# Contents

**1 Introduction** **1**

**2 Valid Time** **1**

2.1 Current Usage . . . . . 2

2.2 Tomorrow's Employment Status . . . . . 2

2.3 The Punctuality Assumption . . . . . 3

2.4 The Non-predictive Update Assumption . . . . . 4

2.5 Now-relative Events . . . . . 4

2.6 An Aside: Now versus Present . . . . . 6

2.7 Valid-time Indeterminacy . . . . . 6

2.8 Now-relative Indeterminate Events . . . . . 8

**3 Transaction Time** **9**

3.1 An Example . . . . . 10

3.2 "Now" in Transaction Time . . . . . 10

3.3 "Forever" in Transaction Time . . . . . 11

3.4 "Until Changed" . . . . . 11

3.5 An Aside: "Beginning" in Transaction Time . . . . . 12

**4 Bitemporal Relations** **12**

4.1 The Transaction Timeslice Operator and "Now" . . . . . 13

4.2 A Formal Definition . . . . . 14

**5 Timestamp Representation** **15**

**6 Timestamp Operations** **17**

**7 Conclusion** **19**

**8 Acknowledgements** **19**

**9 References** **20**

What's time? Leave Now for dogs and apes: Man has Forever.
[Robert Browning: A Grammarian's Funeral [Browning 1899]]

## 1 Introduction

*Now* is an English noun meaning "at the present time" [Sykes 1964]. "Now" is also a distinguished timestamp value in many temporal data model proposals. However, the precise semantics of this value has never been fully specified. As a result "now" has a variety of possible interpretations. Some of these interpretations are overly restrictive while others are inappropriate or incorrect. In this paper, we examine the different semantics given this familiar term and propose specific representations, query language constructs, and query processing strategies that better capture this intuitive yet subtle concept. We also explore the related concepts of "infinite future" and "infinite past." We first consider the valid-time dimension, then the transaction-time dimension (as the semantics of "now" is specific to the kind of time involved), and finally *bitemporal* data models which support both time dimensions [Jensen et al. 1992].

This discussion of "now" adds to the set of temporal values and types developed elsewhere [Soo & Snodgrass 1992A, Soo & Snodgrass 1992B]. In particular, "now" is a distinguished kind of *event*, rather than a *span* or *interval*. An event is a fixed point on an underlying time-line whereas an interval is an anchored segment of the time-line demarcated by two events. Two distinguished events currently exist: *beginning*, which is the earliest time on the underlying time-line (valid, transaction, or user-defined time), and *forever*, which is the latest time. (We consider the relevancy of these values to transaction time in Section 3.) Events can further be classified as *determinate* or *indeterminate*. An indeterminate event is an event that is known to have occurred, but exactly when is unknown [Dyreson & Snodgrass 1993A]. An interval bounded by one or more indeterminate events is an indeterminate interval. A span is the duration between two events, an unanchored segment of the time-line. Spans are either *fixed* or *variable*. A fixed span has a duration that is independent of context, it always represents the same amount of time; but the duration of a variable span varies with supplied context. A common variable span is a "month," which could be 28, 29, 30, or 31 days long (in the Gregorian calendar) depending on whether the context is, say, February 1991, February 1992, June 1993, or May 1993, respectively. A span can also be *indeterminate*, that is, of imprecise duration.

## 2 Valid Time

*Valid time* denotes the time that an event occurred in reality, or the time interval that some fact was true in reality [Snodgrass & Ahn 1985].

| | | *Valid time* | |
|---|---|---|---|
| *Employee* | *Status* | *(from)* | *(to)* |
| Michelle | employed | June 1 | now |

Figure 1: Michelle's employment tuple

## 2.1 Current Usage

In the valid-time dimension, a common use of "now" is to indicate that a fact is valid until the current time [Ariav et al. 1984, Bassiouni & Llewellyn 1992, Elmasri et al. 1990, Gadia 1988, Navathe & Ahmed 1989, Sarda 1990, Tansel 1990, Yau & Chat 1991]. In conventional databases, such facts are the only ones that can be represented. For example, suppose that Michelle began working for the ACME corporation on June 1. Figure 1 shows the relevant tuple from ACME's employment history (the `Employees` relation). Michelle started working at ACME on June 1, as indicated by the "from" attribute (for the examples in this paper we assume a timestamp granularity of one day). The value "now," appearing as the "to" time in Michelle's employment tuple, represents a currently unknown future time when Michelle will stop working for ACME. The result of a query that requests ACME's current employees will include Michelle.

The informal semantics of this value is that Michelle is employed until we learn otherwise. As the current time inexorably advances, the value of "now" also changes to reflect the new current time. Some authors have called this concept "until changed" instead of "now" [Wiederhold et al. 1991], but the semantics is the same.

## 2.2 Tomorrow's Employment Status

There are three problems with this use of "now." First, the database explicitly records that Michelle will *not* be employed tomorrow. Assume, for the purpose of this discussion, that today is July 9. A query that asks who will be employed tomorrow (i.e., July 10) will not have Michelle in the answer, since the "to" time of Michelle's tuple is "now," or in this case, July 9. Yet if nothing changes, and we execute the identical query on July 11 (i.e., who was employed on July 10), we will get a different answer, since Michelle will be included.

Some temporal data models avoid this problem by limiting valid-time to the past, that is, to times before "now" [Gadia 1988, Tansel 1990]. We feel that this limitation is much too restrictive.

Other data models (e.g., [Ben-Zvi 1982, Snodgrass 1993, Snodgrass 1987, Thirumalai & Krishna 1988]) address this problem by using "forever" as the "to" time, as shown in Figure 2. Forever is the largest representable timestamp value, that is, the one furthest in the future. This value admits that we do not know when Michelle will depart the company, and so assumes that she will be working forever. One limitation of this fix is that it is overly optimistic: forever is a long time! In SQL and in IBM's DB2, forever is about 8,000 years from the present [Date & White 1990, Melton 1990]; in our more liberal proposal, it is approximately 18 *billion* years from the present time [Dyreson & Snodgrass 1993B]. Hence, to assert that Michelle will be employed forever is most

| | | Valid time | |
|---|---|---|---|
| *Employee* | *Status* | *(from)* | *(to)* |
| Michelle | employed | June 1 | forever |

Figure 2: Michelle's employment tuple with a large right boundary

assuredly incorrect (others have also noted that a "to" time of $\infty$, or "forever," has erroneous implications for the future [Navathe & Ahmed 1989]). A related limitation is that when Michelle departs the company, forever must be revised with the date of her departure; but the revised date will be an entirely separate time, unrelated to "forever."

