Modification Semantics in Now-Relative Databases

Kristian Torp* Christian S. Jensen Richard T. Snodgrass

February 25, 2004

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 columns, 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 of 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. Finally, implementation is explored. We show how to represent the variable NOW with columns of standard SQL data types and give a mapping from SQL on now-relative data to standard SQL on these columns. The paper thereby completes the semantics, the querying, and the modification of now-relative databases.

Keywords: Temporal data, temporal query language, SQL, now, now-relative information, updates.

1 Introduction

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

*Kristian Torp, Department of Computer Science, Aalborg University, Fredrik Bajers Vej 7E, DK-9220 Aalborg Øst, Denmark, torp@cs.auc.dk; Christian S. Jensen, Department of Computer Science, Aalborg University, Fredrik Bajers Vej 7E, DK-9220 Aalborg Øst, Denmark, csj@cs.auc.dk; Richard T. Snodgrass, Department of Computer Science, University of Arizona, Tucson, AZ 85721, rts@cs.arizona.edu.

While SQL-92 [18] includes the datetime value functions `CURRENT_DATE`, `CURRENT_TIME`, and `CURRENT_TIMESTAMP`, these functions cannot be stored directly as values of columns in tables. 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 the null value, the maximum value of the time domain, or another designated value for the value of the end-time column [25].

Noting the deficiencies of these solutions, previous papers have introduce a variable NOW as a value of a column that may be stored in the database. The semantics of databases including this variable have been examined in some detail [9, 11, 13, 20]. While these papers have considered NOW in queries, they provide few details on the modification of these so-termed *variable databases*.

In the present paper, we complete the picture by defining the semantics of modifications of variable databases containing NOW and $NOW + \Delta$ and we provide means of supporting this semantics. In addition, we show how modifications under this semantics may be implemented within a DBMS and within a user application. The implementation in a DBMS encapsulates the relatively complex modification semantics, thus moving the complexity to the DBMS and thereby making it transparent to the user. This makes the modification semantics practical to use. An approximate semantics that is simpler to implement, but which can produce extra information, can be found in [28].

The presentation is organized as follows. We first give an example to indicate the subtleties and pitfalls inherent in modifications on databases containing the variable NOW and to motivate 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 permit 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. Related work is covered in Section 7; Section 8 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, at least at this stage.

We focus on the valid-time aspect of the tuples, i.e., on *when* the information recorded by tuples is *true in the miniworld* [14]. 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. Hence, the variable NOW as a transaction time value presents none of the difficulties addressed in this paper. The semantic aspects examined here concern only valid time.

When modifying tuples timestamped with periods not including NOW, the period affected by the modification is the intersection of the period associated with the tuple and the period specified in the modification [1]. We term the former the "validity period" (that is, the period during which the tuple was valid) and the latter the "applicable period" (that is, the period to which the modification applies).

To exemplify, in Figure 1 we have stored the tuple ⟨Joe, Shoe, [5,15)⟩, which denotes the fact that Joe worked in the Shoe department from (the validity period of) January 5 through the 14^{th}. We want to execute the following update: "all persons working in the Shoe department should move to the Toy department during the (applicable) period from January 10 until January 20," perhaps to handle a special sale. The result is that Joe will be with the Toy department in the period [10,15). (For simplicity, we assume that all dates are in January 2003 and we utilize closed-open periods [25].)

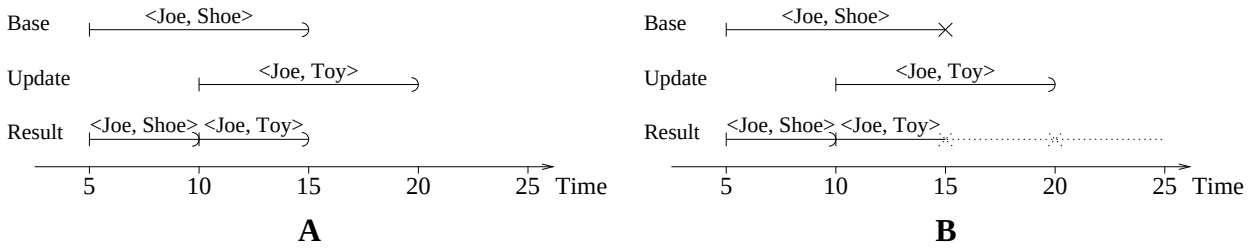


Figure 1: (A) Updating a Fact Without NOW; (B) Updating a Fact With NOW

When allowing periods to include the variable NOW (expressed as `NOBIND(CURRENT_DATE)` in [23]), 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 period [5,NOW). (We denote NOW with '$\times$' in these timeline figures.) What this means informally is that Joe started in the Shoe department on January 5 and continues there indefinitely.

We have indicated an update statement that, when executed on the 15^{th} of January, updates Joe to be with the Toy department in the applicable period [10,20). We want to determine the outcome of this update. Before the 5^{th} of January, Joe is not in the database. Between January 5 and January 10, Joe was with the Shoe department, and this period is not affected by the update, so Joe remains there. Prior to the update, Joe was also with the Shoe department from January 10 to 15. For this period the department value should be updated to Toy. This all makes perfect sense.

The semantics of the update becomes unclear for the period [15,20), when it was originally expected that Joe would be in the Shoe department. It is also unclear what happens after the 20^{th} of January, again when Joe was originally expected to remain in the Shoe department. These two periods are indicated by the dashed lines in Figure 1B.

If we use a pessimistic semantics, Joe could be fired tomorrow, and so we cannot update Joe for the latter period. Further, with the pessimistic approach Joe is not associated with the Shoe department after the 20^{th} of January either. We can instead 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 applicable period [10,20), and associate Joe with the Shoe department again after the 20^{th} 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 period [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 20^{th} of January. Finally, in Table 1C, we assume that NOW is the current date, i.e., the 15^{th} of January and that he should not be in the Toy department after today, even though the update requested a transfer through January 20.

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

None of these approaches are satisfying, for they all require assumptions that may not (probably do not) accord with the future reality. What is needed is a solution that captures in the database in a concise form *all* the information available to us, and does not necessitate mitigating assumptions. This paper provides such a solution.

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 extensions needed to define modifications of databases with the variable NOW, termed *variable databases* [9]. Later we compare the semantics of modifications on ground and variable databases. Most existing temporal data models only support periods without NOW in both queries and modifications, see, e.g., [7, 22]. Note that the semantics defined in the paper are independent of whether the underlying relations are coalesced [6] or not.

3.1 Preliminaries

We first define the union of valid-time relations and the difference and intersection of periods; these operators are used in the definitions of modifications.

We utilize the conventional relational model, but partition the columns into so-called explicit columns and two datetime columns, V-Begin and V-End, denoting a period 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 columns and $\text{VT} = [\text{V-Begin}, \text{V-End})$ record the valid time. The union operator ($\cup^{vt}$) for valid-time relations is defined as follows.

$$r_{vt} \cup^{vt} s_{vt} \triangleq \{t | t \in r_{vt} \vee t \in s_{vt}\}$$

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 column just carried along.

We assume a time domain $\mathcal{T}$ that is isomorphic to a finite subset of the natural numbers, with the classical total order $<$. We denote the minimum and maximum values of the time domain *beginning* and *forever*, respectively. The meaning of a closed-open period is defined as follows, where a and b are in $\mathcal{T}$; the period denotes a set of contiguous chronons [14].

$$[a, b) \triangleq \begin{cases} \{c | a \leq c < b\} & \text{if } a < b \\ \emptyset & \text{otherwise} \end{cases}$$

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

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

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

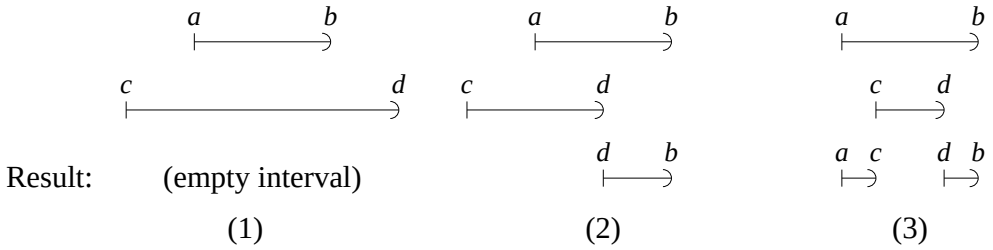


Figure 2: Example Periods Returned by the Difference Operator

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

$$[a, b) \cap [c, d) \triangleq [max(a, c), min(b, d))$$

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, period intersection is convenient in later definitions. Second, the union of periods ($\cup$) can also be defined in terms of the min and max functions on the end points, but the union of periods is not needed in this paper.

