Temporal Databases

Richard T. Snodgrass

Department of Computer Science, University of Arizona, Tucson, AZ 85721
rts@cs.arizona.edu

Abstract. This paper summarizes the major concepts, approaches, and implementation strategies that have been generated over the last fifteen years of research into data base management system support for time-varying information. We first examine the time domain, its structure, dimensionality, indeterminacy, and representation. We then discuss how facts may be associated with time, and consider data modeling and representational issues. We survey the many temporal query languages that have been proposed. Finally, we examine the impact to each of the components of a DBMS of adding temporal support, focusing on query optimization and evaluation.

1 Introduction

Time is an important aspect of all real-world phenomena. Events occur at specific points in time; objects and the relationships among objects exist over time. The ability to model this temporal dimension of the real wold is essential to many computer applications, such as econometrics, banking, inventory control, accounting, law, medical records, land and geographical information systems, and airline reservations.

Conventional databases represent the state of an enterprise at a single moment of time. Although the contents of the database continue to change as new information is added, these changes are viewed as modifications to the state, with the old, out-of-date data being deleted from the database. The current contents of the database may be viewed as a shapshot of the enterprise. In such systems the attributes involving time are manipulated solely by the application programs; the database management system (DBMS) interprets dates as values in the base data types. No conventional system interprets temporal domains when deriving new relations.

Application-independent DBMS support for time-varying information has been an active area of research for about 15 years, with approximately 400 papers generated thus far @cite[Bolour82,mckenzie86A,stam88,soo91]. This paper attempts to capture and summarize the major concepts, approaches, and implementation strategies that have been generated by that research.

We first examine the time domain: its structure, dimensionality (interestingly, there are several time dimensions) and temporal indeterminacy, followed by issues in representing values in this domain. We demonstrate that time is actually more complex than the spatial domain, as the former's dimensions are non-homogeneous.

Section 3 follows a similar organization in examining how facts may be associated with time. Data modeling issues are first examined, then representational alternatives are explored, with frequent comparisons with space. We briefly consider how facts may be simultaneously associated with both space and time, a common phenomena in land and geographic information systems.

We next consider languages for expressing temporal queries. We illustrate the various types of queries through examples in the temporal query language TQuel, and briefly appraise various standards efforts.

Temporal DBMS implementation is the topic of Sec. 5. We examine the impact to each of the components of a DBMS of adding temporal support, discussing query optimization and evaluation in some detail.

We conclude with a summary of the major accomplishments and disappointments of research into temporal databases.

We omit one major aspect, that of database design, due to lack of space.

2 The Time Domain

In this section we focus on time itself: how it is modeled and how it is represented. The next section will then combine time with facts, to model time-varying information.

2.1 Structure

We initially assume that there is one dimension of time. The distinctions we address here will apply to each of the several dimensions we consider in the next section.

Early work on *temporal logic* centered around two structural models of time, *linear* and *branching* @cite[vanBenthem82]. In the linear model, time advances from the past to the future in a totally ordered fashion. In the branching model, also termed the *possible futures* model, time is linear from the past to now, where it then divides into several time lines, each representing a potential sequence of events @cite[Worboys90A]. Along any future path, additional branches may exist. The structure of branching time is a tree rooted at now. The most general model of time in a temporal logic represents time as an arbitrary set with a partial order imposed on it. Additional axioms introduce other, more refined models of time. For example, linear time can be specified by adding an axiom imposing a total order on this set. Recurrent processes may be associated with a *cyclic* model of time @cite[Chomicki89A, Lorentzos88C, Lorentzos88B].

In spatial models, there is much less diversity, and a linear model is generally adequate.

Axioms may also be added to temporal logics to characterize the *density* of the time line @cite[vanBenthem82]. Combined with the linear model, *discrete* models of time are isomorphic to the natural numbers, implying that each point in time has a single successor @cite[Clifford85]. *Dense* models of time are isomorphic to either the rationals or the reals: between any two moments of time

another moment exists. *Continuous* models of time are isomorphic to the reals, i.e., they are both dense and unlike the rationals, contain no "gaps."

In the continuous model, each real number corresponds to a "point" in time; in the discrete model, each natural number corresponds to a nondecomposable unit of time with an arbitrary duration. Such a nondecomposable unit of time is refered to as a *chronon* @cite[Ariav86A,Clifford87B] (other, perhaps less desirable, terms include "time quantum" @cite[Anderson82], "moment" @cite[Allen85B], "instant" @cite[Gadia86A] and "time unit" @cite[Navathe87,Tansel86E]). A chronon is the smallest duration of time that can be represented in this model. It is not a point, but a line segment on the time line.

Although time itself is generally perceived to be continuous, most proposals for adding a temporal dimension to the relational data model are based on the discrete time model. Several practical arguments are given in the literature for this preference for the discrete model over the continuous model. First, measures of time are inherently imprecise @cite[ANDERSON82, CLIFFORD85]. Clocking instruments invariably report the occurrence of events in terms of chronons, not time "points." Hence, events, even so-called "instantaneous" events, can at best be measured as having occurred during a chronon. Secondly, most natural language references to time are compatible with the discrete time model. For example, when we say that an event occurred at 4:30 p.m., we usually don't mean that the event occurred at the "point" in time associated with 4:30 p.m., but at some time in the chronon (perhaps minute) associated with 4:30 p.m. @cite[ANDERSON82, CLIFFORD87B, Dyreson92D]. Thirdly, the concepts of chronon and interval allow us to naturally model events that are not instantaneous, but have duration @cite[ANDERSON82]. Finally, any implementation of a data model with a temporal dimension will of necessity have to have some discrete encoding for time (Sec. 2.4).

Space may similarly be regarded as discrete, dense, or continuous. Note that, in all three of these alternatives, two separate space-filling objects cannot be located in the same point in space and time: they can be located in the same place at different times, or at the same time in different places.

Axioms can also be placed on the *boundedness* of time. Time can be bounded orthogonally in the past and in the future. The same applies to models of space.

Models of time may include the concept of *distance* (most temporal logics do not do so, however). Both time and space are *metrics*, in that they have a distance function satisfying four properties: (1) the distance is nonnegative, (2) the distance between any two non-identical elements is non-zero, (3) the distance from time α to time β is identical to the distance from β to α, and (4) the distance from α to γ is equal to or greater than the distance from α to β plus the distance from β to γ (the triangle inequality).

With distance and boundedness, restrictions on range can be applied. The scientific cosmology of the "Big Bang" posits that time begins with the Big Bang, 14 ± 4 billion years ago. There is much debate on when it will end, depending on whether the universe is *open* or *closed* (Hawking provides a readable introduction to this controversy @cite[Hawking88]). If the universe is closed then time will

have an end when the universe collapses back onto itself in what is called the "Big Crunch." If it is open then time will go on forever.

Similar considerations apply to space. In particular, an open universe implies unbounded space. However, many applications assume a bound as well as a range; geographical information systems don't need to contend with values greater than approximately 70 million meters.

Finally, one can differentiate *relative* time from *absolute* time (more precise terms are *unachored* and *anchored*). For example, "9A.M., January 1, 1992" is an absolute time, whereas "9 hours" is a relative time. This distinction, though, is not as crisp as one would hope, because absolute time is with respect to another time (in this example, midnight, January 1, A.D. 1). We will show in Sec. 2.4 how to exploit this interaction. Relative time differs from distance in that the former has a direction, e.g., one could envision a relative time of -9 hours, whereas a distance is unsigned.

One can also differentiate between relative and absolute space, with the same provisos.

2.2 Dimensionality

Time is multi-dimensional @cite[Snodgrass86A]. *Valid time* concerns the time a fact was true in reality. The valid time of an event is the wall clock time at which the event occurred in the real world, independent of the recording of that event in some database. Valid times can also be in the future, if it is known that some fact will be true at a specified time in the future. *Transaction time* concerns the time the fact was present in the database as stored data. The transaction time (an interval) of an event identifies the transactions that inserted the information about the event into the database and removed this information from the database. As with space, these two dimensions are orthogonal. A data model supporting neither is termed *snapshot*, as it captures only a single snapshot in time of both the database and the enterprise that the database models. A date model supporting only valid time is termed *historical*; one that supports only transaction time is termed *rollback*; and one that supports both valid and transaction time is termed *bitemporal* (*temporal* is a generic term implying some kind of time support).

Figure 1 illustrates a *single* bitemporal relation (i.e., table) composed of a sequence of historical states indexed by transaction time. It is the result of four transactions starting from an empty relation: (1) three tuples (i.e., rows) were added, (2) one tuple was added, (3) one tuple was added and an existing one terminated (logically deleted), and (4) the starting time of a previous tuple [the middle one added in transaction (1)] was changed to a somewhat later time (presumably the original starting time was incorrect) and a recently added tuple (the bottom one) was deleted (presumably it should not have been there in the first place.) Each update operation involves copying the historical state, then applying the update to the newly created state. Of course, less redundant representations than the one shown are possible. While we'll consider only linear time, branching transaction time provides a useful model for *versioning* in

computer-aided design tasks @cite[Dittrich88] such as CAD @cite[Ecklund87, Katz86] and CASE @cite[Bernstein87A, Hsieh89].

