On the Semantics of "Now" in Temporal Databases

James Clifford*, Curtis Dyreson, Tomás Isakowitz, Christian S. Jensen
and Richard T. Snodgrass

Abstract

Most databases record time-varying data, and significant efforts have been devoted to the convenient and efficient management of such data. Perhaps most prominently, numerous data models with varying degrees of built-in support for the temporal dimension of data have been proposed. Some models are quite restricted and simply support uninterpreted attribute domains for times and dates. Other models incorporate either a valid-time dimension, recording when the stored data is true, or a transaction-time dimension, recording when the stored data is current in the database. Bitemporal data models incorporate both valid and transaction time. The special temporal notion of an ever-increasing current-time value has been reflected in some of these data models by inclusion of current-time variables, such as "*now*," "*until-changed*," "∞," "@" and "–." As timestamp values associated with facts in temporal databases, such variables may be conveniently used for indicating that a fact is currently valid. Although the services of time variables are very desirable, their use leads to a new type of database, consisting of tuples with variables, termed *variable databases*.

This paper proposes a framework for defining the semantics of the variable databases of temporal relational data models. A framework is presented because several reasonable meanings may be given to databases that use some of the specific temporal variables that have appeared in the literature. Using the framework, the paper defines a useful semantics for such databases. Because situations occur where the existing time variables are inadequate, two new types of modeling entities that address these shortcomings, timestamps which we call *now-relative* and *now-relative indeterminate*, are introduced and defined within the framework. Moreover, the paper provides a foundation, using algebraic *bind* operators, for the querying of variable databases via existing query languages. This transition to variable databases presented here requires minimal change to the query processor. Finally, to underline the practical feasibility of variable databases, we show that variables may be represented and manipulated efficiently, incurring little space or execution time overhead.

1 Introduction

Now is a noun in the English language that means "at the present time" [Syk64]. A variable with this name has also been used extensively in temporal relational data model proposals, primarily as a timestamp value associated with tuples or attribute values in temporal relations. Yet, the precise semantics of databases with this and other current-time variables have never been fully specified. An important goal of this paper is to give a clear semantics for databases with current-time variables.

Time variables such as *now* are of interest and indeed are quite useful in databases that record time-varying information, the validity of which often depends on the current-time value. Such

*J. Clifford and T. Isakowitz are with the Information Systems Department, Leonard N. Stern School of Business, New York University, 44 W. 4th Street, New York, NY 10012, USA, {jclifford|tisakowi}@stern.nyu.edu. C. Dyreson is with the Department of Computer Science, James Cook University, Townsville, Queensland Q4811, Australia, dyreson@jcu.edu.au. C. S. Jensen is with the Department of Mathematics and Computer Science, Aalborg University, Fr. Bajers Vej 7E, Dk–9220 Aalborg Øst, Denmark, csj@iesd.auc.dk. R. T. Snodgrass is with the Department of Computer Science, University of Arizona, Tucson, AZ 85721, USA, rts@cs.arizona.edu.

databases may be found in many application areas, such as banking, inventory management, and medical and personnel records. For example, in a banking application, it is necessary to record when account balances for customers are valid. Specifically, if a customer opens an account and deposits US$ 200 on January 15 (in some year), the validity of that balance starts when the deposit is made and extends until the current time, assuming no update transactions are committed. Thus, on January 16, the balance is valid from January 15 until January 16; on January 17, the balance is valid from January 15 until January 17, etc. It is impractical to update the database each day (or millisecond) to correctly reflect the valid time of the balance. Rather, it has proven convenient to use a variable, such as *now*, for indicating that the time when a balance is valid depends on the current time. In the example, it would be recorded on January 15 that the customer's balance of US$ 200 is valid from January 15 through *now*.

There have been numerous temporal database models proposed in the literature [Kli93]. Most of these proposals have incorporated one or both of two temporal dimensions that have been identified as important in the data modeling of temporal information and which are given special semantics by the query language. The *valid time* of a database fact is the time when the fact is true in the modeled reality, while the *transaction time* indicates when the fact is current in the database [JCE+94]. The majority of temporal models have incorporated only the valid time dimension, and are termed *valid-time* data models. A far fewer number, the so-called *transaction-time* data models, have incorporated only the transaction-time dimension. A slightly larger number than this, the *bitemporal* data models, have incorporated both dimensions. Time variables are equally relevant to both time dimensions, but we shall show that the appropriate meaning of a tuple with a time variable in its timestamp is dependent on the specific time dimension(s) supported by the database and to which the variable belongs.

In examining the large body of existing temporal data models, it is apparent that two different *types* of models have been proposed. The first type of model essentially accords with the view expressed by Reiter that a relational database can be seen as a set of ground first-order formulæ, for which there is a minimal model [Rei84]. These models have either been presented as logical models directly (e.g., [CW83],[CCT93]), or have been presented in such a way that their logical model was clear.

The second type of model deviates from this tradition. Rather, these models have been presented as a set of formulæ some of which are ground, but others of which have included one or more free, current-time variables. Chief among these current-time variables is "*now*" (e.g., [CC87, Gad88, CT85]), but a variety of other symbols have been used, including "–" [BZ82], "∞" [Sno87], "@" [LJ88], and "*until-changed*" [WJL91, WJL93]. As already mentioned and exemplified, the use of such variables is quite convenient and practical. Thus, these approaches have advantages at the implementation level, namely, they are space efficient and avoid the need for updates at every moment in time. However, nowhere have we found a clear exposition of temporal variables, i.e., nowhere has the semantics of this type of database—a database with current-time variables, here termed *variable databases*—been formally specified so that the logical model represented by the database is clear. Rather, the models have relied on the choice of intuitive names for the variables to convey their meaning. This has led many to suppose that they understood their semantics. However, this reliance on intuition and lack of a clear semantics for databases with current-time variables is unsatisfactory as a foundation for the development and implementation of variable databases, as it is prone to ambiguities and misinterpretations and, therefore, to errors.

In this paper, we present a framework for the specification of the different semantics that may be given to variable databases, which builds on the approach introduced in [CI94]. In the framework, the semantics of a variable database is defined by means of what we term an *extensionalization mapping* from a variable database to a fully ground data model. The actual extensionalization

mappings for valid-time, transaction-time and bitemporal databases with one or more current-time variables are given in subsequent sections. This illustrates that the framework is general enough to allow for the specification of a wide variety of semantics, an important property of a framework. It also illustrates that the framework can capture the semantics of multidimensional databases in a straightforward manner: the multidimensional extensionalization mapping is obtained by a simple, but coordinated, combination of the mappings for the constituent one-dimensional databases.

We also observe that the modeling capabilities of current-time variables are limited. To overcome these limitations, two new modeling entities, *now-relative* and *now-relative indeterminate* timestamps are introduced and defined within the framework. Next, a mechanism for the querying of variable databases using existing query languages is provided. This mechanism provides added functionality, does not require changes to a query language, and is easily integrated into a query processor. It is observed that the incorporation of the notion of perspective into query languages may provide additional functionality when querying variable databases. Finally, to underline the practicality of a variable database, compact physical representations for timestamps involving current-time variables are provided. These formats can be efficiently manipulated during query processing.

In the remainder of this paper we address the issues touched upon in this introduction. To motivate the need for additional current-time-related modeling entities and a framework for expressing the semantics of these and of existing current-time variables, the next section addresses shortcomings of existing current-time variables and informally explores the semantics of databases using such variables. Section 3 then presents the framework for specifying the semantics of variable databases. Within this framework, the semantics of variable databases is given by an extensionalization mapping from a variable database to instances of a fully ground model, i.e., a model without variables. Section 4 considers the use and semantics of new now-relative and now-relative indeterminate timestamps, in addition to the existing variable *now*, in the valid-time dimension. The section illustrates in depth how the semantic framework is used. Further, it is shown how variable valid-time databases may be queried with existing query languages, with minimal changes to their implementation. Finally, the novel use of perspective in querying a variable database is explored through examples. Section 5 provides a parallel treatment of variable transaction-time databases. Specifically, the different semantics of transaction-time gives the current-time variable in transaction time a different semantics than its valid-time counterpart. Section 6 then considers *bitemporal* variable databases, which support both time dimensions. It is shown how the semantics of variable bitemporal databases is obtained by combining the semantics of variable valid-time and transaction-time databases. Section 7 extends an existing timestamp representation scheme for ground timestamps to provide space-efficient timestamp formats for current-time-related timestamps. We show that the new timestamps have little impact on the efficiency of query evaluation. Finally, in Section 8, the paper's contributions are summarized and suggestions for future research are offered.

2 Motivation

To motivate the need for current-time variables in temporal databases, including a solid, formal foundation for their use, this section introduces the use of such variables and explores some of the perhaps unintuitive, semantic subtleties resulting from their incorporation. Further, this section explores the limitations of current-time variables in some realistic situations.

As the meaning of current-time variables depends on whether the context is valid time or transaction time, current-time variables in valid-time and transaction-time databases are considered

in isolation, followed by a short discussion of current-time variables in bitemporal databases.

2.1 Variables in Valid-time Databases

In this section, we first exemplify why it is convenient to use the current-time variable *now* as a timestamp value in a valid-time database. We then explore three current-time-related situations that illustrate shortcomings of a single variable *now* and thus indicate a need for additional current-time modeling entities, which we introduce in Section 4. Finally, we informally discuss how the variable *now* complicates query semantics.

2.1.1 The Use of "now" in Valid-Time Databases

The *valid time* of a fact denotes the time(s) when the fact is true in reality [JCE+94, SA85]. In the valid-time dimension, a timestamp involving *now* is commonly used to indicate that a fact is currently valid [ABM84, BL92, EWK90, Gad88, NA89, Sar90, Tan90, YC91]. In conventional databases, facts that are currently valid are the only ones that are directly supported by the data model.

FACULTY			
		VALID TIME	
NAME	RANK	(from)	(to)
Jane	*Assistant*	*June 1*	*now*

Figure 1: Jane's employment tuple

For example, suppose that a database records that Jane began working as an Assistant Professor on the faculty of "State University" on June 1 (in some particular year, e.g., 1994; which year is not relevant here). Figure 1 shows the relevant tuple from the University's employment database (the **FACULTY** valid-time relation). Jane started working as an Assistant Professor on June 1, as indicated by the "from" attribute. The value *now*, appearing as the "to" time in Jane's employment tuple, represents the (later) time when Jane will stop working for State University. Together, the "to" and "from" attributes encode the valid time associated with the tuple. For simplicity, we assume a timestamp granularity of one day in all examples.

The informal meaning of this tuple is that Jane is a faculty member from June 1 until the current time. Thus, the result of a query that requests the current faculty members will include Jane. 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*" [WJL91, WJL93] or "@" [LJ88] instead of "*now*," but the meaning is the same.

Using the variable *now* in a timestamp is very convenient. To see why, suppose that instead of using the variable as the "to" time, we use a ground time, i.e., a particular date. We start by recording a "to" time of June 1. Then as time advances and Jane remains an Assistant Professor, the "to" time on Jane's tuple must be updated each day to record when she worked. Hence, the "to" time would be updated to June 2, then to June 3, etc. While this representation is faithful to our knowledge at any point in time, having to continuously update the "to" time as time advances is impractical. It is also unclear who should do the updating, as the database has no indication of which tuples have a continuously increasing valid time and which are stable. For these reasons, it is better to use the variable *now*.

In summary, the current-time variable *now* provides a convenient mechanism for allowing the temporal dimension of data in the database to evolve without the need for database updates at each successive tick of the clock.

2.1.2 The Pessimistic Assumption

While using *now* is convenient, using it as the "to" time of a tuple may lead to an overly pessimistic assumption about the modeled reality. The university application introduced in the previous section provides such a situation. Specifically, it is reasonable to expect that if an employee is employed in a certain position today, that employee will also be employed in that position tomorrow (and the next few, following days). Consider, however, the **FACULTY** relation given in Figure 1. It specifically records that Jane will *not* be employed tomorrow. Assume, for the purpose of this discussion, that today is July 9. Then a query asking who will be employed tomorrow (i.e., July 10) will not have Jane in the answer, since the "to" time of Jane's tuple is *now*, or in this case, July 9.

Using a "to" time of *now* leads to an overly pessimistic assumption about Jane's employment. Some temporal data models avoid this problem by limiting valid time to the past, that is, to times before *now* [Gad88, Tan90]. For many applications, e.g., the university application, this limitation is much too restrictive.

Other data models have advocated using one of the special (non-variable) valid-time values, such as *forever*, ∞, or "–" [Sno87, Sno93, BZ82, TK88]) instead of *now*. These symbols (we will use *forever*) denote the largest representable timestamp value, that is, the one furthest in the future. In SQL and in IBM's DB2, forever is about 8,000 years from the present [MS93, DW90]; in our more liberal proposal, it is approximately 18 *billion* years from the present time [DS93a].

FACULTY			
		VALID TIME	
NAME	RANK	(from)	(to)
Jane	*Assistant*	*June 1*	*forever*

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

By using a "to" time of *forever*, as in Figure 2, we certainly avoid the pessimistic assumption, But, we are now being overly optimistic. We have indicated that Jane will be employed as an Assistant Professor not only tomorrow, but *forever*. To assert that Jane will be employed as an Assistant Professor until forever is most assuredly incorrect (others have also noted that a "to" time of ∞, or *forever*, has erroneous implications for the future [NA89]). Another indication that *forever* is inappropriate is that when Jane departs the University, *forever* must be replaced by the date of her departure; but the revised date will be a separate and much earlier time that is inconsistent with *forever*. Rather than having the new information *refine* the old information, the new information contradicts the old information.

Perhaps, instead of *forever*, we might choose some large, application-dependent time value earlier than *forever*. In the university application, the mandatory retirement date is a good choice. While such a date is better than the generic *forever*, it is still overly optimistic.

In conclusion we have shown a practical situation where *now* (and *forever*) falls short of meeting the requirements for recording the valid time of tuples. In Section 4.4, we introduce a new type of timestamp that meets these requirements.

2.1.3 The Punctuality Assumption

The use of the current-time variable *now* in timestamps implies a strong assumption about the punctuality of updates. For example, the tuple in Figure 1 states that Jane will remain an Assistant Professor until the current time. The correctness of this tuple is dependent on the correctness of the assumption that updates are made ahead of time, i.e., predictively. Thus, changes in Jane's employment status and rank are assumed to conform with the punctuality assumption: "changes are recorded in the database no later than the instant they take effect." Specifically, if Jane is promoted to Associate Professor, to start on July 8, it is assumed to be recorded no later than that day. This amounts to assuming that State University's database is a correct, exact, up-to-date model of the modeled mini-world.

The punctuality assumption for valid-time databases is often not satisfied. In many cases information is recorded after the time it became valid, but with a well-specified maximum delay [JS94]. For example, when employees change status, it may be that the database is guaranteed to be updated to reflect this at most three days after the status is changed. If Jane was promoted on July 8, perhaps it is then not until July 10 that her tuple is actually updated to reflect her correct status. With a maximum delay of three days, the database is known to correctly describe the mini-world only in the past, up until three days ago. Within the last three days, it can only be concluded that it is likely, or possible, that Jane is employed as an Assistant Professor. In this case, one could interpret the meaning of Jane's tuple in Figure 1 as of today (July 9) as shown in Figure 3.

FACULTY			
		VALID TIME	
NAME	RANK	(from)	(to)
Jane	*Assistant*	*June 1*	*July 6*
Jane possibly employed as an Assistant		*July 6*	*now*

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

Figure 3 intuitively illustrates the "possible" type of information that we would like to be able to record because it more accurately describes our knowledge of the mini-world. This cannot conveniently be recorded using *now*. Later in this paper, in Sections 4.2 through 4.4, we describe a new kind of timestamp that can be used to address these issues.

2.1.4 The Problem of *Now* in Predictive Updates

This section explores another problem with the variable *now* as a "to" time in a tuple, namely where predictive updates lead to "from" times that are after the current time. In such situations, the "to" time, i.e., the current value of *now*, is before the "from" time, contradicting the intuition that the "from" should always be before the "to" time.

To illustrate this use of *now*, assume that State University sometimes records faculty hires prior to when that faculty member begins work. Further, assume that the tuple in Figure 1 was inserted on May 25. Then, during the remainder of May, the "to" time is before the "from" time.

Some data models do not allow the use of *now* as a "to" time when its value is before the "from" time. Instead a special "to" time value of NULL is used in such situations [EWK90, NA89, YC91]. This value is replaced by *now* when the value of *now* exceeds the "from" time. Tuples with NULL's

are ignored in queries. However, there is a subtle difficulty with this solution. Suppose that today is May 25 and we record that Jane will be an Assistant Professor from June 1 until *now* (or NULL in this case). We then execute a query that determines who will be employed in June barring any changes to the database between now and June. To evaluate this query, we temporarily "observe" the database from the perspective of a user in June even though today is June 1. The result should include Jane; however, Jane's tuple is ignored since it has a "to" time of NULL.

To summarize, we have provided several illustrations of why the use of the valid-time variable *now* in timestamps, while intuitively appealing, has certain shortcomings. In Section 4 we will introduce new modeling entities that address these shortcomings, and we will provide a formal semantics for variable valid-time databases.

2.2 Queries and *Now*

The variable *now* in a timestamp conveniently captures the evolution of current-time-related data. When querying that data, the current time must be clearly specified since the value of the variable *now* depends on the current time.

To illustrate the kind of ambiguity that can result from unclear specification of the current time assume that today is July 9 and that our database is given as in Figure 1. Then, consider the query, "Will we agree on July 13 that Jane was employed on July 11?" Since Jane is employed until *now*, the query cannot be answered unless we know what time it is *now*. Suppose that *now* is interpreted to refer to the time at which the query is asked, in this case July 9. Then Jane will not be employed on July 11 and so we would answer "no." But *now* could be interpreted as the time mentioned in the query about which we were asked to agree, in this case July 13. Then Jane will be employed on July 11 and so we would answer "yes."

Another source of ambiguity is that the constant evolution of the current-time variable *now* appears to cause the "same" query to return different results when evaluated at different times, even if no updates have occurred. For instance, consider the query, "Is Jane employed on July 11?" This simple query asked on July 10 will yield one answer ("no"), but if we ask the query on July 12 we will receive a quite different answer ("yes").

In summary, the querying of variable databases introduces new semantic subtleties, not found when querying non-variable databases, thus motivating a clear and precise definition of the meaning of variable databases.

2.3 Variables in Transaction-time Databases

The *transaction time* of a database fact denotes the time when the fact is (logically) current in the database [SA85]. It is an orthogonal concept to valid time, in that it concerns the evolution of the database, as opposed to the enterprise being modeled. As we will see, the use of current-time relative variables in transaction-time databases introduces a different set of problems.

While a valid-time timestamp is generally supplied by the user, a transaction-time timestamp, an interval from a "start" to a "stop" time, is supplied automatically by the DBMS during an update. Specifically, insertions initialize the "start" time to the "current time" and the "stop" time to *now*[1]. Deletions are accommodated by changing "stop" times of *now* to the value of the current transaction time. 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"

[1]The transaction processing system must also obey the requirement that the "start" times of tuples be consistent with the serialization order of their respective transactions.

time different from *now*. Physical deletion never occurs in a transaction-time relation. Updates may be considered combinations of deletions and insertions.

