LOGICAL MODELING OF TEMPORAL DATA

Arie Segev† and Arie Shoshani‡

† School of Business Administration and
Lawrence Berkeley Laboratory
The University of California
Berkeley, California 94720

‡ Lawrence Berkeley Laboratory
The University of California
Berkeley, California 94720

Abstract

In this paper we examine the semantics and develop constructs for temporal data independent of any traditional data model, such as the relational or network data models Unlike many other works which extend existing models to support temporal data, our purpose is to characterize the properties of temporal data and operators over them without being influenced by traditional models which were not specifically designed to model temporal data We develop data constructs that represent sequences of temporal values, identify their semantic properties, and define operations over these structures

1 INTRODUCTION

Our approach to modeling temporal information is to start with the understanding and specification of the semantics of temporal data independent of any specific logical data model (such as the relational model, the entity-relationship model, the CODASYL network model, etc) We differ from many other works whose starting point is a given model which is extended to support temporal data Examples of works that extend the relational model are [Ariav et al 84, Clifford & Tansel 85, Gadia 86, Lum et al 84], and examples of works that extend the Entity-Relationship model are [Klopproge 81, Adiba & Quang 86] We believe that our approach leads to precise characterization of the properties of temporal data and operators over them without being influenced by traditional models which were not specifically designed to model temporal data Once such characterization is achieved, we can attempt to represent these structures and operations in specific logical models Typically, this will require extensions or changes of the logical models, or perhaps will point out that some models are inadequate for temporal modeling

Our initial motivation for temporal data comes from applications in scientific and statistical databases (SSDBs), where physical experiments, measurements, simulations, and collected statistics are usually in the time domain Unlike many business applications that deal only with current data, SSDB applications are inherently time dependent, and in most cases the concept of a ' current version" does not even exist However, it is obvious that in business applications, temporal data is also essential Many business applications keep a complete history of transactions over the database This is quite obvious in most business applications, such as banking, sales, inventory control, and reservation systems Furthermore, often this history needs to be statistically analyzed for decision making purposes Other applications where the time domain is inherent include engineering databases, econometrics, surveys, policy analysis, music, etc

We do not attempt to model temporal "cause and effect" events (e g send a paper to a referee, if the referee does not respond within a month, send a reminder letter), as described in [Studer 86] We are mainly interested in capturing the semantics of ordered sequences of data values in the time domain, as well as operators over them Consequently, we define the concept of a Time Sequence (*TS*), which is basically the sequence of values in the time domain for a single entity instance, such as the salary history of an individual or the measurements taken by a particular detector in an experiment We define the properties of the *TSs*, such as their type (continuous, discrete, etc), their time granularity (minutes, hours, etc), their life span, and other

The association of these properties to the *TSs* allow us the treatment of such sequences in a uniform fashion First, we can define the same operators for *TSs* of different types, such as to select parts of a *TS* or to aggregate over its values Furthermore, we can define operators between *TSs* of different types, such as multiplying a discrete *TS* with a continuous *TS* Second, we can design the same physical structures for different types of *TSs* We can also take advantage of some of the properties for designing more efficient storage and access of temporal data

In a recent paper [Shoshani & Kawagoe 86], we have described the Time Sequence framework, with

preliminary ideas on operators aver *TSs*, and on physical organization for *TSs* Another paper [Rotem & Segev 87] describes the design of a physical database structure for *TSs* In this paper we are concerned with the precise specification of *TSs*, collections of *TSs*, and operators over them This paper is an extension of an earlier version [Segev & Shoshani 86]

In section 2, we present our view of temporal semantics starting with the basic notion of a temporal value, and leading to constructs representing collections of *TSs* In section 3, we give precise definitions of the temporal constructs for a time sequence (*TS*), and a time sequence collection (*TSC*) We need these constructs in order to define precisely the operators over them In section 4, we describe the general structure and properties of such operators, and define their syntax In section 5, we discuss the requirements that would have to be added to the relational model in order to accommodate the temporal data model developed here Section 6 contains a summary and planned future work

2 TEMPORAL SEMANTICS

In this section we describe the semantic properties of temporal data and the intuition for the data constructs we have chosen In the next section we define precisely these constructs

2 1 Time sequences

In order to capture the semantics of temporal data, we start with some basic concepts A temporal data value is defined for some object (e g , a person), at a certain time point (e g , March, 1986), for some attribute of that object (e g , salary) Thus, a temporal data value is a triplet $<s,t,a>$, where s is the surrogate for the object, t is the time, and a is the attribute value Note that for a non-temporal data value t is considered the "current" value, and therefore omitted

An important semantic feature of temporal data is that for a given surrogate the temporal data values are totally ordered in time, that is they form an ordered sequence For example, the salary history of John forms an ordered sequence in the time domain We call such a sequence a *time sequence* (*TS*) *TSs* are basic structures that can be addressed in two ways Operators over them can be expressed not only in terms of the values (such as "salary greater than 30K"), but also in terms of temporal properties of the sequence (such as "the salary for the last 10 months", or the "revenues for every Saturday") The results of such operators is also a *TS* whose elements are the temporal values that qualified

Since all the temporal values in a *TS* have the same surrogate value, they can be represented as $<s,(t,a)^*>$, that is a sequence of pairs (t,a) for a given surrogate It is convenient to view *TSs* graphically as shown in Figure 1 Imagine that Figure 1a shows a daily balance of a checking account Note that in this case the pairs in the *TS* have the values (1,10), (6,3), (8,7), (14,5), (17,11), but that these values extend to other time points of the sequence as shown We label such behavior of the *TS* "step-wise constant" In contrast, Figure 1b shows a *TS* of the number of copies sold per day for a particular book Here the temporal values apply only to the days they are specified for We call this property of the *TS* "discrete" A further example is shown in Figure 1c which represents measurements of a magnetic field by a particular detector taken at regular intervals (say, every second) In this case, one can interpret the *TS* as being "continuous" in the sense that values in between the measured points can be interpolated if need be

