HISTORICAL DATABASES - IT'S ABOUT TIME!

by

James Clifford

Information Systems Area
Graduate School of Business Administration
New York University
90 Trinity Place
New York, N.Y. 10006

November 1987

Center for Research on Information Systems
Information Systems Area
Graduate School of Business Administration
New York University

Working Paper Series

CRIS #168
GBA #87-114

Abstract

Much recent research in the database community focuses on ways to expand the modelling capabilities of database management systems. The driving force behind this research is the growing size and sophistication of the user community, whose needs and applications seem to always be several steps ahead of the technology. One of the areas where considerable progress has been made in this regard is in the extension of existing data models to represent and manage the temporal dimension of data. In this paper we examine how these enhanced modelling capabilities will increase the *functionality* of the database management systems of tomorrow. We also introduce the notion of *Temporal Representation Transparency* as the appropriate abstraction mechanism for providing this increased functionality with minimum burden to the user.

1. Introduction

Information has become a primary resource for many large organizations, and the Database Management System (DBMS) has become, in turn, a central tool for managing this critical resource. The earliest DBMS's, based on network or hierarchical models of information organization, emphasized the physical level of access mechanisms and file design. Unfortunately these systems, as products of their time, paid little attention to the user interface. The Relational Model of Data, first defined in 1970 [1], was proposed as a model which would liberate the end user from the details of the computer implementation of files and access paths, allowing the user instead to concentrate on *logical* interconnections among the information by means of the simple interface of data stored in tables.

The Relational Model represented a significant advance over the first generation of DBMS's. While early criticisms of RM focussed on the inefficiencies, real or perceived, in the first implementations of relational systems, today such systems as as IBM's DB2 and SQL/DS and Relational Technology's INGRES in the mainframe environment, and DBASE-III and PARADOX in the microcomputer environment, are currently enjoying widespread and growing success. Current criticisms of RM, and indeed of all of the current generation of DBMS's, are directed more at the underlying models on which they are based. These models are now seen as inherently too weak to support the increasingly sophisticated needs of database users.

A major shortcoming of all of the so-called "three great data models" – network, hierarchical, and relational – is their ignorance of *any* notion of time and its relationship to information. None of these models has any built-in notion of time, and as a result the user either is encouraged to think of the database as a repository of only "recent" information, or else is forced to develop *ad-hoc* techniques for managing information across time. These techniques invariably involve some combination of the most recent versions being managed in some fashion in the disk database, and earlier versions being stored in archival tapes. Two problems with this are the need to design special-purpose application software to handle the temporal scheme adopted in the disk database, and the managerial headache of coordinating the disk and tape databases. Clearly many applications, from traditional employee record-keeping, inventory control, and accounts payable and receivable, to more sophisticated computer-aided design, decision support and expert system applications, could all benefit from a DBMS with built-in features for organizing information across time and for accessing that information in time-dependent ways.

1.1. The Research Perspective

Much recent research, for example [2], [3], [4], [5], [6], [7] and [8], has addressed this very issue, focusing in particular on the design of *general*, application-independent modelling structures and operations for providing temporal information management within the DBMS. A survey done in

1982 [9] found nearly 100 articles on the subject, and well over 100 more have appeared since then. Many recent database conferences have devoted an entire session to the issue of temporal data management, and a recent conference [10] was entirely devoted to the subject.

The research in the area varies across a wide range of styles, applications, and methodologies. At the recent conference on Temporal Aspects of Information Systems [10] the research areas presented and discussed included the following cross-section of topics:

- Relational Data Model Extensions
- Non-Relational Data Model Extensions
- Conceptual Data Modelling
- Object Oriented Models and Time
- Temporal Integrity Constraints
- Information System Specification
- Temporal User Interfaces
- Implementation and Access Methods
- Applications
  - Office Automation
  - Medicine

  - Engineering
  - Software Engineering
  - Law
  - Scientific
  - Manufacturing
- Artificial Intelligence

With so much interest, it can be expected that in the near future we will see both historical extensions to existing Relational DBMS's, as well as entirely new Historical Relational DBMS's. In this article we present an overview of the issues involved in historical database management, with particular emphasis on its practical implications. Because of its growing importance, not only by virtue of its role as the vehicle for most new research developments in the database field, but also because of its position as the leading edge in commercial DBMS, the relational model will be used throughout this paper to illustrate the points being made.

2. A Framework for Historical Data Management

In developing a DBMS to provide a general treatment of the temporal dimension of information, it is useful to consider the "roots" of today's

DBMS's and the goals that motivated their development. Figure 1 illustrates the standard ANSI/SPARC view of the "levels of abstraction" which a DBMS is supposed to provide for the user [11]. The motivation for providing these levels is to provide both physical and logical data transparency – that is, to free the end users of the system from having to know details about the physical representation of the data, and about someone else's view of what data is represented. Similarly, when database systems were expanded to provide for distributed database management, it became necessary to handle both fragmented relations and replicated relations properly. A primary goal of the development of distributed DBMSs was to provide location transparency and replication transparency to the end users.

