DEALING WITH GRANULARITY OF TIME IN TEMPORAL DATABASES

Gio Wiederhold†
Department of Computer Science
Stanford University, Stanford, CA 94305-2140, U.S.A.

Sushil Jajodia
Department of Information Systems and Systems Engineering
George Mason University, Fairfax, VA 22030-4444, U.S.A.

Witold Litwin‡
I. N. R. I. A.
78153 Le Chesnay, France

ABSTRACT A question that always arises when dealing with temporal information is the *granularity* of the values in the domain type. Many different approaches have been proposed; however, the community has not yet come to a basic agreement. Most published temporal representations simplify the issue which leads to difficulties in practical applications. In this paper, we resolve the issue of temporal representation by requiring two domain types (*event times* and *intervals*), formalize useful temporal semantics, and extend the relational operations in such a way that temporal extensions fit into a relational representation. Under these considerations, a database system that deals with temporal data can not only present consistent temporal semantics to users but perform consistent computational sequences on temporal data from diverse sources.

1. Introduction

Large databases do not just collect data about the current state of objects but retain information about past states as well. In the past, when storage was costly, such data was often not retained online. Aggregations of past information might have been kept, and detail data was archived.

Certain applications always required that an adequate history be kept. In medical database systems a record of past events is necessary to avoid repeating ineffective or inappropriate treatments, and in other applications legal or planning requirements make keeping of a history desirable. In banking long audit trails are kept to assure traceability of errors and fraud, and apply corrections when required. In planning applications we must project into the future, and trends from the past are an important source of information.

Changes in technology are supporting a movement towards temporal databases. Today the cost of data-entry often exceeds the cost of long-term storage. If we want to take object histories into account, we must identify the stored data with *timestamps* which associate temporal domain values with objects being described. Timestamps require an appropriate domain definition, and any temporal database has to support temporal domains with operations that provide, at the very least, correct comparisons. Temporal values must be associated with events or objects so that for each object a time-precedence and a temporal sequence can be established. We also

† Partially supported by DARPA, contract # N39-84-C-211.
‡ Performed while visiting Stanford University.

need operations that help in temporal data reduction and in combining temporal information from multiple relations.

A question that arises when dealing with temporal information is the *granularity* of the values in the domain type. In some applications the granularity is days, in others it can be seconds or years. Several authors (e.g., see clifford tansel 1985 ramez elmasr gene wuu tempor model query langua databa proc conf februa 1990 gadia vaishnav march 1985 tansel arkun ozsoyo 1989 model time as a discrete linearly ordered set. Discrete time implies that there exists some granularity (years, days, seconds, etc.) which is universally valid, otherwise we are creating semantic mismatches when operations combine temporal information. Figure 1 illustrates such a mismatch. Most representations have ignored mixed granularities. Later in this paper we discuss this problem and our resolution in greater depth.

We have information on playing times of movies given in days,
and on smog alerts, given in hours.

Smokey and the Bandit plays from 12Dec1977 to 15Dec1977.
A smog alert is in force from 5 pm 12Dec.1977 to 6 pm 15Dec1977.

Common sense tells us that 5 pm 12Dec1977 comes after 12Dec1977
but
6 pm 15Dec1977 comes before 15Dec1977.

Figure 1: Mixing granularities.

The thesis by Ladkin ladkin recognizes that distinct granularities cannot be mixed, and develops an algebra where the granularity of the source timestamps is considered throughout. Binary operations that mix granularities have results that are the best approximation, which is typically the coarsest granularity of the two arguments. Unfortunately, such an approach requires a large set of operations, and is difficult to apply to the set-flavored operations required for databases.

For databases many types of time-stamps and interval representations have been proposed. Some researchers gadia vaishnav march 1985 tansel arkun ozsoyo 1989 use only intervals to denote temporal domains. This approach simplifies the model, since now there is only one type. Some shashi gadia homoge relati model query langua tempor databa acm tods decemb 1988 jajodia use unions and cross product of intervals. In richar snodgr ilsoo ahn tempor databa ieee comput septem 1986 richar snodgr tempor query langua tquel acm trans databa system june 1987 a variety of underlying time domains is presented: entry time, valid time, etc.

