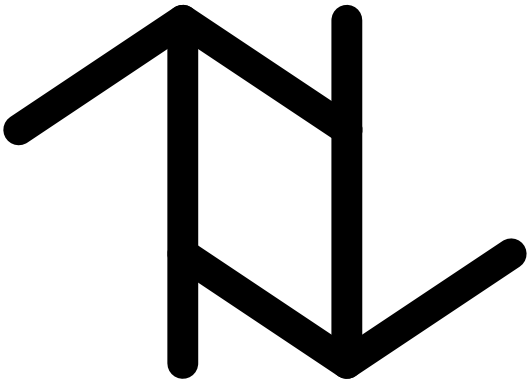

Modification Semantics in Now-Relative Databases

Kristian Torp Christian S. Jensen Richard T. Snodgrass

March 16, 2004

TR-43

A TimeCenter Technical Report

Title	**Modification Semantics in Now-Relative Databases**
Author(s)	Kristian Torp Christian S. Jensen Richard T. Snodgrass
Publication History	First version September 15, 1999

TIMECENTER Participants

Aalborg University, Denmark
Christian S. Jensen (codirector), Michael H. Böhlen, Renato Busatto, Curtis E. Dyreson, Heidi Gregersen, Dieter Pfoser, Simonas Šaltenis, Janne Skyt, Giedrius Slivinskas, Kristian Torp

University of Arizona, USA
Richard T. Snodgrass (codirector), Bongki Moon, Sudha Ram

Individual participants
Anindya Datta, Georgia Institute of Technology, USA
Kwang W. Nam, Chungbuk National University, Korea
Mario A. Nascimento, State University of Campinas and EMBRAPA, Brazil
Keun H. Ryu, Chungbuk National University, Korea
Michael D. Soo, University of South Florida, USA
Andreas Steiner, TimeConsult, Switzerland
Vassilis Tsotras, University of California, Riverside, USA
Jef Wijsen, Vrije Universiteit Brussel, Belgium

For additional information, see The TIMECENTER Homepage:
URL: `<http://www.cs.auc.dk/TimeCenter>`

The TIMECENTER icon on the cover combines two "arrows." These "arrows" are letters in the so-called *Rune* alphabet used one millennium ago by the Vikings, as well as by their precedessors and successors. The Rune alphabet (second phase) has 16 letters, all of which have angular shapes and lack horizontal lines because the primary storage medium was wood. Runes may also be found on jewelry, tools, and weapons and were perceived by many as having magic, hidden powers.

The two Rune arrows in the icon denote "T" and "C," respectively.

Abstract

Most real-world databases record time-varying information. In such databases, the notion of "the current time," or *NOW*, occurs naturally and prominently. For example, when capturing the past states of a relation using begin and end time attributes, tuples that are part of the current state have some past time as their begin time and *NOW* as their end time. While the semantics of such *variable* databases has been described in detail and is well understood, the modification of variable databases remains unexplored.

This paper defines the semantics of modifications involving the variable *NOW*. More specifically, the problems with modifications in the presence of *NOW* are explored, illustrating that the main problems are with modifications and tuples that reach into the future. The paper defines the semantics of modifications—including insertions, deletions, and updates—of databases without *NOW*, with *NOW*, and with values of the type $NOW + \Delta$, where Δ is a non-variable time duration. To accommodate these semantics, three new timestamp values are introduced. An approximate semantics that does not rely on new timestamp values is also provided. Finally, implementation is explored.

1 Introduction

Most real-world database applications record time-varying information. It is typical to represent the time to which the fact(s) recorded by a tuple in a relational database apply by a pair of time-valued attributes, which then encode a time interval. Many of the tuples in a database typically record facts that apply to a time interval that stretches from some past time to the current time, prompting a need for a time value that denotes the "current time" in the to-time attribute of these tuples.

While SQL-92 [22] includes the datetime value functions `CURRENT_DATE`, `CURRENT_TIME`, and `CURRENT_TIMESTAMP`, these functions cannot be stored directly as values of attributes in relations. In the absence of a "current time" value in SQL's `DATE`, `TIME`, and `TIMESTAMP` domains or in the corresponding domains offered by database vendors, common ad-hoc solutions are to use either the null value or the maximum value of the time domain for the value of the to-time attribute.

Noting the deficiencies of these solutions, the variable *NOW* that evaluates to the current time has been introduced as a value of an attribute that may be stored in the database. The semantics of databases including this variable have been examined in some detail [10, 14, 16, 24]. While these papers have considered *NOW* in queries, they provide few details on the modification of variable databases.

In the present paper, we define the semantics of modifications of variable databases containing *NOW* and $NOW + \Delta$, and provide means of supporting these semantics. In addition, we show how modifications under this semantics may be implemented within a DBMS and in a user-application. An approximate semantics that is simpler to implement, but carries with it lower fidelity, is also provided.

The presentation is organized as follows. We first give a simple example to indicate the subtleties and pitfalls inherent in modifications on databases containing *NOW* as well as of the practical importance of such modifications. In Section 3, the semantics of modifications of databases without *NOW* is defined. Section 4 defines the semantics of modifications of databases with *NOW* as a consistent extension. Section 5 extends the approach to also accommodate values of the form $NOW + \Delta$, thereby affording a general solution, with Section 6 providing details on how to implement the semantics defined in the two previous sections. We then provide a simplified, approximate semantics of modifications of variable databases that is easier to implement, in Section 7. Related work is covered in Section 8, and Section 9 concludes the paper.

2 Problem Description

We motivate the problem addressed in this paper with an example that illustrates the utility of *NOW* in capturing time-varying information in the database, but also demonstrates that the semantics of modifications of tuples timestamped with *NOW* is unclear.

When modifying tuples timestamped with intervals not including *NOW*, the interval affected by the modification is the intersection of the interval associated with the tuple and the interval specified in the modification [1]. To exemplify, in Figure 1 we have stored the tuple ⟨Joe, Shoe, [10,15)⟩, and we want to update all persons in the Shoe department to be in the Toy department in the interval [10,20). The result is that Joe will be with the Toy department in the interval [10,15). (For simplicity, we assume that all dates are in January during some year, and we utilize closed-open intervals.)

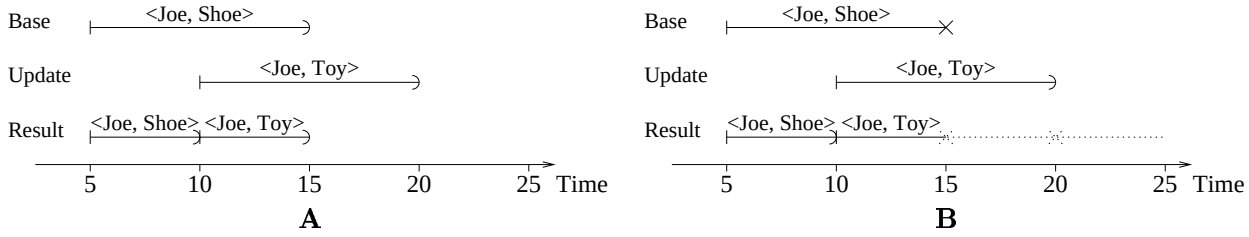

Figure 1: (A) Updating a Fact Without *NOW*; (B) Updating a Fact With *NOW*

When allowing intervals to include the variable *NOW*, it is still desirable that this intersection semantics be maintained. However, there are problems redefining the intersection operator, as illustrated in Figure 1B, where Joe is with the Shoe department in the interval [5,*NOW*). (We denote *NOW* with '×'.) We have also indicated an update statement that, at the 15th of January, updates Joe to be with the Toy department in the interval [10,20).

We want to determine the outcome of the update. Before the 5th of January, Joe is not in the database. In the interval [5,10), Joe was with the Shoe department, and this interval is not affected by the update, so Joe remains there. In the interval [10,15), Joe was also with the Shoe department, and the department value should be updated for this interval. The semantics of the update becomes unclear for the interval [15,20), and it is also unclear what happens after the 20th of January. This is indicated by the dashed line in Figure 1B.

If we use a pessimistic semantics, Joe could be fired tomorrow, and so we cannot update Joe for the latter interval. Further, with the pessimistic approach Joe is not associated with the Shoe department after the 20th of January either. We can also apply an optimistic semantics and assume that Joe is not going to be fired in the near future. We then update Joe to be with the Toy department for the interval [15,20), and associate Joe with the Shoe department again after the 20th of January. A third, intermediate approach would be to bind the value of *NOW* to the current time and then execute the update, with the result that Joe's department is changed over the interval [10,15). These three possible outcomes are shown in Table 1.

Each result reflects its underlying assumptions. With the pessimistic semantics in Table 1A, we assume that Joe is fired tomorrow. With the optimistic semantics in Table 1B, we assume Joe is with the company after the 20th of January. Finally, in Table 1C, we assume that *NOW* is the current time, i.e., the 15th of January.

3 Modifications of Ground Databases

As an outset, we define the semantics of modifications of databases without the variable *NOW*, termed *ground databases* because they are variable-free. This semantics is used to identify the

Name	Dept.	V-Begin	V-End
Joe	Shoe	5	10
Joe	Toy	10	15

A

Name	Dept.	V-Begin	V-End
Joe	Shoe	5	10
Joe	Toy	10	20
Joe	Shoe	20	*NOW*

B

Name	Dept.	V-Begin	V-End
Joe	Shoe	5	10
Joe	Toy	10	15
Joe	Shoe	15	*NOW*

C

Table 1: (A) Optimistic, (B) Pessimistic, and (C) Intermediate Semantics of the Update in Figure 1B

extensions needed to define modifications of databases with the variable *NOW*, termed *variable databases* [10]. Later we compare the semantics of modifications on ground and variable databases. Most existing temporal data models support intervals without *NOW* in both queries and modifications, see, e.g., [6, 26].

We focus on the valid-time aspect of the tuples, i.e., when the information recorded by the tuples is true in the miniworld [19]. The transaction-time aspect, when tuples are current in the database, is a simpler special case because transaction times are maintained by the database management system itself and do not extend into the future. The subtleties examined here thus concern only valid time.

3.1 Preliminaries

We first define the union of valid-time relations and the interval difference and intersection operators, which are used in the definitions of modifications.

We utilize the conventional relational model, but partition the attributes into so-called explicit attributes and two datetime attributes, V-Begin and V-End, denoting an interval in valid time. Let r_{vt} and s_{vt} be two union-compatible valid-time relations with schema $\langle A_1, \ldots, A_n, \text{V-Begin}, \text{V-End}\rangle$, where the A_i are the explicit attributes and VT = [V-Begin, V-End) record the valid time. The union operator ($\cup^{vt}$) for valid-time relations is defined as follows.

$$r_{vt} \ \cup^{vt} \ s_{vt} \ \stackrel{\triangle}{=} \ \{t | t \in r_{vt} \vee t \in s_{vt}\}$$

The valid-time union operator is identical to the conventional relational algebra union operator for ground relations, except that the arguments can be valid-time relations, with their valid-time attribute just carried along.

We assume a time domain $\mathcal{T}$ that is isomorphic to a finite subset of the natural numbers, with the normal total order, $<$. We denote the minimum and maximum values of the time domain *beginning* and *forever*, respectively. The meaning of a closed-open interval is defined as follows, where a and b are in $\mathcal{T}$.

$$[a, b) \ \stackrel{\triangle}{=} \ \begin{cases} \{t | t \leq a \wedge t < b\} & \text{if } a < b \\ \emptyset & \text{otherwise} \end{cases}$$

If the interval start value is smaller than the interval end value, the interval consists of the values between a and b, including a. Otherwise, the interval denotes the empty set.

Let a, b, c, and d be in $\mathcal{T}$. The difference of intervals ($-$) is defined as follows.

$$[a,b) \;-\; [c,d) \;\stackrel{\triangle}{=} \begin{cases} \{[a,c),[d,b)\} & \text{if } a<d \wedge c<b \\ \{[a,b)\} & \text{otherwise} \end{cases}$$

The first line applies when the argument intervals overlap. Zero, one, or two non-empty intervals may be returned. The second line returns the interval [a,b) unchanged if this interval is before or after interval [c,d). The three drawings in Figure 2 illustrate the interval difference operator.

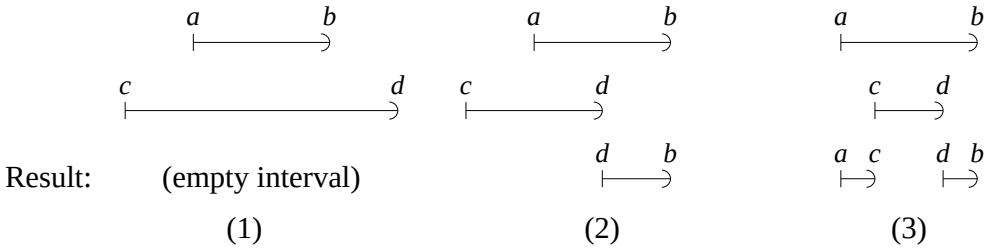

Figure 2: Intervals Returned by the Difference Operator

The intersection operator of intervals ($\cap$) is defined as follows, where *min* and *max* are the conventional minimum and maximum functions returning the smallest and largest argument, respectively.

$$[a,b) \;\cap\; [c,d) \;\stackrel{\triangle}{=} [max(a,c), min(b,d))$$

Two comments are in order. First, intersection is not strictly needed, because $[a,b) \cap [c,d)$ is equal to $[a,b) - ([a,b) - [c,d))$. However, interval intersection is convenient in the later definitions. Second, the union of intervals can also be defined in terms of the *min* and *max* functions on the end points, but the union of intervals is not needed in this paper.

3.2 Semantics of Temporal Modifications on Ground Databases

We define insertion, deletion, and update in turn. Insertion into a valid-time relation r_{vt} is defined as follows, where A is used an abbreviation for $A_1, \ldots, A_n$ and [*vts*,*vte*) is the valid-time interval to be associated with the inserted tuples.

`VALIDTIME PERIOD [`vts`,`vte`) INSERT INTO` r_{vt} `VALUES (`A`)` $\stackrel{\triangle}{=}$

$$r_{vt} \leftarrow r_{vt} \cup^{vt} \{(A,\ [vts, vte))\}$$

A tuple is added to the relation. We associate with the tuple the valid-time interval [*vts*,*vte*) specified in the insert statement. If such an interval is not specified, an interval of now to forever is used, to effect temporal upward compatibility [2].

Deletion from a valid-time relation r_{vt} is defined as next. Again, A abbreviates $A_1, \ldots, A_n$.

`VALIDTIME PERIOD [`vts`,`vte`) DELETE FROM` r_{vt} `WHERE` $cond$ $\stackrel{\triangle}{=}$

$$\begin{aligned} r_{vt} \;\leftarrow\; & \{t | t \in r_{vt}(\neg cond(t))\} \cup^{vt} \\ & \{t | \exists s \in r_{vt}(cond(s) \wedge t[A] = s[A] \wedge t[\mathrm{VT}] \in (s[\mathrm{VT}] - [vts, vte)) \wedge t[\mathrm{VT}] \neq \emptyset)\} \end{aligned}$$

The first line ensures that tuples in r_{vt} not satisfying condition *cond* are included in the result. In the second line, tuples satisfying the condition have their time interval reduced by the part that

overlaps the interval specified in the delete statement. This interval must be non-empty. Note, if a tuple satisfies the condition, but does not overlap the interval specified in the deletion, the tuple is included in the result unchanged.