As an example, consider the transaction-time relation shown in Figure 4. The distinct semantics of transaction time yields a different interpretation of this relation as compared with the one shown in Figure 1. The "start" time of June 1 indicates that this tuple was stored in the database on June 1, i.e., we first became aware that Jane was an Assistant Professor on that date. The value of *now* for the "stop" attribute indicates that the database still records that Jane is an Assistant Professor, i.e., the fact that Jane is an Assistant Professor is current in the database. If we learn on July 10 that Jane left State University and thus (logically) delete the fact, this is reflected in the relation by changing the "stop" time to July 10.

FACULTY			
		TRANS TIME	
NAME	RANK	(start)	(stop)
Jane	*Assistant*	*June 1*	*now*

Figure 4: Jane's employment tuple in a transaction-time relation

In transaction time, a tuple timestamped with a "stop" transaction time of *now* means that this tuple has not yet been logically deleted [YC91]. The problem with using a variable called *now* in transaction time is that the name "now" obscures this meaning. Strictly speaking, it implies that every current tuple was deleted by the current transaction! In Figure 4, if the current time is 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.

As with valid time, some data models address this problem by using *forever* (also called ∞ or "–") instead of *now*, as shown in Figure 5 [BZ82, BG89, Sno87, TK88]. Using this large value, we immediately encounter difficulties. The strict interpretation of this tuple is that some 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 Jane has left State University, then we logically delete this tuple by changing the "stop" time to July 10. Such a change is inconsistent with the previous "stop" time. Put differently, in this scenario the database first records that we believe that Jane is an Assistant Professor from June 1 until "forever." The subsequent update then contradicts this belief by saying that it is only from June 1 until July 10 that we believe Jane is an Assistant Professor. In this one sense, *now* is somewhat more appropriate.

There is a more fundamental problem with *forever* in transaction time. By the semantics of transaction time, storing future transaction times in the database is equivalent to predicting future states of the database, which is a highly problematic proposition. With no crystal ball at hand,

FACULTY			
		TRANS TIME	
NAME	RANK	(start)	(stop)
Jane	*Assistant*	*June 1*	*forever*

Figure 5: Using forever in a transaction-time relation

it is customary to avoid predictions and require that the right endpoint of every interval be less than or equal to the current time. Since the meaning of "now" in the transaction-time dimension differs from its meaning in the valid-time dimension, we propose in this paper to adopt the name "*until changed*" for the former.

2.4 Variables in Bitemporal Databases

Bitemporal databases support both valid time and transaction time [SA85]. The confusion that has arisen in a number of bitemporal data models between the use of the same variable in both dimensions was a prime motivation for the semantic framework which we will present below. In order to allow for a completely general treatment of the semantics of these variables, we will use a different variable in each dimension. In Section 6 we show how the concept of a *reference time* can coordinate the interaction between the current-time variables in both time dimensions.

3 Semantic Framework

In order to provide a precise semantics for databases with current-time variables, in this section we propose a semantic framework for defining the meaning of databases with variables in terms of databases of a fully extensional temporal data model. Databases in this latter model are fully ground, i.e., they do not admit variables. While the model is not suitable for the implementation of temporal databases, it is well suited for capturing the semantics of variable databases.

Since the variables we discuss are temporal, we first sketch a simple model of time that supports valid time and transaction time, and introduces reference time. Reference time captures the perspective of the database user. We discuss the importance of these times in a query showing the pivotal role played by the reference time. We then outline how the mapping from the variable databases to the fully extensional logical model provides a semantics for these databases.

3.1 The Temporal Universe

The framework developed below includes three distinct time dimensions, each with its own temporal universe. The framework requires the existence of well-defined mappings between these universes. Although this requirement does not preclude the possibility of different granularities for the universes, we choose to avoid such diversions and instead use a single, underlying granularity. This yields a homogeneous treatment of all time dimensions and their relationships.

Since most database researchers have adopted the view that valid time in a database is best viewed as discrete, and every database transaction model that we are aware of has this property, we will adopt a discrete model of time here. Therefore, let $\mathcal{T}$ be our *temporal universe*, and let $\mathcal{T}$ satisfy the following properties.

1. $\mathcal{T}$ is a countably infinite set.
2. $\mathcal{T}$ is totally ordered by a relation, which we will symbolize by $<$.
3. $\mathcal{T}$ contains distinguished elements, $\top$ and $\bot$, such that $\forall t \in \mathcal{T}\ (\bot \leq t \leq \top)$. Intuitively, $\bot$ and $\top$ correspond to $-\infty$ and ∞, respectively.

In other words, $\mathcal{T}$ is isomorphic to the set $\{\ldots, -2, -1, 0, +1, +2, \ldots\} \cup \{\bot, \top\}$, i.e., the integers extended with two additional elements that satisfy property 3.

In addition to the concepts of valid time and transaction time, we introduce a third time, *reference time*, to represent the relationship between a temporal database and the "real world" time at which it is viewed. Thus, three temporal universes are required in the framework, namely the *reference time*, the *valid time*, and the *transaction time universe*, and it may be desirable or convenient to restrict them to some subset of $\mathcal{T}$. Therefore, let

- $\mathcal{T}_{RT} \subseteq \mathcal{T}$ denote the *reference time universe* of our database,
- $\mathcal{T}_{VT} \subseteq \mathcal{T}$ denote the *valid time universe* of our database, and
- $\mathcal{T}_{TT} \subseteq \mathcal{T}$ denote its *transaction time universe*.

3.2 Important Times

Throughout our discussion of variable databases and queries on these databases, five times surface repeatedly. The first of these is called *initiation*. It is relative to a specific relation and denotes the time in $\mathcal{T}_{TT}$ when that relation was created. To simplify the discussion that follows, we assume that all relations are created at the same time, denoted by tt_0. Once created, we assume that the database schema never changes. Schema versioning [Rod92] is orthogonal to most of the issues discussed in this paper.

The second important time, which is new to most readers, is the *reference time*. The reference time is the time of the database observer's "frame of reference," denoted by rt_*. Reference time is a term analogous to the *indices* or "points of reference" in intensional logic [Mon74], and discussed more recently in the context of valid-time databases [Fin92]. The reference time facilitates a kind of "time travel" by means of which we may observe the database at times other than the present.

A related time is the *query time*, or *current transaction time*, denoted by $t_{current}$. It is the time at which a query is processed. The reference time, rt_*, and current transaction time, $t_{current}$, are related, but distinct. In general, $t_{current}$ is the time at which a query is initiated, while rt_* is the time at which the user "observes" the database. In many queries, the user "observes" the database with respect to the same frame of reference in which the query was initiated, so the reference time and the query time are the same. But the user may choose to "observe" the database from a previous perspective; for this kind of query, the reference time is earlier than the query time. For example, if today is July 9 and we wish to observe the database from the perspective of a week ago, then $t_{current}$ = July 9, and rt_* = July 2.

The final two times of special interest are the *valid timeslice time*, vt_*, and the *transaction timeslice time*, tt_*. These times are important in this paper because, for expository purposes, we focus exclusively on various timeslice queries. The valid and transaction timeslice times could both be an instant, an interval, or a set of instants or intervals. The valid timeslice time(s) specifies the real-world time about which information is wanted, while the transaction timeslice time(s) is the time(s) during which information must be current in the database in order to be of interest for a query. For the example queries given in this paper, it is advantageous choose an instant as the valid timeslice and transaction timeslice time in timeslice queries—such instants are denoted by vt_* and tt_*, respectively.

Later, we shall see that there are important relationships between the valid timeslice time, the transaction timeslice time, and the reference time.

To illustrate the distinction among these five times, let us consider the following example. A temporal database for recording employment information is created on May 11 (as mentioned above, the particular year is immaterial). Today (which we assume is July 9), the director of the personnel department investigates an apparent discrepancy reported by a co-worker a week earlier,

while using the database on July 2. The co-worker discovered that the database had mistakenly recorded on June 27 that an employee had been hired two weeks earlier, on June 13. The five times in this example are as follows.

1. tt_0 is May 11, the day of the creation of the database;
2. rt_* is July 2, the day when the problem was observed;
3. $t_{current}$ is July 9, the day the personnel department director investigates the database;
4. vt_* is June 13, the real-world day of the problematic information; and
5. tt_* June 27; this is the day for which we are interested in what was recorded as current information in the database.

By using a reference time of July 2, the director can view the identical database state in existence when the co-worker discovered the discrepancy. If a reference time of June 20 had been used instead, it is possible that no discrepancy would have been found, because that date was well before tt_*.

We have the following constraints on these five times.

- $\bot \leq tt_0 \leq tt_* \leq t_{current} \leq \top$
- $\bot \leq rt_* \leq \top$
- $\bot \leq vt_* \leq \top$

3.3 Extensional and Variable Database Levels

It is useful to view the semantics of temporal databases with variables within the context of a two-level framework. This section develops such a framework in two steps, by first presenting the levels of a theoretical framework. Then this framework is augmented, motivated by the practical concerns of easily extending existing temporal data models to admit databases with variables, such as *now*, with minimal impact on the existing query language and query processing engine.

In order to introduce the two-level framework and discuss the issues related to variable databases, we need to adopt the following standard terminology. A relational database consists of a set of relations, where each relation is a set of tuples. Each tuple in a relation has a number of application-specific attribute values. Temporal databases extend this view by incorporating, in addition, some number of timestamp attributes (e.g., "from" and "to", as in Figure 1, or "start" and "stop," as in Figure 4). In a variable database, the value of the timestamp attributes in any tuple is extended to permit instances of one or more current-time variables, as discussed earlier in the paper. This general view of a temporal variable database enables us to focus on the issues surrounding the use of variables in a temporal database, regardless of the number of variables or the number of temporal attributes in a particular data model.

Most temporal data models, such as the one adopted in TSQL2 [SAA+94], have incorporated the temporal dimension with temporal attributes whose values, termed timestamps, are represented using *time intervals* (or just *intervals* for short), rather than *instants*. Such timestamps are very convenient at the conceptual and implementation levels, since they are compact and can represent information about a potentially large number of times in a single tuple.

In the model of time presented in the previous section, instants are points in time and intervals are sequences of temporally consecutive points. In general, intervals are uniquely described or denoted by two bounding instants, termed the *starting* and *terminating* instants. In a valid-time

interval, the starting instant is the "from" time and the terminating instant is the "to" time, whereas transaction-time intervals have "start" and "stop" instants. The use of interval notation to denote time intervals is common. When time is discrete, intervals are merely shorthand for a finite (or countably infinite) set of instants. To summarize, tuples at the variable level are timestamped with intervals whose endpoints can be temporal variables.

At the extensional level, tuples also have temporal attributes as do tuples at the variable level. However, there are three key differences. First, no variables are allowed at the extensional level. The extensional level is fully ground. Second, timestamps are instants, rather than intervals. Third, an extensional tuple has one additional temporal attribute, called a *reference time attribute*. Later in this paper we describe the importance of reference time to the meaning of tuples. For now, it may be thought of as representing the time at which a meaning was given to the temporal variables in the tuple.

Whereas the variable database level offers a convenient representation that end-users can understand and that is amenable to implementation, the mathematical simplicity of the extensional level supports a rigorous treatment of temporal databases in terms of first order logic. A theoretical framework for providing a logical interpretation or "meaning" for a particular variable database, i.e., a "translation" from variable to extensional level, may be based on homomorphic mapping from variable-level databases to extensional-level databases [CI94]. This mapping is termed an extensionalization, and is denoted $[\![\ ]\!]$. In addition to giving the semantics of variable databases, the framework also provides a means for checking the correctness of query languages over variable databases. This is illustrated in Figure 6 and explained in the following.

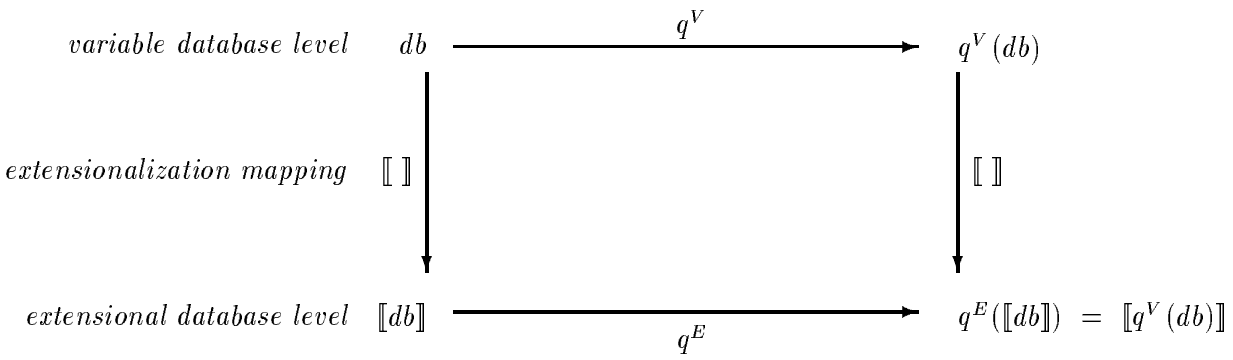


Figure 6: Relationship between variable database and extensional database

The top of the figure, labeled the *variable database level*, represents a database model that allows the use of temporal variables in timestamps of tuples. At the top left, we see a particular variable database, db. A query q^V is applied to this database, resulting in another variable database, $q^V(db)$. The bottom of the figure, labeled the *extensional database level*, represents our fully extensional temporal data model, whose semantics is well-specified in the standard tradition of a first-order logical framework. Developing a query language in this extensional model is relatively straightforward, due to the model's simplicity. In contrast, developing a query language for a more complex variable-level data model is error prone. The framework can be used for checking the correctness of variable-level query constructs. Specifically, variable-level query constructs must commute with the corresponding extensional-level query constructs, as indicated in the figure: $q^E([\![db]\!]) = [\![q^V(db)]\!]$.

A particular extensionalization mapping from the top level to the bottom level is defined in

order to specify the semantics of variable databases. As tuples at the variable database level are independent of each other, an extensionalization mapping may treat each tuple in isolation.

We are concerned in this paper with the practical use of variable databases. In particular, we are interested in how to extend existing temporal data models and query languages with the ability to allow current-time variables in their temporal dimensions, with as little impact as possible on their conceptual model and their associated query processing engines. This is consistent with the philosophy of the designers of the proposed temporal extension to SQL-92, termed TSQL2 [SAA+94]. Thus, we next augment the theoretical framework by introducing a third, intermediate level, as shown in Figure 7[2].

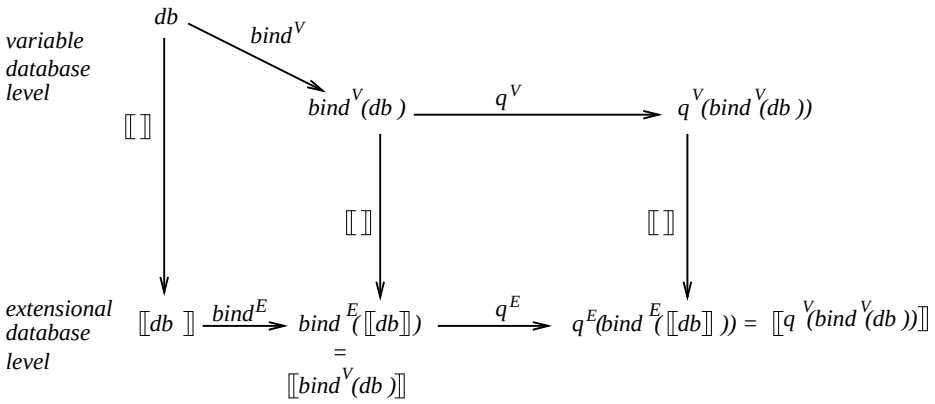


Figure 7: Preprocessing of variable-level databases

Tuples at this intermediate level contain interval timestamps but no variables. The various existing temporal data models, including TSQL2, that do not permit variable timestamps in their databases belong at this level. By mapping variable level databases to this intermediate level, it is possible to reuse existing—or proposed—query engines to query variable databases. This is the motivation for augmenting the framework to permit preprocessing of variable databases before querying them.

The preprocessor substitutes each instance of a variable with a specified time, effectively "binding" the variables in a variable database (as discussed in Section 7.3, this occurs during query evaluation on a per-tuple basis). The bind operator, $bind^V$, maps a database from the variable to the intermediate level, then it is queried with a variable-level query q^V. The correctness of this mechanism is ensured by providing extensional level counterparts to the preprocessor and to the queries, $bind^E$ and q^E respectively, and by demonstrating that the above diagram commutes.

We focus on one particular kind of query, *time-slice*, which, in addition to being simple and well-known, illustrates clearly the interaction among the various interesting times that come into play. We define valid-time and transaction-time operators at both the variable and extensional level, and we develop a framework that leads to the desired correctness results.

In Sections 4 through 6, we define concrete bind and extensionalization mappings for valid-time, transaction-time, and bitemporal databases, respectively, that employ current-time variables as timestamp values. In particular, we will show how the meaning of a tuple with current-time variables depends on the reference time.

[2] In the definitions to follow, *bind* requires a subscript and [[ ]] takes an optional subscript. We omit these subscripts here to simplify the discussion.

Not surprisingly, it is possible to define other semantics for these variables. Moreover, other "useful" semantics may exists for the different database types. So while we give here a specific, and we believe, reasonable, semantics for each type of database, other semantics are possible [CI94].

Finally, we emphasize that many models, e.g., [CW83] and [CCT93], have been presented completely extensionally; they did not present an operational model at the variable level and thus did not make use of any variable symbols requiring further interpretation. However, when variables such as *now* are present, some well-specified mechanism is needed to unambiguously interpret variable databases. We believe that the framework presented here constitutes such a mechanism.

4 Valid-Time Databases

In this section we present a semantics for variable valid-time databases by specifying mappings to the extensional level, of tuples with timestamps that may include such variables. We initially consider the extensionalization mappings for databases with ground timestamps and timestamps with the variable *now*. With varying names, this variable is employed in a number of existing data model proposals. In order to address the shortcomings identified in Section 2, we also introduce additional current-time, modeling entities. Specifically, we consider *now-relative* timestamps that allow for positive and negative displacements from *now*. Next, we introduce so-called indeterminate time values which may be used in timestamps to indicate an imprecise time. This leads to a further generalization of now-relative instants to *now-relative indeterminate* instants, which are values that are imprecise as well as current-time relative. The section concludes with an illustration of the querying of variable databases.

4.1 Extensionalization of Valid-time Tuples with *Now*

We first consider the extensionalization of tuples with ground timestamps. To do this, it is convenient to start by defining the meaning, or denotation, of the ground component in a timestamp. As other timestamp values are introduced, their denotations will also be defined.

Definition 4.1 [Denotation of Time Instants]
The *denotation* of a valid-time instant t at a particular reference time rt_*, written $\langle\langle t \rangle\rangle_{rt_*}$, is defined as follows.

$$\langle\langle t \rangle\rangle_{rt_*} =_{df} t \qquad \blacksquare$$

In general, to map a *ground* valid-time tuple, i.e., a tuple without variables, to the extensional database level, the tuple is *expanded* into a set of tuples, one for each time instant in its associated timestamp. Let us consider first the extensionalization of a ground tuple at a particular reference time. We will use the notation, $[\![T]\!]_{rt_*}$, to denote the extensionalization of tuple T at reference time rt_* .