A major goal of historical database should thus be to provide what I have called *Temporal Representation Transparency* (in presentation at [10]), that is, to shield the end users from the details of the temporal representation scheme utilized by the DBMS and also by the designer of the particular database application. The user should not be required to know how the temporal dimension of the data in the HDBMS is modelled or represented. The temporally transparent HDBMS should therefore allow the user to access data with explicit or implicit temporal reference in his or her own view of this dimension. While it is not yet clear exactly how best to provide this transparency, it is clear that a *temporally transparent system* will need to provide the following kinds of features:

- a rich set of well-defined database structures for handling historical

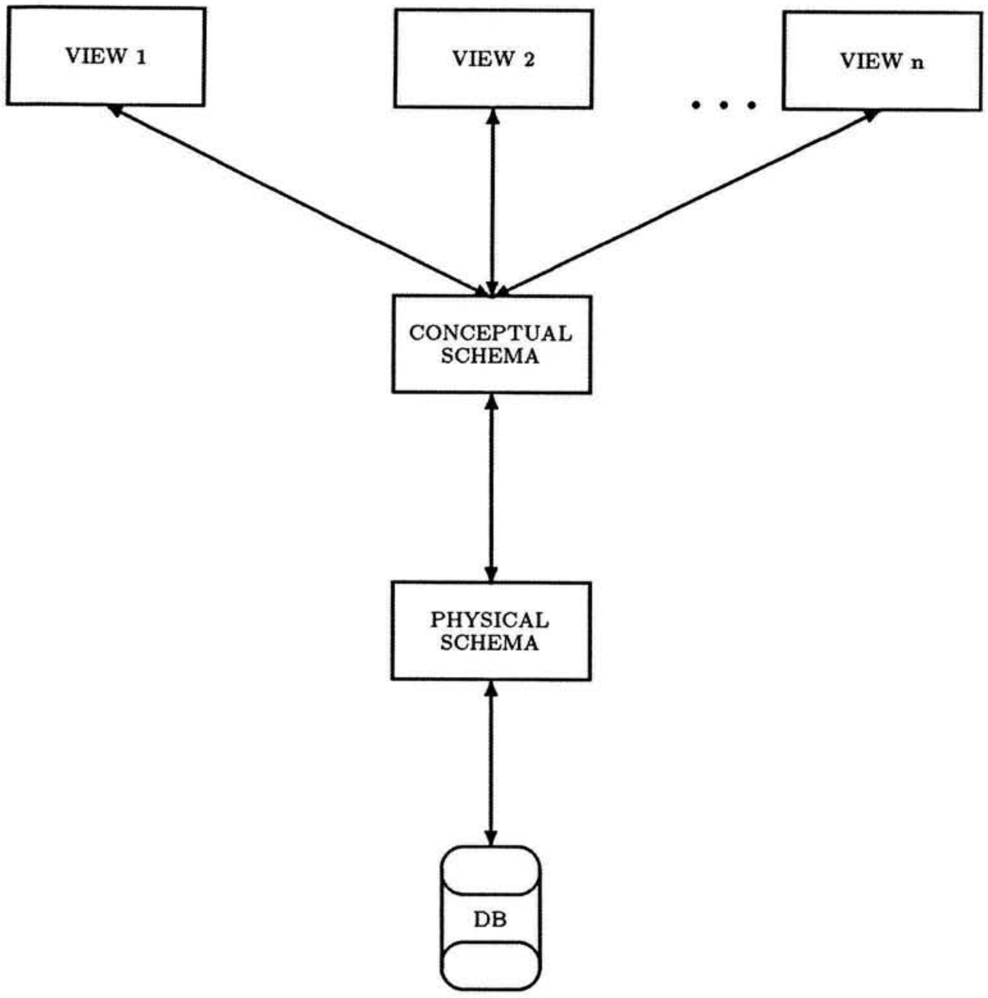


Figure 1: Layers in a DBMS

data

- knowledge about a wide variety of inter-related temporal domains such as seconds, days, and weeks
- the ability to map between different user views of time points, intervals, and sets
- a rich set of operations to access and manipulate both time-series data and the temporal domains themselves

One component of the HDBMS, therefore, must be a subsystem which can translate, whenever possible, between the user's temporal references into the representation scheme used in the database.

An example will help to illustrate the difficulties to be surmounted. Figure 2 shows two time-series values, one for SALARY and one for SALES figures.

Examples of operations a user might want to perform on a database containing these values for the attributes SALARY and SALES-VOLUME are the following:

1. Restrict attention to a particular day, say January 2, 1982
2. Select only those objects with a SALARY value of 30K in February 1983

3. Select only those objects with a SALES-VOLUME value of 600 on January 13, 1982

4. Correlate in time (join together) these SALARY and SALES-VOLUME time-series values

Operation 1 illustrates a major difficulty: neither attribute has a value *explicitly* represented for the date supplied by the user. Operation 2 requires that the system understand something about the standard calendric system, in particular the boundaries and sizes of the months. Operation 3 requires that the system recognize different types of time-value associations, in this instance recognizing that a SALES-VOLUME is an aggregate value over an interval of time, and therefore cannot be associated with a single time point within the interval. Finally operation 4 requires the ability to correlate different types of values associated in different ways with different units of time.

This example should serve to illustrate some of the complexities of adding a general time-handling facility into a DBMS, and particularly of adding them while maintaining Temporal Representation Transparency. In the remaining sections of this paper we will discuss these issues in more detail, and describe several current approaches to handling them.

3. An Example Database

In order to illustrate the issues involved in providing historical data models and database management systems, we will need to utilize a simple database application. Perhaps the best-known database is the familiar world of SUPPLIERs, PARTs and SHIPMENTs; simple as it is, it will suffice for illustrating the need for DBMS facilities to manage the temporal dimension.

3.1. The Conceptual Model

The diagram in Figure 3 shows the conceptual model of the suppliers database as an entity-relationship diagram. A standard, static relational database based on this model is shown in Figure 4.