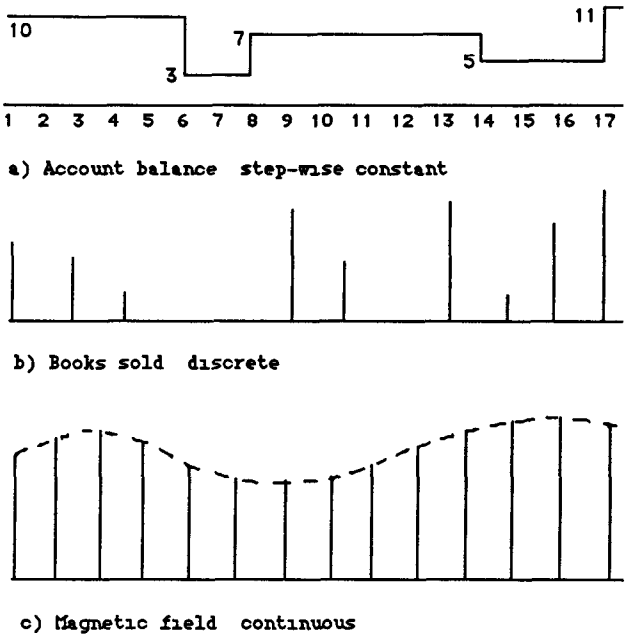

Figure 1 Example of time sequences

These examples illustrate that while a *TS* is defined structurally as an ordered sequence of temporal values, its semantic behavior can differ according to the application involved In the next section, we define precisely *TSs* and their semantic properties Such a definition will permit us to treat *TSs* uniformly when defining operations among them In addition, one can design the same physical structures for *TSs* that have different semantic properties, such as continuous or discrete Physical structures for *TSs* were discussed in a recent paper [Rotem & Segev 87]

2 2 Time sequence collections

It is natural and useful to consider the collection of *TSs* for the objects that belong to the same class (or type) For example, consider the collection of *TSs* that represent the salary histories for all the employees in the database We refer to such a collection as the *time sequence collection* (*TSC*) The usefulness of the *TSC* structure stems from the ability to address the temporal attributes of an entire class, and relate them to other (possibly non-temporal) attributes of the class For example, we may be interested in the salary history

of employees in the computer department for the last 6 months Such operations over *TSCs* are discussed in the Section 4

Since our purpose here is to model temporal semantics, we choose to stay away from modeling concepts of any specific data model, such as relations, entities, relationships, record types, sets, etc Rather, we prefer the concept of a class of objects and the representation of *TSs* for them A *TSC* will then be used as the construct to represent the temporal values associated with a class Our approach is to first define the structure and properties of *TSCs* as well as operations over them, and then find a mapping to any specific data model that we may choose (relational, Entity-relationship, etc) Next, we describe classes more precisely The concepts below have appeared in several forms in the literature We adopt them here because they are convenient for describing *TSCs*

2 3 Classes

A *class* is any collection of objects that have the same attributes (such as a person, a department, or a detector) Every object of a class has a unique identifier, called a *surrogate* A *composite class* is a class whose identifier requires more than a single surrogate For example, "attendance" is a class whose identifier is "student, course" In general, a composite class can be defined by using the identifiers of other classes or other composite classes For example, suppose that a course can be taught by several professors, then an "assignment" class can be defined from the composite class "attendance" and the class "professor" A composite class can be thought of as a result of the "aggregation" construct discussed in [Smith & Smith 77] For our purposes, this definition of a class is sufficient Classes with similar properties will result from the "generalization" construct described in [Smith & Smith 77]

Note that composite classes as described above are constructs that are quite general For example, in the Entity-Relationship (ER) model entities usually correspond to simple classes, and relationships to composite classes However, in the ER model one cannot define new relationships using existing relationships, while composite classes can be defined using other composite classes A CODASYL network model is even more restrictive, not only that sets cannot be used to define further sets or record types, but sets cannot have their own attributes as is the case in relationships of the ER model Composite classes have no such restrictions In the relational model one can define relations with composite keys, but the model carries no explicit information that the keys came from other relations Next, we discuss *TSCs* as the constructs that describe the temporal properties of classes

2 4 Simple *TSCs*

We start with a simple *TSC*, which is defined for a simple class (i e a class with a single surrogate as its identifier), and a single temporal attribute A simple *TSC* can be described as a triple (S,T,A) where S, T, and A are the surrogate, time, and attribute domains, respectively A simple *TSC* can thus be viewed as the collection of all the temporal values of a single attribute for all the surrogates of a simple class It is convenient to think of a simple *TSC* in a two-dimensional space as shown in Figure 2

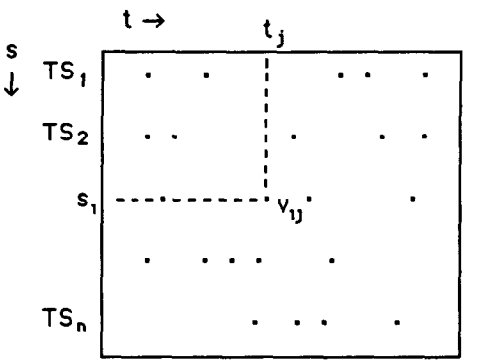

Figure 2 A two-dimensional representation of a time sequence collection

In this representation, each row corresponds to a *TS* for a particular surrogate The dots represent points where temporal values exist In the next two sections we describe the semantics of simple *TSCs* and operations over them

We note here that non-temporal values can be represented as a special case of the *TSC* A non-temporal attribute has a single time point (usually "current time"), and therefore its *TSC* will be reduced to a single column structure This observation is useful later for operations that involve both temporal and non-temporal data

Complex *TSCs* are *TSCs* whose components S, T, and A do not represent a single element We discuss each in turn

The case that S is not a single element corresponds to a composite class We denote this case as $(\bar{S},T,A)$ This case can also be visualized as a two-dimensional structure, where the rows are labeled with the composite surrogate identifier of the class This structure is useful in representing the temporal behavior of relationships and their attributes For example, suppose that people are assigned to different projects over time The history of such a relationship can be represented as $((S,S),T,A)$, where (S,S) corresponds to (people, projects), and A corresponds to a binary assignment attribute (which can be represented as the values 0 and 1, for example)

