

# Handling of Current Time in Native XML Databases

Bela Stantic<sup>1</sup>

Guido Governatori<sup>2</sup>

Abdul Sattar<sup>1</sup>

<sup>1</sup> Institute for Integrated and Intelligent Systems,  
Griffith University, Brisbane Australia

<sup>2</sup> School of Information Technology and Electrical Engineering,  
The University of Queensland, Brisbane Australia,

Email: B.Stantic@griffith.edu.au, guido@itee.uq.edu.au, A.Sattar@griffith.edu.au

## Abstract

The introduction of Native XML databases opens many research questions related to the data models used to represent and manipulate data, including temporal data, in XML. Increasing use of XML for Valid Web pages warrants an adequate treatment of *now* in Native XML databases. In this study, we examined how to represent and manipulate *now-relative* temporal data. We identify different approaches being used to represent current time in XML temporal databases, and introduce the notion of storing variables such as ‘now’ or ‘UC’ as strings in XML native databases. All approaches are empirically evaluated on a query that time-slices the timeline at the current time. The experimental results indicate that the proposed extension offers several advantages over other approaches: better semantics, less storage space and better response time.

*Keywords:* Native XML Databases, Temporal Databases, current time

## 1 Introduction

There has been a lot of research into adding time to different data models, for example to Semantic data model, Knowledge-based data model and Entity Relationship model. But most of the literature in temporal databases is related to Relational and Object-Oriented data model (Jensen 2000). A large number of temporal data models were studied and the design space for the relational data model has been exhaustively explored (Date, Darwen & Lorentzos 2002). Clifford et al. (Clifford, Croker & Tuzhilin 1994) classified them as two main categories: temporally ungrouped and temporally grouped. Although temporally grouped models have long been known to be more expressive and appealing to intuition (Clifford, Croker, Grandi & Tuzhilin 1995), they cannot be supported easily in the framework of flat relations and SQL, and therefore they have not been actually implemented in temporal database projects and prototypes (Ozsoyoglu & Snodgrass 1995).

Recently research in temporal representation and reasoning has been extended to XML. Research on adding temporal features to XML has taken into account change, versioning, evolution and also explicitly temporal aspects. Some extensions of the XML, such as  $\tau$ XQuery language, have been proposed to extend XQuery for temporal support (Chawathe, Abiteboul & Widom 1999), (Gao & Snodgrass 2003). Recently, database researchers, vendors and SQL standardization groups started work toward extensions of SQL

with XML capabilities (<http://www.sqlx.org> 2004) and to support languages such as XQuery (*XQuery 1.0: An XML query language* 2004) on XML data (Carey, Kiernan, Shanmugasundaram & et al 2000) (Funderburk, Kiernan, Shanmugasundaram, Shekita & Wei 2002). XML and XQuery can be viewed as a new powerful data model and query language providing a better basis for representing and querying temporal data. In contrast to relational databases temporally grouped data model is supported well by XML and its query languages.

Most modern database applications involve a significant amount of time dependent data and a substantial proportion of this data is *now-relative*, i.e. the end time of their validity follows the current time. *Now-relative* data are natural and meaningful part of every temporal database as well as being the focus of most queries. It has been shown that different semantics for *now* in temporal relational environment significantly influence performance (Torp, Jensen & Bohlen 1999), (Stantic, Khanna & Thornton 2004). As XML is used for Valid Web, which has temporal features and is associated with current validity of Web pages, handling *now* in XML is even more important than in relational databases. While significant research has been oriented toward adding different temporal dimensions to XML and querying such data with XQuery and on different extensions to XQuery, handling current time or *now* has received only a little attention. The majority of the proposals for simplicity consider only closed intervals. For closed intervals exact starting and ending point must be known up front. This is obviously unrealistic in real application domains. These proposals do not support data where ending time of validity follows the current time, *now-relative* data. Proposals that mentioned ‘now’ usually do so only briefly, in line with temporal relational research, recommending the *MAX* approach to represent current time or suggesting the usage of user defined functions. These recommendations are usually made without any further explanation or empirical results that show the efficiency and support the recommendation (Wang & Zaniolo 2003).

For the above reasons we decided to investigate how different semantic for *now* influence not only the performance but also the accuracy of the queries. Also, we decided to investigate whether temporally grouping of data offers any advantage and better performance than the direct conversion of relations to XML. In order to do this testing we decided to use data set generated in relational environment.

