Bridging Relational Database History and the Web: the XML Approach

Fusheng Wang
Integrated Data Systems Dept
Siemens Corporate Research
Princeton, NJ 08540, USA
fusheng.wang@siemens.com

Xin Zhou
Teradata Division
NCR Corporation
Los Angeles, CA 90245, USA
xin.zhou@ncr.com

Carlo Zaniolo
Computer Science Dept
UCLA
Los Angeles, CA 90095, USA
zaniolo@cs.ucla.edu

Categories and Subject Descriptors

H.2.4 [**Database Management**]: Systems

General Terms

Management, Performance

Keywords

Temporal Database, XML Database, Temporal Grouping, Temporal Query, XQuery

ABSTRACT

The preservation of digital artifacts represents an unanswered challenge for the modern information society: XML and its query languages provide an effective environment to address this challenge because of their ability to support temporal information and queries, and make it easy to publish database history to the Web. In this paper, we focus on the problem of preserving, publishing, and querying efficiently the history of a relational database. Past research on temporal databases revealed the difficulty of achieving satisfactory solutions using flat relational tables and SQL. Here we show that the problem can be solved using (a) XML to support temporally grouped representations of the database history, and (b) XQuery to express powerful temporal queries on such representations. Furthermore, the approach is quite general and it can be used to preserve and query the history of multi-version XML documents. Then we turn to the problem of efficient implementation, and we investigate alternative approaches, including (i) XML DBMS, (ii) shredding XML into relational tables and using SQL/XML on these tables, (iii) SQL:2003 nested tables, and iv) OR-DBMS extended with XML support. These experiments suggest that a combination of temporal XML views and physical relational tables provides the best approach for managing temporal database information.

1. INTRODUCTION

WIDM'06, November 10, 2006, Arlington, Virginia, USA.

There has been a great deal of recent interest on integrating traditional DBMS and XML, because of the significant benefits for web applications expected from this integration. But in addition to these general benefits, there are particular problem areas for which unique technical benefits are gained from this integration. In particular, temporal information management represents an area for which there is much pent-up application demand [1], but which has been difficult to support in the traditional database framework [2]. Indeed, most solutions proposed in the past require non-trivial extensions to the SQL standards, and have not been embraced by commercial vendors. However, the power and flexibility of XML and its query languages make it now possible to achieve this objective without requiring extensions to the standards, since (i) XML provides excellent support for temporally grouped data models, which have been advocated as the most natural and effective representations of temporal information [3], and (ii) unlike SQL, XQuery [4] is natively extensible and Turing-complete [5], thus extensions needed for temporal queries can be defined in the language itself. These important points are illustrated through a number of examples in Section 3 and 4, whereas the rest of the paper focuses on efficient implementation. We show that many alternative implementations are possible for storing and querying the XML-represented history of a relational database, and that the performance delivered by the various approaches can be significantly different.

Therefore, in Section 6, we study an approach based on shredding XML into relational tables and using SQL/XML on these tables. Then, in Section 7, we study the approach of mapping XML into nested relations. Finally we compare the performance of these approaches to that of using the native support provided by the DBMS to store and query such documents, including native XML DB discussed in Section 5, and OR-DBMS based XML DB discussed in Section 8.

2. RELATED WORK

2.1 Time in XML

Some interesting research work has recently focused on the problem of representing historical information in XML. In [6], valid time on the Web is supported by proposing a new <valid> markup tag for XML/HTML documents, thus temporal visualization can be implemented on web browsers with XSL.

There are other approaches to support temporal XML documents through extending XML data models or query languages, such as extending XML data model or XPath to

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

Table 1: The snapshot history of employees

support temporal XML documents in [7], [8] and [9]. (In our approach, we instead support XPath/XQuery without any extension to XML data models or query languages.)

A τXQuery language is proposed in [10] to extend XQuery for temporal support, which has to provide new constructs for the language. An archiving technique for scientific data was presented in [11]. The scheme proposed here presents several similarities to that proposed in [11], but also provides full support for XML query languages.

2.2 Temporal Databases and Grouped Representations

There is a large number of temporal data models and query languages, including [12, 13]; thus the design space for the relational data model has been exhaustively explored [2]. Clifford et al. [3] classified them as two main categories: *temporally ungrouped* and *temporally grouped* data models. The second representation has more expressive power and is more natural since it is history-oriented [3]. TSQL2 [14] tries to reconcile the two approaches [3] 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.

TimeDB [15] is a layered architecture that translates temporal queries into RDBMS, where temporal data is represented as tuples with intervals, thus temporally ungrouped. Flashback [16] and ImmortalDB [17] allow users to rollback to old versions of tables (e.g., to correct errors), but they do not provide complex temporal query support.