The case that T is not a single element corresponds to a situation where temporal values have more than one time sequence associated with them Such situations were discussed in length in [Snodgrass & Ahn 85], where the distinction is made between "transaction time" and "valid time" Transaction time

is the time that an action is recorded, and valid time is the time that the action takes effect (e g a salary raise recorded in March, but it is effective in January) Other authors called such times as "logical" and "physical" times We denote this case as $(S,\bar{T},A)$ This case requires the support of a single *TSC* with multiple time lines

The case that A is not a single element occurs when several attributes occur (or are measured) at precisely the same time points For example, when collecting air pollution samples at regular intervals, several measurements are taken, such as carbon monoxide, nitrogen compounds, etc We denote this case as $(S,T,\bar{A})$ In addition to the semantic information that these attributes occur together in the time domain, this case also provides a concise way of representing together several *TSCs* that have the same temporal behavior An important special case is in representing non-temporal data as the degenerate *TSC* $(S,\bar{A})$, where all the non-temporal attributes can be treated together in a single *TSC*

Obviously, any combination of the above three cases can exist simultaneously To simplify our discussion here, we only describe the constructs (section 3) for simple *TSCs* However, the operators (section 4) are defined for *TSCs* with multiple attributes $(S,T,\bar{A})$ Future work will address other extensions for complex *TSCs*

3 CONSTRUCTS FOR TEMPORAL DATA

In this section, we define the basic constructs of the temporal data model (TDM) These constructs can be processed by the data manipulation operators presented in the next section In Section 2 we defined a temporal value as a triplet $<s,t,a>$, and a time sequence *TS* as an object $<s,(t,a)^*>$ consisting of a time-ordered set of temporal values for a single surrogate instance

We distinguish between the *time points* and the *data points* of a *TS* The time points of a *TS* are all the potential points in time that can assume data values In contrast, the data points of a *TS* are only the points that actually have data values associated with them For example, suppose that the salary of an individual can change during any month of a certain year, but actual changes took place in April and October Then, only these two months are called the data points of that *TS* Since each of the months could potentially have a value, we refer to them as the time points of the *TS* In general, the data points of a *TS* are a subset of the time points Next, we define the properties of a *TS*

Time Granularity

This property specifies the granularity of the time points (t) of a TS, i e the points in time that can potentially have data values We allow for two time granularity representations - ordinal and calendar The ordinal representation simply signifies that the potential time points are counted by integer ordinal position (1,2,3,) The calendar representation can assume the usual calendar time hierarchy values year, month, day, , second, etc

Life Span

Each *TS* has a life span associated with it The life span is specified by a start_point and an end_point defining the range of valid time points of the *TS* The start-times and end-times are also represented as ordinal or calendar Usually, the time granularity and the life span have the same representation, i e they are both ordinal or both calendar However, this is not a requirement For example, an experiment may produce a *TS* of measurements taken every second Suppose that the start and end times of the experiment are not important Thus, this *TS* has a calendar granularity of a second, and an ordinal life span

We are interested in three cases of a life span

a) start_point and end_point are fixed
b) start_point is fixed and end_point is current_time
c) a fixed distance is defined between the start_point and the end_point The end_point is "current_time" and the start_point is dynamically changed to maintain the fixed distance from the end_point

In general, the life span can consist of disjoint non-continuous segments However, this feature can be represented explicitly in the *TS* by using "null" data values A time point with a null value has the meaning that a data value does not exist for this time point Using null data values can simplify the processing of *TS*, since it is not necessary to check the legal segments of the life span Thus, we prefer the use of null values rather than defining multiple segments in the life span

Regularity

A *regular TS* contains a value for each time point in the life span interval Thus, the data points of a regular *TS* are the same as the time points of that *TS* An *irregular TS* contains values for only a subset of the time points within the life span interval

While the specification of this property is quite useful for the design of physical structures, it has semantic value as well It is important for a user to know whether a data value can be expected for every time point of the *TS* Also, most time series analysis methods can only be applied to regular *TSs*

Type

The type of a *TS* determines the data values of the *TS* for time points that do not have explicit data values In general, there is an interpolation function associated with each *TS* Some of the interpolation functions are very common, and therefore are given specific type names below

We are interested in the following types of time sequences

a) Step-wise constant if (t_i,a_i) and (t_k,a_k) are two consecutive pairs in TS such that $t_i < t_k$, then $a_j = a_i$ for $t_i \leq t_j < t_k$

b) Continuous a continuous function is assumed between (t_i,a_i) and (t_k,a_k) which assigns a_j to t_j $(t_i \leq t_j \leq t_k)$ based on a curve fitting function

c) Discrete each value (a_i) in TS is not related to other values Consequently, missing values cannot be interpolated

d) User defined type missing values in TS can be computed based on user defined interpolation functions

It should be noted that the type property may apply to both a regular and irregular TS For example, a type step-wise constant for a regular TS means that the associated interpolation rule applies to all granularities smaller than or equal to the granularity of the TS, this is true for all continuous types of which step-wise constant is a special case

Now, we can define a *time sequence collection* (*TSC*) more precisely A *TSC* is a collection of time sequences for the same surrogate class and with the same properties The *TSC* is a basic construct in the TDM and can be manipulated by the operators discussed in the next section

It follows from the above definition of a *TSC* that the properties of the *TSC* are the same as those defined for a *TS*, since all the *TSs* that belong the the same *TSC* have the same properties Below is an example of a *TSC* and its properties, as well as two instances of *TSs* that belong to it

Example
Surrogate class bank account number
Temporal-attribute type account balance
Time granularity day
Life span start_point=1/1/86 , end_point=1/9/86
Regularity irregular
Type step-wise constant

A *TS* for account number 1462 is
{(1/1/86,57), (1/4/86,50), (1/6/86,65), (1/9/86,60)}
A second *TS* for account number 2526 is
{(1/1/86,35), (1/3/86,45), (1/7/86,55)}

