

TEMPORAL DATA MANAGEMENT *

Arie Shoshani
Kyoji Kawagoe †
Computer Science Research Department
Lawrence Berkeley Laboratory
Berkeley, CA 94720

Abstract

In this paper we develop a framework for the support of temporal data. The concept of a time sequence is introduced, and shown to be an important fundamental concept for representing the semantics of temporal data and for efficient physical organization. We discuss properties of time sequences that allow the treatment of such sequences in a uniform fashion. These properties are exploited in order to design efficient physical data structures and access methods for time sequences. We also describe operations over time sequences, and show their power to manipulate temporal data.

1 Introduction

Recently there has been a surge of interest in temporal data, perhaps because memory and magnetic disk storage costs are rapidly decreasing, and the advances in optical disk technology. In the past, temporal data was mostly delegated to archival storage or discarded altogether because it was too expensive or impractical to access them on-line. While it was recognized that historical data are of great importance

*An expanded version of this paper was issued as LBL technical report LBL-21143. This work was supported by the Applied Mathematical Sciences Research Program of the Office of Energy Research, U.S. Department of Energy, under contract number DE-AC03-76SF00098. Contract DE-AC03-76SF00098.

†On leave from NEC, Japan. Current address: NEC Corporation, Kawasaki City, Kanagawa 213, Japan.

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the VLDB copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Very Large Data Base Endowment. To copy otherwise, or to republish, requires a fee and/or special permission from the Endowment.

to applications such as data analysis for policy decisions, such applications were not viewed as essential for a day-to-day operation. As a result, existing data management systems are designed to support the view of the most current version of the database. The dominant approach is one of data being updated, deleted, and inserted in order to maintain the current version.

In reality, many applications need to maintain a complete record of operations over the database. This is quite obvious in most business applications, such as banking, sales, inventory control, and reservation systems, where the history of all transactions has to be recorded. Furthermore, if this history can be managed as an integral part of the database, it could also be statistically analyzed for decision making purposes. In addition, there are applications that are inherently time dependent. Our interest stems from such applications in scientific and statistical databases (SSDBs), where physical experiments, measurements, simulations, and collected statistics are usually in the time domain. In such applications temporal data are essential, and in many cases the concept of a "current version" does not even make sense. Other applications where the time domain is inherent include engineering databases, econometrics, surveys, policy analysis, music, etc.

There is an extensive literature related to the concept of time, and the support of temporal data. There are several surveys, such as [Bolour et al 82] and [Snodgrass & Ahn 85], that summarize the relevant literature. Most of the work to date concentrates on the semantics of time, and on logical modeling and query languages for temporal data. Little work has been done in the physical organization area. In this paper, we address both the logical and physical aspects of temporal data management.

We refer to a collection of data values over time as a "time sequence". Our purpose here is to develop a framework for the support of time sequences, rather than to define precisely the details of the design. Thus, many concepts are only explained by us-

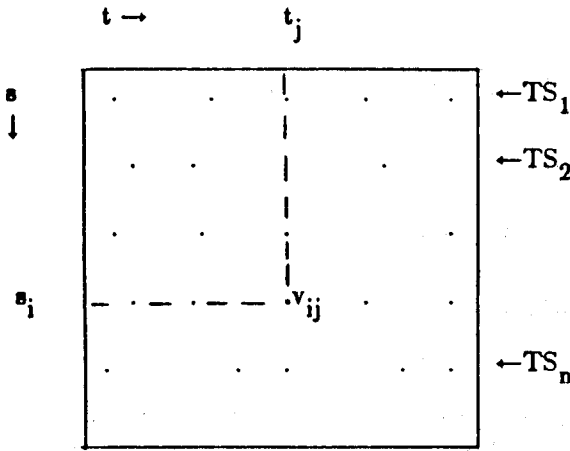


Figure 1: A Time Sequence Array

ing examples.

In section 2, we define time sequences and illustrate with several examples that they are a natural concept for viewing temporal data. In section 3, the different properties of time sequences are described. These properties are important for operations over time sequences, and for their physical implementation. In section 4, we describe operations over time sequences. In section 5 we discuss methods for the physical organization of time sequences. Section 6 contains a summary and areas of future research.

2 Time Sequences

It has long been observed that a temporal value is actually a triplet (s, t, v) where $s, t,$ and v stand for surrogate, time, and value respectively. Thus, the triplet (John, March, 180) may represent John's weight in March drawn from the space (name X month X weight). Surrogates may be either taken from a defined domain, such as "name" or may be assigned automatically by the system. The important thing is that they uniquely identify each element. We note that a collection of values for John over time will have the same surrogate, and thus could be represented as $(s, (t, v)^*)$. $(t, v)^*$ represents an ordered sequence of pairs of times and their associated values. Thus, it is natural to think of temporal data as time ordered sequences of pairs (t, v) for each surrogate. We call each sequence for a single surrogate a *time sequence* (TS).