We identify problems with most of these temporal representations (See Section 3). Although they provide interesting and internally consistent theories, their representations lead to problems in realistic applications. Our experience derives from the development, implementation, and use of a time-oriented database system (TOD) developed in the early seventies and its successors. robert blum discov repres causal relati large orient clinic databa projec spring verlag lectur notes medica inform 1982 james fries dennis mcshan aramis nation chroni diseas databa system procee sympos comput applic medica care octobe 1979 gio wieder fries weyl struct organi clinic databa proc ncc 1975 However, this system did not incorporate an algebra; all temporal operations were embedded in PL/I code modules, invoked by applications and code generators.

In this paper, we will first deal with the representation of time. Timestamps require an appropriate domain definition so that a database system which deals with temporal events can present consistent temporal semantics to the user to avoid confusion. Moreover, if we formalize the relevant semantics, the database management system (DBMS) can determine, schedule, optimize, and execute transactions involving temporal information. Achieving the required consistency means that we avoid mismatch of domains.

We contend that factual data is collected at the source based on the times associated with events. However, queries and snapshots (see, for example, richar snodgr tempor query langua tquel acm trans databa system june 1987 need valid intervals to retrieve information, since their parameters will rarely match event times exactly. Since event times are associated with arbitrarily fine granularity, we need to convert this event data to a well-formed interval representation, which we call *histories*. Unlike the event data, historical data should be time *granularity independent.* Once a history is derived from the source events, a database can carry out the computations to satisfy application semantics.

To benefit from existing research, we base our work on the relational model. We extend the set of relational operations in such away that all extensions fit into a relational representation. Kamens wiederhold An implementation of temporal queries for SQL Only the semantics are extended and new operations take advantage of these semantics. Since we are concerned with the precise operational semantics of temporal operations, we define them as algebraic operators. We follow again the relational development where the algebraic definition preceded the calculus. codd boyce

The organization of this paper is as follows. In Section 2, we start with some definitions so we can distinguish levels of temporal support. In Section 3, we indicate how several temporal representations give difficulty if the data have to merged with other data or if snapshots at different points in time instances have to be derived. We give the steps needed to process the temporal data in Section 4. The critical step in the processing scenario is the conversion of event information to a history. It requires an understanding of the semantics of the time domain, and our history operator permits specification of these semantics. In Section 5, we give a number of basic operations on tuples with timestamps, and in Section 6 we present the operation needed for temporal relations. The conclusion is given in Section 7.

2. Levels of Temporal Support

In this section, we introduce definitions for several types of temporal databases. These types impose constraints on the representation of temporal data. Since these constraints prevent anomalies, they can be viewed as a temporal normalization.

We define a *journal* as a database which collects information about the temporal changes of objects. To support a journal, a database must associate with each object some time-variant attribute values as well as temporal domain values which represent periods of validity for these time-variant attribute values. In a relational model a nest of tuples is needed to represent the changes pertaining to an object. For instance, the salaries of employees together with their departments in Figure 2 constitutes such a journal. The object identifier (attribute ''Name'') and the timestamp (attribute ''Date'') form the relation key. The nest for each object forms a temporal sequence of events. Each employee has a time-variant salary which is valid for some time interval past the time a salary is recorded. The events in a sequence determine intervals. The period that an employee has a particular salary is defined by the time that salary is recorded in the database and the time a change is made to it. With the assumption that a salary is *stable* in that period, the value is valid throughout this period. The temporal relation in Figure 3 is another journal; however, period of validity of a temperature value is very different >From that of a salary value in Figure 2. A temperature value is an instantaneous measurement and is valid only for some small time interval around the hour value.

Name	Salary	Dept	Date
Peter	30K	Shoe	1Jan80
Peter	32K	Shoe	4Jun80
Margi	30K	Shoe	6Mar78
Margi	31K	Shoe	1Jan79
Margi	32K	Shoe	4Jun80
Jack	30K	Linen	4Jul79
Jack	30K	Shoe	4Dec80

Figure 2. A journal with discrete values.

Location	Degrees	Hour
Class room	17.0C	8:00
Class room	22.0C	9:00
Class room	23.0C	10:00
Kitchen	23.0C	9:00
Kitchen	25.0C	10:00

Figure 3. Journal with continuous values.

A *historial database* extends the requirement of a journal by requiring that the sequences be complete between the extreme start-point and finish-point of each object history. Whereas a journal only records events, a history also tells us about the state of the object at intermediate points. The journal in Figure 2 is easily converted to a complete salary history since each employee's salary is known within each interval. The journal in Figure 3, on the other hand, is not directly convertable to a history since temperatures at intermediate points are not known. (See, however, Section 4.2).