3.2 Semantics of Temporal Modifications on Ground Databases

We define insertion, deletion, and update, still on ground valid-time relations, each in turn. Insertion into a valid-time relation r_{vt} is defined as follows, where A is an abbreviation for $A_1, \ldots, A_n$ and [vts,vte) is the valid-time period to be associated with the inserted tuples. To be concrete, we use syntax that derives from a specific approach to building a temporal SQL [7, 24]. However, this syntax may be replaced by syntax from other temporal language approaches without affecting the paper's proposal.

$\texttt{VALIDTIME PERIOD [}vts\texttt{,}vte\texttt{) INSERT INTO }r_{vt}\texttt{ VALUES (}A\texttt{)} \triangleq$

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

A tuple is added to the relation. We associate with the tuple the valid-time period [vts,vte) specified in the insert statement. If such a period is not specified, a period of now to forever is used, so that the semantics is consistent with its nontemporal analogue (termed *temporal upward compatibility* [3]). We do not consider maintaining integrity constraints, such as (conventional or temporal) keys, uniqueness, or referential integrity constraints [25], as those aspects are not related to the main topic of this paper.

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

$\texttt{VALIDTIME PERIOD [}vts\texttt{,}vte\texttt{) DELETE FROM }r_{vt}\texttt{ WHERE }cond \triangleq$

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

The first line ensures that tuples in r_{vt} not satisfying the where condition $cond$ are included in the result. In the second line, tuples satisfying the condition have their time period reduced by the part that overlaps the applicable period specified in the delete statement. This period must be non-empty. Note, if a tuple satisfies the condition and does not overlap the applicable period, the tuple is included in the result unchanged.

In the definition of updates that follows, we assume for brevity that all explicit columns change values (A = v abbreviates A_1 = v_1, ..., A_n = v_n). This simplification does not restrict the generality of the results of this paper.

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

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

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

We now give some examples of the temporal modification statements on ground databases defined above. Assume the database contains the tuple $\langle$Joe, Shoe, $[5, 20)\rangle$ and that we want to delete Joe in the applicable period $[10, 15)$. This can be written as follows.

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

The result of this deletion statement is as follows.

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

From the single tuple stored in the relation, we remove Joe in the applicable period $[10, 15)$, 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 applicable period $[10, 30)$. This can be written as follows.

```
VALIDTIME PERIOD [10,30)
    UPDATE Emp SET Dept = 'Toy' WHERE Name = 'Joe'
```

The result of the update is as follows.

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

From the single tuple stored in the relation, we remove Joe in the applicable period $[10, 30)$. 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 periods $[5, 20)$ and $[10, 30)$, so that Joe is with the Shoe department in the period $[10, 20)$.

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 enables the database to evolve purely through the passage of time, without the need to explicitly modify the database.

SQL includes three nullary functions to access the current date and time: `CURRENT_DATE`, `CURRENT_TIME` and `CURRENT_TIMESTAMP`. These can be used in, e.g., the where clause or in the valid-time clause.

```
VALIDTIME CURRENT_DATE SELECT Dept FROM Emp WHERE Name = 'Joe'
```

If today was January 15, 2003, the above statement is equivalent to

```
VALIDTIME DATE '2003-01-15' SELECT Dept FROM Emp WHERE Name = 'Joe'
```

Storing the variable NOW, rather than its value, results in a variable database. In this section, we formally define the semantics of modifications of variable databases that result from introducing NOW. We first list the properties of the semantics of modifications in the presence of NOW that we require. This is followed by an example that illustrates the desired semantics. Finally, the semantics of modifications involving NOW are defined, with examples.

4.1 The Use of NOW

The use of NOW as a period 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 period-end time, which, using our example database, results in the database indicating that Joe is with the Shoe department for almost 7000 years (assuming the standard `DATE` type, with a maximum value of 9999-12-31).

In order to permit 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 [9], where it is shown that variable databases avoid many of the anomalies that arise when a specific value is required for the begin and end times. 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.

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

Requirement R1 (*Expressive*) The conventional insert, delete, and update statements should be extended to permit constant periods, i.e., $[a, b)$, as well as now-relative periods, e.g., $[a, NOW)$ for storing facts with an indefinite end time.

For example, the last statement in Section 3.2 used the ground period $[10, 30)$; it should be possible to use a now-relative period in its place. As an example consider the following update.

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

This says that Joe is to work in the Toy department until further notice. Had we not requested that NOW remain variable via the syntax `NOBIND( )`, then `CURRENT_DATE` would have been evaluated before the update was performed [9]. This requirement allows the user to decide whether the update should be only "through now" or "until further notice".

Requirement R2 (*Natural*) The semantics of modifications on variable databases should be a natural extension of the existing, non-variable semantics. Specifically, 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 (*Representable*) 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 multiple timestamp columns using existing SQL data types.

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

- The time domain from which the period end-point values are drawn must be extended to include NOW and other values, as we shall see.
- The conventional difference and intersect operators over periods that are used in the definition of the modification semantics must be extended to permit 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 beind the semantics of modifications involving NOW, we show the desired results of sample updates with and without NOW. We start with the following update in Figure 1: "Joe moved to the Toy department from January 10 to January 20" and illustrate this update on a ground tuple and on a variable tuple using *extensionalization diagrams* [9]. These diagrams are very useful for illustrating periods containing NOW. The x-axis denotes reference time, the time when a period is observed. The y-axis denotes valid time. The regions in these diagrams then convey the (possibly) time-varying meanings, or extensionalizations, of periods in tuples stored in the database.

For periods without NOW, extensionalization diagrams convert the illustration of a period from a line, as shown in Figure 1, to a rectangle, as shown in Figure 3A. Figures 3A and 3B illustrate an update not involving NOW. The region bounded by the solid line represents the tuple stored in the database (the validity period), and the region bounded by the dashed line represents the modification (the applicable period). The solid rectangle in Figure 3A indicates that Joe is with the Shoe department in the period [5,15). This information was stored at time 5 and extends to the right, denoting that the interpretation of such a period is non-time-varying: it is always [5,15). The dashed rectangle indicates that at time 15, we update Joe to be with the Toy department in the applicable period [10,20). The result is shown in Figure 3B, which shows that Joe is now with the Shoe department in the period [5,10) and with the Toy department in the period [10,15). We cannot update Joe for the period [15,20) because there is no information to update in this period. The update only affects the overlap of the two rectangles. It is a fundamental assumption in our semantics that an update affects only the overlap of two periods (validity and applicable). There must be information to update when an update statement is executed, so an update should not insert information into the database for valid times at which the information to be updated is not valid.

In Figure 3C, we show an update involving NOW. The database contains the tuple $\langle$Joe, Shoe, [5,NOW)$\rangle$, 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 period is [5,6), at time 7, the period is [5,7), and so on. As time progresses, the employment period for Joe in the Shoe department lengthens.

Again, we update Joe to be with the Toy department in the applicable period [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). If we query the database denoted by Figure 3D on January 25, this database will respond that Joe was in the Shoe department from January 5 to 10, when he moved to the Toy department for 10 days, returning to

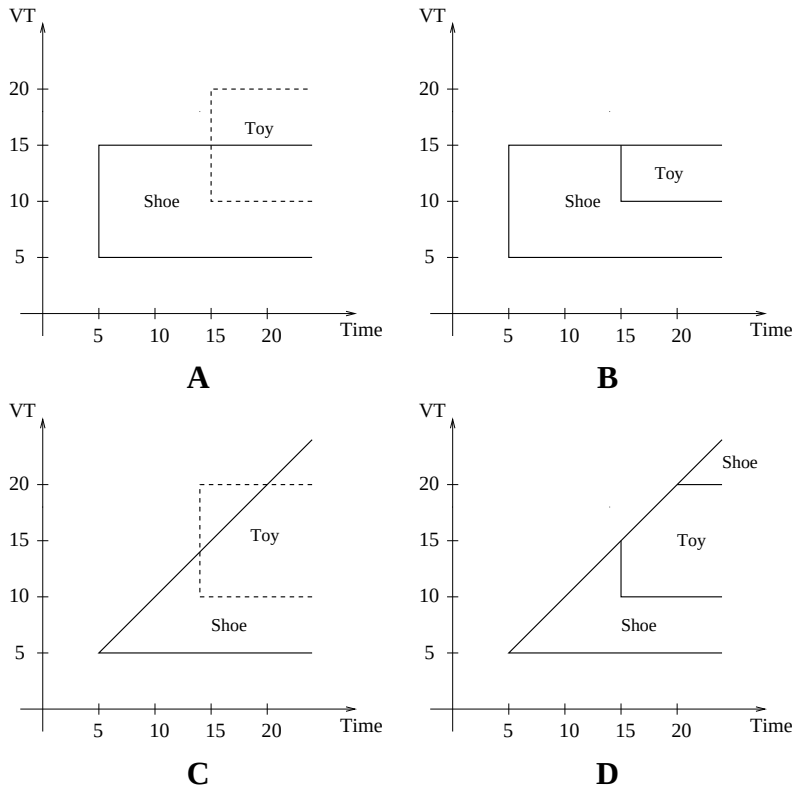


Figure 3: The Results of Updates Without and With NOW

the Shoe department. By updating exactly the overlap, we avoid basing the semantics on assumptions such as the optimistic or pessimistic assumptions discussed in Section 2.

