Representing and Querying the Evolution of Databases and their Schemas in XML

Fusheng Wang and Carlo Zaniolo

Department of Computer Science
University of California, Los Angeles
wangfsh@cs.ucla.edu zaniolo@cs.ucla.edu

Abstract

We show that XML views combined with XML query languages can provide surprisingly effective solutions to the problem of representing and querying the evolution of databases—both the evolution of their contents and the evolution of their schemas. Indeed, using XML, the histories of database relations can be represented naturally by means of temporally grouped data models. We show that schema changes and table histories that would be very difficult to represent in relational databases can be easily represented using XML, and also queried using XQuery.

1 Introduction

Information systems have yet to address satisfactorily the problem of how to deal with the evolution in the structure and content of their underlying databases, and to support the query and retrieval of past information. This is a serious shortcoming for the web, where documents frequently change in structure and content, or disappear all together, causing serious problems, such as broken links and lack of accountability for sites of public interest. Database-centric information systems face similar problems, particularly since current database management systems (DBMSs) provide little help in that respect. Indeed DBMSs do not provide effective means for archiving and supporting historical queries on their past contents—to the point that, e.g., the current SQL standards lack the basic temporal extensions needed to express and support historical queries. Given the strong application demand and the significant research efforts spent on these problems [1], the lack of current solutions must be attributed, at least in part, to the technical difficulty of introducing temporal extensions into relational databases and object-oriented databases. Schema changes represent a particularly difficult and important problem for modern information systems, which need to be designed for evolution [20, 19, 13].

Meanwhile, 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 [2]. The definition of the mapping from the database tables to the XML view is in fact used to translate queries on these views into equivalent SQL queries on the underlying database [3, 4].

As the database is updated, its external XML view also evolves—and many users who are interested in viewing and querying the current database are also interested in viewing and querying its past snapshots and evolving history. In this paper, we show that the history of a relational database can be viewed naturally as yet another XML document. The various benefits of XML-published relational databases (browsers, web languages and tools, unification of database and web information, etc.) are now extended to XML-published relation histories. In fact, we show that we can define and query XML views that support a temporally grouped data model, which has long been recognized as very effective in representing temporal information [5], but could not be supported well using relational tables and SQL. Therefore, temporal queries that would be very difficult to express in SQL can now be easily expressed in standard XQuery. Furthermore, we show that database schema changes can also be represented easily in the XML view, and queries on schema changes can thus be supported.

2 Related Work

Time in XML. Some interesting research work has recently focused on the problem of representing historical information in XML. In [6] an annotation-based object model is proposed to manage historical semistructured data, and a special Chorel language is used to query changes. In [7] 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 [8, 9], 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, TTXPath[10] is another extension of XPath data model and query language to support transaction time semantics. (In our approach, we instead support XPath/XQuery without any extension to XML data models or query languages.)

An archiving technique for scientific data was presented in [11], based on an extension of the SCCS [12] scheme. This approach timestamps elements only when they are different from the parent elements, so the structure of the representation is not fixed; this makes it difficult to support queries in XPath/XQuery, which, in fact, is not discussed in [11]. The scheme we use here to publish the histories of relational tables present several similarities to that proposed in [11], but it also provides full support for XML query languages such as XPath and XQuery.

Temporal Databases and Grouped Representations. There is a large number of temporal data models proposed and the design space for the relational data model has been exhaustively explored [1]. Clifford et al. [5] classified them as two main categories: *temporally ungrouped* and *temporally grouped*. The second representation is said to have more expressive power and to be more natural since it is history-oriented [5]. TSQL2 [13] tries to reconcile the two approaches [5] within the severe limitations of the relational tables. Our approach is based on a temporally grouped data model, since this dovetails perfectly with XML documents' hierarchical structure.

Object-oriented temporal models are compared in [14], and a formal temporal object-oriented data model is proposed in [15]. The problem of version management in object-oriented and CAD databases has receive a significant amount of attention [16, 17]. However, support for temporal queries is not discussed, although query issues relating to time multigranularity were discussed in [18].