We restrict a *proper historical database* to be a historical database such that each object is assigned a unique value at any point within the time interval. This means that for each point in time covered there is one and only one value for salary of any of the employees. We denote periods without a salary by a null (Λ) entry. The journal in Figure 2 represents a proper history. One way to ensure that a history is proper by making the time attribute a part of the object identifier. With that constraint we can be assured of a high quality result. A snapshot of a proper history will, for instance, be in first-normal form.

We now define a *historical database system* to be a temporal database system which supports computations on histories. A historical database permits computation of *snapshots* for any given time instant. The snapshot is a result relation which shows all values as they existed at that time. A frequently needed snapshot is the current state; this result is then similar to the updated relation in a non-temporal database.

3. Choices of Domain Types for Timestamps

Any temporal database system has to support a domain type for timestamps. The values in those domains must be comparable so that a time-precedence and a temporal sequence can be established. A question that always arises when dealing with temporal information is the *granularity* of the values in the domain type. Many different types have been proposed; however, there is currently no consistency among the members of the research community. Several authors model time as a discrete linearly ordered set $T = \{0,1, ..., now\}$ where *now* denotes the changing value of the present time. Some (see, for example, clifford tansel 1985 gadia vaishnav march 1985 tansel arkun ozsoyo 1989 view temporal relations as *event relations* where each attribute or tuple is tagged with an appropriate timestamp. (See Figures 2 and 3.) Unfortunately, few of our common algebraic operations are applicable to discrete timestamps. Timestamps cannot be added; multiplied, or divided; they can only be compared. Subtraction of timestamps gives interval sizes. Thus, event relations have the problem that they cannot directly provide answers to queries dealing with event points that are not explicitly represented. For the event relation in Figure 2, "What was Peter's salary on 1Mar80?" is an example of a hard query. Not only is a type of range query required to locate the prior and successor tuples (see Section 6.1), but also the semantics of having a stable salary are implicitly invoked.

The problem gets even worse when there is a need to combine two or more relations with different temporal domain types into a single relation. As an example, consider the relation given in Figure 4.

Name	Salary	Dept	Start	End
Peter	32K	Toys	Jan77	Jan78
John	15K	Shoe	Jan77	Jan81
John	25K	Shoe	Jan81	Jan83

Figure 4. A temporal relation with different domain type

It is not obvious how the system should combine the two relations in Figures 2 and 4 to form a single relation. Standardization of domain types in a database can prevent this mismatch; however, in practice, since we derive information from diverse sources, not always under the same control, we will always encounter some mismatched domains.†

Let us once again consider the event relation given in Figure 2. If a user wishes to derive a snapshot at time 2Jan80 for the data of Figure 2, it seems reasonable for the system to return the state given in Figure 5. Stability of the discrete event results is assumed here.

Name	Salary	Dept
Peter	30K	Shoe
Margi	31K	Shoe
Jack	30K	Linen

Figure 5. A snapshot

However, the stability assumption is not valid for all attributes. For instance, if the database also records events where bonuses were given, then a snapshot query should not assume that the bonus is valid throughout every implicit granularity unit used for other employment data.

Assumptions other than stability or none also occur. Suppose a user asks for a snapshot at time 9:15 for the relation given in Figure 3 that records room temperatures. If the system uses

† There are other types of mismatches (e.g., semantic mismatch DeMichiel that can occur in these cases, but they are outside the scope of this paper.

the stability assumption, it will derive the answer given in Figure 6 which is not likely to be the desired response. Unless we formalize all relevant temporal semantics, we cannot leave the task of executing queries involving temporal data to the system; the responsibility of defining the semantics for the temporal data falls onto the users. It is specified during the query definition phase.

Location	Degrees
Class room	22.0C
Kitchen	23.0C

Figure 6. An incorrect snapshot using stability assumption

Some authors use intervals as the temporal domain type, but there too some use representations which lead to difficulties. For example, Dutta dutta uses ordered pairs of the type $<closed-interval,value>$ to represent a temporal attribute. So the bank balance of a certain account is represented as

$$\{([2Jan87,3Jan87],3000), ([3Jan87,5Jan87],5000\}.$$

We now have the problem that we are not sure of the exact bank balance on 3Jan87. It is either 3000 or 5000, but we do not know which. One could argue that there is one other possible interpretation that the account balance changed from 3000 to 5000 during the day, but we do not know exactly when it changed inside the granule. The difficulty, however, is that if we wish to have a proper balance history, we will require time granules smaller than a day which requires information which is not available.