At first, we identify available options for representing *now* in XML-Temporal. Interestingly, the options ruled out as not suitable for relational databases can be considered as viable options for XML and Native XML databases. We introduce the notion of storing variables such as ‘now’ or ‘UC’ as strings in XML native databases. All approaches are empirically eval-

uated on a query that time-slices the timeline at the current time. The experimental results indicate that the proposed extension offers several advantages over other approaches: better semantics, less storage space and better response time.

The remainder of this paper is organised as follows, in the next section we look more closely at temporal dimensions of interest in XML. In Section 3 we demonstrate that any temporal data can be viewed in two different ways: the *DIRECT* representation and in temporally grouped data model; both can be represented in XML and efficiently queried by XQuery. Furthermore in this section, we discuss available methods to represent current time and highlight their limitations and disadvantages. Also, in Section 3, we illustrate the experiment taken to evaluate the identified approaches to represent current time and present the experimental results and analysis. Finally, in Section 4, we present our conclusions and suggest future work.

## 2 Temporal data and XML

A significant percentage of data for web pages is dynamic and generated as a result of queries most often in form of XML, so it is more natural and more convenient to store data directly in XML format. There are two basic methods to store XML documents in a database. The first is to map the document's schema to a database schema and transfer data according to that mapping. The second is to use a fixed set of structures that can store any XML document. Databases that support the first method are called XML-enabled databases while databases that support the second method are called native XML databases (xml.com 2005).

XML-enabled databases are useful when publishing existing data as XML or importing data from an XML document into an existing database. For instance, DB2s XML Extender (www3.ibm.com 2005) takes advantage of user-defined functions and stored procedures to map between XML data and relational data. Another approach of XML-enabled databases is a middleware based approach such as used in SilkRoute (Fernandez, Tan & Suci 2000) and XPERANTO (Carey et al. 2000). However, XML-enabled databases are not a good way to store complete XML documents. The reason is because they store data and hierarchy but discard everything else: document identity, sibling order, comments, processing instructions, and so on. This approach obviously has some limitations as XML itself is more powerful and conversion from relational model to XML is considered as straightforward, but the opposite conversion is not always possible without some ingenuity.

Native XML databases, on the other hand, store complete documents and can store any document, regardless of schema. Native XML databases are used in a wide number of fields, such as health care, genetics, insurance, data integration, messaging, Web sites, etc. The most popular of these are storing and querying document-centric XML, integrating data, and storing and querying semi-structured data. Native XML databases are used in these cases because the data involved does not easily fit the relational data model, while it does fit the XML data model.

In order to access data in XML that are valid at particular (and most often present) time, it is necessary to represent time dimensions in XML. This issue opens a big question not only for databases but also in computer science in general, i.e. how to handle and store current time or *now*. The assumption mostly taken in the literature is that only closed intervals of validity exist (Wang & Zaniolo 2003). This is obvi-

ously unrealistic; often ending points will be unknown and will follow the advancing current time indicating that, for example, fact is valid *now* and that its ending time of validity is unknown.

### 2.1 Time dimensionality of interest

Research on adding temporal features to XML has identified different time dimensions of interest. The focus of some approaches was on the representation and management of changes, where different versions of data are produced by updates. In this approach, temporal attributes are often used to timestamp stored versions (Amagasa, Yoshikawa, & Uemura 2000), (Chawathe et al. 1999) and they represent the time the updates were applied, which basically has the same semantics as transaction time. Transaction time is the time that shows the status of the data in a database, from when it is inserted in the database to when it is logically deleted, if ever. With respect to the Web, it represents the on-line availability and versioning of resources in a Web site, even if they are basically not created by *transactions*. This notion of time requires storing the current time or *now* to represent that the element is current and belong to the current database state that is not logically deleted.

Another approach, basically represents the classical notion of valid time. It is a XML/XSL infrastructure, named 'The Valid Web', designed to represent and manage temporal Web documents containing historical information (Grandi & Mandreoli 2000). In this approach timestamps are explicitly encoded by the document authors to assign validity to information content. Temporal documents can then be selectively browsed in accordance with a user-supplied temporal period of interest. This approach is further extended in (Wang & Zaniolo 2002). The valid time is the time when some fact is true in the real world. In Web applications, it concerns the temporal validity of the information carried by the contents of a Web resource. It is obvious that this notion of time also requires storing the current time or *now*. This is because it is expected to have facts that started to be valid at certain past time, they are valid *now*, and the end of their validity is unknown. The end time of validity of facts follows the current time.