In the definition of updates that follows, we assume for brevity that all explicit attributes change values. This simplification does not restrict the generality of the results of this paper. Updating a valid-time relation r_{vt} is defined as follows, where $A = v$ abbreviates $A_1 = v_1, \ldots, A_n = v_n$.

$$\texttt{VALIDTIME PERIOD [}vts\texttt{,}vte\texttt{) UPDATE } r_{vt} \texttt{ SET } A = v \texttt{ WHERE } cond \triangleq$$

$$\begin{aligned} r_{vt} \quad \leftarrow \quad & \{t | t \in r_{vt}(\neg cond(t))\} \cup^{vt} \\ & \{t | \exists s \in r_{vt}(cond(s) \wedge t[A] = s[A] \wedge t[\mathrm{VT}] \in (s[\mathrm{VT}] - [vts, vte)) \wedge t[\mathrm{VT}] \neq \emptyset)\} \cup^{vt} \\ & \{t | \exists s \in r_{vt}(cond(s) \wedge t[A] = v \wedge t[\mathrm{VT}] = s[\mathrm{VT}] \cap [vts, vte) \wedge t[\mathrm{VT}] \neq \emptyset)\} \end{aligned}$$

The first and second lines are identical to the two lines of the delete statement. The third line adds tuples with the updated attribute values to the result. The valid-time intervals associated with these updated tuples are the (non-empty) intersections of the valid-time interval currently associated with each corresponding argument tuple and the interval specified in the update statement.

3.3 Examples of Modifications on Ground Databases

This section exemplifies the temporal modification statements on ground databases defined above. First, an example of a delete is given. Assume the database contains the tuple $\langle$Joe, Shoe, $[5, 20)\rangle$ and that we want to delete Joe in the interval $[10, 15)$. This can be written as follows.

```
VALIDTIME PERIOD [10,15) DELETE FROM Emp WHERE Name = 'Joe'
```

The result of the delete is as follows.

$$\begin{aligned} \emptyset \cup^{vt} & \{\langle \text{Joe}, \text{Shoe}, \{[5, 20) - [10, 15)\}\rangle\} \\ & = \{\langle \text{Joe}, \text{Shoe}, [5, 10)\rangle, \quad \langle \text{Joe}, \text{Shoe}, [15, 20)\rangle\} \end{aligned}$$

From the single tuple stored in the relation, we remove Joe in the interval $[10, 15)$, which results in two tuples.

Next, assume again that the database contains the tuple $\langle$Joe, Shoe, $[5, 20)\rangle$ and that we want to update Joe to be with the Toy department in the interval $[10, 30)$. This can be written as follows.

```
VALIDTIME PERIOD [10,15) UPDATE Emp SET Dept = 'Toy' WHERE Name = 'Joe'
```

The result of the update is as follows.

$$\begin{aligned} \emptyset \cup^{vt} & \{\langle \text{Joe}, \text{Shoe}, \{[5, 20) - [10, 30)\}\rangle\} \cup^{vt} \{\langle \text{Joe}, \text{Toy}, \{[5, 20) \cap [10, 30)\}\rangle\} \\ & = \{\langle \text{Joe}, \text{Shoe}, [5, 10)\rangle, \quad \langle \text{Joe}, \text{Toy}, [10, 20)\rangle\} \end{aligned}$$

From the single tuple stored in the relation, we remove Joe in the interval $[10, 30)$. This results in the tuple $\langle$Joe, Shoe, $[5, 10)\rangle$. Further, we update Joe to be with the Toy department in the intersection of the intervals $[5, 20)$ and $[10, 30)$, so that Joe is with the Shoe department in the interval $[10, 20)$.

4 Semantics of Modifications Involving *NOW*

Ground databases only evolve through the explicit application of user-supplied modification statements. The presence of variable *NOW* in its tuples permits a database to evolve purely through the passage of time.

In this section we formally define the semantics of modifications of the variable databases that result from introducing *NOW*. We first list our requirements to the semantics of modifications in the presence of *NOW*. This is followed by an example that illustrates the desired semantics. Two necessary extensions are identified and defined, namely (a) the extension of the domain of time values and (b) the extension of the conventional interval difference and intersection operators to handle the extended time domain. Finally, the semantics of modifications involving *NOW* are defined and exemplified.

4.1 The Use of *NOW*

The use of *NOW* as an interval end-point helps us to better record information that remains true from some past time until the increasing current time. With *NOW* available, we avoid solutions such as using the maximum value in the time domain as a substitute interval-end time, which, using our example database, results in the database indicating that Joe is with the Shoe department for more than 7000 years (assuming the standard `DATE` type, with a maximum value of 9999-12-31).

In order to accommodate the variable *NOW* in the database, special support is needed in both queries and modifications. The meaning of databases with *NOW* and the querying of such databases has been covered extensively elsewhere [10]. However, the impact on modifications of the presence of *NOW* in the database as well as in the modification statements themselves has not been covered, even though many temporal data models, e.g., [3, 6, 7, 8, 17, 20, 26, 32], assume that *NOW* can be stored in the database.

Before defining the semantics of modifications on variable databases, we specify three requirements to the accommodation of *NOW*.

Requirement R1 The conventional insert, delete, and update statements should be extended to permit constant intervals, i.e., $[a, b)$, as well as now-relative intervals, i.e., $[a, NOW)$ and $[NOW, b)$, as user input.

For example, the last statement in Section 3.3 used the ground interval $[10, 15)$; it should be possible to use a now-relative interval in its place.

Requirement R2 The semantics of modifications on variable databases should reduce to the semantics of modifications on conventional, ground databases. The meaning of a variable-database modification should be the same as the meaning of a ground-database modification in the case that the variable database in fact contains no occurrences of *NOW*.

Requirement R3 The database that results from the modifications to be defined on the variable database should be representable in the common first-normal-form format that employs two timestamp columns.

The following two extensions are needed to define the semantics of variable-database modifications that meet these requirements.

- The time domain from which the interval end-point values are drawn must be extended to include *NOW* and other values, as we shall see.
- The conventional interval difference and intersect operators that are used in the definition of the modification semantics must be extended to accommodate the new kinds of end values.

Before we define these extensions, we illustrate and motivate the desired semantics of modifications involving *NOW*.

4.2 Motivating Example

To convey the intuition for what the semantics of modifications involving *NOW* should be, we show the desired results of sample updates with and without *NOW*. We use the two updates in Figure 1, illustrating them using *extensionalization diagrams* [10]. These diagrams are very useful for illustrating intervals containing *NOW*. The x-axis denotes reference time, the time when an interval is observed. The y-axis denotes valid time. The regions in these diagrams then convey the (possibly) time-varying meanings, or extensionalizations, of intervals in tuples stored in the database.

For intervals without *NOW*, extensionalization diagrams convert the illustration of an interval from a line, as shown in Figure 1, to a rectangle, as shown in Figure 3A. Figures 3A and B illustrate an update not involving *NOW*. The region bounded by the solid line representents the tuple stored in the database, and the region bounded by the dashed line represents the modification. The solid rectangle in Figure 3A indicates that Joe is with the Shoe department in the interval [5,15). This information was stored at time 5 and extends to the right. The dashed rectangle indicates that at time 15, we update Joe to be with the Toy department in the interval [10,20). The result is shown in Figure 3B, which shows that Joe is now with the Shoe department in the interval [5,10) and with the Toy department in the interval [10,15). We cannot update Joe for the interval [15,20) because there is no information to update in this interval. The update only affects the overlap of the two rectangles.

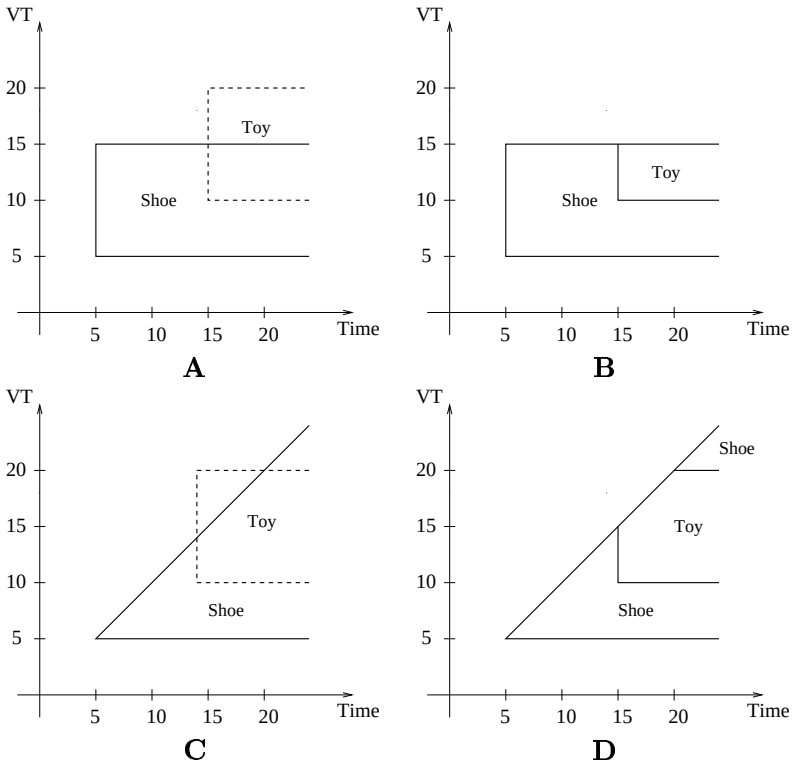

Figure 3: The Results of Updates Without and With *NOW*

In Figure 3C, we show an update involving *NOW*. The database contains the tuple ⟨Joe, Shoe, [5,*NOW*)⟩, indicated by the solid triangle. The end time of *NOW* makes the top of the

extensionalization of the tuple follow the diagonal; at time 6, the interval is [5,6), at time 7, the interval is [5,7), and so on. Again, we update Joe to be with the Toy department in the interval [10,20).

Figure 3D shows the desired result of the update where, as for updates without NOW, we update only the overlap of the region specified in the update and the region specified by the tuple (as shown in Figure 3C). By updating exactly the overlap, we avoid basing the semantics on assumptions such as the optimistic or pessimistic assumptions discussed in the introduction.

Having motivated the desired semantics, the next task is to precisely define the semantics and to illustrate how these semantics can be accommodated with two new types of time-attribute values.

4.3 Road Map for Accommodating the New Semantics

The remainder of this section defines the semantics of modifications involving NOW. The goal is to define a semantics consistent with the semantics for ground databases defined in Section 3.2 and to reuse the template used there.

To accommodate intervals containing NOW, as demonstrated in Section 4.2, we have to extend the interval intersection ($\cap$) and difference ($-$) operators used in the definitions of delete and update in Section 3.2. As the first step in doing so, we must determine the set of values for interval end-points that we have to store in the database and that the generalized operators must then contend with. For example, we must determine what interval-end points are needed to accurately store the intersection of two intervals such as [5,NOW) and [10,20) in the database (see the ⟨Joe, Toy⟩ tuple in Figure 3D).

We extend the domain of interval end-points with two additional types, each of which can be efficiently represented (we term such intervals *normal form intervals*). We then proceed to define the extensions of the interval intersection and difference operators in Section 4.6 and use these new operators for defining the semantics of modifications involving NOW in Section 4.7.

4.4 Auxiliary Functions

So far, we have employed the time domain $\mathcal{T}$ that is isomorphic to a subset of the natural numbers and contains only ground values. We proceed to introduce time domain $\mathcal{T}_1 = \mathcal{T} \cup NOW$ that includes the variable NOW. While including the variable NOW is convenient for end-users, we need to provide a semantics for variable databases. We do so by means of a mapping from a variable database to a fully ground data model, which does not include such variables. A theoretical framework for providing a logical interpretation, or "meaning," of a variable database, i.e., a "translation" from variable to extensional level, may be based on a homomorphic mapping from variable-level databases to extensional-level databases [9]. This mapping is termed an *extensionalization*, and is denoted $[\![\ ]\!]$.

The extensionalization of an element of the time domain at a time c ("c" for chronon) denotes its value on the y-axis of the extensionalization diagram. Returning to Figure 3D, the extensionalization of the start time for ⟨Joe, Shoe⟩ is 5, for any time $c \geq 5$. The extensionalization of the stop time of ⟨Joe, Shoe⟩ is more interesting: at $c = 5$ it is 5, at $c = 10$, 10, at $c = 15$, it is back to 10.

With the domain of a being $\mathcal{T}_1$, we define the extensionalization of a time value at time c as follows.

$$[\![a]\!]_c \triangleq \begin{cases} c & \text{if } a = NOW \\ a & \text{otherwise} \end{cases}$$

As examples, $[\![5]\!]_5 = 5$, $[\![5]\!]_{17} = 5$, and $[\![NOW]\!]_{17} = 17$. Note that the extensionalization is always an element of $\mathcal{T}$, and is thus isomorphic to (and can be represented by) a natural number.

We want to reuse the framework for defining modifications in Section 3 when defining modifications for variable databases. Therefore, we need generalized minimum and maximum functions, min^v and max^v, respectively, that accommodate the variable NOW.

Let the domain of a and b be defined recursively as $\mathcal{T}_2 = \mathcal{T}_1 \cup \{min^v(a,b)\} \cup \{max^v(a,b)\}$, where a and b are elements of $\mathcal{T}_2$. $\mathcal{T}_2$ is a very general domain, consisting of natural numbers and expressions containing arbitrarily nested applications of min^v and max^v. Examples of elements of $\mathcal{T}_2$ include 5, $min^v(5,17)$, and $min^v(6, max^v(4, NOW))$. In the next section we will restrict this domain considerably. However, the exposition is smoother if we start with this general domain.

Elements of $\mathcal{T}_2$ involving NOW are particularly interesting. We show three such examples in Figure 4 using extensionalization diagrams. The examples show that the min^v function (A) gives the time value an upper bound, that the max^v function (C) gives the time value a lower bound, and that the nesting of a max^v function in a min^v function (B) gives the time value both a lower and an upper bound. Notice that the sloping lines follow the diagonal.

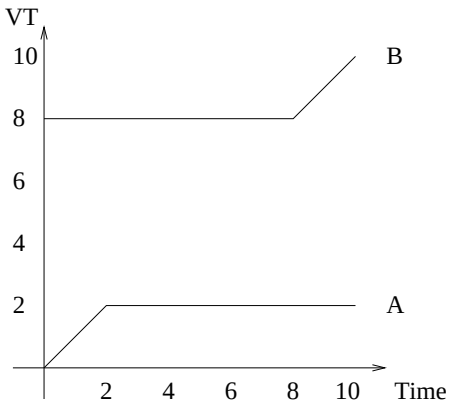

Figure 4: (A) $min^v(2, NOW)$, (B) $min^v(6, max^v(4, NOW))$, and (C) $max^v(8, NOW)$

We define the meaning of the $min^v(a,b)$ and $max^v(a,b)$ functions via their extensionalizations.

$$\begin{aligned} [\![min^v(a,b)]\!]_c &\stackrel{\triangle}{=} min([\![a]\!]_c, [\![b]\!]_c) \\ [\![max^v(a,b)]\!]_c &\stackrel{\triangle}{=} max([\![a]\!]_c, [\![b]\!]_c) \end{aligned}$$

The min^v and max^v functions reduce to their conventional counter-part when the arguments are from the domain $\mathcal{T}$.

$$\begin{aligned} min^v(a,b) &= min(a,b) \quad \text{if } a,b \in \mathcal{T} \\ max^v(a,b) &= max(a,b) \quad \text{if } a,b \in \mathcal{T} \end{aligned}$$

Note that the extensionalization of each element of $\mathcal{T}_2$ is also a natural number.