Fig. 1. A bitemporal relation

A different depiction that has proven useful is to time-stamp each fact with a *bitemporal element*[1], which is a set of *bitemporal chronons*. Each bitemporal chronon represents a tiny rectangle in valid-time/transaction-time space. Figure 2 shows the bitemporal element associated with the middle tuple of Fig. 1. Historical and rollback databases effectively record *historical chronons* and *rollback chronons*, respectively.

Fig. 2. A bitemporal element

[1] This term is a generalization of *temporal element*, previously used to denotes a set of single dimensional chronons @cite[Gadia88B]. An alternative, equally desirable term is *bitemporal lifespan* @cite[Clifford87A].

While valid time may be bounded or unbounded (as we saw, cosmologists feel that it is at least bounded in the past), transaction time is bounded on both ends. Specifically, transaction time starts when the database is created (before which time, nothing was stored), and doesn't extend past now (no facts are known to have been stored in the future). Changes to the database state are required to be stamped with the current transaction time. Hence, rollback and bitemporal relations are *append-only*, making them prime candidates for storage on write-once optical disks. As the database state evolves, transaction times grow monotonically. In contrast, successive transactions may mention widely varying valid times. For instance, the fourth transaction in Fig. 1 added information to the database that was transaction time-stamped with time 4, while changing a valid time of one of the tuples to 2.

The three dimensions in space are truly orthogonal and homogeneous, the one exception being the special treatment sometimes accorded elevation. In contrast, the two time dimensions are not homogeneous; transaction time has a different semantics than valid time. Valid and transaction time *are* orthogonal, though there are generally some application dependent correlations between the two times. As a simple example, consider the situation where a fact is recorded as soon as it becomes valid in reality. In such a *specialized* bitemporal database, termed *degenerate* @cite[Jensen92], valid and transaction time are identical. As another example, if a cloud cover measurement is recorded at most two days after it was valid in reality, and if it takes at least six hours from the measurement time to record the measurement, then such a relation is *delayed strongly retroactively bounded with bounds six hours and two days*.

Multiple transaction times may also be stored in the same relation, termed *temporal generalization* @cite[Jensen92]. These times may also be related to each other, or to the valid time, in various specialized ways. For example, a particular value for the reflectivity of a cloud over a point on the Earth may be recorded by an Earth Sensing Satellite at a particular time. Here, the valid time and transaction time are correlated, and the satellite's database may be considered to be a degenerate bitemporal database. Later, this data is sent to a ground station and stored; the transaction time of the stored data will be different from the valid time; this database may be classified as a *bounded retroactive* database. Later still, the data from several ground stations are merged into a central database, storing the original valid time, the transaction time of the recording into the central database, and the *inherited* transaction time when the data was stored in the ground station database. All three times may be needed, for instance, if data massaging was done with algorithms that were being improved over time. Such multiple transaction time dimensions do not have a spatial analogue.

2.3 Indeterminacy

Information that is *historically indeterminate* can be characterized as "don't know exactly when" information. This kind of information is prevalent; it arises in various situations, including the following.

- *Finer system granularity* — In perhaps most cases, the granularity of the database does not match the precision to which an event time is known. For example, an event time known to within one day and recorded on a system with time-stamps in the granularity of a millisecond happened sometime *during* that day, but during which millisecond is unknown.
- *Imperfect dating techniques* — Many dating techniques are inherently imprecise, such as radioactive and Carbon-14 dating. All clocks have an inherent imprecision @cite[Dyreson92D].
- *Uncertainty in planning* – Projected completion dates are often inexactly specified, e.g., the project will complete three to six months from now.
- *Unknown or imprecise event times* — In general, event times could be unknown or imprecise. For example, if we do now know when an individual was born, the individual's date of birth could be recorded in the database as either unknown (she was born between now and the beginning of time) or imprecise (she was born between now and 100 years ago).

There have been several proposals for adding historical indeterminacy to the time model @cite[Gadia92A,Kahn77A], as well as more specific work on accommodating multiple time granularities @cite[Ladkin87A,Wiederhold91A]. The *possible chronons* model unifies treatment of both aspects @cite[Dyreson92D]. In this model, an event is *determinate* if it is know when (i.e., during which chronon) it occurred. A determinate event cannot overlap two chronons. If it is unknown when an event occurred, but known that it did occur, then the event is historically indeterminate. The indeterminacy refers to the *time* when the event occurred, not *whether* the event did or did not occur.

Two pieces of information completely describe an indeterminate event: a *set of possible chronons* and an *event probability distribution*. A single chronon from the set of possible chronons denotes when the indeterminate event actually occurred. However, it is unknown *which* possible chronon is the actual one. The event probability distribution gives the probability that the event occurred during each chronon in the set of possible chronons.

The implementation of the possible chronons model supports a fixed, minimal chronon size. Multiple granularities are handled by representing the indeterminacy explicitly. For example, if the underlying chronon is a microsecond and an event is known to within a day, then this indeterminate event would be associated with a set of 86,400,000 possible chronons, and perhaps a uniform event probability distribution.

As a practical matter, events that occurred in the prehistoric past cannot be dated as precisely as events that occur in the present. There is an implicit "telescoping view" of time. Dating of recent events can often be done to the millisecond while events that occurred 400 million years ago can be dated to, perhaps at best the nearest 100,000 years. Dating future events is also problematic. It is impossible to say how many seconds will be between Midnight January 1, 1992 and Midnight January 1, 2300 because we don't know how many leap second will be added to correct for changes in the rotational clock. We can guess at the number of seconds, but "leap shifts" to the current clock are likely to

invalidate our guess.

Historical indeterminacy occurs only in valid time. The granularity of a transaction time time-stamp is the smallest inter-transaction time. Transaction times are always determinate since the chronon during which a transaction takes place is always known.

Most of the above may be applied to space. Information that is *spatially indeterminate* can be characterized as "don't know exactly where" information. It is also prevalent, due to granularity concerns, measurement techniques, and unknown or imprecise location specifiers. One could envision an analogous "possible space quanta" model that could capture the variety of spatial indeterminacy. The telescoping view phenomenon also occurs in space, as distant locations are less precisely known.

As with time, a spatial *data granularity* coarser than the *database management system (DBMS) granularity* is often adopted. One of the more common models, Type 0 (Sec. 2.2), covers the two-dimensional space with a grid, with point locations and associated attributes reported to the nearest cell center. In this model, the data is a multiple of the underlying DBMS granularity. For example, the DBMS granularity might be a meter, with all location specifiers being expressed in this unit, while the grid cells may be 2 kilometers on a side.

2.4 Representation

Since time and space are metrics, a system of units is required to represent particular events or locations. A time-stamp or location specifier has a *physical realization* and an *interpretation*. The physical realization is a pattern of bits while the interpretation is the meaning of each bit pattern, that is, the time or location each pattern represents.

Interpretation. For time, the central unit is the *second*. However, there are at least seven different definitions of this fundamental unit @cite[Dyreson92D].

Apparent solar — 1/86400 of the interval from noon to noon; varies from day to day.

Mean solar (UT0) — 1/86400 of a *mean solar day*, averaged over a year; varies from year to year.

Mean sidereal — 1/86400 of a *mean sidereal day*, measuring the rotation of the Earth with respect to a distant star; varies from year to year.

UT1 — UT0 corrected for polar wander.

UT2 — UT1 corrected for seasonal variations.

Ephemeris — mean solar second for the year 1900; does not vary. This was the standard definition from 1960 to 1967.

International System of Units (SI) — the duration of 9,192,631,770 periods of the radiation corresponding to the transition between the two hyperfine levels of cesium-133 atoms @cite[Petley91].

When a range of less than 10,000 years is supported, the differences between these definitions are generally inconsequential, except for the apparent solar

second, which varies by 1% over the course of a year. When ranges of several billion years are supported, however, all of these definitions differ significantly.

Different regions of the time line are used by different communities. For example, apparent solar time is important to historians, who care about whether something happened in the daylight or in darkness, as well as to users of cadastral (real estate) databases, which utilize civil calendars @cite[Hunter90]. Ephemeris time is used by astronomers, while the SI second is the basis for radioactive dating used by geochronologists. Because of these different needs, as well as the telescoping view of time, we have proposed a specific temporal interpretation termed the *base-line clock* that constructs a time-line by using different well-defined clocks in different periods. This clock, shown in Fig. 3 (not to scale), partitions the time line into a set of contiguous *periods*. Each period runs on a different clock. A *synchronization point*, where two clocks are correlated, delimits a period boundary. The synchronization points occur at Midnight on the specified date.