An alternative way to view this problem is that there is a difference between the *actual* and *expected* times of a fact. On a day-to-day basis, we expect Michelle to remain employed. A database that uses "forever" as the "to" time of her employment tuple (very optimistically) records her expected employment, while a database that uses "now" records only her actual employment, the time she has worked to the current time. We shall show how to represent actual and expected times using a single kind of time value in Section 2.8.

## 2.3 The Punctuality Assumption

The second problem shared by both of these approaches (a "to" time of either "now" or "forever") is that they implicitly contain a very strong assumption, that we term the *punctuality assumption*, about the integrity of a valid-time database. The tuples shown in Figures 1 and 2 represent the fact that Michelle is employed until the database indicates otherwise. That is, these tuples assume that ACME's database is a true, exact, up-to-date model of the world. To remain consistent with the world, the instant Michelle is fired, her unemployment must be manifested in the database. Otherwise, queries will return an incorrect result. If Michelle was fired yesterday (i.e., July 8), but the database has yet to be updated, a query requesting the current employees executed on either relation will erroneously include Michelle.

In many relations, the valid time is always earlier than the time of the transaction that stored the information, but within a well-specified, finite bound (we will shortly consider cases where valid time is later than transaction time) [Jensen & Snodgrass 1993]. Typically, when an employee is hired or fired, it is not until several hours or days after the incident that the database record of that employee is updated to indicate the new employment status. For instance, perhaps Michelle was fired on July 8, but it is not until July 11 that her tuple is actually updated to reflect her correct status. If the bound on the relationship between transaction execution time and valid time is known to be at most 3 days (all updates are made within three days), then a query that asks if Michelle was employed four or more days ago can always determine Michelle's correct employment status. Given such an assumption, one could interpret the meaning of Michelle's tuple in Figure 1 as of today (July 9), as shown in Figure 3. This effectively replaces the unrealistic punctuality assumption with a weaker yet more reasonable *bounded assumption*. This interpretation may solve the problem posed by the punctuality assumption, but has the limitation of introducing a possibly employed status (how should that be handled?), and does not address the first problem, that of

| | | *Valid time* | |
|---|---|---|---|
| *Employee* | *Status* | *(from)* | *(to)* |
| Michelle | employed | June 1 | July 6 |
| Michelle | *possibly* employed | July 7 | now |

Figure 3: Meaning of Michelle's tuple, if today is July 9 and the bound is 3 days

Michelle still being employed tomorrow.

## 2.4 The Non-predictive Update Assumption

A third problem raised by this use of "now" is that a "to" time of "now" cannot be used in predictive (future) updates. A predictive update has a "from" valid time that is later than the current transaction time, that is, the time of the transaction that recorded it [Jensen & Snodgrass 1993, Jensen & Snodgrass 1992]. Implicitly, the "to" valid time should also be later than the current transaction time. A common example is a company that records employee hires prior to when that employee begins work. Suppose that today is May 9 and the company decides to hire Michelle beginning in June. It is easy to determine the "from" valid time of Michelle's employment tuple, June 1, but what is the "to" valid time? If we enter a "to" time of "now" (as shown in Figure 1) we get a rather strange interval, one that ends before it starts! Since predictive updates using "now" could inject invalid valid-time intervals into the database, for consistency, we must assume that they cannot be stored. We call this assumption the *non-predictive update assumption*.

Some data models handle this problem by entering a "to" time of NULL, instead of "now" if the "from" time is in the future [Elmasri et al. 1990, Navathe & Ahmed 1989, Yau & Chat 1991]. The NULL timestamp cannot participate in a query and the associated tuple or attribute is ignored during query processing. The NULL value is lazily updated to "now" if the "from" time becomes earlier than the present. This solution does not, however, address the problem posed by future queries. For instance, if today is May 9 and we record that Michelle will begin work in June and continue working to "now" (or NULL in this case), a query executed today that asks who will be employed in June will not include Michelle. A similar solution is to allow the temporary storage of intervals that end before they start, but to render these intervals inert during query processing.

## 2.5 Now-relative Events

Suppose that as time advances, Michelle remains employed by ACME and that, each day, the "to" time on Michelle's employment tuple is updated to record when she worked. While this representation is faithful to our knowledge at any point in time, it is it is unrealistic to assume that the "to" time will be continuously updated as time advances. It is also unclear who should do the updating, as the database has no indication of which timestamp values are stable and which are continuously changing. What is needed is a *now-relative* event. A now-relative event is an event that occurs at a given offset from, or relative to, the present time. By using now-relative events, we can more accurately record our knowledge of Michelle's employment with ACME and replace

| | | Valid time | |
|---|---|---|---|
| *Employee* | *Status* | *(from)* | *(to)* |
| Michelle | employed | June 1 | (now −3 days) |

Figure 4: Using a now-relative event

| | | Valid time | |
|---|---|---|---|
| *Employee* | *Status* | *(from)* | *(to)* |
| Michelle | employed | June 1 | (now +20 days) |

Figure 5: Using a predictive now-relative event

the punctuality assumption with a less-restrictive bounded assumption. Now-relative events were first introduced in transaction time for vacuuming [Jensen & Mark 1990].

As an example, if the bound on the relationship between transaction time and valid time is known to be 3 days (all updates are made 3 days in the future), then Michelle's employment should extend from when she was hired to 3 days before "now" as shown in Figure 4. Here the "to" timestamp value is a rather complex event. The lower delimiter is an expression involving the construct "now" and a span, in this case, 3 days, indicating the punctuality of updates.

The operational semantics of such a timestamp value is quite interesting. Whenever a now-relative timestamp is retrieved from the database, it is replaced, only for the purpose of query processing, by a non-relative determinate event, calculated by replacing "`now`" with the result of evaluating "`present`" and subtracting (or adding) the span. If the query were performed on July 9, this now-relative event would be interpreted as the non-relative event "July 6."

Now-relative events can also be used to relax the non-predictive update assumption. We can record our knowledge on May 12 that Michelle will begin employment on June 1 as shown in Figure 5. Note that if today is July 1, this tuple indicates that Michelle was definitely employed from June 1 to July 20.

One refinement of a now-relative event, is to "round" the span value in such an event to a meaningful calendar-specific boundary by using a variable span rather than a fixed span [Soo & Snodgrass 1992B]. For a predictive update, a useful variable span would be "rest of month." When used in a now-relative event, such a span would model the situation where all hiring decisions take effect on the first day of the next month rather than just 20 days in the future. That is, "now" plus the "rest of the month" would result in the first day of the next month. Note however, that now-relative events still suffer from the first problem (that of making pessimistic assumptions about the future).