Some (e.g., shashi gadia jay vaishn query langua homoge tempor databa proc acm pods march 1985 shashi gadia weak tempor relati proc acm pods march 1986 martin model temporal intervals which do not adjoin; they are always separated by event boundaries of some grain size. This representation has some of the same problems as those seen with event relations. In, shashi gadia homoge relati model query langua tempor databa acm tods decemb 1988 jajodia unions and cross product of intervals are used that require extensions to the relational model.

In clifford tansel 1985 gadia vaishnav march 1985 as well as in many statistical programs, a further assumption is made that all intervals are equal in size. We find this assumption too costly in database practice where histories are long and events occur at unpredictable times. For these reasons it is better to denote interval start and finish explicitly. Statistical programs can then evaluate if their assumptions are sufficiently valid to allow reliable computations.

3.1. Granularity Differences

In any specific application, the granularity of time has some practical magnitude. For instance, the time-point that a business event, like a purchase, is associated with a *date,* so that a *day* is the proper granule for most business transactions. People do not schedule themselves for intervals of less than a minute, while database transactions may be measured in milliseconds. Eventually we are limited by the precision that our hardware can recognize; fractions of microseconds are the finest grain here. We use G to denote the granularity; it is in effect an interval.

The finiteness of measurement granules leads to a confusion of event times and intervals. If we limit our event measures to dates (G = 1 day), and we say that an event occurred on such-and-such a day, then implicit for most of us is also that the event spanned some interval within that day. A point event is then associated with an interval of one granule length. There will be a smallest time granule G, perhaps intervals of seconds or days, which follow each other without gaps, and are identified by the timepoint at their beginning. True Intervals are sequences of event measuring intervals.

However, problems arise with this simplification. Inconsistencies occur when an *inclusive interval* is defined by two event time measurements with an implicit grain. First we have to round actual measurements to the integer grainsize used; then we add the granule size to the result:

$$T_G = t_f - t_s + G$$

where t_s denotes the value corresponding to the start of the interval and t_f the value when the interval is finished. Thus, if movie is shown daily from the 12th to the 19th of a month, there will be 19-12+1 = 8 performance days. While we are all used to performing such adjustments when computing with intervals, a database system which deals with temporal events must present consistent temporal semantics to the user to avoid confusion. We cannot use an event directly to compute an interval but always have to correct for the associated grain size. While in any one application use of a fixed grainsize is feasible, problems arise if we merge information from distinct applications.

A database system has to carry out the computations to satisfy the application semantics. If those include the use of finite events, then the grain size assumption made must be explicitly stated. Many granularities may need to be simultaneously active. In our formulation we will require two datatypes, infinitesimal time points for events and intervals for histories, to deal with all temporal data.

4. Processing of Temporal Data

In this section we outline the steps needed to process the temporal data, given the design decision made above.

1. *Collection of new data.* When temporal information about objects is added to relations describing objects, it does not change the essence of the objects, it merely records their history. The traditional means for identifying the objects remain accessible and manipulable. However, the time-identification is appended to the object identifier, so that the multiple temporal tuples for an object can be distinguished. For each object then we have a nest of temporal tuples. We need to make explicit the underlying domain for the event times. If the time of the event is not given, the data-entry time is used as a surrogate.

2. *Conversion of event data to histories.* We introduce a history operator **H** in Section 4.2 that converts event information to a history by finishing and starting of intervals. An interval is created for every event of an object, intervals are closed by finishing events, and adjoining intervals are merged to create larger intervals.

3. *Retrieval of information.* We next extend the set of operations to permit a larger set of computations, so that we can perform all the functions needed for query answering and general data processing at a high level.

The critical step in the processing scenario outlined above occurs in Step 2. The derivation of historical intervals from start and finish point events requires an understanding of the semantics of the time domain; the history operator **H** permits specification of these semantics. By deriving a history from the source events we prepare the stage for all subsequent operations. Specifically, since a snapshot is only applied to a historical relation, no semantic interpretation is necessary when the queries are being processed.

These processes can be carried out today by conventional data-processing programs. Their formalization has the objective of being able to mechanize more of these tasks, reduce programming effort and failures, and improve efficiency. Operations carried out within DBMS's are subject to optimization, wheras optimization done within user programs tends to be spotty and inflexible.