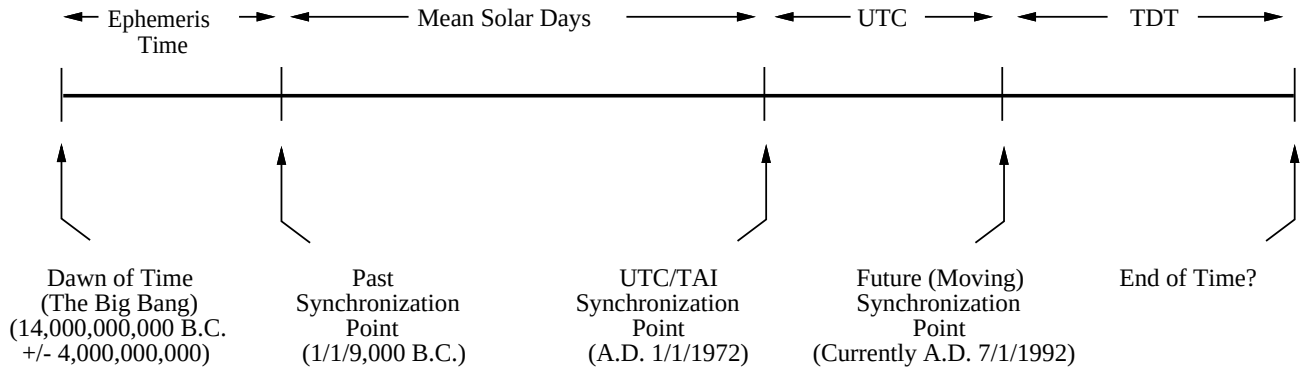

Fig. 3. The base-line clock

From the Big Bang until Midnight January 1, 9000 B.C. the base-line clock runs on ephemeris time. This clock is preferable to the solar clock since ephemeris time is independent of the formation of the Earth and the Solar System. Also, we prefer using the ephemeris clock to the solar clock because an ephemeris year is a fixed duration, unlike the tropical year. For historic events, 9000 B.C. to January 1, 1972, the base-line clock follows the mean solar day clock. Historic events are usually dated with calendars. Calendar dates invariably count days and use an intercalation rule to relate the number of days to longer-term celestial clocks, e.g., the Gregorian calendar relates days to months and tropical years. At Midnight January 1, 1972 the base-line clock switches to Universal Coordinated Time (UTC). Midnight January 1, 1972 is when UTC was synchronized with the SI definition of second and the current system of leap seconds was adopted. The base-line clock runs on UTC until one second before Midnight, July 1,

1992. This is the next time at which a leap second may be added (a leap second will be added on this date according to the latest International Earth Rotation Service bulletin @cite[USNO92]). After Midnight July 1, 1992, until the "Big Crunch" or the end of our base-line clock, the base-line clock follows Terrestrial Dynamic Time (TDT), an "idealized atomic time" @cite[Guinot88] based on the SI second, since both UTC and mean solar time are unknown and unpredictable.

The situation is much simpler for space. Here, the (SI) *meter* is the commonly accepted unit, with a single accepted definition, the length of the path traveled by light in vacuum during a time interval of 1/299,792,458 of a second @cite[Petley91]. Distance is defined in terms of time, rather than the other way around, because time can be measured more accurately (1 part in 10^{10} over long intervals and 1 part in 10^{15} for between a minute and a day @cite[Quinn91,Ramsey91C]).

The base-line clock and its representation are independent of any calendar. We used Gregorian calendar dates in the above discussion only to provide an informal indication of when the synchronization points occurred. Many calendar systems are in use today; example calendars include academic (years consists of semesters), common fiscal (financial year beginning at the New Year), federal fiscal (financial year beginning the first of October) and time card (8 hour days and 5 day weeks, year-round). The usage of a calendar depends on the cultural, legal, and even business orientation of the user @cite[Soo92]. A DBMS attempting to support time values must be capable of supporting all the multiple notions of time that are of interest to the user population.

Space also has multiple notions, though with less variability. The metric, U.S., and nautical unit systems are the most prevalent. Both time and space have precisely defined underlying semantics that may be mapped to multiple display formats. The spatial base-line is less complex than that for time; it consists of a single measure, the meter.

Physical Realization. The base-line clock defines the meaning of each time-stamp bit pattern in the physical realization of a time-stamp. The chronons of the base-line clock are the chronons in its constituent clocks. We assume that each chronon is one second in the underlying constituent clock. A chronon may be denoted by an integer, corresponding to a *single (DBMS) granularity*, or it may be denoted by a sequence of integers, corresponding to a *nested granularity*. For example, if we assume a granularity of a second relative to Midnight, January 1, 1980, then in a single granularity the integer 164,281,022 denotes 9:37:02AM March 15, 1985. If we assume a nested granularity of ⟨year, month, day, hour, minute, second⟩, then the sequence ⟨6,3,15,9,37,2⟩ denotes that same time.

Various time-stamps are in use in commercial database management systems and operating systems; a summary is provided in Table 1. This table compares formats from operating systems (specifically Unix, MSDOS, and the MacIntosh operating systems), the database systems DB2 @cite[Date90B], SQL2 @cite[Date89A,Melton90], and several proposed formats to be discussed shortly. The SQL2 `datetime` time-stamp appears twice in the comparison, once with its optional fractional second precision field set to microseconds, and once without

the optional field. The last three representations have recently been proposed, and will be discussed shortly.

SYSTEM	*Size (bytes)*	*Range*	*Granularity*
OS (several)	4	≈ 136 years	second
DB2 —`date`	4	10,000 years	day
DB2 —`time`	3	24 hours	second
DB2 —`timestamp`	10	10,000 years	microsecond
SQL2 —`datetime`	20	10,000 years	second
SQL2 —fractional `datetime`	27	10,000 years	microsecond
Low Resolution	8	≈ 36 billion years	second
High Resolution	8	≈ 17400 years	microsecond
Extended Resolution	12	≈ 36 billion years	nanosecond

SYSTEM	*Number of Components*	*Bytes Needed*	*Space Efficiency*
OS (several)	1	4	100%
DB2 —`date`	3	2.9	71%
DB2 —`time`	3	2.2	72%
DB2 —`timestamp`	7	7.3	73%
SQL2 —`datetime`	5	4.8	24%
SQL2 —fractional `datetime`	6	7.3	27%
Low Resolution	1	7.6	95%
High Resolution	3	7.5	93%
Extended Resolution	4	11.3	94%

Table 1. A comparison of some physical layouts

Size is the number of bytes devoted to the representation, while *range* refers to the difference between the youngest and oldest time values that can be represented. The *granularity* of a time-stamp is the precision to which a time value can be represented. If a representation has more than one *component*, it is of a nested granularity. The extreme is the DB2 `timestamp` representation, in which the year, month, day, hour, minute, second, and number of microseconds are individually represented. *Space efficiency* is a measure of how much of the representation is actually needed. It is computed as a percentage of the number of bits needed to represent every chronon in the temporal interpretation (DB2 and SQL2 both use the Gregorian calendar temporal interpretation) versus the number of bits devoted to the physical realization. The minimum number of *bytes needed* to store the number of chronons dictated by a time-stamp's granularity and range is shown as a separate column. For instance, SQL2's `datetime` time-stamp uses 20 bytes, but only 4.8 bytes of space are needed to store a range of 10,000 years to the granularity of a second.

The evaluated time-stamps fall into two camps: OS-style time-stamps and database-style time-stamps. OS-style time-stamps have a limited range and granularity; these limitations are dictated by the size of the time-stamp. OS-style time-stamps are maximally space efficient, having a single granularity. The time-stamp itself is merely a count of the number of chronons that have elapsed since the origin in the temporal interpretation. But optimal space efficiency is attained at the expense of some *time efficiency*.

In contrast, database-style time-stamps, as exemplified by the DB2 `timestamp` format, are generally larger than OS-style time-stamps; they have a wider range and finer granularity. But, as a group, they also have poorer space utilization, having a nested granularity. The advantage of representing values separately is that they can be quickly accessed. Extracting the number of years from an OS-style time-stamp is more involved than performing a similar task on an DB2 `timestamp`.

These existing time-stamp representations suffer from inadequate range, too coarse a granularity, excessive space requirements, or a combination of these drawbacks. Finally, none of the time-stamps are able to represent *historical indeterminacy*. Consequently, we recently proposed new time-stamp formats, incorporating features from both the OS-style and database-style time-stamps. These formats combine high space efficiency with high time efficiency for frequent time-stamp operations @cite[Dyreson92D].

There is a natural tradeoff between range and granularity in time-stamp development. Using the same number of bits, a time-stamp designer can make the granularity coarser to extend the range, or she can limit the range to support finer granularities. These observations imply that a format based on a single 32 bit word is inadequate for our purposes; there are simply not enough bits. Since we wanted to keep the time-stamp formats on 32 bit word boundaries, we allocated the next word increment, 64 bits, to our basic format. Using 64 bits, it is possible to represent all of time (that is, a range of 34 billion years) to the granularity of a second, and a range of historical time to granularities much finer than a second.

There are three basic types of time-stamps: events, spans, and intervals @cite[SOO92]. We developed three new event time-stamp formats, with different *resolutions*, high, low, and extended. Resolution is a rough measure of a time-stamp precision. The low resolution format can represent times to the precision of a second. High resolution narrows the precision to a microsecond while extended resolution is even more precise; it can represent times to the precision of a nanosecond. High resolution has limited range but extended precision while low resolution has extended range but limited precision. Extended resolution handles those uncommon cases where the user wants both an extended range and an extended precision, at the cost of an extra word of storage.