There are several other temporal dimensions that have been also mentioned in the literature in relation to XML: *navigation time*, which concerns the interaction of users during their browsing of Web sites. Furthermore, a *publication time*, in the context of legal documents, and *efficiency time* (Grandi, Mandreoli, Tiberio & Bergonzini 2003). Navigation time and publication time do not require to store 'now' and are relevant for this study.

### 2.2 Representation of temporal dimensions

There are basically two different approaches to represent temporal dimensions in XML.

- to represent timestamps by XML elements,
- to represent timestamps by the temporal attributes of the XML elements.

For simplicity, in our running samples we will use data from temporal relational databases but our discussion applies to any more complex nested XML structure. We will consider valid time data but it is applicable to any other temporal dimension that requires to handle 'now'. The running sample used in this paper, which captures the history of Scott's positions, is shown in Table 1.

Name	Position	Vstart	Vend
Scott	A	2000-05-19	2001-03-12
Scott	B	2001-03-12	2004-03-10
Scott	C	2004-03-10	now

Table 1: Employment history

The first method represents a simple flat translation of relational attributes to XML elements, as introduced in (Fernandez et al. 2000). In this model, which we dubbed as the *DIRECT* model, each attribute is represented by an element in XML. According to this approach, the XML structure representing the data shown in Table 1 would look like:

```
<Employment>
<row>
  <Name>Scott </Name>
  <Position>A</Position>
  <Vstart>2000-05-19</Vstart>
  <Vend>2001-03-12</Vend>
</row>
<row>
  <Name>Scott </Name>
  <Position>B</Position>
  <Vstart>2001-03-12</Vstart>
  <Vend>2004-03-10</Vend>
</row>
<row>
  <Name>Scott </Name>
  <Position>B</Position>
  <Vstart>2004-03-10</Vstart>
  <Vend>now</Vend>
</row>
</Employment>
```

The second approach relies on XML ability to have attributes within the elements. By adding attributes to the element it becomes a complex type. But the attributes themselves are always declared as a simple type. This means that an element with attributes always has a complex type definition. For example:

```
<Position Vstart="2000-05-19"
  Vend="2001-03-12">A</Position>
```

This approach is suitable for storing XML temporal data related to the representation and management of changes. Also, it is suitable to manage temporal Web documents containing historical information. Furthermore, this approach enables usage of temporally grouped data model. Clifford et al. have shown that the temporally grouped data model has more expressive power and is more natural since it is history oriented (Clifford et al. 1994).

It is possible to restrict data supplied for attributes and also to ensure that they are supplied. When an XML element or attribute has a type defined, it puts a restriction on the element's or attribute's content. For example, if an XML element is of type "xs:date" and contains a string the element is not valid.

```
<xs:attribute name="Vstart" type="xs:date"
  use="required"/>
```

Considering the running sample from Table 1 temporally grouping data by Employee *Name*, will result in data as represented in Table 2.

Temporally grouped data, presented in Table 2, can be easily represented in XML using the attribute approach:

```
<Employment Vstart="2000-05-19" Vend="now">
  <Name Vstart="2000-05-19"
```

Name	Position
<i>2000-05-19</i>	<i>2000-05-19</i>
<b>Scott</b>	<b>A</b>
	<i>2001-03-12</i>
	<i>2001-03-12</i>
	<i>2003-02-15</i>
	<i>2003-02-15</i>
<i>now</i>	<b>C</b>
	<i>now</i>

Table 2: Temporally Grouped Valid Time History of Employees

```
Vend="now">Scott </Name>
<Position Vstart="2000-05-19"
  Vend="2001-03-12">A</Position>
<Position Vstart="2001-03-12"
  Vend="2003-02-15">B</Position>
<Position Vstart="2003-02-15"
  Vend="now">C</Position>
</Employment>
```

It is considered easy to perform different queries using XQuery, for example to retrieve the snapshot at the certain date on closed intervals in either of approaches to add temporal dimensions to XML. But it is an open question how to represent current time or *now* in XML-temporal in order to efficiently and accurately access open ended intervals, *now-relative* XML-temporal data.

### 2.3 Modelling *now-relative* temporal data in XML

In line with research in temporal databases applied to relational technology, with respect to different representations for *now*, we decided to evaluate same approaches in XML.

#### MAX

The often mentioned approach to represent current time is to represent the current time as unrealistic large date most often used '31-DEC-9999', which in XML to ensure the order has to be in format 'YYYY-MM-DD' or '9999-12-31'. In the reminder of this paper we will refer to this approach as *MAX* approach.