The extensionalization of an interval [a,b) is defined as follows.

$$[\![[a,b)]\!]_c \stackrel{\triangle}{=} [[\![a]\!]_c, [\![b]\!]_c)$$

4.5 Interval Types for Modifications Involving NOW

We use a domain of interval end-points given by $\mathcal{T}_f = \mathcal{T} \cup \{min^v(a, NOW),\ max^v(a, NOW)\}$, where $a \in \mathcal{T}$, which restricts domain $\mathcal{T}_2$ from the previous section. It turns out that this domain of values is sufficient for representing the results of the modifications that we are about to define

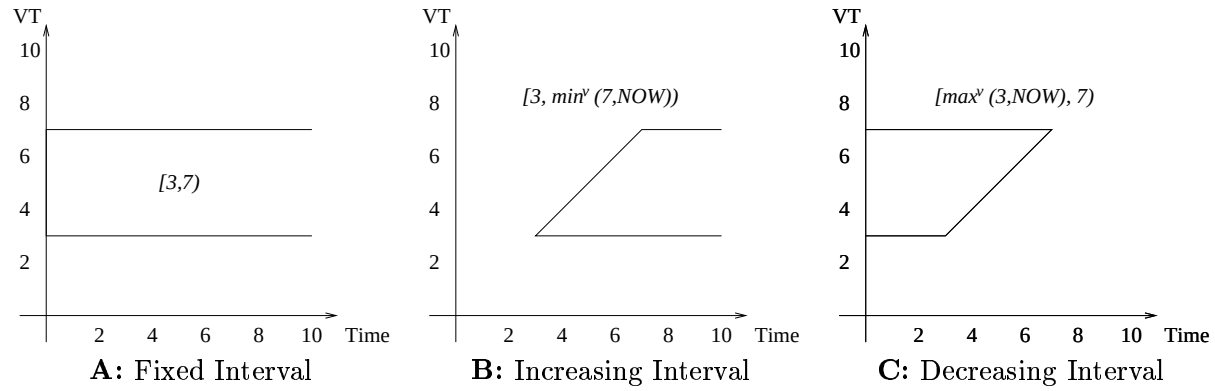

Figure 5: Interval Types Needed for Modifications Involving NOW

formally. Intervals using the three types of values in $\mathcal{T}_f$, termed *canonical intervals*, are shown in Figure 5.

Figure 5A exemplifies a conventional fixed interval of type $[a, b)$, specifically, $[3, 7)$. Figure 5B shows an increasing interval of type $[a, min^v(b, NOW))$, specifically $[3, min^v(7, NOW))$. Note that the interval in Figure 5B may continue to grow; this is specified as $[a, min^v(forever, NOW))$. Figure 5C shows a decreasing interval of type $[max^v(a, NOW), b)$, specifically, $[max^v(3, NOW), 7)$. Note again the special case, specified as $[max^v(beginning, NOW), b)$, where the interval starts at *beginning*. Also note that as time proceeds, the duration of the interval shrinks, until the interval is empty.

4.6 Extending the Interval Difference and Intersection Operators

Having established $\mathcal{T}_f$ as the domain of interval end points, the next step is to extend the interval difference and intersection operators to apply over such intervals. These operators are used in the definition of conventional modifications (Section 3.2) and will be used in the next section to define the semantics of modifications involving NOW.

The cases we must consider when extending the interval difference and intersection operators are the following, where the domain of a, b, c, and d is $\mathcal{T}$, and *int-opr* is the extended difference operator ($-^v$) or intersect operator ($\cap^v$).

$$\left\{ \begin{array}{c} [a, b) \\ [max^v(a, NOW), b) \\ [a, min^v(b, NOW)) \end{array} \right\} \quad \textit{int-opr} \quad \left\{ \begin{array}{c} [c, d) \\ [max^v(c, NOW), d) \\ [c, min^v(d, NOW)) \end{array} \right\} \tag{1}$$

Let the domain of α, β, γ, and δ be $\mathcal{T}_f$. The extended interval difference operators ($-^v$) is defined as follows.

$$[\alpha,\beta) \ -^v \ [\gamma,\delta) \ \triangleq \begin{cases} \{[\alpha,\gamma),[\delta,\beta)\} & \text{if } (\alpha,\beta,\gamma,\delta \in \mathcal{T} \wedge (\alpha<\delta \wedge \gamma<\beta)) \\ \{[\alpha,c),[\delta,\beta),[max(\alpha,c),min^v(min(\beta,\delta),NOW))\} & \\ \qquad \text{if } (\alpha,\beta,\delta \in \mathcal{T} \wedge \gamma = max^v(c,NOW) \wedge (\alpha<\delta \wedge c<\beta)) & \\ \{[\alpha,\gamma),[d,\beta),[max^v(max(\alpha,\gamma),NOW),min(\beta,d))\} & \\ \qquad \text{if } (\alpha,\beta,\gamma \in \mathcal{T} \wedge \delta = min^v(d,NOW) \wedge (\alpha<d \wedge \gamma<\beta)) & \\ \{[max^v(a,NOW),c),[max^v(\delta,NOW),\beta)\} & \\ \qquad \text{if } ((\alpha = max^v(a,NOW) \wedge \beta,\delta \in \mathcal{T}) \wedge & \\ \qquad\quad (\gamma = c \vee \gamma = max^v(c,NOW)) \wedge (a<\delta \wedge c<\beta)) & \\ \{[\alpha,min^v(\gamma,NOW)),[d,min^v(b,NOW))\} & \\ \qquad \text{if } ((\alpha,\gamma \in \mathcal{T} \wedge \beta = min^v(b,NOW)) \wedge & \\ \qquad\quad (\delta = d \vee \delta = min^v(d,NOW)) \wedge (a<d \wedge c<b)) & \\ \{[\alpha,\beta)\} & \text{otherwise} \end{cases}$$

The definition accommodates all nine cases. Note first that the extended difference operator reduces to the conventional difference operator if the domain of end points is $\mathcal{T}$. These situations are covered by the first and last cases in the definition.

Second, note that the third case in the definition takes a constant and an increasing interval as inputs and returns a decreasing interval. This corresponds to subtracting the shape in Figure 5B from the shape in Figure 5A, returning a shape as shown in Figure 5C. As an example, Figure 6 visualizes $[3,7) -^v [4,min^v(6,NOW))$, illustrating also how the third case can return three intervals. In Figure 6A, the extensionalization of the interval $[3,7)$ is illustrated by the solid lines, and the

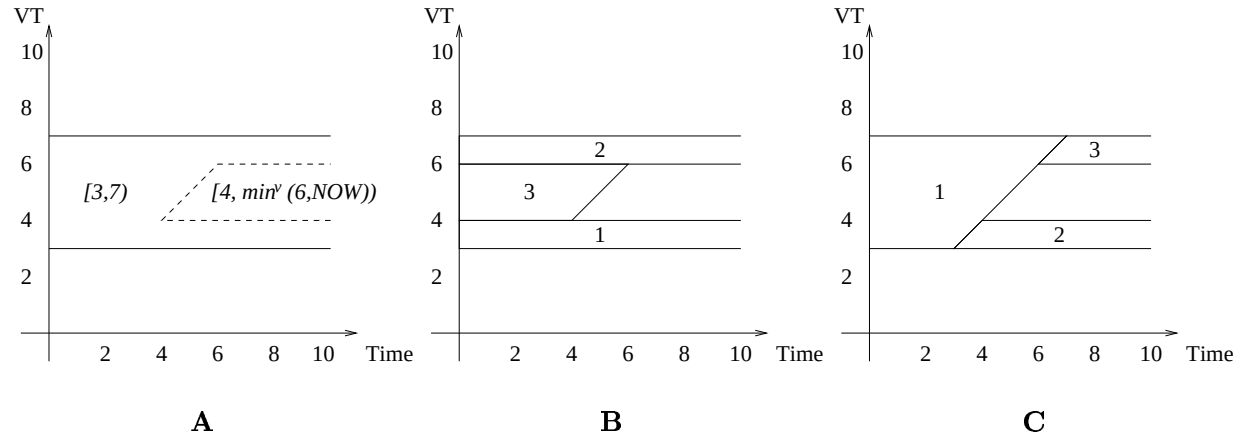

Figure 6: Extended Interval Difference Example

extensionalization of the interval $[4,min^v(6,NOW))$ is given by the dashed line. The three result intervals are shown in Figure 6B. The first interval is $[3,4)$, the second is $[6,7)$, and the third is $[max^v(4,NOW),6)$. This corresponds to the order in which the intervals are specified in the definition of $-^v$. (Figure 6C will be addressed shortly.)

The remaining cases in the definition of the difference can be understood in a similar way.

The extended interval intersection operator ($\cap^v$) is defined next.

$$
[\alpha,\beta)\ \cap^v\ [\gamma,\delta) \triangleq
\begin{cases}
[max(\alpha,\gamma), min(\beta,\delta)) \\
\qquad\qquad \text{if } (\alpha,\beta,\gamma,\delta \in \mathcal{T} \wedge (\alpha < \delta \wedge \gamma < \beta)) \\
[max^v(max(a,c), NOW), min(\beta,\delta)) \\
\qquad\qquad \text{if } ((\alpha = a \vee \alpha = max^v(a,NOW)) \wedge (\gamma = c \vee \gamma = max^v(c,NOW)) \wedge \\
\qquad\qquad\quad (\beta,\delta \in \mathcal{T}) \wedge (a < \delta \wedge c < \beta)) \\
[max(\alpha,\gamma), min^v(min(b,d), NOW)) \\
\qquad\qquad \text{if } ((\alpha,\gamma \in \mathcal{T}) \wedge (\beta = b \vee \beta = min^v(b,NOW)) \wedge \\
\qquad\qquad\quad (\delta = d \vee \delta = min^v(d,NOW)) \wedge (\alpha < d \wedge \gamma < b)) \\
\emptyset \qquad\qquad \text{otherwise}
\end{cases}
$$

Note again that the extended intersection operator reduces to the conventional intersection operator if the argument interval end points are in $\mathcal{T}$. This is handled by the first and last cases.

The extensionalization diagrams in Figure 7 explain the second case in the definition of $\cap^v$; the remaining cases may be explained similarly. The example computes $[3,7) \cap^v [max^v(2,NOW),6)$.

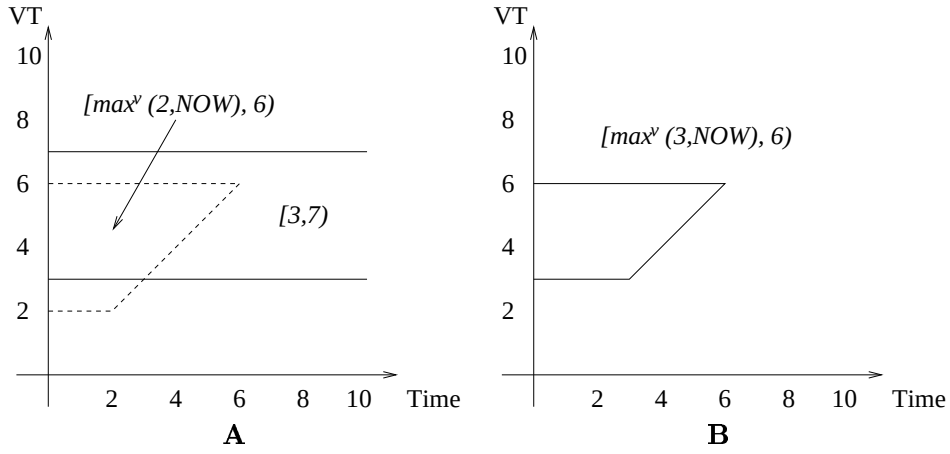

Figure 7: Extended Interval Intersection Example

In Figure 7A, the interval $[3,7)$ is illustrated by the solid lines, and $[max^v(2,NOW),6)$ is given by the dashed line. The resulting interval, $[max^v(3,NOW),6)$, is shown in Figure 7B.

We have aimed to make the extended interval difference and intersection operators as similar as possible to their conventional counterparts. This makes the extended operators easier to understand. Below we summarize the similarities and then the differences.

- The extended operators are equivalent to their conventional counterparts if all interval end points are in $\mathcal{T}$. This was required in Section 4.1.
- The results returned by the extended operators are independent of the time they are evaluated, as is the case for the conventional operators. This independence follows from the definitions, where no comparisons to the current time occur.
- The intervals returned by the extended difference operator are disjoint. For example, the three result intervals in Figure 6B do not overlap.
- The intervals returned by the extended difference operators are coalesced [5], so a result containing, e.g., $[5,7)$ and $[7,8)$ will not occur. Instead the interval $[5,8)$ would be returned. Again this is fulfilled by design.

- The extended intersection operator returns at most one interval, as does the conventional intersection operator.

There are three differences between the extended and the conventional operators.

- The extended difference and intersection operators accommodate intervals defined by using the variable NOW. This was a requirement in Section 4.1.
- The extended difference operator returns from zero to three intervals, whereas the conventional difference operator returns zero, one, or two intervals. In the example in Figure 6, three intervals are returned. This is unavoidable and happens when finding the difference between a constant and a non-constant interval, where the non-constant interval is included in the constant interval, e.g., $[10, 15) -^v [12, min^v(14, NOW))$ and $[10, 15) -^v [max^v(12, NOW), 14)$.
- The results of one extended difference operation can be combined in several ways, whereas the result of the conventional difference operator is unique. As an example, the result shown in Figure 6B can also be given as the following three intervals: $[max^v(3, NOW), 7)$, $[3, min^v(4, NOW))$, and $[6, min^v(7, NOW))$, as shown in Figure 6C.

4.7 Temporal Modification Semantics

With the definitions of the extended difference and intersection operators in place, we can define the semantics of modifications involving NOW. Examples follow the definitions.

The definitions of the insert, delete, and update of intervals involving NOW on a valid-time relation r_{vt} are identical to those of modifications without NOW defined in Section 3.2, except that the extended versions of the interval difference and intersection operators are used. The definitions are shown below.

Insertion into a valid-time relation r_{vt} is defined as follows. Again, if the validity interval is not explicitly provided, a default of (bound) NOW to forever is used.

`VALIDTIME PERIOD [`vts`,`vte`) INSERT INTO` r_{vt} `VALUES (`A`)` $\stackrel{\triangle}{=}$

$$r_{vt} \leftarrow r_{vt} \cup^{vt} \{(A, [vts, vte))\}$$

Deletion from a valid-time relation r_{vt} is defined as follows.