Definition 4.2 [Extensionalization of a tuple at an Instant]
The *extensionalization* of a ground tuple T of the form $T =< X, [vt_1, vt_2] >$ at reference time rt_* is defined as follows.

$$[\![T]\!]_{rt_*} =_{df} \{(X, vt, rt_*) \mid vt \in [\langle\langle vt_1 \rangle\rangle_{rt_*}, \langle\langle vt_2 \rangle\rangle_{rt_*}]\} \qquad \blacksquare$$

Note that each extensional tuple is tagged with a reference time attribute whose value is the reference time at which the tuple was extensionalized. For example, the extensionalization of the tuple $< Jane, Assistant, [June\ 1, June\ 9] >$ at reference time July 9 yields the set

$$\{(Jane, Assistant, June\ 1, July\ 9), \ldots, (Jane, Assistant, June\ 9, July\ 9)\}.$$

Notice that since this tuple does not contain variables, its extensionalization is essentially independent of the reference time. Not surprisingly, the meaning of any ground tuple is *reference-time invariant*, that is, it always has the same meaning no matter when we observe it. The role of the reference time attribute in the meaning of each extensional database level tuple will become clear when we revisit this topic in the presence of variables.

In the extensionalization mapping, a reference time interval may be used rather than a single reference time.

Definition 4.3 [Extensionalization at an Interval]
The extensionalization of the tuple T over the reference time interval $[rt_1, rt_2]$ is defined as follows.

$$[\![T]\!]_{[rt_1, rt_2]} =_{df} \bigcup_{rt_* \in [rt_1, rt_2]} [\![T]\!]_{rt_*}$$ ∎

The complete meaning or extensionalization of a tuple T, denoted $[\![T]\!]$, is simply the extensionalization of T over all reference times.

Definition 4.4 [Extensionalization (Complete)]
The general meaning or extensionalization of a tuple T is: $[\![T]\!] =_{df} [\![T]\!]_{[\perp, \top]}$. ∎

We have found that a two-dimensional graphical notation makes valid-time concepts easier to grasp. In the visualization, reference time corresponds to the X-axis and valid time corresponds to the Y-axis. The graphical representation is a plot of the tuple at the extensional database level. Each cell in the plot stands for a particular reference time, RT, and valid time, VT, combination. The cells corresponding to the temporal coordinates of tuples in the extensional set of tuples are shaded, indicating when a tuple is valid relative to the reference time of an observer. Even though our underlying model of time is discrete, we treat each cell as a region rather than a point since this results in a better visualization. Several tuples may be plotted in the same graph by using different cell colors or patterns. The key, shown below the graph, indicates the explicit attribute values of the corresponding tuple.

As an example, assume that the academic career of Jane at State University is given by the tuple $T =< Jane, Assistant, [June\ 3, June\ 9] >$. Figure 8 graphically depicts this tuple for a sequence of reference times, June 1 through June 11, that is, it visualizes $[\![T]\!]_{[June\ 1, June\ 11]}$. The key to the graph indicates the color or pattern of the cell that represents the value of the attributes in the tuple corresponding to the $< rt, vt >$ combination. For example, at $rt = June\ 4$, $vt = June\ 5$, Jane is an Assistant Professor.

The graph illustrates that the valid time of this tuple is independent of the reference time. So for a tuple with a valid-time interval but without variables, it does not matter at what time the tuple is observed, it is always valid over exactly the same interval.

The extensionalization of tuples containing a time of *now* is straightforward: *now* is bound to the reference time rt_* and the interval is then extensionalized as before.

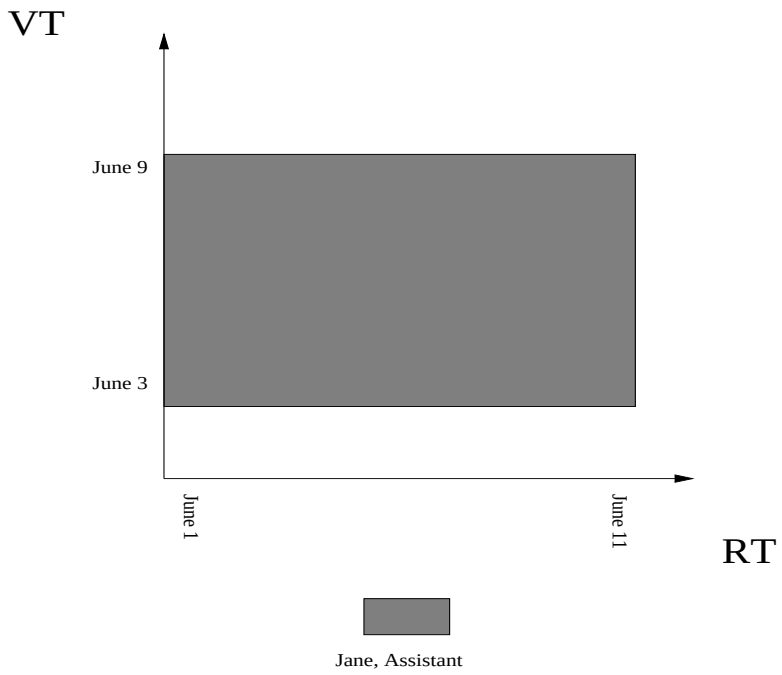


Figure 8: A graphical representation of the extensionalization of a valid-time tuple

Definition 4.5 [Denotation of *Now*]
The *denotation* of the current-time variable *now* at a particular reference time rt_* is defined as follows.

$$\langle\langle now \rangle\rangle_{rt_*} =_{df} rt_* \qquad \blacksquare$$

With this additional timestamp value, the extensionalization of a tuple with *now* is given by Definition 4.2. The definition permits tuples with *now* as the "to" and "from" time. Note also that Definitions 4.3 and 4.4 are applicable to tuples with variables.

To exemplify the extensionalization, the tuple $< Jane, Assistant, [June\ 1, now] >$ of Figure 1 is extensionalized as follows.

$$[\![< Jane, Assistant, [June\ 1, now] >]\!]_{rt_*} = \{< Jane, Assistant, vt, rt_*) \mid vt \in [June\ 1, rt_*]\}.$$

For a given reference time, say $rt_* = June\ 2$, this results in the set of tuples

$$\{(Jane, Assistant, June\ 1, June\ 2), (Jane, Assistant, June\ 2, June\ 2)\}.$$

As can be seen via this example, the extensionalization of a tuple containing *now* depends upon the reference time. The reference time determines the exact bounds of the valid-time interval in the tuple. As the reference time increases so does the duration of the valid-time interval.

Figure 9 visualizes the extensionalization of the tuple given above, for every reference time between June 1 and June 8. Note that before June 1 the *empty interval* is depicted in the figure. This is because a timestamp with a "from" time that is after the "to" time denotes the empty interval. This situation occurs prior to June 1. The valid-time region in the figure is "stair-shaped" since, unlike a valid-time tuple without variables, the extensionalization of a tuple with variables is dependent on the time at which we observe the tuple. The stair-shape is a result of the constraint that the terminating time in the valid time interval is bound to the reference time.

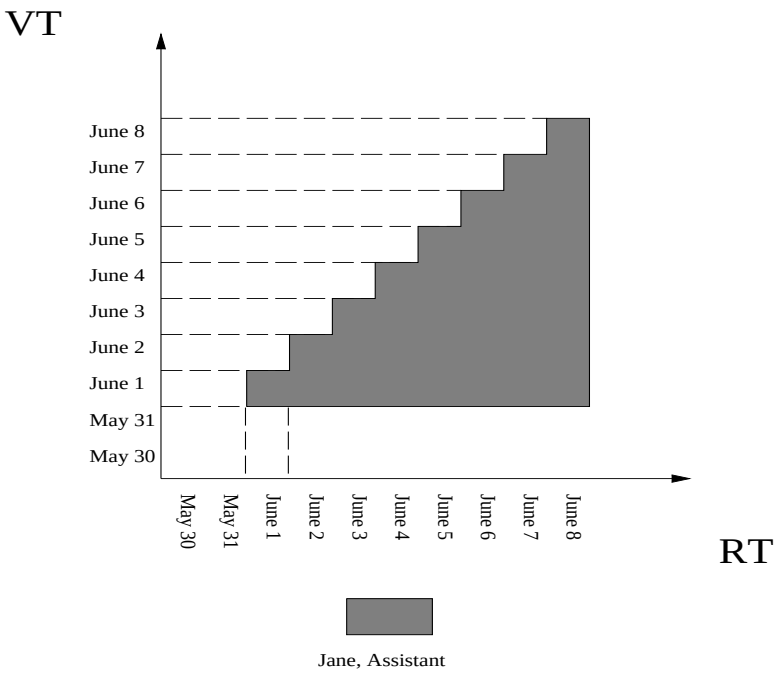


Figure 9: A graphical representation of the extensionalization of a valid-time tuple with a variable

The symbol *forever* has been used in valid-time databases with the following interpretation: *forever* $= \top$. Here, $\top$ is the special element in $\mathcal{T}_{VT}$ that is greater than any other time in $\mathcal{T}_{VT}$. As a consequence, we point out that the symbol *forever* is in fact not a variable, but a distinguished ground value.

It is our contention that all other valid-time current-time variables currently in use (e.g., "@" [LJ88] and *until-changed* [WJL91, WJL93]) have the same meaning as *now*. Thus having covered existing variables, we now proceed by proposing new timestamps that address the shortcomings of *now* discussed in Section 2.

4.2 Now-relative Instants

In this section we introduce a new type of timestamp, called a *now-relative instant*, that adds flexibility to the variable *now*. Now-relative times were first introduced in transaction time for vacuuming [JM90]. A now-relative instant generalizes the variable *now* by allowing an offset from this variable to be specified. The "to" time in Jane's employment tuple shown in Figure 10 is a now-relative variable.

FACULTY			
		VALID TIME	
NAME	RANK	(from)	(to)
Jane	*Assistant*	*June 1*	*now − 3 days*

Figure 10: Using a now-relative instant

With now-relative instants, we have a means of more accurately recording our knowledge of Jane's

employment with State University. As an example, if the bound on the relationship between transaction time and valid time is known to be three days (all updates are made three days after the occurrence of the event), then Jane's employment extends from when she was hired (June 1) to three days before *now*, as shown in Figure 10. As another example, we can record our knowledge on May 12 that Jane will begin employment on June 1, with all terminations given a two-week notice, as seen in Figure 11.

FACULTY			
		VALID TIME	
NAME	RANK	(from)	(to)
Jane	*Assistant*	*June 1*	*now* + 14 *days*

Figure 11: Using a predictive now-relative instant

A now-relative instant includes a displacement, which is a (signed) span, from *now*. In the examples given above, the displacements are minus three days and plus fourteen days, respectively. The extensionalization of tuples with now-relative instants is formalized as follows.

Definition 4.6 [Denotation of Now-relative Instants]
The *denotation* of a now-relative instant, *now* OP n days, OP $\in \{+, -\}$, at a particular reference time rt_* is defined as follows.

$$\langle\!\langle \textit{now OP } n \textit{ days} \rangle\!\rangle_{rt_*} =_{df} \langle\!\langle \textit{now} \rangle\!\rangle_{rt_*} \text{ OP } n \qquad \blacksquare$$

Even with this additional timestamp value, the extensionalization of a valid-time tuple is still given by Definition 4.2. When a now-relative instant appears in a tuple, that tuple is transformed to a set of ground tuples as follows. First, the variable *now* is bound to the reference time, and then the offset is added or subtracted, yielding rt_* OP n. Then the tuple is expanded into a set of ground tuples. For example, consider the extensionalization of the tuple in Figure 10 on July 9, i.e.,

$$[\![< \textit{Jane}, \textit{Assistant}, [\textit{June } 1, \textit{now} - 3 \textit{ days}] >]\!]_{\textit{July } 9} .$$

First *now* $-$ 3 *days* is bound to the reference time, July 9, minus three days, i.e., to July 6, and the resulting tuple is expanded into the set of tuples

$$\{(\textit{Jane}, \textit{Assistant}, \textit{June } 1, \textit{July } 9), \ldots, (\textit{Jane}, \textit{Assistant}, \textit{July } 6, \textit{July } 9)\}.$$

Figure 12 visualizes the tuple in Figure 10 for every reference time between June 1 and June 11. Note that before June 4 the *empty interval* is depicted in the figure. This is because before June 1, the "to" time is earlier than the "from" time. The visualization of this tuple is similar to that of the tuple with just *now* as the terminating time, shown in Figure 9. But in Figure 12, the stair-shape has been shifted by three days along the reference time axis.

One refinement of now-relative instants is to "round" the span values in such instants to a meaningful calendar-specific boundary by using a calendar-specific span [S92c]. For a predictive update, a useful span would be "rest of month." When used in a now-relative instant, such a span would model the situation where all hiring decisions take effect on the first day of the following month rather than just 20 days into the future. That is, *now* plus the "rest of the month" would result in the first day of the following month.

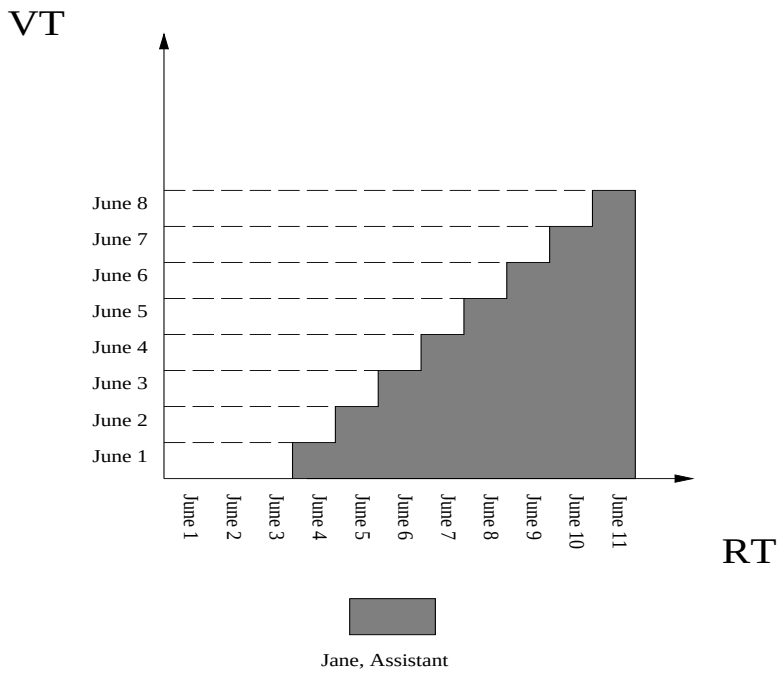


Figure 12: A graphical representation of Jane's employment

In Section 7, we demonstrate that now-relative instants can be stored in the same representation as other instants, with the result that they impose no space overhead. Section 7.3 demonstrates that the execution time overhead of now-relative instants is minimal.

Although now-relative instants allow us to relax the otherwise close coupling between valid and transaction time found in the punctuality assumption, now-relative instants still suffer from making a pessimistic assumption. The use of $now - 3$ days in the first example is an ultra-pessimistic view of the future. Jane would not even be employed *now* since her employment terminates three days prior to *now*. To address this potential shortcoming, we next introduce the notion of indeterminate timestamp values.

4.3 Indeterminate Timestamp Values

It turns out that valid-time indeterminacy, introduced in another context [DS93b, Dyr94], can also alleviate the shortcomings of *now* and now-relative instants. This section introduces the notion of indeterminate timestamp values and considers only ground timestamps. The next section considers variable indeterminate timestamps.

Sometimes, the time when an event occurred is known only imprecisely. For instance, we may know that an event happened "sometime in June 1993," which is an imprecise period of 30 days. An *indeterminate instant* is the time of an event, which is known to have occurred, but exactly when is unknown. As we shall see in later sections, indeterminate instants are very useful in addressing the issues raised by evolution of data in the valid-time dimension.

The times when the event might have occurred is called the *period of indeterminacy* and is delimited by a lower and an upper bound (e.g., the event occurred sometime between June 1 and June 30). An indeterminate instant could have an associated probability distribution that gives the probability that the event occurred for each time in the period of indeterminacy. For the purposes of this paper, we ignore the probability information: every indeterminate instant is treated as

though it has a distribution that is *missing* [Dyr94]. A *determinate* instant may be thought of as an indeterminate instant, with identical lower and upper bounds.

An *indeterminate interval* is an interval bounded by indeterminate instants. The indeterminate interval begins sometime between the lower and upper bounds of the starting instant and ends sometime between the upper and lower bounds of the terminating bounding instant. Since it is unknown precisely when an indeterminate instant occurs, it follows that it is unknown when an indeterminate interval begins or ends. Instead, an indeterminate interval describes a *set of possible intervals*. Every combination of times in the starting and terminating instants' periods of indeterminacy is in the set of possible intervals.

By using indeterminate instants, we can more accurately record our knowledge of Jane's employment with State University. Instead of using *now* as the "to" time in Jane's tuple, we can use an indeterminate instant. Which indeterminate instant to use depends on our knowledge of the situation. If Jane was hired as a limited-term employee, to work between two and three months, we could record this information as shown in Figure 13(a). Here two time bounds, July 31 and August 31, delimit the "to" indeterminate instant. If we knew only that the term would be *at least* two months, we would use the representation shown in Figure 13(b). If State University has a mandatory retirement policy, we could decrease the indeterminacy considerably, as shown in Figure 13(c). If we removed the guarantee that Jane will work at least two months, we arrive at the representation shown in Figure 13(d).

FACULTY

	NAME	RANK	VALID TIME (from)	VALID TIME (to)
(a)	*Jane*	*Assistant*	*June 1*	*July 31 ~ August 31*
(b)	*Jane*	*Assistant*	*June 1*	*July 31 ~ forever*
(c)	*Jane*	*Assistant*	*June 1*	*July 31 ~ January 1, 2028*
(d)	*Jane*	*Assistant*	*June 1*	*June 1 ~ January 1, 2028*

Figure 13: Using indeterminate timestamps for recording Jane's appointment

Indeterminate instants address the first problem, the pessimistic update assumption, providing evidence that Jane might still be employed in the future. They also remove the problem of incompleteness in the non-timestamp attributes (e.g., *possibly* employed, as shown in Figure 3), and ensure that new knowledge acquired later, such as the information that Jane left the company on August 10, is not inconsistent with currently stored information, but rather is a refinement of that information. They also address the problem of *now* in predictive updates; an indeterminate interval is a valid interval no matter when it was stored in the database.

There are two bounds on the information represented by an indeterminate interval [Lip79]. The first bound is the *definite* information. The definite information represents all that is definitely known about the interval and is the intersection of all of the possible intervals. The second bound is the *possible* information. The possible information represents the maximum possible extent of an interval and is the union of all of the possible intervals.

The two bounds have different extensionalizations. The definite information is given by the *definite extensionalization*, given next.

