1987

LOGICAL AND PHYSICAL TEMPORAL-DATABASE DESIGN

Arie Segev
The University of California, Berkeley

LOGICAL AND PHYSICAL TEMPORAL-DATABASE DESIGN

Arie Segev
School of Business Administration
The University of California, Berkeley

ABSTRACT

This paper examines problems and approaches to logical and physical temporal-database design. The logical model is used to determine the functionality required of the physical design. Due to the special nature of temporal data, existing database structures are inadequate. The nature of physical design problems are examined and some solutions proposed. A multi-dimensional file partitioning algorithm is proposed; this algorithm is appropriate for certain temporal-database environments.

1. INTRODUCTION

Recently there has been a significant increase in the amount of research in the area of temporal databases. For a survey of the role of time in information processing, see Bolour et al. (1982), and for a research list see Snodgrass (1986). The importance of incorporating the time dimension into database systems has long been recognized. The relational model (Codd 1970), for example, represents data as a collection of relations, with no support for time-varying attributes. Each relation reflects the most recent snapshot ("current version"), and all past information is either lost or available only through recovery logs or the user's own maintenance. The dramatic decrease in the cost/performance ratio of hardware components as well as the development of new technologies (such as the optical disk, Copeland 1982) have made it feasible to support the time dimension online provided that efficient storage methods and access algorithms are developed.

This paper examines the case of historical temporal databases. Such databases are often characterized by their static nature and the importance of the time domain. 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, this history often 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. (for time-series analysis).

Our approach to modeling temporal information was 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.). This differs 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 (1986), Clifford and Tansel (1985), Gadia (1986), and Lum et al. (1984); examples of works that extend the Entity-Relationship model are Klopproge (1981) and Adiba and Quang (1986). 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. A discussion

of the logical temporal model (Segev and Shoshani 1987) is included here because of its relationship to the functional capabilities required of the physical model.

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 defined the concept of a Time Sequence (Segev and Shoshani 1987; Shoshani and Kawagoe 1986), 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. The semantics of time sequences are described in Section 2. This paper addresses the problem of how to organize time sequences on secondary storage devices. Preliminary ideas on various forms of two-level storage intended to support relational temporal databases were introduced in Ahn (1986). The motivation is that due to the differing access demands on current and historical data, each can be organized and stored in a different manner. The applicability of each schema to attribute and tuple versioning was also discussed. It is our view that, in many instances, a multidimensional partitioning of the file is the appropriate approach to organizing temporal data due to the inherent ordering of the time dimension. In traditional databases, the problem of multidimensional file partitioning (MDFP) arises in many applications where it is required to store files which are indexed by one or more search attributes to disk pages such that the mapping from the key space to the physical address space is order-preserving. Such an order-preserving mapping is important in an environment where the frequency of range queries and sequential access by key values is high.

In a given application environment, data files are classified as static or dynamic depending on the rate of insertion and deletion transactions. In recent years, most of the research has focused on dynamic files, resulting in several file organization methods, e.g., the grid file method (Nievergelt, Hinterberger and Sevcik 1984). Some work has also been done on static MDFP (Merret 1984). For the temporal databases considered in this paper, a static algorithm is more appropriate especially when the database contains historical data for the purpose of analysis and decision making.

In Section 3, we explain the partitioning concept and discuss the parameters relevant to the decision on how to partition the file. Section 4 details a partitioning algorithm. This algorithm is appropriate for totally static data and is independent of a traditional data model. In Section 5, we discuss the physical design issues associated with a generalized temporal database (including an extended relational model). The paper is concluded with a summary in Section 6.

2. A LOGICAL DATA MODEL

In this section we summarize the semantic properties of temporal data and the intuition for the data constructs we have chosen. The logical data model is detailed in Segev and Shoshani (1987).

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 (for example, March of 1986), for some attributes 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 ten months" or the "revenues for every Saturday"). The result 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), (19,8), 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 for which they are specified. 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.

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. The following properties are used to characterize the behavior of a TS:

1. **Type:** determines the interpolation rules for the TS.

2. **Granularity:** determines the granularity of the time points; e.g., days, hours, minutes.

3. **Life Span:** determines the ending points of the TS. We are mainly interested in three cases: fixed start and end times; fixed start with end equal to current time; and end equal to current time with start being a fixed distance from end time (a "moving window").

4. **Regularity:** a regular TS implies that there exists a data point for each underlying time point.