The use of XML in publishing and querying database history was previously proposed in [18]. No system implementation was however discussed in [18]. In this paper, instead, we investigated four alternative solutions.

2.3 SQL:2003

In the implementation of our approach we make full use of the latest SQL features. In particular, SQL:2003 standards are similar to SQL:1999, but provide significant extensions from SQL:1992 [19]. In particular, SQL:2003 O-R features include multiset, nested collection types (supported by both Oracle [20] and Informix [21]), and user-defined types. Another major feature is SQL/XML [22], which defines how SQL can be used together with XML in a database, and is supported by major database vendors. Publishing functions provided by SQL/XML can directly construct query results as XML documents or fragments, and a new XML type can be used to store and query XML documents.

3. RELATION HISTORY IN XML

Table 1 describes the history of employees as they would be viewed in traditional transaction-time databases [2] using a temporally ungrouped representation, where `empno` is the key of the table and remains invariant in the history.

empno	salary	title	deptno
1001 1995-01-01:1996-12-31	60000 1995-01-01:1995-05-31	Engineer 1995-01-01:1995-09-30	d01 1995-01-01:1995-09-30
	70000 1995-06-01:1996-12-31	Sr Engineer 1995-10-01:1996-01-31	d02 1995-10-01:1996-12-31
		Tech Leader 1996-02-01:1996-12-31	

Figure 1: Temporally grouped history of employees

```
<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>
  <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 2: The history of the employee table is published as `employees.xml`

(In the remainder of this paper, our granularity for time is a day; however, all the techniques we present are equally valid for any granularity used by the application. For finer granularity, techniques in [23] can be used. Furthermore, throughout this paper, we assume that relation keys remain invariant.) With this approach, any change in an attribute value will lead to a new history tuple. The drawback for this representation is that, i) redundant information is present across tuples, e.g., the salary value "70000" is repeated in the last three tuples; and ii) temporal queries need to frequently coalesce tuples. Temporal coalescing is a source of complications in temporal databases, which is complex and hard to support in SQL. For instance, a temporal coalescing query can take more than 20 lines of codes in SQL:1992, and the best performance of coalescing on RDBMS is quadratic [24].

These problems can be overcome or reduced using a representation where the timestamped history of each attribute is grouped under the attribute [3] (Figure 1), i.e., value equivalent attribute histories are grouped if the intervals are adjacent or overlap. While this nested representation is hard to be represented in a flat table, it can be naturally represented by an XML-based hierarchical view shown in Figure 2. We will call these *H-documents* (or *H-views* when these are virtual representations). The root element in an H-document represents the history of the corresponding table (i.e., the creation and deletion time of a table), and its child elements represent the grouped history of attribute values. Each element in an H-document is assigned two 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. Note that there is a *temporal covering constraint* that the interval of a parent node (table history) always covers that of its child nodes (attribute histories). The H-document also has a simple and well-defined schema.

Q1: Snapshot(single object): find the salary of employee '100002' on 1993-05-16;
Q2: Snapshot: find the average salary of employees on 1993-05-16;
Q3: History(single object): find the salary history of employee '100002';
Q4: History: find the total number of salary changes;
Q5: Temporal slicing: find the number of employees whose salary was more than 60K between 1993-05-16 and 1994-05-16;
Q6: Temporal join: find the maximum salary increase over a two years period after 04/01/2001;
Q7: Temporal join: find the manager's empno for each employee;
Q8: Temporal join and snapshot: find the average salary in dept 'd001' on 1990-01-01;
Q9: Temporal Join and snapshot: find the youngest employee whose current salary is more than 100K.

Table 2: Sample temporal queries on archived history

Our H-documents use a temporally grouped data model [3]. Clifford, et al. [3, 25, 26] show that temporally-grouped models are more natural and powerful than temporally ungrouped ones. One benefit of our approach is that it greatly reduces the need for coalescing, since an attribute history is grouped in the data model. Another significant benefit is the effectiveness of expressing complex temporal queries with XQuery, as discussed next.

4. TEMPORAL QUERIES IN XQUERY

One key advantage of our approach is that powerful temporal queries can be directly expressed in XQuery.

Table 2 lists common temporal queries which we can support using XQuery, and SQL queries on relational approaches (which will be discussed in next two sections).

Due to limit of space, we choose some queries in Table 2 to illustrate the XQuery support.

Q2. *Snapshot: find the average salary of employees on 1993-05-16:*