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

4.3 Road Map for Defining 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 found there.

To permit periods containing NOW, as demonstrated in Section 4.2, we have to extend the period 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 period end-points that we have to store in the database and that the generalized operators must then contend with. For example, we must determine which period-end points are needed to accurately store the intersection of two periods such as [5,NOW) and [10,20) in the database (see the $\langle$Joe, Toy$\rangle$ tuple in Figure 3D).

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

4.4 Auxiliary Functions

So far, we have employed a 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 that does not include variables. A theoretical framework for providing a logical interpretation of a variable database, i.e., a "translation" from the variable to the extensional level, may be based on a homomorphic mapping from variable-level databases to extensional-level databases [8].

This mapping is termed an *extensionalization*, and is denoted $[\![\ ]\!]$. It maps a database relation by relation, and maps a relation tuple by tuple. Because the explicit columns $(A_1, \ldots, A_n)$ of a tuple contain ground values, the extensionalization mapping simply maps variable timestamp values to ground timestamp values, i.e., it maps elements of $\mathcal{T}_1$ to elements of $\mathcal{T}$. As the mapping of the timestamp values of a tuple is independent of the tuple's explicit-column values, it suffi ces to map the time domain in isolation.

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 begin 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; and at $c = 15$, it is back to 10.

With the domain of a being $\mathcal{T}_1$, we defi ne 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 defi ning modifi cations of Section 3 when defi ning modifi cations for variable databases. To do so, we introduce generalized minimum and maximum functions, min^v and max^v, respectively, that permit the variable NOW.

We defi ne a new domain of period end-points as $\mathcal{T}_f = \mathcal{T} \cup \{min^v(a, NOW),\ max^v(a, NOW)\}$, where $a \in \mathcal{T}$. It turns out that this domain of values is suffi cient for representing the results of temporal modifi cations. We will show in the next section that the domain can easily be represented in standard SQL. We emphasize that storing strings containing min^v and max^v functions will not be necessary.

The elements in $\mathcal{T}_f$ that involve NOW are particularly interesting. We show two such examples in Figure 4 using extensionalization diagrams. The examples show that the min^v function (A) gives the time value an upper bound and that the max^v function (B) gives the time value a lower bound. Notice that the sloping (non-horizontal) lines follow the diagonal.

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

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

Note that the min^v and max^v functions reduce to their conventional counter-part when the arguments are from the domain $\mathcal{T}$. Note also that the extensionalization of an element of $\mathcal{T}_f$ is a natural number.

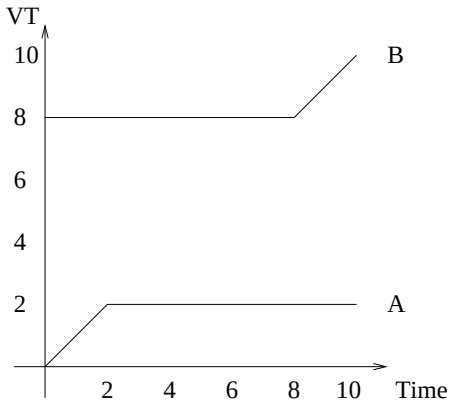


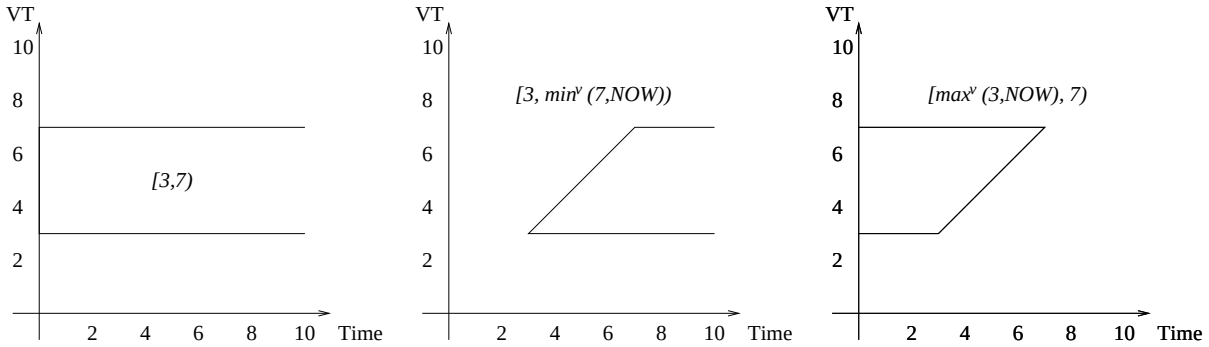
Figure 4: (A) $min^v(2, NOW)$ and (B) $max^v(8, NOW)$

The extensionalization of a period [a,b) is defined in the obvious fashion.

$$[\![[a, b)]\!]_c \triangleq [[\![a]\!]_c, [\![b]\!]_c)$$

4.5 Period Types for Modifications Involving NOW

Periods using the three types of values in $\mathcal{T}_f$, termed *canonical periods*, are shown in Figure 5. The remaining period types over $\mathcal{T}_f$, e.g., $[min^v(a, NOW), b)$ and $[a, max^v(b, NOW))$, create more complex shapes that are a combination of the shapes shown in Figure 5. These period are harder to use in the definition of the semantics and still require the use of the period types shown in Figure 5. A final period type $[max^v(a, NOW), min^v(b, NOW))$ will be introduced in Section 5 to handle an extended semantics, that of now-relative values.



A: Fixed Period **B:** Increasing Period **C:** Decreasing Period

Figure 5: Period Types Needed for Modifications Involving NOW

Figure 5A exemplifies a conventional fixed period of type $[a, b)$, specifically, $[3, 7)$. Figure 5B shows an increasing period of type $[a, min^v(b, NOW))$, specifically $[3, min^v(7, NOW))$. Note that the period type exemplified in Figure 5B may continue to grow; this is specified as $[a, min^v(\textit{forever}, NOW))$. Figure 5C shows a decreasing period of type $[max^v(a, NOW), b)$, specifically, $[max^v(3, NOW), 7)$. Note again the special case, specified as $[max^v(\textit{beginning}, NOW), b)$, where the period starts at *beginning* (*beginning* and *forever* were introduced in Section 3.1). Also note that as time passes, the duration of the period shrinks, until the period is empty. Finally, please observe that the three shapes in Figure 5 can be combined to yield the shapes in Figures 3B and 3D.

4.6 Extending the Period Difference and Intersection Operators

Having established $\mathcal{T}_f$ as the domain of period end points, the next step is to extend the period difference and intersection operators to apply over such periods. 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 period 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 $\mathcal{T}_f$ and recall that the domain of a, b, c, and d is $\mathcal{T}$ (we assume these domains in the rest of this paper). The extended period difference operators ($-^v$) is defined as follows.

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

The definition covers 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.

Note that the third case in the definition takes a constant and an increasing period as inputs and returns a decreasing period. 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 periods.