Definition 4.7 [Definite Extensionalization of an Indeterminate Tuple]
The definite extensionalization of a ground tuple of the form $T =< X, [vt_1 \sim vt_2, vt_3 \sim vt_4] >$, where vt_1 and vt_2, $vt_1 \leq vt_2$, are the lower and upper bound, respectively, of the starting instant and vt_3 and vt_4, $vt_3 \leq vt_4$, are the lower and upper bound, respectively, of the terminating instant, at the reference time rt_* is defined as follows.

$$[\![T]\!]^D_{rt_*} =_{df} \{(X, vt, rt_*) \mid vt \in [\langle\!\langle vt_2 \rangle\!\rangle_{rt_*}, \langle\!\langle vt_3 \rangle\!\rangle_{rt_*}]\}$$ ∎

For example, the *definite extensionalization* of the tuple:

$$< Jane, Assistant, [June\ 1 \sim June\ 3, June\ 5 \sim June\ 9] >$$

at a reference time of July 9 is the set of tuples

$$\{(Jane, Ast., June\ 3, July\ 9), (Jane, Ast., June\ 4, July\ 9), (Jane, Ast., June\ 5, July\ 9)\}.$$

Note that the extensionalization of an indeterminate instant is reference-time invariant.

The possible information is given by the *possible extensionalization*.

Definition 4.8 [Possible Extensionalization of an Indeterminate Tuple]
The possible extensionalization of a ground tuple of the form $T =< X, [vt_1 \sim vt_2, vt_3 \sim vt_4] >$ at the reference time rt_* is defined as follows.

$$[\![T]\!]^P_{rt_*} =_{df} \{(X, vt, rt_*) \mid vt \in [\langle\!\langle vt_1 \rangle\!\rangle_{rt_*}, \langle\!\langle vt_4 \rangle\!\rangle_{rt_*}]\}$$ ∎

The *possible extensionalization* of the above example tuple at reference time July 9 is the following set of tuples.

$$\{(Jane, Ast., June\ 1, July\ 9), (Jane, Ast., June\ 2, July\ 9), \ldots, (Jane, Ast., June\ 9, July\ 9)\}$$

It is always the case that the definite information is a subset of the possible information. Note, that if the bounding instants are determinate, that is, if the lower and upper bounds are the same, then the possible and definite extensionalizations yield exactly the same set of tuples. Consequently, for the extensionalization of determinate intervals, we omit the possible or definite superscript and use $[\![\ ]\!]_{rt_*}$ instead of either $[\![\ ]\!]^P_{rt_*}$ or $[\![\ ]\!]^D_{rt_*}$.

Valid-time tuples timestamped with indeterminate intervals have a graphical representation similar to the one described above. The primary difference between the indeterminate and determinate visualizations is that for indeterminacy, both the possible and definite extensionalizations must be represented. We use different shadings to distinguish the regions in the two extensionalizations.

As an example, assume that the academic career of Jane at State University is given by the tuple

$$< Jane, Assistant, [June\ 1 \sim June\ 3, June\ 7 \sim June\ 10] > .$$

Jane's academic career, for the reference times $[June\ 1, June\ 11]$, is graphically represented in Figure 14. Different colors or patterns, as indicated by the graph's key, are used to depict the definite and possible information in this tuple. Note that the region of possible information is never smaller than the region of definite information and that the valid time is independent of the reference time (just as it is for determinate intervals) when the tuple has no variables.

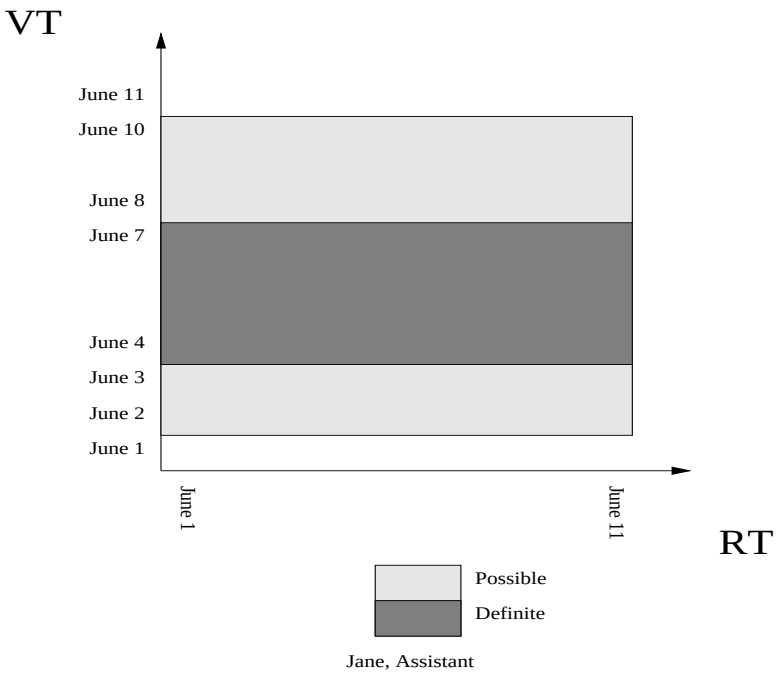


Figure 14: A graphical representation of an indeterminate tuple

4.4 Now-relative Indeterminate Instants

To achieve the full benefit of indeterminate timestamp values, we proceed by introducing now-relative indeterminate instants, which may be understood as generalizations of the ground, indeterminate timestamps presented above and of the now-relative instants presented earlier.

To exemplify and motivate the utility of this new type of instant, assume that today is July 9 and that Jane departed today. Assume also that her departure has not yet been recorded in the database, and that there is at most a three day lag in recording a fact in the database. Jane's tuple in the database should not be that of Figure 13(d), but rather that shown in Figure 15(a) which is more accurate. The state on July 10 is shown in Figure 15(b). (Recall that for simplicity all dates are assumed to be in 1994 unless otherwise specified.) Note how the indeterminacy in the "to" instant has decreased ever so slightly—on July 10 we know that Jane was employed on July 6. On July 11, the indeterminacy disappears as we learn of Jane's departure, as shown in Figure 15(c). Each successive state is consistent with that preceding it, and each accurately records our current knowledge of Jane's status.

To accurately represent our continuously changing knowledge about Jane's employment, we need to combine the best features of now-relative instants and ground indeterminate values, into a new kind of instant, which we call a *now-relative indeterminate instant.* An example is shown in Figure 15(d). Here, the the "to" timestamp is such an instant.

Now-relative indeterminate instants provide a flexible means of precisely capturing our imprecise, but current-time dependent, knowledge of when a fact is valid. For instance, in the tuple given in Figure 15(d), we are certain that Jane was an Assistant Professor starting on June 1, but our knowledge of when she ceases to be an Assistant Professor is imprecise; all we know is that she was definitely an Assistant Professor until three days ago and that it is possible that she will remain an Assistant Professor until retirement on January 1, 2028. The "to" timestamp allows us to capture this precisely.

	FACULTY			
			VALID TIME	
	NAME	RANK	(from)	(to)
(a)	*Jane*	*Assistant*	*June 1*	*July 6* $\sim$ *January 1, 2028*
(b)	*Jane*	*Assistant*	*June 1*	*July 7* $\sim$ *January 1, 2028*
(c)	*Jane*	*Assistant*	*June 1*	*July 9*
(d)	*Jane*	*Assistant*	*June 1*	*now* $-$ 3 *days* $\sim$ *January 1, 2028*

Figure 15: Using indeterminate and now-relative indeterminate timestamps

Put differently, the lower bound of the "to" timestamp expresses, on a day-to-day basis, our changing knowledge of when Jane was employed while the upper bound expresses our knowledge of when she will definitely no longer be an Assistant Professor. Using a now-relative indeterminate instant ensures that continual updates are not required, while capturing all of our knowledge of exactly when Jane is employed by State University.

There is one wrinkle to this scheme. If a now-relative indeterminate instant appears in an indeterminate interval, then the possible or definite information in the interval cannot exceed the instant's upper bound. (The upper bound cannot be now-relative, as that would result in inconsistent information being added to the database solely as a side effect of the passage of time.) So, for instance, the possible or definite information represented by Jane's employment tuple shown in Figure 15(d) cannot extend beyond January 1, 2028, even if today is after January 1, 2028. If today is May 9, then the lower bound is May 9 and the tuple indicates that we *expect* Jane to be (possibly) employed from June 1 to January 1, 2028. If today is January 1, 2050, then the upper bound is January 1, 2028 and the tuple indicates that Jane was *actually* employed from June 1 to January 1, 2028. In short, now-relative indeterminate instants capture the semantics of predictive updates.

Now-relative, indeterminate instants are able to model the evolutionary character of temporal databases. A real-life prediction situation is either confirmed or proven false as time progresses. Similarly, as the reference time increases, values in the *possible* extensionalization of a tuple evolve into *definite* values or are removed from the database. Consider the tuple of Figure 15(d). If the reference time is May 9, then Jane will be *possibly* employed every day between June 1, 1994 and January 1, 2028. However, for a reference time of April 1, 2028, the database records that Jane has *definitely* been employed every day between June 1, 1994 and January 1, 2028.

As an example of the extensionalization, consider a valid-time tuple where a now-relative indeterminate instant appears in the timestamp as a "to" time. The tuple is transformed to a set of ground tuples by either a possible or a definite extensionalization, $[\![\ ]\!]^P_{rt_*}$ or $[\![\ ]\!]^D_{rt_*}$, respectively. Both extensionalizations are similar in that they use the semantics of now-relative determinate instants presented in Section 4.2. First, the reference time, displaced by the span, is substituted for the lower bound. Second, the new lower bound is checked to determine if it later than the upper bound. If the new lower bound is later, then the new lower bound is changed to be the same time as the upper bound. Third, if the upper bound is before the lower bound, the upper bound is replaced by the lower bound. Finally, the tuple is expanded into a set of tuples. The possible and definite

extensionalizations have different expansions.

For clarity and as a precursor to the definition of the extensionalization of tuples with now-relative indeterminate instants, we first extend the extensionalization mapping to cover temporal values, the variable *now*, and now-relative instants.

Definition 4.9 [Possible Extensionalization of a Now-relative Indeterminate Tuple]
The possible extensionalization of the tuple $T =< X,\ [e_1 \sim vt_2, e_3 \sim vt_4], >$ where e_1 and e_3 are any of the timestamp values introduced in Definitions 4.1, 4.5, and 4.6 and vt_2 and vt_4 are ground values (we have only described timestamps of this form) at reference time rt_* is defined as follows.

$$[\![T]\!]^P_{rt_*} =_{df} \{(X, vt, rt_*) |\ vt \in [\min(\langle\!\langle e_1 \rangle\!\rangle_{rt_*}, \langle\!\langle vt_2 \rangle\!\rangle_{rt_*}), \langle\!\langle vt_4 \rangle\!\rangle_{rt_*}]\}.$$ ∎

Definition 4.10 [Definite Extensionalization of a Now-relative Indeterminate Tuple]
The definite extensionalization of the tuple $T =< X,\ [e_1 \sim vt_2, e_3 \sim vt_4], >$ at reference time rt_* is defined as follows.

$$[\![T]\!]^D_{rt_*} =_{df} \{(X, vt, rt_*) |\ vt \in [\langle\!\langle vt_2 \rangle\!\rangle_{rt_*}, \min(\langle\!\langle e_3 \rangle\!\rangle_{rt_*}, \langle\!\langle vt_4 \rangle\!\rangle_{rt_*})]\}.$$ ∎

Note that a tuple with a now-relative indeterminate instant may yield no definite information or may have the same possible and definite information content; it all depends upon when we observe that tuple.

As an example, consider the possible extensionalization of the tuple shown in Figure 15(d) on June 2, e.g.,

$$[\![< Jane, Assistant, [June\ 1, now - 3\ days \sim January\ 1,\ 2028] >]\!]^P_{June\ 2}\ .$$

The "from" time of *June* 1 is the same as the indeterminate value *June* 1 $\sim$ *June* 1, and $\min(\langle\!\langle June\ 1 \rangle\!\rangle_{June\ 2}, \langle\!\langle June\ 1 \rangle\!\rangle_{June\ 2}) = June\ 1$. Similarly, $\langle\!\langle January\ 1,\ 2028 \rangle\!\rangle_{June\ 2} = January$ 1, 2028. Thus, the result is

$$\{(Jane, Assistant, June\ 1, June\ 2), (Jane, Assistant, June\ 2, June\ 2), \ldots, (Jane, Assistant, January\ 1,\ 2028, June\ 2)\}.$$

The definite extensionalization, on June 2, expands the tuple into the empty set since Jane has yet to be definitely employed.

The visualization of a tuple at the extensional database level with an now-relative indeterminate time is similar to the visualization of a tuple with an indeterminate interval. Both the definite and possible regions are plotted on the same graph but using different colors or patterns.

Figure 16 shows a graph of both the possible and definite extensionalizations of the tuple in Figure 15(d) for every reference time between May 30 and January 5, 2028. Note that for all reference times before June 4 the tuple does not contain any definite information, only possible information. The definite information gradually increases as the reference time advances. On January 4, 2028, and for all reference times thereafter, the possible and definite information for the tuple are the same.

In Section 7, we demonstrate that now-relative indeterminate instants can be stored in the same representation as indeterminate instants and non-relative instants, with the result that they impose little space overhead. In addition, we demonstrate that the execution time overhead of now-relative indeterminate instants is not excessive.

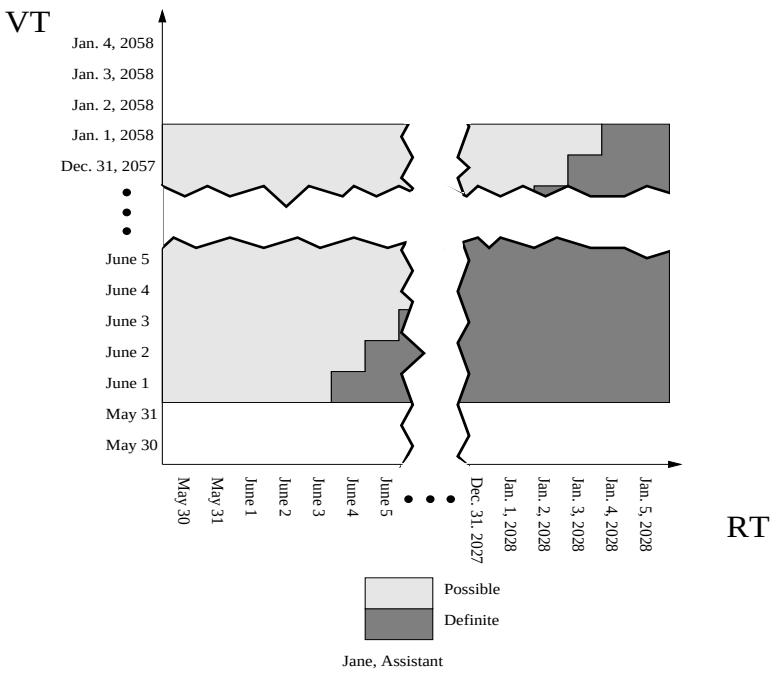


Figure 16: A graphical representation of Jane's possible and actual employment

4.5 Summary of Extensionalizations

Table 1 summarizes some of the valid-time extensionalizations (the most representative cases). Case **v1** (the **v** stands for "valid-time" database) specifies the extensionalization of tuple timestamped with a determinate interval, case **v2** a now-relative interval, case **v3** an indeterminate interval, and case **v4** a now-relative indeterminate interval. Note that the possible and definite extensionalizations in cases **v1** and **v2** are the same since the intervals are determinate.

	Variable Database	**Extensional Database**
v1	$T =< X, [vt_1, vt_2] >$	$[\![T]\!]_{rt_*} = \{(X, vt, rt_*) \mid vt_1 \leq vt \leq vt_2\}$
v2	$T =< X, [vt_1, now \pm n \text{ days }] >$	$[\![T]\!]_{rt_*} = \{(X, vt, rt_*) \mid vt_1 \leq vt \leq rt_* \pm n\}$
$\mathbf{v3}^D$	$T =< X, [vt_1 \sim vt_2, vt_3 \sim vt_4] >$	$[\![T]\!]^D_{rt_*} = \{(X, vt, rt_*) \mid vt_2 \leq vt \leq vt_3\}$
$\mathbf{v3}^P$	$T =< X, [vt_1 \sim vt_2, vt_3 \sim vt_4] >$	$[\![T]\!]^P_{rt_*} = \{(X, vt, rt_*) \mid vt_1 \leq vt \leq vt_4\}$
$\mathbf{v4}^D$	$T =< X, [vt_1, now \pm n \text{ days } \sim vt_2] >$	$[\![T]\!]^D_{rt_*} = \{(X, vt, rt_*) \mid vt_1 \leq vt \leq min(rt_* \pm n, vt_2)\}$
$\mathbf{v4}^P$	$T =< X, [vt_1, now \pm n \text{ days } \sim vt_2] >$	$[\![T]\!]^P_{rt_*} = \{(X, vt, rt_*) \mid vt_1 \leq vt \leq vt_2\}$

Table 1: Extensionalization of valid-time databases.

4.6 Querying Variable Valid-time Databases

Thus far we have presented a framework for defining the meaning of database instances where the timestamps of tuples may include a current-time relative variable. Further, we have defined the semantics of variable valid-time databases that use all the existing and new timestamps. In the process, we have demonstrated how the framework is used.

The next step is to enhance the query facilities of existing (non-variable) data models to support queries on timestamps containing variables. The essential problem is what to do when encountering

a variable during query evaluation. Below, we describe a solution to that problem. Further, we show how the framework may be utilized in defining algebraic operators on variable databases that are consistent with the semantics of variable databases. Specifically, we consider the valid-time timeslice operation.

When evaluating a user-level query, e.g., written in some dialect of SQL, it is common to transform it into an internal algebraic form that is suitable for subsequent rule or cost-based query optimization. As the query processor and optimizer are among the most complex components of a database management system, it is important that the added functionality of current-time-related timestamps necessitates only minimal changes to these components.

While many solutions may be envisioned, a solution that meets this requirement and is natural in our semantic framework is to eliminate variables before they are seen. More specifically, when a timestamp that contains a variable is used during query processing (e.g., in a test for overlap with another timestamp), a ground version of that timestamp is created and is used instead. Thus, only minimal, incremental changes to the query processor are needed. Existing components remain unchanged. Only a new component that substitutes variable timestamps with ground timestamps has to be added.

More specifically, we define a "bind" operator that is added to the set of operators already present. When user-level queries are mapped to the internal representation, this operator will be utilized. Bind accepts any valid-time tuple with variables as defined earlier in the paper. It substitutes a ground value for each variable and thus returns a ground(but still variable-level) tuple. The bind operator is defined below.