4.1. Events and Intervals

As mentioned earlier, we will need two domains types to process temporal data:

1. Event times, and
2. Intervals between events.

Event time typically consist of the time and date of events. These two domains are complementary and distinct. Algebraic operators can convert information among these representations, either can be used to represent temporal data. Intervals are obtained by taking the difference of time-points. Unlike event times, time intervals can be added and subtracted, and therefore, new event points can be computed by adding or subtracting intervals to time-points. Intervals can be multiplied and divided by real or integer values. Furthermore, intervals can be compared as well, although the conditions are more complex, as shown in Section 5.1.2.

We consider time to be infinitely divisible, so we must treat it similarly to real numbers. In our temporal domain we introduce a new symbol *uc* which stands for "until changed." We illustrate its utility by way of an example. Consider the relation given in Figure 7.

loan type	rate (in %)	period
Prime	10	[1988,1989)
Prime	11	[1989, *uc*)

Figure 7. A temporal relation using "until changed" symbol

The interpretation given to the second tuple is that the fact represented by the tuple (the prime rate is 11%) remains true until it is changed. With this simple augmentation in the domain definition, we are now able to record information about the future values of the prime interest rate (not just the past and current values). For example, we can insert in 1990 the the prime rates for 1991. (See Figure 8.)

loan type	rate (in %)	period
Prime	10	[1988,1989)
Prime	11	[1989,1990)
Prime	10.5	[1990,1991)
Prime	10.25	[1991, *uc*)

Figure 8. A temporal relation containing future rates

If we adhere to the usual *now* notation, we find that we cannot represent the semantics of the second relation (although there is no problem with that of the first relation). The first relation can be represented by the relation in Figure 9 while the second relation is given in Figure 10. The second relation does not make sense when *now* has a value which is less than 1991.

loan type	rate (in %)	period
Prime	10	[1988,1989)
Prime	11	[1989, *now*]

Figure 9. The temporal relation in Figure 7 using "now" symbol

loan type	rate (in %)	period
Prime	10	[1988,1989)
Prime	11	[1989,1990)
Prime	10.5	[1990,1991)
Prime	10.25	[1991, *now*]

Figure 10. The temporal relation in Figure 8 using "now" symbol

The exact relationship between *now* and *uc* is as follows: Given an interval $t = [l, uc)$,

$$uc = now \quad \text{if} \quad l < now,$$
$$> now \quad \text{if} \quad l \geq now.$$

We should note that as long as the database is restricted to collecting the factual observations about the real world, *now* is adequate. As soon as we use a database for planning or commitments made into the future, the *now* semantics give the above problems, and *uc* must be used in its place. Also, we have chosen to use $[l, uc)$ instead of $[l, \infty)$ since we wish to distinguish between values (for example, death) which are true forever from those that are true until they are changed.

4.2. Introduction of Temporal Semantics

A critical step in the processing of temporal data is the conversion of event data to histories. The derivation of values of time-variant attributes at intermediate points within the historical intervals, defined by event times, requires an understanding of what actually occurs within the interval. There are several possibilities; some are shown in Table 1.

Event	Interval State	Transform	Type
Salary_change	Salary	Use start value	stable
Hiring	Workshop	Use start value	stable
Bonus	n.a.	not continuing	none
Output_measures	Productivity	Use average of points	AVG
Power_usage	Power_consumption	Use average of points	AVG
Power_usage	Power_rating	Use maximum point	MAX
Light	Illumination	use minimum point	MIN
Inventory	Growth	Use difference of points	RATE

Table 1. Historical attribute semantics

Using these historical attribute semantics, we will define in Section 6.2 a history operator **H** which permits specification of these transforms. Time-variant attribute values ($a_{\mathbf{H}}$) in histories can be computed using computations given in Table 2. As before I_s denotes the event value corresponding to the start of the interval I and I_f the value when the interval is finished.

Assumption	Computation
Stability	$a_{\mathbf{H}} = a_{I_s}$
Constant Rate of Change	$a_{\mathbf{H}} = a_{I_s} + \Delta T(a_{I_f} - a_{I_s})$
Maximum in Interval	$a_{\mathbf{H}} = max(a_{I_s}, a_{I_f})$
Minimum in Interval	$a_{\mathbf{H}} = min(a_{I_s}, a_{I_f})$
Average over Interval	$a_{\mathbf{H}} = a_{I_s} + (a_{I_f} - a_{I_s})/2$

Table 2. Introduction of semantics in the history operator

We can now derive the historical information from Figure 2 using the assumption that salaries are stable during the intervals, as shown in Figure 11. Since Figure 11 is based solely on Figure 2, it shows current information which may not be reasonable (e.g, no change in Peter's salary since June, 1980). The point here is that history relations convey information by resolving the interval value. The relation is Figure 11 contains explicit answer to the query about Peter's salary on 1Mar80. If a query specifies an interval, say [2Feb80,4Jul80), then multiples tuples might be returned.