The standard use of existing database systems models information as it changes over time by keeping only the *latest version* of the data. This version can be queried using a standard query language such as SQL, and can be maintained using the Data Manipulation capabilities of the system, generally including such operations as insert, delete, and replace. In doing so, however, any record of the past state of the enterprise is obliterated. Figure 4, for example, might represent the "current" state of the enterprise's information about PARTs. An SQL query asking about the color of part P3 would be expressed as:

SALARY

$$\begin{array}{l} \begin{array}{lcr} \text{TIME} & & \text{AMOUNT} \end{array} \\ \begin{bmatrix} 1/5/82 & \rightarrow & \$30000 \\ 2/19/83 & \rightarrow & \$31000 \\ 1/7/84 & \rightarrow & \$32000 \end{bmatrix} \end{array}$$

SALES-VOLUME

$$\begin{array}{l} \begin{array}{lcr} \text{TIME} & & \text{QUANTITY} \end{array} \\ \begin{bmatrix} 1/82 & \rightarrow & 300 \\ 2/82 & \rightarrow & 600 \\ 3/82 & \rightarrow & 350 \end{bmatrix} \end{array}$$

Figure 2: Example Time Series Values

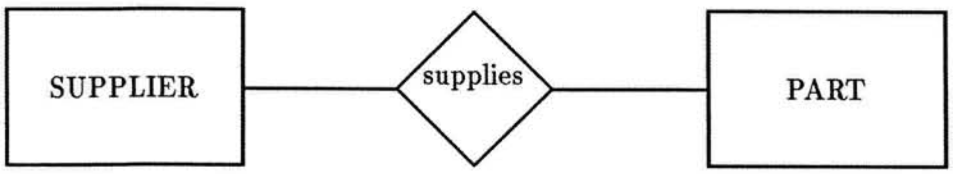


Figure 3: Entity-Relationship Diagram I

SUPPLIER			
SNUM	SNAME	STATUS	CITY
S1	Smith	20	London
S2	Jones	10	Paris
S3	Blake	30	Paris

SUPPLIES		
SNUM	PNUM	QTY
S1	P1	300
S1	P2	200
S1	P3	400
S2	P1	300
S2	P2	400
S2	P2	200

PART				
PNUM	PNAME	COLOR	WEIGHT	CITY
P1	Nut	Red	12	London
P2	Bolt	Green	17	Paris
P3	Screw	Blue	17	Rome
P4	Screw	Red	14	London

Figure 4: Example Static Relational Database

```
SELECT COLOR
FROM PARTS
WHERE PNUM = 'P3'
```

In this static database there is no mention of time; in many applications, of course, this approach would be inadequate. Suppose then, that instead of only keeping track of the latest value of all of this information we are interested in keeping a record of the history of this information over time. When the temporal dimension of this information is considered, even so simple an example as the suppliers database in Figure 3 – surely the most *famous* and by now (one would have thought) most thoroughly understood database conceivable – exhibits shortcomings and ambiguities. The most obvious way to handle these problems within existing database systems is to add an additional attribute, say TIME, to each relation scheme; for example, Figure 5 shows the SUPPLIERs relation in such an extended scheme.

There are several problems with this *ad-hoc* approach to modelling time. Chief among them are that it puts the burden of handling this dimension of the data onto the *user* – either the end user or the applications programmer – rather than allowing the system itself to handle this. For in this approach the DBMS itself knows nothing about the the attribute TIME and so can provide no facilities for interpreting its interactions with the other attributes.

Another problem with this approach is visible in Figure 5. Whereas in the static relational model all of the information about an "object" was located in a single construct, the tuple, with this approach locating all of the information about any given SUPPLIER is a difficult task, since it could be represented anywhere in the relation. For instance, in Figure 5, the information about supplier S2 is located in the third and the fifth tuples. Extrapolating this situation to a relation with hundreds or even thousands of tuples, and with information pertaining to many points in time, it is obvious that this method of incorporating time presents difficulties. In particular, the user must always remember to *sort* each relation on the combined key and TIME fields if this representation is to be meaningful.

The other commonly used *ad hoc* approach to including time is to predetermine, for each attribute, the *maximum* number of time points for which data is to be maintained. Figure 6 illustrates this strategy for the SUPPLIER relation, where the design allows for the maintenance of the three most recent values of the STATUS field. The problems with this approach are similar, stemming from the overall problem that it is the *user*, and not the DBMS, which has the knowledge of the semantics of this representation. In the first place, it is not known to the DBMS which attribute represents the most recent STATUS value. In addition, there is no *explicit* representation of time values, only the relative ordering of these recent three. Finally, such a representation, as seen in Figure 6, requires the use of null values for "objects" which have fewer than the schema-defined *typical* number of values, and in addition to its obvious shortcoming both in requiring the discarding of any values in excess of this number and in requiring the *shifting* of values into prior "positions" whenever a new value is added.

Relationships, such as "supplies" in Figure 3, are somewhat more complicated. Because their existence is more ephemeral, being the "coming together" of entities which already exist in time, their temporality is rich and at this point not thoroughly understood. Such temporal information as when the relationship was "established", when it was "broken off," etc., are crucial to its understanding. Consider again, for a moment, the "supplies" relationship as represented in Figure 4. What exactly does it *mean*? With no reference to its temporal dimension, it is not clear whether the shipments are *actual* or merely *planned*. Had the designer considered the temporal dimension of this relationship, and had an historical DBMS and design methodology been available, something more like the relation in Figure 8 would perhaps have resulted, with *explicit* reference to two temporally anchored properties of the "supplying" relationship.