In Section 5, we demonstrate that now-relative events can be stored in the same 8-byte representation as determinate events, with the result that they impose essentially no space overhead. Section 6 demonstrates that the execution time overhead of now-relative events is not excessive.

|  |  | *Valid time* |  |
|---|---|---|---|
| *Employee* | *Status* | *(from)* | *(to)* |
| Michelle | employed | June 1 | June 13 |

Figure 6: Executing the update with `present`

## 2.6 An Aside: Now versus Present

It is important to distinguish "now" as stored in a database from the like-named constant "now" used in a query. A query to retrieve all current employees might be expressed as follows.

```
SELECT Employee
FROM Employees
WHERE from <= now AND now <= to
```

The semantics of the temporal constant `now` in this query does *not* require that the value change over time. The value is a constant; it is the time at which the query is evaluated. If we evaluate this query on July 9, then the query is equivalent to

```
SELECT Employee
FROM Employees
WHERE from <= July 9 AND July 9 <= to
```

To avoid confusing this use of "now" with the notion of a time-varying "now," we advocate using the nullary function "`present`" in queries [Soo & Snodgrass 1992B]. To further emphasize the difference between the time-varying "`now`" and the function `present`, consider the following three updates.

```
UPDATE Employees (Michelle, employed, June 1, now)

UPDATE Employees (Michelle, employed, June 1, present)

UPDATE Employees (Michelle, employed, present, now)
```

Assume that all three updates were performed on June 13. The first update will store the tuple shown in Figure 1; the second update will store the tuple shown in Figure 6; and the third, Figure 7.

## 2.7 Valid-time Indeterminacy

By using *indeterminate events*, we can more accurately record our knowledge of Michelle's employment with ACME. Recall that an indeterminate event is an event that is known to have occurred,

| | | *Valid time* | |
|---|---|---|---|
| *Employee* | *Status* | *(from)* | *(to)* |
| Michelle | employed | June 13 | now |

Figure 7: Executing the update with `present` and `now`

| | | *Valid time* | |
|---|---|---|---|
| *Employee* | *Status* | *(from)* | *(to)* |
| Michelle | employed | June 1 | July 31 ~ Aug. 30 |

Figure 8: Michelle has a fixed-term appointment, represented with valid-time indeterminacy

but exactly when is unknown [Dyreson & Snodgrass 1993A]. The time when an indeterminate event occurred is delimited by a lower and an upper bound (e.g., the event occurred sometime between July 1 and July 20). The likelihood that the event occurred at any particular time between the lower and upper bound is given by a probability distribution over the times between the upper and lower bounds. The distribution is part of the indeterminate event. In this paper we assume that every time between the lower and upper bounds is equally likely (a uniform or equiprobable distribution) unless otherwise specified, although, in general, indeterminate events can have nonuniform distributions.

Instead of using "now" as the "to" time in Michelle's tuple, we can use an indeterminate event. Which indeterminate event to use depends on our knowledge of the situation. If Michelle was hired as a limited-term employee, to work between two and three months, we could record this information as shown in Figure 8. Here two time bounds, July 31 and August 30, delimit the "to" indeterminate event. If we knew only that the term would be *at least* two months, we would use the representation shown in Figure 9. If ACME has a mandatory retirement policy, we could decrease the indeterminacy considerably, as shown in Figure 10. If we removed the guarantee that Michelle will work at least two months, we arrive at the representation shown in Figure 11.

All of these representations address the first problem with "now" that we identified, ensuring that Michelle might still be employed tomorrow. They also remove the problem of incompleteness in the non-timestamp attributes (e.g., *possibly* employed, c.f., Figure 3) and ensure that new knowledge acquired later, such as the information that Michelle left the company on July 8, is not inconsistent with currently stored information, but rather is a refinement of that information. They also address the third problem, the non-predictive update assumption. Each interval represented is a valid interval no matter when it was stored in the database. However, these representations do not address the second problem, incorporating knowledge of update punctuality. If today is July

| | | *Valid time* | |
|---|---|---|---|
| *Employee* | *Status* | *(from)* | *(to)* |
| Michelle | employed | June 1 | July 31 ~ forever |

Figure 9: Michelle's employment will be for at least two months

| | | *Valid time* | |
|---|---|---|---|
| *Employee* | *Status* | *(from)* | *(to)* |
| Michelle | employed | June 1 | July 31 ~ Jan. 1, 2028 |

Figure 10: Assuming a mandatory retirement

| | | *Valid time* | |
|---|---|---|---|
| *Employee* | *Status* | *(from)* | *(to)* |
| Michelle | employed | June 1 | June 1 ~ Jan. 1, 2028 |

Figure 11: No guaranteed minimum term

9, the state of the Michelle's tuple in the database should not be that of Figure 11, but rather that shown in Figure 12 (again, assuming at most a three day lag in recording a fact in the database). The state on July 10 is shown in Figure 13 (note how the indeterminacy in the "to" event has decreased ever so slightly); on July 11, the indeterminacy disappears as we learn of Michelle's departure. Each successive state is consistent with that preceding it, and each accurately records our current knowledge of Michelle's status.

## 2.8 Now-relative Indeterminate Events

To accurately represent our continuously changing knowledge about Michelle's employment, we need a now-relative indeterminate event, similar to the now-relative (determinate) events discussed in Section 2.5. An example now-relative indeterminate event is shown in Figure 14. Here, the lower delimiter of the "to" timestamp is an expression involving the construct "now" and a span, in this case, 3 days, indicating the punctuality of updates. Note that a now-relative indeterminate event captures both the actual and expected times associated with a fact. For instance, in the tuple given in Figure 14, Michelle's actual employment history is delimited by the lower bound on the "to" time while her expected employment is delimited by the upper bound. The lower bound expresses, on a day-to-day basis, our changing knowledge of when Michelle was employed while the upper bound expresses our expectation of how long she will remain employed.

During query processing, whenever a now-relative indeterminate timestamp is retrieved from the database, it is replaced, only for the purpose of query processing, by a non-relative indeterminate event, calculated by replacing "`now`" with the value of "`present`" and subtracting (or adding) the span, just like for now-relative determinate events. If the query were performed on July 9, the now-relative indeterminate event shown in Figure 14 would be interpreted as the non-relative in-

| | | *Valid time* | |
|---|---|---|---|
| *Employee* | *Status* | *(from)* | *(to)* |
| Michelle | employed | June 1 | July 6 ~ Jan. 1, 2028 |

Figure 12: The situation as of July 9

| | | Valid time | |
|---|---|---|---|
| *Employee* | *Status* | *(from)* | *(to)* |
| Michelle | employed | June 1 | July 7 $\sim$ Jan. 1, 2028 |

Figure 13: The situation as of July 10

| | | Valid time | |
|---|---|---|---|
| *Employee* | *Status* | *(from)* | *(to)* |
| Michelle | employed | June 1 | (now $-3$ days) $\sim$ Jan. 1, 2028 |

Figure 14: Using a now-relative indeterminate event

determinate event "July 6 $\sim$ Jan. 1, 2028." Doing this calculation during query processing ensures that continual updates are not required (in a sense, the updates are lazy and non-persistent), while capturing all of our knowledge of exactly when Michelle was employed by ACME.

