TeXOR: Temporal XML Database on an Object-Relational Database System

Kjetil Nørvåg,* Marit Limstrand, and Lene Myklebust
Department of Computer and Information Science
Norwegian University of Science and Technology
7491 Trondheim, Norway

Abstract

Storage costs are rapidly decreasing, making it feasible to store larger amounts of data in databases. This also makes it possible to store previous versions of data in the databases, instead of only keeping the last version. Recently, the amount of data available in XML has been rapidly increasing. In this paper, we describe *TeXOR*, a temporal XML database system built on top of an object-relational database system. We describe the TXSQL query language used in TeXOR for querying temporal XML documents stored in the system, discuss storage alternatives for XML documents in such a system, and some details about the implementation of the current TeXOR prototype.

1 Introduction

Storage costs are rapidly decreasing, making it feasible to store larger amounts of data in databases. This also makes it possible to store previous versions of data in the databases, instead of only keeping the last version. Recently, the amount of data available in XML has been rapidly increasing. One of the advantages of XML is that a document itself contains information that is normally associated with a schema. This makes it possible to do more precise queries, compared to what has been possible with unstructured data. It also has advantages for long-term storage of data: even though the schema has changed, the data itself can contain sufficient information about the contents, so that meaningful queries can be applied to the data.

When previous versions of data are stored in the database, it is possible to search in the historical (old) versions, retrieve documents that were valid at a certain time, query changes to documents, etc. Our main context is storage of XML documents, and querying versions of these. Thus, we will in this paper restrict the discussion to *transaction-time*, i.e., versions are timestamped with the commit time of the transaction that created the versions (in contrast to *valid time*, where a time interval is associated with every tuple/object, denoting when the object is valid in the modeled world).

A temporal database can be realized either through the *integrated* approach, in which the internal modules of a DBMS (database management system) are modified or extended to support time-varying data, or a *stratum* approach, in which a layer converts temporal query language statements into conventional statements, executed by an underlying database system [10]. Although we consider the

*Corresponding author. Email address: Kjetil.Norvag@idi.ntnu.no

integrated approach as the long-term solution, it is not appropriate for storing and querying temporal XML data *today*, for the simple reason that no such system with product quality exists. Thus, for the time to come, a stratum approach is the most adequate solution. The stratum approach also makes it possible to utilize the existing support for XML data management in the system (including XPath based queries), in addition to benefit from the quality of the system from a database point of view.

In order to demonstrate the usefulness of a temporal XML databases in general, and gain experience from actual use of such systems, we have implemented *TeXOR*, a temporal XML database system on top of the commercial object-relational database system Oracle. In this paper, we describe the functionality of TeXOR, and how we implemented it on top of Oracle. It should be emphasized that although Oracle is used in our prototype, the use of JDBC between TeXOR and Oracle means that it should not be too difficult integrating TeXOR with another system providing support for XML data. It should also be noted that even if we call TeXOR a temporal XML database system, it can also be used for storing general temporal data. However, the design decisions have been made and optimized based on the assumption that temporal XML storage will be its most important application.

The organization of the rest of this paper is as follows. In Section 2 we give an overview of related work. In Section 3 we give an overview of storage of XML documents. In Section 4 we describe the functionality provided by TeXOR. In Section 5 we describe the design and implementation of TeXOR. Finally, in Section 6, we conclude the paper and outline issues for further research.

2 Related Work

A model for representing changes in semistructured data (DOEM) and a language for querying changes (Chorel) were presented by Chawathe et al. in [3, 4]. Chorel queries were translated to Lorel (a language for querying semistructured data), and can therefore be viewed as a stratum approach. The work by Chawathe et al. has later been extended by Oliboni et al. [15].

In order to realize an efficient temporal XML database system, several issues have to be solved, including efficient storage of versioned XML documents, efficient indexing of temporal XML documents, and temporal XML query processing. Storage of versioned documents is studied by Marian et al. [11] and Chien et al. [5, 6, 16]. Chien et al. also consider access to previous versions, but only snapshot retrievals. Temporal query processing is discussed in [12, 13].

An approach that is orthogonal, but related to the work presented in this paper, is to introduce valid time features into XML documents. One such approach is presented by Grandi and Mandreoli in [8].