2.2 Time Sequence Collections (TSCs)

A TSC is the collection of TSs for the objects belonging to the same class; for example, the collection of all the price histories of products in the database. A TSC can be described as a triple (S,T,A) where S, T, and A are the surrogate, time and attribute[1] domains, respectively. A TSC can thus be viewed as the collection of all the temporal values of a single attribute for all the surrogates of a class. It is convenient to think of a TSC in a two-dimensional space as shown in Figure 2. In this representation, each row corresponds to a TS for a particular surrogate. The dots represent points where temporal values exist.

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.

3. MULTIDIMENSIONAL PARTITIONINGS AND TEMPORAL DATABASES

We now explain the concept of multidimensional file partitioning (MDFP) and show how it relates to temporal databases. Examples of MDFP schemes are the Grid Files of different types and the file structures described in Merret (1978, 1984) and Merret and Otoo (1982). In all of these methods the possible range of values for each attribute is partitioned into segments; the intersection of these segments define hyper-rectangles or cells. Each logical record is associated with a cell based on the segments to which its attribute values belong. An MDFP structure is a three level hierarchical structure. The first level is a relatively small file with information about the partitioning points and is kept in fast storage. The second level is a directory with a single entry for each cell of the partitioning. This entry contains a pointer to the data page which contains the logical records associated with this cell. The last level is the actual file where the data records are stored. Typically the last two levels are resident on disk because of their size. A query against this structure is answered as follows: Using the attribute values specified by the query, the information kept in fast storage is used to compute the disk address of the relevant directory entry or entries on disk, these in turn contain pointers to the data pages where the logical records which form the answer to the query are stored.

As explained above, all of the logical records associated with the same cell of the partitioning are stored on the same disk page. An overflow occurs when the number of tuples associated with a given cell exceeds the capacity of a disk page. We assume that there is a single common overflow area. This paper deals with the case of two search attributes (one of which is the time dimension), but the analysis can be easily generalized to more than two dimensions. We will refer to an algorithm that determines the partitioning of the attribute space as a "partitioning algorithm." The parameters that determine the best partitioning are the size of the primary data area (in number of pages) -- K; the page capacity (in number of tuples) -- c; the