It is convenient to view a collection of TSs for a set of surrogates as a two dimensional array, as shown in Figure 1. We call such an array a *time sequence array*

(TSA). Note that each row of the array represents a TS, and that v_{ij} represents a value associated with some surrogate s_i and some time t_j . The existence of a value v_{ij} depends on the properties of the TS: some TSs may have many "null" values. A TSA made up from such TSs will therefore be *sparse*. As an example, consider the case of a bookstore database where the surrogates represent books, the time points represent days, and the values represent the number of copies of a certain book sold in a particular day. In that case, the two dimensional array will be fairly sparse, since only a small fraction of the books sell in any given day.

Viewing temporal data as TSs is an important fundamental concept to the logical and physical support of such data. As we will see next, categorizing the semantics of TSs facilitates the support of different types of temporal data uniformly. In section 4, we discuss operations over entire TSAs that are both powerful and practical. The physical structures and access methods proposed in section 5 hinge on the ability to treat sets of TSs collectively.

2.1 Examples

Before discussing properties of TSs, we examine a few examples shown in Figure 2. The first three examples are from business applications and the last three are from scientific applications. Figure 2a shows the variation over time in the cost of a certain item (e.g. a part sold in a shop.) Note that the cost in this example changes every few days (shown as notches in the time scale). If this sequence were to be presented as the (time, cost) pairs $(0,1)(4,2)(7,3)(11,2)(13,3)...$, then the interpretation of the sequence is that the cost of 1 applies to the time points 0 to 3, the cost 2 applies to time points 4 to 6, etc. We say that this TS has the property of being *irregular* in the time domain since the intervals between the time points 0,4,7,11,13,... vary. In addition, this TS has the property of being *step-wise constant* during each interval.

Figure 2b represents the number of units that a certain item sold per day. Assuming that this item sells every day, the TS is said to be *regular*. In addition, this TS is *discrete* since data values exist only at specific time points. Unlike the previous example, where values exist during the intervals, a discrete TS has no values associated with intervals. Values are associated only with discrete points.

In Figure 2c we show the sequence of a patient's visit to a hospital. This is a special case of the irregular, step-wise constant TS, in that only two values are used. The values can be thought of as (off, on), (0,1), (does not exist, exist), etc. according to the applica-

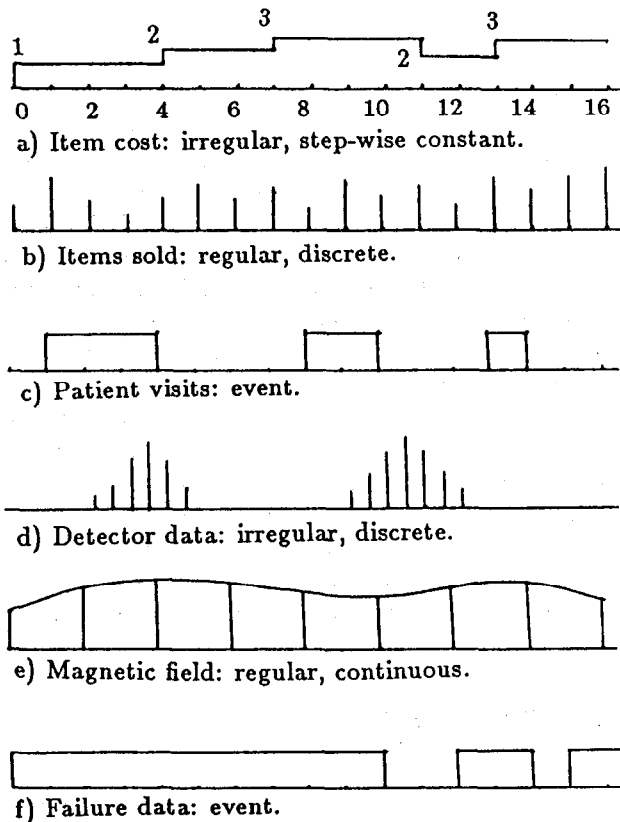


Figure 2: Examples of Time Sequences

tion. This special case is important since it represents *events* (a commonly occurring sequence). In addition, event TSs can be exploited in the physical organization of the data to achieve better storage utilization, as is pointed out in Section 5.

Figures 2d, 2e, and 2f show data from a typical physics experiment. High energy particles are collided and the paths of the resulting sub-particles are recorded by detectors. Figure 2d shows the pattern of measurements that a particular detector registers as sub-particles go by. This pattern can be categorized as *irregular* and *discrete*.