Obviously such temporal issues are ubiquitous in our processing of information. *Time* is a feature of all information, and so it is appropriate that a general treatment of its structure and properties be placed *into the DBMS itself*, thereby providing a general mechanism for its handling across a diversity of applications. Figure 7 illustrates these two modes of providing the needed temporal data management. In (a) the temporal data is encoded in some *ad hoc* fashion into the underlying static database system, and the temporal semantics and operations are provided by a combination of DML and applications programming. By contrast, in (b) the general temporal information structures, temporal operators, and temporal constraint managers *built into* the HDBMS provide all of these features automatically with no additional programming.

The remaining sections of this paper will discuss the issues involved in providing general-purpose time-handling capabilities in an Historical Database Management System (HDBMS) in the framework illustrated in Figure 7; the example "historical" relation in Figure 8 will be used to provide illustrations. For purposes of reference, the example queries, updates and constraints are described, together with the major issues that each illustrates, in Table Tab1.

3.2. Query and Application Examples

In a database which records information over time, the DBMS must be able to respond to queries and update requests which make explicit or implicit reference to the temporal dimension. Because of the lack of historical data, such operations are not allowed in a static database and, indeed, facilities for handling them are not incorporated into the DML. In this section we illustrate some of the kinds of queries which ought to be handled by the query and constraint enforcement subsystems of an HDBMS.

	Example	Issue
Q1	What was the city address of supplier S2 on 8/25/86?	Time Selection
Q2	What was the status of supplier S2 two years ago?	Relative Time
Q3	When was the city of supplier S3 London?	When question
Q4	What was the status of Supplier S1 when the last shipment of part P3 was made?	Correlation of Time Series
Q5	How many shipments from Supplier S3 were sent more than one week after they were ordered?	Complex Operations
Q6	How has the quantity of shipments of part P5 changed over the past year?	Trend analysis
Q7	Compare the quantity of shipments of parts P5 and P6 over the past year?	Correlated trend analysis
Q8	Assuming a steady 10% growth rate, what will our orders with Supplier S2 look like over the next five years?	Forecasting
U1	Supplier S3 is now located in Amsterdam.	Update as Addition
U2	Part P5 was moved to the warehouse in London at the beginning of last month.	Retroactive change
U3	Supplier S2 will have status 30 starting next week.	Proactive change
U4	Upgrade the status of every supplier who has delivered all orders within 2 weeks.	Selective update
U5	Initiate a standing order of 600 Widgets from Supplier S2 every week.	Time Demons
C1	No supplier can be downgraded in status by more than 10 units.	Dynamic constraint
C2	No supplier can be downgraded in status by more than 10 units in any given month.	Time/value constraint
C3	No supplier can be located in Amsterdam if they were ever located in Berlin.	Multiple time reference
C4	No supplier can be re-instated as a client more than three times.	Event reference

Table 1: Example Queries, Updates and Constraints

Historical query 1: What was the CITY address of supplier S2 on 8/25/86?
Query 1 is perhaps the most *basic* type of historical query an HDBMS should support, namely allowing the user to focus attention on the data as it was relevant at a particular point in time. To answer it the system must be able to perform a kind of temporal "selection" based upon the user-supplied time value.

Historical query 2: What was the status of supplier S2 two years ago?
To handle this type of query, involving *relative* time, the system needs to have knowledge of the standard calendric system and of the kind of temporal values utilized in the STATUS field of the Supplier relation, and be able to perform arithmetic on these values.

Historical query 3: When was the CITY of supplier S3 London?
Query 3 is another basic kind of temporal query. To answer it the system must be able to *decompose* a time-series value and tell the user which times are associated with a given value or values.

Historical query 4: What was the status of Supplier S1 when the last shipment of part P3 was made?
This kind of query involves more than one time-series value. It is a kind of nested query in which the system, as in query 3, must first determine *when* something occurred, and then relate this answer to some other time-series value.

Historical query 5: How many shipments from Supplier S3 were sent more than one week after they were ordered?

This query also involves a combination of operations, including the determination of *when* something happened ("they were ordered"), as well as arithmetic ("one week after"), temporal selection ("shipments" at that time) and counting ("How many") operations.

Historical query 6: How has the quantity of shipments of part P5 changed over the past year?

This kind of "trend analysis" query is very important in Decision Support Systems. To handle it the system should be able to provide graphical output and perhaps statistical analysis of time-series data.

Historical query 7: Compare the quantity of shipments of parts P5 and P6 over the past year?

Similar to Query 6, this trend analysis query requires a correlation of the values of two different time series.

Historical query 8: Assuming a steady 10what will our orders with Supplier S2 look like over the next five years?

One of the major roles of the Decision Support System is to aid in forecasting the future based upon past experience and projected situations. An historical database is ideally suited to this kind of application, and this example represents the kind of "parameterized query" (the "10rate being the parameter") which should be included in the DML of an HDBMS to support these applications.

3.3. Update Examples

In addition to handling more complex queries which contain explicit or implicit reference to the temporal dimension, an HDBMS has an entirely different view of updates from the view of a "static" DBMS. The basic philosophy of the HDBMS must be that all updates to the database, with rare exception, are simply *additions* of more recent information. They are thus treated as insertions of a data value and an associated time. The examples in this section illustrate this point and some of the other nuances associated with updating in an HDBMS environment.

Historical Update 1: Supplier S3 is now located in Amsterdam.
Even Simple updates like this one are handled differently than in static databases, for in an historical database this update must be handled as an insertion rather than a modification. That is, the HDBMS would implement this update by adding a new piece of data into the database, namely a new <time,data-value>.