Obviously, our work is heavily inspired by previous work in extending SQL to the temporal domain, for example TSQL2 [17], and previous systems based on the stratum approach. A good overview of the stratum approach, and systems using it, is given in [19].

Other relevant work includes work on temporal document databases [1], temporal object query languages [7], and temporal object database systems [18].

3 Storage of XML documents

In order to put our work in context, we will in this section give an overview of alternative approaches for storing XML documents, and describe in more detail the XML support in Oracle, the system that is used by TeXOR.

3.1 Approaches

At first, XML documents were mostly stored as ordinary text files. This is still common. However, in many cases the features provided by DBMSs are desired, for example support for transactions, recovery and querying. As a result several approaches to support these features have been introduced.

Before we describe the approaches in more detail, it can be useful to have in mind that what will be the best approach for a given application, depends very much on whether the XML documents mostly are document-centric (this is often documents meant for human consumption, like books, papers, etc.), or data-centric (this is often documents meant for computer consumption, and that uses XML as a data transport). Many documents also have features of both categories, we call these hybrid documents.

Several approaches to XML storage in DBMSs have been proposed, and can be classified into three categories:

- **XML-enabled database systems:** Traditional database systems (typically object-relational systems), extended with support for XML documents. Examples of XML-enabled database systems are Oracle, IBM DB2, and Microsoft SQL Server. The supported features include storing/retrieving XML documents, as well as query capabilities. Examples of XML-enabled database systems are Oracle, IBM DB2, and Microsoft SQL Server. In general, the XML documents can be stored in two ways:
  - *In CLOBs:*[1] Each document is essentially an attribute in a tuple. In order to make it possible to efficiently query the documents, the contents is indexed (for example in a text index), in order to avoid scans of all the documents stored in one relation. One of the most important advantages of this approach is efficient retrieval of single documents (no need for reconstruction/join operations). This is often a good approach for document-centric documents, where retrievals of complete documents/parts of documents are frequent.
  - *Mapped to relations:* The documents are mapped to relations, for example using a *table-based mapping* or a *object-relational mapping* (cf. [2] for a more detailed description of the mapping techniques). This is often a good approach for data-centric documents, where we usually are not interested in retrieving the whole documents, only parts of the data stored in it. These documents also often have a regular structure because they are computer-generated, making the mapping easier.
- **Native XML database systems:** These systems define a data model for XML documents (for example the DOM or XPath data model), and all retrieval/storage is based on this model. This makes it possible to support XML-specific features better and more efficiently than the other approaches. Examples of native XML database systems are the Tamino XML Server, eXcelon Extensible Information Server, and X-hive/DB.
- **Middleware:** Transfer data between XML documents and databases, usually using protocols as ODBC/JDBC. One variant is transferring XML documents into data in tables. Another approach is simply to store the XML documents as CLOBs in the database system. Often, the support for queries when this approach is used, is limited.

3.2 XML support in Oracle

Support for XML documents in Oracle has evolved from a middleware approach in Oracle8i, where the documents where stored outside the database, to XML support and the new XML datatype in the

[1] A CLOB is a large object that holds text data, and is similar to a BLOB, which can store any kind of binary data.

database in Oracle9i Release 1 (this is the version that TeXOR is based on).[2]

Oracle supports both storage of documents in CLOBs, as well as mapping to relations. Our intended application area is mostly document-centric, and exact round-trip (supporting exact round-trip means that a document retrieved from the database is exactly the same as it was when it was stored) of documents is required. As a result, CLOB storage is the most appropriate of the two alternatives, and here we restrict the description to this alternative.

A document stored in a CLOBs in an Oracle relation, is (logically) essentially stored in an attribute of the relation, where the attribute is of *XMLType*. It should be mentioned that one disadvantage of this approach, is that even if only a small part of a document is updated, the whole document has to be updated. However, this is only a problem for very large documents, and many of these can be expected to be fairly static. The XMLType has a set of functions operating on the CLOB, of which the most important for our purpose are `createXML()` for storing documents, and `getClobVal()` for retrieving an XML document from an attribute.

Two indexing techniques can be used in Oracle for indexing XMLType columns: 1) functional indexes, and 2) text indexes. Functional indexes can be declared as SQL/XPath queries, so that subsequent uses of the particular query queries do not involve parsing a lot of documents.

In order to query XML documents stored in XMLType attributes using SQL, it is possible to use XPath expressions in the query. It is also possible to use the `CONTAINS` and `WITHIN` functions for ordinary text searching.