Interval time-stamp formats are simply two event time-stamps, one for the starting event of the interval and one for the terminating event of the interval. We use this representation because all operations on intervals are actually operations on their delimiting events @cite[SOO92A]. There are sixteen interval time-stamp

formats in toto. The type fields in the delimiting event time-stamps distinguish each format.

Spans are relative times. There are two kinds of spans, fixed and variable @cite[SOO92]. A *fixed span* is a count of chronons. It represents a fixed duration (in terms of chronons) on the base-line clock between two time values. The fixed span formats use exactly the same layouts as the standard event formats, with a different interpretation. The chronon count in the span representation is independent of the origin, instead of being interpreted as a count from the origin. The sign bit indicates whether the span is positive or negative rather than indicating the direction from the origin.

A *variable span*'s duration is dependent on an associated event. A common variable span is a *month*. The duration represented by a month depends on whether that month is associated with an event in June (30 days) or in July (31 days), or even in February (28 or 29 days). Variable spans use a specialized format requiring 64 bits.

To represent indeterminate events, we added nine formats, three of each resolution. There are three analogous formats for low resolution, and three for extended resolution. The most common, for high resolution with a uniform distribution, requires only 64 bits.

As we have seen in other areas, the considerations for space are similar, yet considerably simpler. A spatial representation of 32 bits (per dimension) for a range required to map the Earth results in a granularity of one decimeter, and of one centimeter for the third dimension (the atmosphere, the oceans, and the Earth's interior) or for restricted areas such as the United States or Europe. Moving up to 64 bits makes spatial indeterminacy representations feasible, and reduces the granularity to a nanometer, which should be adequate for quite a while.

3 Associating Facts with Time

The previous section explored models and representations for the time domain. We now turn to associating time with facts.

3.1 Underlying Data Model

Time has been added to many data models: the entity-relationship model @cite[DeAntonellis79,Klopprogge81], semantic data models @cite[Hammer81,Urban86], knowledge-based data models @cite[Dayal86A], deductive databases @cite[Chomicki88,Chomicki89A,Chomicki90,Kabanza90], and object-oriented models @cite[Dayal92,Manola86A,Narasimhalu88,Rose91,Sciore91,Sciore91A,Wuu92]. However, by far the majority of work in temporal databases is based on the relational model. For this reason, we will assume this data model in subsequent discussion.

3.2 Attribute Variability

There are several basic ways in which an attribute associated with an object can interact with time and space. A *time-invariant* attribute @cite[Navathe89] does not change over time. Some temporal data models require that the key of a relation be time-invariant; some others identify the object(s) participating in the relation with a time-invariant surrogate, a system-generated, unique identifier of an item that can be referenced and compared for equality, but not displayed to the user @cite[Hall76]. Secondly, the *value* of an attribute may be drawn from a temporal domain. An example is *date stamping*, where cadastral parcel records in a land information system contain fields that note the dates of registration of deeds, transfers of titles, and other pertinent historical information @cite[Vrana89]. Such temporal domains are termed *user-defined time* @cite[Snodgrass86A]; other than being able to be read in, displayed, and perhaps compared, no special semantics is associated with such domains. Interestingly, most such attributes are time-invariant. For example, the transfer date for a particular title transfer is valid over all time.

An analogous sitation exists for space. There are space-invariant attributes as well as attributes that are drawn from spatial domains, an example being an attribute recording the square feet of a residence, which is a relative spatial measure in two dimensions.

Time- and *space-varying* attributes are more interesting. There are five basic cases to consider.

- The value of an attribute associated with a *space-invariant* object may vary over time, termed *attribute temporality* @cite[Vrana89]. An example is percentage of cloud cover over the Earth, which has a single value at each point in time, but varies over time. The value is uniquely specified by the temporal coordinate(s) (either valid time, or a combination of valid and transaction time).
- The value of an attribute associated with a region in space may be time invariant. An example is elevation: the value varies spatially but not temporally (assuming historical time; certainly the elevation varies over geologic time!). The value is unique given spatial coordinates.
- The value of an attribute associated with a region in space may vary over time. An example is the percentage of cloud cover over each 10-kilometer square grid element. Here, the object is identified spatially, and each object is associated with a time-varying sequence of values. Both the temporal and the spatial coordinates are required to uniquely identify a value.
- The boundary lines identifying a cadastral object, e.g., a particular land packet, may vary over time, termed *temporal topology* @cite[Vrana89]. The orientation and interaction of spatial objects change over time; such objects could nevertheless have time-invariant attributes, such as initial purchase price. The temporal and spatial coordinates are required to uniquely identify a value, but this identification is indirect, via the topology.
- The final case is the most complex: the value of an attribute, which varies over time, is associated with a cartographic feature that also varies over time

@cite[Langran88]. An example is the appraised value of a land packet. The appraised value may change yearly, and the boundary of the land packet changes as it is reapportioned and parts sold to others. As with the previous two cases, both temporal and spatial coordinates are required to identify a value, but the temporal coordinate(s) are utilized twice, first to identify a cartographic feature and then to select a particular attribute value associated with that feature.

A further categorization is possible concerning which temporal domains are involved (only valid time, only transaction time, or both) and which spatial domains are involved (two dimensions, $2\frac{1}{2}$ dimensions, or a full three dimensions).

The first case is the traditional domain of temporal DBMS's. The second case is the domain of conventional LIS's and GIS's. While there has been some conceptual work on merging temporal and spatial support, as discussed throughout this paper, current implemented systems are fairly weak in this regard.

3.3 Representational Alternatives

Over two dozen extensions to the relational model to incorporate time have been proposed over the last 15 years. These models may be compared by asking four basic questions: how is valid time represented, how is transaction time represented, how are attribute values represented, and is the model *homogeneous*, i.e., are all attributes restricted to be defined over the same valid time(s) @cite[Gadia88B].

Data Models. Table 2 lists most of the temporal data models that have been proposed to date. If the model is not given a name, we appropriate the name given the associated query language, where available. Many models are described in several papers; the one referenced is the initial journal paper in which the model was defined. Some models are defined only over valid time or transaction time; others are defined over both. Whether the model is homogeneous is indicated in the next column. Tuple-timestamped data models, to be identified in the next section, and data models that use single chronons as time-stamps are of necessity homogeneous. The issue of homogeneity is not relevant for those data models supporting only transaction time. The last column indicates a short identifier which denotes the model; the table is sorted on this column.

We omit a few intermediate data models, specifically Gadia's multihomogeneous model @cite[Gadia86A], which was a precursor to his heterogeneous model (Gadia-2), and Gadia's two-dimensional temporal relational database model @cite[Bhargava89B], which is a precursor to Gadia-3. We also do not include the data model used as the basis for defining temporal relational completeness @cite[Tuzhilin90], because it is a generic data model that does not force decisions on most of the aspects to be discussed here.

More detail on these data models, including a comprehensive comparison, may be found elsewhere @cite[McKenzie91B,Snodgrass87A].

Data Model	*Citation*	*Temporal Dimension(s)*	*Homogeneous*	*Identifier*
—	@cite[Snodgrass86A]	both	yes	Ahn
Temporally Oriented Data Model	@cite[Ariav86A]	both	yes	Ariav
Time Relational Model	@cite[BenZvi82]	both	yes	Ben-Zvi
Historical Data Model	@cite[Clifford83]	valid	yes	Clifford-1
Historical Relational Data Model	@cite[Clifford87A]	valid	no	Clifford-2
Homogeneous Relational Model	@cite[Gadia88B]	valid	yes	Gadia-1
Heterogeneous Relational Model	@cite[Gadia88A]	valid	no	Gadia-2
TempSQL	@cite[Gadia92]	both	yes	Gadia-3
DM/T	@cite[Jensen91D]	transaction	N/A	Jensen
LEGOL 2.0	@cite[Jones79]	valid	yes	Jones
DATA	@cite[Kimball78]	transaction	N/A	Kimball
—	@cite[Lomet89A]	transaction	N/A	Lomet
Temporal Relational Model	@cite[Lorentzos88B]	valid	no	Lorentzos
—	@cite[Lum84]	transaction	yes	Lum
—	@cite[McKenzie91C]	both	no	McKenzie
Temporal Relational Model	@cite[Navathe89]	valid	yes	Navathe
HQL	@cite[Sadeghi87B]	valid	yes	Sadeghi
HSQL	@cite[Sarda90]	valid	yes	Sarda
Temporal Data Model	@cite[Segev87]	valid	yes	Shoshani
TQuel	@cite[Snodgrass87A]	both	yes	Snodgrass
Postgres	@cite[Stonebraker87D]	transaction	no	Stonebraker
HQuel	@cite[Tansel86B]	valid	no	Tansel
Accounting Data Model	@cite[Thompson91A]	both	yes	Thompson
Time Oriented Databank Model	@cite[Wiederhold75]	valid	yes	Wiederhold