Historical Update 2: Part P5 was moved to the warehouse in London at the beginning of last month.
This kind of update involves a "retroactive change," i.e., a change made *now* about a fact *in the past.* It might be handled in several different ways, depending on how many time dimensions the database was supporting. If the database had only one time dimension (data time), this operation would cause a *change* to the database, basically rectifying an "error" in

the stored data. If, however, the database were supporting both data time and transaction time, this operation would result in the insertion of a new triple <transaction-time,data-time,value> with a transaction time of *now*, a data-time of the beginning of the month, and a value "London."

Historical Update 3: Supplier S2 will have status 30 starting next week. Similar to Update 2, this kind of "proactive " update in an HDBMS supporting both transaction time and data time allows the user to post information *now* about information which will hold in the real world at some *future* point in time. A typical example of this kind of update would be a "raise" in salary which was decided upon at some time before it was actually to take effect. For budgetary planning and other purposes it might be very useful to record the intended raise before its effective date. As in Update 2, a new <transaction-time,data-time,value> triple is added to the database to implement this transaction.

Historical Update 4: Upgrade the status of every supplier who has delivered all orders within 2 weeks.
This type of "selective" update involves a nested query to select just those "objects" meeting the selection criteria. It then performs the same type of update, a calculation of the new status and the addition of a new <time,value> pair to all of these objects.

Historical Update 5: Initiate a standing order of 600 Widgets from Supplier S2 every week.
Update 5 illustrates a type of update requiring a "time demon" in the

HDBMS whose job it is to accept and store requests for database updates and, with knowledge of the periodicity of time, initiate the desired updates (additions) at the appropriate times until the request is cancelled.

3.4. Constraint Examples

Many of the familiar classes of constraints for the relational model, notably functional and multi-valued dependencies (FDs and MVDs, respectively), appear to have exact counterparts in the various historical relational models which have appeared in the literature. ([2], for example, discusses FDs and MVDs in their historical data model.) However, in a database which records information as it changes over time, the possibility presents itself for *expressing* and *enforcing* constraints more powerful than these familiar constraints of the static relational model.

Constraint 1: No supplier can be downgraded in status by more than 10 units.

With this simple constraint, often called a "dynamic constraint," the user requests that the *new* value be compared with the *old* value and the update be made only if a specified property holds between them (here, that their difference be no more than 10.) This type of constraint is even expressible in "static" models using, for instance, QUEL or SQL and keywords such as OLD and NEW to refer to these two distinguished values.

Constraint 2: No supplier can be downgraded in status by more than 10 units in any given month.

In a slight variation on the previous constraint, Constraint 2 refers not simply to the most recent value in the database, but also to the *time* associated with this value.

Constraint 3: No supplier can be located in Amsterdam if they were *ever* located in Berlin.

Although fanciful for the example application, this type of constraint, representing again only a slight modification to the first example, requires an examination of *all* past values of an attribute, and cannot be handled with the simple OLD and NEW technique.

Constraint 4: No supplier can be re-instated as a client more than three times.

This constraint makes explicit reference to certain application "events" which the HDBMS should be able to record and access through some mechanism for maintaining the "lifespan" of its objects. This constraint then, is satisfied by any employee objects who have no more than three disjoint intervals in their "lifespan."

4. The Nature of Time

One of the major issues involved in designing a data model that will have a built-in notion of time and its properties is, just what is the nature of time and what are its properties? This is a question with a long philosophical tradition, but a somewhat shorter history *vis-a-vis* the database realm. Three basic issues have been studied.

4.1. The Time Elements

An essential issue when considering a model or system for handling the temporal dimension of data is the nature of the time dimension itself. Basically this issue boils down to whether time should be modelled as discrete elements (such as the integers) or as densely packed elements (as the rationals or the reals). While there are proponents of both types of time, it appears that the use of discrete time points is more widespread. A related issue concerns whether to associate data values with points in time or with intervals. Clearly this issue evaporates when using discrete time, since then the two representation schemes are equivalent.

4.2. Relative Time

Most of the research in HDBMS has focused on modelling and managing *absolute time* in some form; that is, exact time points or intervals are associated with each data value. In AI research, however, a great deal of attention has been placed on relative times; for example, in natural language understanding and story understanding systems, where phrases like "last week" and "a year ago" abound. Clearly an HDBMS will have to address this issue in order to provide a *representationally transparent* user interface.

4.3. Periodic Time

The final overall issue with respect to the nature of time, and one that is particularly important in a scheduling context such as might take place in an office automation or job scheduling environment, is that of periodic time. A temporally transparent HDBMS should be able to respond appropriately to queries and updates which refer to the periodicity of the familar time units of weeks, months, etc., and also allow for user-defined periods such as work-weeks, weekends, payroll periods, etc.

5. The Interaction of Time and Information

While at first glance it may appear that the interaction of time and information is straightforward and only needs to be added to existing DBMSs to provide the needed facilities, the considerable body of research into this problem has shown that time interacts with information in a variety of different ways, not all of which are as yet well understood. In this section various aspects of this interaction are discussed and illustrated.

5.1. Attribute Types

There appear to be three primary relationships that can exist between time and the values of an attribute. Some attributes are constant over time, for example NAME and SEX; key attributes are assumed to be of this type. Other attributes are time-varying, taking on different values at different moments in time, such as STATUS or CITY. Finally some attributes take on values which are themselves times, for instance BIRTH-DATE or PROMOTION-DATE.