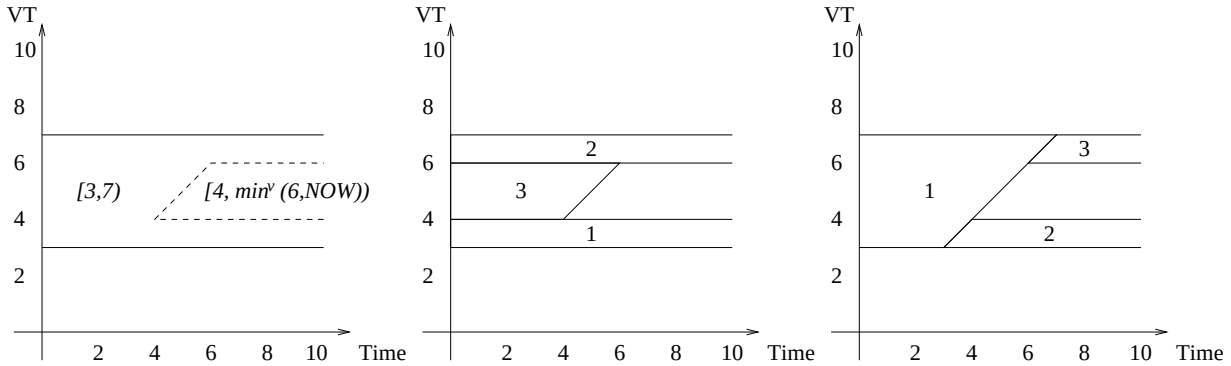


Figure 6: Extended Period Difference Example

In Figure 6A, the extensionalization of the period $[3, 7)$ is illustrated by the solid lines, and the extensionalization of the period $[4, min^v(6, NOW))$ is given by the dashed line. The three resulting periods are shown in Figure 6B. The first period 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 periods 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. An important corollary is that $\mathcal{T}_f$ is closed under the extended period difference ($-^v$).

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

$$[\alpha, \beta) \cap^v [\gamma, \delta) \triangleq$$

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

Note again that the extended intersection operator reduces to the conventional intersection operator if the argument period 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 $[3, 7) \cap^v [max^v(2, NOW), 6)$.

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

We have defined the extended period difference and intersection operators to be 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 period end points are in $\mathcal{T}$. This was required in Section 4.1 (Requirement **R2**: *naturalness*, or equivalently, re-

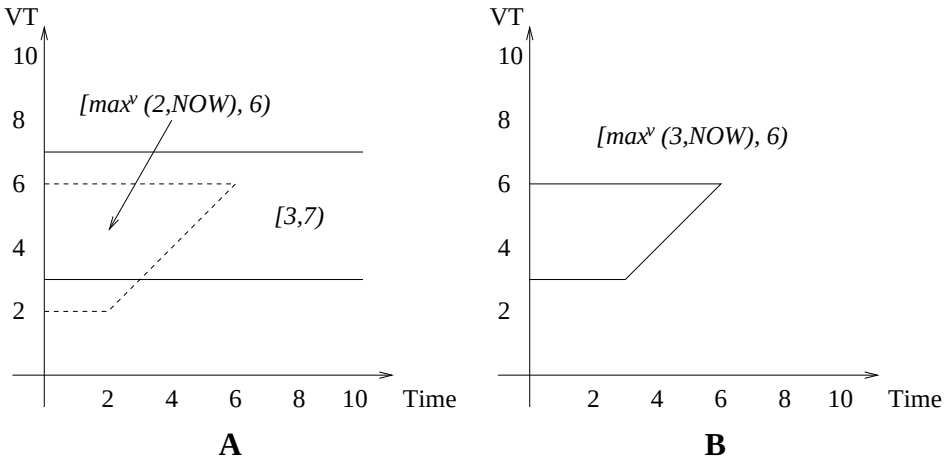


Figure 7: Extended Period Intersection Example

ducibility [22]).

- 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.
- $\mathcal{T}$ is closed under period difference and intersection; $\mathcal{T}_f$ is closed under the extended analogues of these two operators.
- The periods returned by the basic (respectively extended) difference operators are pairwise disjoint. For example, the three result periods in Figure 6B do not overlap.
- The periods returned by the extended difference operators are coalesced [6], so a result containing, e.g., $[5, 7)$ and $[7, 8)$ will not occur. Instead the period $[5, 8)$ would be returned.
- The extended intersection operator returns at most one period, as does the conventional intersection operator.
- As we will show in Section 6.1, the domains of both the conventional period operators ($\mathcal{T}$) and the extended variants ($\mathcal{T}_f$) may be represented in standard SQL, thus satisfying requirement **R3**.

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

- The extended difference and intersection operators accommodate periods defined by using the variable NOW. This was a requirement in Section 4.1 (**R1**: *expressiveness*).
- The extended difference operator returns zero to three periods, whereas the conventional difference operator returns zero, one, or two periods. In the example in Figure 6B, three periods are returned. This is unavoidable and happens when finding the difference between a constant and a non-constant period, where the non-constant period is included in the constant period (illustrated in Figure 6B).
- The result of an extended difference operation may be composed of different sets of tuples with different periods, but with the same semantics, whereas the result of a conventional difference

operation is unique. As an example, the result in Figure 6B can also be given as the three periods $[max^v(3, NOW), 7)$, $[3, min^v(4, NOW))$, and $[6, min^v(7, NOW))$ in Figure 6C. Both results (6B and 6C) have the same semantics, so it is merely a matter of taste which one to return. We choose to return as few periods as possible with the min^v and max^v functions, to make the results easy for end-users to interpret.

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

4.7 Temporal Modification Semantics

The definitions of the insert, delete, and update of periods 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 period difference and intersection operators are used. This provides an entirely straightforward satisfaction of all three requirements of Section 4.1.

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

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

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

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

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

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

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

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

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

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 period $[10, 15)$. 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.

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

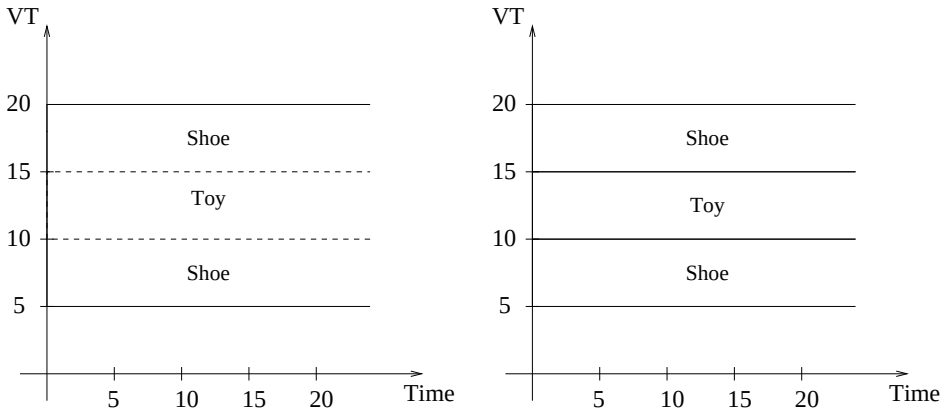


Figure 8: Joe is in the Toy Department for the Period from 10 to 15

From the single tuple stored in the relation, we remove Joe from the Shoe department in the period $[10, 15)$. This results in two tuples. Further, we update Joe to be with the Toy department in the intersection of the periods $[5, 20)$ 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.

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

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 period $[5, NOW)$. The dashed rectangle corresponds to the period [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 expected one. The scope of the update is just 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 period $[5, min^{v}(10, NOW))$ (the lower region) and $[20, min^{v}(\textit{forever}, NOW))$ (the upper region).

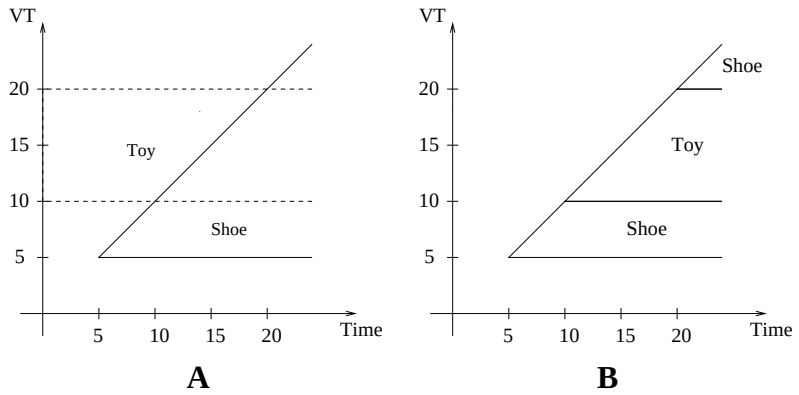


Figure 9: Update of a NOW-Relative Database

5 Semantics of Modifications Involving Now-Relative Values

In some applications, the periods 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 (a *retroactively bounded* database [15]). 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 [9].

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 period, used for supporting the more general databases that result from the NOW-relative values, is introduced, and the period operations ($-^v$ and $\cap^v$) are extended once again to also permit these new periods. On this basis, the modifications are defined. This development parallels that of Section 4, though the technical details are somewhat more involved.