Table 2. Temporal Data Models

Valid Time. Two fairly orthogonal aspects are involved in representing valid time. First, is valid time represented with single chronon identifiers (i.e., event time-stamps, Sec. 2.4), with intervals (i.e., as interval time-stamps, Sec. 2.4), or as historical elements (i.e., as a set of chronon identifiers, or equivalently as a finite set of intervals)? Second, is valid time associated with entire tuples or with individual attribute values? A third alternative, associating valid time with sets of tuples, i.e., relations, has not been incorporated into any of the proposed data models, primarily because it lends itself to high data redundancy. The data models are elevated on these two aspects in Table 3. Interestingly, only one quadrant, time-stamping tuples with an historical element, has not been considered (but see Sec. 3.3)

Transaction Time. The same general issues are involved in transaction time, but there are about twice as many alternatives. Transaction time may be associated with

	Single chronon	Interval (pair of chronons)	Historical element (set of chronons)
Time-stamped attribute values	Lorentzos Thompson	Gadia-2 McKenzie Tansel	Clifford-2 Gadia-1 Gadia-3
Time-stamped tuples	Ariav Clifford-1 Lum Sadeghi Shoshani Wiederhold	Ahn Ben-Zvi Jones Navathe Sarda Snodgrass	

Table 3. Representation of Valid Time

- a single chronon, which implies that tuples inserted on each transaction signify the termination (logical deletion) of previously current tuples with identical keys, with the time-stamps of these previously recorded tuples not requiring change.
- an interval. A newly inserted tuple would be associated with the interval starting at now and ending at the special value U.C., *until-changed*.
- three chronons. Ben-Zvi's model records (1) the transaction time when the valid start time was recorded, (2) the transaction time when the valid stop time was recorded, and (3) the transaction time when the tuple was logically deleted.
- a transaction-time element, which is a set of not-necessarily-contigous chronons.

Another issue concerns whether transaction time is associated with individual attribute values, with tuples, or with sets of tuples.

The choices made in the various data models are characterized in Table 4. Gadia-3 is the only data model to time-stamp attribute values; it is difficult to efficiently implement this alternative directly. Gadia-3 also is the only data model that uses transaction-time elements (but see Sec. 3.3). Ben-Zvi is the only one to use three transaction-time chronons. All of the rows and columns are represented by at least one data model.

Attribute Value Structure. The final major decision to be made in designing a temporal data model is how to represent attribute values. There are six basic alternatives. In some models, the time-stamp appears as an explicit attribute; we do not consider such attributes in this analysis.

- *Atomic valued*—values do not have any internal structure. Ariav, Ben-Zvi, Clifford-1, Jensen, Jones, Kimball, Lomet, Lorentzos, Lum, Navathe, Sadeghi, Sarda, Shoshani, Snodgrass, Stonebraker and Thompson all adopt this approach. Tansel allows atomic values, as well as others, listed below.

	Single chronon	Interval (pair of chronons)	Three Chronons	Transaction-time element (set of chronons)
Time-stamped attribute values				Gadia-3
Time-stamped tuples	Ariav Jensen Kimball Lomet	Snodgrass Stonebraker	Ben-Zvi	
Time-stamped sets of tuples	Ahn Thompson	McKenzie		

Table 4. Representation of Transaction Time

- *Set valued*—values are sets of atomic values. Tansel supports this representation.
- *Functional, atomic valued*—values are functions from the (generally valid) time domain to the attribute domain. Clifford-2, Gadia-1, Gadia-2, and Gadia-3 adopt this approach.
- *Ordered pairs*—values are an ordered pair of a value and a (historical element) time-stamp. McKenzie adopts this approach.
- *Triplet valued*—values are a triple of attribute value, valid from time, and value to time. This is similar to the ordered pairs representation, except that only one interval may be represented. Tansel supports this representation.
- *Set-triplet valued*—values are a set of triplets. This is more general than ordered pairs, in that more than one value can be represented, and more general than functional valued, since more than one attribute value can exist at a single valid time @cite[Tansel86B]. Tansel supports this representation.

In the conventional relational model, if attributes are atomic-valued, they are considered to be in *first normal form* @cite[Codd72B]. Hence, only the data models placed in the first category may be considered to be strictly in first normal form. However, in several of the other models, the non-atomicity of attribute values comes about because time is added. It turns out that the property of "all snapshots are in first normal form" is closely associated with homogeneity (Sec. 3.3).

Separating Semantics from Representation. It is our contention that focusing on data *presentation* (how temporal data is displayed to the user), on data *storage*, with its requisite demands of regular structure, and on efficient *query evaluation* has complicated the central task of capturing the time-varying semantics of data. The result has been, as we have seen, a plethora of incompatible data models, with many query languages (Sec. 4.1), and a corresponding surfeit of database design and implementation strategies that may be employed across these models.

We advocate instead a very simple *conceptual temporal data model* that captures the essential semantics of time-varying relations, but has no illusions of being suitable for presentation, storage, or query evaluation. Existing data model(s) may be used for these latter tasks. This conceptual model time-stamps tuples with bitemporal elements, sets of *bitemporal chronons*, which are rectangles in the two-dimensional space spanned by valid time and transaction time (see Fig. 2). Because no two tuples with mutually identical explicit attribute values (termed *value-equivalent* tuples) are allowed in a bitemporal relation instance, the full time history of a fact is contained in a single tuple.

In Table 3, the conceptual temporal data model occupies the unfilled entry corresponding to time-stamping tuples with historical elements, and occupies the entry in Table 4 corresponding to time-stamping tuples with transaction-time elements. An important property of the conceptual model, shared with the conventional relational model but not held by the representational models, is that relation instances are semantically unique; distinct instances model different realities and thus have distinct semantics.

It is possible to demonstrate equivalence mappings between the conceptual model and several *representational* models @cite[Jensen92A]. Mappings have already been demonstrated for three data models: Gadia-3 (attribute time-stamping), Jensen (tuple time-stamping with a single transaction chronon), and Snodgrass (tuple time-stamping, with interval valid and transaction times). This equivalence is based on *snapshot equivalence*, which says that two relation instances are equivalent if all their snapshots, taken at all times (valid and transaction), are identical. Snapshot equivalence provides a natural means of comparing rather disparate representations. An extension to the conventional relational algebraic operators may be defined in the conceptual data model, and can be mapped to analogous operators in the representational models. Finally, we feel that the conceptual data model is the appropriate location for database design and query optimization.

In essence, we advocate moving the distinction between the various existing temporal data models from a semantic basis to a physical, performance-relevant basis, utilizing the proposed conceptual data model to capture the time-varying semantics. Data presentation, storage representation, and time-varying semantics should be considered in isolation, utilizing different data models. Semantics, specifically as determined by logical database design, should be expressed in the conceptual model. Multiple presentation formats should be available, as different applications require different ways of viewing the data. The storage and processing of bitemporal relations should be done in a data model that emphasizes efficiency.

4 Querying

A data model consists of a set of objects with some structure, a set of constraints on those objects, and a set of operations on those objects @cite[Tsichritzis82]. In the two previous sectins we have investigated in detail the structure of and con-

straints on the objects of temporal relational databases, the temporal relation. In this section, we complete the picture by discussing the operations, specifically temporal query languages.

Many temporal query languages have been proposed. In fact, it seems obligatory for each researcher to define their own data model and query language (we return to this issue at the end of this section). We first summarize the twenty-odd query languages that have been proposed thus far. We then briefly discuss the various activities that should be supported by a temporal query language, using a specific language in the examples. Finally, we touch on work being done in the standards arena that is attempting to bring highly needed order to this confusing collection of languages.

We do not consider the related topic of *temporal reasoning* (also termed *inferencing* or *rule-based search*) @cite[Chomicki90,Kahn77A,Karlsson86A,Lee85,Sheng84,Sripada88] that uses artificial intelligence techniques to perform more sophisticated analyses of temporal relationships, generally with much lower query processing efficiency.

4.1 Language Proposals

Table 5 lists the major temporal query language proposals to date. While many of these languages each have several associated papers, we have indicated the most comprehensive or most readily available reference. The underlying data model is a reference to Table 2. The next column lists the conventional query language the temporal proposal is based on, from the following.

SQL Structured Query Language @cite[Date89B], a tuple calculus-based language; the lingua franca of conventional relational databases.

Quel The tuple calculus based query language @cite[Held75] originally defined for the Ingres relational DBMS @cite[Stonebraker76A].

QBE Query-by-Example @cite[Zloof75], a domain calculus based query language.

IL_s An intensional logic formulated in the context of computational linguistics @cite[Montague73].

relational algebra A procedural language with relations as objects @cite[Codd72].

DEAL An extension of the relational algebra incorporating functions, recursion, and deduction @cite[Deen85].

Most of the query languages have a formal definition. Some of the calculus-based query languages have an associated algebra that provides a means of evaluating queries.

More comprehensive comparisons may be found elsewhere @cite[McKenzie91B,Snodgrass87A].

4.2 Types of Temporal Queries

We now examine the types of temporal queries that each of the above-listed query languages support to varying degrees. We'll use TQuel @cite[Snodgrass93A] in the examples, as it is the most completely defined temporal language @cite[Snodgrass87A].