Considering the nature of XML and Native XML databases, and the fact that data are stored as text, there are several other different representations for current time that could be of interest.

#### Null Data

Despite being ruled out as not suitable in relational databases, as columns with *NULL* can not be efficiently indexed, we decided to evaluate the *NULL* approach due to the specific nature of *NULL* in XML. In the database world, null data means that data simply is not there. Considering the XML, *NULL* can mean that value is inapplicable or value is missing. XML supports the concept of *NULL* data through optional element types and attributes. If the value of an optional element type or attribute is null, it simply is not included in the document. As with databases, attributes containing zero length strings and empty elements are not null: their value is a zero-length string.

Some Native XML databases offer the choice of defining what constitutes *NULL* in an XML document, including support for *xsi:null* attribute from XML Schema.

## Variables

Due to the nature of XML and Native XML databases to store all data as text, it is possible to consider variables such as ‘now’ or ‘UC’ (until changed) to be stored as words ‘now’ or ‘UC’ to represent current time. These variables can be simple stored as text in native XML databases. Variables to represent current time are widely recommended in the literature but have not been appropriate for storing it in relational temporal database environment, and this approach has been ruled out. This is because *date type* cannot accommodate such variables. In contrast XML offers the possibility to create new complex data-type that inherit properties of the data-types used to create them. Thus it is possible to create a new temporal data-time as the union of normal time data-type and the string ‘now’. In any case, even if XML has data-types these are just defined as combinations of Unicode characters, thus they are simply strings of text.

A further advantage of the ‘now’ approach is that opens the possibility to separate the time value representation from the representation of *now*, for example by introducing an empty and optional sub-element `<now/>` of `<vend>`. Thus we can have:

```
<vend><now/></vend>
```

for the ending time of *now*. Accordingly queries that require ‘now’ can traverse a path expression that descends to *now*, e.g.,

```
//vend/now
```

while, for queries where this is not needed, the expression

```
//vend < current-date()
```

is well-defined.

In addition, this strategy can take advantage of DBMS offering element indexes beside word indexes. In this case all `<now/>` will be included in an element index and the index can be used to faster search. This is in contrast with word indexes where ‘now’ can occur inside a sentence instead of as a special timestamp; consequently we have to check that the elements where ‘now’ occurs are of the right type.

## 3 Empirical study

In order to find the best choice for representing *now* in XML we decided to test performance and accuracy of all identified approaches. Testing was performed both on XML structure where timestamps are represented as XML elements (Direct model) and where timestamps are represented as attributes of XML elements (on temporally grouped data). Also, we intended to compare query response time and space usage for *Direct* and grouped model.

During the experiment we identified that some of the approaches to represent *now* do not yield the correct answer. This is the case if data contains closed intervals where the ending point is bigger than the current time. For that reason, we decided to include in our tests checking whether the query yields the correct number of elements that satisfy the given condition.

### 3.1 Environment

The experimental results presented in this section are computed on four 450MHZ CPU - SUN UltraSparc II processor machine running

the open source Native XML eXist database (*eXist*:<http://exist.sourceforge.net/> 2004). During the testing server did not have any other significant load.

We decided to test the performance of a point query that timeslices time line at the current time. Point queries, as a special type of range queries, are considered to be the most important query type for temporal data. This is because it is expected that the current state of reality will be queried most often.

Searching a native XML database is handled in different ways, depending on the vendor of the database. Some native XML databases require the user to select the elements or attributes to be indexed. This information is then used to build an index that the searching mechanism can use to faster locate matching documents. Other native XML databases simply index all elements in a document, which obviously causes the need for more storage space. Indexing all elements in native XML databases has more sense compared with indexing of all columns in a relational database. While most of the XML native databases use well proven  $B^+$ -tree structures for indexing, specific demand of XML databases has forced introduction of different approaches such as: Reverse-Lookup indexing and Forward Dictionary Segment Build-Up indexing invented by QuiLogic, as well as traditional indexing technologies like hashing. The native XML eXist database, used for this experiment, uses  $B^+$ -tree structures for indexing. Users have option to define the elements and depth that should be indexed.

In our experiments index depth was set to three and all elements were included in the index.

### 3.2 Data sets

In order to investigate the effect of different percentage of *now-relative* data we used different data distributions. The start position of the intervals was always uniformly distributed on the interval domain, while the duration and percentage of *now-relative* data was varied. The following data distributions have been considered:

- Uniformly distributed interval start and uniform distributed duration within the range [1,10000] with 10% of uniformly distributed *now-relative* data.
- Uniformly distributed interval start and uniform distributed length within the range [1, 10000] with 20% of uniformly distributed *now-relative* data.