Definition 4.11 [Variable-level Valid-time Bind]
Given an arbitrary valid-time tuple $T =< X, [e_1 \sim vt_2, e_3 \sim vt_4] >$ and a reference time rt_*, the variable-level valid-time bind operation eliminates all variables is defined as follows.

$$bind^{V,VT}_{rt_*}(T) =_{df} < X, [\langle\langle e_1 \rangle\rangle_{rt_*} \sim \langle\langle vt_2 \rangle\rangle_{rt_*}, \langle\langle e_3 \rangle\rangle_{rt_*} \sim \langle\langle vt_4 \rangle\rangle_{rt_*}] >$$ ∎

This operation can be extended in the obvious way to an operator on sets of tuples, i.e., relations. In the superscript, "^{V,VT}," the "^{V}" indicates that this is a variable-level operator and the "^{VT}" signifies that it is also a valid-time operator. Note that two tuples that have timestamps "$[vt_1 \sim vt_1, vt_2 \sim vt_2]$" and "$[vt_1, vt_2]$," but are otherwise identical, have the same extensionalizations. Thus the timestamps are equivalent, and therefore the definition above also covers determinate timestamps.

The outcome of a query on a variable database generally depends on the specific reference-time argument given to the *bind* operator. To provide a foundation for understanding how to use the *bind* operator when mapping user-level queries to algebraic equivalents, we must explore its meaning.

The *bind* operator with reference-time argument rt_* replaces each variable by its denotation or value at time rt_*. Put differently, the operator replaces each variable timestamp with a ground timestamp that has the special property of having the same denotation, or value, as the variable timestamp at the reference time rt_*. At other reference times, the original and the ground timestamps will generally not have the same denotation. This semantics may be expressed at the extensional level as follows.

Definition 4.12 [Extensional-Level Valid-time Bind]
Given an arbitrary set S of extensional-level valid-time tuples of the form (X, vt, rt) and a reference time rt_*, the extensional-level valid-time bind operation is defined as follows.

$$bind^{E,VT}_{rt_*}(S) =_{df} \{(X, vt, rt) \mid (X, vt, rt_*) \in S \ \wedge \ rt \in \mathcal{T}_{RT}\}$$ ∎

The "^{E}" in the operator's superscript indicates that this is an extensional-level operator. At the extensional level, the bind operator chooses the meaning of a tuple at the indicated reference time and propagates that meaning over every possible reference time, resulting in a reference-time invariant meaning. To prove that this definition is correct, we need to show that given a tuple T, and a reference time rt_*, $[\![bind^{V,VT}_{rt_*}(T)]\!] = bind^{E,VT}_{rt_*}([\![T]\!])$. This follows directly from the definitions. For brevity, we omit the proof.

Intuitively, the *bind* operator sets the perspective of the observer, i.e., it sets the the reference time as described in Section 3.2. Existing query languages generally assume that the perspective of a user observing the database is the same as what we termed the query time or current transaction time and denoted $t_{current}$ in that section. However, as we shall see, a bind operator provides a basis for added functionality.

Recall that the definition of query operators at the variable level is complex and that current temporal data models have not satisfactorily resolved the complex problems involved. In our approach, we first preprocess the variable-level database by binding timestamps to rt_*, effectively removing the variables. We can then apply any algebraic operators from an existing temporal query language. It should be clear from the discussion above that the composition of bind with any of these algebraic operators is well-defined, and the timestamps have the appropriate meaning.

To explore in more detail the interaction among the valid timeslice time, query time, and reference time of a query, and to show how operators are defined within the semantic framework, we now define several valid-time timeslice operators.

Valid-time timeslice is a fairly standard operation; some variant of timeslice is a component of virtually all temporal algebras. Standard definitions of determinate and indeterminate timeslice operators are given below. Note that these do not have to contend with variables; because of the use of the bind operator, they can be defined solely on ground tuples.

Definition 4.13 [Variable-level Definite Valid-time Timeslice]
Let S be a set of tuples at the variable database level, i.e., a set of tuples of the form $T =< X, [vt_1 \sim vt_2, vt_3 \sim vt_4] >$, where the vt_i are ground values. The definite valid-time timeslice of S at valid time vt_* is defined as follows.

$$\Pi^{D,V,VT}_{vt_*}(S) =_{df} \{< X, [vt_*, vt_*] > \mid \exists\, T =< X, [vt_1 \sim vt_2, vt_3 \sim vt_4] > \in S\ (vt_* \in [vt_2, vt_3])\} \quad \blacksquare$$

Definition 4.14 [Variable-level Possible Valid-time Timeslice]
Let S be a set of tuples at the variable database level, i.e., a set of tuples of the form $T =< X, [vt_1 \sim vt_2, vt_3 \sim vt_4] >$, where the vt_i are ground values. The possible valid-time timeslice of S at valid time vt_* is defined as follows.

$$\Pi^{P,V,VT}_{vt_*}(S) =_{df} \{< X, [vt_*, vt_*] > \mid \exists\, T =< X, [vt_1 \sim vt_2, vt_3 \sim vt_4] > \in S\ (vt_* \in [vt_1, vt_4])\} \quad \blacksquare$$

The superscript "^{D,V,VT}" of the first operator indicates that it considers only the definite information contents, that it belongs at the variable level, and that it is a valid-time timeslice. Also, recall that definite timestamps are special cases of indeterminate timestamps, which are then also covered by the definition. The straightforward extensions of the operator to slice on valid-time intervals and to take as input a set of tuples (i.e., a relation), are omitted for brevity. A timeslice operator at the extensional level that satisfies the correctness criterion of the framework, specifically, $\Pi^{E,VT}_{vt_*}([\![T]\!]^D) = [\![\Pi^{D,V,VT}_{vt_*}(T)]\!]$, is given next. The proof of this statement is omitted for space considerations.

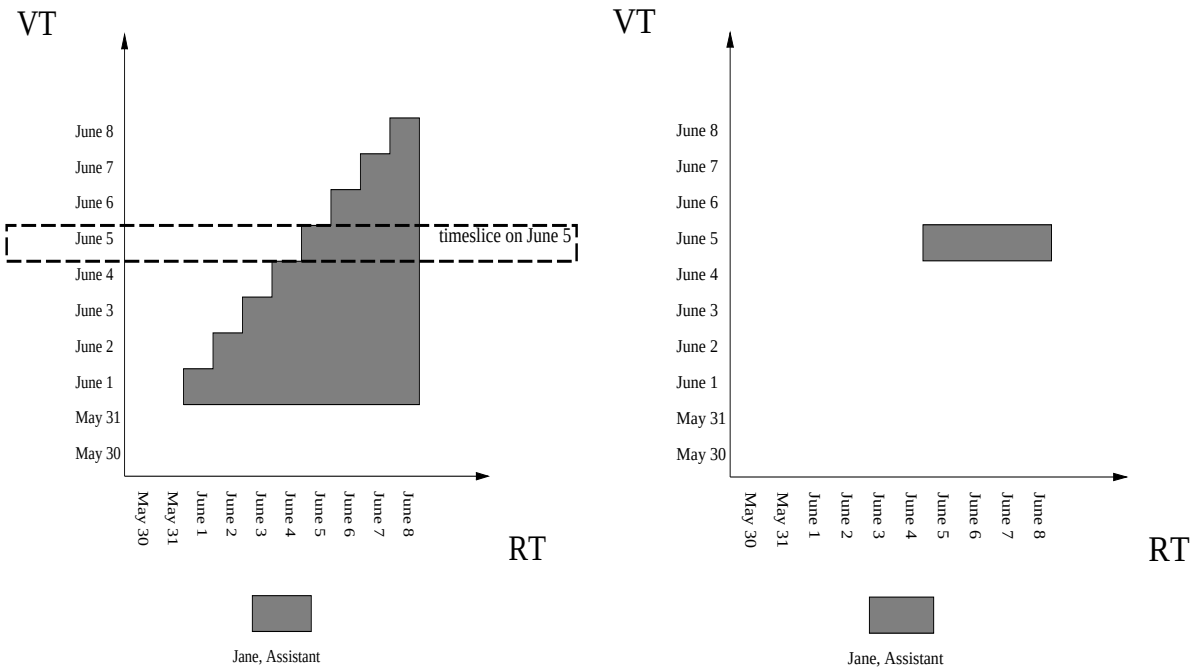
Definition 4.15 [Extensional-level Valid-time Timeslice]
At the extensional database level, the valid-time timeslice of a set S consisting of tuples of the form (X, vt, rt) is defined as follows.

$$\Pi^{E,VT}_{vt_*}(S) \ =_{df} \ \{(X, vt_*, rt) \mid (X, vt_*, rt) \ \in \ S\}$$ ∎

Note that once the extensionalization mapping has been applied to a variable-level tuple, there is no indeterminacy and thus only one extensional-level timeslice operator.

We are now in a position to explore the interaction of the important times (Section 3.2) by using the new *bind* operator and existing (variable-level) timeslice operators. We will generally consider only the definite version of the timeslice operator.

As an aside, observe that we cannot formulate the query to select employees from the faculty relation who are employed on June 5 using only timeslice: $\Pi^{D,VT}_{June\ 5}(Faculty)$. With variables present in timestamps, the operator is not well-defined. Consider the tuple $T =< Jane,\ Assistant,\ [June\ 1,\ now] >$. Whether or not Jane is in the result depends on the perspective of the queryer. The impact of perspective in query evaluation is graphically illustrated in Figure 17. The left panel of Figure 17 shows the extensionalization of T for reference times May 30 through June 8. In the figure, the region chosen by a valid-time timeslice of June 5 is enclosed in dashed lines. The (collective) result of the timeslice, from multiple perspectives, from May 30 until June 8, is shown in the right panel of Figure 17. It is clear that Jane will not be in the result for perspectives with a reference time before June 5. Hence, the timeslice operator alone in insufficient.



a) A timeslice on Jane's employment b) The result of the slice

Figure 17: A graphical representation of a timeslice at June 5 on Jane's employment

The *bind* operator sets the perspective and is combined with timeslice to formulate queries. In the first three example queries given below, we assume that the database is to be observed from the perspective of June 5. For all examples, the query time is assumed to also be June 5.

- Who is employed on June 5?

$$\Pi^{D,V,VT}_{June\ 5}(bind^{V,VT}_{June\ 5}(Faculty))$$

- Who was employed on June 3?

$$\Pi^{D,V,VT}_{June\ 3}(bind^{V,VT}_{June\ 5}(Faculty))$$

- Who will actually be employed on June 7?

$$\Pi^{D,V,VT}_{June\ 7}(bind^{V,VT}_{June\ 5}(Faculty))$$

  Tuples with a "to" time of *now* will *not* be in the result.

- Who do we expect to be employed on June 7?

$$\Pi^{D,V,VT}_{June\ 7}(bind^{V,VT}_{June\ 7}(Faculty))$$

  The best we can do to answer the query is to adopt a June 7 perspective of the database and assume that there are no changes to the database between the current time and June 7. Then all tuples with a "to" time of *now* will contribute to the result, since we "expect" such employees to remain employed.

- Who will possibly be employed on June 7?

$$\Pi^{P,V,VT}_{June\ 7}(bind^{V,VT}_{June\ 5}(Faculty))$$

  We adopt a June 5 perspective and query about a possible future from that perspective. Tuples with intervals with a "to" time of $now \sim June\ 7$ (or a later upper bound) will be in the result, although those with a "to" time of *now* will not be in the result.

- Who will definitely be employed on June 7?

$$\Pi^{D,V,VT}_{June\ 7}(bind^{V,VT}_{June\ 5}(Faculty))$$

  Intervals with a terminating time of $now \sim June\ 7$ (or later) will *not* be in the result.

We have seen that the binding of *now* impacts the meaning of query results and that query results must be interpreted with respect to a particular perspective. Existing query languages, e.g., TSQL2 [SAA+94], generally assume that the perspective and the query time coincide. This assumption leads to a restriction in functionality, but it also simplifies the interpretation of answers.

5 Transaction-time Databases

The use of a current-time variable in the transaction-time dimension is not as fraught with problems as its use in the valid-time dimension. The reason for this lies in the different meaning of transaction time in a database. The valid time of a tuple indicates when it is considered valid, and, as such, valid timestamps of tuples are generally provided by the users. In contrast, transaction timestamps are supplied by the database management system itself. This is a consequence of the meaning of transaction time: the transaction timestamp indicates when the tuple is current in the database.

Although several variable names, e.g., *forever* and *now*, have been used, it is our contention that they all have the same meaning. Specifically, they are all employed as a "stop" timestamp that indicates that the tuple stamped is current (from the "start" time) until the database is updated to indicate otherwise. However, the various names used do not convey the intuitive semantics of the variable in this dimension. A term more precise than *now* or *forever* for this meaning of "not yet logically deleted or updated" is *until changed*—a fact is current in the database until changed. It has no counterpart in valid time. Using *until changed* instead of *now* avoids also potential confusion with *now* in valid time, although some authors have used *until changed* in valid time [WJL91, WJL93]. Unlike the (valid-time variable) *now*, *until changed* can only be used as the "stop" time; it is meaningless to use it as the "start" time.

5.1 Extensionalization of a Ground Transaction-time Tuple

We first examine the meaning of a tuple without variables in transaction time. The extensionalization of a transaction-time tuple without variables differs from its valid-time counterpart, because the semantics of transaction time does not allow future transaction times to be predicted in the database. Hence, the extensionalization of such tuples must be restricted to ensure that no matter when we look at the database, we can never see a "future" transaction time. Since the future depends on when we observe the database, the reference time is used to constrain the transaction-time in the expanded set of tuples.

In Definition 4.1, the denotation at any reference time of a ground valid-time instant was given to be the instant itself. The same applies to ground transaction-time instants.

Definition 5.1 [Transaction-time Extensionalization of a Ground Tuple]
The transaction-time extensionalization of a tuple of the form $T =< X, [tt_1, tt_2] >$, where X is some set of attribute values and tt_1 and tt_2 are transaction-time instants, at the reference time rt_*, where $tt_0 \leq rt_* \leq t_{current}$, is defined as follows.

$$[\![T]\!]^{TT}_{rt_*} =_{df} \{(X, tt, rt_*) \mid tt \in [\langle\!\langle tt_1 \rangle\!\rangle_{rt_*}, \langle\!\langle tt_2 \rangle\!\rangle_{rt_*}]\}$$ ∎

We use a ^{TT} superscript to differentiate this mapping from a valid-time extensionalization.

The visualization of a transaction-time tuple is similar to that of a valid-time tuple. Again, a two-dimensional graph is used. The X-axis of the graph is the reference time, while the Y-axis is the transaction time. However, unlike a valid-time tuple without variables, the transaction-time interval for a tuple is not independent of the time at which we observe the tuple. Figure 18 depicts the extensionalization of the transaction-time tuple $< Jane, Assistant, [June\ 5, June\ 8] >$ for a sequence of reference times, June 1 through June 11. Note that the depicted region has a "stair shaped" feature which is a result of the constraint that the transaction time cannot exceed the reference time.

5.2 Semantics of "Until Changed"

The current-time variable in transaction time indicates that the associated fact is current in the database until the fact is changed by a subsequent update. Substituting transaction time for valid time in our running example yields the relation shown in Figure 19.

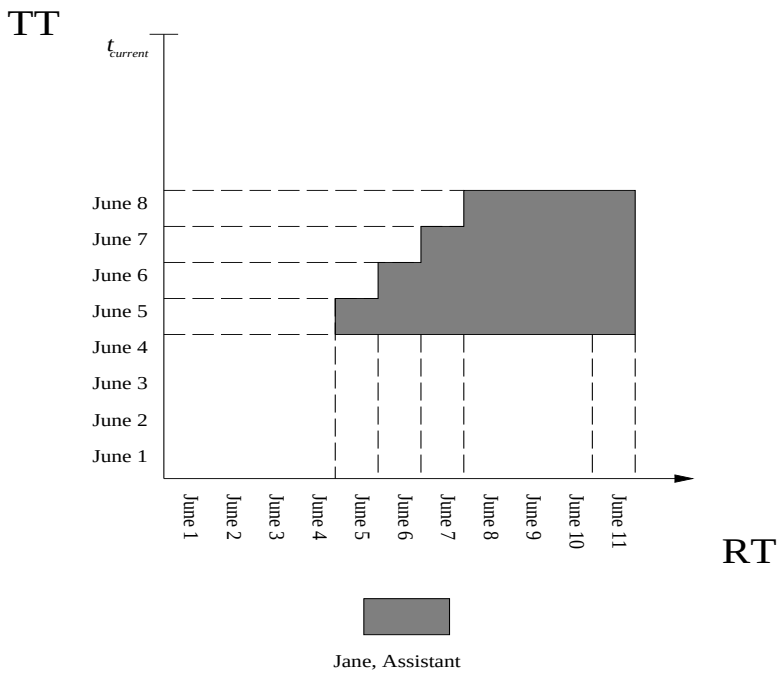


Figure 18: A graphical representation of a transaction-time tuple

FACULTY			
		TRANS TIME	
NAME	RANK	(start)	(stop)
Jane	*Assistant*	*June 1*	*until changed*

Figure 19: Using until changed in a transaction-time relation

Definition 5.2 [Denotation of *Until Changed*]
The *denotation* of the transaction-time variable *until changed* at a particular reference time rt_*, where $tt_0 \leq rt_* \leq t_{current}$, is defined as follows.

$$\langle\langle until\ changed \rangle\rangle_{rt_*} =_{df} rt_* \qquad \blacksquare$$

The extensionalization of a transaction-time tuple with the variable *until changed* as the value of its "stop" time is obtained by generating tuples for each instant in the ground interval that results from substituting *until changed* by rt_*. Thus, Definition 5.1 also applies when *until changed* is allowed as a "stop" time.

To exemplify, recall that even when the database itself does not change at time rt_*, its extensional interpretation might be different. For example, consider the tuple

$$T =< Jane, Assistant, [June\ 5, until\ changed] > .$$

The timestamp specifies that the fact that Jane is an Assistant Professor was added to the database on June 5 and has yet to be deleted or updated. The extensionalization at reference time June 5 is

$$\{(Jane, Assistant, June\ 5, June\ 5)\}.$$

Similarly, the extensionalization at reference time June 6 is

$$\{(Jane, Assistant, June\ 5, June\ 6), (Jane, Assistant, June\ 6, June\ 6)\}.$$

Hence, the semantics of the same tuple depends on when it is observed.

The visualization of a tuple with *until changed* is similar to the visualization of a tuple without that variable discussed in Section 5.1. Figure 20 depicts the extensionalization of the tuple $< Jane, Assistant, [June\ 5, until\ changed] >$ for a sequence of reference times, June 1 through June 11. The only difference is that the stair step continues until a reference time of $t_{current}$.