```
let $s := doc("employees.xml")/employees/employee
    /salary[@tstart <="1993-05-16"
    and @tend >= "1993-05-16"]
return <avg_salary>{avg($s)}</avg_salary>
```

Q3. *History: find the salary history of employee '100002':*

```
let $s := doc("employees.xml")/employees
    /employee[empno='100002']/salary
return <salary_history>{$s}</salary_history>
```

Q5. *Temporal slicing: find the number of employees whose salary was more than 60K between 1993-05-16 and 1994-05-16:*

```
let $e := doc("employees.xml")/employees/
    employee/salary[. >= 60000 and
   (@tstart >= "1993-05-16" and
    @tstart < "1994-05-16" or
    @tend > "1993-05-16" and
    @tend <= "1994-05-16")]
return <count>count($e)</count>
```

Q8. *Temporal join: find the average salary in department 'd001' on 1990-01-01:*

```
let $s := document("employees.xml")/employees/
 employee[deptno[@tstart < "1990-01-01" and
                 @tend< "1990-01-01"] = "d001"]
         /salary[@tstart <="1990-01-01" and
                 @tend >= "1990-01-01"]
return <avg_salary>{avg($s)}</avg_salary>
```

5. IMPLEMENTATION: NATIVE XML DBMS

The simplest approach for implementing the archival and querying approach we have described is to use native XML DBMSs such as [27, 28], that are fast-maturing into reliable and easy-to-use systems. These systems are quite different from commercial OR-DBMS that are now being extended with XML support, insofar as they only support XML documents as their logical schema model, and they use text-oriented storage at the physical level. (A few native XML DBMSs are instead using DOM trees and OODBMS-based approaches.)

The smallest logical unit of storage for these systems is an XML document. However, documents with same schema can be stored in a collection, and manipulated as a set. All XML databases support XPath queries, and XQuery are also supported by vendors such as X-Hive and Tamino. Indexes can also be built on values, keywords, and names.

Advantages of native XML databases include: i) any XML node can be preserved; ii) mapping from schema to storage is not needed; iii) XML documents with any schema (including text-centric documents) can be stored; and iv) native XML query language (e.g., XPath and XQuery) interfaces are provided.

Our experience with the above mentioned systems was a pleasant one since we found them easy to learn and convenient to use. For example, with X-Hive, we were able to load our test-bed (991MB in size) in 10 minutes and started searching the database with XQuery soon after that. However, in terms of performance, these systems suffer from the scalability limitations that will be discussed in Section 9. In order to address these issues we have investigated the use of commercial SQL:2003-compliant OR-DBMS and the several alternative implementation architectures that are available using these systems. These are discussed next.

6. ARCHIS: MAPPING INTO RELATIONS

In ArchIS (the Archival Information System) (Figure 3), each H-document is stored in the database as internal H-tables, and XQuery is mapped into SQL/XML queries based on the mapping relationships between XML view and these H-tables. For each table in the current relational database we build a static key table and several attribute history tables.

The Static Key Table:

```
employee_static(empno, tstart, tend)
```

The static key table basically stores the key and other invariant values of each instance, such as name and sex, if

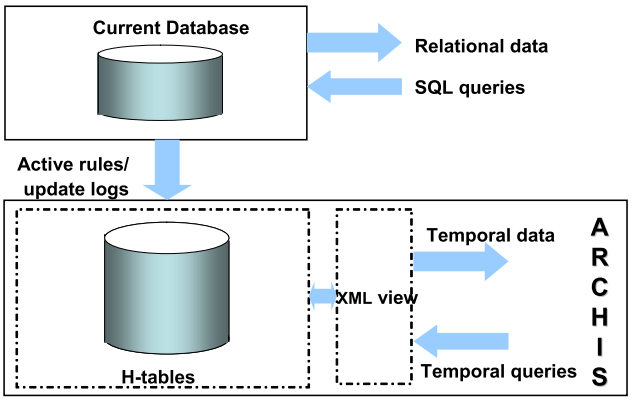

Figure 3: ArchIS: Archival Information System

any. Since `empno` will not change along the history, the period `(tstart, tend)` in the static key table also represents the valid period of the employee. The use of keys is for easy joining of all attribute histories of an object such as an employee.

Attribute History Tables:

```
employee_salary(empno, salary, tstart,tend)
employee_title (empno, title,  tstart,tend)
employee_deptno(empno, deptno, tstart,tend)
```

An attribute history table is built for each attribute to store the changing history of such attribute. The values of `empno` in the above tables are the corresponding key values, thus indexes on such `empno` can efficiently join these relations.

Our design builds on the assumption that keys (e.g., `empno`) remain invariant in the history. Otherwise, a system-generated surrogate key can be used.

Temporal queries supported by XQuery in Section 4 can be mapped into equivalent SQL/XML statements on H-tables. The mapping consists in 5 major steps: i) identification of variable range; ii)generation of join conditions; iii) generation of the WHERE conditions, iv) translation of built-in functions, and v) output generation. The detailed algorithm is discussed in [29]. In the following, we demonstrate several SQL/XML queries automatically translated from XQuery:

Q2H. *Snapshot: find the average salary of employees on 1993-05-16:*

```
SELECT XMLElement(name 'avg_salary', s.salary)
FROM employee_salary s
WHERE s.tstart <= DATE '1993-05-16'
  AND s.tend >= DATE '1993-05-16';
```

Q3H. *History: find the salary history of employee '100002':*

```
SELECT XMLElement(Name "salary",
    XMLAttributes (s.tstart AS "tstart",
     s.tend AS "tend"), s.salary)
FROM employee_salary s
WHERE s.empno = '100002';
```

Q5H. *Temporal slicing: find the number of employees whose salary was more than 60K between 1993-05-16 and 1994-05-16:*

```
SELECT count(s.empno)
FROM employee_salary s
WHERE s.salary >= 60000
  AND(s.tstart >= DATE '1993-05-16'
      AND s.tstart < DATE '1994-05-16'
      OR s.tend > DATE '1993-05-16'
      AND s.tend <= DATE '1993-05-16');
```

Q8H. *Temporal join: Find the average salary in department 'd001' on 1990-01-01:*

```
SELECT avg(s.salary)
FROM employee_salary s, employee_deptno d
WHERE  d.deptno = 'd001'
  AND  d.empno = s.empno
  AND  d.tstart <= DATE '1990-01-01'
  AND  d.tend > DATE '1990-01-01'
  AND  s.tstart <= DATE '1990-01-01'
  AND  s.tend >= DATE '1990-01-01';
```

The mapping approach here described has been implemented in ArchIS system [30] using the architecture shown in Figure 3. ArchIS uses lightweight implementation strategy that exploits the simple and well-defined mappings between our external views and internal relations, achieving performance better than other generic tools (e.g, [31]) for translating XQuery into SQL on arbitrary XML-published RDBMS, which will be discussed in Section 9.1. ArchIS also supports efficient structuring and tagging of the output using the constructs provided by SQL/XML [32, 22]. Temporal clustering, compression, and other advanced features implemented in ArchIS were discussed in [30], and are outside the scope of this paper.

7. USING NESTED TABLES

Nested relations are part of the latest SQL:2003 standards, and are also supported by some commercial database vendors [20, 21]. A temporally grouped representation, similar to that used with XML, can be exactly represented by nested relational schema. Based on such nested tables, temporal queries can be written by SQL:2003, without any extension to its current standards.

For instance, for our employee history example, we can use the following schema containing the nested table (H-table for short, or H-view if it is a nested view) n_employee:

```
CREATE TYPE salary_typ AS OBJECT(
salary    NUMBER(7),
tstart    DATE,
tend      DATE );
CREATE TYPE salary_tbl AS TABLE OF
salary_typ;
...
CREATE TABLE
n_employee(
empno    VARCHAR2(8),
tstart   DATE,
tend     DATE,
n_salary salary_tbl,
n_title  title_tbl,
n_deptno deptno_tbl)
NESTED TABLE n_salary STORE AS n_salary,
NESTED TABLE n_title  STORE AS n_title,
NESTED TABLE n_deptno STORE AS n_deptno;
```

Temporal queries supported by XQuery in Section 4 are now written by SQL:2003 as follows.

Q2N. *Snapshot: find the average salary of employees on 1993-05-16:*

```
SELECT avg(s.salary)
FROM n_employee e, TABLE(e.n_salary) s
WHERE s.tstart <= DATE '1993-05-16'
  AND s.tend >= DATE '1993-05-16';
```

Q3N. *History: find the salary history of employee '100002':*

```
SELECT s.salary, s.tstart, s.tend
FROM n_employee e, TABLE(e.n_salary) s
WHERE e.empno = '100002';
```

Q5N. *Temporal slicing: find the number of employees whose salary was more than 60K between 1993-05-16 and 1994-05-16:*

```
SELECT count(e.empno)
FROM n_employee e, TABLE(e.n_salary) s
WHERE s.salary >= 60000
  AND (s.tstart >= DATE '1993-05-16'
       AND s.tstart < DATE '1994-05-16'
       OR s.tend > DATE '1993-05-16'
       AND s.tend <= DATE '1993-05-16');
```

Q8N. *Temporal join: Find the average salary in department 'd001' on 1990-01-01:*