In relational environment 100.000 rows of sample data was generated and then converted to XML format. Same sample data set represented in *Direct* model required 400.000 elements while temporally grouped model for same data, due to the grouping, required only 154.256 XML elements. Part of data for temporary grouped model is shown in Table 2. For each approach to represent *now* we created two XML files, one for temporally grouped model and one for *Direct* model. XML files differ only on the semantic to represent *now*. The resulting files were imported into *eXist* XML Native database.

### 3.3 Query Sets

We focus on intersection queries and particularly on Point queries as specific cases. The results for intersection queries also hold for the containment and enclosure queries, as those are a subset of the intersection query. We use two different query sets:

- Window: a set of queries sorted according to the start point and with a fixed length. This query set is covering the whole data space.
- Random: A set of random query intervals with different answer size.

### 3.4 Results

All identified approaches to represent *now* have been tested. For *MAX* approach, where current time is stored as some unrealistically big date, we used 9999-12-31. XML does not support data types in any meaningful sense of the word, all data in an XML document are stored as text. This is even if data represents another data type, such as date or integer. For that reason, it was necessary to represent date in format YYYY-MM-DD to ensure that dates can be ordered into ordered list.

The XQuery code used to test the performance of the *MAX* approach without referencing the current time approach is:

```
for $b in doc("/db/now/dirmax.xml")/table/row
  where $b/vend='9999-12-31' and
         $b/vstart<xs:string(current-date())
  return
  <result> { $b } </result>
```

We found that without referencing to the current time the query yields the wrong answer if data contains closed intervals with ending point bigger than the current date. These intervals are meaningful for all temporal data. In order to get the correct answer, considering the number of elements returned, there is a need to compare ending point of the interval validity with the current time. This can be achieved using the XQuery function `current-date()` that references to the current time:

```
for $b in doc("/db/now/dirmax.xml")/
  table/row
  where $b/vend>=xs:string(current-date())
  and $b/vstart<xs:string(current-date())
  return
  <result> { $b } </result>
```

Usage of user-defined functions ensures that the query yields the correct answer when current time is represented with *MAX* approach. A sample user-defined function `check_now` (expressed in XQuery) returns `vend`, if the value is different from '9999-12-31' and `current-date` otherwise.

```
declare function local:check_now
  ($n as xs:string) as
  xs:string{ if ($n="9999-12-31") then
    xs:string(current-date()) else $n };
for $b in doc("/db/now/dirmax.xml")/
  table/row
  where local:check_now($b/vend)>=
    xs:string(current-date())
  and $b/vstart<
    xs:string(current-date())
  return
  <result> { $b } </result>
```

Usage of this user-defined function guarantees correct answers but suffers from extremely poor performance as it cannot use indexes, so a sequential search is required.

The nature of the XML native databases, all data are stored as text, opens the possibility to store *now* as the word "now". This approach could not be considered for relational databases, since it is not possible to store characters into date data type. Introduction

of variables in temporal relational databases leads to the under researched area of *Variable databases*. In XML it is possible to create a new complex data-type as the union of normal time data-type and the string 'now'. Because storing variables as text in XML is straightforward we decided to test performance and accuracy of representing current time with the variable such as 'now'. To indicate that the fact is currently valid the variable 'now' is assigned for the ending point of their validity. The following XQuery was used for performance and accuracy testing of variable approach for representing current time.

```
for $b in doc("/db/now/dirnow.xml")/
  table/row
  where $b/vend>='now' and
         $b/vstart<xs:string( current-date())
  return
  <result> { $b } </result>
```

Without referencing to the current time the variable approach also does not yield correct answers if data contains closed intervals with ending point bigger than the current date. Referencing to the current time yields correct answer.

Because XML native databases have a different view to *NULL* data than the relational databases we decided to reconsider and test the performance and accuracy of the *NULL* approach to represent current time. Advantages of *NULL* are obviously from used space point of view. Response time for *NULL* approach is very good in case of not referencing to the current time but due to the wrong answer problem it can not be considered. To get the correct answer there is a need to reference to the current time with user defined functions. Despite being previously suggested as favorite in the literature, usage of user defined functions performed very poorly.