4 TeXOR Functionality

In this section we describe the functionality provided by TeXOR. We describe the query language, and how to access the system through the terminal interface and the API.

4.1 The TXSQL query language

The TXSQL query language is an extension of the XPath extended SQL query language that Oracle uses, and it also contains support for declaring and querying temporal tables.

4.1.1 Creating tables

A temporal tables is created by using the `t_time` modifier in a `create table` statement. For example, the following statement creates a temporal table where each row contains an oid and an XML document:

```
create table usertable t_time (
    oid number(8),
    xml sys.xmltype)
```

The result of this statement will be `Table created`. Each tuple in such a table contains timestamp(s), but this is transparent to the user. As will be shown later, the data might be stored in more than one Oracle table, in order to make some operations more efficient.

[2] After we made our prototype implementation, Oracle9i Release 2 with even better XML support has been released.

4.1.2 Querying time in temporal tables

Although the timestamps of the tuples normally are hidden, they can be retrieved by explicitly declaring them in the query statement. The (normally hidden) column containing the timestamp is named `t_start`, and is used in `select`-part of queries when querying temporal tables. For example, issuing the following query:

```
select oid, t_start
from usertable
```

gives the following results (the timestamp consists of date and time):

```
oid      t_start
--------------------------
1        20.05.02 12:15:00
2        23.05.02 08:05:00
```

If `t_start` is not included explicitly, the timestamp is not returned. This also applies when using "*". If only "*" is used to retrieve all columns, the timestamp is not returned. This is illustrated by the following query:

```
select *
from usertable
```

which gives the following results:

```
oid      xml
------------------
1        xmltype()
2        xmltype()
```

As can be seen, no timestamp is returned. This also illustrates another aspect of XML support in Oracle: the result is a set of `xmltype()`. In order to retrieve the XML data itself, which is what we normally would want, `xml.getclobval()` has to be used.

A version in a temporal database is valid in a certain time interval, from `t_start` to `t_end`. Querying the end time `t_end` can be performed similar to querying for the start time as shown above, and the end timestamp is essentially the start timestamp of the next version.

4.1.3 Querying particular versions

When querying a temporal table, we often want to restrict the query to particular versions. This could for example be the first version, versions valid at a particular time, or all versions valid during a particular time interval. In TXSQL, the `version` function is used in the `where` part of the `select` statements in order to determine which versions should be involved in the query. In order to specify versions, one of the following parameters is used (also illustrated in Figure 1, which shows which versions will be returned by using the different parameters):

- `all`: all versions.
- `first`: the first version.

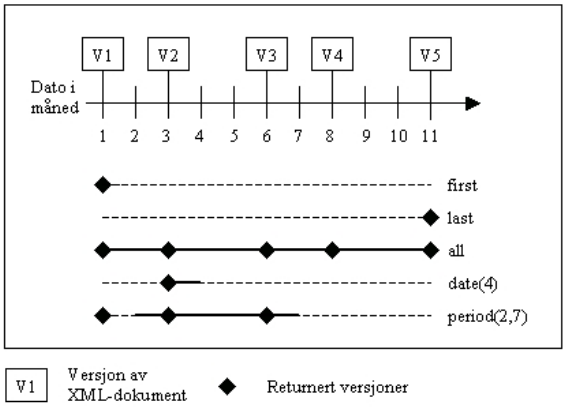

Figure 1: Some version parameters and the versions they will return.

- `last`: the last version.
- `date(timestamp)`: uses a particular timestamp as parameter (we use the DATE type in Oracle for storing timestamps), and returns the versions valid at that time.
- `period(`$timestamp1$`, `$timestamp2$`)`: uses two timestamps as parameters, and returns all versions valid in the period from $timestamp1$ to $timestamp2$.

If `version` is not specified, only the current version will be queried, in order to be compatible with non-temporal queries.

Example of use of `version`: The following query illustrates how to retrieve all versions of the tuple with a given oid:

```
select x.xml.getClobVal()
from usertable
where oid = 123 and version(all)
```

Note that `xml` is the name of the attribute containing the XMLType data, and that the contents is retrieved by using `getClobVal()`. The result is:

```
x.xml.getClobVal()
-----------------------------------
<name>Per</name><tel>73882934</tel>
<name>Per</name><tel>56894534</tel>
```

4.1.4 Non-implemented features

In this section we give an overview of some interesting features of the TXSQL-language that are not yet implemented. This includes statements for vacuuming and control of granularity.

