Three Proposals for a Third-Generation Temporal Data Model

Christian S. Jensen
Department of Mathematics and Computer Science
Aalborg University
Fredrik Bajers Vej 7
DK-9220 Aalborg Ø, DENMARK
csj@iesd.auc.dk

Richard Snodgrass
Department of Computer Science
University of Arizona
Tucson, AZ 85721
rts@cs.arizona.edu

Abstract

We present three general proposals for a next-generation temporal data model. Each of these proposals express a synthesis of a variety of contributions from diverse sources within temporal databases. We believe that the proposals may aid in bringing consensus to the area of temporal data models.

The current plethora of diverse and incompatible temporal data models has an impeding effect on the design of a consensus temporal data model. A single *data model is highly desirable, both to the temporal database community and to the database user community at large. It is our contention that the simultaneous foci on the modeling, presentation, representation, and querying of temporal data have been a major cause of the proliferation of models. We advocate instead a separation of concerns.*

As the next step, we propose a data model for the single, central task of temporal data modeling. In this model, tuples are stamped with bitemporal elements, i.e., sets of pairs of valid and transaction time chronons. This model has no intention of being suitable for the other tasks, where existing models may perhaps be more appropriate. However, this model does capture time-varying data in a natural way.

Finally, we argue that flexible support for physical deletion is needed in bitemporal databases. Physical deletion requires special attention in order not to compromise the correctness of query processing.

1 Introduction

The first generation of temporal data models supported only snapshot relations and had no special notions of time built into the query languages. At best, a user-defined time domain was supported. This restricted temporal support is in principle similar to the support for other domains, such as text strings and monetary values. However, adequate support for user-defined time has proven to be considerably more involved than support for other domains. The use of multiple units of time is one complicating factor [DS92]. For example, there exists at least half a dozen definitions of a second. The use of multiple calendars, possible with variable spans (e.g., month), is another source of complication [SS92b, SSD$^+$92, SS92a]. Current commercial database management systems, as well as the SQL89 [ANS89] and SQL2 [Mel90] standards, support first generation temporal data models.

The second generation of temporal data models supports one or more built-in notions of time. Most such data models support relations where facts have an implicit valid-time dimension, indicating when the facts are true in the modeled reality. Other models support transaction-time relations, where it is recorded for each fact when it was current in the relation. Finally, some second generation data models support both valid time and transaction time. Relations in such models are termed *bitemporal relations* [JCG$^+$92].

More than two dozen second generation data models exist [Sno92]. We have come to a point where the novelty of data models often is only in the mix—the ingredients have often and to a large extent been previously used in earlier data models. The continued production of such models may be a disadvantage to the field. Rather than continued fragmentation among researchers, each with their own data model, there is a need for creating consensus.

We envision a third generation of temporal data models that, unlike the second generation models, are consensual and comprehensive. We hope that the three proposals presented here will help in achieving this goal.

The first proposal is that there be a separation of concerns in the design of a temporal data model. We feel that the plethora of temporal data models amply demonstrates that the design of a *single* data model that satisfies the diverse modeling, presentation, and efficient representation and processing requirements is

This paper appeared in Proceedings of the International Workshop on an Infrastructure for Temporal Databases, Arlington, TX, June 14-16, 1993, pp. T-1 - T-10.

not possible. We advocate instead that multiple data models be used, with well-defined mappings between both the data and the operations of each of the models.

This proposal should aid in bringing about consensus. By focusing on one task at a time, the problem is simpler and more clearly defined. Unproductive situations are avoided where, e.g., a proponent of one model criticizes a second model for not being first normal form (and thus not suited for implementation) and the proponent of the second model criticizes the first model for providing a fragmented view of data (i.e., it is a 1NF model and may not be ideal for presentation).

We then propose that tuples of temporal relations be stamped by bitemporal elements, arbitrary sets of points in the space spanned by transaction time and valid time. In combination with the application of appropriate normalization, this may be seen as stamping of atomic, non-temporal facts with their complete temporal aspect, yielding atomic, temporal facts. We hope this simple proposal, which is in some sense a compromise between 1NF and non-1NF approaches, may be accepted as a solution for the isolated task of data modeling. We admit at the onset that this model is not especially appropriate for representation or for presentation, but we have elsewhere proved the correspondence between this simple model and other representational models more suited to these other aspects [JSS92].

Finally, we propose that physical deletion is necessary, and may be supported through cooperative query modification. Despite the dramatic decrease in storage costs, it is still not acceptable to always require that all data be saved forever, and our proposal demonstrates how selective deletion, termed *vacuuming*, may be done in such a way that correctness is not compromised.

In the remainder of this position paper, we visit each of these proposals in turn. Lack of space prevents a detailed discussion of these issues. Instead, we provide pointers to recent papers that do examine these approaches in a comprehensive fashion.

