database—with the exception of the special "now" symbol discussed later. Thus a powerful query language such as XQuery can be directly applied to this data model. In terms of data types, XML and XQuery support an adequate set of built-in temporal types, including datetime, date, time, and duration [5]; they also provide a complete set of comparison and casting functions for duration, date and time values, making snapshot and period-based queries convenient to express in XQuery. Furthermore, whenever more complex temporal functions are needed, they can be defined using XQuery functions that provide a native extensibility mechanism for the language.

Next we show that we can specify temporal queries with XQuery on the VH-document, such as temporal projection, snapshot queries, temporal slicing, temporal joins, etc.

QUERY V1: Temporal projection: retrieve the history of departments where Bob was employed:

```
<dept>
 for $s in doc("emps.xml")/employees/employee[name="Bob"]/dept
 return $s
</dept>
```

QUERY V2: Snapshot: retrieve the managers of each department on 1999-05-01:

```
for $m in doc("depts.xml")/depts/
   dept/mgrno[vstart(.)<="1999-05-01" and vend(.)>="1999-05-01"]
return $m
```

Here **`depts.xml`** is the VH-document that includes the history of dept names and managers. **`vstart()`** and **`vend()`** are user-defined functions (expressed in

XQuery) that return the starting date and ending date of an element's valid time respectively, thus the implementation is transparent to users.

QUERY V3: Continuous Period: find employees who worked as a manager for more than 5 consecutive years (i.e., 1826 days):

```
for $e in doc("emps.xml")/employees/employee[title="Manager"]
for $t in $e/title[.="Manager"]
let $duration := subtract-dates( vend($t), vstart($t)  )
where dayTimeDuration-greater-than($duration,"P1826D")
return  $e/name
```

Here "P1826D" is a duration constant of 1826 days in XQuery.

QUERY V4: Temporal Join: find employees who were making the same salaries on 2001-04-01:

```
for $e1 in doc("emps.xml")/employees/employee
for $e2 in doc("emps.xml")/employees/employee
where $e1/salary[vstart(.)<='2001-04-01'
  and vend(.)>= '2001-04-01'] =
  $e2/salary[vstart(.)<= '2001-04-01' and vend(.)>='2001-04-01']
  and $e1/name != $e2/name
return ($e1/name , $e2/name)
```

This query will join **emps.xml** with itself. It is also easy to support *since* and *until* connectives of first-order temporal logic [18], for example:

QUERY V5: A Until B: find the employee who was hired and worked in dept "RD" until Bob was appointed as the manager of the dept:

```
for $e in doc("emps.xml")/employees/employee
for $b in doc("emps.xml")/employees/employee[name='Bob']
let $t := $b/title[.='manager']
let $bd := $b/dept[.='RD']
let $d := $e/dept [1][.='RD']
where vmeets($d, $t) and vcontains($bd,$t)
return <employee>{$e/name}</employee>
```

4.2 Temporal Operators

In the temporal queries, we used functions such as **vstart** and **vend** to shield users from the implementations of representing time. Functions predefined include: timestamp referencing functions, such as **vstart, vend**; interval comparison functions, such as **voverlaps, vprecedes, vcontains, vequals, vmeets, voverlapinterval**; and during and date/time functions, such as **vtimespan, vinterval**. For example, **vcontains** is defined as follows:

```
define function vcontains($a, $b){
 if ($a/@vstart<= $b/@vstart and $a/@vend >= $b/@vend )
 then true()
 else false()
}
```

Internally, we use "end-of-time" values to denote the 'now' and 'UC' symbol. For instance for dates we use "9999-12-31." The user does not access this value directly, but accesses it through built-in functions. For instance, to refer to the ending valid time of a node **s**, the user uses the function **vend(s)**, which returns **s**'s end, if this is different from '9999-12-31" and CURRENT_DATE otherwise. The nodes returned in the output, normally use the "9999-12-31" representation used for internal data. However, for data returned to the end-user, two different representations are preferable. One is to return the CURRENT_DATE by applying function *rvend*() that, recursively, replaces all the occurrence of "9999-12-31" with the value of CURRENT_DATE. The other is to return a special string, such as *now* to be displayed on the end-user screen.