`VALIDTIME PERIOD [`vts`,`vte`) DELETE FROM` r_{vt} `WHERE` $cond$ $\stackrel{\triangle}{=}$

$$\begin{aligned} r_{vt} \leftarrow\ & \{t | t \in r_{vt}(\neg cond(t))\} \cup^{vt} \\ & \{t | \exists s \in r_{vt}(cond(s) \wedge t[A] = s[A] \wedge t[\mathrm{VT}] \in (s[\mathrm{VT}] -^v [vts, vte)) \wedge t[\mathrm{VT}] \neq \emptyset)\} \end{aligned}$$

Update of a valid-time relation r_{vt} is defined as follows.

`VALIDTIME PERIOD [`vts`,`vte`) UPDATE` r_{vt} `SET` A `=` v `WHERE` $cond$ $\stackrel{\triangle}{=}$

$$\begin{aligned} r_{vt} \leftarrow\ & \{t | t \in r_{vt}(\neg cond(t))\} \cup^{vt} \\ & \{t | \exists s \in r_{vt}(cond(s) \wedge t[A] = s[A] \wedge t[\mathrm{VT}] \in (s[\mathrm{VT}] -^v [vts, vte)) \wedge t[\mathrm{VT}] \neq \emptyset)\} \cup^{vt} \\ & \{t | \exists s \in r_{vt}(cond(s) \wedge t[A] = v \wedge t[\mathrm{VT}] = (s[\mathrm{VT}] \cap^v [vts, vte)) \wedge t[\mathrm{VT}] \neq \emptyset)\} \end{aligned}$$

Note that the first two lines are identical to the two lines of the delete. Updates are similar to a delete followed by an insert; this similarity will be exploited in the implementation described in Section 6.3.

We examine two sample modifications. First an example of an update without NOW is given. Assume the database contains the tuple $\langle$Joe, Shoe, $[5, 20)\rangle$ and that we want to update Joe to be with the Toy department in the interval $[10, 15)$. This may be written as follows.

```
VALIDTIME PERIOD [10,15) UPDATE Emp SET Dept = 'Toy' WHERE Name = 'Joe'
```

The result of the update, also illustrated in Figure 8, is as follows.

$$\emptyset \cup^{vt} \{\langle \text{Joe}, \text{Shoe}, \{[5,20) -^{v} [10,15)\}\rangle\} \cup^{vt} \{\langle \text{Joe}, \text{Toy}, \{[5,20) \cap^{v} [10,15)\}\rangle\}$$
$$= \{\langle \text{Joe}, \text{Shoe}, [5,10)\rangle,\ \langle \text{Joe}, \text{Shoe}, [15,20)\rangle,\ \langle \text{Joe}, \text{Toy}, [10,15)\rangle\}$$

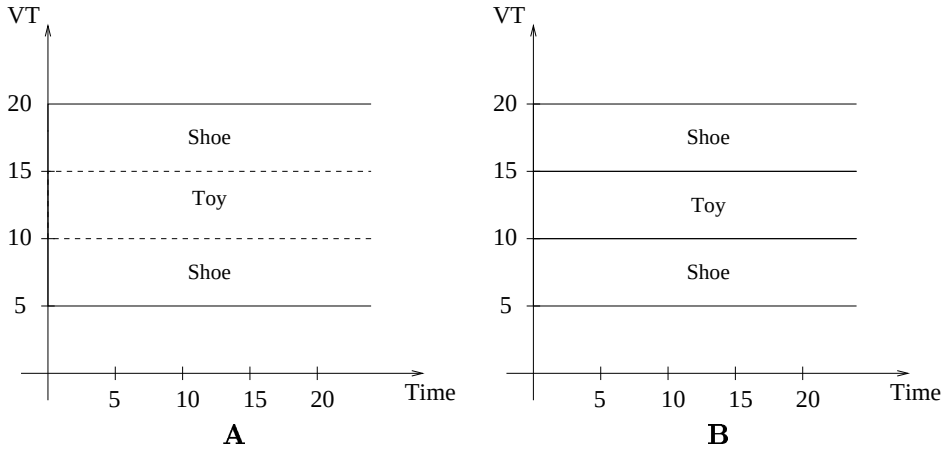

Figure 8: Joe is in the Toy department for the interval from 10 to 15.

From the single tuple stored in the relation, we remove Joe from the Shoe department in the interval $[10, 15)$. This results in two tuples. Further, we update Joe to be with the Toy department in the intersection of the intervals $[5, 20)$ and $[10, 15)$. The result is the same as the result obtained by using the earlier definition of update for ground databases.

The next update involves NOW. We use the update in Figure 1 and assume that the database contains the tuple $\langle \text{Joe}, \text{Shoe}, [5, NOW)\rangle$. This update may be written as follows.

```
VALIDTIME PERIOD [10,20) UPDATE Emp SET Dept = 'Toy' WHERE Name = 'Joe'
```

The result of the update is as follows.

$$\emptyset \cup^{vt} \{\langle \text{Joe}, \text{Shoe}, \{[5, NOW) -^{v} [10,20)\}\rangle\} \cup^{vt} \{\langle \text{Joe}, \text{Toy}, \{[5, NOW) \cap^{v} [10,20)\}\rangle\}$$
$$= \{\langle \text{Joe}, \text{Shoe}, [5, min^{v}(10, NOW))\rangle,\ \langle \text{Joe}, \text{Shoe}, [20, min^{v}(forever, NOW))\rangle,$$
$$\langle \text{Joe}, \text{Toy}, [10, min^{v}(20, NOW))\rangle\}$$

The resulting tuples contain min^{v} functions and are easily explained by the diagrams in Figure 9. The solid line denotes the tuple stating that Joe was with the Shoe department in the interval $[5, NOW)$. The dashed rectangle corresponds to the interval [10,20) for which the update is to be applied. The update takes effect in the region where the solid-line and dashed-line regions overlap, and the result is given in Figure 9B.

This result is the desired one. We only update in the overlap between the temporal scope specified in the update statement and the data stored in the database. It is still correct that Joe was with the Shoe department in the interval $[5, min^{v}(10, NOW))$ and $[20, min^{v}(forever, NOW))$. Should Joe also have been updated to be with the Toy department in the latter two intervals, a different temporal scope should have been given in the update statement, e.g, the interval $[5, NOW)$.

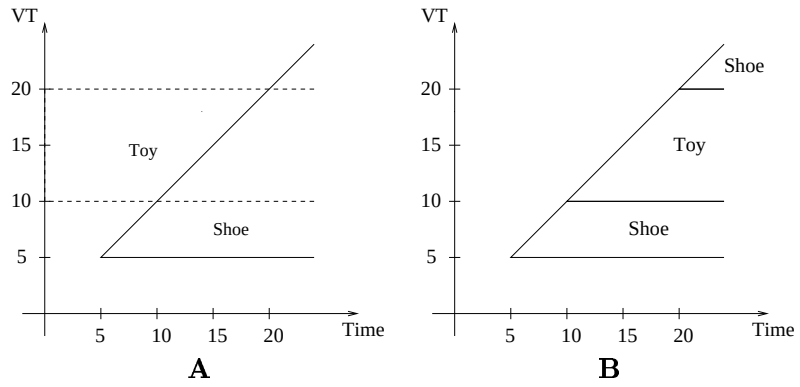

Figure 9: Update of a NOW-Relative Database

5 Semantics of Modifications Involving Now-Relative Values

In some applications, the intervals associated with the tuples do not coincide with the current time, but still vary with the current time. For example, the hiring and termination of personnel may be recorded in the database only three days after they are effective. For cases like these, *now-relative* time values, e.g., $NOW - 3$ days, which track the current time, but with a displacement, and which generalize NOW, are very useful [10].

This section considers the modification of databases in the presence of such values. First, NOW-relative values are defined, and then a new kind of interval, used for accommodating the more general databases that result from the NOW-relative values, is introduced, and the interval operations ($-^v$ and $\cap^v$) are extended to also accommodate these new intervals. On this basis, the modifications are defined.

5.1 Definition of NOW-Relative Values

NOW-relative values generalize variable NOW by allowing offsets from NOW to be specified [10]. For example, assume that Joe started in the Shoe department on January 10 and remains there, but may be assigned to another department with two days' notice. This may be captured using a NOW-relative value, as follows: $\langle$Joe, Shoe, [10,$NOW + 2$)$\rangle$, where the $+2$ indicates the two days' notice.

Formally, the extensionalization at time c of a NOW-relative value, NOW OP n, where OP $\in \{+, -\}$ and n belongs to a domain of durations that is isomorphic to a subset of the integers, is defined as follows [10].

$$[\![NOW \text{ OP } n]\!]_c \stackrel{\triangle}{=} [\![NOW]\!]_c \text{ OP } n$$

5.2 A New Interval Type

To extend the modifications to accommodate NOW-relative intervals, a single new interval type is needed over the three introduced in Section 4.5. The extensionalization graph in Figure 10 gives an example of this new type of interval, namely the interval $[max^v(3, NOW - 3), min^v(7, NOW + 2))$. The dashed line indicates the diagonal. This interval has a max^v function in its starting point and a min^v function in its ending point; the earlier intervals had at most a function in either the starting or the ending point. Note also the now-relative offsets, -3 and $+2$. The -3 in the starting point indicates that the start point is three units below the diagonal, and the $+2$ indicates that the ending point is two units above the diagonal. In previous sections all offsets were 0 and all non-vertical or non-horizontal lines were on the diagonal.

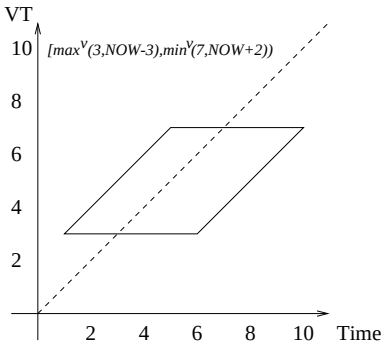

Figure 10: New Interval Type for NOW-Relative Modifications

The offset of the starting point of an interval must be smaller than or equal to the offset of its ending point. Otherwise, the interval is undefined. For example, the interval $[max^v(3, NOW + 2), min^v(7, NOW - 3))$ is undefined.

Formally, we define the meaning of a NOW-relative interval at time c as follows, where a and b are in $\mathcal{T}$ and a_off and b_off are in the domain of durations.

$$[\![[max^v(a, NOW + a\_off), min^v(b, NOW + b\_off))]\!]_c \stackrel{\triangle}{=}$$

$$\begin{cases} [a, min^v(b, NOW + b\_off)) & \text{if } a - b\_off \leq c < b - b\_off \\ [a, b) & \text{if } b - b\_off \leq c < a - a\_off \\ [max^v(a, NOW + a\_off), b) & \text{if } a - a\_off \leq c < b - a\_off \\ \emptyset & \text{otherwise} \end{cases}$$

We will show next how the new interval type comes into existence and define also how it is handled in the interval difference and intersection operators.

5.3 Extending the Interval Operators

The sixteen cases we must consider when extending the interval difference and intersection operators are enumerated below, where the domain of a, b, c, and d is $\mathcal{T}$, a_off, b_off, c_off, and d_off are signed durations (i.e., corresponding to positive or negative integers), and *int-opr* is the extended difference operator ($-^v$) or intersection operator ($\cap^v$).

$$\left\{ \begin{array}{ll} 1: & [a, b) \\ 2: & [a, min^v(b, NOW + b\_off)) \\ 3: & [max^v(a, NOW + a\_off), b) \\ 4: & [max^v(a, NOW + a\_off), min^v(b, NOW + b\_off)) \end{array} \right\} \; \textit{int-opr} \; \left\{ \begin{array}{ll} 1: & [c, d) \\ 2: & [c, min^v(d, NOW + d\_off)) \\ 3: & [max^v(c, NOW + c\_off), d) \\ 4: & [max^v(c, NOW + c\_off), min^v(d, NOW + d\_off)) \end{array} \right\} \quad (2)$$

Note that allowing only '+', but then allowing positive and negative durations is equivalent to the definition of now-relative values in Section 5.1.

The extended interval difference operation, $[\alpha, \beta) \; -^v \; [\gamma, \delta)$, where the argument intervals are as enumerated above, is defined in Table 2. The table has seventeen cases, and each of the sixteen combinations above gives rise to two cases. The first case for a combination is identified by the integers in Formula 2 and in the second and third columns in the table (the first "column" is used

for numbering the cases in the discussion below). The fourth column gives a condition that must be satisfied for this case to apply. The second case is the last, "otherwise" case in the table, which applies if the condition in the first case is not satisfied. The result for a case, a set of intervals, is given in the last column of the table.

Case	L	R	Condition	Resulting Rows
1	1	1	$\alpha < \delta \wedge \gamma < \beta$	$[\alpha, \gamma), [\delta, \beta)$
2	1	2	$\alpha < d \wedge \gamma < \beta$	$[\alpha, \gamma), [d, \beta), [max^v(max(\alpha, \gamma), NOW + d\_off), min(\beta, d))$
3	1	3	$\alpha < \delta \wedge c < \beta$	$[\alpha, c), [\delta, \beta), [max(\alpha, c), min^v(min(\beta, \delta), NOW + c\_off))$
4	1	4	$\alpha < d \wedge c < \beta$	$[\alpha, c), [d, \beta), [max(\alpha, c), min^v(min(\beta, d), NOW + c\_off)),$ $[max^v(max(\alpha, c), NOW + d\_off), min(\beta, d))$
5	2	1	$\alpha < \delta \wedge \gamma < b$	$[\alpha, min^v(\gamma, NOW + b\_off)), [\delta, min^v(b, NOW + b\_off))$
6	2	2	$\alpha < d \wedge \gamma < b$	$[\alpha, min^v(\gamma, NOW + b\_off)), [d, min^v(b, NOW + b\_off)),$ $[max^v(max(\alpha, \gamma), NOW + d\_off), min^v(min(b, d), NOW + b\_off))$
7	2	3	$\alpha < \delta \wedge c < b$	$[\alpha, min^v(c, NOW + b\_off)), [\delta, min^v(b, NOW + b\_off)),$ $[max(\alpha, c), min^v(min(b, \delta), NOW + c\_off))$
8	2	4	$\alpha < d \wedge c < b$	$[\alpha, min^v(c, NOW + b\_off)), [d, min^v(b, NOW + b\_off)),$ $[max(\alpha, c), min^v(min(b, d), NOW + c\_off)),$ $[max^v(max(\alpha, c), NOW + d\_off), min^v(min(\beta, d), NOW + b\_off))$
9	3	1	$a < \delta \wedge \gamma < \beta$	$[max^v(a, NOW + a\_off), \gamma), [max^v(\delta, NOW + a\_off), \beta)$
10	3	2	$a < d \wedge \gamma < \beta$	$[max^v(a, NOW + a\_off), \gamma), [max^v(d, NOW + a\_off), \beta),$ $[max^v(max(a, \gamma), NOW + d\_off), min(\beta, d))$
11	3	3	$a < \delta \wedge c < \beta$	$[max^v(a, NOW + a\_off), c), [max^v(\delta, NOW + a\_off), \beta),$ $[max^v(max(a, c), NOW + a\_off), min^v(min(\beta, \delta), NOW + c\_off))$
12	3	4	$a < d \wedge \gamma < b$	$[max^v(a, NOW + a\_off), c), [max^v(d, NOW + a\_off), \beta),$ $[max^v(max(a, c), NOW + a\_off), min^v(min(\beta, d), NOW + c\_off)),$ $[max^v(max(a, c), NOW + d\_off), min(\beta, d))$
13	4	1	$a < \delta \wedge \gamma < b$	$[max^v(a, NOW + a\_off), min^v(\gamma, NOW + b\_off)),$ $[max^v(\delta, NOW + a\_off), min^v(b, NOW + b\_off))$
14	4	2	$a < d \wedge \gamma < b$	$[max^v(a, NOW + a\_off), min^v(\gamma, NOW + b\_off)),$ $[max^v(d, NOW + a\_off), min^v(b, NOW + b\_off)),$ $[max^v(max(a, \gamma), NOW + d\_off), min^v(min(b, d), NOW + b\_off))$
15	4	3	$a < \delta \wedge c < b$	$[max^v(a), NOW + a\_off), min^v(c, NOW + b\_off)),$ $[max^v(d, NOW + a\_off), min^v(b, NOW + b\_off)),$ $[max^v(max(a, c), NOW + a\_off), min^v(min(b, \delta), NOW + c\_off))$
16	4	4	$a < d \wedge c < b$	$[max^v(a, NOW + a\_off), min^v(c, NOW + b\_off)),$ $[max^v(d, NOW + a\_off), min^v(b, NOW + b\_off)),$ $[max^v(max(a, c), NOW + a\_off), min^v(min(b, d), NOW + c\_off)),$ $[max^v(max(a, c), NOW + d\_off), min^v(min(b, d), NOW + b\_off))$
17			otherwise	$[\alpha, \beta)$

Table 2: The Interval Difference Operator Extended for NOW-Relative Values

It may be observed that the extended difference operator reduces to the conventional difference operator when the interval end points are in $\mathcal{T}$. These situations are covered by the first and last cases in the definition. Next, the operator returns up to four intervals (in cases 4, 8, 12, and 16), and the new interval type defined in Section 5.2 is returned in cases 6, 8, and 11–16.