2 Separation of Concerns

As previously mentioned, there are now over two dozen temporal data models, each with one or more associated query languages. While such a diversity of approaches is a reflection of the excitement and ferment in the area of temporal databases, it also at some point may become counter-productive. We have become convinced that a single solution will not be forthcoming and that a different approach is now necessary.

It is our contention that focusing on data *semantics* (what is the meaning of the data stored in the data model), data *presentation* (how temporal data is displayed to the user), on data *storage* (what regular storage structures can be employed with temporal data), and on efficient *query evaluation*, has complicated the primary task of capturing the time-varying semantics. The result has been a plethora of incompatible data models and query languages, and a corresponding surfeit of database design and implementation strategies that may be employed across these models.

We advocate instead adopting a very simple *conceptual* data model that captures the essential semantics of time-varying relations, but has no illusions of being suitable for presentation, storage, or query evaluation [JS92]. We rely instead on existing data models for these tasks, and we have demonstrated equivalence mappings between the conceptual model and several *representational* models [JSS93]. This equivalence is based on *snapshot equivalence*, which says that two relation instances are equivalent if all their snapshots, taken at all times (valid and transaction), are identical. Snapshot equivalence provides a natural means of comparing rather disparate representations. Finally, while not addressed here, we feel that the conceptual data model is the appropriate location for logical database design and query optimization.

The previously proposed representations arose from several considerations. They were all extensions of the conventional relational model that attempted to capture the time-varying semantics of both the enterprise being modeled and the state of the database. They attempted to retain the simplicity of the relational model; the tuple-timestamping models were perhaps most successful in this regard. They attempted to present all the information concerning an object in one tuple; the attribute-value timestamped models were perhaps best at that. And they attempted to ensure ease of implementation and query evaluation efficiency; the backlog representation may be advantageous here.

Most proposed models aim at being suitable for data presentation, for data storage, and for capturing the temporal semantics of data. Seen solely as means of capturing the temporal semantics, such models exhibit presentational and representational anomalies because they encode the temporal semantics in ways that are more complicated than necessary. Put differently, the time-varying semantics is obscured in the representation schemes by other considerations of presentation and implementation.

It is clear from the number of proposed representations that meeting all goals simultaneously is a difficult, if not impossible, task. We therefore advocate a separation of concerns. Figure 1 shows the placement of the conceptual temporal data model with respect to the tasks of logical and physical database design, storage representation, query optimization, and display. As the figure shows, logical database design produces the conceptual relation schemas, which are then refined into relation schemas in some representational data model(s). Query optimization may be performed on the logical algebra, parameterized by the cost models of the representation(s) chosen for the stored data. Finally, display presentation should be decoupled from the storage representation.

Note that this arrangement hinges on the semantic equivalence of the various data models. It must be possible to map between the conceptual model and the various representational models, as discussed in the next section.

We have identified four central temporal data model uses, namely that of data display, data storage, query evaluation, and capturing the time-semantics of data. It is our belief that separating concerns by focusing on one task at a time will prove more productive than trying to meet each of the highly diverse requirements in a single model.

3 Bitemporal Element Stamping of Tuples in the Conceptual Model

We now address the conceptual model at the center of Figure 1. Such a model must be amenable to logical design. Its focus should *not* be on suitable presentation of time-varying data, *nor* on the storage of such data. Rather, it should capture the semantics of the data in as simple and straight-forward a manner as possible. We term our new model the *bitemporal conceptual data model*.

The primary reason behind the success of the relational model is its simplicity. A bitemporal relation is necessarily more complex. Not only must it associate values with facts, as does the relational model, it must also specify *when* the facts were valid in reality, as well as *when* the facts were current in the database. Because our emphasis is on semantic clarity, we will extend the conventional relational model as small an extent as necessary to capture this additional information.

Tuples in a conceptual bitemporal relation instance are associated with time values from two orthogonal time domains, namely valid time and transaction time. Valid time is used for capturing the time-varying nature of the part of reality being modeled, and transaction time models the update activity of the relation. For both domains, we assume that the database system has limited precision, and we term the smallest time unit a *chronon*. For both transaction time and valid time, we assume also that there is a smallest and largest chronon. As we can number the chronons, both domains are isomorphic to a finite subset of the domain of natural numbers.

We propose that *tuples* be timestamped, rather than *attribute values*, in this conceptual model, primarily for simplicity. There are cogent arguments for attribute-value timestamping, primarily addressing how data should be presented to the user or stored on disk. Such concerns are not relevant in this *conceptual* data model.

In the relational model, a tuple encodes some relationship between the values of its attributes. Assigning a timestamp to that tuple delimits this relationship to the specified times in the timestamp. This is the simplest possible arrangement.