There is one wrinkle to this scheme, if the now-relative indeterminate event is the "to" time of an indeterminate interval, then the lower bound on the event must lie between the upper bound on the "from" event and the upper bound on the "to" event. For instance, the "to" lower bound of Michelle's employment tuple is constrained to be sometime between June 1 and Jan. 1, 2028. If today is May 9, then the lower bound is June 1 and the tuple indicates that we *expect* Michelle to be employed from June 1 to Jan. 1, 2028. If today is Jan. 1, 2050, then the lower bound is Jan. 1., 2028 and the tuple indicates that Michelle was *actually* employed from June 1 to Jan. 1, 2028. In short, now-relative indeterminate events capture the semantics of predictive updates.

In Section 5, we demonstrate that now-relative indeterminate events can be stored in the same 8-byte representation as determinate events and non-relative indeterminate events, with the result that they impose essentially no space overhead. Section 6 demonstrates that the execution time overhead of now-relative indeterminate events is not excessive.

In summary, like now-relative determinate events, now-relative indeterminate events relax the strict punctuality assumption. But now-relative indeterminate events also support future queries and can be used in predictive updates.

## 3 Transaction Time

*Transaction time* denotes the time interval between a fact being stored in the database and the fact being (logically) deleted from the database [Snodgrass & Ahn 1985]. It is an orthogonal concept to valid time, in that it concerns the history of the database, as opposed to the history of the enterprise being modeled.

|  |  | *Transaction time* |  |
|---|---|---|---|
| *Employee* | *Status* | *(start)* | *(stop)* |
| Michelle | employed | June 1 | now |

Figure 15: Michelle's employment tuple in a transaction-time relation

## 3.1 An Example

As an example, consider the transaction-time relation shown in Figure 15. Note that the attributes here are "start" and "stop." The distinct semantics of transaction time yields a different interpretation of this relation as compared with the one shown in Figure 1. The "start" timestamp of June 1 indicates that this tuple was stored in the database on June 1, i.e., we first became aware that Michelle was an employee on that date. The value of "now" for the "stop" attribute indicates that we still believe that Michelle is employed by ACME. When we learn on July 10 that Michelle left ACME, we will logically delete this tuple by changing the "stop" timestamp to July 10.

Transaction-time relations support the timeslice operation. Transaction timeslice allows a user to query the state *of the database* at some previous point in time. For example, using timeslice, we could ask "which employees were stored in the database on June 30;" the result would include Michelle.

## 3.2 "Now" in Transaction Time

The transaction time concept that heretofore has been labeled with "now" is somewhat simpler than the valid time concept of "now." We will show that this concept is well-understood and adequately modeled by most transaction-time databases (unlike the valid-time concept of "now"), but is misleadingly called "now."

Transaction-time timestamps are supplied automatically by the DBMS during updates (valid-time timestamps are generally supplied by the user). Specifically, insertions initialize the "start" time to the value of `present` and the "stop" time to "now." (There is an additional requirement that the transaction time be consistent with the transaction serialization order.) Updates change the "stop" time of "now" to the value of `present`. Hence, in transaction-time relations, deletion is logical. The information is not physically removed from the relation, rather it is tagged as no longer current by having a "stop" time different from "now." Physical deletion never occurs in a transaction-time relation.

Requesting information as best known now involves use of the `present` clause. So one difference is that the user never has to explicitly specify `now` in a query. We saw in Section 2.6 that the situation is different with regard to valid time: `present` and `now` could even appear in the same statement.

Now has acquired a slightly differently meaning in the transaction-time dimension. In transaction time, a tuple timestamped with a "stop" transaction time of "now" means that this tuple

|  |  | *Transaction time* |  |
|---|---|---|---|
| *Employee* | *Status* | *(start)* | *(stop)* |
| Michelle | employed | June 1 | forever |

Figure 16: Using forever in a transaction-time relation

|  |  | *Transaction time* |  |
|---|---|---|---|
| *Employee* | *Status* | *(start)* | *(stop)* |
| Michelle | employed | June 1 | until changed |

Figure 17: Using until changed in a transaction-time relation

has not yet been logically deleted [Yau & Chat 1991]. But the label "now" actually obscures this meaning. Strictly speaking, it implies that every current tuple was deleted by the current transaction! In Figure 15, if `present` evaluates to July 9, then a strict interpretation of a "stop" time of "now" suggests that the "stop" time is July 9 (we used exactly the same interpretation for "now" in valid time). This is not what was intended.

## 3.3 "Forever" in Transaction Time

As with valid time, some data models address this problem by using "forever" instead of "now," as shown in Figure 16 [Ben-Zvi 1982, Bhargava & Gadia 1989, Snodgrass 1987, Thirumalai & Krishna 1988]. And as before, we immediately encounter other difficulties. The strict interpretation of this tuple is that a transaction executing a (very) long time in the future will logically delete this tuple from the relation. In the meantime, it will remain in the database. If, on July 10, it becomes known that Michelle has left ACME, then we logically delete this tuple by changing the "stop" time to July 10. Such a change is inconsistent with the previous "stop" time, thus implying that the label "forever" is not an adequate solution. In this one sense, "now" is somewhat more appropriate.

## 3.4 "Until Changed"

A more precise label for the transaction-time concept of "not yet logically deleted" is "until changed." The most recent transaction for a fact is considered the current state of that fact, *until changed* by some later transaction. Querying the current state, perhaps using the construct "`as of present`" considers all tuples with a "stop" time of "until changed." We advocate using "until changed" instead of "now" in transaction time to make clear the special, transaction-time specific meaning of now, and to ensure that updates are consistent with, and in fact a refinement of, currently stored information. For our example, this would appear as shown in Figure 17.

"Until changed" is a distinguished transaction timestamp value. It has no counterpart in valid time (using "until changed" instead of "now" avoids potential confusion with "now" in valid time, although some authors use "until changed" in valid-time [Wiederhold et al. 1991]). Also, it

| | | *Valid time* | | *Transaction time* | |
|---|---|---|---|---|---|
| *Employee* | *Status* | *(from)* | *(to)* | *(start)* | *(stop)* |
| Michelle | employed | June 1 | (now −3 days) ∼ Jan. 1, 2028 | June 2 | until changed |

Figure 18: A bitemporal relation

can only be used in the "stop" time; it is nonsensical to use it in the "start time."