These valid-time queries are similar to those transaction time history, as discussed in [13]. However, unlike transaction-time databases, valid time databases must also support explicit update. This is not discussed in [13] and will be discussed next.

4.3 Database Modifications

An update task force is currently working on defining standard update constructs for XQuery [37]; moreover, update constructs are already supported in several native XML databases [38]. Our approach to temporal updates consists in supporting the operations of insert, delete, and update via user-defined functions. This approach will preserve the validity of end-user programs in the face of differences between vendors and evolving standards. It also shields the end-users from the complexity of the additional operations required by temporal updates, such as the coalescing of periods, and the propagation of updates to enforce the covering constraints.

INSERT. When a new entity is inserted, the new employee element with its children elements is appended in the VH-Document; the **vstart** attributes are set to the valid starting timestamp, and **vend** are set to *now*. Insertion can be done through the user-defined function **VInsert(**$**path,**$**newelement)**. The new element can be created using the function **VNewElement(**$**valueset,** $**vstart,** $**vend)**.

For example, the following query inserts Mike as an engineer into RD dept with salary 50K, starting immediately:

```
for $s in doc("emps.xml")/employees/employee[last()]
return VInsert($s, VNewElement(
  ["Mike", "Engineer", "RD", "50000"], current-date(),"now" ))
```

DELETE. There are two types of deletion: deletion without valid time and deletion with valid time. The former assumes a default valid time interval: (current_date, forever), and can be implemented with the user defined function **VNodeDelete(**$**path)**. For deletion with a valid time interval **v** on node **e**, there can be three mutually exclusive cases: (i) **e** is removed if its valid time interval

is contained in **v**, (ii) the valid time interval of **e** is extended if the two intervals overlap, but do not contain each other, or (iii) **e**'s interval is split if it properly contains **v**. Deletions on a node are then propagated downward to its children to satisfy the covering constraint. Node deletion (with downward propagation) is supported by the function **VTimeDelete(**$**path,** $**vstart,** $**vend)**.

UPDATE. Updates can be on values or valid time, and coalescing is needed. There are two functions defined: **VNodeReplace(**$**path,**$**newValue)**, and **VTimeReplace(**$**path,** $**vstart,**$**vend)**. For value update, propagation is not needed; for valid time update, it is needed to downward update the node's children's valid time. If a valid time update on a child node violates the valid time covering constraint, then the update will fail.

5 Viewing Transaction Time History as XML

In [13] we have proposed an approach to represent the transaction-time history of relational databases in XML using a temporally grouped data model. This approach is very effective at supporting complex temporal queries using XQuery [5], without requiring changes in this standard query language.

In [13] we used these features to show that the XML-viewed transaction time history(TH-document) can be easily generated from the evolving history of the databases, and implemented by either using native XML databases or, after decomposition into binary relations, by relational databases enhanced with tools such as SQL/XML [4]. We also showed that XQuery without modifications can be used as an effective language for expressing temporal queries.

A key issue not addressed in [13] was whether this approach, and its unique practical benefits of only requiring off-the-shelf tools, can be extended to support bitemporal databases. With two dimensions of time, bitemporal databases have much more complexity, e.g., coalescing on two dimensions, explicit update complexity, and support of more complex bitemporal queries. In the next section, we explore how to support a bitemporal data model based on XML.

6 An XML-based Bitemporal Data Model

6.1 The XBiT Data Model

In practice, temporal applications often involve both transaction time and valid time. We show next that, with XML, we can naturally represent a temporally grouped data model, and provide support for complex bitemporal queries.

Bitemporal Grouping Figure 5 shows a bitemporal history of employees, using a temporally ungrouped representation. Although valid time and transaction time are generally independent, for the sake of illustration, we assume here that employees' promotions are scheduled and entered in the database four months before they occur.