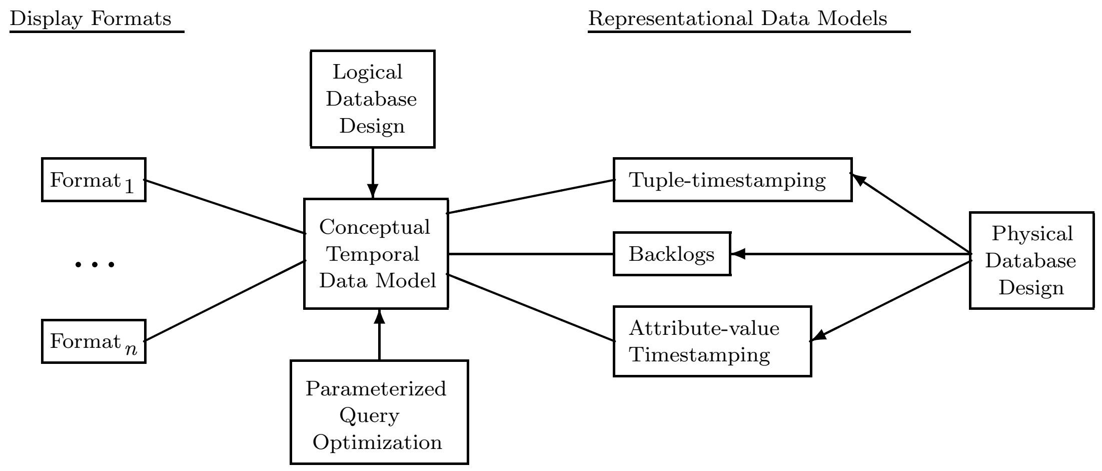
In general, the schema of a conceptual bitemporal relation, $\mathcal{R}$, consists of an arbitrary number of explicit attributes, A_1, A_2, ..., A_n, encoding some fact (possibly composite) and an implicit timestamp attribute, T. Thus, a tuple, $x = (a_1, a_2, \ldots, a_n \mid t)$, in a conceptual bitemporal relation instance, $r(\mathcal{R})$, consists of a number of attribute values associated with a timestamp value.

An arbitrary subset of the domain of valid times is associated with each tuple, meaning that the fact recorded by the tuple is *true in the modeled reality* during each valid time chronon in the subset. Each individual valid time chronon of a single tuple has associated an arbitrary subset of the domain of transaction times, meaning that the fact, valid during the particular chronon, is *current in the relation* during each of the transaction time chronons in the subset.

Associated with a tuple is a set of so-called *bitemporal chronons* (tiny rectangles) in the two-dimensional space spanned by valid time and transaction time. Such a set is termed a *bitemporal element*[1], denoted t_b. Because no two tuples with mutually identical explicit attribute values (termed *value-equivalent*) are allowed in a bitemporal relation instance, the full time history of a fact is contained in a single tuple.

EXAMPLE: Consider a relation recording employee/-department information, such as "Jake works for the Shipping department." We assume that the granularity of chronons is one day for both valid time and

[1]This term is a simple generalization of *temporal element*, used to denotes a set of single dimensional chronons [Gad88]. Alternative, equally desirable terms include *bitemporal time period set* [BZ82] and *bitemporal lifespan* [CC87]. We adopt the terminology of the recently published glossary [JCG+92].



Display Formats
Representational Data Models
Logical Database Design
Format 1
...
Format n
Conceptual Temporal Data Model
Parameterized Query Optimization
Tuple-timestamping
Backlogs
Attribute-value Timestamping
Physical Database Design

Figure 1: Interaction of Conceptual and Representational Data Models

transaction time, and the period of interest is the month of June 1992.

Figure 2 shows how the bitemporal element in an employee's department tuple may change.