Name	*Citation*	*Underlying Data Model*	*Based On*	*Formal Semantics*	*Equivalent Algebra*
HQL	@cite[Sadeghi87A]	Sadeghi	DEAL	partial	@cite[Sadeghi87B]
HQuel	@cite[Tansel86B]	Tansel	Quel	yes	@cite[Tansel86B]
HSQL	@cite[Sarda90]	Sarda	SQL	no	@cite[Sarda90B]
HTQuel	@cite[Gadia88B]	Gadia-1	Quel	yes	@cite[Gadia88B]
Legol 2.0	@cite[Jones79]	Jones	relational algebra	no	N/A
Postquel	@cite[Stonebraker90B]	Stonebraker	Quel	no	none
TDM	@cite[Segev87]	Shoshani	SQL	no	none
Temporal Relational Algebra	@cite[Lorentzos88B]	Lorentzos	relational algebra	yes	N/A
TempSQL	@cite[Gadia92]	Gadia-3	SQL	partial	none
Time-By-Example	@cite[Tansel89]	Tansel	QBE	yes	@cite[Tansel86B]
TOSQL	@cite[Ariav86A]	Ariav	SQL	no	none
TQuel	@cite[Snodgrass87A]	Snodgrass	Quel	yes	@cite[McKenzie91C]
TSQL	@cite[Navathe89]	Navathe	SQL	no	none
—	@cite[BenZvi82]	Ben-Zvi	SQL	yes	@cite[BenZvi82]
—	@cite[Clifford83]	Clifford-1	IL_s	yes	N/A
—	@cite[Clifford87A]	Clifford-2	relational algebra	yes	N/A
—	@cite[Gadia86A]	Gadia-2	Quel	no	none
—	@cite[Jensen91A]	Jensen	relational algebra	yes	N/A
—	@cite[McKenzie91C]	McKenzie	relational algebra	yes	N/A
—	@cite[Thompson91A]	Thompson	relational algebra	yes	N/A
—	@cite[Tuzhilin90]	*several*	relational algebra	yes	N/A

Table 5. Temporal query languages

Schema Definition. We will use one relation in these examples.

Example. *Define the* `Cities` *relation.*

```
create persistent interval Cities(Name is char, State is char,
          Population is I4, IncorporationDate is event,
          Size is area)
```

`Cities` has five explicit attributes: two strings (denoted by `char`), a 4-byte integer (denoted by `I4`), a user-defined event, and a user-defined area. The ***persistent*** and ***interval*** keywords specify a bitemporal relation, with four implicit time-stamp attributes: a valid start time, a valid end time, a transaction start time, and a transaction end time. The valid time-stamps define the interval when

the attribute values were true in reality, and the transaction time-stamps specify the interval when the information was current in the database. □

Quel Retrieval Statements. Since TQuel is a strict superset of Quel, all Quel queries are also TQuel queries @cite[Snodgrass87A]. Here we give one such query, as a review of Quel.

The query uses a ***range*** statement to specify the *tuple variable* `C`, which will remain active for use in subsequent queries.

Example. *What is the current population of the cities in Arizona?*

```
range of C is Cities

retrieve (C.Name, C.Population)
where C.State = "Arizona"
```

The *target list* specifies which attributes of the qualifying tuples are to be retained in the result, and the where clause specifies which underlying tuples from the underlying relation(s) qualify to participate in the query. Because the defaults have been defined appropriately, each TQuel query yields the same result as its Quel counterpart. □

Rollback (Transaction-time Slice). The ***as of*** clause rolls back a transaction-time relation (consisting of a sequence of snapshot relation states) or a bitemporal relation (consisting of a sequence of valid-time relation states) to the state that was current at the specified transaction time. It can be considered to be a transaction time analogue of the where clause, restricting the underlying tuples that participate in the query.

Example. *What was the population of Arizona's cities as best known in 1980?*

```
retrieve (C.Name, C.Population)
where C.State = "Arizona"
as of begin of |January 1, 1980|
```

This query uses an event temporal constant, delimited with vertical bars, "$|\cdots|$". TQuel supports multiple calendars and calendric systems @cite[Soo92, Soo92A, Soo92C]. In this case, the default is the Gregorian calendar with English month names. □

Valid-time Selection. The ***when*** clause is the valid-time analogue of the where clause: it specifies a predicate on the event or interval time-stamps of the underlying tuples that must be satisfied for those tuples to participate in the remainder of the processing of the query.

Example. *What was the population of the cities in Arizona in 1980 (as best known right now)?*

```
retrieve (C.Name, C.Population)
```

```
where C.State = "Arizona"
when C overlap |January 1, 1980|
as of present
```

A careful examination of the prose statement of this and the previous query illustrates the fundamental difference between valid time and transaction time. The *as of* clause selects a particular transaction time, and thus *rolls back* the relation to its state stored at the specified time. Corrections stored after that time will not be incorporated into the retrieved result. The particular *when* statement given here selects the facts *valid in reality* at the specified time. All corrections stored up to the time the query was issued are incorporated into the result. In this case, all corrections made after 1980 to the census of 1980 will be included in the resulting relation. □

Example. *What was the population of the cities in Arizona in 1980, as best known at that time?*

```
retrieve (C.Name, C.Population)
where C.State = "Arizona"
when C overlap |January 1, 1980|
as of |January 1, 1980|
```

The result of this query, executed any time after January 1, 1980, will be identical to the result of the first query specified, "*What is the current population of the cities in Arizona?*", executed exactly on midnight of that date. □

Valid Time Projection. The *valid* clause serves the same purpose as the target list; it specifies some aspect of the derived tuples, in this case, the valid time of the derived tuple.

Example. *For what date is the most recent information on Arizona's cities valid?*

```
retrieve (C.All)
valid at begin of C
where C.State = "Arizona"
```

This query extracts relevant events from an interval relation. □

Aggregates. As TQuel is a superset of Quel, all Quel aggregates are still available @cite[Snodgrass92A].

Example. *What is the current population of Arizona?*

```
retrieve (sum(C.Population where C.State = "Arizona"))
```

Note that this query only counts city residents. □

This query applied to a bitemporal relation yields the same result as its conventional analogues, that is, a single value. With just a little more work, we can extract its time-varying behavior.

Example. *How has the population of Arizona flucuated over time?*

```
retrieve (sum(C.Population where C.State = "Arizona"))
when true
```
□

New, temporally-oriented aggregates are also available in TQuel. One of the most useful computes the interval when the argument was rising in value. This aggregate may be used wherever an interval expression is expected.

Example. *For each growing city, when did it start growing?*

```
retrieve (C.Name)
valid at begin of rising(C.Population by C.Name
                         where C.State = "Arizona")
```
□

Historical Indeterminacy. Indeterminacy aspects can hold for individual tuples, or for all the tuples in a relation.

Example. *The information in the* `Cities` *relation is known only to within thirty days.*

```
modify cities to indeterminate span = %30 days%
```

`%30 days%` is a *span*, an unanchored length of time @cite[Soo92C]. Spans can be created by taking the difference of two events; spans can also be added to an event to obtain a new event. □

Example. *What cities in Arizona definitely had a population over 500,000 at the beginning of 1980?*

```
retrieve (C.Name)
where C.State = "Arizona" and C.Population > 500000
when C overlap |January 1, 1980|
```

The default is to only retrieve tuples that fully satisfy the predicate. This is consistent with the Quel semantics. □

Historical indeterminacy enters queries at two places, specifying the *credibility* of the underlying information to be utilized in the query, and specifying the *plausibility* of temporal relationships expressed in the when and valid clauses. We'll only illustrate plausibility here.

Example. *What cities in Arizona had a population over 500,000 probably at the beginning of 1980?*

```
retrieve (C.Name)
where C.State = "Arizona" and C.Population > 500000
when C overlap |January 1, 1980| probably
```

Here, "`probably`" is syntactic sugar for "`with plausibility 70`". □

Schema Evolution. Often the database schema needs to be modified to accommodate a changing set of applications. The ***modify*** statement has several variants, allowing any previous decision to be later changed or undone. Schema evolution involves transaction time, as it concerns how the data is stored in the database @cite[McKenzie90A]. As an example, changing the type of a relation from a bitemporal relation to an historical relation will cause future intermediate states to not be recorded; states stored when the relation was a temporal relation will still be available.

Example. *The* `Cities` *relation should no longer record all errors.*

```
modify Stocks to not persistent
```
□

4.3 Standards

Support for time in conventional data base systems (e.g., @cite[ENFORM83, ORACLE87]) is entirely at the level of user-defined time (i.e., attribute values drawn from a temporal domain). These implementations are limited in scope and are, in general, unsystematic in their design @cite[DATE88, DATE90B]. The standards bodies (e.g., ANSI) are somewhat behind the curve, in that SQL includes no time support. Date and time support very similar to that in DB2 is currently being proposed for SQL2 @cite[MELTON90]. SQL2 corrects some of the inconsistencies in the time support provided by DB2 but inherits its basic design limitations @cite[Soo92C].

An effort is currently underway within the research community to consolidate approaches to temporal data models and calculus-based query languages, to achieve a consensus extension to SQL and an associated data model upon which future research can be based. This extension is termed the *Temporal Structured Query Language*, or TSQL (not to be confused with an existing language proposal of the same name).