### 3.5 An Aside: "Beginning" in Transaction Time

As we have just seen, `forever` is inappropriate in transaction time. No tuple can be associated with a transaction time greater than that of the most recently executed transaction. The earliest representable timestamp, `beginning`, is also inappropriate in transaction time, as no tuple can be associated with a transaction time less than that of the first transaction to be executed on the database. There is, however, a notion of "beginning" that *is* relevant to transaction time, and that is the transaction time of the first transaction. To distinguish this specific time from `beginning` (which is *much* earlier, the time of the big bang, around 14 billion years ago, in one proposal [Dyreson & Snodgrass 1993B], and 1 A.D. in others [Date & White 1990, Melton 1990]), we propose a new nullary function named `initiation` that evaluates to the time of the first transaction, that is, the time when the database schema was created.

## 4 Bitemporal Relations

A *bitemporal* relation supports both transaction and valid time [Jensen et al. 1992]. Since valid time and transaction time are orthogonal concepts, they can be combined without adverse effects.

Figure 18 shows that Michelle's employment tuple was added to the database on June 2. Note that it contains a now-relative "to" time and "until changed" as the "stop" time. To illustrate the information content of this tuple, we examine several queries (we give a formal definition of information content in Section 4.2). The following informal queries include Michelle in the answer set (we assume that today is July 9).

- As best known today, who is probably employed?
- As best known today, who was probably an employee on July 7?
- As best known today, who was definitely employed on July 1?
- As best known on July 1, who was definitely employed on June 15?
- As best known today, who will probably be employed on September 1?

The following informal queries do *not* include Michelle in the result.

| | *Valid time* | | *Transaction time* | |
|---|---|---|---|---|
| *Name* | *(from)* | *(to)* | *(start)* | *(stop)* |
| Bill Clinton | now −1 day ∼ Jan. 20, 1993 | Jan. 20, 1993 ∼ Jan. 19, 2001 | Nov. 5, 1992 | until changed |

Figure 19: A bitemporal presidential relation

- As best known today, who is definitely an employee?
- As best known today, who was definitely employed on July 7?
- As best known on July 1, who was definitely employed on July 1?
- As best known today, who will definitely be employed on September 1?

To illustrate the full generality of our proposal, we examine another scenario. Melanie downloads the AP News into her workstation each morning at 2 AM to save on telecommunications costs. Since she is very busy, she obtains all of her news from this one source, which she reads during breakfast. On November 5, 1992, the day after the U.S. presidential elections were held, she stored the tuple shown in Figure 19. The "start" time is the time the tuple was stored; the "stop" time is "until changed" because the tuple has not been logically deleted. The "from" time contains the (erroneous) assumption that if the current President, George Bush, were incapacitated, then Bill Clinton would be sworn in as President (in actuality, the Vice-President, James Danforth Quayle III, would be sworn in and would serve until the inauguration on January 20, 1993). The one day lag is for the delay of the newsfeed; something could happen and Melanie would not know about it until the next day. The "to" time incorporates the constitutional limitation of two four-year terms for a President.

## 4.1 The Transaction Timeslice Operator and "Now"

When a now-relative event is evaluated during query processing, the "now" portion of the event is replaced, for the purposes of the query, with the current time. The current time is the result of evaluating the nullary function `present`, which, in turn depends on the current state of the database. In a transaction or bitemporal database, the state of the database that is considered "current" is query specific. That is, the database can be (temporarily) changed to a previous state during a query by the transaction timeslice operator. Since the current time is part of the state of a database, transaction timeslice also selects the time for "now."

For example, suppose that today is August 8, 1993 and that we execute a transaction timeslice for July 1, 1970. Such an operation asks for the state of the database as of July 1, 1970. Our knowledge about the database is current as of July 1, 1970 rather than any earlier or later date. The timeslice changes the "current" state of the database to the state known on July 1, 1970. In this database state, "now" is July 1, 1970 not August 8, 1993. If the tuple given in Figure 4 were evaluated during this query, its "to" time (`now - 3 days`) would be June 28, 1970.

|  | *Valid time* |  | *Transaction time* |  |
|---|---|---|---|---|
| *Employee* | *(from)* | *(to)* | *(start)* | *(stop)* |
| Michelle | 1 | now | 2 | until changed |

Figure 20: A simple bitemporal tuple

|  | *Valid time* |  | *Transaction time* |  |
|---|---|---|---|---|
| *Employee* | *(from)* | *(to)* | *(start)* | *(stop)* |
| Michelle | 1 | now $\sim$ 3 | 2 | until changed |

Figure 21: A bitemporal tuple with a now-relative indeterminate event

## 4.2 A Formal Definition

The previous section informally discussed the subtle interaction between transaction timeslice and "now" in valid time. In this section we formally define the meaning of a tuple and the nature of this interaction. This definition is based on the transaction timeslice operator, $\rho^i$ where $i \in \{\texttt{initiation}, \ldots, \texttt{until changed}\}$, and complements rather than supersedes the informal description given above. If $x$ is a tuple in a bitemporal relation, then the meaning of $x$ at time $t$, denoted $S^t(x)$, is

$$\bigcup_{i \leq t} \langle \rho^i(x), i \rangle$$

Intuitively, the meaning of a tuple is a sequence of tuples indexed by transaction time. Note that the meaning of a tuple depends on the time chosen to examine that tuple. For example, if $x$ is the tuple given in Figure 20, then (assuming $\texttt{initiation} = 1$)

$$S^1(x) = \{\langle \emptyset, 1 \rangle\}$$

$$S^2(x) = \{\langle \emptyset, 1 \rangle, \langle (\texttt{Michelle}, [1, 2]), 2 \rangle\}$$

$$S^3(x) = \{\langle \emptyset, 1 \rangle, \langle (\texttt{Michelle}, [1, 2]), 2 \rangle, \langle (\texttt{Michelle}, [1, 3]), 3 \rangle\}$$

The meaning of now-relative indeterminate events can be given similarly, for instance, the meaning of the tuple shown in Figure 21 is

$$S^1(x) = \{\langle \emptyset, 1 \rangle\}$$

$$S^2(x) = \{\langle \emptyset, 1 \rangle, \langle (\texttt{Michelle}, [1, 2 \sim 3]), 2 \rangle\}$$

$$S^3(x) = \{\langle \emptyset, 1 \rangle, \langle (\texttt{Michelle}, [1, 2 \sim 3]), 2 \rangle, \langle (\texttt{Michelle}, [1, 3]), 3 \rangle\}$$

$$S^4(x) = \{\langle \emptyset, 1 \rangle, \langle (\texttt{Michelle}, [1, 2 \sim 3]), 2 \rangle, \langle (\texttt{Michelle}, [1, 3]), 3 \rangle \langle (\texttt{Michelle}, [1, 3]), 4 \rangle\}$$

## 5 Timestamp Representation

This paper has proposed four new timestamps that must be represented: the distinguished transaction timestamps "until changed" and "initiation," and now-relative determinate and indeterminate events. We have proposed timestamp formats for events, intervals and spans elsewhere [Dyreson & Snodgrass 1993B] (the indeterminate formats are described in a different document [Dyreson & Snodgrass 1992]). In this section we extend these formats to represent the new timestamp values. We extend only the event formats; intervals are represented as a pair of their bounding events.