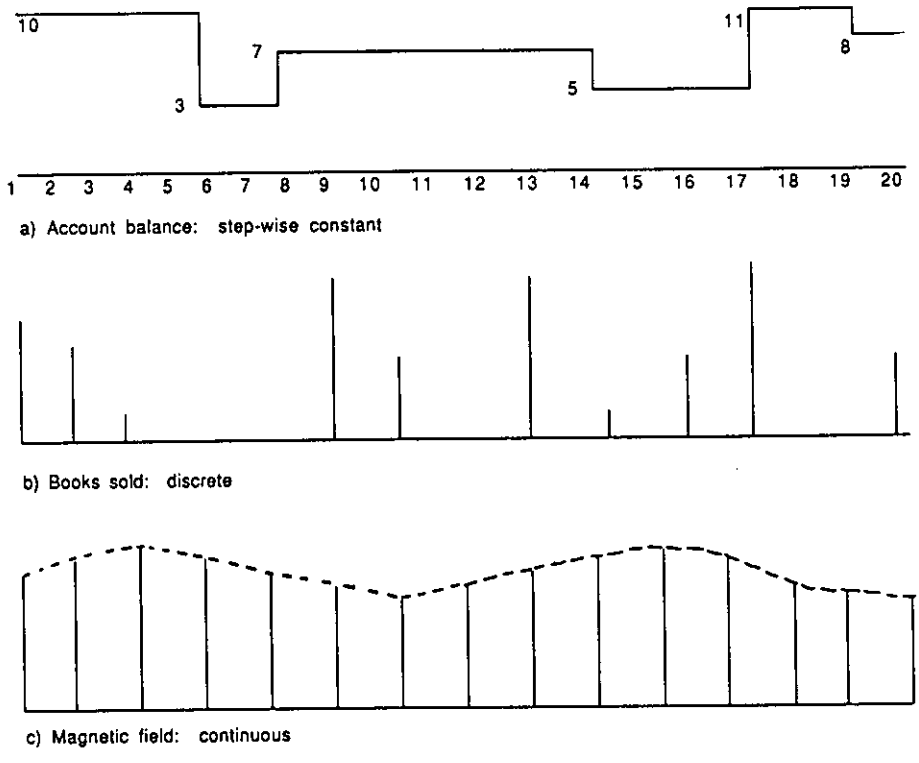

Figure 1. Example of Time Sequences

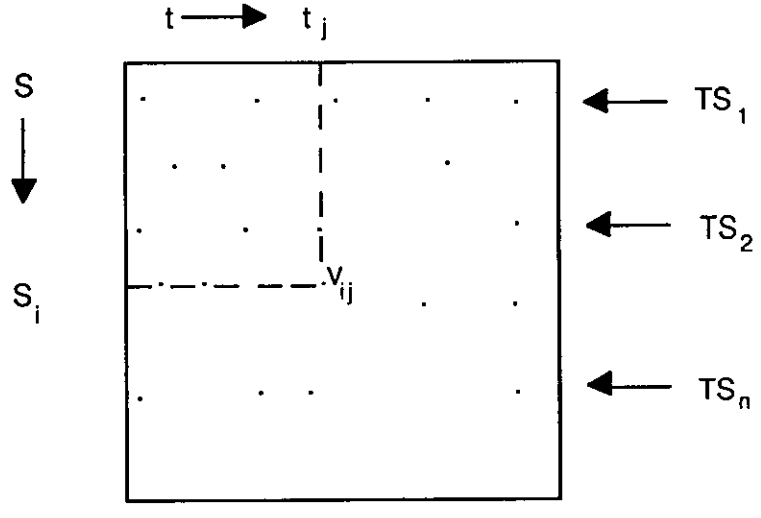

Figure 2. A Two-Dimentional Representation of a Time Sequence Collection

storage utilization (or load factor) -- α; the number of unique values in the search attribute (to simplify the exposition, and without loss of generality, we assume that this number is the same for each search attribute) -- n; and the frequency matrix $F = \{f_{ij}\}$, where f_{ij} is the number of tuples having values i and j in the first and second search attributes, respectively. It should be noted that the matrix F is different from the TSC of Figure 2; in F, we store in position (i,j) the value f_{ij} which represents the number of data tuples with values t_i and s_j in the time and surrogate attributes respectively. For example $f_{2,3} = 5$ means that there are five transactions for surrogate s_3 at time unit t_2. The details of each such transaction is a record which will be kept on a data page. Note that all five records (and possibly others associated with that cell) will reside on the same disk page. That is, the matrix F serves as a basis for the allocation of data tuples to disk pages and the creation of an index to the data file; that is, F is the input to a partitioning algorithm that determines which tuples will fall into the same cell and therefore will reside in the same disk page.

We consider two types of partitioning -- symmetric and asymmetric. Figure 3 presents two examples of a symmetric partitioning. The matrix F may represent daily sales transactions of a group of products. In this case, the first row of F in Figure 3(a) has the following interpretation: the number of transaction records for products 1,2,3,4,5 in day 1 is 1,2,4,0,0 respectively. The second row represents the number of sales records in the second day, and so on. As can be seen from Figure 3, the matrix F

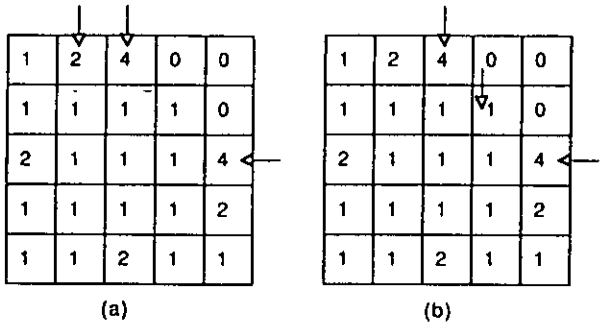

Figure 3. Two Examples of a Symmetric Partitioning with c=3 and K=16. In (a), total overflow is 5 whereas in (b) it is 3. The arrows point to cells where overflow has occurred.

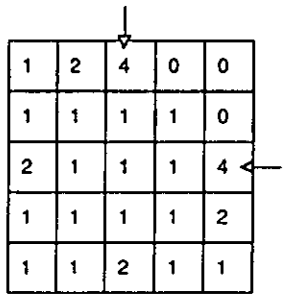

Figure 4. An Example of an Asymetric Partitioning which Further Reduces the Overflow.

is partitioned vertically and horizontally by grid lines. The figure illustrates that the overflow can be reduced by changing the partitioning. Algorithms for asymmetric partitioning are presented in Rotem and Segev (1986). Figure 4 presents an asymmetric partitioning of the same matrix. In this case, we partition F vertically by grid lines, and each vertical segment (the submatrix between two adjacent grid lines) is partitioned horizontally. As can be seen from Figure 4, the asymmetric partitioning helps to further reduce the amount of overflow.