We motivate the definition by considering the four examples illustrated in Figure 11. Figure 11A illustrates the difference $[5, min^v(9, NOW + 2)) -^v [3, 7)$, which is covered by case 5 in Table 2. The result, $[7, min^v(9, NOW + 2))$, is illustrated in Figure 12A. The first interval in case 5 is empty because $\alpha > \gamma$ $(5 > 3)$.

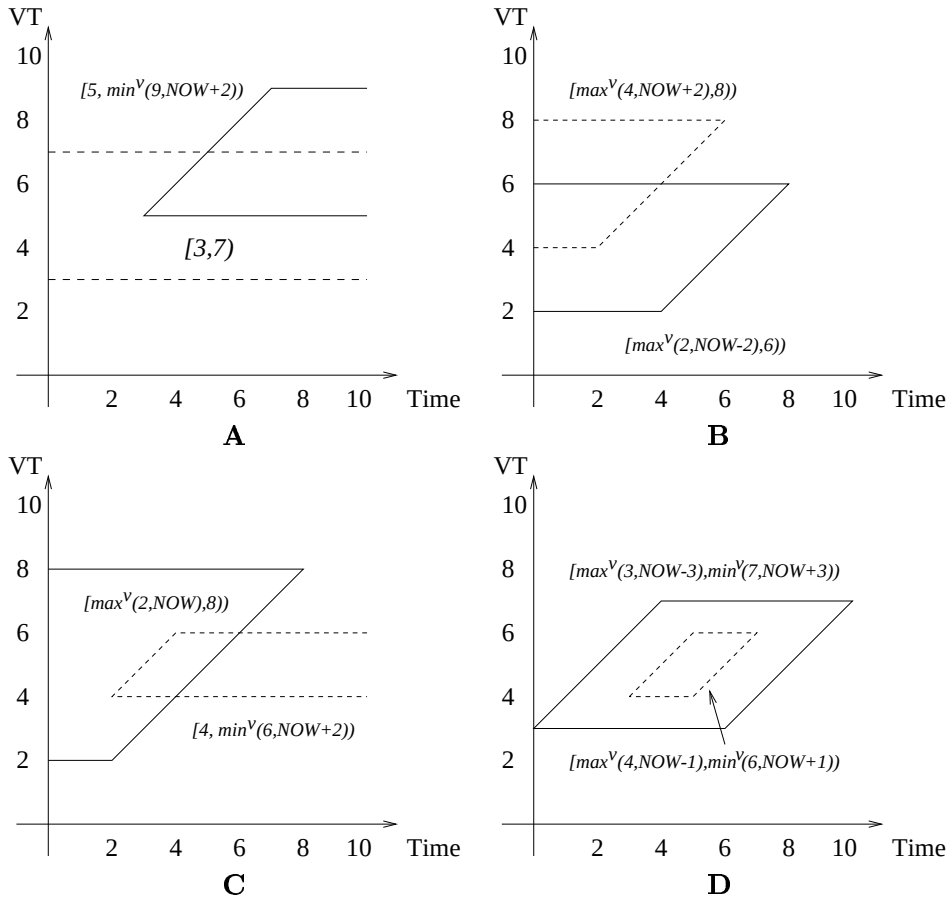

Figure 11: NOW-Relative Interval Difference Examples

Figure 11B illustrates the difference $[max^v(2, NOW - 2), 6) -^v [max^v(4, NOW + 2), 8)$, which is covered by case 11 in the definition. The result is the two intervals $[max^v(2, NOW - 2), 4)$ and $[max^v(4, NOW - 2), min^v(6, NOW + 2))$, which are shown in Figure 12B. These intervals derive from the first and third intervals in case 11; the second interval is empty because $\delta > \beta$ $(8 > 6)$. When we compute the difference of two decreasing intervals as here, the new type of interval from Section 5.2 is results.

Figure 11C illustrates $[max^v(2, NOW), 8) -^v [4, min^v(6, NOW + 2))$. Here, the offset of the first interval is 0. Finding the difference of a decreasing and increasing interval (or vice versa) is not as simple as when both offsets are 0, where the left argument is the result. Case 10 in the definition applies, and the result is the three intervals $[max^v(2, NOW), 4)$, $[max^v(6, NOW), 8)$, and $[max^v(4, NOW + 2), 6)$, see Figure 12C.

Finally, Figure 11D illustrates $[max^v(3, NOW - 3), min^v(7, NOW + 3)) -^v [max^v(4, NOW - 1), min^v(6, NOW+1))$, which is covered by case 16. The result is the four intervals $[max^v(3, NOW - 3), min^v(4, NOW + 3))$, $[max^v(6, NOW - 3), min^v(7, NOW + 3))$, $[max^v(4, NOW - 3), min^v(6, NOW + 1))$, and $[max^v(4, NOW + 1), min^v(6, NOW + 3))$, shown in Figure 12D. This example illustrates the overall strategy used in defining interval difference. We look above, below, to the right, and to the left of the second argument interval, determining what remains of the first argument interval. Looking to the right and left, we consider only the parts of the first argument interval that have not been covered by looking above and below. This is implemented in the definition using the standard max and min functions. Using this strategy, the difference operator returns non-overlapping regions and does not duplicate information.

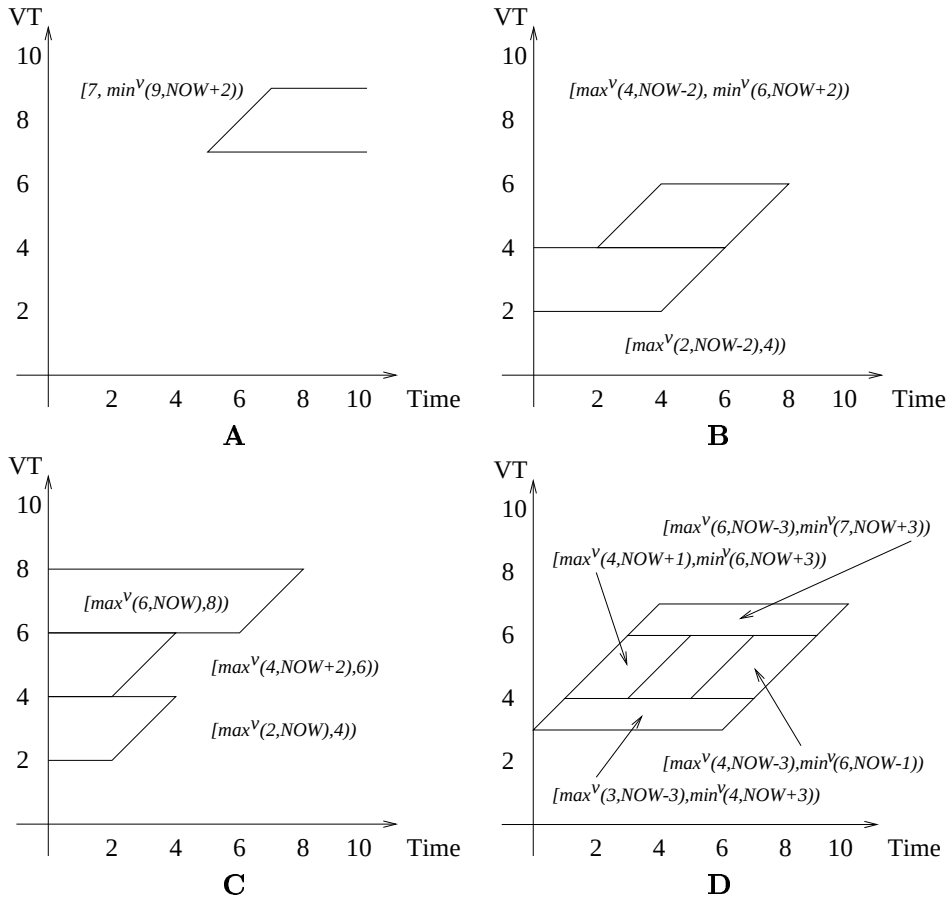

Figure 12: NOW-Relative Interval Difference Results

The extended interval intersection operator ($\cap^v$) is defined below. Like the difference operator above, this operator reduces to the conventional intersection operator if the end points of the argument intervals are in $\mathcal{T}$, which also here is covered by the first and last cases in the definition. The operator operator always returns only one interval, as does the conventional intersection operator, and the new interval type is returned by case 4.

$$
\begin{aligned}
&[\alpha,\beta)\ \cap^v\ [\gamma,\delta) \triangleq \\
&\begin{cases}
[max(\alpha,\gamma), min(\beta,\delta)) \\
\qquad \text{if } (\alpha,\beta,\gamma,\delta \in \mathcal{T} \wedge (\alpha < \delta \wedge \gamma < \beta)) \\
[max^v(max(a,c), NOW + max(a\_off, c\_off)), min(\beta,\delta)) \\
\qquad \text{if } ((\alpha = a \vee \alpha = max^v(a, NOW + a\_off)) \wedge (\gamma = c \vee \gamma = max^v(c, NOW + c\_off)) \wedge \\
\qquad\quad \beta,\delta \in \mathcal{T} \wedge a < \delta \wedge c < \beta) \\
[max(\alpha,\gamma), min^v(min(b,d), NOW + min(b\_off, d\_off))) \\
\qquad \text{if } ((\beta = b \vee \beta = min^v(b, NOW + b\_off)) \wedge (\delta = d \vee \delta = min^v(d, NOW + d\_off)) \wedge \\
\qquad\quad \alpha,\gamma \in \mathcal{T} \wedge \alpha < d \wedge \gamma < b \wedge \\
[max^v(max(a,c), NOW + max(a\_off, c\_off)), min^v(min(b,d), NOW + min(b\_off, d\_off))) \\
\qquad \text{if } (((\alpha = max^v(a, NOW + a\_off) \wedge \beta = min^v(b, NOW + b\_off)) \vee \\
\qquad\quad (\gamma = max^v(c, NOW + c\_off) \wedge \delta = min^v(d, NOW + d\_off))) \wedge a < d \wedge c < b \wedge \\
\qquad\quad max(a\_off, c\_off) < min(b\_off, d\_off) \\
\emptyset \qquad \text{otherwise}
\end{cases}
\end{aligned}
$$

The extensionalization graphs in Figure 11 may also serve to illustrate the extended union definition. Figure 11A illustrates the union $[5, min^v(9, NOW + 2)) \cap^v [3, 7)$, which is covered by case 3. The result, interval $[5, min^v(7, NOW + 2))$, is shown in Figure 13A. Next, Figure 11B illustrates $[max^v(2, NOW - 2), 6) \cap^v [max^v(4, NOW + 2), 8)$, which is covered by case 2 and results in $[max^v(4, NOW + 2), 6)$ as shown in Figure 13B.

The examples in Figure 11C and Figure 11D are both covered by case 4 in the definition. Figure 11C illustrates $[max^v(2, NOW), 8) \cap^v [4, min^v(6, NOW + 2))$, which results in the interval $[max^v(4, NOW), min^v(6, NOW + 2))$ (shown in Figure 13C). Thus the offsets cause the intersection operator to return an interval of the new type introduced in this section. Figure 11D computes $[max^v(3, NOW - 3), min^v(7, NOW + 3)) \cap^v [max^v(4, NOW - 1), min^v(6, NOW + 1))$, resulting in the interval $[max^v(4, NOW - 1), min^v(6, NOW + 1))$, shown in Figure 13D.

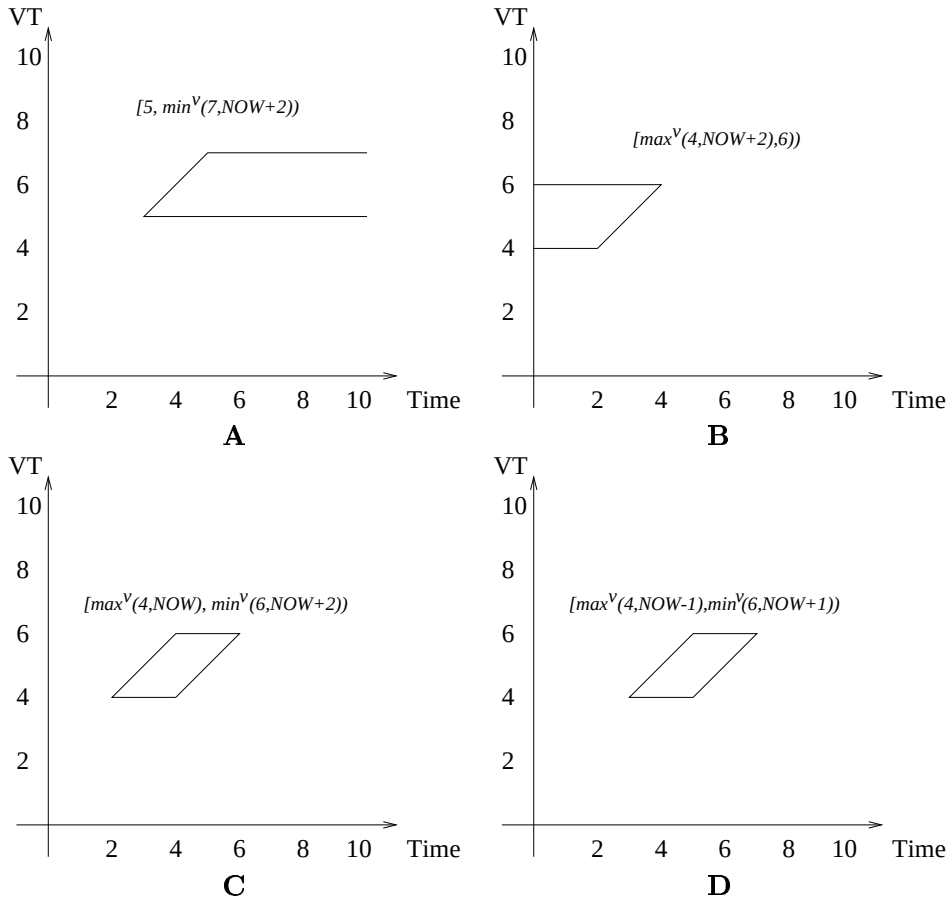

Figure 13: NOW-Relative Interval Intersection Results

5.4 Temporal Modification Semantics Including NOW-Relative Values

With the new difference and intersection operators in place, we can define the semantics of modifications involving NOW-relative values. These definitions may be given by re-using the template employed for the definitions in Section 4.7, the only difference being that the extended difference and intersection operators are to be used. For brevity, we do not repeat the definitions, but instead exemplify the utility of NOW-relative values.