Name	Salary	Dept	Period
Peter	30K	Shoe	[1Jan80,4Jun80)
Peter	32K	Shoe	[4Jun80, *uc*)
Margi	30K	Shoe	[6Mar78,1Jan79)
Margi	31K	Shoe	[1Jan79,4Jun80)
Margi	32K	Shoe	[4Jun80, *uc*)
Jack	30K	Linen	[4Jul79,4Dec80)
Jack	30K	Shoe	[4Dec80, *uc*)

Figure 11. History relation derived from the journal in Figure 2

When the history operation converts event information to a history, it depends on the assumption that the journal of the events has been made complete so that it can generate a correct history, especially if different histories are to be combined. For the relation in Figure 4, we need to add the information that the start date for employee is the first of the month. Once this is done, we can apply the history operator to the relation in Figure 4, and the system can then combine the two histories to form a common history which is shown in Figure 12.

It is worth noting in Figure 12 that since the last tuple for the employee John does not contain "*uc*," we know that John is no longer employed by the company. Thus, if we assure that each object history is complete between the extreme start- and finish-points, then deleted objects can be easily represented in history relations.

Name	Salary	Dept	Period
Peter	32K	Toys	[1Jan77,1Jan78)
Peter	30K	Shoe	[1Jan80,4Jul80)
Peter	32K	Shoe	[4Jun80, *uc*)
Margi	30K	Shoe	[6Mar78,1Jan79)
Margi	31K	Shoe	[1Jan79,4Jun80)
Margi	32K	Shoe	[4Jun80, *uc*)
Jack	30K	Linen	[4Jul79,4Dec80)
Jack	30K	Shoe	[4Dec80, *uc*)
John	15K	Shoe	[1Jan77,1Nov81)
John	25K	Shoe	{1Jan81,1Nov83)

Figure 12. Combined history

The history relation for the journal in Figure 3 is given in Figure 13, assuming that the average value is to be used at intermediate points of an interval. The benefit here is that we can now easily determine the snapshot at time 9:15, which is given in Figure 14.

Location	Degrees	Interval
Class room	19.5C	[8:00, 9:00)
Class room	22.5C	[9:00,10:00)
Kitchen	24.0C	[9:00,10:00)

Figure 13. History relation derived from the journal in Figure 3

Location	Degrees
Class room	22.5C
Kitchen	24.0C

Figure 14. A correct snapshot

5. Temporal Operations

In this section, we define various operators on our two domain types (event times and intervals), most of which are taken from. allen Using these operators, it is straightforward to extend the usual relational operations (select, project, join, and others) for event as well as history relations.

We use the variables t, u, v to denote the time and date of events, and variables T, U, V to denote time intervals. We define two functions, *min* and *max*, which, when applied to intervals, return the starting and finishing event times. Thus, if T is the interval $[t, u)$, then $min(T) = t$ and $max(T) = u$.

Let T, U, V to denote the time intervals between events t and u, u and v, and t and v, respectively. We denote by $|T|$ the size of an interval T. Then the following transformations hold:

$$t = min(T) = min(V) \qquad |T| = u - t$$

$$u = max(T) = min(U) \qquad |U| = v - u$$

$$v = max(U) = max(V) \qquad |V| = v - t$$

$$|V| = |T| + |U|$$

5.1. Temporal Comparison

For database searching, the primary operations is comparison. It can be applied to pairs of events, to pairs of intervals, and to combinations of both. Temporal comparison occurs so frequently in daily life, we have words (e.g. before, after, etc.) for the various cases with fairly well understood semantics. Using these words can help avoid errors when dealing with historical databases.

5.1.1. Event Comparisons

The comparison operations for events are listed in Table 3.

Name	*Definition*
t BEFORE u	$t < u$
t AT u	$t = u$
t AFTER u	$t > u$

Table 3. Comparing events

5.1.2. Interval Comparisons

When we want to deal with independent events, and see if they coincide, interval comparison is appropriate. First we may want to see if one interval is longer or shorter than another, and then we may want to check how intervals occur relative to each other. The two size comparisons given in Table 4 are symmetric.

Name	*Definition*
T LONGER U	$(max(T) - min(T)) > (max(U) - min(U))$
T SHORTER U	$(max(T) - min(T)) < (max(U) - min(U))$

Table 4. Comparing Interval Sizes

Dealing with the relative position of intervals is necessary when we evaluate actions that take time to develop an effect. For instance, a drug has to be in the body for some time before it can affect a disease. An analysis of medical data has to take those lags into account.