The asymmetric partitioning is appropriate in many instances of temporal data, where the dominant interest is in the attribute of an entity or a group of entities as a function of time, e.g., what were the sales of an item during a certain time period?

4. AN ASYMMETRIC PARTITIONING ALGORITHM

This section describes the asymmetric algorithm for placing the horizontal and vertical lines so that a minimum overflow partitioning is achieved subject to the constraint on the number of pages in the primary area. As mentioned before, an asymmetric partitioning generalizes the symmetric one because it is less constrained in the way in which it partitions the attribute value space. Specifically, in the case of a temporal database we may partition the values of the surrogate into segments representing groups of surrogates and then perform a different horizontal partitioning for each such segment. The horizontal partitioning required for

each segment may contain a different number of cells due to the different activity rates of surrogate groups. For example, in an inventory system some parts will be moved in and out of the warehouse at a much faster rate than others and therefore more transactions will be generated for these parts. This in turn will require allocating more horizontal lines to the segment which contains these parts.

The reason for partitioning the surrogate values into segments (rather than the time values) is that we expect that most queries will specify a surrogate value (or a small range) and try to access the records corresponding to this surrogate within a specified time range or obtain the whole time sequence for that surrogate. We would like to have an efficient data structure with respect to such queries while still supporting queries which specify a time range and then access transactions which refer to all or some subset of the surrogates.

An algorithm for solving this problem must perform the following:

a. Determine the number of vertical lines and their exact placement. This will partition the space into vertical segments.

b. For each vertical segment determined in (a), find how many pages should be allocated to it so that the total number of pages allocated to all segments is equal to the constraint K. Assume that k_i pages are allocated to the i^{th} segment.

c. Find an optimal partitioning for the i^{th} segment into k_i pages. In terms of our previous algorithm, this requires placing k_i-1 horizontal lines in the i^{th} segment.

An algorithm for optimally solving this problem has to inspect a large number of choices for step (a). We devised a heuristics algorithm which takes advantage of some steps of a symmetric partitioning algorithm[2]. We start by summarizing the logic of Algorithm 1. Given a pair of integers n_1 and n_2, they are called permissible factors of K if $n_1 X n_2 = K$ and $n_i \leq n$ for $i = 1,2$. Let P be the set of all permissible factor pairs of K. We can partition the matrix F into K cells by partitioning the rows into n_1 segments and the columns into n_2 segments. This is done by placing n_1-1 horizontal lines between the rows of F and n_2-1 vertical lines between the columns of F. Let $C(F)$ and $R(F)$ be the two vectors obtained from F in the following way: The vector $C(F) = \langle c_1, c_2, \ldots, c_n \rangle$ is obtained from F by setting

$$c_k = \sum_{i=1}^{n} f_{ik} \text{ for } 1 \leq k \leq n,$$

i.e., each component c_k is the sum of the elements of the k^{th} column of F. Similarly,

$$R(F) = \langle r_1, r_2, \ldots, r_n \rangle$$

is the vector obtained by letting r_i be the sum of elements in the i^{th} row of F. Let $\{V,k,c\}$ denote a one dimensional partitioning problem of the vector V into k pages each with capacity c. The same notation will be used for the case of a two dimensional problem except for the following: V will denote an n by n matrix and k will be replaced by two factors. For each pair n_1, n_2 in P, we solve a pair of one dimensional problems $\{R(F),n_1,n_2 X c\}$ and $\{C(F),n_2,n_1 X c\}$. Let g_{n1} and g_{n2} be the optimal solutions obtained for these two problems. By combining the horizontal and vertical line positions computed by the solutions g_{n1} and g_{n2}, we obtain a solution $y(g_{n1},g_{n2})$ to $\{F,n_1,n_2,c\}$.

Algorithm 2 works as follows. It inspects all the permissible factor pairs (n_1,n_2) in P (as in Step 1 of Algorithm 1) and takes the second component n_2 as a value for the number of vertical segments. Once the number of vertical lines is chosen, the exact placement of these lines is determined by solving a one-dimensional partitioning problem on $C(F)$, the column sum vector of the matrix F (an optimal algorithm is given in Rotem and Segev 1986). From now on, let us assume that step a) of Algorithm 1 is completed and we have a fixed vertical partitioning f with n_2 vertical segments.