```
SELECT avg(es.salary)
FROM n_employee e, TABLE(e.n_salary) es,
    TABLE(e.n_deptno) ed
WHERE  ed.deptno = 'd001'
AND  ed.tstart  <= DATE '1990-01-01'
AND  ed.tend  > DATE '1990-01-01'
AND  es.tstart  <= DATE '1990-01-01'
AND  es.tend  >= DATE '1990-01-01';
```

Although they are not supported by all vendors, the nested table schemas provide a natural middle ground between XML and the traditional relational databases.

8. OR-DBMS EXTENSIONS FOR XML

OR-DBMS vendors are extending current technology to support XML through the SQL/XML initiative [32, 22]. SQL/XML publishing functions can now publish XML documents directly from SQL queries, and a new data type *XMLType* is used to store XML documents. The XMLType can use two kinds of storage, unstructured storage with CLOB for any type of XML documents (normally for text-centric XML documents), and structured storage for well structured XML documents.

Only the structured storage approach is of interest for our problem. In this approach, documents are 'shredded' into a set of tables, whereby the indexing and optimization techniques developed for queries on relational tables can now be used for XML. Different solutions are being pursued by different vendors. For instance, in Oracle XML DB [33], an XML document is decomposed into a set of "hidden" index-organized nested tables by adding mapping annotations in the XML Schema of the documents. To manage our H-documents with this system, we first created the XML Schema for employees documents, and then register the schema in the database. As a result of this operation, mapping rules are generated automatically from XML Schema and nested tables. The employees table is created by the following statement:

```
CREATE TABLE employees of xmltype xmlschema
"http://localhost:8080/public/employees.xsd"
element "employees";
```

While XPath queries are supported in the current release of the system, XQuery is not. Complex temporal queries can however be specified through a combination of SQL and XPath. Two example queries are shown as follows:

Q2R. *Snapshot: find the average salary of employees on 1993-05-16:*

```
SELECT avg(extractValue(value(s),'/salary/text()'))
FROM employees e,
TABLE(XMLSequence(extract(value(e),
'/employees/employee/salary[@tstart <= "1993-05-16"
and @tend >="1993-05-06"]'))) s;
```

Q3R. *History: find the salary history of employee '100002':*

```
SELECT extract(value(e),'/employees/employee/salary')
FROM employees e
WHERE existsNode( value(e),
 '/employees/employee[empno="100002"]') = 1;
```

Q5R. *Slicing: find the number of employees whose salary was more than 60K between 1993-05-16 and 1994-05-16:*

```
SELECT count(extractValue
((extract(value(s),'/employee'))
FROM employees e,
TABLE (XMLSequence(extract(value(e),
'/employees/employee[salary/text() >= 60000
and (salary/@tstart >= "1993-05-16"
    and salary/@tstart < "1994-05-16"
    or  salary/@tend < "1993-05-16"
    and salary/@tend >="1990-01-01")]' ))) s;
```

Observe that XPath expressions can be used through the `extract` and `extractValue` functions.

Q8R. *Temporal join: Find the average salary in department 'd001' on 1990-01-01:*