XBiT supports a temporally grouped representation by coalescing attributes' histories on both transaction time and valid time. Temporal coalescing on two

Name	Title	Dept	Salary	Valid Time	Transaction Time
Bob	Engineer	RD	65000	1995-01-01:now	1995-01-01:1997-08-31
Bob	Engineer	RD	65000	1995-01-01:1997-12-31	1997-09-01:UC
Bob	Sr Engineer	RD	70000	1998-01-01:now	1997-09-01:1999-08-31
Bob	Sr Engineer	RD	70000	1998-01-01:1999-12-31	1999-09-01:UC
Bob	Sr Engineer	RD	85000	2000-01-01:now	1999-09-01:UC

Fig. 5. Bitemporal History of Employees

temporal dimensions is different from coalescing on just one. On one dimension, coalescing is done when: i) two successive tuples are value equivalent, and ii) the intervals overlap or meet. The two intervals are then merged into maximal intervals.

For bitemporal histories, coalescing is done when two tuples are value-equivalent and (i) their valid time intervals are the same and the transaction time intervals meet or overlap; or (ii) the transaction time intervals are the same and the valid time intervals meet or overlap. This operation is repeated until no tuples satisfy these conditions.

For example, in Figure 5, to group the history of titles with value 'Sr Engineer' in the last three tuples, i.e., (title, valid_time, transaction_time), the last two transaction time intervals are the same, so they are coalesced as **(Sr Engineer, 1998-01-01:now, 1999-09-01:UC)**. This one again has the same valid time interval as the previous one: (**(Sr Engineer, 1998-01-01:now, 1997-09-01:1999-08-31)**, thus finally they are coalesced as **(Sr Engineer, 1998-01-01:now, 1997-09-01:UC)**, as shown in Figure 7.

Data Modeling of Bitemporal History with XML With temporal grouping, the bitemporal history is represented in XBiT as an XML document (BH-document). This is shown in the example of Figure 6, which is snapshot-equivalent to the example of Figure 5. Each employee entity is represented as an employee element in the BH-document, and table attributes are represented as employee element's child elements. Each element in the BH-document is assigned two pairs of attributes **tstart** and **tend** to represent the inclusive transaction time interval, and **vstart** and **vend** to represent the inclusive valid time interval. Elements corresponding to a table attribute value history are ordered by the starting transaction time **tstart**. The value of **tend** can be set to *UC* (until changed), and **vend** can be set to *now*. There is a covering constraint whereby the transaction time interval of a parent node must always cover that of its child nodes, and likewise for valid time intervals.

Figure 7 displays the resulting temporally grouped representation, which is appealing to intuition, and also effective at supporting natural language interfaces, as shown by Clifford [7].

```
<employees vstart="1995-01-01" vend="now" tstart="1995-01-01" tend="UC">
 <employee vstart="1995-01-01" vend="now" tstart="1995-01-01" tend="UC">
  <name vstart="1995-01-01" vend="now" tstart="1995-01-01" tend="UC">Bob</name>
  <title vstart="1995-01-01" vend="now" tstart="1995-01-01" tend="1997-08-31"> Engineer</title>
  <title vstart="1995-01-01" vend="1997-12-31" tstart="1997-09-01" tend="UC"> Engineer</title>
  <title vstart="1998-01-01" vend="now" tstart="1997-09-01" tend="UC">Sr Engineer</title>
  <dept vstart="1995-01-01" vend="now" tstart="1995-01-01" tend="UC">RD</dept>
  <salary vstart="1995-01-01" vend="now" tstart="1995-01-01" tend="1997-08-31">65000</salary>
  <salary vstart="1995-01-01" vend="1997-12-31" tstart="1997-09-01" tend="UC">65000</salary>
  <salary vstart="1998-01-01" vend="now" tstart="1997-09-01" tend="1999-08-31">70000</salary>
  <salary vstart="1998-01-01" vend="1999-12-31" tstart="1999-09-01" tend="UC">70000</salary>
  <salary vstart="2000-01-01" vend="now" tstart="1999-09-01" tend="UC">85000</salary>
 </employee>
</employees>
```

Fig. 6. XML Representation of the Bitemporal History of Employees(BH-document)