Let us denote by $FL(i,j)$ the overflow incurred by optimally partitioning the i^{th} segment into j pages. We note that under the fixed partitioning f, finding the value of $FL(i,j)$ is a standard one dimensional dynamic program (Horowitz and Sahni 1978) for every value of i and j. Let $TF(j,l)$ denote the overflow incurred by the optimal partitioning of the first j segments into a total of l pages. We have the following recursive equation:

$$TF(j,l) = \min_{m} (TF(j-1,m) + FL(j,l-m)).$$

This equation expresses the fact that the optimal partitioning of the first j segments into l pages is achieved by choosing some m where $m<l$, and optimally partitioning the first j-l segments into m pages and then partitioning the j^{th} segment into l-m pages. The boundary condition is

$TF(1,l) = FL(1,l)$ for every l.

A dynamic programming approach is used to find the value of m which minimizes the above equation. We have to compute $TF(n_2,K)$ for each n_2 in P and then choose as our optimal solution the vertical partitioning which produces the smallest value of $TF(n_2,K)$.

5. A GENERALIZED TEMPORAL DATABASE DESIGN

There are many parameters that effect the physical design of a temporal database. We consider here three groups of parameters -- the underlying data model; the properties of the TSs; and the access requirements -- and then discuss their effect on the database design.

5.1 The Underlying Data Model

There are three main strategies to mapping the conceptual temporal data model (see Section 2) into an internal architecture: (1) building a temporal interface to a traditional relational database management system, (2) building an interface to a modified relational database management system, and (3) direct implementation of the temporal data model.

In the first strategy, we map the three-dimensional TSC into flat relational tables. This will then permit us to exploit the established internal architecture of the relational model. This approach can be viewed as adding a temporal interface above that of the relational model within the logical level of the ANSI/SPARC architecture. The file, data structures, and access methods will be those used by the traditional database management system. The advantage of this strategy is its relative ease of implementation.

In the second strategy, the temporal data constructs are mapped onto the relational model, but we allow for modifications of the physical relational architecture. The advantage of this approach is in benefiting from the consistency and integrity of abstract relational structures while tailoring the physical organization and design of access methods in order to benefit from the unique nature of historical data.

The third strategy requires a completely new design, where we can adopt the most appropriate logical as well as physical structures for the representation TSCs. The difficulty with this approach is the inability to take advantage of the body of theory available to the relational model and its implementation. On the other hand, we need not restrict ourselves to the tuple based record structure inherent to the relational model.

5.2 Properties of TSs

The properties of the TSs affect the physical design considerations. In particular, we are interested in the life span property. We can characterize it according to t_s, t_e and t_c: the starting and ending points of the TS and the current point in time respectively. Then, the life spans discussed in Section 2 can be stated as: (a) $[t_s,t_e]$ are fixed, (b) t_s is fixed and $t_e = t_c$, and (c) $t_e = t_c$, and $t_s = t_e - k$, where k is some constant.

Four cases of life spans should be considered. The first case is when $t_c>t_e$, and both t_s and t_e are fixed. We have complete information pertaining to the number of data points within the fixed interval. This situation arises when we have reached the end of a type (a) life span, terminate life spans of types (b) and (c), or create TSCs for the sole purpose of historical analysis. The latter is common in scientific and other research work.

The second case is when $t_e>t_c$ and t_s is fixed. Although information on future database activities may be uncertain, the maximum number of data points is equal to the number of time points within the interval. For example, it may be decided on 5/1/87 that the percentage of defective production output should be monitored dating from 1/1/87 until 12/31/87.

In the third case, $t_e = t_c$, which means that the number of data points is indeterminate. Thus the allocation of storage space will be more difficult and there is a higher likelihood of overflows. This life span would correspond to TSCs that model ongoing entities.

The fourth case is a life span of type (c) (a "moving window"). Moving windows dynamically change the range over the stored values corresponding to insertions/deletions of data, and/or time itself, depending upon the way the time granularity was defined. An example is the requirement of maintaining data for continually updated five-year trend analysis.

5.3 Access Requirements

This group of parameters can be classified as follows:

1. Read only: This relates to query retrievals.

2. Read and append: Relevant for append-only databases, where no deletions are allowed. Thus errors are retained and corrections appended as new records. Further, all non-valid historical transactions are also retained.