The sub-particles move in a magnetic field so that the paths of the charged sub-particles are curved. Since the magnetic field may fluctuate, it is measured in regular intervals, as shown in Figure 2e. (These measurements are needed later in the analysis phase in order to interpret the detector measurements.) Since detector measurements are taken at a higher frequency than magnetic field measurements, it is necessary to compute (interpolate) values for points in between those taken for the magnetic field. As a result, the TS for the magnetic field is *continuous*. In general, determining the value of a continuous TS at an intermediate point may not be a simple function.

For example, interpolation functions may require the values of several points to each side, not only the two adjacent points. Thus, continuous TSs will usually have an interpolation function associated with them. The default is a simple interpolation of the two adjacent points.

Figure 2f represents the history of failures of detectors. It is quite slow relative to the frequency of detector measurements taken (the time scale is measured in hours, rather than seconds). Nevertheless, this data is essential in order to disqualify some measurements taken at times that a detector is close to failing, because they may not be reliable. This is another case of an *event* TS.

2.2 Viewing time sequences

As can be seen from the examples above it may be more convenient to view TSs and TSAs graphically. The issue of the most appropriate user interface for temporal data is beyond the scope of this paper. However, it is worth considering briefly how TSAs could be represented in the context of the relational model.

One possibility is to view the TSA as a relation with the columns: surrogate, time, attribute. This will be a special type of an order preserving relation, where the sequence of values in time for the same surrogate have a special interpretation (i.e. that of a time sequence). This possibility is quite compatible with the relational model, as TSAs can be viewed as tables.

Another possibility is to introduce a special temporal data type, where for each surrogate a sequence of (time, value) pairs could exist. It is more difficult to represent such complex data types in a table form, but perhaps a two level representation could be used.

Regardless of the specific user interface chosen, we wish to emphasize here that the modeling concept of a TSA is an important one, since it carries temporal semantics, and powerful operators can be define over it. We will discuss the semantics and operators of TSAs next.

3 Properties of time sequences

There are four properties of TSs that are of interest to us, two of which were already observed in the examples above: regularity, type, static/dynamic, and time unit. These properties can be represented in a concise form in the data description part of a system. We discuss each below.

3.1 Regularity

There are two reasons for distinguishing between regular and irregular TSs. First, this is a semantic property that is needed by the system in order to interpret the TS. The sequence of time points of regular TSs can be described concisely as part of the data definition section. In addition, it is needed for operations on TSs and among TSs. This will be discussed further in Section 4. Second, this property is extremely important for the physical organization of TSs. As discussed in Section 5, the techniques for supporting regular TSs are much simpler than those needed for irregular TSs.

regular TSs are very common in scientific database applications. Most scientific experiments and simulations measure or compute series of data values at regular intervals, often by some mechanical device or detector. Irregular values usually result from manual measurements or from unpredictable events, such as the failure of a detector. Statistical databases also tend to be regular, since statistics are usually collected at regular intervals for analysis purposes. There is, of course, the possibility that data values are null (missing or unknown). TSs that contain a large number of null values can be thought of as irregular TSs. The term "time series" refers to a regular TSs, and is important in statistical analysis, because special analysis methods can take advantage of the regularity of the data. On the other hand, business transaction data, such as items sold in a store, are typically irregular over the time dimension.

3.2 Type

In the examples above, we have seen four types of TSs and their semantics explained. The four types are: discrete, continuous, step-wise constant, and event. They seem to be sufficient to describe the applications we encountered so far. Other types could be added if such a need arises.

The classification of TSs by type is important, since it permits the treatment of TSs in a uniform way. For example, two regular TSs of different types, say discrete and continuous can now be implemented using the same data structures. In effect, the type information removes the "behavioral" part of TSs from consideration of physical organization.

The type information is also important for operations over and among TSs. It can be used to interpret TSs as to whether a value exists for a certain time point, and whether a value can be implied or interpolated if it does not exist.

3.3 Static/dynamic

Another aspect of temporal databases that effects their logical and physical properties, is whether the temporal data set is static or dynamic. By "static" we mean a data set that has been fully collected, and no more additions over the time dimension are expected. Many examples can be found in SSDBs, such as data from an experiment which were fully collected, and are now ready for analysis. Similarly, many statistical databases represent collections that are considered complete, such as the census data or gasoline consumption over the last 10 years.

On the other hand, "dynamic" data sets are continuously growing. Most business data are dynamic since transactions, such as sales, represent a continuous process. Of course, one can choose to cut off the process at some point and consider the set of data collected so far as a static data set. As we will see later, dynamic databases are more difficult to support physically.