We first describe extensions to handle "until changed", "initiation," and now-relative determinate events and only later consider now-relative indeterminate events. There are three *resolutions* for events: high, low, and extended. Resolution is a rough measure of timestamp precision and range. The low resolution format can represent times over a range of 36 billion years to the precision of a second. High resolution increases the precision to a microsecond while narrowing the range to 17,400 years. Both the high and low resolution timestamp formats are two 32-bit words in size. Extended resolution is even more precise; it can represent a range of 36 billion years to the precision of a nanosecond, but has higher space requirements, generally an additional four bytes.

Each representation has a 4-bit type field to distinguish the resolution. To date, thirteen event types have been allocated: three determinate event formats, one at each resolution; nine indeterminate formats, three for each resolution; and one special format type, to support distinguished values such as "beginning" and "forever." In some sense, "until changed," "initiation," and now-relative determinate events are "special" events, hence we need only add to this format, shown in Figure 22, while leaving the other formats unchanged. For "until changed" and "initiation," the extension is straightforward. We merely add each as a new special event value. The addition of "until changed" and "initiation" raises the total number of special events to six. Since we anticipate that few other special events will be added, we can exploit the unused bits in the special event format to add now-relative determinate events.

Now-relative determinate events are of the form "now $\pm$ *span*." An example is "now $-1$ day." We use a bit to distinguish now-relative from other special events. The span associated with the event is stored the remaining bits. A second bit is used to distinguish between fixed and variable spans. The span value will usually be but a few days, but might be as long as a few weeks, or even a year. All of these spans can easily be represented by the second format (for fixed spans) and the third format (for variable spans) shown in Figure 22. In fact the fixed span format can represent a span value over a range of 8,700 years to the precision of a microsecond (in other words, it is almost the same as the high resolution span format, only one bit, used to indicate a fixed or variable span, has been removed from the seconds from the origin field). As an example, the now-relative event `|now - 3 days|` has the hexadecimal value `0A00002A30000000`.

A now-relative *indeterminate* event is quite a bit more complex. It is of the form `|now` $\pm$ *span* $\sim$ *upper-bound*`|`. One example, from Figure 19, is `|now - 1 day` $\sim$ `Jan. 20, 1993|`. Surprisingly, it is possible to fit all of this information in the same space that a determinate event requires, 8 bytes in the common case. We employ a variation of a *chunked* indeterminate format that has the same format, but a different interpretation. A chunked indeterminate event format encodes the upper bound on an event's period of indeterminacy implicitly. The implicit encoding

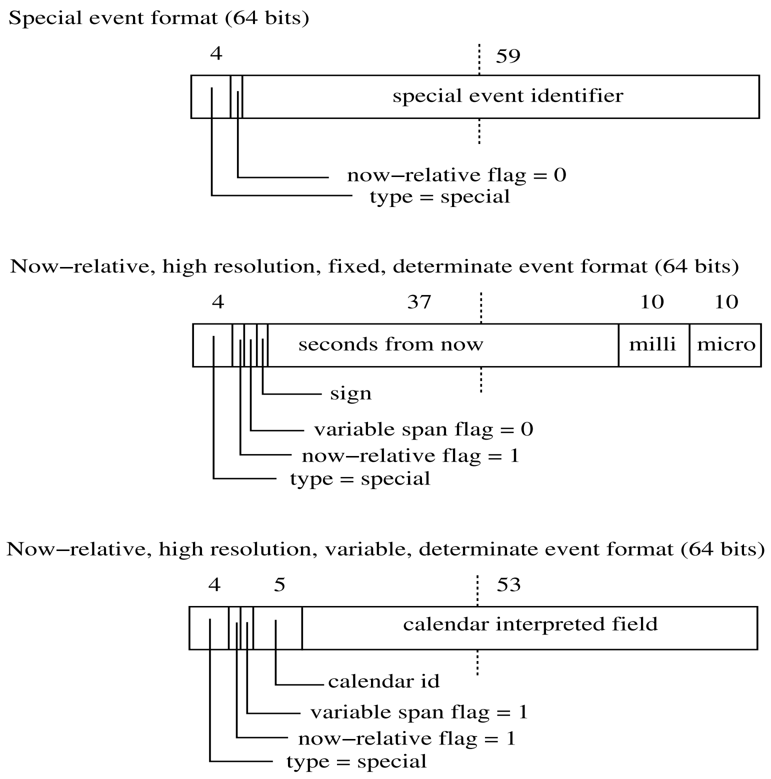
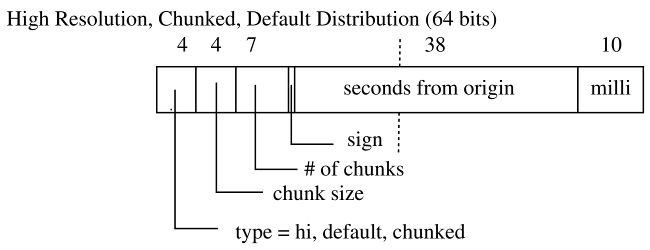


Figure 22: The special event and now-relative determinate event formats

is composed of a *chunk size* and a number of *chunks*. A chunk is a fixed duration (a span). The length of the duration is given by the chunk size. The terminating time is computed by adding the number of chunks, each of size *chunk size*, to the lower bound (encoded in the other bits). For example, to represent a period of 7 hours using chunks, the timestamp would record that there are seven chunks with a chunk size of an hour. The chunk sizes range from 10 microseconds to 1 millennium, spread out so that any possible span can be approximately represented (to within a precision of at worst less than 10%, and generally less than 1%).

Both the original format and the now-relative version are shown in Figure 23 (only for the high resolution format; the other resolutions are similar). A now-relative indeterminate event has a different type field since there was no other way to distinguish between the two formats. The sign bit is not needed for the upper bound on the period of indeterminacy, since we are assured that this time will be after January 1, 1970, which is the origin for this representation (we assume that the sign is positive). The bit fields devoted to the chunking information are used to record the span associated with the now-relative indeterminate event, while the remaining fields encode the upper bound on the event's period of indeterminacy. For example, suppose that the event to be encoded is `|now - 3 days ~ Jan. 20, 1993|` (the "to" valid time in Figure 18). The upper bound is 7274880000 seconds from the origin, while three days corresponds to three chunks of a chunk-size one day (chunk-size #7 for the high resolution format [Dyreson & Snodgrass 1992]). The timestamp value for this now-relative event is (in hexadecimal) `B707002B5C960000`.