3. Read, append and delete: Unlike the above, deletions are also allowed. This means that when a record has become invalidated, it can be removed from the database. Nonetheless, all updates (replacement of field value(s) of existing records) are not allowed.

4. Read, append, delete and update: This allows the full range of operations that one finds in a conventional database management system. Note that an update constitutes a delete followed by an insert, except that the new record overwrites the old, unless this is disallowed due to the existence of pinned records.

5.4 Parameter's Effect on the Design

There are three major cases to be considered, corresponding to the underlying data model.

Case One

The major design problem here is the construction of an interface between the temporal and relational data models. There are four major issues involved in the design. The first concerns techniques for the relational representation of temporal data. Aside from considering the efficiency of the mapping algorithms, we need to ensure that the resulting relations maintain all the properties of the original TSCs.

The second issue relates to data operators: will they be mapped onto some relational calculus equivalent, or will they be maintained separately? Mapping them into the relational operators would mean not just modifying or augmenting them, but also the addition of operators such as accumulation and composition (Segev and Shoshani 1987). We then have to guarantee that extended relational calculus is closed. Moreover, the relations must satisfy notions of normal forms that are appropriate for temporal data.

The third factor pertains to query optimization. The use of relational operators allows access to the query optimizer of the database management system. Carrying out the operations directly on the TSCs still requires a way to translate query strategies into access path selection plans for the stored tuples. Otherwise all tuples corresponding to each TSC specified in the query would have to be retrieved.

The fourth issue is performance of the design. With few changes in the file structures and access methods, serious performance problems may result. Moreover, the impact of selected data structures on query retrievals pertaining to current (snapshot) information has to be measured. As far as the other two design criteria are concerned, they have limited applicability.

Case Two

In this case, we can consider both the logical and physical levels of the database design. Thus we will look into the characteristics of temporal data as defined by the life spans and associated operations on them and investigate ways in which they can be organized.

For completed static lifespans, the prime concern is with read only transactions, since data is used mainly for query retrievals. The only appends, deletes or updates that matter are those related to possible errors in recording. These are too irregular to require special consideration. It is likely that TSCs with this life span will contain large amounts of data and have high access frequencies, e.g., scientific or economic data. The partitioning methods discussed in Section 3 and 4 enable data to be retrieved by time or surrogate values, or both, and maintains the ordering along each dimension.

For the incomplete static lifespan, operations of types (2) to (4) are relevant according to the nature of the database and also for insertions of new data. The above partitioning technique could be extended here, where we allocate additional cells to surrogates, to be filled within the interval $[t_c,t_e]$. The rest of the life span can be treated as completed, since probably only read operations are needed. A major consideration is the regularity of the TSC, since it would be much simpler to efficiently allocate empty cells when we can estimate the number of data tuples that would be inserted in the future. In many instances, one can expect a great deal of updates within the incomplete span, since much research oriented work will have this type of span. If the database is append only, then the problem of optimal organization is even more difficult.

In the case of a dynamic life span, while we can also reorganize data along the interval $[t_s,t_c]$ as a static structure, the allocation of appropriate storage and indexing may be more difficult for future data, since the terminal point is unknown. On the other hand, since most ordinary database entities exhibit this behavior and are often predictable or infrequent in their update and query patterns, simple methods such as reverse chaining may be adequate. We may also allow all possible operations on these structures.

As for a moving window, we are able to delete invalidated data, i.e., data where the corresponding time attribute is less than $t_e - k$ or archive them. If data updates are periodic and uniformly distributed, such techniques as stacked versions and cellular chaining (Ahn 1986) may be applicable. Moving windows are regularly used to provide such information as aggregates, thus the ability to have random access may be unimportant.

Case Three

We will briefly discuss the issues associated with a non-relational architecture. Physical design can exploit the advantages inherent in temporal data, and especially TSCs. Thus the physical record need not correspond to a tuple. There are many ways of efficiently compressing and factoring out redundant information, such as repetitive time and surrogate values. In relational based schemes, the tuples have to be reconstructed before further processing, if compressed or partial information has been stored in their place.

Nonetheless, there are great disadvantages to this architecture, since most aspects of database design have to be redefined or assessed carefully. For example, by redefining the basic unit of storage, the validity of conventional concurrency and recovery methods of database management systems must be evaluated.