A two-dimensional representation of this *TSC* is shown in Figure 3a, where rows represent different surrogates and columns represent the time points of the *TSC* Although it is convenient to view a *TSC* in this two-dimensional space, we do not imply that this is the preferred structure either for physical or logical representation Indeed there may be different representations that can be used for illustrative or visual purposes For example, Figure 3b shows a graphical representation of the *TSC*, where the interpolation function (in this case step-wise constant) is used to fill in missing time points Figure 3c shows a tabular (relational) representation of the *TSC* Figure 3d is another tabular representation, where the surrogate values are not repeated (this representation was referred to in the literature as non-first normal form of a relation) Finally, Figure 3e shows an expanded tabular form of the *TSC*, where the missing values have been interpolated

The above examples illustrate that the operators defined for *TSCs* should not be dependent or selected according to a particular representation Rather, when a particular representation is chosen to conform to a given model (such as tables as in Figure 3c for the relational model), then the operators defined in the next section should be supported by that model

4 OPERATIONS OVER *TSCs*

4 1 Principles

The operators presented in this section obey two principles These principles hold regardless of the complexity of the operators

The first principle is that every operator over one or more source *TSCs* will produce a single target *TSC* This principle permits the iterative application of operators to form a sequence of complex operations when needed It should be noted that, in a particular implementation, the basic operators can be combined into higher-level operators

The second principle is that every operator must have three functional parts target specification, mapping, and function, we describe them below This principle ensures that all the operators are consistent It also permits complex user-defined operators to conform to the format of the other operators

Target Specification

The target specification part determines the valid points of the target *TSC* A point of a *TSC* is specified uniquely by the s and t components of the temporal value As will be shown later, a target specification can result in a subset of the data points of the source *TSC*, or can have different data points specified (e g the source *TSC* specifies days, and the target *TSC* specifies months)

Mapping

The mapping† part specifies for each point of the target *TSC* the set of points of the source *TSC* to be manipulated to generate a target temporal value For example, in an aggregation operation, for each target point there is a set of source points used to generate the target value

† We use here the term mapping in the sense of correspondence between source and target points rather than in its precise mathematical meaning

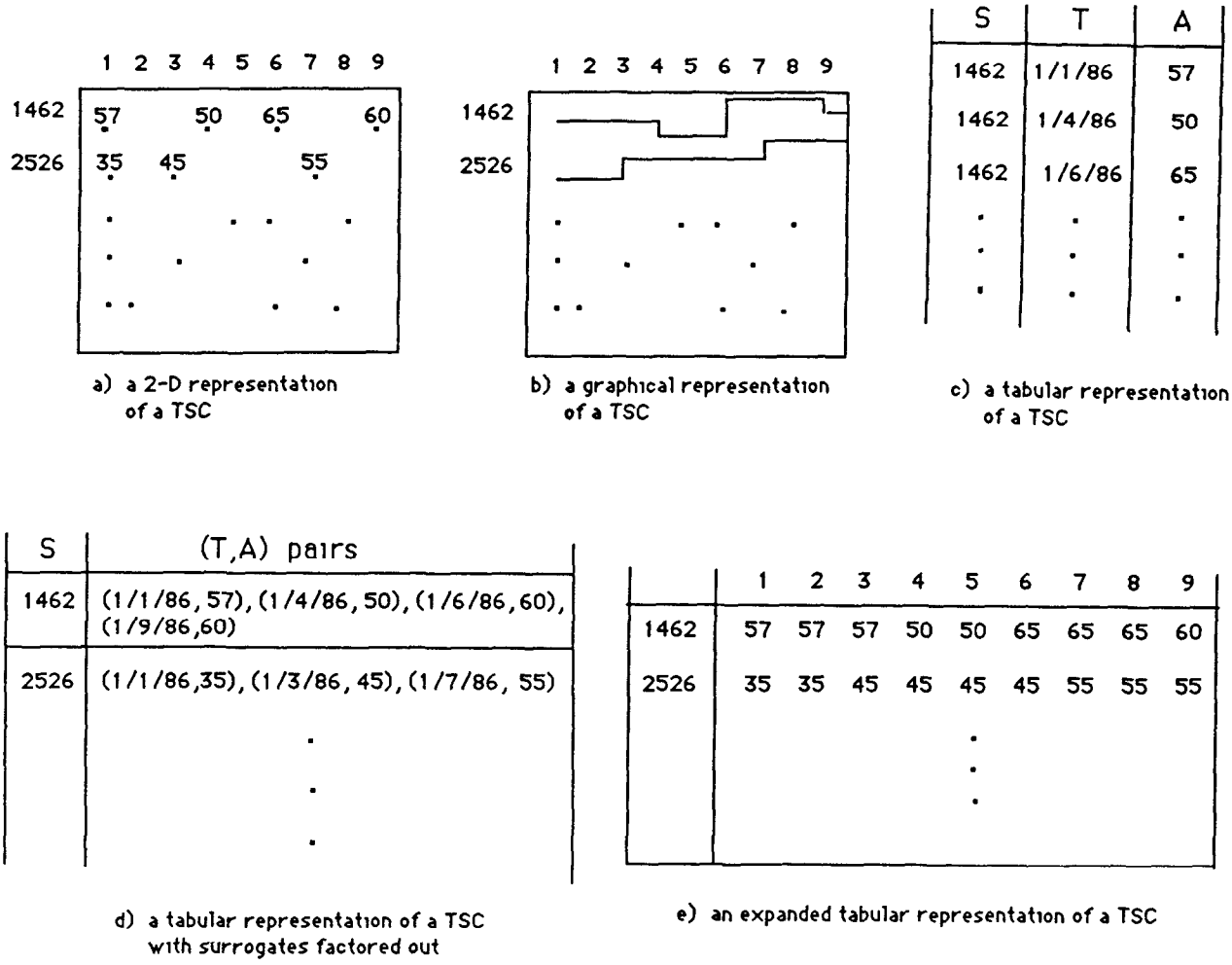

Figure 3 Different representations of a *TSC*

Function

The function part specifies the function to be applied to the values of the source points in order to generate the target value This function may be as simple as a sum and other arithmetic operations, or it can involve complex computations that could be user specified by means of a program