5.1 Definition of NOW-Relative Values

NOW-relative values generalize variable NOW by allowing offsets from NOW to be specified [9]. 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, $NOW + n$, where n belongs to a domain of durations that is isomorphic to a subset of the integers (both positive and negative values are allowed), is defined as follows [9].

$$[\![NOW + n]\!]_c \triangleq [\![NOW]\!]_c + n$$

5.2 A New Period Type

To extend the modifications to permit NOW-relative periods, a single new period type is needed over the three introduced in Section 4.5. Intuitively, we need a period whose extensionalization diagram is a parallelogram. With this extra period type, the domain of values is closed under the operations defined, and we can thus define the semantics of insert, delete, and update. The extensionalization

diagram in Figure 10 gives an example of this new type of period, namely the period $[max^v(3, NOW - 3), min^v(7, NOW + 2))$. The dashed line indicates the diagonal. This period has a max^v function in

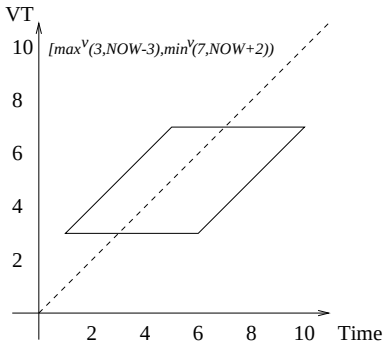


Figure 10: New Period Type for NOW-Relative Modifi cations

its beginning point and a min^v function in its ending point; the earlier periods had at most a function in either the beginning or the ending point. Note also the now-relative offsets, -3 and $+2$. The -3 in the beginning point indicates that the beginning 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.

The offset of the beginning point of a period must be smaller than or equal to the offset of its ending point. Otherwise, the period is undefi ned. For example, the period $[max^v(3, NOW+2), min^v(7, NOW-3))$ is undefi ned.

Formally, we defi ne the meaning of a NOW-relative period 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 period type comes into existence and defi ne also how it is handled in the period difference and intersection operators.

5.3 Extending the Period Operators

The sixteen cases we must consider when extending the period 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 ($\cap^v$).

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

The extended period difference operation, $[\alpha, \beta) \ -^v \ [\gamma, \delta)$, where the argument periods 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, a conditional case and its complement, an "otherwise" case. The first case for a combination is identified by the integers in Formula 2 and in the first and second columns in the table (the case numbering is used in the discussion below). The third column gives a condition that must be satisfied for this case to apply. The last, "otherwise" case in the table applies if the prior condition is not satisfied. The result in each circumstance, a set of periods, is given in the rightmost column of the table.

It may be observed that the extended difference operator reduces to the conventional difference operator when the period 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 periods (in cases 4, 8, 12, and 16), and the new period 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 period in case 5 is empty because $\alpha > \gamma$ ($5 > 3$).

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 periods shown in Figure 12B. These periods derive from the first and third periods in case 11; the second period is empty because $\delta > \beta$ ($8 > 6$). When we compute the difference of two decreasing periods as here, the new type of period from Section 5.2 results.

Figure 11C illustrates $[max^v(2, NOW), 8) -^v [4, min^v(6, NOW + 2))$. Here, the offset of the first period is 0. Finding the difference of a decreasing and increasing period (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 periods in 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 periods shown in Figure 12D. This example illustrates the overall strategy used in defining period difference. We look above, below, to the right, and to the left of the second argument period, determining what remains of the first argument period. Looking to the right and left, we consider only the parts of the first argument period that have not been covered by looking above and below. This is implemented in

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

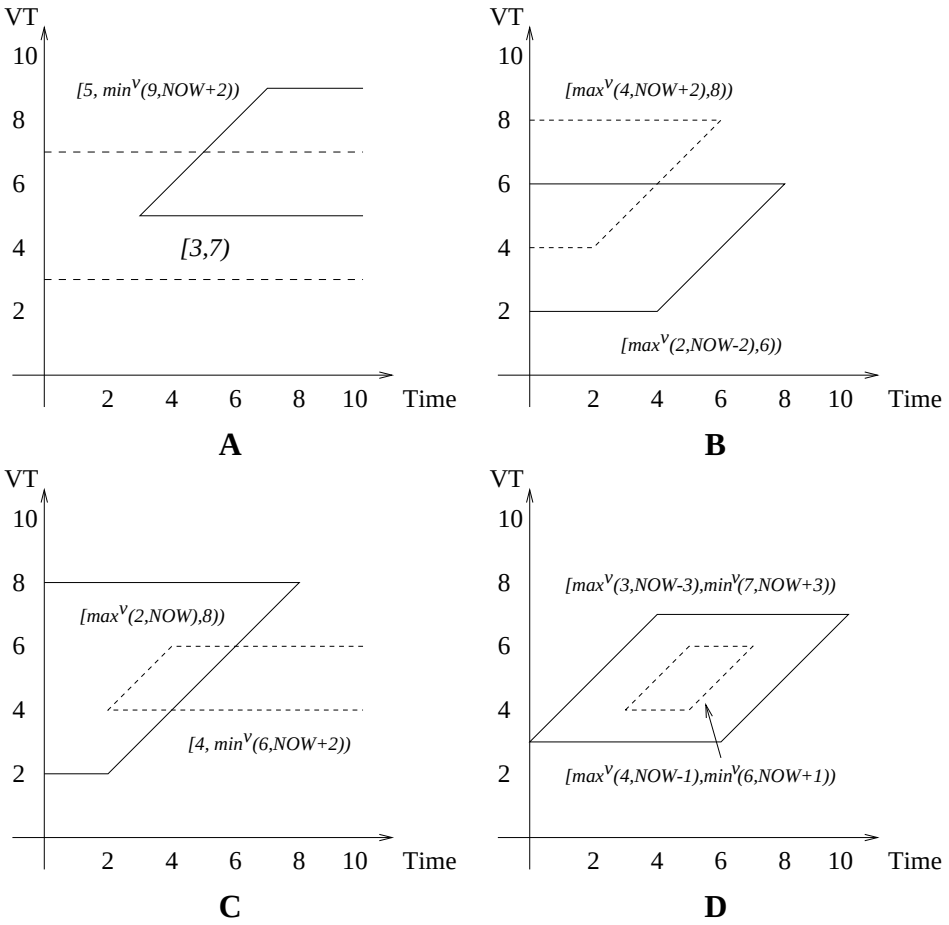


Figure 11: NOW-Relative Period Difference Examples

the definition using the standard max and min functions. Using this strategy, the difference operator returns non-overlapping regions and does not duplicate information.

The extended period 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 periods are in $\mathcal{T}$, which is also covered here by the first and last cases in the definition. The operator always returns only one period, as does the conventional intersection operator, and the new period type is returned by case 4.