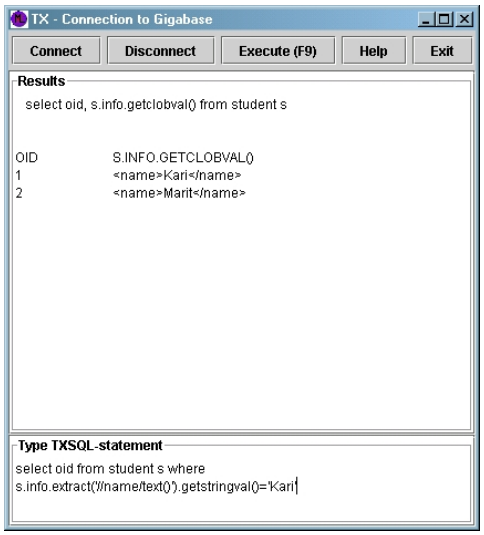

Figure 2: The TX terminal interface.

Vacuuming. Even with cheap disks, it will sometimes not be feasible to store all versions. The `vacuum(date | period)` statement is applied in order to physically delete all versions created before a particular time, or delete all versions created during a particular time period. A current version (non-deleted and most recent version) will not be removed during vacuuming even if it was created before the particular date or time period. Example:

```
alter table usertable vacuum(date(01.01.00))
```

Granularity control. It should be possible to state how close to each other two versions can be in the database by using a `granularity` statement. For example, `set granularity day(3)` says that if a new version is inserted, and it is less than 3 days since the previous version was inserted, only the last version will actually be kept. Granularity control can be useful in some applications where intermediate versions are not required to be kept (or not wanted). Such versions can for example be documents under revision/updating, where only the resulting document is really of interest. Granularity control can be an alternative to the use of traditional versioning/use of revisions. Another use of granularity is in vacuuming. Often, when versions gets older, it is not necessary to have a large number of versions created during short time interval. If versions were originally created several times a day, some months later one version for each day might suffice. This can be stated as, for example:

```
granularity_vacuum(date(01.01.98), day(5))
```

4.2 Interface

Users of TeXOR have two interfaces available to the system. One interface is a terminal interface, similar to SQL*Plus in Oracle, and the other is an API for interaction through Java programs.

4.2.1 Terminal interface

The terminal interface TX, illustrated by a screenshot in Figure 2, has a functionality similar to SQL*Plus in Oracle, but with some extended editing features. From TX the user can connect to

the database, enter an TXSQL query, and execute the query. The result from the query is displayed in the terminal window.

4.2.2 API

Even though the terminal interface is nice for testing and experimenting, when using TeXOR from a program, the API is more useful. Using the TeXOR API, the query is submitted as a string to the TeXOR object, and the result can be retrieved by calling another function in the interface.

5 Design of TeXOR

The main part of TeXOR is essentially a middleware layer between the TX/API interfaces and the Oracle DBMS. TeXOR rewrites the TXSQL expressions into SQL, and post-processes the results before they are returned to the user/application. It also manages some extra metadata, for example which tables are temporal tables. The metadata is itself stored in Oracle. In the rest of this section, we will describe in more detail how data is stored. It should again be noted that in the cases where we have several alternative approaches to choose from, the application of XML storage has been the driving decision factor.

5.1 Timestamp management

There are several alternative approaches to store the timestamp in the database. One is to store the timestamp inside the XML document. However, retrieval of the timestamp is expensive with this alternative. A better alternative in the context we are working, is to store timestamps together with identifier in a separate table. For example, if one table is used to store both historical and current versions, and the user creates a table `Usertable` with the column XML, the following table will actually be created (and managed):

```
Usertable {oid, xml}
Time_usertable {doc_id, oid, start_time, end_time}
```

The actual storage of timestamps in TeXOR will be described below.

Timestamps will in our system only be used for retrieval/querying purposes, and not for concurrency control or similar purposes. For this reason, it is not critical *when* the timestamp is decided, as long as all document versions stored during one transaction are assigned the same timestamp.[3] If the actual commit timestamp should be used, it would be necessary to postpone until commit time the updates to the table containing the timestamps. This would increase commit time considerably, so that instead, we decide transaction timestamp when the first document is stored in a transaction, and this timestamp also for the other documents stored during this transaction.