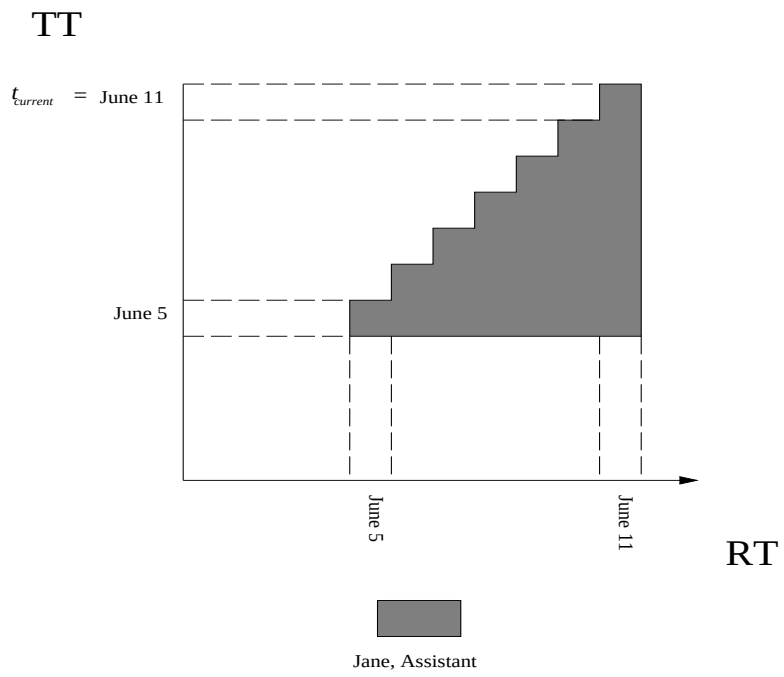


Figure 20: A graphical representation *until changed*

Table 2 summarizes the extensionalizations presented for transaction time. Case **t1** (the **t** stands for "transaction-time" database) applies to tuples with fully ground timestamp values only, whereas Case **t2** covers the case where *until changed* is the "stop" time.

	Variable Database	**Extensional Database**
t1	$T =< X, [tt_1, tt_2] >$	$[\![T]\!]_{rt_*} = \{(X, tt, rt_*) \mid tt_1 \leq tt \leq min(tt_2, rt_*)\}$
t2	$T =< X, [tt_1, until\ changed] >$	$[\![T]\!]_{rt_*} = \{(X, tt, rt_*) \mid tt_1 \leq tt \leq rt_*\}$

Table 2: Extensionalization of transaction-time databases

5.3 Querying Variable Transaction-time Databases

Section 4.6 illustrated how variable valid-time databases may be integrated into data models that previously permitted only ground databases. The emphasis was on the querying of variable databases. Although this section considers variable transaction-time databases, it has the same purpose and is thus similar in structure. As we shall see, the different semantics of transaction time, including the absence of indeterminacy, leads to different, simpler definitions.

The "bind" operator for transaction time eliminates occurrences of *until changed* in the "stop" component of timestamps.

Definition 5.3 [Variable-level Transaction-time Bind]
Given a tuple $T =< X, [tt_1, e_2] >$, where tt_1 is a ground transaction time and e_2 is *until changed* or a ground transaction time, the variable-level transaction-time bind operation is defined as follows.

$$bind^{V,TT}(T) =_{df} < X, [\langle\langle tt_1 \rangle\rangle_{t_{current}}, \langle\langle e_2 \rangle\rangle_{t_{current}}] > \quad \blacksquare$$

Again, this operation can be extended in the obvious way to relations. Transaction-time *bind* is very similar to valid-time bind, but differs in one important respect. The $bind^{V,TT}$ operator does not accept any time argument, but *always* binds *until changed* to the query time or current transaction time, $t_{current}$. This is because the transaction time of a tuple specifies which database state(s) it belongs to in the trajectory of database states. A tuple with the timestamp of [tt_1, *until changed*] belongs to all of the database states from tt_1 to the current state. Binding *until changed* to a time other than the $t_{current}$ could result in an incorrect trajectory since the tuple might not be in states in which it should be. The transaction-time timeslice operator (defined below) is used to project the states that are of interest in a query.

Since the $bind^{V,TT}$ operator lacks a time parameter and is always applied before any other operator, it is feasible to omit the operator and instead build it into the transaction timeslice operator, as has been done in some variable-level transaction-time algebras [JM92]. However, it would also need to be built into any additional operators, so to preserve the parallel with Section 4.6, we choose not to do this. The definition of the extensional-level bind for transaction time is omitted because it is very similar to Definition 4.12.

The standard transaction-time timeslice operator defined next selects tuples with a transaction time that overlaps the time instant tt_*, i.e., all information that was current at time tt_*, and restricts the timestamps of qualifying tuples to tt_*.

Definition 5.4 [Variable-level Transaction-time Timeslice]
Let S be a set of tuples at the variable database level, i.e., a set of tuples of the form $T =< X, [tt_1, tt_2] >$, where tt_1 and tt_2 are ground transaction times. The transaction-time timeslice of S at transaction-time tt_* is defined as follows.

$$\Pi^{V,TT}_{tt_*}(S) \ =_{df} \ \{< X, [tt_*, tt_*] > \mid \exists\, T = < X, [tt_1, tt_2] > \in S \ (tt_* \in [tt_1, tt_2])\} \quad \blacksquare$$

Definition 5.5 [Extensional-level Transaction-time Timeslice]
At the extensional database level, the transaction-time timeslice of a set S consisting of tuples of the form (X, tt, rt) is defined as follows.

$$\Pi^{E,TT}_{tt_*}(S) \ =_{df} \ \{(X, tt_*, rt) \mid (X, tt_*, rt) \ \in \ S\} \quad \blacksquare$$

As with valid-time queries, a combination of bind and timeslice supports transaction-time queries. When asking queries about a transaction-time database, there are two important times to consider: (i) the transaction-time timeslice time, tt_*, indicating that information is sought that was current in the database at time tt_*, and (ii) the query time, $t_{current}$, the time at which the query is asked.

As a first example, we consider several timeslice operations on the tuple

$$T =< Jane, Assistant, [June\ 5, June\ 8] > .$$

The extensionalization of T is depicted in Figure 18 on page 31. For the following queries, it is assume that $t_{current}$ is June 11.

- $\Pi^{V,TT}_{June\ 11}(bind^{V,TT}(T))$ yields an empty result because the interval associated with T is before the timeslice time—tuple T ceased to be current starting on June 9and was not current on June 11.
- $\Pi^{V,TT}_{June\ 7}(bind^{V,TT}(T))$ yields tuple T, but with "start" and "stop" times of June 7. This is so because the information recorded by T was current on June 7.
- $\Pi^{V,TT}_{June\ 1}(bind^{V,TT}(T))$ results in an empty result, since the timeslice instant of June 1 is before the "start" time of the tuple, June 5. Put differently, the query requests information that was current in the database prior to when T was inserted into the database.

The following queries further illustrate the use of the timeslice time in transaction-time queries; we again assume that $t_{current}$ is June 11.

- According to the current state, who is employed?

$$\Pi^{V,TT}_{June\ 11}(bind^{V,TT}(Faculty))$$

- On June 6, who did the database record as being employed?

$$\Pi^{V,TT}_{June\ 6}(bind^{V,TT}(Faculty))$$

  In this query, the timeslice time is June 6, since the query is asks for the information current on June 6.

6 Bitemporal Databases

A *bitemporal* relation supports both transaction and valid time [CI93, JCE+94, SA85]. The combination of these two temporal dimensions empowers the database to record time-dependent information as well as earlier database states. Bitemporal databases thus combine the advantages of valid-time and transaction-time databases. Yet, this greater flexibility comes at a cost: increased complexity derives from the interactions between the two temporal dimensions which must be carefully considered. However, as we will see below, the logical framework that we have presented here for expressing the semantics of current-time variables has been designed to make it relatively straightforward to obtain the semantics of bitemporal databases. The interaction between the the current-time variable for valid time, *now*, and transaction time, *until changed*, is coordinated through the reference time. We will demonstrate *one* possible (and, we think, reasonable) semantics for this combination, but we emphasize that the framework is general enough to allow the definition of other, alternative semantics for these variables.

Below we first examine the extensionalization mapping for bitemporal databases. We then explore pictorial representations for bitemporal databases and conclude with a discussion of queries and their evaluation.

6.1 Extensionalization of Bitemporal Databases

The timestamp of a bitemporal tuple contains both a valid-time and a transaction-time component. Since the valid-time component may be indeterminate, it is necessary to distinguish between a definite and a possible extensionalization, $[\![\ ]\!]^{BT,D}_{rt_*}$ and $[\![\ ]\!]^{BT,P}_{rt_*}$, respectively.

Definition 6.1 [Definite Extensionalization of a Bitemporal Tuple]
The definite extensionalization of a bitemporal tuple T of the form $T =< X, [vt_1, vt_2], [tt_1, tt_2] >$, where X is some set of attribute values and the timestamp $[vt_1, vt_2], [tt_1, tt_2]$ may contain any of the variables introduced earlier, at the reference time rt_* is defined as follows.

$$[\![T]\!]^{BT,D}_{rt_*} =_{df}$$
$$\{(X, vt, tt, rt_*) \mid (X, vt, rt_*) \in [\![< X, [vt_1, vt_2] >]\!]^{VT,D}_{rt_*} \wedge (X, tt, rt_*) \in [\![< X, [tt_1, tt_2] >]\!]^{TT}_{rt_*}\}$$ ∎

Definition 6.2 [Possible Extensionalization of a Bitemporal Tuple]
The possible extensionalization of a bitemporal tuple T of the form $T =< X, [vt_1, vt_2], [tt_1, tt_2] >$ at the reference time rt_* is defined as follows.

$$[\![T]\!]^{BT,D}_{rt_*} =_{df}$$
$$\{(X, vt, tt, rt_*) \mid (X, vt, rt_*) \in [\![< X, [vt_1, vt_2] >]\!]^{VT,P}_{rt_*} \wedge (X, tt, rt_*) \in [\![< X, [tt_1, tt_2] >]\!]^{TT}_{rt_*}\}$$ ∎

The definitions show that the framework has been constructed so that the extensionalization of bitemporal tuples is the combination of the extensionalizations for valid and transaction time. It also shows how the reference time rt_* serves as an essential coordination mechanism between the valid and transaction time components of the timestamp: the same reference time appears in the valid-time and in the transaction-time denotations. Although, it is possible and may be interesting to consider situations where the two reference times differ, we have found that for all practical purposes this coordination is desirable. Nevertheless, other kinds of coordination through the reference time are possible. For example, instead of the standard Cartesian product used here, a coordination mechanism that utilizes a step-wise cross product of the two temporal dimensions is possible [CI94].

Another feature of the framework is that the uniform and component-wise treatment of time dimensions makes it easy to include additional dimensions. To specify the semantics of a variable database with additional dimensions, it is necessary to first specify the semantics of the variables and tuples in that new dimension, e.g., as is done for the transaction-time dimension in Section 5. Subsequently, the new dimension can be easily integrated with the other dimensions in a definition similar to the one above. Thus, our framework can be extended to encompass multi-dimensional temporal databases, for example *temporally generalized* [JS94], *indexical* [Cli93], *parametric* [GN93] and *spatio-temporal* [ASS94] databases.

Tables 1 and 2 may be combined to cover the bitemporal extensionalizations. The combination of Case **v1** from Table 1 and Case **t1** from Table 2 gives the bitemporal extensionalization for a tuple timestamped with a determinate valid time interval, $[vt_1, vt_2]$, and a transaction time interval, $[tt_1, tt_2]$, both without variables. Note that the transaction time in this case is restricted to the "past" relative to the reference time, just as in transaction-time tuples. For example, the extensionalization at reference time June 2 of the tuple

$$< Jane, Assistant, [June\ 3, June\ 10], [June\ 1, June\ 3] >$$

is

$$\{(Jane, Assistant, vt, tt, June\ 2) \mid vt \in [June\ 3, June\ 10] \wedge tt \in [June\ 1, \min(June\ 2, June\ 3)]\}.$$

In this example, the terminating transaction time, June 3, is constrained by the reference time, June 2.

The graphical representation of bitemporal tuples is also a combination of the valid time and transaction time graphs. The representation is three-dimensional; transaction time is the X-axis, valid time is the Y-axis, and reference time is the Z-axis. To this point, the reference time has been the X-axis, but making the reference time the Z-axis in the three-dimensional visualization results in a better picture. The graph is displayed so that the Z-axis goes "into" the page.

A three-dimensional picture of bitemporal tuples allows us to represent the passage of time as a spatial displacement, and provides a visual representation for interesting phenomena such as history changes and predictions about the future, as well as incorporating the viewpoint of an observer into these phenomena. It provides a uniform representation of both transaction time and valid time. This makes explicit the homogeneous nature of these entities and enables a uniform treatment of both time dimensions. As we will see below, the graphical representation shows the subtle interaction between *now*, *until changed*, and the reference time. Variations of these graphs have been independently explored [JSS94, JS92, CI94].

As an example, assume that the academic career of Jane at State University is given by the tuple

$$< Jane, Assistant, [June\ 3, June\ 5], [June\ 1, June\ 4] > .$$

Here the fact that Jane was an Assistant Professor from June 3 to June 5 was recorded in the database on June 1 and logically deleted on June 4, as it was found to be in error. Figure 21 depicts the extensionalization of this tuple over a sequence of reference times, from June 1 through June 11.

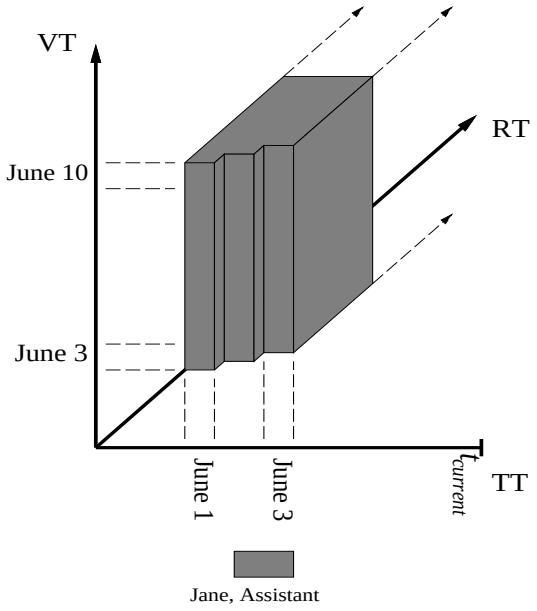


Figure 21: A graphical representation of Jane's employment.

The combinations of the extensionalizations presented in Tables 1 and 2 are graphically depicted in Figures 22 and 23. The dotted line vectors in the graph represent directions of growth as either the reference time, valid time, or transaction time extends to $\top$. The evolutionary nature of temporal databases, a key concept, comes through very clearly in the figures. Notice how the shaded areas grow as reference time increases, most prominently for tuples containing variables, indicating an

accumulation of knowledge stored in the database. Note also how information in later reference times is always consistent with that in earlier reference times.

Figure 22 illustrates the determinate cases. For example, the lower right corner of Figure 22 depicting the **v2**$\times$ **t2** case, shows how *now* and *until changed* are bound to an increasing reference time, resulting in a three-dimensional stair-shaped pattern. The tuple's extensionalization grows as time passes encompassing more points. In contrast, case **v1** $\times$ **t1** depicts constrained growth, as the tuple ceases to exist beyond transaction time tt_2. Note that, unless a tuple is known to have been deleted from the database, its "transaction-stop time" is *until changed*, and hence it has unlimited growth in the transaction-time dimension. This is true for the determinate cases shown in Figure 22 as well as for the indeterminate cases of Figure 23. Notice for example, how the possible and definite extensionalizations in cases $\mathbf{v3}^D \times \mathbf{t2}$ and $\mathbf{v3}^P \times \mathbf{t2}$, the upper right-hand corner of Figure 23, remain constant in the valid-time dimension while growing in transaction-time. In contrast, case $\mathbf{v4}^D \times \mathbf{t2}$ illustrates constrained growth, i.e., constant evolution up through time vt_2.

6.2 Querying Variable Bitemporal Databases

In Sections 4.6 and 5.3, we illustrated how variable valid-time and transaction-time databases, respectively, were integrated into data models that have yet to allow the use of current-time variables in timestamps. We also explored the querying of variable databases, with an emphasis on the interaction of the important times mentioned in Section 3.2.

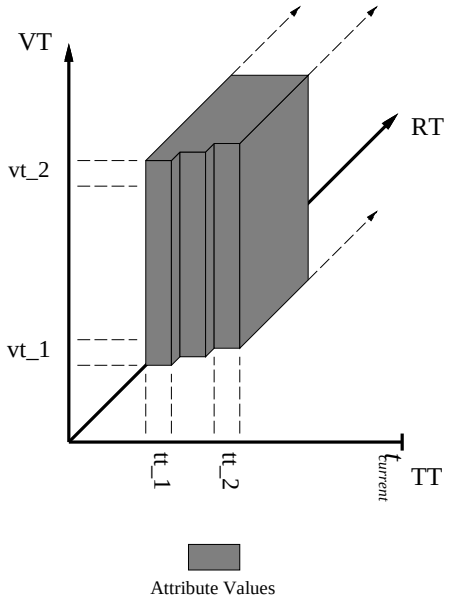
This section has the same agenda as those two previous sections. In analogy with those sections, we first to define "bind" and timeslice operators for bitemporal relations. However, the existing bind and timeslice operators, developed for valid-time and transaction-time databases, are easily generalized to apply to bitemporal databases. So rather than present new formal definitions for the bitemporal operators, we indicate how the existing operators are generalized and subsequently utilize these generalized operators. A bitemporal tuple differs from a valid-time tuple by having a transaction time interval in its timestamp. The valid-time operators are generalized to corresponding bitemporal operators by simply ignoring this extra timestamp. For example, the definite bitemporal valid-time timeslice is defined by generalizing Definition 4.13 as follows.

Definition 6.3 [Variable-level Definite Bitemporal Valid-time Timeslice]
Let S be a set of tuples at the variable database level, i.e., a set of tuples of the form $T =< X, [vt_1 \sim vt_2, vt_3 \sim vt_4], [tt_1, tt_2] >$, where T is ground. The definite bitemporal valid-time timeslice of S at valid time vt_* is defined as follows.

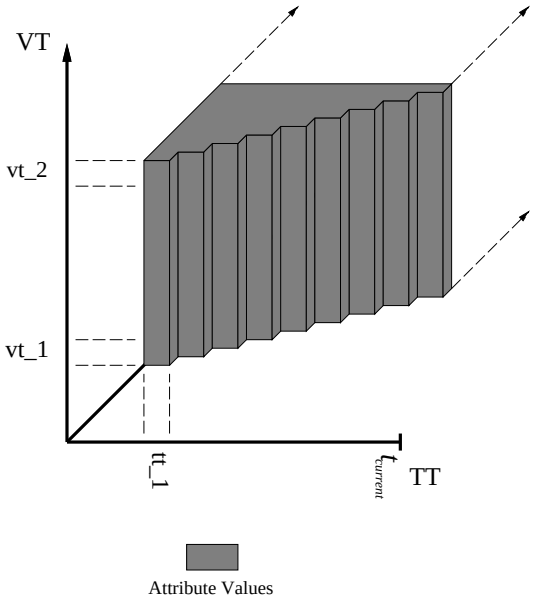
$$\Pi^{D,V,VT,BT}_{vt_*}(S) =_{df}$$
$$\{< X, [vt_*, vt_*], [tt_1, tt_2] > \mid \exists\, T =< X, [vt_1 \sim vt_2, vt_3 \sim vt_4], [tt_1, tt_2] > \in S\ (vt_* \in [vt_2, vt_3])\}\ \blacksquare$$

The superscript "D,V,VT,BT" indicates that the operator considers only the definite information in the tuple, belongs at the variable level, performs a timeslice in the valid-time dimension, and is applicable to bitemporal tuples. In addition to this operator, the subsequent discussion will use the operators $\Pi^{P,V,VT,BT}$, $\Pi^{V,TT,BT}$, $bind^{V,VT,BT}$, and $bind^{V,TT,BT}$ which are all similar generalizations of previous definitions.