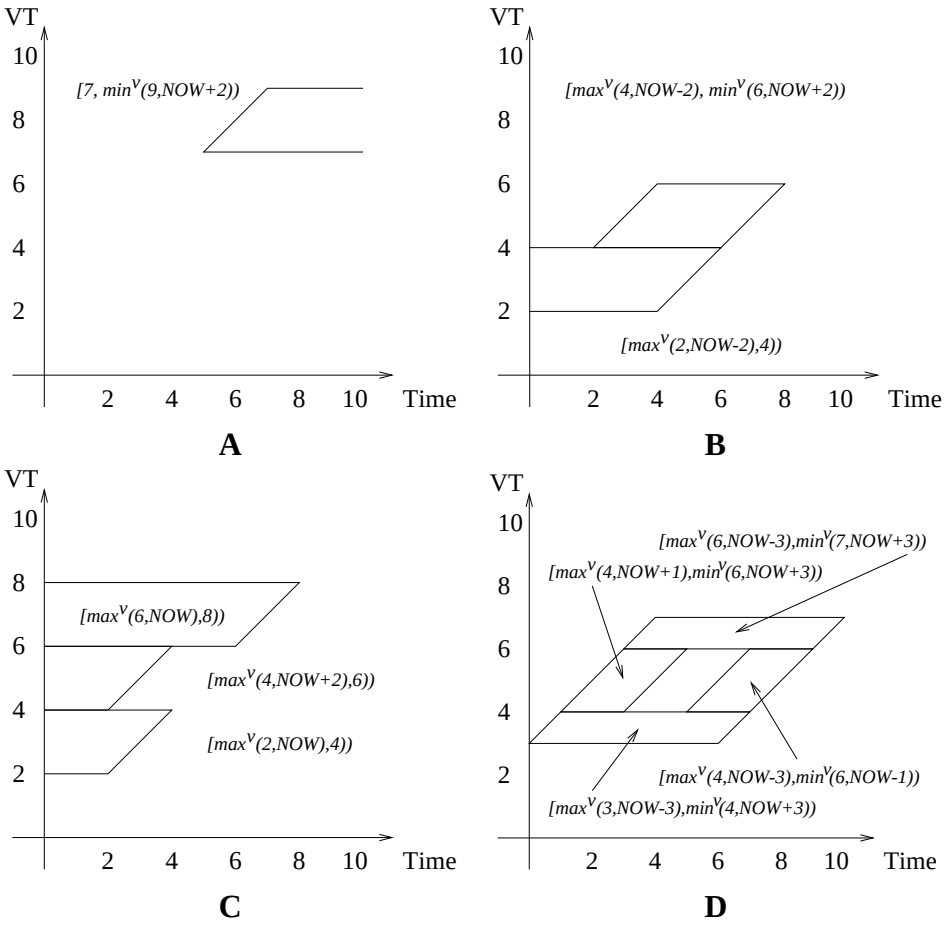


Figure 12: NOW-Relative Period Difference Results

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

The extensionalization diagrams in Figure 11 may also serve to illustrate the extended period intersection operator. Figure 11A illustrates $[5, min^v(9, NOW + 2)) \cap^v [3, 7)$, which is covered by case 3 (lines 7–9). The result, period $[5, min^v(7, NOW + 2))$, 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 (lines 4–6) 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 (lines 10–13) in the definition. Figure 11C illustrates $[max^v(2, NOW), 8) \cap^v [4, min^v(6, NOW + 2))$, which results in the period $[max^v(4, NOW), min^v(6, NOW + 2))$ (shown in Figure 13C). Thus the offsets cause the intersection operator to return a period 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 period $[max^v(4, NOW - 1), min^v(6, NOW + 1))$, shown in Figure 13D.

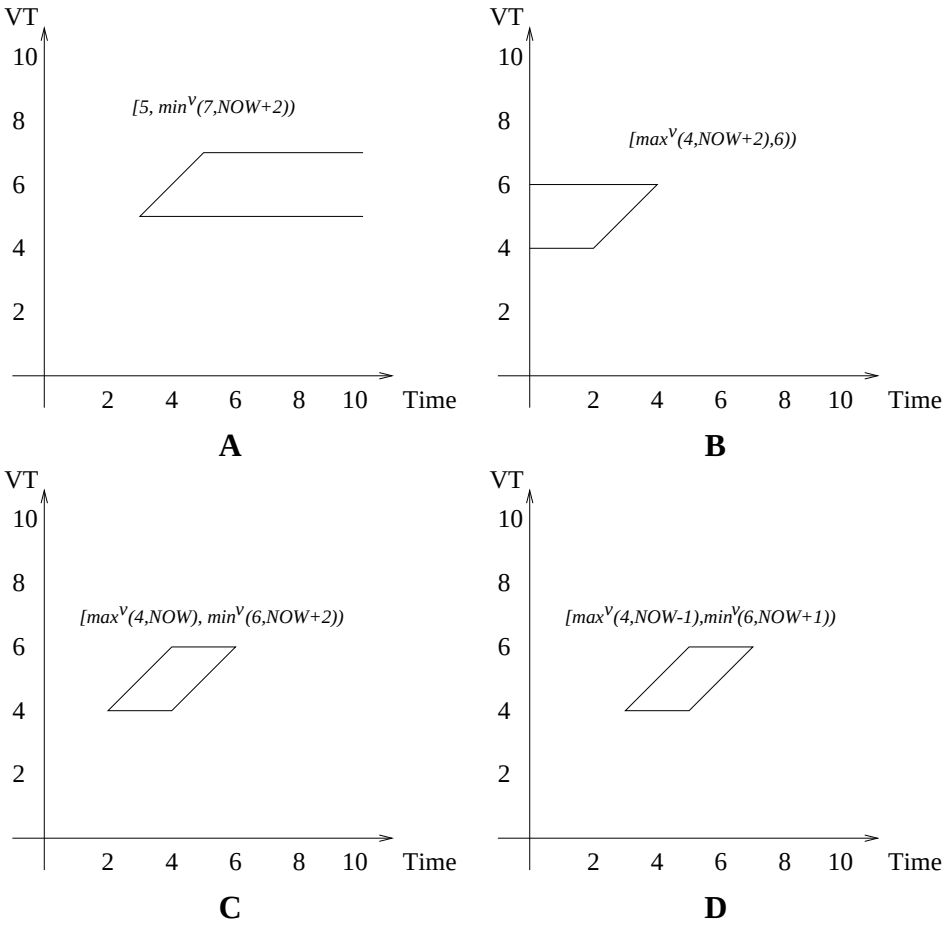


Figure 13: NOW-Relative Period Intersection Results

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 illustrate the utility of NOW-relative values through an example.

In the following scenario, Joe joins the Shoe department on the 3^{rd}. At any time, he can leave his job with eight days notice. On the 6^{th}, he is told that with two days notice, he can be reassigned to the Toy department for four days. On the 8^{th}, he gives notice that he will leave his job (i.e., his last day is the 16^{th}).

This scenario is captured by the following database modifications, where `NOBIND(CURRENT_DATE)` denotes the value NOW [9] and a non-null value is used as the offset. Further, `FOREVER` denotes the

maximum timestamp.

```
-- 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 dept. for 4 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 will leave on the 16th
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 14B.

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

We plan that Joe may be temporarily in the Toy department in the period [NOW+2,NOW+6) where the +2 indicates the two days notice. This period 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 14C.

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

From the three tuples stored in the relation at the outset, we delete the period [16,*forever*), to indicate that Joe is leaving the company on the 16^{th}. The period 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.

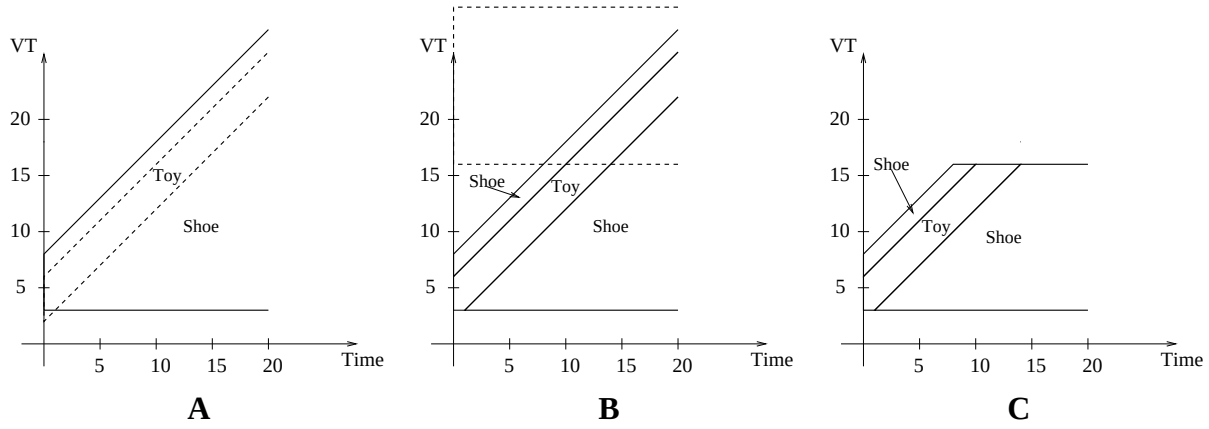


Figure 14: The Results of Updates With NOW-Relative

6 Implementing Modification Statements Involving NOW-Relative Values

This section considers the implementation of the modifications defined in the previous two sections. By encapsulating the modification semantics within the DBMS, the complexity of their definition becomes transparent to the users, which in turn ensures that they are useful in practice.

There are two basic approaches to implement the now-relative semantics described above. The first, termed the *stratum* approach [27], imposes a layer that translates variable queries and modifications into conventional SQL, which is then executed by an (unaltered) conventional DBMS. The generated SQL might exploit advanced features provided by the underlying DBMS, such as abstract data types or object-relational features. The second, *integrated*, approach is to modify the DBMS directly, a task that is perhaps harder initially but may yield better performance. These two approaches permit a two-pronged migration path: first a stratum is implemented, then key parts of the stratum are later migrated into the DBMS.

In the following, we emphasize the stratum approach while also considering the integrated approach. We note in passing that these somewhat complex mappings are needed only when the user explicitly requests them, via `NOBIND` in modifications.

6.1 Representing Canonical Periods

The canonical periods, defined in Section 4.5, can be implemented in SQL by using four columns. Two columns record the `V-Begin` and `V-End` columns, 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 table 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 a V-End-Offset value different from NULL indicates a max^v and a min^v function, respectively. This representation exploits the convenient fact that max^v can only occur in the beginning delimiting timestamp and min^v can only occur in the ending delimiting timestamp of a canonical period. Recall that NOW can be written as $max^v(\textit{beginning}, NOW)$ in the V-Begin column and $min^v(\textit{forever}, NOW)$ in the V-End column. For example, the tuple $\langle \text{Joe}, \text{Toy}, [10, min^v(20, NOW))\rangle$ is stored as the row ('Joe', 'Toy', DATE '2003-01-10', NULL, DATE '2003-01-20', 0) in the EMP table.

The valid-time period can also be implemented as an abstract data type (ADT), e.g., in the Oracle DBMS. The main advantage of utilizing a separate ADT is that the valid-time period 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 begin time, and BIND_E to bind the end time. These functions in concert provide a variant of the extensionalization function $[\![\ ]\!]_c$ defined in Section 5.2.

To illustrate querying, suppose we want to retrieve all tuples with a valid time that overlaps the period [2003-01-05,2003-01-15). This can be written in a temporal SQL [7, 24] as follows.