5 System Architecture

The three previous sections in concert sketched the boundaries of a temporal data model, by examining the temporal domain, how facts may be associated with time, and how temporal information may be queried. We now turn to the implementation of the temporal data model, as encapsulated in a temporal DBMS.

Adding temporal support to a DBMS impacts virtually all of its components. Figure 4 provides a simplified architecture for a conventional DBMS. The *database administrator (DBA)* and her staff design the database, producing a physical schema specified in a *data definition language (DDL)*, which is processed by the *DDL Compiler* and stored, generally as system relations, in the *System Catalog*. Users prepare queries, either ad hoc or embedded in procedural code, which is submitted to the *Query Processor*. The query is first lexically and syntactically analyzed, using information from the system catalog, then optimized

for efficient execution. A query evaluation plan is sent to the *Query Evaluator*. For ad hoc queries, this occurs immediately after processing; for embedded queries, this occurs when the cursor associated with a particular query is opened. The query evaluator is usually an interpreter for a form of the relational algebra annotated with access methods and operator strategies. While evaluating the query, this component accesses the database via a *Stored Data Manager*, which implements concurrency control, transaction management, recovery, buffering, and the available data access methods.

Fig. 4. Components of a data base management system

In the following, we visit each of these components in turn, reviewing what changes need to be made to add temporal support.

5.1 DDL Statements

Relational query languages such as Quel and SQL actually do much more than simply specify queries; they also serve as data definition languages (e.g., through Quel's `create` statement, c.f., Sec. 4.2) and as data manipulation languages (e.g., through SQL's `INSERT`, `DELETE` and `UPDATE` statements). The changes to

support time involve adding temporal domains, such as event, interval, and span @cite[Soo92C] and adding constructs to specify support for transaction and valid time, such as the TQuel keywords ***persistent*** and ***interval***.

5.2 System Catalog

The big change here is that the system catalog must consist of transaction-time relations. Schema evolution concerns only the recording of the data, and hence does not involve valid time. The attributes and their domains, the indexes, even the names of the relations all vary over transaction time.

5.3 Query Processing

There are two aspects here, one easily extended (language analysis) and one for which adding temporal support is much more complex (query optimization).

Language analysis needs to consider multiple calendars, which extend the language with calendar-specific functions. An example is `monthof`, which only makes sense in calendars for which there are months. The changes to language processing, primarily involving modifications to semantic analysis (name resolution and type checking), have been worked out in some detail @cite[Soo92A].

Optimization of temporal queries is substantially more involved than that for conventional queries, for several reasons. First, optimization of temporal queries is more critical, and thus easier to justify expending effort on, than conventional optimization. The relations that temporal queries are defined over are larger, and are growing monotonically, with the result that unoptimized queries take longer and longer to execute. This justifies trying harder to optimize the queries, and spending more execution time to perform the optimization.

Second, the predicates used in temporal queries are harder to optimize @cite[Leung90,Leung91A]. In traditional database applications, predicates are usually equality predicates (hence the prevalence of equi-joins and natural joins); if a less-than join is involved, it is rarely in combination with other less-than predicates. On the other hand, in temporal queries, less-than joins appear more frequently, as a conjunction of several inequality predicates. As an example, the TQuel ***overlap*** operator is translated into two less-than predicates on the underlying time-stamps. Optimization techniques in conventional databases focus on equality predicates, and often implement inequality joins as cartesian products, with their associated inefficiency.

And third, there is greater opportunity for query optimization when time is present @cite[Leung91A]. Time advances in one direction: the time domain is continuous expanding, and the most recent time point is the largest value in the domain. This implies that a natural clustering or sort order will manifest itself, which can be exploited during query optimization and evaluation. The integrity constraint $beginof(t) < endof(t)$ holds for every time-interval tuple t. Also, for many relations it is the case that the intervals associated with a key are contiguous in time, with one interval starting exactly when the previous interval ended. An example is salary data, where the intervals associated with

the salaries for each employee are contiguous. *Semantic query optimization* can exploit these integrity constraints, as well as additional ones that can be inferred @cite[Shenoy89].

In this section, we first examine local query optimization, of a single query, then consider global query optimization, of several queries simultaneously. Both involve the generation of a *query evaluation plan*, which consists of an algebraic expression annotated with access methods.

Local Query Optimization. A single query can be optimizing by replacing the algebraic expression with an equivalent one that is more efficient, by changing an access method associated with a particular operator, or by adopting a particular implementation of an operator. The first alternative requires a definition of equivalence, in the form of a set of tautologies. Tautologies have been identified for the conventional relational algebra @cite[Enderton77,Smith75B,Ullman88B], as well as for many of the algebras listed in Table 5. Some of these temporal algebras support the standard tautologies, enabling existing query optimizers to be used.

To determine which access method is best for each algebraic operator, *meta-data*, that is, statistics on the stored temporal data, and *cost models*, that is, predictors of the execution cost for each operator implementation/access method combination, are needed. Temporal data requires additional meta-data, such as *lifespan* of a relation (the time interval over which the relation is defined), the lifespans of the tuples, the surrogate and tuple arrival distributions, the distributions of the time-varying attributes, regularity and granularity of temporal data, and the frequency of null values, which are sometimes introduced when attributes within a tuple aren't synchronized @cite[Gunadhi89A]. Such statistical data may be updated by random sampling or by a scan through the entire relation.

There has been some work in developing cost models for temporal operators. An extensive analytical model has been developed and validated for TQuel queries @cite[Ahn88B,Ahn89], and *selectivity estimates* on the size of the results of various temporal joins have been derived @cite[Gunadhi89D,Gunadhi89A].

Global Query Optimization. In global query optimization, a collection of queries is simultaneously optimized, the goal being to produce a single query evaluation plan that is more efficient than the collection of individual plans @cite[Satoh85A,Sellis86A]. A state transition network appears to be the best way to organize this complex task @cite[Jensen92G]. *Materialized views* @cite[Blakeley86B,Blakeley90,Roussopo
are expected to play an important role in achieving high performance in the face of temporal databases of monotonically increasing size. For an algebra to utilize this approach, incremental forms of the operators are required (c.f., @cite[McKenzie88,Jensen91D]).

5.4 Query Evaluation

Achieving adequate efficiency in query evaluation is very important. We first examine operations on time-stamps, some of which are critical to high performance. We then review a study that showed that a straightforward implementation would not result in reasonable performance. Since joins are the most expensive, yet very common, operations, they have been the focus of a significant amount of research. Finally, we will examine the many temporal indexes that have been proposed.

Domain Operations. In Sec. 2 we outlined the domain of time-stamps. Query evaluation performs input, comparison, arithmetic, and output operations on values of this domain. Ordered by contribution to execution efficiency, they are comparison (which is often in the "inner loop" of join processing), arithmetic (which is most often performed during creation of the resulting tuple), output (which is only done when transferring results to the screen or to paper, a much slower process than execution or even disk I/O), and finally input (which is done exactly once per value). However, the SQL2 format, with its five components (see Table 1) is optimized for the relatively infrequent operations of (Gregorian) input and output, and is rather slow at comparison and addition. The proposed formats instead optimize comparison at the expense of input and output. For a sequence of operations that inputs two relations and computes and outputs the overlap (favoring input and output more than expected), the high resolution format is more efficient, with only 50 tuples, than the SQL2 format, even though the high resolution format has much greater range and smaller granularity @cite[Dyreson92C].

Performing these operations efficiently in the presence of historical indeterminacy is more challenging. For the default range credibility and ordering plausibility, and for comparing events whose sets of possible chronons do not overlap, there is little overhead even when historical indeterminacy is supported @cite[Dyreson92D]. The average *worse case* for comparison, over all plausibilities, when the sets of possible chronons overlap signficantly, is less than 100 microseconds on a Sun-4, or about the time to transfer a 100-byte tuple from disk.

A Straightforward Implementation. The importance of efficient query optimization and evaluation for temporal databases was underscored by an initial study that analyzed the performance of a brute-force approach to adding time support to a conventional DBMS. In this study, the university Ingres DBMS was extended in a minimal fashion to support TQuel querying @cite[Ahn86B]. The results were very discouraging for those who might have been considering such an approach. Sequential scans, as well as access methods such as hashing and ISAM, suffered from rapid performance degradation due to ever-growing overflow chains. Because adding time creates multiple tuple versions with the same key, reorganization does not help to shorten overflow chains. The objective

of work in temporal query evaluation then is to avoid looking at all of the data, because the alternative implies that queries will continue to slow down as the database accumulates facts.

There were four basic responses to this challenge. The first was a proposal to separate the *historical* data, which grew monotonically, from the *current* data, whose size was fairly stable and whose accesses were more frequent @cite[Lum84]. This separation, termed *temporal partitioning*, was shown to significantly improve performance of some queries @cite[Ahn88B], and was later generalized to allow multiple cached states, which further improves performance @cite[Jensen92G]. Second, new query optimization strategies were proposed (Sec. 5.3). Third, new join algorithms, to be discussed next, were proposed. And finally, new temporal indexes, also to be discussed, were proposed.

Joins. Three kinds of temporal joins have been studied: binary joins, multiway joins, and joins executed on multiprocessors.