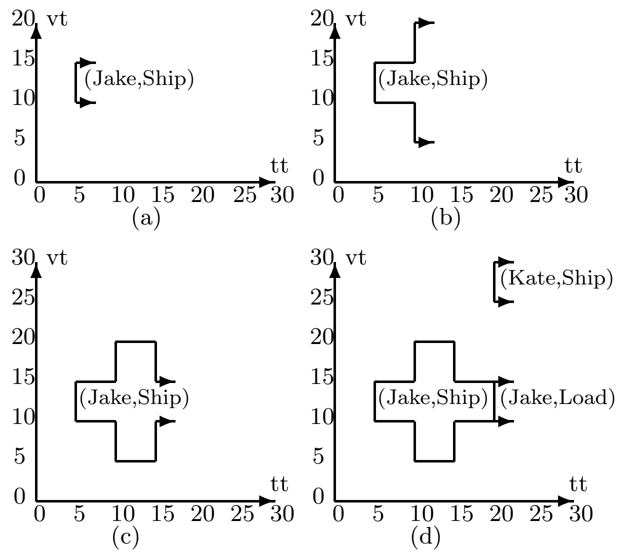


Figure 2: Bitemporal Elements

The x-axis denotes transaction time, and the y-axis denotes valid time. Employee Jake was hired by the company as temporary help in the Shipping department for the interval from June 10th to June 15th, and this fact is recorded in the database predictively on June 5th. This is shown in Figure 2(a). The arrows pointing to the right signify that the tuple has not been logically deleted; it continues through to the transaction time *NOW*. On June 10th, the Personnel department discovers an error. Jake had really been hired for the valid time interval from June 5th to June 20th. The database is corrected on June 10th, and the updated bitemporal element is shown in Figure 2(b). On June 15th, the Personnel department is informed that the correction was itself incorrect; Jake really was hired for the original time interval, June 10th to June 15th, and the database is corrected the same day. This is shown in Figure 2(c). Lastly, Figure 2(d) shows the result of three updates to the relation, all of which take place on June 20th. While the the period of validity was correct, it was discovered that Jake was not in the Shipping department, but in the Loading department. Consequently, the fact (Jake, Ship) is removed from the current state and the fact (Jake, Load) is inserted. A new employee, Kate, is hired for the Shipping department for the interval from June 25th to June 30th.

We note that the bitemporal chronons in a given bitemporal element, associated with a fact, are represented by the area enclosed by the bitemporal element. The bitemporal element for (Jake, Ship) in Figure 2(d) contains 140 bitemporal chronons.

The example illustrates how transaction time and valid time are handled. As time passes, i.e., as the computer's internal clock advances, the bitemporal elements associated with current facts are updated. For example, when (Jake, Ship) was first inserted, the six valid time chronons from 10 to 15 had associated the transaction time chronon *NOW*. At time 5, the six new bitemporal chronons, $(5, 10), \ldots, (5, 15)$, were appended. This continued until time 9, after which the valid time was updated. Thus, starting at time 10, 16 bitemporal chronons are added at every clock tick.

The actual bitemporal relation corresponding to the graphical representation in Figure 2(d) is shown below. This relation contains three facts. The timestamp attribute T shows each transaction time chronon associated with each valid time chronon as a set of ordered pairs.

Emp	Dept	T
Jake	Ship	$\{(5,10),\ldots,(5,15),\ldots,(9,10),\ldots,$ $(9,15),(10,5),\ldots,(10,20),\ldots,$ $(14,5),\ldots,(14,20),(15,10),\ldots,$ $(15,15)\ldots,(19,10),\ldots,(19,15)\}$
Jake	Load	$\{(NOW,10),\ldots,(NOW,15)\}$
Kate	Ship	$\{(NOW,25),\ldots,(NOW,30)\}$

□

We consider the three forms of update, insertion, deletion, and modification, in turn.

An insertion is issued when recording in bitemporal relation instance r that a currently unrecorded fact $(a_1,\ldots,a_n)$ is true for some period(s) of time. These periods of time are represented by a valid-time element, i.e., a set of valid-time chronons, t_v. When the fact is stored, its valid-time element stamp is transformed into a bitemporal-element stamp to capture that, from now on, the fact is current in the relation. We indicate this with a special value, *NOW*, in the domain of transaction-time chronon identifiers.

The arguments to the `insert` routine are the relation into which a fact is to be inserted, the explicit values of the fact, and the set of valid-time chronons, t_v, during which the fact was true in reality. Insert returns the new, updated version of the relation. There are three cases to consider. First, if $(a_1,\ldots,a_n)$ was never recorded in the relation, a completely new tuple is appended. Second, if $(a_1,\ldots,a_n)$ was part of some previously current state, the tuple recording this is updated with the new valid-time information. Third, if $(a_1,\ldots,a_n)$ is already current in the relation, a modification is required, and the insertion is rejected. (In the following, we denote valid-time chronons with c_v and transaction-time chronons with c_t.)

$$\texttt{insert}(r,(a_1,\ldots,a_n),t_v) = \begin{cases} r\cup\{(a_1,\ldots,a_n|\ \{NOW\}\times t_v)\} & \text{if } \neg\exists\, t_b\ ((a_1,\ldots,a_n|\ t_b)\in r) \\ r-\{(a_1,\ldots,a_n,t_b)\} \\ \quad\cup\{(a_1,\ldots,a_n|t_b\cup\{NOW\}\times t_v)\} & \text{if } \exists\, t_b\ ((a_1,\ldots,a_n|\ t_b)\in r\ \wedge \\ & \quad \neg\exists\,(NOW,c_v)\in t_b) \\ r & \text{otherwise} \end{cases}$$

The `insert` routine adds bitemporal chronons with a transaction time of *NOW*.