```
SELECT avg(extractValue
    ((extract(value(s),'/salary/text()')))
FROM employees e,
    TABLE (XMLSequence(extract(value(e),
      '/employees/employee[deptno = "d001"
       and deptno/@tstart <= "1990-01-01"
       and deptno/@tend >="1990-01-01"]
      /salary[@tstart <= "1990-01-01"
      and @tend >="1990-01-01"]' )))  s;
```

While the situation is fast improving, the XML extensions provided by commercial vendors leave a lot to be desired, as illustrated by the lack of XQuery and other problems we encountered with Oracle XML DB[33]. For instance, the nested tables automatically generated are hidden from users, and indexes on them are not easy to generate. (Indexes can only be built after those tables are found by searching among the system tables.) Even when introduced, indexes might be ignored by the optimizer, which frequently neglects them and ends up choosing a very inefficient query plan. These problems are limited to XML extensions, and suggest a lack of maturity and robustness in such extensions. Due to these problems and the lack of direct support for XQuery, no performance results will be reported for this approach.

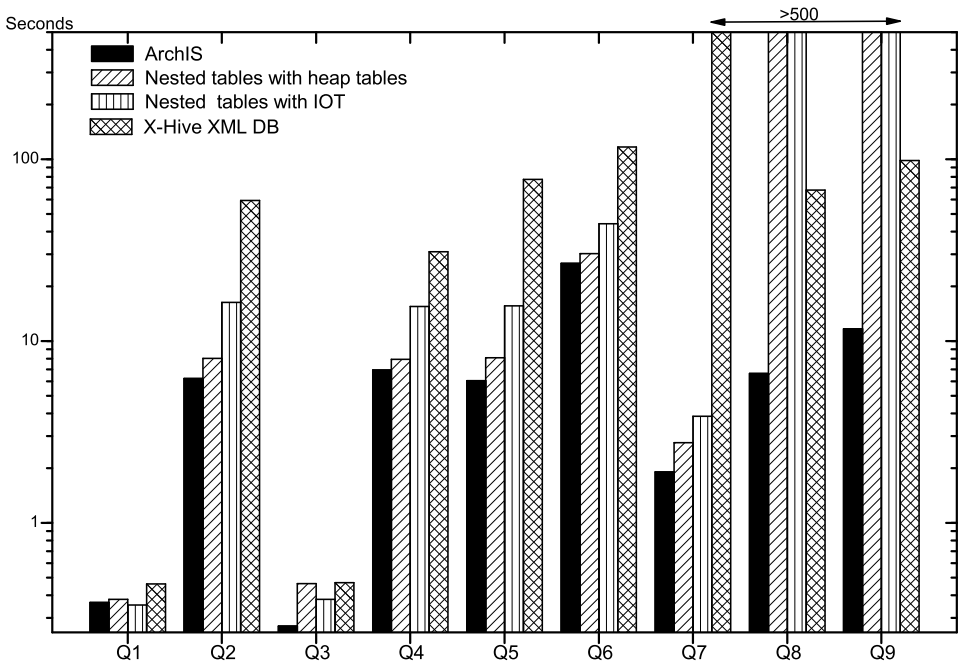

Figure 4: Performance of different schemes for archiving relational database history

9. PERFORMANCE RESULTS

We investigate the performance of the following four systems for archiving RDBMS database history: i) ArchIS; ii) nested tables, with nested tables stored as heap tables; iii) nested tables, with nested tables stored as index organized tables; iv) native XML database X-Hive/DB.

We use a simulated temporal data set from the history of 900,000 employees over 17 years. The data set models increases of salaries, changes of titles, and changes of departments. The total size of the published XML documents from the history data is 991MB.

The experiments are performed on a Pentium IV 3GHz PC with Windows XP Professional, 512MB memory and a 200GB ATA hard drive. The DBMSs we use are the latest release of major commercial database server.

We run the queries from Table 2 for the experiments. Proper indexes are built for each scheme for best performance.

Figure 4 shows the performance comparison among the four systems[1]. ArchIS shows performance advantage over all other approaches. Especially, the performance advantage on ArchIS over native XML DB is most significant. For example, Q2 is 9.5 times faster on ArchIS than on X-Hive, Q3 4.5, Q4 4.5, Q5 12.8, Q6 4.4, Q8 10, and Q9 8.4 times faster. Q7 on X-Hive even runs "out of the top".

ArchIS also wins over heap-based nested tables (index-organized one is even much slower than heap-based nested one). However, queries on nested tables often require more joins which can lead to slow performance for join queries. For example, Q8 and Q9 run "out of the top" on nested tables.

Nested tables based approach has considerable performance advantage over native XML DB on queries Q1-Q7, except for queries Q8 and Q9, which require joins between child tables.

Overall, ArchIS is clearly the winner of all these approaches.

[1]Queries reaching the top line of the graph are more than 500 seconds and run out of physical memory.

9.1 XTABLES

We have also explored the use of XTABLES [31] - a general XML-view based approach - in supporting our temporally grouped historical views on the stored H-tables. Using XTABLES, users can query the history of database relations using only commercially available software, with no need to install ArchIS. However, this approach also encounters several limitations. The first is that ArchIS' library of special functions designed for temporal queries can not be easily incorporated. Thus some temporal queries will become harder to express and less efficient to execute. The second problem is that even the queries that can be easily expressed without any function from the temporal library become significant less efficient to compile and execute.