For temporally grouped model we performed queries on identified approaches to representing *now* with and without referencing to the current time. Same as for *DIRECT* model, any approach that does not reference to the current time yields wrong answer if data contains closed intervals of validity where the ending point of validity is beyond the current time. All such intervals are omitted and not included in the answer. The sample query for *MAX* approach that references to the current date function is as follows:

```
for $s in doc("/db/now/grmax.xml")/
  table/row/
  position[@vstart<xs:string(current-date())
  and
  @vend>=xs:string(current-date())]
  return
  <result> { $s/./name }
  { $s }
</result>
```

Also, we performed a query that finds all employees that have current position and have started their employment before the certain date. We performed this experiment to investigate how answer size effects the response time.

For *NULL* approach to represent current time it is not possible to compare with the current time directly because *NULL* is represented by the empty string and it is not clear whether the empty string is bigger or smaller than any string. Only possibility is to use user defined functions.

Testing *NULL* with temporally grouped model yielded totally wrong number of elements and had a very long query response time so we did not include the results in the above table.

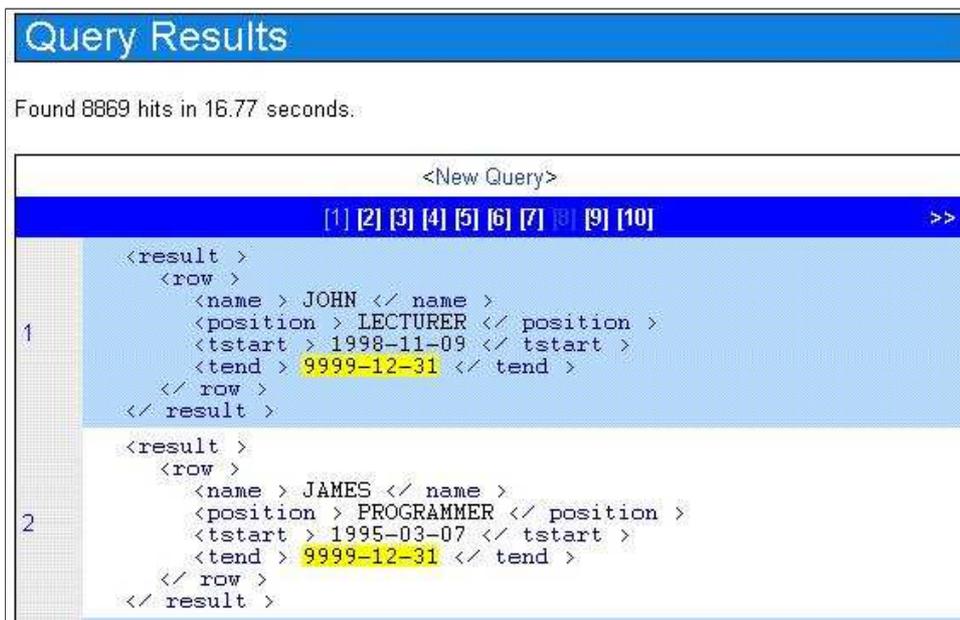


Figure 1: MAX approach without reference to the current date

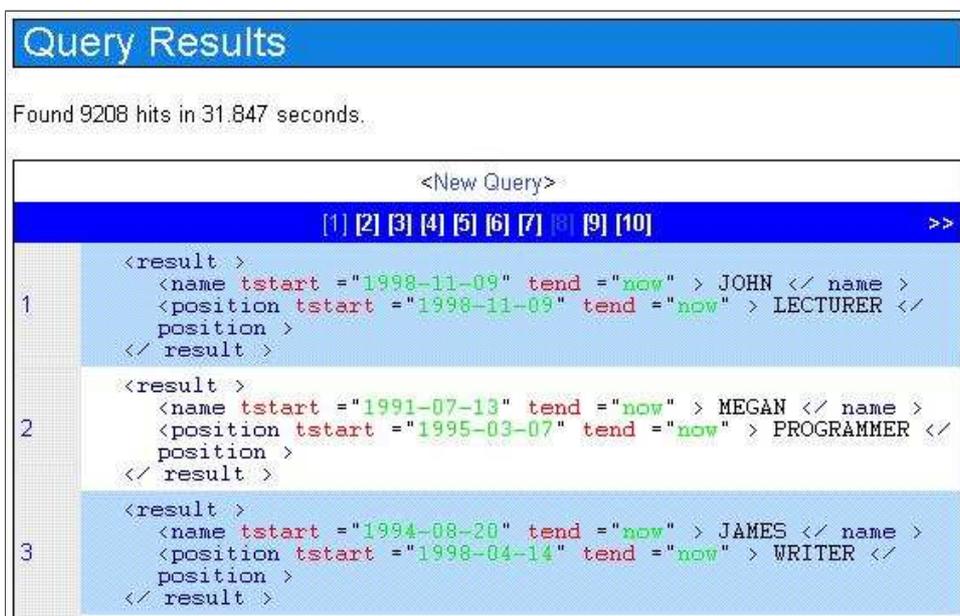


Figure 2: Temporally Grouped model - variable approach with reference to the current time