```
VALIDTIME PERIOD [DATE '2003-01-05', DATE '2003-01-15')
SELECT * FROM Emp;
```

This query formulated in SQL 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 '2003-01-15'
AND    DATE '2003-01-05' < BIND_E(V-End, V-End-Offset)
AND    BIND_B(V-Begin, V-Begin-Offset) < BIND_E(V-End, V-End-Offset)
```

In the first two lines of the where clause, it is checked that the valid time associated with a tuple overlaps with the temporal scope, i.e., the period [2003-01-05,2003-01-15). This corresponds to a simple overlap predicate. In the last line, it is checked that the V-Begin column is smaller than the V-End column. This last check is needed when valid-time columns are bound to the current date. Note that this query returns a ground result, i.e., binds the valid-time values in the select list.

As an example, the BIND_B function can be specified in PSM [19] as follows.

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

If the offset is null, V-Begin is returned, i.e., the V-Begin column does not contain a NOW-relative value. V-Begin is also returned unchanged if V-Begin is larger than NOW plus the offset (in days), i.e., implementing the maximum function. Otherwise, NOW plus the offset is returned. The BIND_E function that binds V-End values is analogous. These functions can also be implemented in Oracle's PL/SQL or as user-defined functions in Informix or DB2. One could envision a BINDOVERLAPS function that would greatly simplify the WHERE clause.

If implemented inside the DBMS, rather than with an external translator 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.

It should be noted that no existing temporal SQL supports the domain $\mathcal{T}_f$ for its valid times. Thus, extensions are needed in order for these languages to support the full functionality presented in this paper. Such extensions are orthogonal to this paper's objective: that of giving semantics for modifications in variable databases that are independent of any specific query language.

6.3 Implementing Modifications

Having illustrated querying when representing temporal data using the format with four timestamp columns, we proceed to consider modification. This is more complicated than implementing queries, and we split the presentation into implementing inserts, then deletes and finally updates.

6.3.1 Implementing Insert

Insertions are easy to map to SQL: we simply set the offset columns depending on the presence of NOW. As an example, we insert the tuple ⟨Joe, Shoe, [2003-01-05,2003-01-07) ⟩. This can be expressed in a temporal SQL [7, 24] as follows.

```
VALIDTIME PERIOD [DATE '2003-01-05', DATE '2003-01-20')
  INSERT (Joe, Shoe)
```

This insertion may be mapped by the stratum into following SQL statement:

```
INSERT INTO Emp VALUES
  ('Joe', 'Shoe', DATE '2003-01-05', NULL, DATE '2003-01-20', NULL);
```

There are no NOW-relative values so both offset columns have null values inserted.

When the value NOW is used, (specified as NOBIND(CURRENT_DATE)) the offset columns are used. Let us assume that we want to insert the tuple ⟨Joe, Shoe, $[2003-01-05, NOW)$ ⟩. This can be expressed as follows in a temporal SQL [7, 24].

```
VALIDTIME PERIOD [DATE '2003-01-05', NOBIND(CURRENT_DATE))
  INSERT (Joe, Shoe)
```

It is mapped into the following SQL statement:

```
INSERT INTO Emp VALUES
  ('Joe', 'Shoe', DATE '2003-01-05', NULL, DATE '9999-12-31', 0);
```

6.3.2 Implementing Delete

Deletions should conform to the semantics presented in Sections 4.7 and 5.4. The main problem when implementing deletions is that the extended difference operator may return up to three periods for the semantics specified in Section 4.7 and up to four periods for the semantics specified in Section 5.4.

To solve this problem we use the idea illustrated by the extended period 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 period $[c, d)$ and determine what remains of period $[a, b)$.

To illustrate the most involved case (that is shown in Figure 11D), consider a database containing the tuple $\langle$Joe, Shoe, $[max^v(\text{2003-01-03}, NOW - 3), min^v(\text{2003-01-07}, NOW + 3))\ \rangle$ and a temporal deletion statement that causes us to delete Joe from the Shoe department for the applicable period $[max^v(\text{2003-01-04}, NOW - 1), min^v(\text{2003-01-06}, NOW + 1))$. Note that most deletes will be simpler and will generate shorter code; we cover this rather extreme case here to illustrate the subtleties that are encountered.

The stratum maps the example temporal delete statement just mentioned into the following four SQL insert statements and a single SQL delete statements. The four insert statement each covers the four subregions ("above," "below," "right," and "left") of the resulting region. The delete statement removes the old tuple. Note that all of the insert statements are very similar in form, and that the last three lines of the where clause of each are identical. The use of "1" and "-1" in the SQL code will be explained shortly. Again, this SQL code would be greatly simplified by using appropriate functions defined say in PSM [19].

```
-- Above
INSERT INTO Emp
SELECT Name, Dept, DATE '2003-01-06', V-Begin_Offset, V-End, V-End-Offset
FROM   Emp
WHERE  Name = 'Joe' AND V-End > DATE '2003-01-06' AND
       V-Begin < DATE '2003-01-06' AND DATE '2003-01-04' < 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 '2003-01-04', V-End-Offset
FROM   Emp
WHERE  Name = 'Joe' AND V-Begin < DATE '2003-01-04' AND
       V-Begin < DATE '2003-01-06' AND DATE '2003-01-04' < 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 '2003-01-04'), V-Begin-Offset,
       LEAST (V-End, '2003-01-06'), -1
FROM   Emp
WHERE  Name = 'Joe' AND -1 IS NOT NULL AND
       GREATEST (V-Begin, DATE '2003-01-04') < LEAST (V-End, DATE '2003-01-06')
       AND (NOT (V-Begin-Offset IS NOT NULL AND V-Begin-Offset > -1)) AND
       V-Begin < DATE '2003-01-06' AND DATE '2003-01-04' < 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 '2003-01-04'), 1,
       LEAST (V-End, DATE '2003-01-06'), V-End-Offset
FROM   Emp
WHERE  Name = 'Joe' AND 1 IS NOT NULL
       GREATEST (V-Begin, DATE '2003-01-04') < LEAST (V-End, DATE '2003-01-06')
       AND (NOT (V-End-Offset IS NOT NULL AND 1 > V-End-Offset)) AND
       V-Begin < DATE '2003-01-06' AND DATE '2003-01-04' < 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 '2003-01-06' AND DATE '2003-01-04' < 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 code above uses the Oracle-specific functions GREATEST and LEAST, which correspond to the conventional max and min functions used in this paper.

In the where clauses, the time overlap predicate (the third-to-last line) and the offset overlap predicate (the last two lines) check that the periods associated with the tuples overlap with the period specified in the delete statement. The offset overlap predicate checks for overlap between the offsets on the periods in the database and the offset specified in the period to delete. Because these offsets can

have the value NULL, we must, for each less-than operator, check if the operands are NULL. The '1' in , e.g., in `1 IS NULL` is the V-End-Offset of the applicable period. Similarly, the '-1', e.g., in `-1 < V-End-Offset`, is the V-Begin-Offset of the applicable period.

In the first insert statement the check `V-End > DATE '2003-01-06'` ensures that an "above" tuple is generated when appropriate. Similarly, the check in the second insertion, `V-Begin < DATE '2003-01-04'`, 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 period 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` column is `NOT NULL` then its value cannot be larger than -1, which is the `V-Begin-Offset` of the period 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 period 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 period specified in the temporal deletion statement.

After the four insertions, all tuples that overlap with the period 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 period.

Note that because the semantics specified in Sections 4.7 only allows a max^v function in the `V-Begin` column or a min^v function in the `V-End` column, 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.

6.3.3 Implementing Update

Updates are implemented similarly to deletes. Assume that we want to update Joe for the period [2003-01-04,max^v(2003-01-06, NOW)) to be with the Toy department. In temporal SQL, this may be written as follows [7, 24].