In the following scenario, Joe joins the Shoe department on the 3rd. At any time, he can leave his job with eight days' notice. On the 6th, he is told that with two days notice, he can be

reassigned to the Toy department for four days. On the 8th, he gives notice that he will leave his job (i.e., his last day is the 16th).

This scenario is captured by the following database modifications.

```
-- 3rd: Joe is hired on the 3rd with 8 days notice
VALIDTIME PERIOD [3, NOBIND(CURRENT_DATE + 8))
INSERT INTO Emp VALUES ('Joe', 'Shoe');

-- 6th: Plan that Joe can temporarily be in the Toy department
        for four days
VALIDTIME PERIOD [NOBIND(CURRENT_DATE + 2), NOBIND(CURRENT_DATE + 6))
UPDATE Emp
SET Dept = 'Toy'
WHERE Name = 'Joe';

-- 8th: Joe quits his job and has 8 days' notice
VALIDTIME PERIOD [16, FOREVER)
DELETE FROM Emp
WHERE Name = 'Joe';
```

The single tuple that results from the first statement is $\langle$Joe, Shoe, [3,NOW+8)$\rangle$ and is illustrated with the solid line in Figure 14A. The result of the first update is as follows, and is illustrated in Figures 14A and B.

$$\begin{aligned}\emptyset \cup^{vt} & \{\langle \text{Joe}, \text{Shoe}, \{[3, NOW+8) -^{v} [NOW+2, NOW+6)\}\rangle\} \cup^{vt} \\ & \{\langle \text{Joe}, \text{Toy}, \{[3, NOW+8) \cap^{v} [NOW+2, NOW+6)\}\rangle\} \\ & \quad = \{\langle \text{Joe}, \text{Shoe}, [3, NOW+2)\rangle, \quad \langle \text{Joe}, \text{Shoe}, [max^{v}(3, NOW+6), NOW+8)\rangle, \\ & \qquad \langle \text{Joe}, \text{Toy}, [max^{v}(3, NOW+2), NOW+6)\rangle\}\end{aligned}$$

We plan that Joe may be temporarily in the Toy department in the interval [NOW+2,NOW+6) where the +2 indicates the two days' notice. This interval can be rewritten as [max^{v}(*beginning*, $NOW+2$),min^{v}(*forever*, $NOW+6$)) and is indicated by the two dashed lines in Figure 14A. The result is the three tuples indicated by solid lines in Figure 14B.

The result of the deletion is as follows, and is illustrated in Figures 14B and C.

$$\begin{aligned}\emptyset \cup^{vt} & \{\langle \text{Joe}, \text{Shoe}, \{[3, NOW+2)\rangle -^{v} [16, forever)\}\rangle\} \cup^{vt} \\ & \{\langle \text{Joe}, \text{Shoe}, \{[max^{v}(3, NOW+6), NOW+8) -^{v} [16, forever)\}\rangle\} \cup^{vt} \\ & \{\langle \text{Joe}, \text{Toy}, \{[max^{v}(3, NOW+2), NOW+6) -^{v} [16, forever)\}\rangle\} \\ & \quad = \{\langle \text{Joe}, \text{Shoe}, [3, min^{v}(16, NOW+2))\rangle, \\ & \qquad \langle \text{Joe}, \text{Shoe}, [max^{v}(3, NOW+6), min^{v}(16, NOW+8))\rangle, \\ & \qquad \langle \text{Joe}, \text{Toy}, [max^{v}(3, NOW+2), min^{v}(16, NOW+6))\rangle\}\end{aligned}$$

From the three tuples stored in the relation at the outset, we delete the interval [16,*forever*), to indicate that Joe is leaving the company on the 16th. The interval to be deleted is indicated by the dashed line in Figure 14B. The result of the deletion is the three tuples indicated by solid lines in Figure 14C. The tuples with end-point values above 16 from Figure 14B are "truncated" here.

Having defined the semantics for modifications involving NOW, we consider the implementation of the modifications.

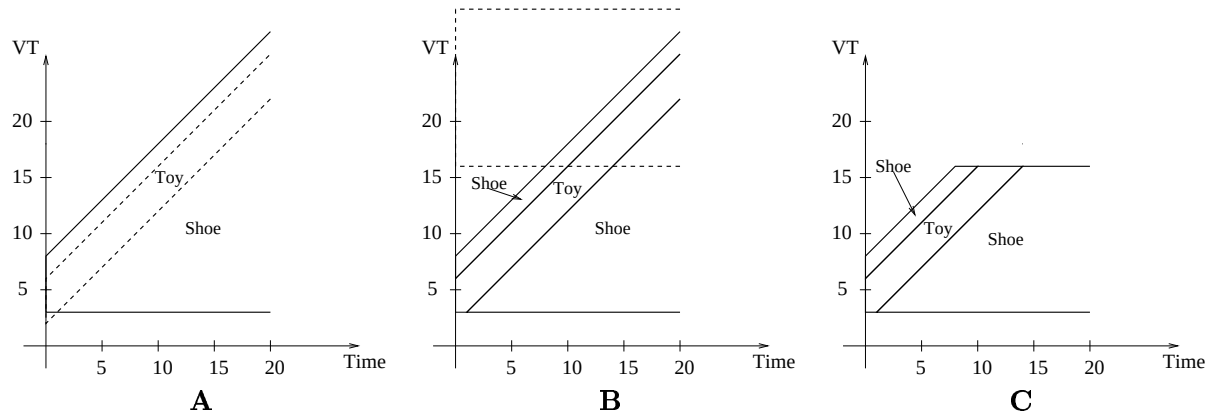

Figure 14: The Results of Updates With *NOW*-Relative

6 Implementing Modification Statements Involving *NOW*-Relative Values

This section considers the implementation of the modifications defined in the previous two sections. Implementation using only SQL-92 and using the object-relational features provided by some commercial DBMSs are considered.

6.1 Representing Canonical Intervals

The canonical intervals can be implemented in SQL-92 by using four columns. Two columns record the `V-Begin` and `V-End` attributes, and two columns, named `V-Begin-Offset` and `V-End-Offset`, indicate the valid-time begin offset and the valid-time end offset, respectively. Our sample relation may then be declared as follows.

```
CREATE TABLE Emp (
        Name            VARCHAR (20)  NOT NULL,
        Dept            VARCHAR (20)  NOT NULL,
        V-Begin         DATE,                     -- Assuming day granularity
        V-Begin-Offset  NUMERIC (10,0),
        V-End           DATE,                     -- Assuming day granularity
        V-End-Offset    NUMERIC (10,0))
```

Here, a `V-Begin-Offset` and `V-End-Offset` value different from `NULL` indicates a max^v and a min^v function, respectively. The representation of the different shapes is demonstrated below. Recall that *NOW* can be written as $max^v(beginning, NOW)$ in the `V-Begin` attribute and $min^v(forever, NOW)$ in the `V-End` attribute.

The valid-time interval can also be implemented as an abstract data type (ADT), e.g., in the Informix and Oracle DBMSs. The main advantage of this is that the valid-time interval is encapsulated and is treated as a single unit.

6.2 Implementing Queries

When querying, *NOW* values must be bound to the current time. We do so with two functions, `BIND_B` to bind the beginning time, and `BIND_E` to bind the end time.

To illustrate querying, suppose we want to retrieve all tuples with a valid time that overlaps the interval [1999-01-05,1999-01-15). This can be written in a temporal SQL [27] as follows.

```
VALIDTIME PERIOD [DATE '1999-01-05', DATE '1999-01-15')
SELECT * FROM Emp;
```

This query formulated in SQL-92 is shown below.

```
SELECT Name, Dept,
          BIND_B(V-Begin, V-Begin-Offset) AS V-Begin,
          BIND_E(V-End, V-End-Offset) AS V-End
FROM   Emp
WHERE  BIND_B(V-Begin, V-Begin-Offset) < DATE '1999-01-15' AND
          DATE '1999-01-05' < BIND_E(V-End, V-End-Offset) AND
          BIND_B(V-Begin, V-Begin-Offset) < BIND_E(V-End, V-End-Offset)
```

The query checks if the valid time associated with a tuple overlaps with the temporal scope, i.e., the interval [1999-01-05,1999-01-15). In the last line, it is checked that the `V-Begin` attribute is smaller than the `V-End` attribute, when these attributes are bound to the current date. Note that a query always returns a ground result.

As an example, the `BIND_B` function can be specified in PSM [23] as follows.

```
DECLARE FUNCTION BIND_B (Val DATE, Offset NUMERIC (10,0)) RETURNS DATE
   IF Offset IS NULL
   THEN RETURN Val
   ELSE IF Val > CURRENT_DATE + CAST(Offset AS INTERVAL DAY)
   THEN RETURN Val
   ELSE RETURN CURRENT_DATE + CAST(Offset AS INTERVAL DAY);
```

The `BIND_E` function is analogous. These can be implemented similarly in Oracle's PL/SQL or as user-defined functions in Informix or DB2.

If implemented inside the DBMS, rather than with an external translater from temporal SQL to conventional SQL, the binding functions need not be called multiple times, as in the SQL code above. Implementing the binding within the DBMS might also enable other simplifications.

6.3 Implementing Modifications

Having illustrated querying when representing temporal data using the format with four four extra attributes, we proceed to consider modification. First, insertions are easy to map to SQL: we simply set the offset columns depending on the presence of NOW.

```
VALIDTIME PERIOD [DATE '1999-01-05', DATE '1999-01-20')
INSERT (Joe, Shoe)
```

may be mapped into

```
INSERT INTO Emp VALUES
    ('Joe', 'Shoe', DATE '1999-01-05', NULL, DATE '1999-01-20', NULL);
```

When the value NOW, specified as `NOBIND(CURRENT_DATE))` [10], is present, a non-null value of the offset is used.

```
VALIDTIME PERIOD [DATE '1999-01-05', NOBIND(CURRENT_DATE))
INSERT (Joe, Shoe)
```

is mapped into

```
INSERT INTO Emp VALUES
    ('Joe', 'Shoe', DATE '1999-01-05', NULL, DATE '9999-12-31', 0);
```

Deletions and updates should conform to the semantics presented in Sections 4.7 and 5.4. The main problem when implementing these is that the extended interval difference operator that they make use of may return up to three intervals for the semantics specified in Section 4.7 and up to four intervals for the semantics specified in Section 5.4.

To solve this problem we use the idea illustrated by the extended interval difference example in Figure 11D and 12D. To determine the result of a difference, e.g., the difference $[a, b) -^v [c, d)$, we look "above," "below," "right," and "left" of interval $[c, d)$ and determine what remains of interval $[a, b)$.

As an example, consider the case where the database contains the tuple $\langle$Joe, Shoe, $[max^v(\text{1999-01-03}, NOW - 3), min^v(\text{1999-01-07}, NOW + 3))$ $\rangle$ and a temporal deletion statement causes us to delete Joe from the Shoe department in the interval $[max^v(\text{1999-01-04}, NOW - 1), min^v(\text{1999-01-06}, NOW + 1))$. This latter interval then is the interval specified in the temporal deletion statement. This is corresponds to the example in Figure 11D.

The following four SQL-92 insertions and one SQL-92 deletion cover all the cases. Note that they are similar in form, and that the last three lines of each are identical.

```
-- Above
INSERT INTO Emp
SELECT Name, Dept, '1999-01-06', V-Begin_Offset, V-End, V-End-Offset
FROM   Emp
WHERE  Name = 'Joe' AND V-End > '1999-01-06' AND
       V-Begin < DATE '1999-01-06' AND DATE '1999-01-04' < V-End AND
       (V-Begin-Offset < 1 OR (V-Begin-Offset IS NULL OR 1 IS NULL)) AND
       (-1 < V-End-Offset OR (-1 IS NULL or V-End-Offset IS NULL));
-- Below
INSERT INTO Emp
SELECT Name, Dept, V-Begin, V-Begin-Offset, DATE '1999-01-04', V-End-Offset
FROM   Emp
WHERE  Name = 'Joe' AND V-Begin < '1999-01-04' AND
       V-Begin < DATE '1999-01-06' AND DATE '1999-01-04' < V-End AND
       (V-Begin-Offset < 1 OR (V-Begin-Offset IS NULL OR 1 IS NULL)) AND
       (-1 < V-End-Offset OR (-1 IS NULL or V-End-Offset IS NULL));
-- Right
INSERT INTO Emp
SELECT Name, Dept,
       GREATEST (V-Begin, DATE '1999-01-04'), V-Begin-Offset,
       LEAST (V-End, '1999-01-06'), -1
FROM   Emp
WHERE  Name = 'Joe' AND -1 IS NOT NULL AND
       GREATEST (V-Begin, DATE '1999-01-04') < LEAST (V-End, '1999-01-06') AND
       (NOT (V-Begin-Offset IS NOT NULL AND V-Begin-Offset > -1)) AND
       V-Begin < DATE '1999-01-06' AND DATE '1999-01-04' < V-End AND
       (V-Begin-Offset < 1 OR (V-Begin-Offset IS NULL OR 1 IS NULL)) AND
       (-1 < V-End-Offset OR (-1 IS NULL or V-End-Offset IS NULL));
```

```
-- Left
INSERT INTO Emp
SELECT Name, Dept, GREATEST (V-Begin, DATE '1999-01-04'), 1,
       LEAST (V-End, '1999-01-06'), V-End-Offset
FROM   Emp
WHERE  Name = 'Joe' AND 1 IS NOT NULL
       GREATEST (V-Begin, DATE '1999-01-04') < LEAST (V-End, '1999-01-06') AND
       AND (NOT (V-End-Offset IS NOT NULL AND 1 > V-End-Offset)) AND
       V-Begin < DATE '1999-01-06' AND DATE '1999-01-04' < V-End AND
       (V-Begin-Offset < 1 OR (V-Begin-Offset IS NULL OR 1 IS NULL)) AND
       (-1 < V-End-Offset OR (-1 IS NULL or V-End-Offset IS NULL));
-- Delete the old tuple
DELETE FROM Emp
WHERE  Name = 'Joe' AND
       V-Begin < DATE '1999-01-06' AND DATE '1999-01-04' < V-End AND
       (V-Begin-Offset < 1 OR (V-Begin-Offset IS NULL OR 1 IS NULL)) AND
       (-1 < V-End-Offset OR (-1 IS NULL or V-End-Offset IS NULL));
```

The SQL-92 code above uses the Oracle-specific functions `GREATEST` and `LEAST`, which correspond the conventional *max* and *min* functions used in this paper.