Comparison of interval positions is more complex than comparing event occurrences. When we compare intervals the comparison must hold for all time points within the intervals. We assess completeness of our operations by considering the start-points ($min(T)$), finish-points ($max(T)$), and the relative length of both intervals. Table 5 presents the terms and illustrates eleven conditions. Four symmetric cases, *DURING* and *SPANS*, and the last three conditions are symmetric for U, T are listed to provide the full terminology. They provide the basis for a complete comparison algebra on intervals, so that no decomposition into event timestamps is required. The three event comparisons for the two pairs of points in each interval actually permit $2 \times 3^2 \rightarrow 18$ comparison cases. Of those five are invalid, since they require that we have intervals with $max(U) < min(U)$. Three cases are symmetric for U longer than T and vice versa. For two of those cases, *STARTS* and *FINISHES*, we do not have common distinctive terms; they are not shown in Table 5.

	Name	*Definition*	
1.	*T BEFORE U*	*max(T) < min(U)*	
2.	*T UNTIL U*	*max(T) = min(U)*	
3.	*T LEADS U*	*min(T) ≤ min(U), max(T) < max(U), min(U) < max(T)*	
4.	*T STARTS U*	*min(T) = min(U), max(T) < max(U)*	
5.	*T EQUALS U*	*min(T) = min(U), max(T) = max(U)*	[unique]
6.	*T DURING U*	*min(T) > min(U), max(T) <max(U)*	
7.	*T SPANS U*	*min(T) < min(U), max(T) > max(U)*	[Complement of 6.]
8.	*T FINISHES U*	*min(T) > min(U), max(T) = max(U)*	[Symmetric with 4.]
9.	*T LAGS U*	*min(T) ≥ min(U), max(T) > max(U), min(T) < max(U)*	[Symmetric with 3.]
10.	*T FROM U*	*min(T) = max(U)*	[Symmetric with 2.]
11.	*T AFTER U*	*max(T) > min(U)*	[Symmetric with 1.]

Table 5. Comparing intervals

5.1.3. Comparing Events and Intervals

Although algebras typically do not permit comparison of different domain types, we find that such comparisons occur frequently in practice. Quite unambiguous are the comparisons shown in Table 6. The terms themselves are overloaded, so that a system has to know the temporal domain type to carry the operations correctly.

Name	Definition
t BEFORE T	$t < min(T)$
t DURING T	$t > min(T)$, $t < max(T)$
t AFTER T	$t > max(T)$

Table 6. Comparing events and intervals

5.1.4. Temporal Computation

We define some additional operations on intervals which will be useful in the next section when we use the history operator **H** to create a history from a journal. We will need to concatenate (*cat*) or shorten (*uncat*) history tuples.

Given a pair of intervals T and U, the operation $T\ cat\ U$ is defined iff $T\ UNTIL\ U$ holds, in which case

$$T\ cat\ U = [min(T),\ max(U)).$$

The operation $T\ uncat\ U$ is defined iff either $T\ STARTS\ U$ or $T\ FINISHES\ U$ holds. If $T\ STARTS\ U$, then

$$T\ uncat\ U = [max(T),\ max(U)),$$

and if $T\ FINISHES\ U$, then

$$T\ uncat\ U = [min(U),\ min(T)).$$

When we convert event data to histories, each event tuple is converted into a history tuple by replacing each time attribute by an interval. The *cat* and *uncat* operations are used to replace different history tuples with adjoining intervals for the same object by a single history tuple with larger interval as the time attribute.

6. Operations on Temporal Relations

We now deal with proper histories only. Relations that are less constrained introduce more complexity and will not be discussed here. One way to ensure that a history is proper is by augmenting each object identifier by the time attribute.

We define two operations: **H** to compose a proper history >From journal data and **I** to create a snapshot at a given instance in time from a proper history. But first we show how we can derive a temporal sequence from a journal or a history.

We denote relations throughout as $R(S, E)$ where E is the temporal extension with n temporal tuples e_i. An attribute containing temporal information is denoted as A_t, with event or interval values a_i in tuple e_i.

6.1. The Temporal Sequence

The constraints imposed on journals and histories mean that we can establish a temporal sequence. Within a journal or a history we can speak of the *NEXT*, *CURRENT*, or *PRIOR* event or interval. When there is no *NEXT* tuple the result is *uc*. When there is no *PRIOR* tuple, the result is Λ.