Figure 5 (a) and (b) show the ratios of the query execution time and translation time of XTABLES over ArchIS respectively. (Queries Q6 and Q7 are ignored since XTABLES "run out of the top" for these two queries.) The average query translation cost for XTABLES is 4.8 seconds, about 100 times slower than that of ArchIS. Furthermore, the translated queries with XTABLES are quite complex and often not optimized, which lead to slower performance. Figure 5 (a) shows that all queries are slower on XTABLES than on ArchIS, especially for Q1 and Q3, which are more than 13 times slower.

Naturally, these queries were executed on a DBMS that supports XTABLES, rather than the DBMS that support nested relations which was used in the experiments shown in Figure 4.

However the performance of the two systems in running the SQL queries on base relations is close enough to conclude that the drop in performance is due to the translation

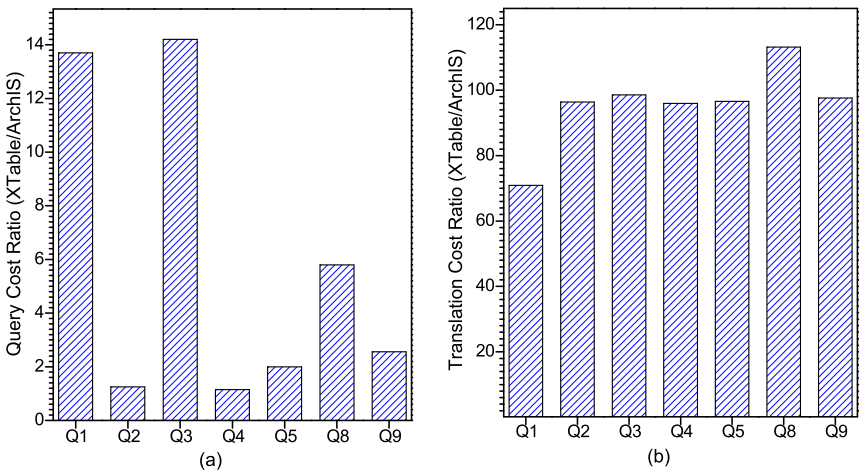

Figure 5: Comparison of query execution and translation cost between XTABLES and ArchIS: (a) query cost ratio; (b) translation cost ratio

from XQuery to SQL queries, rather than the different performance of the two DBMS in executing SQL queries.

Therefore the highly specialized XML-publishing and view-support mechanisms provided by ArchIS are significantly more efficient than the very general ones provide by systems such as XTABLES.

10. CONCLUSION

XML provides a powerful environment for publishing and querying not only the current content of relational databases but also their histories. This is because XML is effective at representing and querying temporal information due to its ability to support a temporally grouped data model and a powerful extensible language such as XQuery. Thus effective temporal representations can be achieved in XML without requiring changes in the current standards—an important point since the temporal database experience has shown that this would be an uphill battle.

In general, we found that temporal information, that a long stream of database research [2] proved difficult for relational databases and its query languages, can now be handled quite naturally using XML and its query languages. The significance of this finding is underscored by the fact that the solution is quite general, and it is also effective at representing and querying the evolution history of multi-version XML documents. In [34], we have shown how the history of multi-version documents can be represented by simply timestamping their elements with their time span, and using queries similar to those used on the histories of relational tables. The approach has been applied successfully to real-life XML documents, such as the UCLA course catalog, the CIA World FactBook and W3C standard specifications. Therefore, we have a solution that is general and very flexible, since it can be customized via different temporal libraries.

While XML is clearly preferable at the logical level, it is clear that relational databases are still preferable in terms of performance and scalability. Indeed we investigated four alternative solutions that use: i) Native XML DBMS, ii) Flat relational tables and SQL/XML, iii) SQL:2003 nested tables, and iv) OR-DBMS extended with XML support.

The approach of shredding the historical views into relational tables proved to be the overall winner, in terms of performance and reliability. Relational tables win over other approaches, in which the running times of some of their queries went 'through the roof'. Our experiments showed that nested tables only slightly improved performance on one query.

In many applications performance and scalability do not represent pressing issues. This is, for instance, the case of the real-life examples studied in [34] where native XML databases proved effective at managing multi-version XML documents and answering complex queries on them. In [34] we demonstrated our approach on several interesting documents, including the UCLA course catalog, and the CIA World FactBook [35], and the W3C XLink specifications [36]. These documents happened to be only a few megabytes in size, and native XML databases support them efficiently. Our experiments, however, show that the approach of supporting logical historical XML views as stored relational tables is clearly the best, since it combines the advantages of XML at the logical level, with the better performance of relational tables at the physical level.