Name	Title	Dept	Salary
t:1995-01-01 v:1995-01-01 **Bob** t:UC v:now	v:1995-01-01 t:1995-01-01 **Engineer** v:now t:1997-08-31	t:1995-01-01 v:1995-01-01 **RD** t:UC v:now	v:1995-01-01 t:1995-01-01 **65000** v:now t:1997-08-31
	v:1995-01-01 t:1997-09-01 **Engineer** v:1997-12-31 t:UC		v:1995-01-01 t:1997-09-01 **65000** v:1997-12-31 t:UC
	v:1998-01-01 t:1997-09-01 **Sr Engineer** v:now t:UC		v:1998-01-01 t:1997-09-01 **70000** v:now t:1999-08-31
			v:1998-01-01 t:1999-09-01 **70000** v:1999-12-31 t:UC
			v:2000-01-01 t:1999-09-01 **85000** v:now t:UC

Fig. 7. Temporally Grouped Bitemporal History of Employees

6.2 Bitemporal Queries with XQuery

The XBiT-based representation can also support powerful temporal queries, expressed in XQuery without requiring the introduction of new constructs in the language. We next show how to express bitemporal queries on employees.

QUERY B1: Temporal projection: retrieve the bitemporal salary history of employee "Bob":

```
<salary_history>
 for $s in doc("emps.xml")/employees/employee[name="Bob"]/salary
 return $s
</salary_history>
```

This query is exactly the same as query V1, except that it retrieves both transaction time and valid time history of salaries.

QUERY B2: Snapshot: according to what was known on 1999-05-01, what was the average salary at that time?

```
let $s := doc("emps.xml")/employees/employee/salary
where tstart($s)<="1999-05-01" and  tend($s) >=  "1999-05-01"
  and vstart($s)<="1999-05-01" and vend($s) >=  "1999-05-01"
return  avg($s)
```

Here **tstart()**, **tend()**, **vstart()** and **vend()** are user-defined functions that get the starting date and ending date of an element's transaction-time and valid-time, respectively.

QUERY B3: Diff queries: retrieve employees whose salaries (according to our current information) didn't changed between 1999-01-01 and 2000-01-01:

```
let $s := doc("emps.xml")/employees/employee/salary
where tstart($s)<=current-date() and  tend($s)>=current-date()
  and vstart($s)<="1999-01-01" and vend($s)>= "2000-01-01"
return  $s/..
```

This query will take a transaction time snapshot and a valid time slicing of salaries.

QUERY B4: Change Detection: find all the updates of employee salaries that were applied retroactively.

```
 for  $s in doc("emps.xml")/employees/employee/salary
 where tstart($s) > vstart($s) or  tend($s) > vend($s)
```

QUERY B5: find the manager for each current employee, as best known now:

```
for $e in doc("emps.xml")/employees/employee
for $d in doc("depts.xml")/depts/dept/name[.=$e/dept]
where tend($e)="UC" and tend($d)="UC"
 and vend($e)="now" and vend($d)="now"
return $e, $d
```

This query will take the current snapshot on both transaction time and valid time.

6.3 Database Modifications

For valid time databases, both attribute values and attribute valid time can be updated by users, and XBiT must perform some implicit coalescing to support the update process. Note that only elements that are current (ending transaction time as *UC*) can be modified. A modification combines two processes: explicit modification of valid time and values, and implicit modification of transaction time.

Modifications of Transaction Time Databases Transaction time modifications can also be classified as three types: insert, delete, and update.

INSERT. When a new tuple is inserted, the corresponding new element (e.g., employee 'Bob') and its child elements in BH-document are timestamped with starting transaction time as current date, and ending transaction time as *UC*. The user-defined function **`TInsert($node)`** will insert the node with the transaction time interval(current date, *UC*).

DELETE. When a tuple is removed, the ending transaction time of the corresponding element and its current children is changed to current time. This can be done by the function **`TDelete($node)`**.

UPDATE. Update can be seen as a delete followed by an insert.

Database Modifications in XBiT Modifications in XBiT can be seen as the combination of modifications on valid time and transaction time history. XBiT will automatically coalesce on both valid time and transaction time.

INSERT. Insertion is similar to valid time database insertion except that the added element is timestamped with transaction time interval as (current date, *UC*).