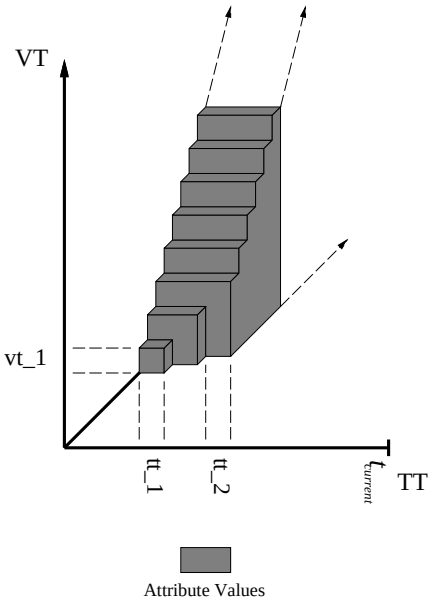
As with valid-time and transaction-time databases, queries are evaluated by combining bitemporal timeslice and bind operations. Also as before, valid and transaction times must be bound before the bitemporal valid-time timeslice or bitemporal transaction-time timeslice, respectively, can be applied.



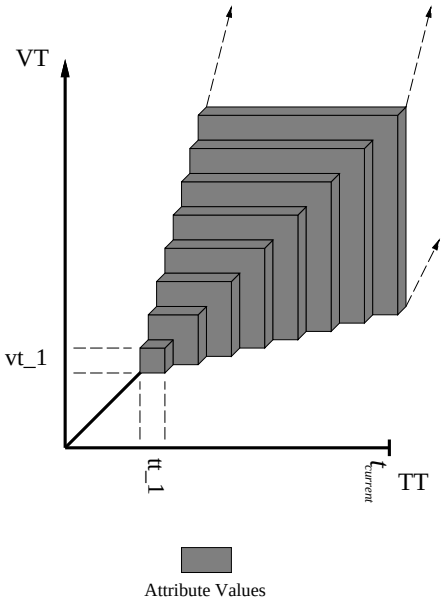
Case **v1** × **t1**



Case **v1** × **t2**

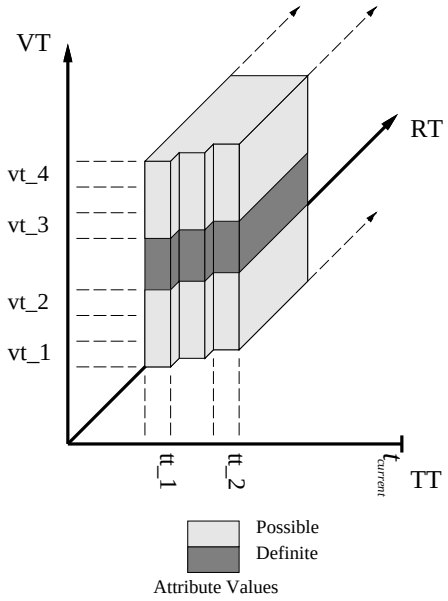


Case **v2** × **t1**

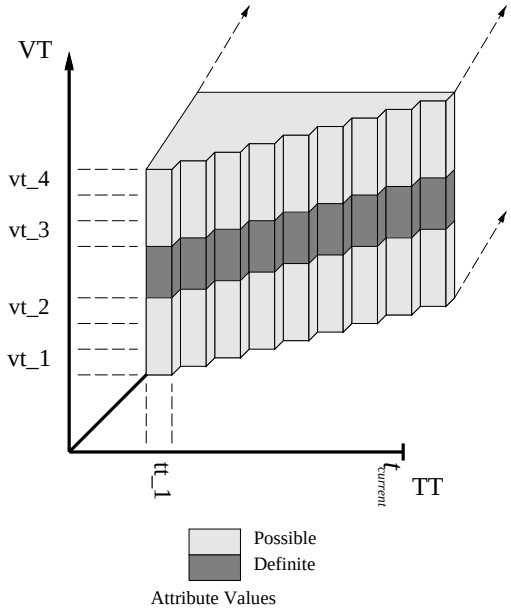


Case **v2** × **t2**

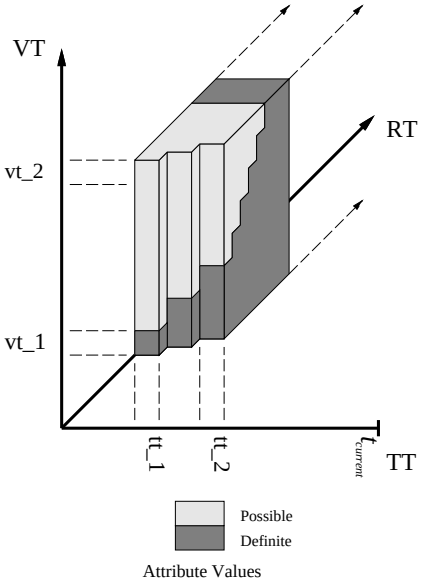
Figure 22: The bitemporal determinate cases



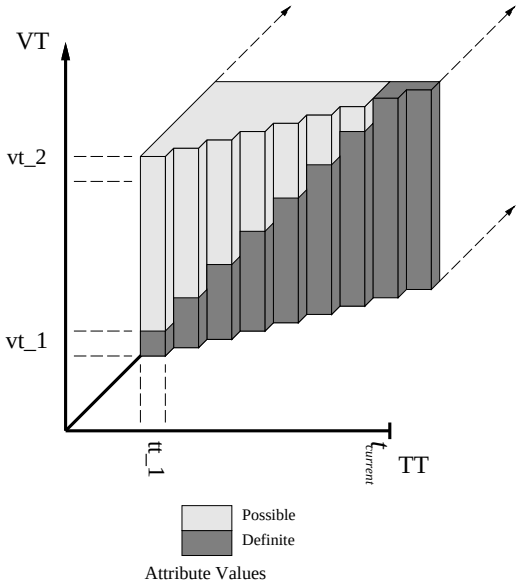
Cases $\mathbf{v3}^D \times \mathbf{t1}$ and $\mathbf{v3}^P \times \mathbf{t1}$



Cases $\mathbf{v3}^D \times \mathbf{t2}$ and $\mathbf{v3}^P \times \mathbf{t2}$



Cases $\mathbf{v4}^D \times \mathbf{t1}$ and $\mathbf{v4}^P \times \mathbf{t1}$



Cases $\mathbf{v4}^D \times \mathbf{t2}$ and $\mathbf{v4}^P \times \mathbf{t2}$

Figure 23: The bitemporal indeterminate cases

To explore the interaction of times in queries on bitemporal databases, we consider a number of queries on the simple database depicted in Figure 24 which shows that Jane's employment tuple was added to the database on June 2. Note that it contains an now-relative indeterminate "to" time and "*until changed*" as the "stop" time. For the purpose of the example, we assume that today is July 9. Thus, the transaction-time bind operator binds *until changed* to July 9 in all queries. The first four queries all include Jane in the result.

FACULTY					
		VALID TIME		TRANS TIME	
NAME	RANK	(from)	(to)	(start)	(stop)
Jane	*Assistant*	*June 1*	*(now − 3 days) ∼ January 1, 2028*	*June 2*	*until changed*

Figure 24: A bitemporal relation

- As best known today, who was possibly a faculty member on July 7?

$$\Pi^{P,V,VT,BT}_{July\ 7}(bind^{V,VT,BT}_{July\ 9}(\Pi^{V,TT,BT}_{July\ 9}(bind^{V,TT,BT}(Faculty))))$$

  In this query, we transaction timeslice to get only the most current information. The valid-time bind ensures a perspective of today, and the valid timeslice retrieves those tuples that were valid two days ago (on July 7).

- As best known today, who was definitely a faculty member on July 1?

$$\Pi^{D,V,VT,BT}_{July\ 1}(bind^{V,VT,BT}_{July\ 9}(\Pi^{V,TT,BT}_{July\ 9}(bind^{V,TT,BT}(Faculty))))$$

  As before, the lower bound of the "to" time is ground to July 6 (July 9 − 3 days). The difference is solely in the valid timeslice; we require definite information, and so we use a definite timeslice. Since July 1 is before July 6, Jane is in the result.

- As best known on July 1, who was definitely a faculty member on June 15?

$$\Pi^{D,V,VT,BT}_{June\ 15}(bind^{V,VT,BT}_{July\ 1}(\Pi^{V,TT,BT}_{July\ 1}(bind^{V,TT,BT}(Faculty))))$$

  The transaction timeslice retrieves the information current on July 1. The valid-time bind adopts this day as the perspective of the subsequent valid timeslice which retrieves information about June 15. Since Jane's tuple was current on July 1 and June 15 is more than three days before July 1, Jane will be in the result.

- As best known today, who will we say on July 12 is possibly a faculty member on September 1?

$$\Pi^{P,V,VT,BT}_{September\ 1}(bind^{V,VT,BT}_{July\ 12}(\Pi^{V,TT,BT}_{July\ 9}(bind^{V,TT,BT}(Faculty))))$$

  We first consider only current information. Then we adopt a valid-time perspective of July 12 to examine the database as it will appear on July 12 if no updates are made, i.e., our best guess as to what will be current information on July 12. Finally, using that perspective, we ask about possible information on September 1. Jane will thus be in the result.

In contrast to the queries above, the following three queries do *not* include Jane in the result.

- As best known today, who was definitely on the faculty of State University on July 7?

$$\Pi^{D,V,VT,BT}_{July\ 7}(bind^{V,VT,BT}_{July\ 9}(\Pi^{V,TT,BT}_{July\ 9}(bind^{V,TT,BT}(Faculty))))$$

  Here, the valid-time bind operation yields a ground "to" time of July 6 $\sim$ January 1, 2028. Since July 7 is after July 6, Jane is possibly, but not definitely, on the faculty.

- As best known on July 1, who was definitely a faculty member on July 1?

$$\Pi^{D,V,VT,BT}_{July\ 1}(bind^{V,VT,BT}_{July\ 1}(\Pi^{V,TT,BT}_{July\ 1}(bind^{V,TT,BT}(Faculty))))$$

- As best known today, who will we say on July 12 is definitely a faculty member on September 1?

$$\Pi^{D,V,VT,BT}_{September\ 1}(bind^{V,VT,BT}_{July\ 12}(\Pi^{V,TT,BT}_{July\ 9}(bind^{V,TT,BT}(Faculty))))$$

In most of the examples above, the transaction timeslice time and the valid-time bind time, or reference time, are the same. Indeed, this is the typical and most useful scenario as the following example makes clear. Suppose that today, July 9, we execute a transaction-time timeslice with time argument February 1. This operation chooses the most up-to-date information as of February 1, and disregards information that was not up-to-date on February 1 or was recorded at a later time. The user's perspective for subsequent operations using this information should naturally switch to the frame of reference of the chosen information. Hence, for this example, it would be natural to also bind *now* to February 1.

Yet, two of the queries given above illustrate that this in not a necessary restriction. Lifting it leads to increased functionality, but also to queries that are conceptually more involved. Existing query languages generally enforce this restriction.

We complete this discussion with another scenario that uses the various time variables, illustrating their interaction. Melanie downloads the AP News into her workstation each morning at 2 a.m. 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 first in Figure 25.

PRESIDENTS				
	VALID TIME		TRANS TIME	
NAME	(from)	(to)	(start)	(stop)
Bill Clinton	*now − 1 day ∼ 1/20/93*	*1/20/93 ∼ 1/19/2001*	*11/5/92*	*11/9/92*
Dan Quayle	*now − 1 day ∼ 1/20/93*	*1/20/93*	*11/10/92*	*until changed*
Bill Clinton	*1/20/93*	*now ∼ 1/19/2001*	*11/10/92*	*until changed*

Figure 25: A bitemporal presidential relation

The "start" time is the time the tuple was stored; the "stop" time was originally *until changed* at the time she inserted it because the tuple had yet to be logically deleted (we explain below how it became 11/10/92). The "from" time contains the (erroneous) assumption that if the President at that time, George Bush, were incapacitated, Bill Clinton, the newly elected but not yet inaugurated President, would be sworn in as President. In actuality, the Vice President, Dan Quayle, would

be sworn in and would serve until the inauguration on January 20, 1993. The one day lag is for the delay of the news-feed; 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.

Five days later, on November 10, 1992, she realized her mistake, and corrected it by changing the first tuple and introducing two more. It is at this time, that the "stop" time of the first tuple was set to 11/10/92, signifying its logical deletion. The second tuple was inserted at this time to indicate that it is really Vice President Quayle who would be President through inauguration day in the event that President Bush were incapacitated. The third tuple correctly asserts that Bill Clinton will assume the presidency on inauguration day and possibly remain President for eight years.

To complete the example, assume that today is January 1, 1993. Consider the following queries.

- As best known today, who could possibly be President today?

$$\Pi^{P,V,VT,BT}_{January\ 1,\ 1993}(bind^{V,VT,BT}_{January\ 1,\ 1993}(\Pi^{V,TT,BT}_{January\ 1,\ 1993}(bind^{V,TT,BT}(Presidents))))$$

  This query results in the following set of tuples.

$$\{< \text{Dan Quayle},\ [1/1/93, 1/1/93],\ [1/1/93, 1/1/93] >\}$$

- As was best known on November 9, 1992, who could possibly be President on January 1, 1993?

$$\Pi^{P,V,VT,BT}_{January\ 1,\ 1993}(bind^{V,VT,BT}_{November\ 9,\ 1992}(\Pi^{V,TT,BT}_{November\ 9,\ 1992}(bind^{V,TT,BT}(Presidents))))$$

  This query results in the following set of tuples.

$$\{< \text{Bill Clinton},\ [1/1/93, 1/1/93],\ [11/9/92, 11/9/92] >\}$$

- As was best known on November 9, 1992, who could possibly be President on November 6, 1992?

$$\Pi^{P,V,VT,BT}_{November\ 6,\ 1992}(bind^{V,VT,BT}_{November\ 9,\ 1992}(\Pi^{V,TT,BT}_{November\ 9,\ 1992}(bind^{V,TT,BT}(Presidents))))$$

  This query results in the empty set. Only the first tuple is in the transaction-time timeslice result, and the "from" time of this tuple evaluates to November 8, 1992, two days after the valid-time timeslice argument.

7 Timestamp Implementation

This paper has proposed four new current-time-related timestamps; namely, *until changed*, *now*, now-relative instants, and now-relative indeterminate instants. In this section we demonstrate how these timestamps may be efficiently represented and manipulated.

Elsewhere we have proposed timestamp formats for determinate instants and intervals [DS93a, DS93d] and indeterminate values [Dyr94]. Following a quick review, the existing formats are extended to also represent the new timestamp values, and it is shown that the now-relative formats have only a minor impact on storage costs. We extend only the instant formats; intervals are represented as a pair of bounding instants.

7.1 Representation of Determinate and Indeterminate Timestamps

An instant timestamp must meet several requirements. First, the timestamp must support a multitude of *ranges*. Range is a measure of how much of a time-line can be represented. A timestamp should be capable of storing times that range over just a few seconds to those that range over the age of the universe. Second, it must support a variety of *time granularities* [DS94], from those as large as a millennium to those smaller than a femtosecond. Third, the timestamp must be capable of storing times both before and after a granularity *anchor* (an anchor is a point on the time-line). Finally, since it is difficult to anticipate the demands of future language designers, the format must allow for growth, primarily the addition of new timestamp types.

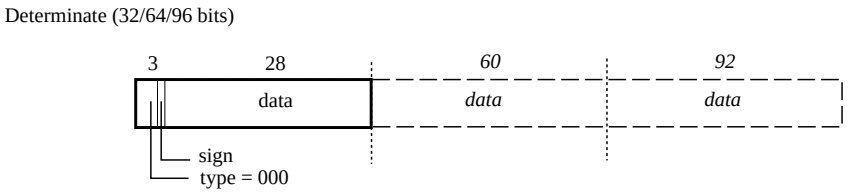


Figure 26: The determinate instant format

The determinate instant timestamp is shown in Figure 26. The dashed lines mark word boundaries. The number above a subfield is the size of that subfield, in bits. The size of the timestamp, in particular, the size of the *data field*, varies depending on the *range* and *granularity* of the timestamp. We assume that the range and granularity are specified for the instant as part of a create or modify statement and stored with the schema. Hence, the interpretation of the data field requires information from the schema, provided by the query processor. Larger ranges and finer granularities require more bits to represent, consequently the size of the data field, and, in turn, that of the timestamp, varies. The one, two, and three word formats are depicted in the figure. Two words should be sufficient for most applications. The two word timestamps can store a range of historical times to the granularity of a microsecond, or times within a range of 36 billion years to the granularity of a second. Three words, which can represent times over a range of 36 billion years to a nanosecond, should take care of the few remaining applications.

To differentiate amongst the timestamp formats, each format has a type field. The type field is stored in the high order portion rather than the low order portion because not every format is the same size. The type field distinguishes special instants, such as *beginning* and *forever*, from other instants. The special instant format is shown in Figure 27. Note that the timestamp is only as long as the range and granularity dictates.

There are several formats to represent indeterminate instants. Without loss of generality, we

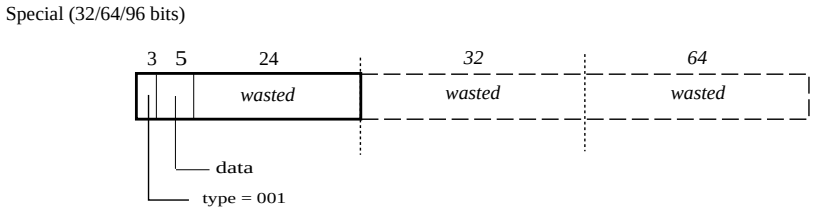


Figure 27: The special instant format

will discuss only the *chunked* format shown in Figure 28.

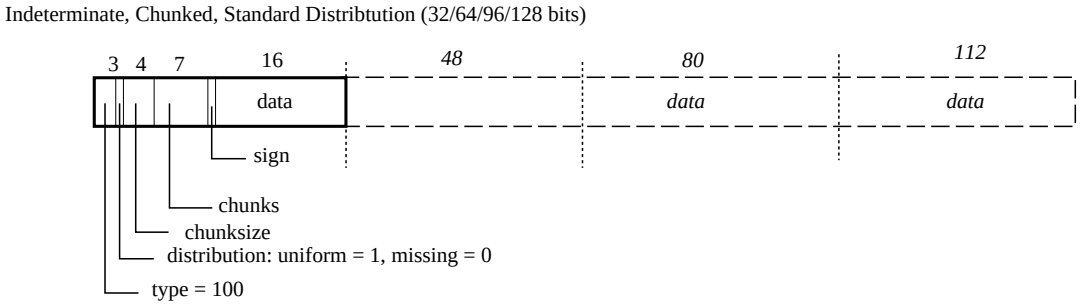


Figure 28: The chunked format

The chunked format stores a lower bound, an upper bound, and a probability distribution (in this paper we ignore the distribution). The lower bound is explicitly encoded in the chunked format, however, the upper bound is implicitly encoded. The upper bound is composed of a *chunk size* and a number of *chunks*. A chunk is a span, the length of which is specified by the chunk size field. The upper support is computed by adding the number of chunks, each of size *chunk size*, to the lower support. For example, to represent a period of indeterminacy of seven hours using chunks, the timestamp would record that there are seven hour-sized chunks. The chunk sizes that can be used depend on the granularity of the timestamp. Chunk sizes smaller than the granularity cannot be used since the timestamp cannot store these times. One of the duties of the database implementor is to build site-specific chunk size tables, one for each granularity.