It should be pointed out that any of the above parts may be specified as "identity" In the case of target specification, an identity specification means that all the points in the source *TSC* will appear in the target *TSC* An identity mapping means that each target point corresponds to the same source point An identity function means that the target value is the same as the source value At the other extreme, each of these parts can be completely user specified by means of a program We will give below an example that requires user defined parts

4 2 Common operators

In this section we describe the basic retrieval operators of the TDM The retrieval operators and their three functional parts are summarized in Table 1, their precise syntax is given in the Appendix The syntax that we adopted is SQL-like and its general form is given below (the use of '[', ']', '|', and '*' are explained in the Appendix)

operator-name INTO target-tsc function
FROM source-tsc [,source-tsc]*
WHERE target-specification
GROUP BY|TO mapping-specification

In general, there are many operators over time sequences that are useful for different applications For example, there is a large body of literature on time series analysis that uses different operators for statistical analysis, such as regression, cross correlation, etc Our purpose here is to identify several common operator classes by means of the three parts mentioned above target specification, mapping, and function For more complex operations, user defined routines (which can be stored in libraries) can be incorporated in the queries in place of each of these three parts The most general case is when all three parts are user defined as shown in the last entry of Table 1 In general, user defined parts can replace any of the parts of the operators shown in Table 1

In addition to the operators shown in Table 1, there are additional operators that we do not specify here in detail These include a set operator to combine *TSCs*, update operators, and a data definition operator The data definition operator is needed in order to create *TSCs* and to define their properties (discussed in Section 3) The data definition operator can also be used to explicitly change the properties of an existing *TSC* (implicit changes may occur as a result of data manipulation operations, e g the time granularity of a *TSC* is changed when the user specifies an aggregation along the time dimension) It should also be noted that certain shortcuts can be incorporated into the syntax by combining functionalities of the basic operators in Table 1 into higher-level operators (including the incorporation of property definition syntax into the basic operators) We do not discuss here such shortcuts any further

Each operator and its syntax will be explained by examples The examples utilize the following *TSCs*

BOOK_SALES (type = discrete) - contains the daily sales of books (surrogates), the temporal attribute contains the number of books sold, and is named QUANTITY
BOOK_PRICE (type = stepwise constant) - contains the daily prices (temporal attribute named PRICE) of books (surrogates)
BOOK (type = nontemporal) - contains three attributes for each book TYPE (math, computers, etc), AUTHOR-NAME, and DISCOUNT (% discount for QUANTITY>10)
EMP_COMMISSION (type = discrete) - contains the daily commissions (temporal attribute named COMMISSION) of employees (surrogates)
EMP_SALARY (type = stepwise constant) - contains the monthly salaries (temporal attribute named SALARY) of employees (surrogates)

Operator	Target Specification	Mapping	Function
select	predicate conditions over S,T and A	identity	arithmetic operations over attributes or identity
aggregate	implied by mapping	group specification over S or T	aggregation operators (sum, maximum,)
accumulate	identity	sequence specification over T	aggregation operators (sum, maximum,)
restrict	surrogate restriction by auxiliary TSC	identity	identity
composition	identity	corresponding points of source TSCs	arithmetic or aggregation operators
general	user defined	user defined	user defined

Table 1 Classification of temporal operators

SELECTION

The selection operator extracts parts of a *TSC* that satisfy a predicate referencing *s* and/or *t* and/or *a* values The target specification part determines the points (s,t) of the source *TSC* that will appear in the target *TSC* The mapping part is identity in this case, while the function can be either an identity or a manipulation of the *a*-values By default, the target *TSC* inherits all the properties of the source *TSC*, except the lifespan which may be changed as a result of the temporal clause

The predicates over the *s* and *a* values are the usual predicates found in query languages However, the predicates over *t* values have additional features that refer to sequences of the temporal values Selection over *t* values can be made by specifying intervals or sequences (in addition to the usual predicates) Intervals are specified by start and end points Sequences are specified by giving the number of points desired from a reference point The sequence points can be specified looking forward (using NEXT) or backwards (using LAST) There is a distinction made between requesting time points or data points (i e only time points that have data values) The predicates T-NEXT, T-LAST, V-NEXT, V-LAST are used for this purpose When the reference point of a sequence is specified as ordinal number, it refers to data points or time points according to the predicate (T-LAST, T-NEXT, etc) used

The time points can be specified as calendar values (in the same units of the *TSC*, or as ordinal values referring to their position in the *TS* They can also assume the values BEGIN and END to specify the beginning and end of the *TS* For example, an interval can be specified as (1/1/86 TO END) We give below several SELECT queries that illustrate some of the above predicates

Example 1
"Get the January sales figures for books #5 and #9 "

```
SELECT INTO JAN_SALES  QUANTITY
FROM BOOK_SALES
WHERE S IN (5,9) AND T IN (1/1/86 TO 1/31/86)
```

The example illustrated one possible specification of the time points In examples 2 and 3 below, assume that the salary *TS* of employee #10 is {(4/85,24000), (6/85,25000), (8/85,28000), (9/85,30000), (11/85,33000), (1/86,36000)}

Example 2
"Get the 4 salary values preceding January 1986 for employee #10 " Note that we want the last four distinct salary values, not the salary for the last four months Thus, the predicate V-LAST is used

```
SELECT INTO EMP10_SALARY SALARY
FROM EMP_SALARY
WHERE S=10 AND T IN (V-LAST 4 FROM 1/86)
```

The resulting time and salaries are {(6/85,25000), (8/85,28000), (9/85,30000), (11/85,33000)} That is, the values of the 4 *TS* data points preceding 1/86 are retrieved **Example 3**
"Get the salary values of employee #10 in the 4 months preceding January 1986 "

This query is the same as in Example 2, except that we use the predicate T-LAST instead of V-LAST In this case, the resulting time and salaries are {((9/85,30000), ((10/85,30000), (11/85,33000), (12/85,33000)} Note that in this case, the values for October and December were interpolated using the interpolation rule associated with the stepwise-constant type property