Approach	Reference current time	Query yields correct answer	Timeslice query (sec)	Contained query (sec)
MAX	No	No	16.77	7.78
MAX	Yes	Yes	27.47	14.08
MAX with user defined function	Yes	Yes	100.48	98.98
'now'	No	No	14.96	7.14
'now'	Yes	Yes	23.88	12.25
NULL	No	No	22.08	9.13
NULL with user defined function	Yes	Yes	97.64	94.26

Table 3: Direct model - response time for different representations of *now*, 20 % of *now*-relative data

### 3.5 Analysis

It is significant to note that, if there is no reference and comparison to the current time the query yields the wrong answer, considering the number of elements returned that satisfy the query criteria. This is because all elements with closed interval whose ending time is bigger than the current time are not included in the answer. Usage of user defined functions, which

was previously suggested as favorite in the literature, has poor response time as index can not be used and sequential search is required. Due to the nature of Native XML databases, where all data is stored as text, the current time can be stored as variable such as 'now' or 'UC'. Because the ASCII code of the mentioned variables is bigger than `xs:string(current-date())`, the usage of variables yields correct answer and at the same time uses less space. Surprisingly,

Approach	Reference current time	Query yields correct answer	Timeslice query (sec)	Contained query (sec)
MAX	No	No	32.36	20.78
MAX	Yes	Yes	36.10	20.23
MAX with user defined function	Yes	Yes	104.43	94.18
'now'	No	No	27.03	16.95
'now'	Yes	Yes	31.84	18.52

Table 4: Temporally Grouped Model - response time for different representations of *now*, 20 % of now-relative data

temporally grouped model has worse response time than the *Direct* model, despite having more expressive power and is more natural since it is history oriented. This is due to more complex structure of elements. Grouped model is slower due to the complexity of elements that consist of attributes `vstart` and `vend` and index can not be used efficiently.

Faster response time when there is no reference to the current time is partly due to the wrong and smaller answer size and because in case of referencing to the current time there is additional processing cost for build-in function `current-date()` and need for conversion of the date to the string.

Slightly better performance of the variable approach to represent current time in comparison to the *MAX* approach is due to the smaller length of string 'now' comparing to the '9999-12-31', which causes smaller XML file size and also higher fanout of elements in index nodes. Also, computational cost for comparison is smaller due to the shorter length of the string 'now'.

#### 4 Conclusion and future work

This study makes the following contributions to the field:

- By investigating different representations of *now* in XML, we presented a better understanding of the significance of modelling current time, particularly how it influences the efficiency and accuracy;
- We identified available approaches for adding time dimension to XML;
- We empirically demonstrated that usage of user defined functions to handle current time in XML, which is previously recommended as favorite in the literature, is basically inefficient and is obviously not appropriate;
- We showed that the flat translation of temporal attributes to XML elements (*DIRECT* model), is more efficient than the usage of attributes within the XML elements.
- We identified available options to represent and store current time in native XML databases and empirically evaluated their suitability;
- We concluded that any approach to represent 'now' if not referencing to the current time yields wrong answer.
- We introduced the notion of storing variables such as 'now' or 'UC' as strings in XML native databases. Usage of variables at the same time yields the correct answer and query response time is good. Another advantage of using variables is clear semantics, the meaning of the word 'now' suggests of current time in contrast to '9999-12-31' where the meaning is introduced by convention. For those reasons storing variables

to represent current time can be considered as the most appropriate approach in XML temporal.

The present paper shows that native XML databases offer better support for temporal reasoning than relational databases and at the same time they support richer data models. As we have argued, temporal data is very frequent in real life application, thus we believe that native XML database will present a viable alternative to relational temporal database when complex time dependent data has to be manipulated and recorded. On the contrary due to the nature of XML data and the verbosity of XML, the response time of the Native XML temporal databases does not compare with the response time of relational databases. This also indicates the need for further research on efficient storage architecture and access methods for Native XML temporal databases.