The time overlap predicate (the third-to-last line) and the offset overlap predicate (the last two lines) check that the intervals associated with the tuples overlap with the interval specified in the delete statement. The offset overlap predicate checks for overlap between the offsets on the intervals in the database and the offset specified in the interval to delete or update. Because these offsets can have the value NULL we must for each less than operator, check if either of the operands are NULL. The '1' in , e.g., in `1 IS NULL` is the V-End-Offset of the interval specified in the delete. Similar, is the '-1', e.g., in `-1 < V-End-Offset`, is the V-Begin-Offset of the interval specified in the delete.

In the first insert statement the check `V-End > '1999-01-06'` ensures that an "above" tuple is generated when appropriate. Similarly, the check in the second insertion, `V-Begin < '1999-01-04'`, ensures that a "below" tuple is generated.

The "right" and "left" cases, the third and fourth insert statements, are slightly more complicated because we ensure (1) that a tuple is generated, (2) that it has no overlap with the "above" and "below" tuples, (3) that its `V-Begin` value is smaller than its `V-End` value, and (4) that its `V-Begin-Offset` value is smaller than its `V-End-Offset` value.

In the third insert statement, the first check is that `-1 IS NOT NULL`. A "right" tuple is only generated if the `V-Begin-Offset` of the interval specified in the temporal deletion statement is `NOT NULL`. The second and third checks are done with the `GREATEST` and `LEAST` functions in the select clause and in the where clause, respectively. The fourth check occurs in the last line. We must ensure if the `V-Begin-Offset` attribute is `NOT NULL` then its value cannot be larger than -1, which is the `V-Begin-Offset` of the interval specified in the temporal delete statement.

In the fourth insert statement, the first check is that `1 IS NOT NULL`. A "left" tuple is only generated if the `V-End-Offset` of the interval specified in the temporal deletion statement is `NOT NULL`. Again the second and third checks are done using `GREATEST` and `LEAST`, and the fourth check occurs in the last line. If the `V-End-Offset` is `NOT NULL` then its value cannot be larger than 1, which is the `V-End-Offset` of the interval specified in the temporal deletion statement.

After the four insertions, all tuples that overlap with the interval specified in the deletion statement are deleted. This is correct because we have just created up to four tuples that represent what remains of the ordinal interval.

Note that because the semantics specified in Sections 4.7 only allows a max^v function in the `V-Begin` attribute or a min^v function in the `V-End` attribute, the "right" and "left" tuples are

mutually exclusive. This semantics will therefore result in a maximum of three new tuples. This restriction does not apply to the semantics specified in Section 5.4; in the presence of now-relative values, as many as four tuples may result.

Updates are implemented similarly to deletes. Assume that we want to update Joe for the interval [1999-01-04,max^v(1999-01-06, NOW)) to be with the Toy department. In an extended temporal SQL, this may be written as follows [27].

```
VALIDTIME PERIOD [DATE '1999-01-04', NOBIND(CURRENT_DATE))
UPDATE Emp SET Dept = 'Toy'
WHERE Name = 'Joe';
```

This is written in SQL-92 as four insertions and an update. The four insertions are identical to those displayed for the temporal deletion above. The update statement follows.

```
-- Update self
UPDATE EmpNow
SET    Name, Dept,
       V-Begin        = GREATEST (V-Begin, DATE '1999-01-04'),
       V-End          = LEAST    (V-End,   NULL),
       V-Begin-Offset = GREATEST (V-Begin-Offset, NULL),
       V-End-Offset   = LEAST    (V-End-Offset, 0)
WHERE  Name = 'Joe' AND
       NOT (GREATEST (V-Begin, DATE '1999-01-04') IS NOT NULL AND
            LEAST (V-End, NULL) IS NOT NULL AND
            GREATEST (V-Begin, DATE '1999-01-04') >
               LEAST (V-End, NULL)) AND
       NOT (GREATEST (V-Begin-Offset, NULL) IS NOT NULL AND
            LEAST (V-End-Offset, 0) IS NOT NULL AND
            GREATEST (V-Begin-Offset, NULL) > LEAST (V-End-Offset, 0)) AND
       V-Begin < DATE '1999-01-06' AND DATE '1999-01-04' < V-End AND
       (V-Begin-Offset < 1 OR (V-Begin-Offset IS NULL OR 1 IS NULL)) AND
       (-1 < V-End-Offset OR (-1 IS NULL or V-End-Offset IS NULL));
```

In line 3, the `V-Begin` attribute is set to the maximum of the start of the interval of the tuple being updated and the start of the interval specified in the update. In line 4, the `V-End` attribute is set to the minimum of its current value and the end of the interval in the update. This is done similarly for the `V-Begin-Offset` and `V-End-Offset` attributes in lines 5 and 6, respectively. In the where clause, it is checked in lines 8 to 10 that the new interval is non-empty. Similarly, lines 11 to 13 check that the `V-Begin-Offset` is smaller than the `V-End-Offset`. Line 14 checks overlap on the `V-Begin` and `V-End` attributes, and the `V-Begin-Offset` and `V-End-Offset` attributes. The last three lines duplicate those of the above insertions.

In this example, the where clause is particularly simple, and the provided mapping works fine. For more complex predicates, for example those containing subqueries, care must be taken to ensure that these are afforded the correct semantics, following the mapping for queries [27].

A full solution includes support for ground interval end points, which is already included in SQL-92, and for *NOW* and for now-relative values, as provided in this paper. In addition, support can be provided for indeterminate versions of these values (so-called "don't know exactly when" values) [12]. Support for indeterminate values can be defined in terms of the operations on determinate values (for example, an indeterminate ground time value can be represented and manipulated as a pair of determinate ground time values). As proof of concept, the TIMEADT

prototype [28] implements all the usual predicates and constructors for all six classes of period values: ground determinate, variable determinate, now-relative determinate, ground indeterminate, variable indeterminate, and now-relative indeterminate.

Triggers and methods on a new data type for the time intervals considered constitute two alternatives to the SQL-92 statements above. Both have associated problems. For example, an implementation using triggers for the deletions and updates illustrated above is not directly implementable in Oracle8i because the trigger will result in a *mutating table* [25], which is not allowed (this condition occurs when a statement in the trigger body accesses the table that the trigger was fired on).

Defining methods on an interval ADT is also quite challenging. The main problem is having to generate up to four separate tuples when a single tuple is being manipulated. This seems to be beyond ADTs. Another problem is how to handle the where clauses of temporal modifications. The where predicate can be specified by the user at run-time, making its necessary for the insert, delete, and update methods on an interval ADT to accept a string argument containing the where clause. The modification methods must then parse the where clause and dynamically generate appropriate SQL-92 statements that mirror to the temporal modification statement.

7 Approximate Modification Semantics

As the previous section showed, representing the new time values requires either an ADT or multiple physical columns. The present section will explore what semantics may be achieved with just a single additional NOW value, which can be denoted with a particular existing value, such as `NULL` or `DATE '9999-12-31'` [31]. Specifically, we define approximate semantics for modifications involving NOW that do not require the use of the min^v and max^v functions. This will help indicate precisely what the rather complex implementation of the previous section buys us.

We first introduce a set of auxiliary functions. Then the semantics of insert, delete, and update are defined, followed by examples and a discussion of the differences between the accurate and the approximate semantics.

7.1 Possible Approximate Semantics of Temporal Modifications

Our definitions of modification lead to tuples that contain the min^v and max^v functions in their timestamps. Existing temporal data models do not accommodate these functions, leading to a violation of **Requirement R3** in Section 4.1. We thus explore alternative semantics of modifications that avoid these values.

We use the last example in Section 4.7 of an update for exploring possible alternative semantics. The result of that update was illustrated in Figure 9B and is also given in Table 3.

Name	Dept.	V-Begin	V-End
Joe	Shoe	5	$min^v(10, NOW)$
Joe	Toy	10	$min^v(20, NOW)$
Joe	Shoe	20	$min^v(forever, NOW)$

Table 3: Result of the Update

The objective is to accomplish this update without using the min^v (and max^v) value. We assume that the current time is 15. The current time, not an issue in the exact semantics, will be important in the approximate semantics. Three possible, approximate update semantics are illustrated Figure 15.

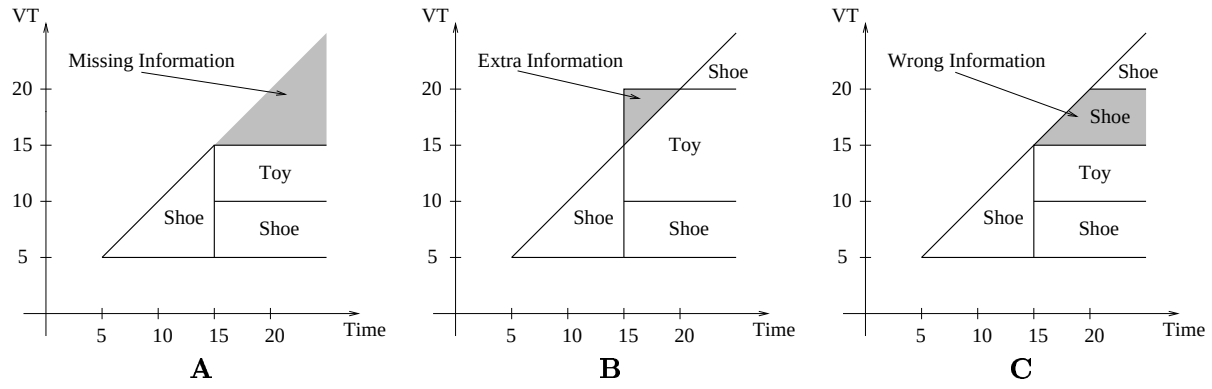

Figure 15: Different Approximate Solutions

Figure 15A adopts a pessimistic approach, which was also shown in Table 1A. (Note that the region labeled "Shoe" corresponds to a tuple that was only present in the database until time 15.) The drawback is that information is missing in the shaded region. The second approximate semantics, the result of which is shown in Figure 15B, has the drawback that the information in the shaded region in Figure 15B has been manufactured and is extraneous. This figure corresponds to Table 1B. The third approximate update semantics, shown in Figure 15C, is similar to the optimistic approach, shown in Table 1C. There is no extra or missing information; however, wrong information is created. The department "Shoe" in the shaded region should be "Toy."

Each approximate semantics has drawbacks, but the second semantics appears to be preferable. The extraneous information is present in the second tuple of Table 1B: ⟨Joe, Toy, 10, 20⟩. If we consult this table on January 16 and ask, "what department is (actually, will) Joe work in on January 18?", the result will be: Toy. The result is slightly misleading, as Joe may be fired. On January 16, all we should know is that Joe started working in the Toy department on January 10, and is still working there. (Table 3 would return no department for this query, because we do not *know* whether he will be working on January 12.) If the same question, "what department is Joe in on January 18?", is asked on January 20, both Table 1B and Table 3 will reply, "Toy," because we now know Joe was not fired.

In summary, the approximate semantics sometimes represents as the case information that will in fact be true if nothing changes. In defense of this semantics, the current situation, in which data models, such as that underlying SQL, do not support storing NOW, extra information of a similar flavor is routinely represented.

We provide an approximate semantics for modifications consistent with the second approach that avoid the two new types of time values and that thus may be stored using the formats proposed by existing temporal data models. But first we introduce some useful functions.

7.2 Auxiliary Functions

We define the functions min^e and max^e as follows, where a and b are in $\mathcal{T}$. Superscript "e" denotes "eventual," and the intuition behind the definitions is to make the difference and intersection operators produce the results that they would eventually produce, if one waited to apply them long enough.

$$min^e(a,b) \triangleq \begin{cases} a & \text{if } a \in \mathcal{T} \wedge b = NOW \\ b & \text{if } a = NOW \wedge b \in \mathcal{T} \\ NOW & \text{if } a = NOW \wedge b = NOW \\ min(a,b) & \text{otherwise} \end{cases}$$

$$max^e(a,b) \stackrel{\triangle}{=} \begin{cases} a & \text{if } a \in \mathcal{T} \wedge b = NOW \\ b & \text{if } a = NOW \wedge b \in \mathcal{T} \\ NOW & \text{if } a = NOW \wedge b = NOW \\ max(a,b) & \text{otherwise} \end{cases}$$

If exactly one of the arguments is NOW, the other argument is returned; and if both arguments are NOW, NOW is returned. The functions reduce to the conventional counterparts if both a and b are in $\mathcal{T}$.

We define the maximum time of a modification statement, e.g., `[`vts`,`$vte`$`) DELETE FROM` $r_{vt}$ `WHERE` *cond* as follows, where ct is the current time.

$$max\_time([vts, vte), ct) \stackrel{\triangle}{=} max(vts, [\![vte]\!]_{ct})$$

Function *max_time* will be used in the definitions of deletion and update in Section 7.3.

We define the eventual difference, $-^e$, of intervals as follows, where a and c are in $\mathcal{T}$ and b and d are in $\mathcal{T}_1$ as follows.

$$[a,b) \ -^e \ [c,d) \stackrel{\triangle}{=} \begin{cases} [a, min^e(b,c)), [max^e(a,d), b) & \text{if } a < [\![d]\!]_{forever} \wedge c < [\![b]\!]_{forever} \\ [a,b) & \text{otherwise} \end{cases}$$

The eventual difference is identical to the conventional difference operator, except that the value of NOW is bound to *forever*.

We define the eventual intersection ($\cap^e$) of two intervals, where a and c again are in $\mathcal{T}$ and b and d are in $\mathcal{T}$, as follows.

$$[a,b) \ \cap^e \ [c,d) \stackrel{\triangle}{=} \begin{cases} [max^e(a,c), min^e(b,d)) & \text{if } a < [\![d]\!]_{forever} \ \wedge \ c < [\![b]\!]_{forever} \\ \emptyset & \text{otherwise} \end{cases}$$

The eventual intersection is identical to the conventional intersection, except that the value of NOW is bound to the maximum value in $\mathcal{T}$. Two tuples that do not currently overlap may still have an eventual intersection. For example, the intervals $[10, NOW)$ and $[20, 30)$ do not overlap at time 15, but if they do not change, they will eventually overlap, and their eventual intersection is $[20, 30)$.

7.3 Approximate Modification Semantics

We can now define the approximated semantics of modifications involving NOW.

Insertion into a valid-time relation r_{vt} is defined as follows.

`VALIDTIME PERIOD [`vts`,`vte`) INSERT INTO` r_{vt} `VALUES (`A`)` $\stackrel{\triangle}{=}$

$$r_{vt} \leftarrow r_{vt} \cup^{vt} \{(A, [vts, vte))\}$$

A tuple is added to the relation. We associate with the tuple the valid-time interval [*vts*,*vte*) specified in the insertion statement. This semantics is identical to the accurate semantics.

Deletion from a valid-time relation r_{vt} is defined as follows, where $mt = max\_time([vts, vte), ct)$ is the maximum time of the delete statement, and $\mathrm{VT}^{\vdash}$ and $\mathrm{VT}^{\dashv}$ denote the start and end points of valid-time interval VT.