As time passes, new chronons must be added. We assume that a special routine `ts_update` is applied to all bitemporal relations at each clock tick. We also assume that the transaction-time granularity is sufficiently small that only one transaction can execute within a transaction-time chronon. This function simply updates the timestamps to include the new transaction-time value. The timestamp of each tuple is examined in turn. When a bitemporal chronon of the type (NOW, c_v) is encountered in the timestamp, a new bitemporal chronon (c_t, c_v), where time c_t is the new transaction-time value, is made part of the timestamp.

```
ts_update($r, c_t$) :
    for each $x \in r$
        for each $(NOW, c_v) \in x[T]$
            $x[\mathrm{T}] \leftarrow x[\mathrm{T}] \cup \{(c_t, c_v)\}$
```

Deletion concerns the (logical) removal of a complete tuple from the *current* valid-time state of the bitemporal relation. We distinguish between the case where there is a tuple to delete and the case where no tuple matching the one to be deleted is current.

$$\texttt{delete}(r,(a_1,\ldots,a_n)) = \begin{cases} r-\{(a_1,\ldots,a_n|\ t_b)\} \\ \quad\cup\{(a_1,\ldots,a_n|\ t_b-\texttt{now\_ts}(t_b))\} \\ & \text{if } \exists\, t\ ((a_1,\ldots,a_n|\ t_b)\in r) \\ r & \text{otherwise} \end{cases}$$

where $\texttt{now\_ts}(t_b) = \{(NOW, c_v) \mid (NOW, c_v) \in t_b\}$.

Finally, a modification of an existing tuple may be defined by a deletion followed by an insertion as follows.

$$\texttt{modify}(r,(a_1,\ldots,a_n),t_v) = \texttt{insert}(\texttt{delete}(r,(a_1,\ldots,a_n)),(a_1,\ldots,a_n),t_v)$$

Example: The sequence of bitemporal elements shown in Figure 2 is created by the following sequence of commands, invoked at the indicated transaction time (*TT*).

Command	*TT*
`insert(dept,("Jake","Ship"),[6/10,6/15])`	6/5
`modify(dept,("Jake","Ship"),[6/5,6/20])`	6/10
`modify(dept,("Jake","Ship"),[6/10,6/15])`	6/15
`delete(dept,("Jake","Ship"))`	6/20
`insert(dept,("Jake","Load"),[6/10,6/15])`	6/20
`insert(dept,("Kate","Ship"),[6/25,6/30])`	6/20

□

Valid-time relations and transaction-time relations are special cases of bitemporal relations that support only valid time and transaction time, respectively. Thus an valid-time tuple has associated a set of valid-time chronons (termed a *valid-time element* and denoted t_v), and a transaction-time tuple has associated a set of transaction-time chronons (termed a *transaction-time element* and denoted t_t). For clarity, we use the term snapshot relation for a conventional relation. Snapshot relations support neither valid time nor transaction time.

For comparison, we illustrate next how the bitemporal relation from above may be represented in sample data models that employ employ tuple-timestamped relations, backlogs, and attribute-value timestamped relations.

EXAMPLE: The TQuel data model [Sno87] employs tuple timestamping. A 1NF representation of bitemporal relations is achieved by first partitioning the bitemporal elements of facts into rectangular regions. Then one tuple is generated for each region associated with a fact. The relation corresponding to the conceptual relation in Figure 2(d) is shown below.

Emp	Dept	T_s	T_e	V_s	V_e
Jake	Ship	6/5	6/9	6/10	6/15
Jake	Ship	6/10	6/14	6/5	6/20
Jake	Ship	6/15	6/19	6/10	6/15
Jake	Load	6/20	*NOW*	6/10	6/15
Kate	Ship	6/20	*NOW*	6/25	6/30

□

EXAMPLE: A backlog is simply a timestamped sequence of change requests where each change request is an insertion (I) or a deletion request (D) [JMR91]. Requests for modifications are represented by a deletion request followed by an insertion request. The backlog relation corresponding to the conceptual relation in Figure 2(d) is shown below.

Emp	Dep	V_s	V_e	TT	Op
Jake	Ship	6/10	6/15	6/5	I
Jake	Ship	6/10	6/15	6/10	D
Jake	Ship	6/5	6/20	6/10	I
Jake	Ship	6/5	6/20	6/15	D
Jake	Ship	6/10	6/15	6/15	I
Jake	Ship	6/10	6/15	6/20	D
Jake	Load	6/10	6/15	6/20	I
Kate	Ship	6/25	6/30	6/20	I

□

EXAMPLE: The data model of the TempSQL query language uses attribute-value timestamping [Gad92]. In this model, information may be grouped within a single tuple, based on the value of any attribute or set of attributes. For example, we could represent the conceptual relation in Figure 2(d) by grouping on the employee attribute. Then all information for an employee is contained within a single tuple, as shown below.

Emp		Dept	
[20,*NOW*] × [25,30]	Kate	[20,*NOW*] × [25,30]	Ship
[5,9] × [10,15]	Jake	[5,9] × [10,15]	Ship
[10,14] × [5,20]	Jake	[10,14] × [5,20]	Ship
[15,19] × [10,15]	Jake	[15,19] × [10,15]	Ship
[20,*NOW*] × [10,15]	Jake	[20,*NOW*] × [10,15]	Load

A tuple in the above relation shows all departments for which a single employee has worked. A different way to view the same information is to perform the grouping by department. A single tuple then contains all information for a department, i.e., the full record of employees who have worked for the department.