This can be done by the funciton **`BInsert($path, $newelement)`**, which combines **`VInsert`** and **`TInsert`**.

DELETE. Deletion is similar to valid time database insertion, except that the function **`TDelete`** is called to change **`tend`** of the deleted element and its current children to current date. Node deletion is done through the function **`BNodeDelete($path)`**, and valid time deletion is done through the function **`BTimeDelete($path, $vstart, $vend)`**.

UPDATE. Update is also a combination of valid time and transaction time, i.e., deleting the old tuple with **`tend`** set to current date, and inserting the new tuple with new value and valid time interval, **`tstart`** set to current date and **`tend`** set to *UC*. This is done by the functions **`BNodeReplace($path, $newValue)`** and **`BTimeReplace($path, $vstart, $vend)`** respectively.

6.4 Temporal Database Implementations

Two basic approaches are possible to manage the three types of H-documents discussed here: one is to use a native XML database, and the other is to use traditional RDBMS. In [13] we show that a transaction time TH-document can be stored in a RDBMS and has significant performance advantages on temporal queries over a native XML database. Similarly, RDBMS-based approach can be applied to the valid history and bitemporal history. First, the BH-document is shredded and stored into H-tables.

For example, the employee BH-document in Figure 6 is mapped into the following attribute history tables:

```
employee_name(id,name,vstart,vend,tstart,tend)
employee_title(id,title,vstart,vend,tstart,tend)
employee_salary(id,salary,vstart,vend,tstart,tend)
...
```

Since the BH-document and H-tables have a simple mapping relationship, temporal XQuery can be translated into SQL queries based on such mapping relationship, using the techniques discussed in [13].

7 Conclusions

In this paper, we showed that valid-time, transaction-time, and bitemporal databases can be naturally managed in XML using temporally-grouped data models. This approach is similar to the one we proposed for transaction-time data bases in [13], but we have here shown that it also supports (i) the temporal EER model [16], and (ii) valid-time and bitemporal databases with the complex temporal update operations they require. Complex historical queries, and updates, which would be very difficult to express in SQL on relational tables, can now be easily expressed in XQuery on such XML-based representations.

The technique is general and can be applied to historical representations of relational data, XML documents in native XML databases, and version management in archives and web warehouses [12]. It can also be used to support schema evolution queries [39].

Acknowledgments

The authors would like to thank Xin Zhou for his help and comments.
This work was supported by the National Historical Publications and Records Commission and a gift by NCR Teradata.

References

1. R. T. Snodgrass. Developing Time-Oriented Database Applications in SQL. *Morgan Kaufmann*, 1999.
2. G. Ozsoyoglu and R.T. Snodgrass. Temporal and Real-Time Databases: A Survey. *IEEE Transactions on Knowledge and Data Engineering*, 7(4):513–532, 1995.
3. F. Grandi. An Annotated Bibliography on Temporal and Evolution Aspects in the World Wide Web. In *TimeCenter Technical Report TR-75*, 2003.
4. SQL/XML. http://www.sqlx.org.
5. XQuery 1.0: An XML Query Language. http://www.w3.org/XML/Query.
6. M. Carey, J. Kiernan, J. Shanmugasundaram, and et al. XPERANTO: A Middleware for Publishing Object-Relational Data as XML Documents. In *VLDB*, 2000.
7. J. Clifford. *Formal Semantics and Pragmatics for Natural Language Querying*. Cambridge University Press, 1990.
8. J. Clifford, A. Croker, and A. Tuzhilin. On completeness of historical relational query languages. *ACM Trans. Database Syst.*, 19(1):64–116, 1994.
9. J. Clifford, A. Croker, F. Grandi, and A. Tuzhilin. On Temporal Grouping. In *Recent Advances in Temporal Databases*, pages 194–213. Springer Verlag, 1995.
10. S. Kepser. A Proof of the Turing-Completeness of XSLT and XQuery. In *Technical report SFB 441, Eberhard Karls Universitat Tubingen*, 2002.
11. ICAP: Incorporating Change Management into Archival Processes. http://wis.cs.ucla.edu/projects/icap/.