3.4 Time units

Regardless of whether the TS is regular or irregular, part of its semantic description should be the *time unit*. The time unit determines the interval between time points, such as seconds, hours, days, etc. In regular TSs a value exists for every time point, while in irregular TSs values exist only for some of the time points.

Every TS has a *start time* associated with it. Clearly, the start time and the time unit can be used to map the real time specification of a time point (e.g. March 17, 1986) to an ordinal position of the columns of the TSA (e.g. column 372). Such a mapping function should be supported by a temporal data management system. Note that a *end time* exists for static TSs only.

Properties summary

In summary, the following properties should be identified for TSs and made part of the data description file.

1. Regular/irregular.
2. Type: discrete, continuous, step-wise constant, event.
3. Static/Dynamic.
4. Time unit, start time, end time (end time in the static case only.)

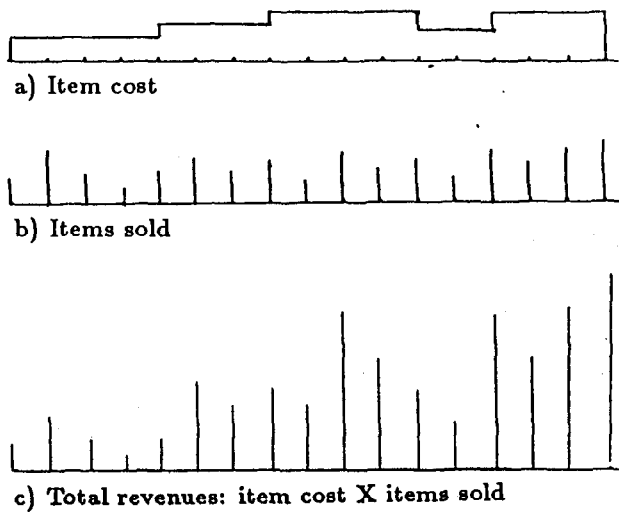


Figure 3: Composition operator

4 Operations Over Time Sequences

One of the advantages of representing temporal data with TSs is that it is possible to specify operations over the entire collection of values of TSs (or any desired part thereof) using a single operator. Consider, for example, that we wish to get the revenues per day of selling a certain book during the month of January. This situation is illustrated in Figure 3. Suppose that we have a TS that reflects the variations of the book price as shown in Figure 3a, and a second TS that represents the number of copies sold, as shown in Figure 3b. We want a new TS to be generated with the desired result, as shown in Figure 3c. To achieve this we need only to specify the multiplication of the two TSs. The effect of this operation would be to find matching pairs (in time) using the properties of the TSs, and to perform the multiplication for all matching pairs of the TSs. Furthermore, such an operation (which we call *composition*) can be specified for a set of books, i.e. it can be applied to entire TSAs.

Operations over the time domain have been suggested by several authors [e.g. Snodgrass 84, Clifford & Tansel 85]. Such operations (for example, selecting a time interval) are also useful in order to specify restrictions in the time dimension of TSs. However, we want to emphasize in this section the type of operations that can be applied directly to collections of TSs. We will only cover the functionality of such operations in a generic descriptive form, as the specification of such detail would require a separate paper.

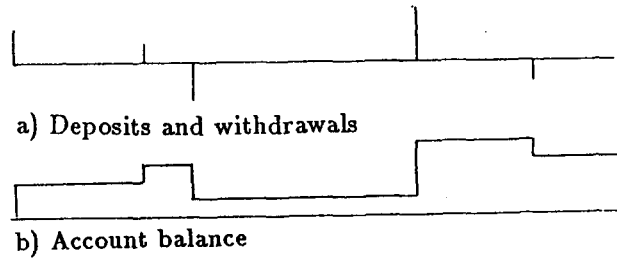


Figure 4: operation over a single TSA

4.1 Restriction

The purpose of the restriction operator is to derive sub-arrays. It is the conventional restriction operator applied to both the time and surrogate dimensions of a TSA. For example, referring back to the bookstore example above, we may wish to restrict our attention to mathematics books and the month of March only. The result of this operation is another TSA.

4.2 Operations over a single TSA

These operators can be applied to a single TSA to produce another TSA with the same number of surrogates and time points. Consider the example of Figure 4. Figure 4a shows an example TS from a TSA that represents deposits and withdrawals to accounts. Suppose that we wish to generate a running balance from this TS. Figure 4b shows the result of such an operation on the TS of Figure 4a. This is an example of an operation on a single TSA of type "discrete" that generates a TSA of type "step-wise constant"

In general, operations over a single TSA may change the type, but do not change the time unit of the original TSA. Operations that change the time unit involve aggregate functions over groups in the time domain. This case is discussed in section 4.5 below.