Emp		Dept	
[20,*NOW*] × [10,15]	Jake	[20,*NOW*] × [10,15]	Load
[5,9] × [10,15]	Jake	[5,9] × [10,15]	Ship
[10,14] × [5,20]	Jake	[10,14] × [5,20]	Ship
[15,19] × [10,15]	Jake	[15,19] × [10,15]	Ship
[20,*NOW*] × [25,30]	Kate	[20,*NOW*] × [25,30]	Ship

Grouping by both attributes would yield three tuples, (Jake, Load), (Jake, Ship), and (Kate, Ship). □

The timestamping of tuples with bitemporal elements is a conceptually very simple extension of the standard relational model, with several nice properties. First, traditional normalization techniques that work with first generation data models may be generalized to apply to this model [JSS92]. Second, the complete bitemporal history of a fact is recorded in a single tuple. Third, time variables ranging over the domain of bitemporal elements lead to a simple set algebra-based query language (a point Shashi Gadia made originally [Gad88], and which has been echoed by several others in the field).

4 Vacuuming—Safe, Flexible, and Cooperative Support for Physical Deletion

Base relations in temporal database systems supporting transaction time are ever-growing because all logical updates, including deletion, transform into insertions at the physical level. As physical deletion capabilities exist in conventional database systems, such capabilities should also exist in temporal database systems. However, when transaction time is supported, straight-forward deletion has the unacceptable consequence that history is forged.

Example: Consider an example where a publisher decides to remove certain phrases from a new printing of a thirty year old history book while maintaining that this printing has the original contents. It is clear that history has then been changed in a highly problematic way. The book now gives incorrect information about the authors view of history thirty years ago.

This scenario is in principle similar to straight-forward physical deletion in a bitemporal database. By deleting from previously current states of the database, we create incorrect information.

Note that if the publisher, on the other hand, clearly states that the new printing constitutes a new, abridged version of the original printing then the correctness problems are avoided. □

In order to avoid the correctness problems, physical deletion in temporal databases needs special attention. Below, we first outline the reasons why physical deletion is necessary and then illustrate in more detail the problem with straight-forward deletion. Next, we outline a possible solution, termed *vacuuming*, to the problem and touch upon selected aspects of this proposed solution [JM90].

Flexible deletion is necessary for several reasons. First, in many installations ever-growing relations will eventually outgrow the available mass storage devices (e.g., magnetic disks). In order to ensure continued operation, it will become necessary to physically delete data. It must be possible to delete data that are no longer needed, or when additional space needs to be freed for more important data.

Second, the efficiency of query processing generally degrades as relations grow. For this reason, flexible means of controlling the relation sizes are highly desirable.

Third, many countries have strict laws that require the ability to delete certain records of previous history. Customers may demand that no information about them exists in some database. Other laws may require that certain records be kept for a fixed duration of time. For example, information related to personal income tax must in some countries be retained by the citizens for five years. Business policies also pose requirements and are in this respect similar to laws.

To see in more detail why straight-forward deletion is problematic, consider the following example. We first show a sample relation and its physical representation. Then we consider querying the relation while allowing for physical deletion.

Example: Assume that a transaction-time relation schema, *EmpDep*, is defined as follows.

$$EmpDep = (\text{Emp},\ \text{Dep} \mid \text{T})$$

As in the examples of the previous section, Emp is the employee name and Dep is the department where the employee works. Attribute T is the implicit transaction-time attribute that takes transaction-time elements as values. Let *empDep,* as illustrated next, be an instance of this schema as of 6/25. The transaction timestamps refer to the month of June 1992.

Emp	Dept	T
Jake	Ship	$\{6/5, 6/6, 6/7, \ldots, 6/19\}$
Jake	Load	$\{6/20, 6/21, 6/22, \ldots, 6/25, NOW\}$
Kate	Load	$\{6/10, 6/11, 6/12, \ldots, 6/25, NOW\}$
Kate	Ship	$\{6/23, 6/24, 6/25, NOW\}$

To discuss physical deletion, it is necessary to consider a physical representation of the conceptual relation. It may be implemented as a backlog relation. A backlog is simply a timestamped sequence of change requests, and the following snapshot schema may be used.

$$B_{EmpDep} = (\text{Emp},\ \text{Dep},\ \text{Op},\ \text{TT})$$

Attribute Op indicates whether the request is an insertion or a deletion request (modifications are combinations of deletions and insertions), and attribute TT is a transaction timestamp indicating the time of the request. The backlog, b_{empDep}, corresponding to the relation *empDep* is shown below.

Emp	Dept	T	Op
Jake	Ship	6/5	I
Kate	Load	6/10	I
Jake	Ship	6/20	D
Jake	Load	6/20	I
Kate	Ship	6/23	I

To compute a timeslice such as $\tau_{6/17}(empDep)$, the change requests are simply processed in time order, starting with the first change request and ending with the last timestamp not exceeding 6/17 (this may be optimized by means of caching). The result is a snapshot relation consisting of the two tuples (Jake, Ship), and (Kate, Load).