A proper history requires a contiguous sequence. If there is truly an interval with unknown information it must be represented explicitly. This is done by specifying a tuple with the interval and a value of null (Λ) or *none* or *n. a.* if it is known to be missing or not applicable. Specifically,

$$NEXT(e_c) \ := \ \{e_i : object_c = object_i,\ min(a_i) = \max(a_c)\}.$$

The *PRIOR* tuple is found similarly. *CURRENT* operation is defined in Section 6.2.1.

6.2. The History Operator

The history operator **H** converts event information to a history. It depends on the assumption that the journal of the events has been made complete. We showed an example of what might be needed to complete a journal in Section 4.2. There are two phases to generating a history:

1. An interval is created for every event on an object.
2. Adjoining intervals are merged to create larger intervals.

More formally, suppose we are given a journal R with attributes S and a temporal extension E. Let a_i denote the interval in tuple e_i. The history operation **H** is defined as follows:

$$\mathbf{H}(\mathrm{R(S,E)}) \ := \ \{\ e_i = (\ object_i,\ a_i)\}$$

where $object_i \in S$ and for each $e_d \in S$, whenever $NEXT(e_i) = e_d$, $a_i = a_i\ cat\ a_d$.

Any tuple e_d of R processed in this manner is not distinctly represented in the history **H**(R). This reduction is reminiscent of the reduction that is required to avoid duplicate entries during projection, Here we wish to avoid duplicate information in adjoining intervals.

6.2.1. Computation of Snapshots

A *snapshot* (**I**) provides object information as of a given point in time. The most frequent snapshot is one that obtains *CURRENT* information, but any other time value may be used. The definition of result values from temporal relations is quite straightforward. The snapshot at a time t is determined as follows:

$$\mathbf{I}(\mathrm{R}(\mathrm{S},\mathrm{E})) \ := \ \{ \ object_i : e_i \in R \text{ and } min(a_i) \leq t < max(a_i) \}$$

All other relational operations (such as projection, selection, and join) map straightforwardly into event and history relations, specifics will be given in wiederhold semantic database design mcgraw-hill

7. Conclusion

In this paper, we present an algebra which addresses the issue of time granularity in temporal relations. The algebra permits merging, abstraction, and other computations to reduce temporal information. Often the factual data collected at the source is at too fine a granularity to be useful to the decision maker. The data have to be aggregated, merged with other data, etc., before we have the information needed for decision making. Many of these tasks are traditional functions of application programs. However, when applications are shared by many users, it is important that they be consistent. Since one of the roles of a DBMS is to assure consistency, it is reasonable that not only shared data, but also shared computations are handled by the DBMS.

We have shown that it is wise to have both explicit journal and history representations in view of the need to make semantic choices when converting events to histories and snapshot results explicit, as shown in Section 4.2. An implicit conversion cannot capture the range of options that might be needed.

With these considerations, we have also defined a representation and corresponding operations for temporal data. The relational temporal algebra is a straightforward and sound extension of the relational algebra. A requirement for an algebra is that the results of operations on objects are in turn objects of the same type. The relational algebra satisfies that condition since all results are in turn relations. An implementation need not be restricted to today's traditional relational DBMS's. Formulation as nested relations roth korth silber theory non normal form relati databa univ texas austin dec 1984 revise jan 1986 mark roth henry korth abraha silber extend algebr calcul nestedrelati databa acm tods decemb 1988 are likely to provide more efficient processing.

An obvious question is whether the proposed extensions are worth the increased complexity. We believe that this is so. Many data-processing applications deal with temporal information. The operational semantics of time are not well captured by operations that expect integer, real, or characterstring types. Programmers have to repeatedly code operations which recreate temporal semantics. However, for any single application a less general collection of representations and operations are adequate so that the programmer selected representations, the granularity, and the result types differ from case to case. Mismatch problems when combining temporal information are frequent. Errors are common and high-level integration is inhibited Chaudhury

By using *uc* in the temporal domain instead of the usual *now*, we can record in the database expectations about the future. It would be interesting to extend this capability to allow consideration of alternate futures. As far as we know those complexities have never been rigorously addressed.

Acknowledgements

The authors wish to thank Surajit Chaudhuri, Oliver Costich, Michael Walker, and the users of the TOD and MEDLOG systems for many valuable discussions. Sam Kamens has recently validated the approach using a preprocessor for SYBASE. Kamens