In the course of this research we had an opportunity to experiment with the XML extensions now being introduced by database vendors. At the time of this writing, these XML extensions are not very mature, and need more time before they become robust and efficient. Even so, it is not clear whether they will ever be able to outperform our mapping into relational tables and SQL/XML, that we designed and optimized for this specific application. Indeed, we conjecture that the ArchIS architecture shown in Figure 3 will remain the solution of choice for archiving, publishing and querying the history of relational databases.

11. 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] J. Clifford, A. Croker, F. Grandi, and A. Tuzhilin. On Temporal Grouping. In *Recent Advances in Temporal Databases*, pages 194–213. Springer Verlag, 1995.
[4] XQuery 1.0: An XML Query Language. http://www.w3.org/XML/Query.
[5] S. Kepser. A Proof of the Turing-Completeness of XSLT and XQuery. In *Extreme Markup Language*, 2004.
[6] F. Grandi and F. Mandreoli. The Valid Web: An XML/XSL Infrastructure for Temporal Management of Web Documents. In *ADVIS*, 2000.
[7] T. Amagasa, M. Yoshikawa, and S. Uemura. A Data Model for Temporal XML Documents. In *DEXA*, 2000.
[8] C.E. Dyreson. Observing Transaction-Time Semantics with TTXPath. In *WISE*, 2001.
[9] S. Zhang and C. Dyreson. Adding Valid Time to XPath. In *DNIS*, 2002.
[10] D. Gao and R. T. Snodgrass. Temporal Slicing in the Evaluation of XML Queries. In *VLDB*, 2003.
[11] P. Buneman, S. Khanna, K. Tajima, and W. Tan. Archiving scientific data. *TODS*, 29(1):2–42, 2004.
[12] R. T. Snodgrass. *The TSQL2 Temporal Query Language*. Kluwer, 1995.
[13] J. Chomicki, D. Toman, and M.H. Böhlen. Querying ATSQL Databases with Temporal Logic. *TODS*, 26(2):145–178, June 2001.
[14] C. Zaniolo, S. Ceri, C.Faloutsos, R.T. Snodgrass, V.S. Subrahmanian, and R. Zicari. *Advanced Database Systems*. Morgan Kaufmann Publishers, 1997.
[15] A. Steiner. *A Generalisation Approach to Temporal Data Models and their Implementations*. PhD thesis, ETH Zurich, 1997.
[16] Oracle Flashback Technology. http://otn.oracle.com/deploy/availability /htdocs/flashback_overview.htm.
[17] D. Lomet, R. Barga, M. F. Mokbel, G. Shegalov, R. Wang, and Y. Zhu. Transaction Time Support Inside a Database Engine. In *ICDE*, 2006.
[18] F. Wang and C. Zaniolo. Publishing and Querying the Histories of Archived Relational Databases in XML. In *WISE*, 2003.
[19] Database Languages SQL, ISO/IEC 9075-*:2003.
[20] SQL 2003 Standard Support in Oracle Database 10g, otn.oracle.com /products/database/ application_development/pdf/SQL_2003_TWP.pdf.
[21] Informix Universal Server. http://www.ibm.com/informix.
[22] ISO. Information technology - Database languages - SQL Part 14: XML-Related Specifications. 2003.
[23] C. S. Jensen and D. B. Lomet. Transaction Timestamping in Temporal Databases. In *VLDB*, 2001.
[24] M. H. Böhlen, R. T. Snodgrass, and M. D. Soo. Coalescing in Temporal Databases. In *VLDB*, 1996.
[25] J. Clifford. *Formal Semantics and Pragmatics for Natural Language Querying*. Cambridge University Press, 1990.
[26] J. Clifford, A. Croker, and A. Tuzhilin. On Completeness of Historical Relational Query Languages. *TODS*, 19(1):64–116, 1994.
[27] X-Hive/DB. http://www.x-hive.com.
[28] Tamino XML Server. http://www.tamino.com.
[29] F. Wang, X. Zhou, and C. Zaniolo. Using XML to Build Efficient Transaction-Time Temporal Database Systems on Relational Databases. Technical Report 81, TimeCenter, www.cs.auc.dk/TimeCenter, Mar. 2005.
[30] ArchIS: the Archival Information Systems Project. http://wis.cs.ucla.edu/projects/archis/index.html.
[31] J.E. Funderburk, G. Kiernan, J. Shanmugasundaram, E. Shekita, and C. Wei. XTABLES: Bridging Relational Technology and XML. *IBM Systems Journal*, 41(4), 2002.
[32] SQL/XML. http://www.sqlx.org.
[33] Oracle XML. http://otn.oracle.com/xml/.
[34] F. Wang, X. Zhou, C. Zaniolo, and H. Moon. Managing Multi-version Documents and Historical Databases: a Unified Solution Based on XML. In *WebDB*, 2005.
[35] CIA. *the world factbook. http://www.cia.gov/cia/publications/factbook.*
[36] W3C. XML Linking Language (XLink). http://www.w3.org/TR/xlink/.