4.3 Composition

Figure 3 was an example of a composition operator. In general, two or more TSAs can be composed to form a new TSA. The composition may involve simple arithmetic functions, or complex functions that require a program. As an example of the need for complex functions with the composition operator, refer back to the scientific examples in Figures 2d, 2e, and 2f. In this application, we need to correct the detector data according to the magnetic field. This function requires the use of a program as it is quite complex. The corrected values need to be further composed with the

failure data, so that we can invalidate (say generate a null) incorrect values.

The use of TSAs for such problems simplifies the specification of the operations to be performed. A system that supports TSAs can handle the entire problem in a clean and concise fashion (including the interpolation of values of the magnetic fields.) Furthermore, the performance of such operations may benefit from efficient storage structures and access methods, such as those proposed in section 5.

The time unit of the new TSA will assume the time unit of one of the original TSAs involved in the composition. In the example just discussed, the new TSA will have the time unit of the detectors TSA. This needs to be specified with the composition operator.

4.4 Operations over surrogate groups

Suppose that in the bookstore example above, we want to get the total revenues per day for all mathematics books. We will have to total the revenue figure over all surrogates of the TSA representing mathematics books. This is an example of an aggregate function (sum) over a TSA that generates a single TS.

This example can be generalized to the case of generating totals for groups of surrogates according to some grouping function, say groups by category, such as mathematics, biology, etc. This is equivalent to the aggregate functions in data management systems, as applied to TSAs. The aggregate functions can, in general, include any function over groups, such as standard deviation. The result of such an operator is a TSA that contains a TS for each group that was aggregated. The time unit is the same as that of the original TSA.

4.5 Operations over the time domain

This class of operations is symmetric to the previous one, but it applies to the time domain of the TSAs. Aggregate operations in the time dimension cause the time unit to change. Suppose that we wish to get the total revenue per month for all mathematics books sold during 1985. The result would be a TSA with time points representing months for the year 1985.

In this case, we need the capability to describe groupings in the time domain. The most common case, we believe, will generate regular TSAs, since time is organized in a hierarchy of regular groupings, such as second, minute, hour, day, etc.

5 Physical Organization

Several authors [e.g. Lum et al 84, Ahn 86] have pointed out that storage efficiency is the key to the practicality of supporting temporal data. The reason is that history data can be very large relative to the current version of the database. Therefore, the design of the physical organization should take advantage, whenever possible, of the properties of temporal data so as to minimize the amount of storage used while maintaining reasonable access time. One obvious property that can be exploited is that while new data may be appended, changes to previous data are rare. A few changes (updates, deletes, and insertions) may be made in order to correct mistakes, but then the data stay practically unchanged. Thus, non-updatable data structures can be used to achieve better storage and access efficiency.

Other properties of TSs were discussed in Section 3. Two of these properties, "regular/irregular" and "static/dynamic", effect greatly the design of the physical structures. Their effect is discussed briefly below.

5.1 Design goals

It is easy to visualize the goals of the physical structure design of TSAs by referring to the two dimensional representation shown in Figure 1. There are three main issues: storage efficiency, efficient indexes, and partitioning of the array.

The most storage efficient physical structure that can be expected is a structure that stores the surrogate values and the time values only once, rather than with each data value, i.e. the surrogate and time values are "factored out". Obviously, if we implement the TSA as a two-dimensional array structure we achieve this goal. This can work well for regular TSAs, but special methods are needed for storing sparse arrays for irregular TSAs.

The second issue is providing indexes for the surrogate and time domains. In the surrogate domain conventional indexes can be used to locate the rows that represent TSs. The physical order of the rows and the best index to use depend on the access patterns to the TSA. We discuss the effects of access patterns below. The index in the time domain can take advantage of the fact that time points are ordered. Thus, in the case of regular TSAs, no explicit index needs to be created since the column for a given time point can be calculated using the start time and the time unit of the TSA. In the case of irregular TSAs, the array may be compressed and an index to the compressed time domain is needed. More details will be discussed

in the section on file structures below.

The third issue is the physical partitioning of the TSA into blocks. We are primarily interested in storage structures that optimize access to secondary storage, such as magnetic or optical disks. We believe that even if large amounts of main memory exist on a system, temporal data will be stored primarily on secondary devices. One reason is clearly the large amount of temporal data. Another reason is that temporal data will tend to be accessed less frequently as it gets older, and would be delegated to secondary storage.