`VALIDTIME PERIOD [`vts`,`vte`) DELETE FROM` r_{vt} `WHERE` *cond* $\stackrel{\triangle}{=}$

$$\begin{aligned} r_{vt} \ \leftarrow \ & \{t | t \in r_{vt}(\neg cond(t) \vee (\mathrm{VT}^{\vdash} \geq [\![vte]\!]_{mt} \vee vts \geq [\![\mathrm{VT}^{\dashv}]\!]_{mt}))\} \cup^{vt} \\ & \{t | \exists s \in r_{vt}(cond(s) \wedge t[A] = s[A] \wedge \mathrm{VT}^{\vdash} < [\![vte]\!]_{mt} \wedge vts < [\![\mathrm{VT}^{\dashv}]\!]_{mt} \wedge \\ & \qquad t[\mathrm{VT}] \in (s[\mathrm{VT}] -^e [vts, vte)) \wedge t[\mathrm{VT}] \neq \emptyset)\} \end{aligned}$$

In the first line, tuples that do not fulfill the condition *cond* or do not overlap with the interval specified in the delete, at the maximum time, are retained unchanged. In the second and third lines, tuples that both fulfill the condition and overlap at the maximum time have their valid-time intervals reduced by the parts that overlap the interval specified in the delete statement.

Update of a valid-time relation r_{vt} is defined as follows, where $A = v$ as usual is short for $A_1 = v_1, \ldots, A_n = v_n$ and is the explicit attributes, which are assigned new values. (For brevity, we assume that all explicit attributes change values.)

`VALIDTIME PERIOD [`vts`,`$vte`$`) UPDATE` $r_{vt}$ `SET` $A$ `=` $v$ `WHERE` $cond \triangleq$

$$
\begin{aligned}
r_{vt} \leftarrow\ & \{t | t \in r_{vt}(\neg cond(t) \vee (\mathrm{VT}^{\vdash} \geq [\![vte]\!]_{mt} \vee vts \geq [\![\mathrm{VT}^{\dashv}]\!]_{mt}))\} \cup^{vt} \\
& \{t | \exists s \in r_{vt}(cond(s) \wedge t[A] = s[A] \wedge \mathrm{VT}^{\vdash} < [\![vte]\!]_{mt} \wedge vts < [\![\mathrm{VT}^{\dashv}]\!]_{mt} \wedge \\
& \qquad t[\mathrm{VT}] \in (s[\mathrm{VT}] -^{e} [vts, vte)) \wedge t[\mathrm{VT}] \neq \emptyset)\} \cup^{vt} \\
& \{t | \exists s \in r_{vt}(cond(s) \wedge t[A] = v \wedge \mathrm{VT}^{\vdash} < [\![vte]\!]_{mt} \wedge vts < [\![\mathrm{VT}^{\dashv}]\!]_{mt} \wedge \\
& \qquad t[\mathrm{VT}] = (s[\mathrm{VT}] \cap^{e} [vts, vte) \wedge t[\mathrm{VT}] \neq \emptyset)\}
\end{aligned}
$$

The first three lines are identical to the three lines for the delete statement. In lines four and five, tuples with the new explicit attribute values are inserted. The intervals associated with the new tuples are the eventual intersections of the intervals currently associated with each tuple and the interval specified in the update.

7.4 Examples of the Approximate Semantics

From the auxiliary functions defined in Section 7.2, it follows that the approximate semantics for modifications involving *NOW* reduce to the conventional semantics when only fixed intervals are considered. For this reason, we only show modification examples involving *NOW*.

The first example is an update of a tuple containing *NOW*. We use the update in Figure 1 and assume the database contains the tuple $\langle \text{Joe}, \text{Shoe}, [5, NOW) \rangle$. The update occurs at time 15.

```
VALIDTIME PERIOD [10,20) UPDATE Emp SET Dept = 'Toy' WHERE Name = 'Joe'
```

The result of the update is as follows.

$$
\begin{aligned}
\emptyset \cup^{vt} \{\langle \text{Joe}, \text{Shoe}, \{[5, NOW) -^{e} [10, 20)\}\rangle\} \cup^{vt} \{\langle \text{Joe}, \text{Toy}, \{[5, NOW) \cap^{e} [10, 20)\}\rangle\} \\
= \{\langle \text{Joe}, \text{Shoe}, [5, 10)\rangle, \langle \text{Joe}, \text{Shoe}, [20, NOW)\rangle, \langle \text{Joe}, \text{Toy}, [10, 20)\rangle\}
\end{aligned}
$$

The maximum time of the update is 20. All tuples in the relation are affected by the update, which explains the initial empty set. The two next terms reduce the interval associated with the existing tuple and update Joe to be with the new department. The result is that shown in Table 1B.

The next example is a deletion on a relation with the tuple $\langle \text{Joe}, \text{Shoe}, [15, NOW) \rangle$. The deletion occurs at time 20.

```
VALIDTIME PERIOD [10,NOW) DELETE FROM Emp WHERE Name = 'Joe'
```

The maximum time of the delete statement is 20. Because the interval specified in the delete statement totally overlaps the tuple in the relation at the maximum time, the result is the empty relation. Note that if the delete statement had been executed at the time 12, the maximum time of the delete statement would then be 12. At time 12, the valid-time interval associated with the tuple and the interval specified in the delete statement do not overlap. The delete statement would then not have affected the tuple.

7.5 Comparison of the Accurate and Approximate Semantics

This section describes the main differences between the accurate semantics proposed in Section 4 and the approximate semantics just proposed.

For the approximate semantics, we are looking at the database as of the maximum time because if the user is making changes to future data, the user is looking at the current database content as of a time into the future. This leads to the following differences.

- For the approximate semantics, only two types of intervals are stored in the database, namely constant intervals [a,b) and increasing intervals [a,NOW), whereas three interval types using the max^v and min^v functions are needed to store the results of modifications that follow the accurate semantics.
- The approximate semantics simplifies the extensionalization of tables referenced in queries.
- The approximate semantics is more easily implementable in an existing temporal data model or in a layer on top of an existing relational DBMS, e.g., using a substitute value for NOW [30, 31].
- The effect of a delete or an update depends on when it is executed in the approximated semantics, whereas modifications are time-independent in the accurate semantics.

8 Related Work

Most prominently, this paper proposes definitions of modifications involving NOW and explores how the resulting semantics can be accommodated in the database. To the best of our knowledge, the semantics of modifications involving NOW have not been defined previously. Only modifications involving fixed time intervals have been defined and implemented.

In the perhaps most closely related paper [10], the semantics of NOW is described in substantial detail. It proposes a formal framework for the meaning of databases with variables in general, and it explores the querying of variable databases, but does not consider modification. To be consistent with that paper's approach, we borrow its notion of extensionalization of time values and extensionalization diagrams. We extend that paper by defining the semantics of modifications involving NOW.

The approach of timestamping tuples with intervals, as adopted in this paper, generalizes the timestamping tuples with single time point values, e.g., as done in time series. With using an interval representation, we can capture constant, increasing, and decreasing intervals. Had single time points been used instead, it would only be possible to capture either increasing or decreasing intervals, by assuming that the recorded time is the start (or stop) time and assuming the (implicit) stop (or start) time to be NOW.

Lorentzos and Manolopoulos [21] extend SQL-92 to handle general interval data, e.g., intervals in space or time. The semantics of modifications involving intervals is defined, and details are provided for how to retain relations coalesced. However, the use of variables such as NOW is not considered. The modification semantics therefore cover only the special case when intervals have their end points in domain $\mathcal{T}$.

Finger and McBrien [14, 15] discuss the semantics of NOW in connection with transactions, exploring which value to use for NOW when performing updates in a transaction. They showed that if a value for NOW is not chosen carefully, the correctness of transactions can be violated. However, the issues of which value to choose for NOW in a transaction are orthogonal to the issues discussed in this paper; the semantics of modifications involving NOW are independent of the values used for NOW.

In a previous paper [31], we considered the timestamping of modifications involving NOW in detail, showing that the commit time of a transaction has to be used as the value assigned to NOW when a modification statement in the transaction leads to a modification of the database. Again, the issues of which values to use for NOW are orthogonal to the issues discussed in this paper.

9 Summary and Research Directions

The paper's main contribution is to explore and formally define the semantics of modifications in relational databases, where NOW and $NOW + \Delta$ may be stored in timestamp columns in the database. In addition, the paper considers the implementation of such modifications.

The definitions of modifications—insertion, deletion, and update—proceed in three steps. First, the semantics of modifications on ground databases, not containing variable NOW are defined. Then the semantics of modifications in the presence of NOW are defined based on these semantics. Finally, these semantics are extended to cover also now-relative time values of the form $NOW + \Delta$.

These semantics involve extending the conventional minimum and maximum functions, as well as the interval intersection and difference operators. It is shown that the databases that result from modifications involving NOW can be represented by using three types of intervals with three types of values: normal intervals with fixed end points, a new kind of intervals that increase with time, and another new kind of intervals that decrease as time passes. These intervals involve two new kinds of time values. By including a fourth kind of interval, now-relative values are accommodated.

The paper also proposes an approximate semantics for modifications involving NOW that is easily implementable in existing temporal data models or on top of a relational DBMS using existing data types for time.

By defining modifications, the paper consistently extends past work [10] and completes the understanding of the semantics, the querying, and the modifications of NOW-relative databases.

It is challenging to index intervals containing the two new types of values used in the definition of the accurate semantics. Existing index structures generally support only fixed values; how to index these new values that change as time progresses remains an open issue, though some work has been done (e.g., [4]).

Acknowledgments

This research was supported in part by the Danish Technical Research Council through grant 9700780, by the CHOROCHRONOS project, funded by the European Commission DG XII, contract no. FMRX-CT96-0056, by a grant from the Nykredit corporation, and by grants IRI-9632569 and IIS-9817798 from the U.S. National Science Foundation.

References

[1] I. Ahn and R. T. Snodgrass. Partitioned Storage for Temporal Databases. *Information Systems*, 13(4):369–391, 1988.

[2] J. Bair, M. Böhlen, C. S. Jensen, and R. T. Snodgrass. Notions of Upward Compatibility of Temporal Query Languages. *Wirtschafts Informatik*, 39(1):25–34, 1997.

[3] J. Ben-Zvi. *The Time Relational Model.* Ph.D. thesis, Computer Science Department, UCLA, 1982.

[4] R. Bliujute, C. S. Jensen, S. Saltenis, and G. Slivinskas. R-Tree Based Indexing of Now-Relative Bitemporal Data. In *Proceedings of the VLDB Conference*, pp. 345–356, 1998.

[5] M. H. Böhlen, R. T. Snodgrass, and M. D. Soo. Coalescing in Temporal Databases. In *Proceedings of the VLDB Conference*, pp. 180–191, 1996.

[6] M. H. Böhlen and C. S. Jensen. Seamless Integration of Time into SQL. Technical Report R-96–2049, Aalborg University, Denmark, 1996.

[7] J. Clifford and A. U. Tansel. On an Algebra for Historical Relational Databases: Two Views. In *Proceedings of the ACM SIGMOD Conference*, pp. 247–265, 1985.

[8] J. Clifford and A. Croker. The Historical Relational Data Model (HRDM) and Algebra Based on Lifespans. In *Proceedings of ICDE*, pp. 528–537, 1987.

[9] J. Clifford and T. Isakowitz. On The Semantics of (Bi)Temporal Variable Databases. In *Proceedings of the Fourth International Conference on Extending Database Technology*, pp. 215–230, 1994.

[10] J. Clifford, C. E. Dyreson, T. Isakowitz, C. S. Jensen, and R. T. Snodgrass. On the Semantics of "NOW" in Databases. *ACM TODS*, 22(2):171–214, 1997.

[11] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introductions to Algorithms.* MIT Press 1990.

[12] C. E. Dyreson and R. T. Snodgrass. Supporting Valid-time Indeterminacy. *ACM TODS*, 23(1):1–57, 1998.

[13] O. Etzion, S. Jajodia, and S. Sripada (eds.). *Temporal Databases: Research and Practice.* LNCS 1399, Springer 1997.

[14] M. Finger and P. McBrien. On the Semantics of 'Current-Time' in Temporal Databases. In *Proceedings of the 11th Brazilian Symposium on Databases*, pp. 324–337, 1996.

[15] M. Finger and P. McBrien. Concurrency Control for Perceivedly Instantaneous Transactions in Valid-Time Databases. In *Proceedings of the Fourth International Workshop on Temporal Representation and Reasoning*, 1997.

[16] S. K. Gadia and S. Nair. *Temporal Databases: A Prelude to Parametric Data.* [29, Ch. 2, pp. 28–66].

[17] S. K. Gadia. A Homogeneous Relational Model and Query Languages or Temporal Databases. *ACM TODS*, 13(4):418–448, 1988.

[18] C. S. Jensen and R. T. Snodgrass. *Temporal Specialization and Generalization. IEEE TKDE*, 6(6):954–974, 1994.

[19] C. S. Jensen and C. E. Dyreson (eds.). A Consensus Glossary of Temporal Database Concepts—February 1998 Version. [13, pp. 367–405].

[20] N. Lorentzos and R. Johnson. Extending Relational Algebra to Manipulate Temporal Data. *Information Systems*, 15(3):289–296, 1988.

[21] N. Lorentzos and Y. Manolopoulos. SQL Extension for Interval Data. *IEEE TKDE*, 9(3):480–499, 1997.

[22] J. Melton and A. R. Simon. *Understanding the New SQL: A Complete Guide.* Morgan Kaufmann Publishers 1993.

[23] J. Melton. *Understanding SQL's Stored Procedures: A Complete Guide to SQL/PSM.* Morgan Kaufmann Publishers 1998.

[24] S. Navathe and R. Ahmed. *Temporal Extensions to the Relational Model and SQL.* [29, Ch. 4, pp. 92–109].

[25] Oracle. *Oracle8i Application Developer's Guide* Oracle Corporation, 1998.

[26] R. T. Snodgrass. The Temporal Query Language TQuel. *ACM TODS*, 12(2):247–298, 1987.

[27] R. T. Snodgrass, M. H. Böhlen, C. S. Jensen and A. Steiner. *Adding Valid Time to SQL/Temporal.* ANSI X3H2-96-501r2, ISO/IEC JTC 1/SC 21/WG 3 DBL-MAD-146r2, November 1996.

[28] R. T. Snodgrass, C. E. Dyreson, C. S. Jensen, N. Kline, J. Li, W. Li, M. D. Soo, L. So, and J. Whelan. The TIMEADT System, Release 1, in progress.

[29] A. Tansel, J. Clifford, S. Gadia, S. Jajodia, A. Segev, and R. T. Snodgrass, editors. *Temporal Databases: Theory, Design, and Implementation.* Benjamin/Cummings 1993.

[30] K. Torp, C. S. Jensen, and M. H. Böhlen. *Layered Implementation of Temporal DBMSs—Concepts and Techniques.* In *Proceedings of the DASFAA Conference*, pp. 371–380, 1997.

[31] K. Torp, C. S. Jensen, and R. T. Snodgrass. *Efficient Timestamping in Databases. VLDB Journal*, to appear.

[32] G. Wiederhold, S. Jajodia, and W. Litwin. *Integrating Temporal Data in a Heterogeneous Environment.* [29, Ch. 22, pp. 563–579].