AGGREGATION

The aggregation operator can be applied over groups in the time dimension or the surrogate dimension For the time dimension, the target specification part determines the new time points of the target *TSC* In many cases, time aggregations are for calendar time where the new time points of the target *TSC* will be of granularity higher than that of the source *TSC* For the surrogate dimension, the target specification part will determine new surrogate values, these values are usually *a*-values from another *TSC* For example, books may be aggregated by type, and therefore the new surrogate values will be the TYPE values For all cases of aggregation, each point in the target *TSC* is mapped to a set of points in the source *TSC* (note that in the case of aggregation these sets of points are disjoint) The function to be applied to each set of mapped points can be any aggregate that generates a single-valued output (such as sum, average, count, etc) By default, the target *TSC* inherits all the properties of the source *TSC*, except the time granularity which may be changed, and the type which is changed to discrete

Example 4
"Sum the book sales by month "

```
AGGREGATE INTO MONTHLY_SALES
   SUM QUANTITY
FROM BOOK_SALES
GROUP T BY MONTH
```

Note that the time hierarchy and its time-unit keywords are known to the system
Example 5
"Sum the daily book sales by type "

```
AGGREGATE INTO TYPE_SALES
   SUM QUANTITY
FROM BOOK_SALES
GROUP S BY BOOK TYPE
```

In this case, the grouping information (namely, the type of book) is obtained from another *TSC*

ACCUMULATION

The accumulation operator is carried out along the time dimension Its purpose is to obtain a new value for each data point based on a sequence of values preceding or following it in the *TS* For example, getting the balance of an account from the *TS* of deposits and withdrawals involves a SUM accumulation from the beginning of the *TS*

The target specification part for this operator is "identity", i e the points of the target *TSC* are the same as the points of the source *TSC* (which also implies that there is no change in the time granularity) Each point of the target *TSC* is mapped to a set of points in the source *TSC* Unlike the aggregation operation, the sets of mapped points are not disjoint The function part may be any aggregation function The default properties of the target *TSC* are the same as for aggregation

Example 6
Assume that the temporal values of a *TS* are 6,4,7,3,5,4 The following illustrates typical mappings and the results of applying the SUM function to them

GROUP TO BEGIN, the result is 6,10,17,20,25,29 Each value in the source sequence was replaced by the sum of itself and all the values preceding it
GROUP TO END, the result is 29,23,19,12,9,4 Each value in the source sequence was replaced by the sum of itself and all the values following it
GROUP TO V-LAST 2, the result is 6,10,11,10,8,9 Each value in the source sequence was replaced by the sum of itself and the value preceding it
GROUP TO V-NEXT 2, the result is 10,11,10,8,9,4 Each value in the source sequence was replaced by the sum of itself and the value succeeding it The usage of T-LAST and T-NEXT is the same as in Example 3

Example 7
This example calculates a series of moving averages useful in forecasting applications
"Get a series of 7-day moving averages of book sales "

```
ACCUMULATE INTO AVG_SALES
   AVG QUANTITY
FROM BOOK_SALES
GROUP TO T-LAST 7
```

RESTRICTION

A restriction operator involves a target *TSC*, a source *TSC* and an auxiliary *TSC* (following the BY keyword) This operation is similar to the semi-join operation in the relational model Its purpose is to select only those surrogates of the source *TSC* that also appear in the auxiliary *TSC* For example, suppose that we want to look at sales records of mathematics books only Since the type of book is in a *TSC* other than the sales *TSC*, a restriction operator is needed

Both the source and the auxiliary *TSC* must have the same surrogate type The target specification part qualifies time points of the source *TSC* only for those surrogates that appear in the auxiliary *TSC* as explained above The auxiliary *TSC* can have predicate conditions applied to its attributes in order to select the surrogates of interest The mapping and the function parts of the operation are identity The target *TSC* inherits all the properties of the source *TSC*

Example 8
"Get the salary history of employees who are paid a commission greater than 1000 "

```
RESTRICT INTO COM_EMP
FROM EMP_SALARY
BY EMP_COMMISSION COMMISSION > 1000
```

COMPOSITION

The composition operator enables manipulation of related data that are part of two *TSCs* We distinguish between three types of compositions pairwise, by surrogate, and by time In the case of pairwise composition (this is the default when the 'BY comp-method' clause is not specified), the source *TSCs* must have the same surrogates, time granularity, and lifespan In this case, a function is applied to each corresponding pair of points (one from each *TSC*) to produce a single value in the target *TSC* The target specification part is identity (with respect to either of the source *TSCs*) A target point is mapped to two points, each of which is an identity mapping to one of the source *TSCs* The function part manipulates each pair of source a-values (values are interpolated when necessary) to produce a single target value The following example illustrates a pairwise composition

Example 9
"Get the daily book revenues (assume no discounts) "

```
COMPOSE INTO BOOK_REVENUE
   REVENUE=QUANTITY×PRICE
FROM BOOK_SALES,BOOK_PRICE
```

The default properties of the target *TSC* in pairwise composition are as follows The time granularity and lifespan properties of the target *TSC* remain the same as those of the source *TSCs* If both source *TSCs* are regular so is the target *TSC*, otherwise, the target *TSC* is irregular The type of the target *TSC* is discrete, if either of the source *TSCs* is discrete, stepwise constant, if both of the source *TSCs* are stepwise constant, and continuous, if both source *TSCs* are continuous or one is continuous and the other is stepwise constant

The more general case of composition is between the corresponding values of multiple *TSCs* We chose to show here the case of two *TSCs* only since it simplifies its presentation Clearly, we can get the same effect by multiple applications of the pairwise compositions

When composition is done by surrogate, the first source *TSC* must be the *TS* of a single surrogate with time granularity and lifespan the same as those of the second source *TSC* This single surrogate row (when viewed in the 2-dimensional space of Figure 2) is applied to each of the surrogate rows of the second *TSC* The target specification part is an identity with

respect to the second source *TSC* Each target point is mapped to two points, the first point is the identity point in the second source *TSC*, and the other point is the point in the first *TSC* having the same time value as the mapped point in the second *TSC* The function part and the default properties of the target *TSC* are the same as in the case of pairwise comparison