Schema Evolution. Besides the change of data, the schema of the database can evolve over time, e.g., to adapt to new applications, or merge with other ones. Schema evolution and schema versioning issues in relational database have been investigated in a number of studies that are discussed in [19], where schema changes are classified into schema modification, schema evolution and schema versioning. Partial schema versioning [19] is a weaker concept compared to schema versioning, where data stored under any historical schema can be viewed through any other schema, but updates can only be issued through the current schema. A model for schema versioning in temporal database systems is discussed in [20], where an extension to temporal data models is proposed to support partial schema versioning. Schema evolution is also a very important issue for object-oriented DBMS and has been implemented in commercial OODBMSs such as O2 [13].

Publishing Relational Databases in XML. There is much current interest in publishing relational databases in XML. One approach is to publish relational data at the application level, such as DB2's XML Extender [21], which uses user-defined functions and stored procedures to map between XML data and relational data. Another approach is a middleware based approach, such as in SilkRoute [22] and XPERANTO [3, 4], which define XML views on top of relational data for query support. 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 DB2 engine. This approach utilizes RDBMS technology and provides users with a unified general interface.

Several DBMS vendors are jointly working toward new SQL/XML standards [23]; the objective is to to extend SQL to support XML, by defining mappings of data, schema, actions, etc., between SQL and XML. A new XML data type and a set of SQL XML publishing functions are also defined, and are partly supported in Oracle 9i [24].

3 Publishing Relational Database Histories as XML Documents

This problem was studied in [25] where alternative publication schemes were explored. The scheme used next is the one which was found to be the most effective [25].

Table 1 and Table 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 contains the evolving snapshots of these relations using a temporally-grouped data model that takes advantage of the richer structure of XML, and the expressive power of XQuery.

For instance, we represent and preserve the evolving history of Table 1 and Table 2 by means of the XML documents shown in Figure 1 and Figure 2. We will call these *H-documents*. Each element in an H-document is assigned the attributes `tstart` and `tend`, to represent the inclusive time-interval of the element. The value of `tend` can be set to *now*, to denote the ever-increasing current time. In this way, the history of each attribute is grouped and represented by such XML elements. The H-document also has a simple and well-defined schema that can be derived directly from the schema of the current data.

Table 1. The snapshot history of employees

name	empno	salary	title	deptno	start	end
Bob	1001	60000	Engineer	d01	1995-01-01	1995-05-31
Bob	1001	70000	Engineer	d01	1995-06-01	1995-09-30
Bob	1001	70000	Sr Engineer	d02	1995-10-01	1996-01-31
Bob	1001	70000	TechLeader	d02	1996-02-01	1996-12-31

Table 2. The snapshot history of departments

deptname	deptno	mgrno	start	end
QA	d01	2501	1994-01-01	1998-12-31
RD	d02	3402	1992-01-01	1996-12-31
RD	d02	1009	1997-01-01	1998-12-31
Sales	d03	4748	1993-01-01	1997-12-31

For updates on a node, when there is a `delete`, the value of `tend` is updated to the current timestamp; when there is an `insert`, a new node is appended with `tstart` set to current timestamp and with `tend` set to *now*; and `update` can be implemented as a delete followed by an insert.

Note that there is an intrinsic constraint that the interval of a parent node always covers that of its child nodes, and this constraint is always preserved while nodes are updated. Indeed, when the salary of an employee is updated, `tend` of the employee must be *now*, and `tend` of the old salary is set to the current timestamp, `tstart` of the new salary element is set to the current timestamp, and `tend` of the new salary is set to *now*, thus the constraint is preserved. For the case where an employee is deleted, the `tend` of the employee elements and all its children are set to the current timestamp.

Our H-documents use a temporally grouped data model [5]. Clifford et al. [5] 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 [13]. TSQL2 [13] attempts to achieve a compromise between these two [5], using an implicit temporal model, which is not without serious problems [26]. With our model, the history of each attribute is at hand thus eliminating the most common need for coalescing; moreover, while coalescing is still be needed for certain queries, this can be expressed in XQuery without any language extension.

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. In later sections we will show how to express temporal projections, snapshot queries, joins and historical queries on `employees` and `departments`. Furthermore, database schema evolution queries are also supported in our approach.

3.1 Publishing Each Table as an XML Document with Columns as Elements

A natural way to publish relational table histories is to publish the history of each table as a separate XML document, where relational columns are mapped into XML elements [27]. 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.