We intend to work on more efficient access method for temporal XML data, based on the intrinsic nature and format of temporal data types in XML databases. We believe that the resulting access method will prove useful not only in dealing with XML temporal data, but also can be employed on XML data of different nature.

#### References

- Amagasa, T., Yoshikawa, M., & Uemura, S. (2000), 'A Data Model for Temporal XML Documents', *In Proc. of 11th Intl' Conf. on Database and Expert Systems Applications (DEXA 2000), London, England*.
- Carey, M., Kiernan, J., Shanmugasundaram, J. & et al (2000), 'XPERANTO: A middleware for publishing objectrelational data as XML documents', *VLDB*.
- Chawathe, S. S., Abiteboul, S. & Widom, J. (1999), 'Managing Historical Semistructured Data', *Theory and Practice of Object Systems* 5(3), 143–162.
- Clifford, J., Croker, A., Grandi, F. & Tuzhilin, A. (1995), 'On temporal grouping', *In Recent Advances in Temporal Databases, Springer Verlag* p. 194213.
- Clifford, J., Croker, A. & Tuzhilin, A. (1994), 'On Completeness of Historical Relational Query Languages', 19(1), 64–116.
- Date, C., Darwen, H. & Lorentzos, N. (2002), *Temporal Data and the Relational Model*, Morgan Kaufmann.
- eXist*:<http://exist.sourceforge.net/> (2004).  
\*<http://exist.sourceforge.net/>
- Fernandez, M., Tan, W. & Suciu, D. (2000), 'Silkroute: trading between relations and XML', 33(16), 723–745.

- Funderburk, J., Kiernan, G., Shanmugasundaram, J., Shekita, E. & Wei, C. (2002), 'XTABLES: Bridging Relational Technology and XML', *IBM Systems Journal* **41**(4).
- Gao, D. & Snodgrass, R. T. (2003), 'Syntax, Semantics, and Query Evaluation in the  $\tau$ XQuery, Temporal XML Query Language', *TR-72 A TIMECENTER Technical Report*.
- Grandi, F. & Mandreoli, F. (2000), 'The Valid Web: an XML/XSL Infrastructure for Temporal Management of Web Document', *In Proc. of the Intl' Conf. on Advances in Information Systems (ADVIS'2000), Izmir, Turkey*.
- Grandi, F., Mandreoli, F., Tiberio, P. & Bergonzini, M. (2003), 'A Temporal Data Model and System Architecture for the Management of Normative Texts (Extended Abstract)', *Proc. SEBD 2003 - Natl' Conf. on Advanced Database Systems, Cetraro, Italy* pp. 169–178.
- <http://www.sqlx.org> (2004).  
\*<http://www.sqlx.org>
- Jensen, C. S. (2000), '<http://www.cs.auc.dk/csj/Thesis/pdf/>'.  
\*<http://www.cs.auc.dk/csj/Thesis/pdf/>
- Ozsoyoglu, G. & Snodgrass, R. (1995), 'Temporal and real-time databases: A survey', *IEEE Trans. On Knowledge and Data Engineering* **7**(4), 513–532.
- Stantic, B., Khanna, S. & Thornton, J. (2004), 'An Efficient Method for Indexing Now-relative Bitemporal data', *In Proceeding of the 15th Australasian Database conference (ADC2004), Denidin, New Zealand* **26**(2), 113–122.
- Torp, K., Jensen, C. S. & Bohlen, M. (1999), 'Layered implementation of temporal DBMS concepts and techniques', *A TimeCenter Technical Report TR-2*.
- Wang, F. & Zaniolo, C. (2002), 'Preserving and Querying Histories of XML-published Relational Databases', *In Proc. of 2nd Intl' Workshop on Evolution and Change in Data Management (ECDM 2002), Tampere, Finland* pp. 26–38.
- Wang, F. & Zaniolo, C. (2003), 'Temporal Queries in XML Document Archives and Web Warehouses', *in 'In Proceeding of the 10th International Symposium on Temporal Representation and Reasoning (TIME-ICTL 2003), Cairns, Australia'*, pp. 47–55.
- [www3.ibm.com](http://www3.ibm.com) (2005), 'In DB2 XML Extender'.  
\*<http://www3.ibm.com/software/data/db2/extenders/xmlxext/>
- [xml.com](http://www.xml.com) (2005), 'Making the case for XML Native databases'.  
\*<http://www.xml.com/pub/a/2005/03/30/native.html>
- XQuery 1.0: An XML query language* (2004).  
\*<http://www.w3.org/TR/xquery/>