5.2. Object Identification Across Time

A primary component of any information management system is the need to be able to provide unique and unambiguous identification of the objects being represented; this is the purpose of the *key* in database terminology. In an historical DBMS object identification takes on an added dimension since it is now necessary to recognize that an object currently under examination is in fact the same as some other object already known to the database. Thus there is the need for a time-invariant key – provided either by the user or perhaps by the system – to insure that a single real-world object is represented only once in the database, together with all of its time-varying properties.

5.3. Object Birth and Death

Unlike "static" databases which record information about only the *current* objects of interest, an historical database maintains a more-or-less *permanent* record of objects that are now or were *ever* of interest to an enterprise. Over time, in fact, some objects may come in and out of interest (employees may be hired, leave, and subsequently be re-hired; business with a supplier may be discontinued only to be reinstated at some later date, etc.) These "birth" and "death" events in the real world need to be adequately represented in an HDBMS, for several reasons: to keep track of when objects are *actively* of interest, to be able to identify objects across time, to record multiple "incarnations" of the same object, and to save storage space when objects are inactive.

5.4. Tuple or Attribute Time-Stamping

Another controversy exists within the research community relating to the *level* in the database at which the temporal dimension should be incorporated. In relational terms, this has taken basically two forms: time-stamping at the tuple level or at the attribute level. Tuple time-stamping is somewhat more restrictive, in that it requires all attributes of a tuple to have the same temporal properties and time span. Attribute time-stamping allows for greater flexibility by allowing for locality of changes (to a single value rather than an entire tuple), control at the attribute level over such temporal properties as rate of change over time, function interpolation, and aggregate values, some or all of which can also contribute to greater storage efficiency. Moreover representing attribute values as time-series data is generally thought to be closer to the view of most users.

5.5. Continuous vs. Discrete Changes

Some information changes discretely at a precise moment in time; for example, the STATUS of a supplier might change from 20 to 30 at 9:00am on a particular day. Other information is changing continuously, and is only monitored occasionally, when discrete updates are made to the database. Although these real-world situations are quite different, in both cases they are reflected by discrete state changes in the database. In order to properly represent the real world distinction between these two situations, therefore, the DBMS must know for any attribute which type of changes it is representing.

5.6. Interpolation

There are two instances when some kind of *interpolation* of a time-series function by the system is necessary. One occurs when the system uses a compressed method of storage at the physical level (representing only the points of change); the other occurs when the system models a continuous function with discrete sampling points.

5.7. Aggregation and Disaggregation

Some time-varying attributes represent aggregate values while others do not. Those that do may or may not be disaggregatable. Moreover, for *long past* data it might be convenient for the system to automatically utilize some aggregation function (with a computable inverse or at least acceptable approximate inverse) in the interests of saving storage.)

5.8. Data Time and Transaction Time

A major aspect of the relationship between time and information is that there are multiple such relationships. The two which have received the most attention have been called "data time" – i.e., the *real-world time* at which an attribute took on a particular value – and "transaction time" – i.e., the time when this fact was *recorded in the database.* Some systems in the literature have supported only one of these two times (generally the data time), while others have supported both. In [12] this issue has been generalized to a model supporting n-dimensions of time (or other *indices*).

5.9. Events

In contrast to time-varying properties such as SALARY or ADDRESS, another way in which time interacts with information is to record the occurrence of some "event" which takes place in real time. Rather than a property which can take on different values over the course of time, an event is a "one-shot deal" that has a starting point and an ending point, never to occur again.

5.10. Periodicity and Duration

In many environments it is necessary to have an awareness of the periodicity of time – any application that involves scheduling or the regular performance of certain functions or procedures would find this built-in knowledge useful. In order to support this aspect of time the system would need to have knowledge of the standard calendric periods (hours, days, months, etc.) and perhaps the ability to support user-defined periods (work-week, weekend, etc.). Furthermore, the system would need knowledge of the duration of these units and how to convert between units at any level. Finally, the system should support operations with these units, including activation of transactions when a specified period begins or has elapsed, as well as arithmetic on periods (next week, n days ago,...), in other words, "relative time" should be supported.

5.11. Evolving Schemas

A final aspect of the way in which time interacts with information concerns the relationship between time and the database schema itself. When one considers the two components of the database, the data itself and the metadata (the information *about* the data) it is clear that both of these components may evolve over time. While the subject of schema evolution has received some study in its own right, except for a few researchers (notably [13] and [14]) this aspect of historical data management has received little attention. [14] incorporate both notions, that of the "lifespan" of an object and the "lifespan" of its attributes, into their historical algebra in a uniform manner.

6. Temporal Information Structures

In order to support the temporal dimension of information in a DBMS, certain storage structures must be defined to serve as the repository of this time-stamped data. In the various proposals for historical database models, two primary mechanisms have been utilized. One approach is to add some kind of time-stamping attribute or set of attributes to the schema of a relation and requiring that each tuple have, in addition to its "regular" values, a value or values for the temporal attribute(s). An example of this approach is seen in Figure 5. The other approach is to view the temporal component as a property of each *value* stored in the database. This view leads to a structure such as that depicted in Figure 8. In [12] the latter approach is generalized to allow multiple "indexical" dimensions beyond time (e.g., space, observer, etc.), leading to relations such as that shown in Figure 9.

While these two approaches are the most common, it is not yet clear whether some new approach, perhaps building on the network or hierarchical models, or perhaps upon some entirely new model growing out of the specific needs of historical data management, might in fact prove to be more appropriate.