6. SUMMARY

We have introduced a temporal data model which is independent of any specific traditional data model, such as the relational model. We have defined a logical temporal data structure that supports sequences of temporal values naturally. The TSC structure has to be mapped into a physical structure, and we have detailed a multi-dimensional file partitioning scheme to achieve that. Two alternatives were examined for multidimensional partitioning of temporal databases. It seems that the unique structure of these databases requires different data structures than simple grid files which perform very well for regular databases. The reason for this difference is that there is an inherent asymmetry of the time attribute with respect to the other attributes which requires special treatment. The structures proposed here are static in the sense that all data or at least the distribution of the tuples over the attribute values must be known before a partitioning algorithm can be utilized.

The main objective functions that we were trying to minimize were the total amount of overflow subject to a constraint on the number of pages K. The same techniques can be used for different objective functions. For example, we can minimize K subject to a zero overflow constraint or try to maximize storage utilization. Another important advantage of our scheme is that we can solve the dynamic program subject to constraints on where the partitioning lines must be placed. For example, we can specify that surrogates s_1, s_2 and s_3 must belong to the same segment. The reason for this constraint is that a very common query type specifies all three surrogates. In other words, we can adapt our partitioning scheme to the query pattern as well as the data.

Finally, we have examined the parameters that affect the physical design of a generalized temporal database. The important parameters have to do with the underlying data model, the properties of the time sequences, and the access requirements.

As in the case of traditional database design, there is no one design that can satisfy all requirements, but the given taxonomy can serve as a guide to the right design.

ENDNOTES

[1] This paper deals only with single-attribute TSCs.

[2] We will refer to the symmetric- and asymmetric-partitioning algorithms as Algorithm 1 and Algorithm 2 respectively.

REFERENCES

Adiba, M., and Quang, N. B. "Historical Multi-Media Databases." *Proceedings of the International Conference on Very Large Data Bases*, 1986, pp. 63-70.

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

Ariav, G. "A Temporally Oriented Data Model." *ACM Transactions on Database Systems*, Vol. 11, No. 4, December 1986, pp. 499-527.

Bolour, A.; Anderson, T. L.; Dekeyser, L. J.; and Wong, H. K. T. "The Role of Time in Information Processing: A Survey." *IACM-SIGMOD Record*, Vol. 12, No. 3, 1982, pp. 27-50.

Clifford, J., and 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.

Codd, E. F. "A Relational Model of Data for Large Shared Data Banks." *Communications of the ACM*, Vol. 13, No. 6, June 1970, pp. 377-387.

Copeland, G. "What if Mass Storage Were Free." *IEEE Computer*, July 1982, pp. 27-35.

Gadia, S. K. "Toward a Multihomogeneous Model for a Temporal Database." *Proceedings of the International Conference on Data Engineering*, 1986, pp. 390-397.

Horowitz, E., and Sahni, S. *Fundamentals of Computer Algorithms*. Computer Science Press, Inc., Rockville, MD, 1978.

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, V.; Dadam, P.; Erbe, R.; Guenauer, J.; Pistor, P.; Walch, G.; Werner, H.; and 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.

Merret, T. H. "Multidimensional Paging for Efficient Database Querying." *ICMOD 1978*, pp. 277-290.

Merret, T. H. *Relational Information Systems*. Reston Publishing Company, 1984.

Merret, T. H., and Otoo, E. J. "Dynamic Multipaging: A Storage Structure for Large Shared Data Banks." In P. Scheurmann (ed.), *Improving Database Usability and Responsiveness*, Jerusalem, June 1982, Academic Press, New York, pp. 237-256.

Nievergelt, J.; Hinterberger, H.; and Sevcik, K. C. "The Grid File: An Adaptable, Symmetric Multikey File Structure." *ACM TODS*, Vol. 9, No. 1, March 1984, pp. 38-71.

Rotem, D., and Segev, A., "Algorithms for Multidimensional File Partitioning." *IEEE Transactions on Software Engineering*, to appear.

Segev, A., and Shoshani, A. "Logical Modeling of Temporal Data." *Proceedings of SIGMOD 1987*, to appear.

Shoshani, A., and Kawagoe, K. "Temporal Data Management." *Proceedings of the International Conference on Very Large Data Bases*, Kyoto, Japan, 1986, pp. 79-88.

Snodgrass, R. (Ed.) "Research Concerning Time in Databases Project Summaries." *ACM-SIGMOD Record*, Vol. 15, No. 4, December 1986.