Several alternative representations for table histories were studied in [25]. For instance, multiple tables can be first joined together and then represented by a single XML document. This approach offers no advantage compared to the one described above [25]. However IDs can be added to this representation to make some join queries easier [25]. Yet another approach consists of representing the joined tables by a hierarchically structured XML document. This approach simplifies some queries but complicate others [25]. The last approach is to represent the tuples of a relational table by the attribute values of the XML document. Then, the XML document reproduces the flat structure of tables with timestamped tuples, and the well-known problems of this temporally ungrouped representation [25]. In summary, publishing each table as a separate XML document with columns as elements was shown to be the approach of choice in [25].

3.2 Preserving Schema Changes

The database history not only includes the changes of relational data, but also the changes of the database schema. A variety of temporal data models have been proposed [1] and few of them provide support for schema evolution or schema versioning [20]. The flat structure of relational database makes it difficult to support changes—particularly schema changes.

The most basic schema evolutions in RDBMS are attribute evolution and relation evolution. Attribute evolution includes adding an attribute to the database and removing an attribute from the database. Relation evolution include adding a relation, removing a relation, joining two relations into one, and decomposing a relation into two or more relations.

By publishing the database history as XML documents (H-documents), we can also represent the schema history, with the help of the rich structure of XML.

In Figure 1, the two attributes `tstart` and `tend` associated with the relation `employees` (the root node) actually represent the time of creation and the time of deletion

```
<employees tstart="1995-01-01" tend="1996-12-31">
  <employee tstart="1995-01-01" tend="1996-12-31">
    <empno tstart="1995-01-01" tend="1996-12-31">1001</empno>
    <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>
    <deptno tstart="1995-01-01" tend="1995-09-30">d01</deptno>
    <deptno tstart="1995-10-01" tend="1996-12-31">d02</deptno>
  </employee>
  <!-- ... -->
</employees>
```

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

```
<depts tstart="1991-01-01" tend="1998-12-31">
  <dept tstart="1991-01-01" tend="1998-12-31">
      <deptno tstart="1991-01-01" tend="1998-12-31">d02</deptno>
      <deptname tstart="1991-01-01" tend="1998-12-31">RD</deptname>
      <mgrno tstart="1992-01-01" tend="1996-12-31">3402</mgrno>
      <mgrno tstart="1997-01-01" tend="1998-12-31">1009</mgrno>
  </dept>
  <!-- ... -->
</depts>
```

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

of the relation. By searching all the root nodes of the H-documents, we can have a complete history of all relations in the database.

Similarly, by coalescing the intervals of all `empno`, we get the interval of `empno` in the relation. Thus, the change history of every attribute of the relation is preserved. This is true under the assumption that there is no empty relation, and e.g., a relation begins when its first element is inserted. Indeed, content-free elements can be used to fill empty relations. For example, for the `salary` column, we can add the following element that specifies the period of existence of the salary attribute, independent of the existence of any actual salary value.

```
<salary tstart="2003-01-01" tend="now" />
```

The case of relations joined or decomposed can be handled by the usual view mechanism. For example, say that the two tables `employees(empno, name,salary)` and `addresses(empno, address)` are joined into a new table `newemployees (empno, name, salary, address)`. Then, we have three H-documents: an H-document for `employees` and an H-document for `addresses` which stop at the joining time, and an H-document for `newemployees` which starts from the joining time. Thus, to query the history we can i) query each H-document individually and join the results, or ii) merge the histories from the three H-documents, e.g., publish a new H-document `allemployees`, that includes all the histories before and after the join. Therefore schema changes involving the join or decomposition of relations can be handled through such H-documents.

4 Historical Queries with XQuery

4.1 Data History Queries

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

Query 1: Temporal projection: retrieve the history of managers in dept 'QA':

```
element mgrs{
  for $s in document("depts.xml")/departments
    /dept[deptname="QA"]/mgrno
   return $s }
```

Query 2: Snapshot queries: find the count of employees on 1996-01-31:

```
let $e := document("employees.xml")/employees
    /employee[@tstart<="1996-01-31"
    and @tend>="1996-01-31"]
return count($e)
```

Query 3: Interval History: find the depts history from 1995-05-01 to 1996-04-30:

```
for $d in document("depts.xml")/depts
    /dept
let $ol:=overlapinterval(
        $d, telement("1995-05-01","1996-4-30") )
where not (empty($ol))
return ( $d/deptname, $ol )
```

Here `overlapinterval($a, $b)` is a user-defined function that returns an element `interval` with overlapped interval as attributes`(tstart, tend)`. User-defined functions are functions defined by users with the syntax of XQuery [2]. If there is no overlap, then no element is returned which satisfies the XQuery built-in function `empty()`. `telement()` will generate an element in the form of <`interval tstart= "..." tend="..."/`>. The next query is a join query:

Query 4: Join Queries: Find the manager(s) for employee 'Bob':

```
let $e := document("employees.xml")/employees
    /employee[name="Bob"]
for $ed in $e/deptno
let $d := document("depts.xml")/depts/dept
   [deptno=$ed]
for $m in $d/mgrno
let $ol:=overlapinterval($m,$ed)
where not (empty($ol))
return( $e/name, $m, $ol )
```

This query will join `employees.xml` and `depts.xml` by `dept`, and the `overlapinteral()` function will return only managers that overlap with the employee's dept.

Query 5: Grouping: 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/deptno = $d/deptno and
        not(empty(overlapinteral($e, $d)))
     return($e/name, overlapinterval($e,$d))
}}}
```

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

4.2 Schema History Queries

Since the schema history is incorporated in the H-documents, with powerful XML query languages such as XQuery, we can query the schema history directly without any extensions. We can retrieve all the schema information along the history, retrieve the schema snapshot at any timestamp, and detect when the schema changes. Our scheme can not only model the schema evolutions, but also query the changes of the database schema. The following queries on schema history were tested with Quip [28] (SoftwareAG's implementation of XQuery).

Schema Q1: Schema Snapshot: find all the columns of `employees` relation on '1995-01-01':

```
<columns>{
for $e in distinct-values(collection("employees")
  /employees/employee/*[@tstart<="1995-01-01"
  and @tend>="1995-01-01"]/local-name(.)  )
return <column>{$e}</column>
}</columns>
```

Schema Q2: Schema History of Relations: find the history of the relation `employees`.

```
let $rel := collection("employees")/employees
return <tstart>{$rel/@tstart}</tstart>,
          <tend>{$rel/@tend}</tend>
```

Schema Q3: Schema History in One Relation: find the history of all attributes in the `employees` relation:

```
<columns>{
for $name in distinct-values(collection("employees")
  /employees/employee/*/local-name(.)  )
let $e := collection("employees")/employees
    /employee
for $mergelist in mergeIntervals(
    $e/*[local-name(.)=$name])
return <column>{$name,$mergelist/@tstart,
       $mergelist/@tend}</column>
}</columns>
```

where `mergeIntervals($list)` is a user-defined function that coalesces a list of history intervals and return the merged lists. Note that the history of a node has to be continous unless the schema changes. e.g., if the attribute salary is added in the table, then dropped, and later added back, then the `mergeIntervals` function will return a list of two intervals, with each interval representing an existing interval of the attribute salary.

Schema Q4: Schema Change Detection: find when the address column was first added into the table:

```
let $a := collection("employees")/employees
    /employee/address
let $ts := min( $a/@tstart )
return $ts
```

Schema Q5: Joined Relations: Find the salary history of employees '10000'. Assume that two tables `employees`(empno, name,salary) and `addresses`(empno, address) are joined into a new table `newemployees`(empno, name, salary, address).

There are two ways to specify the query, one is based on a composed H-document that represent the whole history of employees. And the other one is to specify queries on different H-document and merge the history in the result as follows:

```
let $s1:= document("employees")/employees/
     employee[empno='10000']/salary
let $s2:= document("newemployees")/employees/
     employee[empno='10000']/salary
return merge($s1, $s2)
```

In the above query, `merge($s1,$s2)` is a user-defined function to coalesce two history lists (only the last node in the first list `a` and the first node in the second list `b` need to be compared).

5 Conclusion

In this paper, we have shown that XML-based representations based on a temporally-grouped data model provide efficient ways to represent the evolution of the content of a database, and also the evolution history of its schema. These representations would be very difficult to realize in the context of the flat relational data model. We also showed that, without any extension, the coming standard XML query language XQuery can express complex temporal queries, including schema evolution queries, on such XML representations.

An important issue which was not discussed here is efficient support for historical queries. This problem was discussed in [29], where we compared the two approaches of storing the XML-viewed history of a database using either a native XML database or a relational DBMS. The experiments presented in [29] suggest that relational DBMSs can offer better query performance, while native XML databases can offer better storage efficiency using data compression techniques. A segment-based archiving scheme based on the notion of page usefulness was also proposed in [29] to improve the overall performance on temporal queries.

Acknowledgments

The authors would like to thank Shy-Yao Chien and Vassilis Tsotras for many inspiring discussions and Richard Luo Chang for helping with the experiments. This research was supported under grant NSF-IIS 0070135.

References

[1] G. Ozsoyoglu and R.T. Snodgrass. Temporal and real-time databases: A survey. *Knowledge and Data Engineering*, 7(4):513–532, 1995.

[2] XQuery 1.0: An XML query language. http://www.w3.org/TR/xquery/.

[3] M. Carey, J. Kiernan, J. Shanmugasundaram, and et al. XPERANTO: A middleware for publishing object-relational data as XML documents. In *VLDB*, 2000.

[4] J.E. Funderburk, G. Kiernan, J. Shanmugasundaram, E. Shekita, and C. Wei. XTABLES: Bridging relational technology and XML. *IBM Systems Journal*, 41(4), 2002.

[5] J. Clifford, A. Croker, F. Grandi, and A. Tuzhilin. On temporal grouping. In *Recent Advances in Temporal Databases*, pages 194–213. Springer Verlag, 1995.

[6] S.S. Chawathe, S. Abiteboul, and J. Widom. Managing historical semistructured data. *Theory and Practice of Object Systems*, 24(4):1–20, 1999.

[7] F. Grandi and F. Mandreoli. The valid web: An XML/XSL infrastructure for temporal management of web documents. In *ADVIS*, 2000.

[8] T. Amagasa, M. Yoshikawa, and S. Uemura. A data model for temporal xml documents. In *DEXA*, 2000.

[9] T. Amagasa, M. Yoshikawa, and S. Uemura. Realizing temporal xml repositories using temporal relational databases. In *CODAS*, pages 63–68, 2001.

[10] C.E. Dyreson. Observing transaction-time semantics with TTXPath. In *WISE (1)*, 2001.

[11] P. Buneman, S. Khanna, K. Ajima, and W. Tan. Archiving scientific data. In *ACM SIGMOD*, 2002.

[12] M.J. Rochkind. The source code control system. *IEEE Transactions on Software Engineering*, SE-1(4):364–370, 1975.

[13] C. Zaniolo, S. Ceri, C.Faloutsos, R.T. Snodgrass, V.S. Subrahmanian, and R. Zicari. *Advanced Database Systems*. Morgan Kaufmann Publishers, 1997.

[14] R. Snodgrass. Temporal object-oriented databases: a critical comparision. *Modern Database Systems: The Object Model, Interoperability and Beyond. Addions-Wesley/ACM Press*, 1985.

[15] E. Bertino, E. Ferrai, and G. Guerrini. A formal temporal object-oriented data model. In *EDBT*, 1996.

[16] D. Beech and B. Mahbod. Generalized version control in an object-oriented database. In *ICDE*, pages 14–22, 1988.

[17] H. Chou and W. Kim. A unifying framework for version control in a cad environment. In *VLDB*, pages 336–344, 1986.

[18] G. Guerrini M. Mesiti E. Camossi, E. Bertino. Automatic evolution of multigranular temporal objects. In *TIME02*, 2002.

[19] J.F. Roddick. A survey of schema versioning issues for database systems. *Information and Software Technology*, 37(7):383–393, 1995.

[20] J.F. Roddick. A model for schema versioning in temporal database systems. In *Proc. 19th. ACSC Conf.*, pages 446–452, 1996.

[21] DB2 XML Extender. http://www-3.ibm.com/software/data/db2/extenders/xmlext/.

[22] M. Fernandez, W. Tan, and D. Suciu. Silkroute: Trading between relations and xml. In *8th Intl. WWW Conf.*, 1999.

[23] SQL/XML. http://www.sqlx.org.

[24] Oracle XML. http://otn.oracle.com/xml/.

[25] Fusheng Wang and Carlo Zaniolo. Preserving and querying histories of xml-published relational databases. In *ECDM*, 2002.

[26] C. Chen and C. Zaniolo. Universal temporal extensions for database languages. In *ICDE*, 1999.

[27] J. Shanmugasundaram and et al. Efficiently publishing relational data as xml documents. In *VLDB*, 2000.

[28] Quip: Software AG's XQuery prototype. http://www.softwareag.com/tamino.

[29] Fusheng Wang and Carlo Zaniolo. Publishing and querying the histories of archived relational databases in xml. In *Submitted for Publication*.