7. Temporal Operations

In addition to providing structures capable of representing historical information, an HDBMS must provide operators capable of providing the functionality required of an historical database. To date most of the research in this area has focussed on the development of query languages for an HDBMS, and analogous to the work on relational query languages, this research has explored the definition of an extended relational algebra and an extended relational calculus.

7.1. Relational Algebra Extensions

[14], [7], and [15] have all explored extensions to the standard relational algebra for historical relational databases. Because of the different structures of their models, and to some extent the different functionality that they provide, these algebras are somewhat different. Nevertheless they are similar enough that they can be discussed as a group., Essentially all of these extensions provide the standard algebraic operations of Project, Select and Join, extended where necessary to handle the ***historical*** relations

of their model, and then add an additional "Time Slice" operator whose function is to provide a kind of "temporal selection," allowing the user to focus on only temporally relevant sections of the database.

7.2. Relational Calculus Extensions

SQL is rapidly becoming the standard user-level query language for relational databases. [3] therefore investigated extensions to SQL to include the ability to reference the temporal dimension of historical databases in an SQL-like manner.

Analogous to the work on extending SQL, [4] has developed an extension to QUEL, called TQUEL, to provide similar functionality. Finally, there has been some research into incorporating certain standard statistical operators directly into the query language of an historical DBMS [16]. Further research is needed to identify the necessary operators for these and other types of access to historical information.

8. The User Interface to Historical Databases

8.1. Extended Relational Languages

We have already noted that there have been a number of proposals to take standard relational query languages (the algebra, QUEL and SQL) and extend them to provide temporal functionality. Another approach has taken the Query-By-Example mode of access and incorporated a temporal dimension [17] into the graphical query structure and the concept of "example queries" from [18].

8.2. Graphical Interfaces

[3], recognizing that temporally oriented information is richer than static information and, in fact, intrinsically "dynamic" in nature, was the first to explore the use of graphics in the user interaction with historical databases as a means of more effectively conveying this dynamism to the user. Much additional work needs to be done in this area.

9. The Physical Implementation of Historical Databases

9.1. Early Ad-Hoc Systems

Early information systems which handled time-varying data developed their own specific storage and access strategies based upon the specific needs of their application. [19] is a noteworthy example, being perhaps the earliest published account of such a system. Chief among these techniques was an indexed file structure for each attribute, with a primary key of the time field. Undoubtedly many of the techniques developed in these pioneering systems will be adopted by tomorrow's general-purpose Historical DBMSs, as in the case of the migration of early file structures and access methods into today's DBMSs. Additional file structuring techniques await further research, as prototype Historical DBMSs begin to be built.

9.2. Optical Disks

A major obstacle to the development of Historical DBMS has always been the tremendous storage requirements that their underlying philosophy – never throw any data away – implies. Many corporate static databases, storing only the latest values of their attributes, utilize scores of today's magnetic disk packs. Contemplation of the orders of magnitude more storage required by treating such databases as "historical" would be daunting on current technology. The development of optical disk technology presents a technological development analogous to the development of the random access magnetic disk which laid the necessary technological foundation for today's DBMS. [20] is one of the early works to explore new file access techniques appropriate to this new device. Doubtless the perfection and widespread adoption of read/write optical disks will hasten this research and lead to new access methods which will support the development of historical DBMS.

10. Summary and Conclusions

In this paper we have presented the need for a new data model with built-in facilities to handle the ubiquitous temporal dimension of information. We believe that the long neglect of this dimension, occasioned by the lack of the technology to support the huge databases which such a model will engender, will no longer be tolerated by the database user community. The growing body of research in this area, along with the parallel research into so-called Knowledge-Base Management Systems, clearly point to the development of a new generation of more "intelligent" Information Systems, an intelligence that is sure to include general knowledge about time and its inter-relationships with information.

We have argued that the notion of Temporal Representation Transparency, an extension of the ANSI/SPARC DBMS levels, should serve as the framework within which historical database systems and information systems are developed. An HDBMS exhibits Temporal Representation Transparency when it can respond to a user's reference to the data's temporal dimension regardless of the representation scheme adopted by the system. The extent to which such transparency can be provided is still an open question, but it should be the goal of the user interface to provide it.

Finally, we have indicated that research into all aspects of historical data management is progressing rapidly, and there is therefore much reason to be optimistic that the era of database systems which "forget" is drawing to a close.

References

[1] E.F. Codd. A relational model of data for large shared data banks. *Comm. of the ACM*, 13(6):377–387, June 1970.

[2] J. Clifford and D.S. Warren. Formal semantics for time in databases. *ACM Trans. on Database Systems*, 6(2):214–254, June 1983.

[3] G. Ariav. *Preserving the Time Dimension in Information Systems.* Technical Report DS-WP 83-12-06, Decision Sciences Dept., Univ. of Penn., December 1983. (Ph.D. Thesis).

[4] R. Snodgrass. The temporal query language tquel. In *Proceedings of the 3rd ACM SIGACT-SIGMOD Symp. on Principles of Database Systems*, pages 204–212, Waterloo, Ontario, Canada, April 1984.

[5] M.R. Klopprogge. Term: an approach to include the time dimension in the entity-relationship model. In P.P.S. Chen, editor, *Entity-Relationship Approach to Information Modeling and Analysis*, pages 477–512, ER Institute, 1981.

[6] J. Ben-Zvi. *The Time Relational Model.* PhD thesis, Dept. of Computer Science, University of California, Los Angeles, 1982. (Unpublished).