5.2 Storage of versions

As described previously, XML documents can be stored in Oracle either as 1) CLOBs or 2) by extracting documents into tables. In the TeXOR prototype, only CLOB storage is supported. The main reason for this decision, is that the CLOB alternative 1) permits exact round-trip of documents, and 2) it uses text indexes, which should enhance efficiency for typical XML-queries in our application areas, and in particular for XPath-queries which are supported by this storage alternative.

[3] This is a simplification. For a more thorough discussion on this topic, we refer to [9] and [14].

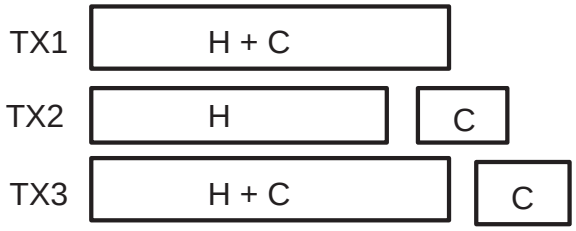

Figure 3: Storage alternatives.

As illustrated in Figure 3, there are several alternative approaches for storing a temporal relation:

- TX1: Store both historical (previous) and current versions of the data in one relation. This is the easiest alternative, and in this case the temporal table is essentially the non-temporal equivalent with additional timestamp attributes for start and end time. Although simple to realize, this approach can have problems in some queries for current versions. If a table scan is necessary, it is necessary to scan all the historical versions in addition to the current versions.
- TX2: Store historical and current versions in separate relations. Using this approach, the previous current version is moved to the historical relation when it is updated, and the new current version inserted into the current version relation. Using this approach, only start time is needed in the current version relation, and that might save some space. The most important advantage of this approach is efficient queries for current versions, as all these are stored in a separate table. This table is likely to be much smaller than the size of the historical table.
- TX3: Store current versions in a separate relation, and historical versions plus a copy of current versions in another relation. A possible advantage of this approach is that queries for current versions are efficient, and at the same time queries involving all versions are simple because only one table is involved. However, this is not likely to outweight the disadvantages of using more space, which will in particular be a problem in the case of few versions of each tuple.

Based on the informal arguments given above, we decided to use alternative TX2. This alternative also has the additional advantage of simplifying the vacuuming process, because only historical versions need to be considered. The tables that are created in the ORDBMS for one temporal table in TeXOR, are summarized as an example in Figure 4.

6 Summary

We have in this paper described stratum temporal XML database management, based on an object-relational database system. In order to provide support for querying temporal data, we designed the TXSQL query language, which is SQL extended with support for querying versions and temporal aspects. The ideas have been realized in the TeXOR prototype, where most aspects described in the paper have been integrated, except vacuuming, granularity control, and some of the version selection support.

As a result of using a stratum approach, some problems and bottlenecks are inevitable. For example, no efficient time-index for our purpose is available, and query optimization can be a problem when part of the “knowledge” is outside the database system. Thus, we suspect that for large amounts

OID	XML
3	<name>Ola</name>
4	<name>Kari</name> <tlf>555 4826</tlf>
5	<name>Lise</name> <age>26< /age>

(a) Usertable

DocOID	OID	Start_Time
3	3	20.04.02 12:45:00
1	4	24.04.02 08:05:00
2	5	03.05.02 09:00:00

(b) time_Usertable

OID	XML
1	<name>Kari</name> <tlf>555 2345</tlf>
2	<name>Lise</name> <age>24</age>

(c) Usertable_old

DocOID	Oid	Start_Time	End_Time
1	1	02.03.02 12:00:00	24.04.02 08:05:00
2	2	20.03.02 07:15:00	03.05.02 09:00:00

(d) time_Usertable_old

Figure 4: ORDBMS tables used for one TeXOR temporal table.

of data, where disk accesses are needed, query cost can be high. However, it should be kept in mind that with the improving size of main memory the effect of this is not necessarily disastrous. For example, a desktop computer today often has 512MB or more of main memory.

It should be noted that even if we call TeXOR a temporal XML database system, it can also be used for storing general temporal data. However, the design decisions have been made and optimized based on the assumption that temporal XML storage will be its most important application.