Relation b_{emp} is ever-growing and is likely to eventually contain data irrelevant to its users.

On June 25, before any deletions are carried out, assume that a user issues the query $\pi_{\mathrm{Emp}}(\tau_{6/17}(empDep))$ which retrieves the names of all employees in the *empDep* relation as of June 17. The result is $\{\mathrm{Jake}, \mathrm{Kate}\}$.

Next, on June 27, 1993, it is decided that data related to the shipping department that has not been current for the last five days is not to be retained any longer. In response to this decision, old data is deleted in the straight-forward manner. The result is shown below.

Emp	Dept	T	Op
Kate	Load	6/10	I
Jake	Load	6/20	I
Kate	Ship	6/23	I

On June 29 the user from before issues the same query as on June 25. The new result is different from the previous result. Employees that left the shipping department after June 17 will appear in the previous result, but employees that left before June 22 will not appear in the new result. Thus Jake, who left between June 17 and June 22, will not appear in the new result.

Correctness has been compromised because a situation has occurred where the *same* query on the *same* old state has given two *different* results, both of which cannot be correct. □

We now present a possible solution, termed vacuuming, to the problem of providing safe physical deletion. In addition to being safe, vacuuming is flexible and cooperative.

The data to be physically deleted from base relations is expressed by *deletion specifications.* With vacuuming, a wide class of query expression are legal deletion specifications. Also, vacuuming may be specified at any time during the life of a base relation [JM90], not only at the time the relation is created [RS87]. This provides flexibility.

There are two steps in allowing physical deletion without compromising the correctness of query processing.

1. Detect all queries that may compromise correctness.

2. Do not evaluate the unsafe query, but instead offer, in a cooperative fashion, to evaluate a similar query that is guaranteed to produce a correct result.

The first step of vacuuming may be accomplished as follows. A deletion specification is a query expression that retrieves data from a base relation. For each base relation, all its deletion specifications are combined into a single query expression. The relation name minus the combined query expression specifies which data remain after deletion. Then, when a query expression is issued for evaluation, all references to base relations are replaced with the expression just constructed. This is similar to the implementation of views by query modification [Sto75]. This modified query is then tested for equivalence with the original query [ASU79]. If the expressions cannot be proved equivalent then the original query may compromise correctness.

The second step is accomplished by interactively considering the modified query expression (which by construction is guaranteed to not violate correctness). The system may offer to evaluate this expression. In addition, it is possible to offer to evaluate variations of the modified expression. specifically, the modified expression may be simplified by means of query specialization and generalization [Cha90, Mot84].

Several interesting issues have not been addressed above. For example, not all deletion specifications are appropriate. A specification telling to delete data that is between one and two years old is an example. Data more than two years old cannot be deleted, and data not yet two years old will eventually become two years old and can therefore not be deleted. In consequence, nothing can be deleted and the specification is at best useless. As other examples, deletion specifications should not delete data needed by other specifications, integrity constraints, and views.

The actual physical deletion is performed by an asynchronous vacuuming demon according to the specifications. While vacuuming logically has eager semantics, any degree of eagerness or laziness can be adopted for the actual physical removal of base data, and a variety of conditions triggering the demon can be employed.

5 Summary

In this paper, we have argue that a third generation of temporal data models is desirable. In contrast to the first two generations, this new generation of models should be consensual and comprehensive.

Because of the current plethora of diverse and mutually exclusive temporal data models, we propose that an approach based on separation of concerns is more productive and should be adopted for the development of the next generation of temporal data models. We observed that data presentation, data storage and data modeling pose very diverse and seemingly incompatible requirements. Based on this, we recommend that different data models be adopted for each of the three purposes.

In continuation of the separation-of-concerns approach, we propose a very simple data model for the task of temporal data modeling. This data model stamps tuples with bitemporal elements. Bitemporal elements are sets of pairs of transaction and valid time chronons. Being in some sense a compromise between first and non-first normal form models, the explicit schema of a relation is in first normal form, but the implicit timestamp attribute is set valued. Relations in this model store exactly one fact (possibly composite) per tuple, and conventional normalization techniques are easily extended to these [JSS92].

Some temporal data models support grouping the tuples of a relation that agree on some arbitrarily chosen subset of attributes of the relation schema. This makes it possible to specify and manipulate, e.g., the employment histories of individual employees in a company, or the employee records of individual departments. We believe that this grouping, like the grouping in conventional data models, should be confined to the query language. For that reason grouping is not addressed here.

Finally, we have argued that new temporal models supporting transaction time should allow for the physical deletion of data. We have discussed the problems involved in providing such support, and we have outlined a solution to the problems.

Acknowledgements

We thank Mike Soo, Curtis Dyreson, Nick Kline, and Suchen Hsu for their participation in inspiring discussions of topics related to the contents of this paper.