```
VALIDTIME PERIOD [DATE '2003-01-04', NOBIND(CURRENT_DATE))
UPDATE Emp SET Dept = 'Toy'
WHERE Name = 'Joe';
```

This is converted by the stratum into conventional SQL as four insertions and an update. The four insertions are identical to those displayed for the temporal deletion in the previous section. The update statement follows.

```
-- Update self
UPDATE EmpNow
SET    Name, Dept,
       V-Begin        = GREATEST (V-Begin, DATE '2003-01-04'),
       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 '2003-01-04') IS NOT NULL AND
            LEAST (V-End, NULL) IS NOT NULL AND
            GREATEST (V-Begin, DATE '2003-01-04') >
               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 '2003-01-06' AND DATE '2003-01-04' < 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` column is set to the maximum of the start of the period of the tuple being updated and the beginning of the period specified in the update. In line 4, the `V-End` column is set to the minimum of its current value and the end of the period in the update. This is done similarly for the `V-Begin-Offset` and `V-End-Offset` columns in lines 5 and 6, respectively. In the where clause, it is checked in lines 8–11 that the new period is non-empty. Similarly, lines 12–14 check that the `V-Begin-Offset` is smaller than the `V-End-Offset`. Line 15 checks overlap on the `V-Begin` and `V-End` columns, and the `V-Begin-Offset` and `V-End-Offset` columns. 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 in order to enforce the correct semantics with respect to the translation of queries [24].

7 Related Work

Most prominently, this paper proposes definitions of modifications involving NOW and explores how the resulting semantics can be realized 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 periods have been defined and implemented.

In the perhaps most closely related paper [9], the semantics of NOW is described in substantial detail. That paper 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 periods, as adopted in this paper, generalizes timestamping tuples with single time point values, e.g., as done in time series. By using a period representation, we can capture constant, increasing, and decreasing periods. Had single time points been

used instead, it would only be possible to capture either increasing or decreasing periods 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 [17] extend SQL-92 to handle general period data, e.g., periods in space or time (in some of the temporal database literature, periods are referred to as intervals, which is confusing in our context, as SQL provides its own (and distinct) notion of interval). The semantics of modifications involving periods is defined in that paper, and details are provided for how to retain coalescing of relations. However, the use of variables such as NOW is not considered. The modification semantics therefore cover only the special case of periods whose end points belong to domain $\mathcal{T}$.

Finger and McBrien [11, 12] 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 [29], 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 involved in choosing which value to use for NOW are orthogonal to the issues discussed in this paper.

8 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, either in a stratum as a mapping to standard SQL, or as a more efficient internal implementation.

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 period intersection and difference operators. It is shown that the databases that result from modifications involving NOW can be represented by using three types of periods with three types of values: normal periods with fixed end points, a new kind of period that increases with time, and another new kind of period that decreases as time passes. These periods involve two new kinds of time values. By including a fourth kind of period, now-relative values are permitted.

By defining modifications, the paper consistently extends past work [9] and completes the semantics, the querying, and the modification of NOW-relative databases. The extension to the syntax of SQL is restricted to a single new `NOBIND( )` function, which avails full variable and now-relative support to the application, which previously had to resort to complex ad hoc schemes involving particular DATE values and user code to manage these values. Utilizing a simple syntax to stand in for

a sophisticated concept follows the time-honored emphasis in database systems of moving complexity (such as concurrency control, recovery, indexing, buffer management) from the application into the underlying DBMS. Note that this mapping is required only when the user requests it, through `NOBIND( )`. This default follows another time-honored tradition of incorporating a more sophisticated semantics and implementation only when explicitly requested by the application.

It is challenging to index periods containing the two new types of values used in the definition of the semantics. While conventional indexes for periods support only fixed values, indexing techniques have recently been proposed that support valid-time periods that end with NOW [16], combined valid-time and transaction-time periods that end with NOW in so-called bitemporal databases [4, 5], and combined valid-time and transaction-time periods that end with NOW or $NOW + \Delta$ in spatio-bitemporal databases [21]. However, indexing periods that *begin* with the two new types of values has yet to be considered.

In early work, Allen [2] proposed a set of 13 binary relations among regular, ground periods. Any two periods are related according to exactly one of these 13 relations. It would be of interest to identify a set of relations with the same property, but for the generalized periods proposed in this paper.

Acknowledgments

The authors would like to thank the anonymous reviewers and especially the handling editor, Nicole Bidoit, for their insightful reviews and suggestions, which helped improve the presentation of the research.

This research was supported in part by the Danish Technical Research Council through grant 9700780, by a grant from the Nykredit corporation, and by grants IRI-9632569, IIS-9817798, IIS-0100436 and EIA-0080123 from the U.S. National Science Foundation.

The research was performed in part while the first author was employed by Logimatic Software A/S, Denmark.

References

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

[2] J. F. Allen. Maintaining Knowledge about Temporal Intervals. *CACM*, 26(11): 832–843 (1983).

[3] 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.

[4] R. Bliujūtė, C. S. Jensen, S.Šaltenis, and G. Slivinskas. R-Tree Based Indexing of Now-Relative Bitemporal Data. In *Proceedings of the Conference on Very Large Databases*, pp. 345–356, 1998.

[5] R. Bliujūtė, C. S. Jensen, S.Šaltenis, and G. Slivinskas. Light-Weight Indexing of Bitemporal Data. In *Proceedings of the International Conference on Scientific and Statistical Database Management*, pp. 125–138, 2000.

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

[7] M. Böhlen, C. S. Jensen, and R. T. Snodgrass. Temporal Statement Modifiers. *ACM Transactions on Database Systems*, 25(4):407–456, December 2000.

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

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

[10] O. Etzion, S. Jajodia, and S. Sripada (eds.). *Temporal Databases: Research and Practice*. LNCS 1399, Springer, 1998.

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

[12] 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.

[13] S. K. Gadia and S. Nair. Temporal Databases: A Prelude to Parametric Data. [26, Ch. 2, pp. 28–66].

[14] C. S. Jensen and C. E. Dyreson (eds.). The Consensus Glossary of Temporal Database Concepts—February 1998 Version. [10, pp. 367–405].

[15] Jensen, C. S. and R. T. Snodgrass. Temporal Specialization and Generalization. *IEEE Transactions on Knowledge and Data Engineering*, Vol. 6, No. 6, December, 1994, pp. 954–974.

[16] H.-P. Kriegel, M. Pötke, and T. Seidl. Managing Intervals Efficiently in Object-Relational Databases. In *Proceedings of International Conference on Very Large Data Bases*, pp. 407–418, 2000.

[17] N. Lorentzos and Y. Manolopoulos. SQL Extension for Interval Data. *IEEE Transactions on Knowledge and Data Engineering*, 9(3):480–499, 1997.

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

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

[20] S. Navathe and R. Ahmed. Temporal Extensions to the Relational Model and SQL. [26, Ch. 4, pp. 92–109].

[21] S. Šaltenis and C. S. Jensen. Indexing of Now-Relative Spatio-Bitemporal Data. *VLDB Journal*, 11(1):1–16, 2002.

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

[23] R. T. Snodgrass. (editor), I. Ahn, G. Ariav, D. Batory, J. Clifford, C. E. Dyreson, R. Elmasri, F. Grandi, C. S. Jensen, W. Käfer, N. Kline, K. Kulkarni, T. Y. C. Leung, N. Lorentzos, J. F. Roddick, A. Segev, M. D. Soo and S. M. Sripada. **The TSQL2 Temporal Query Language**, Kluwer Academic Publishers, 1995, 674+xxiv pages.

[24] 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.

[25] R. T. Snodgrass. *Developing Time-Oriented Database Applications in SQL*. Morgan Kaufmann Publishers, 1999.

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

[27] 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.

[28] K. Torp, C. S. Jensen, and R. T. Snodgrass. Modification Semantics in Now-Relative Databases. TIMECENTER Technical Report TR-43, September 1999.

[29] K. Torp, C. S. Jensen, and R. T. Snodgrass. Effective Timestamping in Databases. *VLDB Journal*, Vol. 8, Issue 3+4, February 2000, pp. 267–288.