In the case of composition by time, the first source *TSC* has a single time point An example is a non-temporal *TSC* with a single attribute This operator is similar to the composition by surrogate, except that a time column is applied to each of the columns of the second source *TSC* The target specification part is an identity with respect to the second source *TSC* Each target point is mapped to two points, the first point is the identity point in the second source *TSC*, and the other point is the point in the first *TSC* having the same surrogate value as the mapped point in the second *TSC* The function part is the same as in the case of pairwise comparison The properties of the target *TSC* are the same as those of the second source *TSC* The next example illustrates composition by time

Example 10
"Get the daily book revenues for discounted sales only" Since discounts exist only when the quantity is greater than 10, we first have to eliminate smaller quantities (these values are replaced by nulls) This is done with the SELECT statement below Then, we have to find the discounted price for each book However, the price of the books changes over time while there is only a single discount value for each book (in the BOOK *TSC*) Thus, we need to apply the column DISCOUNT to the BOOK PRICE *TSC* This is done in the first composition (by time) The second composition (pairwise) creates the desired revenue *TSC*

```
SELECT INTO DISCOUNT_QUANTITY
   D_QTY=QUANTITY
FROM BOOK_SALES
WHERE QUANTITY > 10

COMPOSE INTO DISCOUNT_PRICE
   D_PRICE=(1-(DISCOUNT/100))×PRICE
FROM BOOK,BOOK_PRICE
BY T

COMPOSE INTO DISCOUNT_REVENUE
   D_REVENUE=D_QTY×D_PRICE
FROM DISCOUNT_QUANTITY,DISCOUNT_PRICE
```

4 3 An example of user defined operators

The following is a fairly complex example that is taken from a real application We go through it in some detail to illustrate how it can be handled with user defined operators over *TSCs*

Consider the measurements that result from a typical high energy physics experiment Such experiments often use several kinds of detectors to determine (eventually) the trajectory, energy, velocity, etc of particles We restrict our attention to two kinds of measurements electric voltage that corresponds to the position of the particle relative to the detector, and the magnetic field measured at regular intervals

Suppose that the above measurements are organized as *TSCs* where the surrogates represent the different detectors There is one *TSC* for electric measurements and one *TSC* for magnetic measurements The goal is to correct the electric measurements according to the magnetic fields To achieve this we first have to find the magnetic fields at the positions of the electric detectors, and then use these values to compute the corrected electric values

Let us consider first computing the magnetic fields at the position of the electric detectors A *general* operator can be defined as follows The target specification part specifies that the surrogates of the target *TSC* are the same as those of the electric detector *TSC* The temporal points stay the same as the magnetic field *TSC* The mapping part specifies for each electric detector position (of the target *TSC*) the relevant magnetic detectors that will be used for computing the magnetic values This is done by a user provided program that uses the coordinates of the detectors (for example, finding a set of near neighbors) Finally, the function part specifies the computation to be used to generate the target values (for example, a weighted average) Here, again, the function is provided by the user

Next, the operator for correcting the electric field values is specified It is a fairly simple *composition* operator over the electric *TSC* and the magnetic *TSC* computed in the previous step The function part is a user defined program that computes the corrections Note that the two *TSCs* may generally have different time points, since the measurements of the magnetic field are taken at regular intervals, while the electric fields are taken only when induced by passing particles However, this is taken care of automatically by the system, since the magnetic field *TSC* has the type "continuous", and the values are interpolated by a system supplied interpolation routine As noted in section 3, the interpolation routine can also be user defined if necessary

5 REPRESENTING THE TDM IN THE RELATIONAL MODEL

We indicated in the introduction that our goal is to develop a temporal model which is independent of any specific traditional data model Once such a goal is achieved, we can examine what is required to represent the structures and operators we have developed in existing logical models We examine here the effects of representing the TDM in the relational model The discussion below is only an outline of such a representation, detailed specifications are the subject of current research

First, we need to represent *TSCs* At first glance this could be simply achieved by defining a relation

with three columns for S, T, and A However, a *TSC* has several semantic properties that need to be maintained by the system, i e , granularity, lifespan, type, and regularity maintaining these properties is quite important since this is the means by which the system can perform interpolation of data values for missing temporal values and to interpret the operations correctly Thus, we will need a special relation type (a temporal relation) that supports these semantic properties

A temporal relation must also have the semantics of being time ordered, as is a *TSC* While the tuples can be stored in principle in any order there is an implicit time order for certain operations For example, assume that the temporal relation represents salary history by month as a step-wise constant sequence A select operator for the salary in a month that is not stored in the relation would have to order the salary values by month in order to determine the salary in the last appropriate month Other operators, such as accumulation and composition, also use this implicit order

The representation of a *TSC* as a three column relation conforms with the representation of relations as tables However, it implies a repetition of the surrogate values for all the time values that exist This is the reason that some authors have chosen to use non-first-normal form relations For example, in [Clifford & Tansel 85], the authors proposed some variations of relations Clifford suppresses the surrogate repetition and lists the time-value pairs of each temporal attribute in successive tuples Tansel defines a complex data type that contains a list of triples for start-time, end-time, and value While such representations may improve the visualization of a temporal relation, they are not essential to the representation of a *TSC* As long as the system recognizes and supports the logical construct of a temporal relation (with the structural and semantic properties of *TSCs*), it may choose any of several visual representations, such as those shown in Figure 3

Next, we examine the operators on *TSCs* and whether they can be supported by current relational operators The *select* operator can be supported with predicate conditions on relations, except that new predicates will have to be defined for the time domain that refer to ordered values (e g last 10 months), and calendar values (e g every Thursday) The *aggregation* operator could be supported by the aggregate functions of relational languages, except that the language would need to recognize the calendar hierarchy (e g days grouped into months) for the purpose of aggregation

The *accumulation* operator cannot be directly represented with relational operators, since it requires a running accumulation of values Similarly, the *composition* operator that finds matching values (if necessary by interpolation) in the time domain is an operator that has to be added The *restriction* operator which behaves like a semi-join, can probably be defined by a combination of a join and a projection Finally, to accommodate user defined operators, there would have to be a facility where any of the three parts of an operator discussed in section 4 (i e , target specification, mapping, function) can be replaced by a user defined program Also, the system should permit the interpolation function of a temporal relation to be user defined