7.2 Representation of Now-related Timestamps

In some sense, *until changed* and *now* are "special" instants, hence we need only add to this format, shown in Figure 27, leaving the other formats unchanged. The variables *until changed* and *now* are stored as special instants. Two new special instant values are allocated for the two variables raising the total number of special instants to six.

Now-relative instants are a new format based on the determinate instant format. The now-relative format is show in Figure 29.

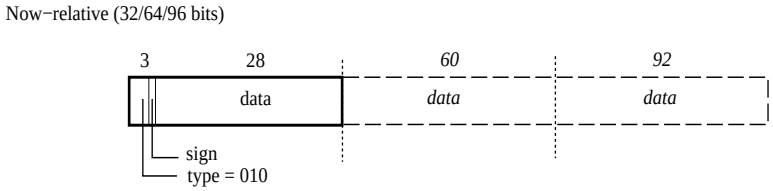


Figure 29: The now-relative instant format

Now-relative instants are of the form "*now* ± *span*." The span associated with the instant is stored in the data field (just as spans are currently stored). The span value is stored to the range and granularity of the other instant timestamps in that column. The sign bit indicates whether the span is added or subtracted to *now*. Note that the now-relative format has a unique type field.

A now-relative indeterminate instant is quite a bit more complex. It is of the form "*now* ± *span* ∼ *upper bound*." A now-relative indeterminate instant format must store an upper bound, a

span, *now*, and whether or the span is added or subtracted. Surprisingly, it is usually possible to fit all of this information in the same space that a determinate instant requires, eight bytes in the common case. We employ a variation of a chunked indeterminate format that has essentially the same format, but a different interpretation. The new format is shown in Figure 30.

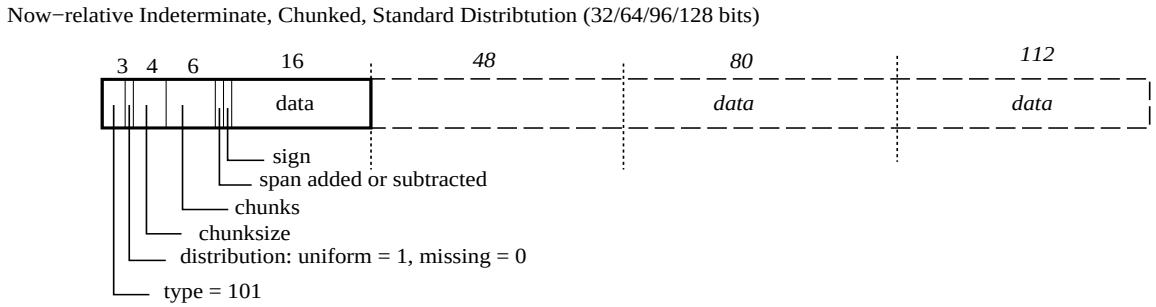


Figure 30: The now-relative indeterminate instant format

A now-relative indeterminate instant has a different type field than an indeterminate instant since there is no other way to distinguish between the two formats. The upper bound is explicitly encoded in the data field, while the span is implicitly encoded in the chunking information. The now-relative indeterminate format has two sign bits. The first sign bit indicates whether the span is added or subtracted to *now*. The second sign bit positions the upper bound before or after the granularity anchor point, just as does the sign bit for determinate instants.

If the chunking yields an inadequate precision, or the user wishes to associate a distribution other than the missing or uniform distribution then more storage will be required. In this case, we use variants of other indeterminate instant formats. These variants are essentially the same as the indeterminate formats, but have a different type value and use a different interpretation of the fields in the format. For brevity, we omit these variants. We also omit discussion of the *nonstandard* distribution formats, which are slightly larger than the *standard* formats presented here [Dyr94].

As a point of comparison, the SQL-92 `datetime` format, which has a range of only 10,000 years and a granularity of only one second, and which does not incorporate either indeterminacy or now-relativity, requires twenty positions (80 bits). For most users, our eight byte (64 bit) format should suffice.

7.3 Timestamp Operations

In this paper, we proposed adding bind operations for valid time, transaction time, and bitemporal databases; no other operations are needed to support current-time-related modeling entities. The bind operations have no significant impact on the run-time efficiency of a temporal database. The transaction-time bind is very efficient. It simply replaces *until changed* with the current transaction time. The valid and bitemporal bind operations are only slightly less efficient. For now-relative instants (and now-relative indeterminate instants) these operations replace *now* with the reference time and then displace that reference time by a span. The displacement costs one integer addition operation.

Now-relative instants also add an extra comparison to interval constructors. As we observed in Section 2.1.4, predictive updates could insert into the database intervals that end before they start. For a tuple without variables, such intervals can be detected and eliminated when the tuple is first inserted into the database. But a tuple with a variable might initially end before it starts,

and only later evolve into a valid interval. Consequently, during run-time each interval involving a variable must be tested to ensure that the starting instant is before the terminating instant. This test needs be performed only once per interval per query.

8 Summary and Research Directions

The overall conclusion of this paper is a recommendation that timestamps involving current-time variables, that is, *now*, *until changed*, now-relative, and now-relative indeterminate timestamps, be used for timestamping database facts with their valid and transaction times. A summary of the most important contributions of this paper as well as refinements to the overall conclusion follow.

First, this paper provides a formal basis for defining the semantics of databases with variables. The use and generality of the framework was demonstrated by giving a semantics for valid-time, transaction-time and bitemporal databases with all existing variables. Apart from specifying reasonable semantics for such databases, this exercise demonstrates two important properties of the framework. The first property is that it is capable of capturing the semantics of a wide range of variables. The second is that the semantics of a multi-dimensional database may be specified as a coordinated combination of the semantics of the constituent one-dimensional databases. The reference-time dimension in the framework provides the coordination mechanism. For example, the semantics of variable bitemporal databases was specified very easily by using the already specified semantics for valid-time and transaction-time databases. This property makes it relatively easy to specify the semantics of multi-dimensional databases.

Second, without current-time variables, temporal databases provide inadequate support for their applications. The paper demonstrates that existing variables, such at *now* and *until changed*, are indispensable in temporal databases. It also identifies situations were even these variables are inadequate, and introduces new *now-relative* and *now-relative indeterminate* instants that provide the desired support. The semantics of databases with variables are also defined within the framework.

Third, a foundation for the querying of variable databases from existing temporal query languages was presented. The paper provides algebraic "bind" operators for valid-time, transaction-time, and bitemporal databases, and it shows how these are used in order to permit existing query languages to access variable databases. As a first step during query processing, the bind operation is applied to variable databases, thus temporarily replacing all variables with ground values appropriate for the processing of the query at hand. What are appropriate ground values follows quite easily when the semantics of the variable databases have been defined within the framework. This approach encapsulates the handling of variables in a single operator. It also requires only minimal changes to the query processor: support for one new operator has to be added, but all other components remain unchanged.

Fourth, compact formats are provided for the physical representation of the variables. The paper extends an existing timestamp representation scheme to also include compact formats for the variables, and it is shown that the presence of variable timestamps do not have significant, adverse effects on query processing performance.

The simplest variable is a distinguished valid-time value *now* (and the analogous transaction-time value *until changed*). Supporting these variables requires essentially no space, but engenders several modeling problems, as discussed in Section 2. Now-relative instants require minimal space, specifically one bit to distinguish them from non-relative instants, and add some modeling power. Incorporating indeterminacy increases the modeling capability further, yet adds from 13 bits to 3 words to the representation of a timestamp, depending on how finely the user wishes to model the indeterminacy. Indeterminate now-relative instants require no additional space over non-relative

indeterminate instants, and solve all the problems listed in Section 2.

These four observations provide the rationale for the conclusion that variable databases are viable. A number of secondary, but noteworthy, contributions also deserve mention. The paper resolves the meaning of the use of variables in existing temporal data models. A graphical notation with two or three dimensions used throughout the paper proved to be helpful when describing the semantics of variable databases. The complex interactions of current time, reference time, transaction time, and valid time within queries and variable databases were investigated in detail. These interactions were not thoroughly understood or explicated in existing bitemporal data models. The concept of "perspective" within queries was illustrated. Perspective adds the ability to bind the valid-time variable *now* to times other than the current time. Supporting this notion within a query languages enhances its functionality when querying variable databases.

This framework has implications for database query language design. The user-defined time types available in SQL-92 could be extended to support now-relative and indeterminate non-relative variables. The TSQL2 language [SAA+94] does so, and also supports those variables for valid and transaction time. In TSQL2 the "bind" operation is implicit; `NOBIND` is provided to store variables in the database.

There are several directions for future research. The precise semantics of several temporal models proposed in the literature could profitably be examined in light of the framework presented here. In defining the semantics for bitemporal databases, we have chosen but one possible way of combining the semantics of valid-time and transaction-time databases; other possible combinations of these two temporal dimensions might also prove useful. In addition, the use of the graphical representation of temporal relations at the user interface—for displaying the results of queries and, e.g., for the assertion of temporal integrity constraints—seems to us a promising one for further research. Finally, new kinds of variables, such as *here* for spatial and spatiotemporal databases, should be investigated, as an extension of the framework introduced here.

9 Acknowledgments

Partial support for Curtis Dyreson and Richard Snodgrass was provided by the National Science Foundation through grants IRI–8902707 and IRI–9302244, the IBM Corporation through Contract #1124, and the AT&T Foundation. Partial support for Christian S. Jensen was provided by the Danish Natural Science Research Council through grants 11–9675–1 SE and 11–0061. Nick Kline provided helpful comments on a previous draft.

References

[ASS94] K. K. Al-Taha, R. T. Snodgrass, and M. D. Soo. Bibliography on Spatiotemporal Databases. *International Journal of Geographical Information Systems*, 8(1):95–103, January-February 1994.

[ABM84] G. Ariav, A. Beller, and H. L. Morgan. A Temporal Data Model. Technical Report DS-WP 82-12-05, Decision Sciences Department, University of Pennsylvania, December 1984.

[BG89] G. Bhargava and S. Gadia. Achieving Zero Information Loss in a Classical Database Environment. In *International Conference on Very Large Databases*, pages 217–224, Amsterdam, August 1989.

[BL92] M. A. Bassiouni and M. J. Llewellyn. A Relational-calculus Query Language for Historical Databases. *Computer Languages*, 17(3):185–197, 1992.

[BZ82] J. Ben-Zvi. *The Time Relational Model.* PhD thesis, University of California at Los Angeles, 1982.

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

[CCT93] J. Clifford, A. Croker, and A. Tuzhilin. On Completeness of Historical Relational Query Languages. *ACM Transactions on Database Systems*, 19(2):64–116, March 1994.

[Cli93] J. Clifford. Indexical Databases. In *Advanced Database Systems*, Lecture Notes in Computer Science 759, Springer-Verlag, 1993.

[CI93] J. Clifford and T. Isakowitz, On The Semantics of Transaction Time and Valid Time in Bitemporal Databases. In R. T. Snodgrass, editor, *Proceedings of the ARPA/NSF International Workshop on an Infrastructure for Temporal Databases*, pp. I.1–I.17, Arlington, TX, June 1993.

[CI94] J. Clifford and T. Isakowitz, On The Semantics of (Bi) Temporal Variable Databases. In *Proceedings of the Fourth International Conference on Extending Database Technology*, pp. 215–230, Cambridge, England, March 1994.

[CT85] J. Clifford and A. U. Tansel. On an algebra for historical relational databases: Two views. In S. Navathe, editor, *Proceedings of ACM SIGMOD International Conference on Management of Data*, pp. 247–265, Austin, TX, May 1985.

[CW83] J. Clifford and D. S. Warren. Formal Semantics for Time in Databases. *ACM Transaction On Database Systems*, 8(2):214–254, 1983.

[DS93a] C. E. Dyreson and R. T. Snodgrass. Timestamp Semantics and Representation. *Information Systems*, 18(3):143–166, 1993.

[DS93b] C. E. Dyreson and R. T. Snodgrass. Valid-time Indeterminacy. In *Proceedings of the International Conference on Data Engineering*, pp. 335–343, Vienna, Austria, April 1993.

[DS93d] C. E. Dyreson and R. T. Snodgrass. A Timestamp Representation for TSQL2. *A TSQL2 Commentary.* September 1994, 21 pages.

[DS94] C. E. Dyreson and R. T. Snodgrass. Temporal Granularity and Indeterminacy: Two Sides of the Same Coin. Technical Report TR 94-06, University of Arizona, Departmentn of Computer Science, Tucson, AZ, February 1994.

[Dyr94] C. E. Dyreson. Valid-time Indeterminacy. Ph.D. thesis, The University of Arizona. October 1994, 187 pages.

[DW90] C. J. Date and C. J. White. *A Guide to DB2*, volume 1, 3rd edition. Addison-Wesley, Reading, MA, September 1990.

[EWK90] R. Elmasri, G. Wuu, and Y. Kim. The Time Index — an Access Structure for Temporal Data. In *International Conference on Very Large Databases*, Brisbane, Australia, August 1990.

[Fin92] M. Finger. Handling Database Updates in Two-dimensional Temporal Logic. *Journal of Applied Non-Classical Logics*, 2(2), 1992.

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

[GN93] S. Gadia and S. Nair. Temporal Databases: A Prelude to Parametric Data. In A. Tansel, J. Clifford, S. Gadia, S. Jajodia, A. Segev, and R. T. Snodgrass, editors, *Temporal Databases: Theory, Design, and Implementation*, chapter 2, pp. 28–66. Benjamin/Cummings, 1993.

[JCE+94] C. S. Jensen, J. Clifford, R. Elmasri, S. K. Gadia, P. Hayes, and S. Jajodia (editors). A Consensus Glossary of Temporal Database Concepts. *ACM SIGMOD Record*, 23(1):52–65, March 1994.

[JM90] C. S. Jensen 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, College Park, MD, August 1990.

[JM92] C. S. Jensen and L. Mark. Queries on Change in an Extended Relational Model. *IEEE Transactions on Knowledge and Data Engineering*, 4(2):192–200, April 1992.

[JS92] C. S. Jensen and R. T. Snodgrass. Temporal Specialization. In F. Golshani, editor, *Proceedings of the IEEE International Conference on Data Engineering*, pp. 594–603, Tempe, AZ, February 1992.

[JS94] C. S. Jensen and R. T. Snodgrass. Temporal Specialization and Generalization. *IEEE Transactions on Knowledge and Data Engineering*, 6(6), December 1994, to appear.

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

[JSS94] C. S. Jensen, M. D. Soo, and R. T. Snodgrass. Unifying Temporal Data Models via a Conceptual Model. *Information Systems*, to appear, December 1994.

[Kli93] N. Kline. An Update of the Temporal Database Bibliography. *ACM SIGMOD Record*, 22(4):66–80, December 1993.

[Lip79] W. Lipski, Jr. On Semantic Issues Connected with Incomplete Information Databases. *ACM Transactions on Database Systems*, 4(3):262–296, September 1979.

[LJ88] N.A. Lorentzos and R.G. Johnson. Extending Relational Algebra to Manipulate Temporal Data. *Information Systems*, 13(3):286-296, 1988.

[Mon74] R. Montague. *Formal Philosophy: Selected Papers of Richard Montague.* Yale University Press, New Haven, 1974.

[MS93] J. Melton and A. R. Simon. *Understanding the New SQL: A Complete Guide.* Morgan Kaufmann Publishers, Inc., San Mateo, CA, 1993.

[NA89] S. B. Navathe and R. Ahmed. A Temporal Relational Model and a Query Language. *Information Sciences*, 49:147–175, 1989.

[Rei84] R. Reiter. Towards a Logical Reconstruction of Relational Database Theory. In *On Conceptual Modelling*, pp. 191–233. Springer Verlag, 1984.

[Rod92] J. F. Roddick. Schema Evolution in Database Systems — An Annotated Bibliography. *SIGMOD Record*, 21(4):35–40, December 1992.

[SA85] R. T. Snodgrass and I. Ahn. A Taxonomy of Time in Databases. In S. Navathe, editor, *Proceedings of ACM SIGMOD International Conference on Management of Data*, pp. 236–246, Austin, TX, May 1985.

[SAA+94] R. T. Snodgrass, I. Ahn, G. Ariav, D. S. Batory, J. Clifford, C. E. Dyreson, R. Elmasri, F. Grandi, C. S. Jensen, W. Kafer, N. Kline, K. Kulkanri, T. Y. C. Leung, N. Lorentzos, J. F. Roddick, A. Segev, M. D. Soo, and S. M. Sripada. TSQL2 Language Specification. *ACM SIGMOD Record*, 23(1):65–86, March 1994.

[Sar90] N. L. Sarda. Algebra and Query Language for a Historical Data Model. *The Computer Journal*, 33(1):11–18, February 1990.

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

[Sno93] R. T. Snodgrass. An Overview of TQuel. In A. Tansel, J. Clifford, S. Gadia, S. Jajodia, A. Segev, and R. T. Snodgrass, editors, *Temporal Databases: Theory, Design, and Implementation*, chapter 6, pp. 141–182. Benjamin/Cummings, 1993.

[S92c] M. D. Soo and R. T. Snodgrass. Mixed Calendar Query Language Support for Temporal Constants. TempIS Technical Report No. 29, Revised May 1992.

[Syk64] J. B. Sykes, editor. *The Concise Oxford Dictionary.* Oxford University Press, Oxford, England, 1964.

[Tan90] A. U. Tansel. Modelling Temporal Data. *Information and Software Technology*, 32(8):514–520, October 1990.

[TK88] S. Thirumalai and S. Krishna. Data Organization for Temporal Databases. Technical report, Raman Research Institute, India, Bangalore, India, 1988.

[WJL91] G. Wiederhold, S. Jajodia, and W. Litwin. Dealing With Granularity of Time in Temporal Databases. In *Proceedings of the 3rd Nordic Conf. on Advanced Information Systems Engineering*, Trondheim, Norway, May 1991.

[WJL93] G. Wiederhold, S. Jajodia, and W. Litwin. Integrating Temporal Data in a Heterogeneous Environment. In A. Tansel, J. Clifford, S. Gadia, S. Jajodia, A. Segev, and R. T. Snodgrass, editors, *Temporal Databases: Theory, Design, and Implementation*, chapter 22, pp. 563–579. Benjamin/Cummings, 1993.

[YC91] C. Yau and G. S. W. Chat. TempSQL — a Language Interface to a Temporal Relational Model. *Information Sc. & Tech.*, pp. 44–60, October 1991.