The choice of physical partitioning depends on the expected access patterns to the data. For example, if we expect that most accesses in the time domain are for consecutive time points, then we should try and cluster the consecutive values of the TSs into blocks in order to minimize I/O to secondary storage for a given query. In addition, the property that TSAs are static or dynamic may effect the array partitioning. In the case of a static TSA, the distribution of the data values is known, and the array partitioning can be analyzed and optimized ahead of time. In the case of dynamic TSAs the partitioning has to adjust dynamically to the arrival of new data points. These issues are discussed in more detail in the file structures section below. But first we discuss several assumptions we make about access patterns.

5.2 Access patterns

The *first* assumption is that the order of values in TSs is important, and should be preserved in the physical structure. The reason is that operations in the time domain are usually over periods of time, such as asking for occurrences before or after a certain time or within a time range. Thus, one design consideration is to minimize the the number of pages (blocks) read from secondary storage for range queries in the time domain.

The *second* assumption is that we wish to have random access in the time domain. While in some applications one can envision accessing entire TSs, we believe that efficient access to parts of the sequences is necessary. Thus, some indexing method on the time domain is necessary.

The *third* assumption is that there may be applications where ordering in the surrogate domain is desirable. For example, in the books database mentioned above, we may wish to order the books by category in order to facilitate more efficient access to groups of books according to their categories. Thus, we would like to permit the specification of partitioning in the (ordered) surrogate domain as an option. We call this

option below the “surrogate partitioning” option.

The *fourth* assumption is that random access to surrogates is necessary. This implies that there must be a mapping (an index) from the list of surrogates to the ordinal position of the rows of the TSA. An ordinary index can be used for this purpose, such as a B-tree or a hashing method. We will assume the existence of such an index. In many cases, an index for the surrogates already exists in order to support the non-temporal part of the database. If the order of the surrogates in the TSA is the same, then the same index can be augmented for accessing the TSA.

The *fifth* assumption is that a secondary index over the data values is not needed in most applications. Such an index can potentially be very expensive in terms of storage, because the number of entries for such an index is in the order of the number of data values. In any case, such an index provides a marginal benefit in situations where the typical access to the data involves restrictions on the surrogate and time domains. We will assume that such indexes (if absolutely necessary) would use conventional indexing methods.

5.3 File structures

Physical organization methods for temporal data were recently discussed in the literature. In [Lum et al 84] the authors consider the support of temporal data in the context of the relational model. Their methods consist of chained structures of values in the time domain. These chained structures are similar in concept to TSs. However, our approach differs in that we consider the support of the set of all TSs over all the surrogates in order to achieve common storage efficiency and common indexing structures. The concept of “attribute versioning” mentioned in [Ahn 86] can also be thought of as representing TSs. While clustering of attribute versions is proposed generically, there is no discussion of how to store and access the clusters jointly.

In the designs below, we distinguish between the cases of regular and irregular TSAs. For each case we discuss the effects TSAs being static or dynamic and the effects of the “surrogate partitioning” option. Because of space limitations the designs are only sketched here. More details can be found in the full version of this paper issued as report LBL-21143.

5.3.1 Regular TSAs

Regular TSAs can be simply represented as a two dimensional non-sparse array. All the surrogates have regular TSs whose time points coincide, and thus the

time points can be “factored out”. Furthermore, it is not necessary to store the sequence of time points or to build an index over them. As mentioned before, given a time point, it’s corresponding column position can be calculated using the start time and time units of the TSA.

a) The static case

In order to preserve the temporal order, the array is stored in a row-wise fashion, i.e. the TSs are concatenated in the order of the surrogates. The allocation to pages (or blocks) of secondary storage is straight forward, in the order of the concatenated list. The access to this structure is achieved with array linearization. It requires a simple calculation that uses the ordinal positions of the row(s) and column(s) requested. (The array linearization algorithm is well known, and will not be discussed further here.)

The effect of the “surrogate partitioning” option is that we now want to have clustering of data values in both the temporal and surrogate dimensions. We partition the array into cells of equal size. As a practical matter the size of the cells can be chosen to be the size of secondary storage pages. The dimensions of the cells are parameters that can be chosen by a database administrator to fit the application. The order of elements within a cell is row-wise to preserve the order of TSs. Thus, the elements of a cell can also be accessed randomly using array linearization. In order to read a data value, two array linearization computations are needed: one to determine the appropriate cell, and one to locate the position of the value in the cell.

This cell partitioning scheme still retains the following properties: the surrogate and time values are “factored out”, the time points are not stored explicitly (they can be computed), random access in the time dimension can be achieved without the overhead of an index, and random access in the surrogate dimension can be achieved as long as the ordinal position of the surrogate is known, usually available from existing indexes. The dimensions of the cells are a design choice that determines the effectiveness of access in the time and surrogate domains. Note that choosing a surrogate dimension of 1 is equivalent to the initial design where the “surrogate partitioning” option is not required.

b) The dynamic case

The dynamic case can be handled as an extension to the cell partitioning mentioned above. The surrogates are assigned to cells, and the cells fill up as new values arrive. Since the rate of arrivals is the same for all

surrogates, we only need to keep track of the active cells currently being filled and treat the previous cells as static.