A wide variety of binary joins have been considered, including *time-join*, *time-equijoin* (TE-join) @cite[Clifford87A], *event-join*, *TE-outerjoin* @cite[Gunadhi91], *contain-join*, *contain-semijoin*, *intersect-join* @cite[Leung91A], and *contain-semijoin* @cite[Leung92]. The various algorithms proposed for these joins have generally been extensions to nested loop or merge joins that exploit sort orders or local workspace.

Leung argues that a checkpoint index (Sec. 5.4) is useful when stream processing is employed to evaluate both two-way and multi-way joins @cite[Leung92].

Finally, Leung has explored in depth partitioning strategies and temporal query processing on multiprocessors @cite[Leung91].

Temporal Indexes. Conventional indexes have long been proposed to reduce the need to scan an entire relation to access a subset of its tuples. Indices are even more important in temporal relations that grow monotonically in size. In table 6 we summarize the temporal index structures that have been proposed to date. Most of the indexes are based on B^+-Trees @cite[Comer79], which index on values of a single key; the remainder are based on R-Trees @cite[Guttman84], which index on ranges (intervals) of multiple keys. There has been considerable discussion concerning the applicability of point-based schemes for indexing interval data. Some argue that structures that explicitly accommodate intervals, such as R-Trees and their variants, are preferable; others argue that mapping intervals to their endpoints is efficient for spatial search @cite[Lomet91].

If the structure requires that exactly one record with each key value exist at any time, or if the data records themselves are stored in the index, then it is designated a *primary* storage structure; otherwise, it can be used either as a primary storage structure or as a secondary index. The checkpoint index is associated with a particular indexing condition, making it suitable for use during the processing of queries consistent with that condition.

A majority of the indexes are tailored to transaction time, exploiting the append-only nature of such information. Most utilize as a key the valid-time or

transaction-time interval (or possibly both, in the case of the Mixed Media R-Tree). Lum's index doesn't include time at all; rather it is a means of accessing the history, represented as a linked list of tuples, of a key value. The Append-only Tree indexes the transaction-start time of the data, and the Lop-Sized B^+-Tree is most suited for indexing events such as bank transactions. About half the indexes utilize only the time-stamp as a key; some include a single non-temporal attribute; and the two based on R-Trees can exploit its multi-dimensionality to support an arbitrary number of non-temporal attributes. Of the indexes supporting non-temporal keys, most treat such keys as a true separate dimension, the exceptions being the indexes discussed by Ahn, which support a single composite key with the interval as a component.

While preliminary performance studies have been carried out for each of these indexes in isolation, there has been little effort to compare them concerning their space and time efficiency. Such a comparison would have to consider the differing abilities of each (those supporting no non-temporal keys would be useful for doing temporal cartesian products, but perhaps less useful for doing temporal joins that involved equality predicates on non-temporal attributes) as well as various underlying distributions of time and non-temporal keys (the indexes presume various non-uniform distributions to achieve their performance gains over conventional indexes, which generally assume a uniform key distribution).

5.5 Stored Data Manager

We examine three topics, storage structures (including page layout), concurrency control, and recovery. Page layout for temporal relations is more complicated than conventional relations if non-first normal form (i.e., non-atomic attribute values) are adopted, as is proposed in many of the temporal data models listed in Sec. 3.3. Often such attributes are stored as linked lists, for example representing a valid-time element (set of valid-time chronons) as a linked list of intervals. Hsu has developed an analytical model to determine the optimal block size for such linked lists @cite[Hsu91A].

Many structures have been proposed, including *reverse chaining* (all history versions for a key are linked in reverse order) @cite[BenZvi82,Dadam84,Lum84], *accession lists* (a block of time values and associated tuple id's between the current store and the history store), *clustering* (storing history versions together on a set of blocks), *stacking* (storing a fixed number of history versions), and *cellular chaining* (linking blocks of clustered history versions), with analytical performance modeling @cite[Ahn86C] being used to compare their space and time efficiency @cite[Ahn88B].

Several researchers have investigated adapting existing concurrency control and transaction management techniques to support transaction time. The subtle issues involved in choosing whether to time-stamp at the beginning of a transaction (which restricts the concurrency control method that can be used) or at the end of the transaction (which may require data earlier written by the transaction to be read again to record the transaction) have been resolved in favor of the latter through some implementation tricks @cite[Dadam84,Stonebraker87D,Lomet90A].

Name	*Citation*	*Based On*	*Primary/ Secondary*	*Temporal Dimension(s)*	*Temporal Key(s)*	*Non-Temporal Key(s)*
Append-only Tree	@cite[Gunadhi91A]	B^+-Tree	primary	transaction	event	0
Checkpoint Index	@cite[Leung92]	B^+-Tree	secondary	transaction	event	0
Lop-Sided B^+-Tree	@cite[Kolovson90B]	B^+-Tree	both	transaction	event	0
Monotonic B^+-Tree	@cite[Elmasri92]	Time Index	both	transaction	interval	0
—	@cite[Lum84]	B^+-Tree or Hashing	primary	transaction	none	1
Time-Split B-Tree	@cite[Lomet90]	B^+-Tree	primary	transaction	interval	1
Mixed Media R-Tree	@cite[Kolovson89]	R-Tree	both	transaction, trans+valid	interval, pairs of intervals	k ranges, $k \geq 1$
Time Index	@cite[Elmasri90]	B^+-Tree	both	both	interval	0
Two-level Combined Attribute/Time Index	@cite[Elmasri91]	B^+-Tree +Time Index	both	both	interval	1
—	@cite[Ahn88B]	B^+-Tree, Hashing	*various*	*various*	interval	1
SR-Tree	@cite[Kolovson90]	Segment Index + R-Tree	both	both	interval, pairs of intervals	k ranges, $k \geq 1$

Table 6. Temporal Indexes

The Postgres system is an impressive prototype DBMS that supports transaction time @cite[Stonebraker90B]. Time-stamping in a distributed setting has also been considered @cite[Lomet90A]. Integrating temporal indexes with concurrency control to increase the available concurrency has been studied @cite[Lomet91B].

Finally, since a transaction-time database contains all past versions of the database, it can be used to recover from media failures that cause a portion or all of the current version to be lost @cite[Lomet91].

6 Conclusion

We conclude with a list of accomplishments, a list of disappointments, and a pointer to future work.

There have been many significant accomplishments over the past fifteen years of temporal database research.

- The semantics of the time domain, including its structure, dimensionality, and indeterminacy, is well-understood.
- Representational issues of time-stamps have recently been resolved.
- Operations on time-stamps are now well-understood, and efficient implementations exist.
- A great amount of research has been expended on temporal data models, addressing this extraordinarily complex and subtle design problem.
- Many temporal query languages have been proposed. The numerous types of temporal queries are fairly well-understood. Half of the proposed temporal query languages have a strong formal basis.
- Temporal joins are well-understood, and a multitude of implementations exist.
- Approximately a dozen temporal index structures have been proposed.
- The interaction between transaction time support and concurrency control and transaction management has been studied to some depth.
- Several prototype temporal DBMS implementations have been developed.

There have also been some disappointments.

- The user-defined time support in the SQL2 standard is poorly designed. The representation specified in that standard suffers from inadequate range, excessive space requirements, and inefficient operations.
- There has been almost no work done in comparing the two dozen temporal data models, to identify common features and define a consensus data model upon which future research and commercialization may be based. It is our feeling that expecting a data model to simultaneously express time-varying semantics while optimizing data presentation, data storage, and query evaluation is unrealistic. We advocate a two-tiered data model, with a conceptual data model expressing the semantics, and with several representational data models serving these other objectives.
- There is also a need to consolidate approaches to temporal query languages, identify the best features of each of the proposed languages, and incorporate these features into a consensus query language that could serve as the basis for future research into query optimization and evaluation. Also, more work is needed on adding time to so-called fourth generation languages that are revolutionizing user interfaces for commercially available DBMS's.
- It has been demonstrated that a straightforward implementation of a temporal DBMS will exhibit poor performance.
- More empirical studies are needed to compare join algorithms, and to possibly suggest even more efficient variants.
- While there are a host of individual approaches to isolated portions of a DBMS, no coherent architecture has arisen. While the analysis given in Sec. 5 may be viewed as a starting point, much more work is needed to integrate these approaches into a cohesive structure.
- There has been little effort to compare the relative performance of temporal indexes, making selection in specific situations difficult or impossible.

- Temporal database design is still in its infancy, hindered by the plethora of temporal data models.
- There are as yet no prominent commercial temporal DBMS's, despite the obvious need in the marketplace.

Obviously these disappointments should be addressed. In addition, future work is also needed on adding time to the newer data models that are gaining recognition, including object-oriented data models and deductive data models. Finally, there is a great need for integration of spatial and temporal data models, query languages, and implementation techniques.

7 Acknowledgements

This work was supported in part by NSF grant ISI-8902707. James Clifford was helpful in understanding structural aspects of models of time. Curtis Dyreson, Christian S. Jensen, Nick Kline and Michael Soo provided useful comments on a previous draft.

8 Bibliography

@bib

Table of Contents

This article was processed using the LaTeX macro package with LLNCS style