The work was supported by the Danish Natural Science Research Council through grant 11–9675–1 SE, the National Science Foundation through grant IRI–8902707, and the IBM Corporation through Contract #1124.

References

[ANS89] American National Standards Institute. *X3.135-1989 Database Language SQL*, 1989.

[ASU79] A. V. Aho, Y. Sagiv, and J. D. Ullman. Equivalences among Relational Expressions. *SIAM Journal of Computing*, 8(2):218–246, May 1979.

[BZ82] J. Ben-Zvi. *The Time Relational Model*. PhD thesis, Computer Science Department, UCLA, 1982.

[CC87] J. Clifford and A. Croker. The Historical Relational Data Model (HRDM) and Algebra Based on Lifespans. In *Proceedings of the Third International Conference on Data Engineering*, pages 528–537, Los Angeles, CA, February 1987.

[Cha90] S. Chaudhuri. Generalization as a Framework for Query Modification. In *Proceedings of the Sixth International Conference on Data Engineering*, pages 138–145, February 1990.

[DS92] C. E. Dyreson and R. T. Snodgrass. Time-stamp Semantics and Representation. Technical Report TR 92-16a, Department of Computer Science, University of Arizona, July 1992.

[Gad88] S. K. Gadia. A Homogeneous Relational Model and Query Languages for Temporal Databases. *ACM Transactions on Database Systems*, 13(4):418–448, December 1988.

[Gad92] S. K. Gadia. A Seamless Generic Extension of SQL for Querying Temporal Data. Technical Report TR-92-02, Computer Science Department, Iowa State University, May 1992.

[JCG$^+$92] C. S. Jensen, J. Clifford, S. K. Gadia, A. Segev, and R. T. Snodgrass. A Glossary of Temporal Database Concepts. *SIGMOD Record*, 21(3), September 1992.

[JM90] C. S. Jensen and L. Mark. A Framework for Vacuuming Temporal Databases. Technical Report CS-TR-2516/UMIACS-TR-90-105, Department of Computer Science, University of Maryland, College Park, MD, August 1990.

[JMR91] C. S. Jensen, L. Mark, and N. Roussopoulos. Incremental Implementation Model for Relational Databases with Transaction Time. *IEEE Transactions on Data Engineering*, 3(4):461–473, December 1991.

[JS92] C. S Jensen and R. T. Snodgrass. Proposal of a Data Model for the Temporal Structured Query Language. TempIS Technical

Report 37, Department of Computer Science, University of Arizona, Tucson, AZ, July 1992.

[JSS92] C. S. Jensen, R. T. Snodgrass, and M. D. Soo. Extending Normal Forms to Temporal Relations. TR 92-17, Department of Computer Science, University of Arizona, July 1992.

[JSS93] C. S. Jensen, M. D. Soo, and R. T. Snodgrass. Unification of Temporal Relations. In *Proceedings of the Nineth International Conference on Data Engineering*, Vienna, Austria, April 1993.

[Mel90] J. Melton (ed.). *Solicitation of Comments: Database Language SQL2*. American National Standards Institute, Washington, DC, July 1990.

[Mot84] A. Motro. Query Generalization: A Technique for Handling Query Failure. In *Proceedings of the First International Workshop on Expert Database Systems*, pages 314–325, October 1984.

[RS87] L. Rowe and M. Stonebraker. The Postgres Papers. Technical Report UCB/ERL M86/85, University of California, Berkeley, CA, June 1987.

[Sno87] R. T. Snodgrass. The Temporal Query Language TQuel. *ACM Transactions on Database Systems*, 12(2):247–298, June 1987.

[Sno92] R. T. Snodgrass. *Temporal Databases*, in *Theory and Methods of Spatio-Temporal Reasoning in Geographic Space*, Springer-Verlag, Pisa, Italy, September, 1992, Vol. 639, A.U. Frank, I. Campari, and U. Formentini, eds, pp. 22-64.

[SS92a] M. D. Soo and R. T. Snodgrass. Mixed Calendar Query Language Support for Temporal Constants. TempIS Technical Report 29, Department of Computer Science, University of Arizona, Tucson, Arizona 85721, Revised May 1992.

[SS92b] M. D. Soo and R. T. Snodgrass. Multiple Calendar Support for Conventional Database Management Systems. Technical Report 92-7, Department of Computer Science, University of Arizona, February 1992.

[SSD$^+$92] M. Soo, R. Snodgrass, C. Dyreson, C. S. Jensen, and N. Kline. Architectural Extensions to Support Multiple Calendars. TempIS Technical Report 32, Department of Computer Science, University of Arizona, Revised May 1992.

[Sto75] M. Stonebraker. Implementation of Integrity Constraints and Views by Query Modification. Memorandum, ERL-M514, Electronics Research Laboratory, College of Engineering, University of California, Berkeley 94720, March 1975.