The access in the time domain is also similar to the static case. Given a surrogate and a time point, one can calculate the cell number and the displacement within the cell.

One difficulty associated with dynamic TSAs is that the cells currently being filled have to be managed. Suppose that there are m surrogates. The worst case is when each cell is assigned to a single surrogate, since we need m cells to be “active” at the same time. If we do not have a large enough buffer to hold the m cells, we need to write the cells to secondary storage in a piece-wise fashion.

5.3.2 Irregular TSAs

The support of irregular TSAs is more complex. Clearly, we could store the (irregular) TSAs as series of (time, value) pairs. However, the ability to “factor out” time values that are shared across surrogates is lost, as well as the simple indexing capability over the time domain that exists in the regular case. In the following design we take advantage of the static nature of the TSs to achieve these goals.

a) The static case

Consider, for example, the bookstore example mentioned before. Each TS will have points only for the days that books were sold. Assume that each non-existing point is represented as an explicit null value.

Obviously, this sparse array can be accessed in a way similar to the regular case using array linearization. The problem is how to get rid of the nulls while preserving efficient access. Such a technique, called “header compression”, was developed previously in [Eggers & Shoshani 80]. It is essentially a run-length compression method where counts of null and non-null sequences are stored in a separate header. This permits the elimination of all the nulls from the stored data. The header can be organized as an index so that it can be searched in logarithmic time. We can apply this technique to the sparse array by concatenating the rows, resulting in a long sequence of values, many of them null. In order to find an element in the array we first use the array linearization computation and then the header to map into the non-null values.

As mentioned in section 3, the event type is a special case of irregular TSs since it has only two values. Using the header compression method mentioned above, it can be handled with a header only where the header keeps track of sequences of 0’s and 1’s. There

is no compressed data values list, only the header. Thus, it is worth treating the event type as a special case as it greatly enhances storage utilization.

The above method may not be very effective in cases that the array has little overlap of the time points across surrogates. In the worse case, if there is no overlap at all, the number of points on the time line is the same as the number of data values. Thus, the storage requirements are equivalent to storing the (time, value) pairs. However, we do not believe that such applications are common.

Now, consider the effects of the "surrogate partitioning" option. As in the static case we can partition the array into cells. However, in this case the array is sparse. This problem is precisely that of designing a grid file [Nievergelt et al 84] in a two dimensional space. The selection of the partition lines (in both the time and surrogate dimensions) is determined by the grid file method so as to minimize empty space in cells. This method partitions the array dynamically: each time that a cell overflows a new partition line is introduced. Experimental results in [Nievergelt et al 84] show that a storage utilization of about 70% can be expected on the average. However, since we are dealing with the static case, better methods could be used that take advantage of pre-analyzing the entire data set. In a recent paper [Rotem & Segev 85] which deals with the static case of grid files, the authors have developed methods that achieve better storage utilization than the dynamic grid file.

There are other cell partitioning methods that could be used (e.g. Quad-trees). Experimentation would determine the most appropriate methods. Our main point here is that cell partitioning methods seem to be the most appropriate in this case, and that pre-analysis for the static case should yield better partitioning.

b) The dynamic case

As can be expected by now, this is the most complex case as we need to deal with irregular TSs which grow dynamically. In addition, different TSs have different rates of growth that can change over time.

First, we consider the case where the "surrogate partitioning" option is not required. In this case, cell partitioning methods are not effective because we wish to preserve the physical ordering of TSs, and these methods partition the TSs into cells according to the data distribution. Thus, we consider other alternatives.

Suppose that pages (blocks) are assigned to each surrogate. The pages fill up at different rates and therefore at different times. The effect is that each

surrogate has a string of pages associated with it, where each page has a different start time and duration. This information can be organized into an index where each surrogate has an ordered list of (page number, start time) elements. Given a surrogate and time (or range of time), such an index can be searched for the page(s) holding the corresponding values. The cost of such an index is obviously proportional to the number of pages. If we assume that a typical page holds 100-200 values, then the index size would be about 1% of the array size.

There are two problems associated with this scheme. One is indexing in the time domain over the pages which have different start times and durations. The second problem is that the pages of slow rate surrogates may remain mostly empty for a long time. This wastes storage.