We believe that many of the constructs for operators over *TSCs* developed in this paper can be readily used with temporal relations This is particularly true for extending SQL since we have used an SQL-like format

6 SUMMARY AND FUTURE WORK

In this paper we have developed a temporal data model which is independent of any specific traditional data model, such as the relational model We believe that this approach is more appropriate than extending existing models to accommodate temporal data, since such models were not designed especially to model temporal semantics

We have defined temporal data structures that support naturally sequences of temporal values We have described the semantic properties of such structures, and developed operators that manipulate them We have demonstrated by means of several examples the capability to represent the semantics of and to manipulate temporal data We have also outlined the requirements to be added to the relational model in order to support the temporal data model presented in this paper Current research addresses the detailed specifications of such requirements We also plan to develop the representation of the temporal data model developed here as part or extensions of existing models (other than the relational)

Other research work includes the extension of the operators for complex *TSCs* to include structures that permit multiple surrogates, as well as multiple time lines, and an investigation of the use of abstract data types methods to define operators on time series

Acknowledgement

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, and by a grant from Institute of Business and Economic Research in the University of California at Berkeley

Appendix

The following notation is used in the syntax below lower-case letters are used for variable names, upper-case letters are used for keywords, | denotes 'or', [ ] is used to designate optional expressions, and * denotes zero or more occurrences The general syntax of an operator is

operator-name INTO target-tsc function
FROM source-tsc [,source-tsc]*
WHERE target-specification
GROUP BY|TO mapping-specification

The Syntax of TDM's Data Retrieval Language

SELECT [INTO target-tsc] att-list
FROM source-tsc
[WHERE select-clause]

AGGREGATE [INTO target-tsc] function
    [UNIQUE] [attribute-name]
FROM source-tsc
GROUP [S|T] BY group-clause

ACCUMULATE [INTO target-tsc] function
    [UNIQUE] [attribute-name]
FROM source-tsc
GROUP TO acc-clause

RESTRICT [INTO target-tsc]
FROM source-tsc
BY aux-tsc att-clause

COMPOSE [INTO target-tsc] c-function
FROM source-tsc,source-tsc
[BY comp-method]

target-tsc = tsc-name
source-tsc = tsc-name
aux-tsc = tsc-name
select-clause = select-element
    | (select-clause boolean-op select-clause)
select-element = surr-clause
    | temp-clause | att-clause
boolean-op = AND | OR
surr-clause = [NOT] surr-predicate
temp-clause = [NOT] temp-predicate
att-clause = [NOT] att-predicate

surr-predicate = S predicate-op value
    | S IN (values)
predicate-op = = | ≠ | > | ≥ | < | ≤
values = value [,value]*
value = integer | real | string

att-predicate = att predicate-op value
    | value predicate-op att | att IN (values)
    | att predicate-op att
att = attribute-name | (att arith-op att)
arith-op = + | - | × | / | **
att-list = [new-name=] att [,[new-name=] att]*

temp-predicate = T predicate-op t-value
    | T IN (t-values)
    | T IN (t-range [,t-range]*)
t-value = t-ordinal | t-calendar
t-ordinal = integer | BEGIN | END
t-calendar = gregorian-value | BEGIN | END
t-values = t-ordinal [,t-ordinal]*
    | t-calendar [,t-calendar]*

t-range = interval | sequence
interval = t-start TO t-end
sequence = sequence-op integer
        FROM reference-time
sequence-op = V-LAST | V-NEXT
    | T-LAST | T-NEXT
reference-time = t-value
t-start = t-value
t-end = t-value

function = AVG | MAX | MIN | SUM | COUNT
group-clause = time-unit | integer
    | tsc-name[attribute-name]
acc-clause = sequence-op integer | BEGIN | END
time-unit = YEAR | MONTH | DAY
    | HOUR | MINUTE | SECOND

c-function = new-name= aggr-expr
    | new-name= arith-expr
aggr-expr = aggr-op [attribute-name,attribute-name]
aggr-op = MAX | MIN | AVE
arith-expr = ((arith-pair) arith-op value)
    | (value arith-op (arith-pair))
arith-pair = c-att arith-op c-att
c-att = att-name | (c-att arith-op value)
    | (value arith-op c-att)
comp-method = S | T

Notes
1) All tokens in the syntax that end with '-name' are strings of characters amd/or digits
2) The token 'gregorian-value' can assume different formats (such as mm/dd/yy for months, days, years) which depend on the application We chose not to specify the formats here

REFERENCES

[Adiba & Quang 86]
Adiba, M , Quang, N B , Historical Multi-Media Databases, *Proceedings of the International Conference on Very Large Data Bases (VLDB),* 1986, pp 63-70

[Ariav et al 84]
Ariav, G , Beller, A , Morgan, H , A Temporal Data Model, Technical Report, New York University, December, 1984

[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

[Gadia 86]
Gadia, S K , Toward a Multihomogeneous Model for a Temporal Database, *Proceedings*

of the International Conference on Data Engineering, 1986, pp 390-397

[Klopproge 81]
Klopproge, M R , TERM An Approach to Include the Time Dimension in the Entity-Relationship Model, *Proceedings of the Second International Conference on E-R Approach,* 1981, pp 477-512

[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 March, 1986

[Rotem & Segev 87]
Rotem, D , Segev, A , Physical design of Temporal Databases, to appear in *Proceedings of the Third International Conference on Data Engineering,* February, 1987

[Segev & Shoshani 86]
Segev, A , Shoshani, A , Modeling Temporal Semantics, Lawrence Berkeley Laboratory Technical Report LBL-22337, October 1986 To appear in TAIS (Temporal Aspects in Information Systems) conference, May 1987

[Shoshani & Kawagoe 86]
Shoshani, A , Kawagoe, K , Temporal Data Management, *Proceedings of the International Conference on Very Large Databases,* August 1986, pp 79-88

[Smith & Smith 77]
Smith, J M , and Smith, D C P , Database Abstractions Aggregation and Generalization, *ACM TODS 2,* 2, June 1977

[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

[Studer 86]
Studer, R , Modeling Time Aspects of Information Systems, *Proceedings of the International Conference on Data Engineering,* 1986, pp 364-373