12. F. Wang and C. Zaniolo. Temporal Queries in XML Document Archives and Web Warehouses. In *TIME-ICTL*, 2003.
13. F. Wang and C. Zaniolo. Publishing and Querying the Histories of Archived Relational Databases in XML. In *WISE*, 2003.
14. H. Gregersen and C. S. Jensen. Temporal Entity-Relationship Models - A Survey. *Knowledge and Data Engineering*, 11(3):464–497, 1999.
15. H. Gregersen and C. Jensen. Conceptual Modeling of Time-varying Information. In *TIMECENTER Technical Report TR-35, September 1998.*, 1998.
16. R. Elmasri and G.T.J.Wuu. A Temporal Model and Query Language for ER Databases. In *ICDE*, pages 76–83, 1990.
17. R. T. Snodgrass. *The TSQL2 Temporal Query Language*. Kluwer, 1995.
18. J. Chomicki, D. Toman, and M.H. Böhlen. Querying ATSQL Databases with Temporal Logic. *TODS*, 26(2):145–178, June 2001.
19. M. H. Böhlen, J. Chomicki, R. T. Snodgrass, and D. Toman. Querying TSQL2 Databases with Temporal Logic. In *EDBT*, pages 325–341, 1996.
20. J. Chomicki and D. Toman. Temporal Logic in Information Systems. In *Logics for Databases and Information Systems*, pages 31–70. Kluwer, 1998.
21. C. S. Jensen and C. E. Dyreson (eds). A Consensus Glossary of Temporal Database Concepts - February 1998 Version. *Temporal Databases: Research and Practice*, pages 367–405, 1998.
22. S. K. Gadia and C. S. Yeung. A Generalized Model for a Relational Temporal Database. In *SIGMOD*, 1988.
23. C. Zaniolo, S. Ceri, C.Faloutsos, R.T. Snodgrass, V.S. Subrahmanian, and R. Zicari. *Advanced Database Systems*. Morgan Kaufmann Publishers, 1997.
24. Adam Bosworth, Michael J. Franklin, and Christian S. Jensen. Querying the Past, the Present, and the Future. In *ICDE*, 2004.
25. M. Fernandez, W. Tan, and D. Suciu. SilkRoute: Trading Between Relations and XML. In *8th Intl. WWW Conf.*, 1999.
26. Oracle XML. http://otn.oracle.com/xml/.
27. S.S. Chawathe, S. Abiteboul, and J. Widom. Managing Historical Semistructured Data. *Theory and Practice of Object Systems*, 24(4):1–20, 1999.
28. F. Grandi and F. Mandreoli. The Valid Web: An XML/XSL Infrastructure for Temporal Management of Web Documents. In *ADVIS*, 2000.
29. M. Gergatsoulis and Y. Stavrakas. Representing Changes in XML Documents using Dimensions. In *Xsym*, 2003.
30. T. Amagasa, M. Yoshikawa, and S. Uemura. A Data Model for Temporal XML Documents. In *DEXA*, 2000.
31. C.E. Dyreson. Observing Transaction-Time Semantics with TTXPath. In *WISE*, 2001.
32. S. Zhang and C. Dyreson. Adding valid time to xpath. In *DNIS*, 2002.
33. D. Gao and R. T. Snodgrass. Temporal Slicing in the Evaluation of XML Queries. In *VLDB*, 2003.
34. P. Buneman, S. Khanna, K. Tajima, and W. Tan. Archiving scientific data. *ACM Trans. Database Syst.*, 29(1):2–42, 2004.
35. M.J. Rochkind. The Source Code Control System. *IEEE Transactions on Software Engineering*, SE-1(4):364–370, 1975.
36. The Extensible Stylesheet Language (XSL). http://www.w3.org/Style/XSL/.
37. M. Rys. Proposal for an XML Data Modification Language. In *Microsoft Report*, 2002.
38. Tamino XML Server. http://www.tamino.com.
39. F. Wang and C. Zaniolo. Representing and Querying the Evolution of Databases and their Schemas in XML. In *Intl. Workshop on Web Engineering, SEKE*, 2003.