The problem of searching pages with different start times and durations was addressed recently in the context of proximity search in space and searching of time intervals [Hinrichs & Nievergelt 83, Rubenstein 85]. We describe the solution in terms of our context. The idea is to represent pages in a two dimensional space, whose coordinates are start time and duration. Thus, each page is represented as a point in this space. A search for all pages that contain a range in the time domain translates into searching a region in this two dimensional space. By organizing this two dimensional space as a grid file, one can efficiently find all the pages that fall in that region. This technique seems most appropriate for our indexing problem.

The solution to the problem of wasted storage is not obvious. The intuitive solution is to store several slow rate surrogates on the same page. The higher the rate of the surrogate, the fewer the number of surrogates that will be stored on the same page. However, such a scheme introduces more complex indexes and access methods. It requires further research.

Now, we consider the effects of the "surrogate partitioning" option. Obviously, the dynamic grid file method can be used here. However, this is an unusual case of a grid file where the boundary keeps growing and new elements are only added at the end of the time domain. We are not familiar with any work that treats this case, and consider the performance of such a method an open problem. However, we note that we can cut off the grid file at selected intervals (say, every month in the bookstore example above) and treat the arrays thus created as static. Each static array created periodically can be organized as a static grid file using the methods mentioned above. This could be an effective way of exploiting the linear insertion property of TSs. As mentioned before, other cell partitioning methods could be considered. The study of

their relative performance is open for further research.

6 Summary and Conclusions

In this paper we have described a framework for the support of temporal data. It is based on the concept of a time sequence, which represents sequences of data values over time for a given surrogate. We discussed properties of time sequences that allow the treatment of different types of time sequences in a uniform fashion. We described operations over time sequences, and their power to manipulate temporal data. Finally, we have analyzed the design criteria for the physical support of time sequences, and presented designs for physical data structures and access methods for such sequences.

The main advantage of our approach is the ability to operate on an array of time sequences with a single operator. Also, viewing the data as time sequence arrays suggests efficient physical structures for the support of temporal data.

Future work includes the development of detailed syntax for operations on time sequences, and the evaluation of the physical designs with analytical methods and real data.

7 Acknowledgements

We gratefully acknowledge the many discussions and support of our colleagues Frank Olken, Doron Rotem, and Harry Wong. Arie Segev and the referees also provided many useful comments.

References

[Ahn 86]

Ahn, I., Towards an Implementation of Database Management Systems with Temporal Support, *Proceedings of the International Conference on Data Engineering*, 1986, pp. 374-381.

[Bolour et al 82]

Bolour, A., Anderson, T.L., Dekeyser, L.J., Wong, H.K.T., The Role of Time in Information Processing: A Survey, *ACM-SIGMOD Record*, 12, 3, 1982, pp. 27-50.

[Clifford & Tansel 85]

Clifford, J., Tansel, A., On an Algebra for Historical Relational Databases: Two Views, *Proceedings of the ACM SIGMOD International Conference on Management of Data*, May 1985, pp. 247-265.

[Eggers & Shoshani 80]

Eggers, S. J., Shoshani, A. "Efficient Access of Compressed Data," *Proceedings of the International Conference on Very Large Databases*, 6, 1980, pp. 205-211.

[Hinrichs & Nievergelt 83]

Hinrichs, K., Nievergelt, J., The Grid File: a Data Structure Designed to Support Proximity Queries on Spatial Objects, *Proceedings of the Workshop on Graph Theoretic Concepts in Computer Science*, (Osnabruck, 1983).

[Lum et al 84]

Lum, V., Dadam, P., Erbe, R., Guenauer, J., Pistor, P., Walch, G., Werner, H., Woodfill, J., Designing Dbms Support for the Temporal Dimension, *Proceedings of the ACM SIGMOD International Conference on Management of Data*, June 1984, pp. 115-130.

[Nievergelt et al 84]

Nievergelt J., Hinterberger H., Sevcik K.C., The Grid File: An Adaptable, Symmetric Multikey File Structure, *ACM Transactions on Database Systems* 9, 1, March 1984, pp. 38-71.

[Rotem & Segev 85]

Rotem, D., Segev, A., Optimal and Heuristic Algorithms for Multi-Dimensional Partitioning, Lawrence Berkeley Laboratory report LBL-20676, December 1985.

[Rubenstein 85]

Rubenstein, B., Indexes for Time-Ordered Data, University of California Technical Report, Berkeley, 1985.

[Snodgrass 84]

Snodgrass, R., The Temporal Query Language TQuel, *Proceedings of the Third ACM SIGMOD Symposium on Principles of Database Systems (PODS)*, Waterloo, Canada, April 1984, pp. 204-213.

[Snodgrass & Ahn 85]

Snodgrass, R., Ahn, I., A Taxonomy of Time in Databases, *Proceedings of the ACM SIGMOD International Conference on Management of Data*, May 1985, pp. 236-246.