High Resolution, Chunked, Default Distribution (64 bits)
4
4
7
38
10
seconds from origin
milli
sign
# of chunks
chunk size
type = hi, default, chunked

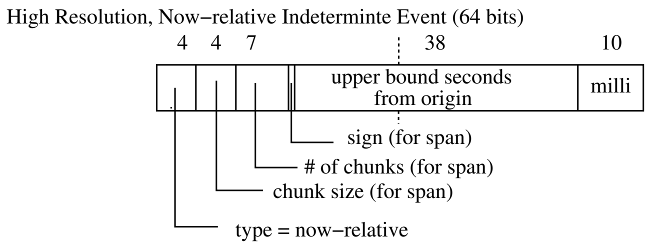

High Resolution, Now−relative Indeterminte Event (64 bits)
4
4
7
38
10
upper bound seconds from origin
milli
sign (for span)
# of chunks (for span)
chunk size (for span)
type = now−relative

Figure 23: The compact (high resolution) now-relative indeterminate format

If the chunking yields an inadequate precision, or the user wishes to associate a distribution other than that specified as the default by the data base administrator, then more storage will be required. In this case, we use variants of other non-relative indeterminate as shown in Figure 24. The first format adds a non-default distribution (the default is specified by the database administrator), while the second explicitly encodes the associated span. The variants are distinguished by setting the now-relative flag.

As a point of comparison, the SQL2 datetime format, which has a range of only 10,000 years and a precision of only one second, and which does not incorporate either indeterminacy or now-relativity, requires 20 bytes [Melton 1990]. For most users, our 8-byte format should suffice.

## 6 Timestamp Operations

Only now-relative events impact query processing costs; "until changed" is a special event that is processed exactly the same as other special events. The most common operation during query processing is event comparison [Dyreson & Snodgrass 1993B]. Comparison determines if one event is earlier than another in the temporal ordering.

Now-relative events make comparison more costly. Consider the comparison of a now-relative determinate event and a non-relative determinate event. To compare a now-relative event, the span associated with the event must be added or subtracted from the present time (in the worst case). This adds one extra addition or subtraction to the comparison. Indeterminate event comparisons are, in general, much more complex, and an extra addition or subtraction has little impact on the overall cost of the operation [Dyreson & Snodgrass 1992].

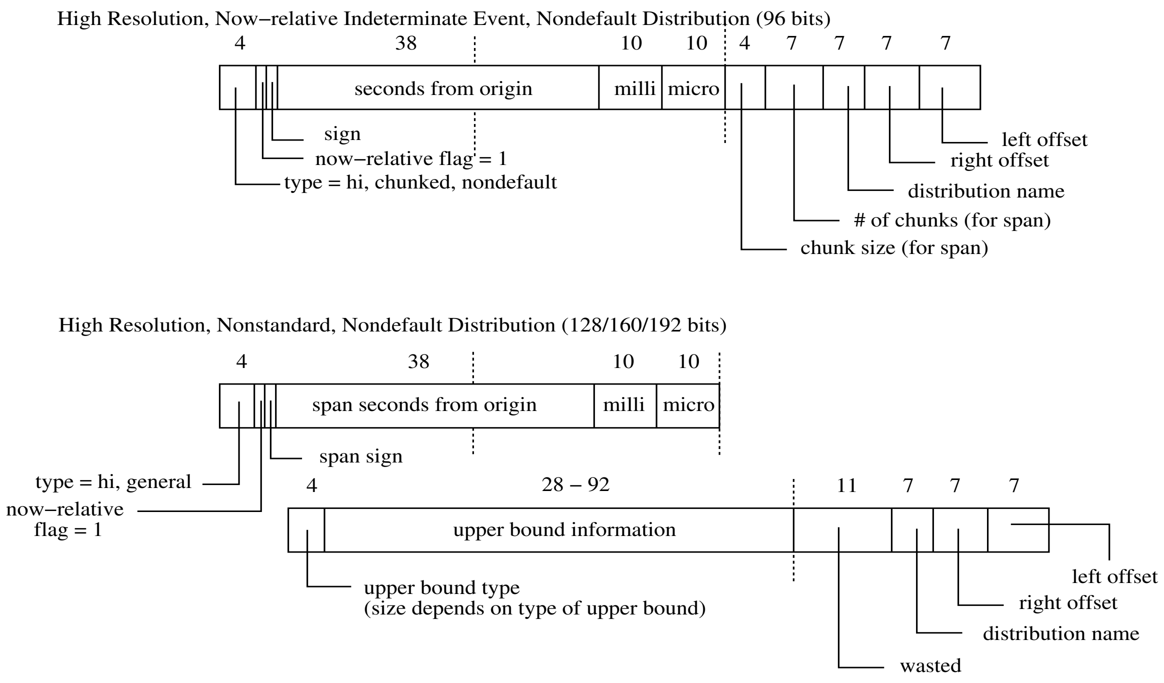


Figure 24: The other (high resolution) now-relative indeterminate format

Three factors further mitigate the cost for determinate now-relative comparisons (the first consideration also applies to indeterminate now-relative comparisons). First, the addition or subtraction must only be performed once per query, and the result can be cached. Subsequent comparisons or other operations on the same now-relative event can use the more efficient, computed non-relative form. More than one comparison is in fact common, for instance, an event participates in at least two comparisons during a single overlap. An event may also be reused in the cartesian product of two (or more) relations. A now-relative event in the outer relation can be computed and cached when iterating over the inner relation. Second, the addition or subtraction can be avoided in many instances. If the non-relative event precedes "now" and the span associated with the now-relative event is positive, then the comparison can be made without adding the span. Third, comparing two now-relative events is as efficient as comparing two non-relative events. Only the associated spans need be compared with no additions or subtractions needed.

Other operations on now-relative events are less costly. Now-relative events at worst double the cost of arithmetic operations, and increase the cost of output even less. The cost of output is typically far greater than addition or subtraction and if the events are output in their now-relative form, then no addition or subtraction is even required during output. For input, now-relative events impose no special costs since the event does not have to be computed, just the string describing the event must be translated in a fashion similar to other events [Soo & Snodgrass 1992A].

## 7 Conclusion

In temporal databases "now" is a commonly used value in both transaction and valid time. However, in valid time, the concept of "now" is inadequately modeled while in transaction time it is misleadingly labeled.

In valid time, now is typically handled by simply replacing it with the present time during query evaluation. But this simple strategy has shortcomings: future queries may return inconsistent answers, the database must assume punctual updates, and special processing strategies are required to support predictive updates. Using indeterminate and now-relative events can provide a far richer semantics for "now." Now-relative events relax the strict punctuality assumption, replacing it with a more reasonable bounded assumption while indeterminate events naturally support the uncertainty in future queries and allow for predictive updates. Both of these positive aspects are combined in now-relative indeterminate events.