The rationale behind development of TeXOR was to gain experience in temporal XML data management, and results so far have been useful input to the V2 project, which is another temporal XML database project we are currently working on (see [14]). In the V2 project, we study more in more detail the aspects of efficiency in temporal XML databases. V2 is based on an integrated approach, where also the more low-level parts of the system are implemented.

Acknowledgments

We would like to thank Jon Olav Hauglid for help during the implementation of TeXOR, and for volunteering for the job as database administrator. We would also like to thank Gregory Cobena for providing us with temporal XML testdata.

References

[1] M. J. Aramburu-Cabo and R. B. Llavori. A temporal object-oriented model for digital libraries of documents. *Concurrency and Computation: Practice and Experience*, 13(11), 2001.

[2] R. Bourret. XML and databases, February 2002 (most recent version available at http://www.rpbourret.com/xml/XMLAndDatabases.htm).

[3] S. S. Chawathe, S. Abiteboul, and J. Widom. Representing and querying changes in semistructured data. In *Proceedings of the Fourteenth International Conference on Data Engineering*, 1998.

[4] S. S. Chawathe, S. Abiteboul, and J. Widom. Managing historical semistructured data. *TAPOS*, 5(3), 1999.

[5] S.-Y. Chien, V. J. Tsotras, and C. Zaniolo. A comparative study of version management schemes for XML documents (short version published at WebDB 2000). Technical Report TR-51, TimeCenter, 2000.

[6] S.-Y. Chien, V. J. Tsotras, and C. Zaniolo. Version management of XML documents: Copy-based versus edit-based schemes. In *Proceedings of the 11th International Workshop on Research Issues on Data Engineering: Document management for data intensive business and scientific applications (RIDE-DM'2001)*, 2001.

[7] L. Fegaras and R. Elmasri. A temporal object query language. In *Proceedings of the Fifth International Workshop on Temporal Representation and Reasoning*, 1998.

[8] F. Grandi and F. Mandreoli. The valid web: An XML/XSL infrastructure for temporal management of web documents. In *Proceedings of Advances in Information Systems, First International Conference, ADVIS 2000*, 2000.

[9] C. S. Jensen and D. B. Lomet. Transaction timestamping in (temporal) databases. In *Proceedings of the 27th VLDB Conference*, 2001.

[10] C. S. Jensen and R. T. Snodgrass. Temporal data management. *IEEE Transactions on Knowledge and Data Engineering*, 11(1), 1999.

[11] A. Marian, S. Abiteboul, G. Cobena, and L. Mignet. Change-centric management of versions in an XML warehouse. In *Proceedings of VLDB 2001*, 2001.

[12] K. Nørvåg. Algorithms for temporal query operators in XML databases. In *Proceedings of Workshop on XML-Based Data Management (XMLDM) (in conjunction with EDBT'2002)*, 2002.

[13] K. Nørvåg. Temporal query operators in XML databases. In *Proceedings of the 17th ACM Symposium on Applied Computing (SAC'2002)*, 2002.

[14] K. Nørvåg. V2: A database approach to temporal document management (submitted for publication), 2002. Available from http://www.idi.ntnu.no/∼noervaag/papers/V2.pdf.

[15] B. Oliboni, E. Quintarelli, and L. Tanca. Temporal aspects of semistructured data. In *Proceeding of TIME-01*, 2001.

[16] C. Z. Shu-Yao Chien, Vassilis J. Tsotras. Efficient management of multiversion documents by object referencing. In *Proceedings of VLDB 2001*, 2001.

[17] R. T. Snodgrass (ed.), I. Ahn, G. Ariav, D. S. Batory, J. Clifford, C. E. Dyreson, R. Elmasri, F. Grandi, C. S. Jensen, W. Kfer, N. Kline, K. G. Kulkarni, T. Y. C. Leung, N. A. Lorentzos, J. F. Roddick, A. Segev, M. D. Soo, and S. M. Sripada. *The TSQL2 temporal query language*. Kluwer Academic, 1995.

[18] A. Steiner. *A Generalisation Approach to Temporal Data Models and their Implementations*. PhD thesis, Swiss Federal Institute of Technology, 1998.

[19] K. Torp, C. S. Jensen, and R. T. Snodgrass. Stratum approaches to temporal DBMS implementation. In *Proceedings of the 1998 International Database Engineering and Applications Symposium*, 1998.