In transaction time, "now" has been associated with events that "have yet to be logically deleted" from the database. The label "now" is misleading because it implies that all transactions are logically deleted now and because it can be confused with the different notion of "now" in valid time. To address these issues, we advocate using "until-changed" instead of "now" in transaction time.

The temporal types and values proposed in this paper add to those previously proposed as discussed in the introduction. Below, we list each type or value that we introduced in this paper:

- *initiation* (the earliest time in transaction time),
- *present* (a function used in a query that returns the current time),
- *until changed* (the current time in transaction time),
- *now-relative determinate events* (a specialized value that captures the subtle interaction between the current valid time and the punctuality of updates), and
- *now-relative indeterminate events* (a generalization of now-relative determinate events that captures both actual and expected times).

We showed that the new kinds of events needed to combat the semantic difficulties associated with "now" have a compact representation, just 2 or 3 words in most cases, with virtually no space overhead compared with other timestamp representations. Timestamp operation efficiency remains high, even for these complex events.

## 8 Acknowledgements

Several discussions with James Clifford helped in understanding the semantics of "now" in both valid and transaction time.

## 9 References

[Ariav et al. 1984] Ariav, G., A. Beller and H.L. Morgan. "A Temporal Data Model." Technical Report DS-WP 82-12-05. Decision Sciences Department, University of Pennsylvania. Dec. 1984.

[Bassiouni & Llewellyn 1992] Bassiouni, M.A. and M.J. Llewellyn. "A Relational-Calculus Query Language For Historical Databases." *Computer Languages*, 17, No. 3 (1992), pp. 185–197.

[Ben-Zvi 1982] Ben-Zvi, J. "The Time Relational Model." PhD. Dissertation. Computer Science Department, UCLA, 1982.

[Bhargava & Gadia 1989] Bhargava, G. and S. Gadia. "Achieving Zero Information Loss in a Classical Database Environment," in *Proceedings of the Conference on Very Large Databases.* Amsterdam: Aug. 1989, pp. 217–224.

[Browning 1899] Browning, R. "The complete works of Robert Browning." Boston, MA: Houghton Mifflin, 1899.

[Date & White 1990] Date, C. J. and C. J. White. "A Guide to DB2." Reading, MA: Addison-Wesley, 1990. Vol. 1, 3rd edition.

[Dyreson & Snodgrass 1992] Dyreson, C. E. and R. T. Snodgrass. "Time-stamp Semantics and Representation." TempIS Technical Report 33. Computer Science Department, University of Arizona. Feb. 1992.

[Dyreson & Snodgrass 1993A] Dyreson, C. E. and R. T. Snodgrass. "Valid-Time Indeterminacy," to appear in *Proceedings of the International Conference on Data Engineering.* Vienna, Austria: Apr. 1993.

[Dyreson & Snodgrass 1993B] Dyreson, C. E. and R. T. Snodgrass. "Timestamp Semantics and Representation, to appear." *Information Systems*, 18, No. 3, Sep. 1993.

[Elmasri et al. 1990] Elmasri, R., G. Wuu and Y. Kim. "The Time Index - An Access Structure for Temporal Data," in *Proceedings of the Conference on Very Large Databases.* Brisbane, Australia: Aug. 1990.

[Gadia 1988] Gadia, S.K. "A Homogeneous Relational Model and Query Languages for Temporal Databases." *ACM Transactions on Database Systems*, 13, No. 4, Dec. 1988, pp. 418–448.

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

[Jensen et al. 1992] Jensen, C.S., J. Clifford, S.K. Gadia, A. Segev and R.T. Snodgrass. "A Glossary of Temporal Database Concepts." *SIGMOD Record*, 21, No. 3, Sep. 1992.

[Jensen & Snodgrass 1992] Jensen, C.S. and R. Snodgrass. "Temporal Specialization," in *Proceedings of the International Conference on Data Engineering.* Ed. F. Golshani. IEEE. Tempe, AZ: Feb. 1992, pp. 594–603.

[Jensen & Snodgrass 1993] Jensen, C. S. and R. Snodgrass. "Temporal Specialization and Generalization, to appear." *IEEE Transactions on Knowledge and Data Engineering*, (1993).

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

[Navathe & Ahmed 1989] Navathe, S. B. and R. Ahmed. "A Temporal Relational Model and a Query Language." *Information Sciences*, 49 (1989), pp. 147–175.

[Sarda 1990] Sarda, N. "Algebra and Query Language for a Historical Data Model." *The Computer Journal*, 33, No. 1, Feb. 1990, pp. 11–18.

[Snodgrass & Ahn 1985] Snodgrass, R. and I. Ahn. "A Taxonomy of Time in Databases," in *Proceedings of ACM SIGMOD International Conference on Management of Data.* Ed. S. Navathe. Association for Computing Machinery. Austin, TX: May 1985, pp. 236–246.

[Snodgrass 1987] Snodgrass, R. T. "The Temporal Query Language TQuel." *ACM Transactions on Database Systems*, 12, No. 2, June 1987, pp. 247–298.

[Snodgrass 1993] Snodgrass, R. "An Overview of TQuel," in Temporal Databases: Theory, Design, and Implementation. Benjamin/Cummings, 1993. Chap. 6.

[Soo & Snodgrass 1992A] Soo, M. and R. Snodgrass. "Multiple Calendar Support for Conventional Database Management Systems." Technical Report 92-7. Computer Science Department, University of Arizona. Feb. 1992.

[Soo & Snodgrass 1992B] Soo, M. and R. Snodgrass. "Mixed Calendar Query Language Support for Temporal Constants." TempIS Technical Report 29. Computer Science Department, University of Arizona. Revised May 1992.

[Sykes 1964] "The Concise Oxford Dictionary." Oxford, England: Oxford University Press, 1964.

[Tansel 1990] Tansel, A U "Modelling temporal data." *Information and Software Technology*, 32, No. 8, Oct. 1990, pp. 514–520.

[Thirumalai & Krishna 1988] Thirumalai, S. and S. Krishna. "Data Organization for Temporal Databases." Technical Report. Raman Research Institute, India. 1988.

[Wiederhold et al. 1991] Wiederhold, G., S. Jajodia and W. Litwin. "Dealing with Granularity of Time in Temporal Databases," in *Proc. 3rd Nordic Conf. on Advanced Information Systems Engineering.* Trondheim, Norway: May 1991.

[Yau & Chat 1991] Yau, C. and G. S. W. Chat. "TempSQL – A Language Interface to a Temporal Relational Model." *Information Sc. & Tech.*, , Oct. 1991, pp. 44–60.