[7] S. K. Gadia and J. Vaishnav. A query language for a homogeneous temporal database. In *Proc. of The Fourth Annual ACM SIGACT-SIGMOD Symposium on Principles of Database Systems*, pages 51–56, 1985.

[8] A. U. Tansel. Adding time dimension to relational model and extending relational algebra. *Information Systems*, 11(4):343–355, 1986.

[9] A. Bolour, T. L. Anderson, L. J. Deketser, and H. K. T Wong. The role of time in information processing: a survey. *ACM SIGMOD Record*, 12(3):28–48, April 1982.

[10] AFCET. *Temporal Aspects in Information Systems*, May 1987.

[11] ANSI/X3/SPARC. Study group on database management systems: interim report. *FDT (Bulletin of ACM SIGFIDET*, 7(2), February 1975.

[12] J Clifford. *Indexical Databases.* Technical Report, Dept. of Information Systems, New York Univ., 1987. (To appear).

[13] J. Shiftan. *Assessing The Temporal Differentiation of Attributes as an Implementation Strategy for a Temporally Oriented Relational DBMS.* PhD thesis, Dept. of Information Systems, New York Univ., December 1986.

[14] J. Clifford and A. Croker. The historical relational data model (hrdm)

and algebra based on lifespans,. In *Proc. Third International Conference on Data Engineering*, IEEE, Los Angeles, February 1987.

[15] A. U. Tansel. An extension of relational algebra to handle time in relational databases. In *ACM-SIGMOD International Conference on Management of Data*, Austin, May 1985.

[16] A. U. Tansel. *A Statistical Interface for Historical Relational Databases.* Technical Report, Bernard Baruch College, City University of New York, February 1986.

[17] A. U. Tansel and M.E. Arkun. *Time-By-Example Query Language for Historical Databases.* Technical Report, Bernard Baruch College, City University of New York, 1986.

[18] M. Zloof. Query by example. In *Proceedings of the AFIPS 1975 Spring Conference*, pages 431–438, AFIPS Press, Arlington, Va, 1975.

[19] G. Wiederhold, J.F. Fries, and S. Weyl. Structured organization of clinical databases. In *Proceedings of the NCC*, pages 479–485, AFIPS Press, Montvale, N.J., 1975.

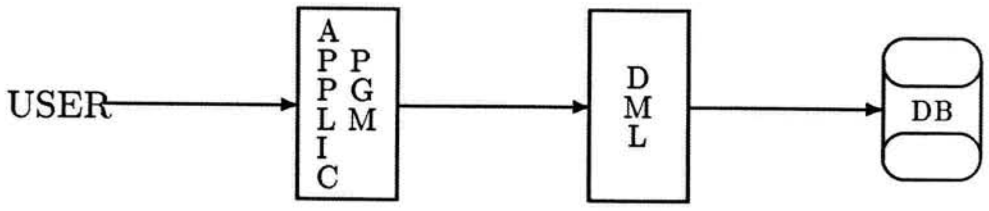
[20] D. Maier. *Using Write-Once Memory for Database Storage.* Technical Report, Dept. of Computer Science, SUNY at Stony Brook, 1982.

TIME	SNUM	SNAME	STATUS	CITY
9/1/87	S1	Smith	20	London
9/3/87	S1	Smith	20	London
8/31/87	S2	Jones	10	Paris
8/05/87	S3	Blake	30	Paris
8/25/87	S2	Jones	10	Paris
8/06/87	S3	Blake	30	Paris

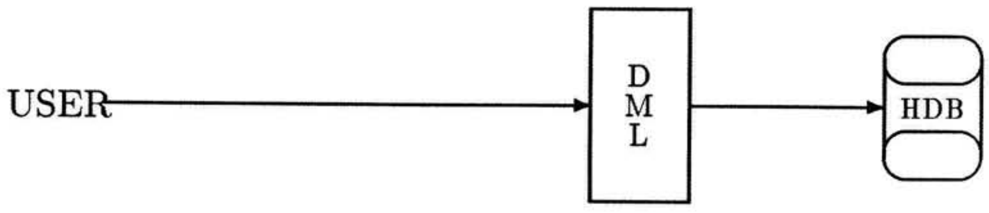
Figure 5: Example Time-Stamped Relation

SNUM	SNAME	STATUS-1	STATUS-2	STATUS-3	CITY
S1	Smith	20	21	23	London
S2	Jones		10	15	Paris
S3	Blake			30	Paris

Figure 6: Example of Pre-determined Slots



(a)



(b)

Figure 7: Ad-Hoc versus HDBMS Approaches

SNUM	PNUM	QTY-ORDERED	QTY-SHIPPED
S1	P1	$\begin{bmatrix} 9/1/87 & \rightarrow & 300 \\ 9/10/87 & \rightarrow & 300 \\ 10/5/87 & \rightarrow & 500 \end{bmatrix}$	$\begin{bmatrix} 9/10/87 & \rightarrow & 300 \\ 9/15/87 & \rightarrow & 300 \\ 10/6/87 & \rightarrow & 500 \end{bmatrix}$
S1	P2	$\begin{bmatrix} 8/31/87 & \rightarrow & 500 \\ 9/10/87 & \rightarrow & 100 \\ 10/5/87 & \rightarrow & 200 \end{bmatrix}$	$\begin{bmatrix} 9/1/87 & \rightarrow & 500 \\ 9/12/87 & \rightarrow & 100 \\ 10/5/87 & \rightarrow & 200 \end{bmatrix}$
S1	P3	$\begin{bmatrix} 8/1/87 & \rightarrow & 200 \\ 9/1/87 & \rightarrow & 100 \\ 10/5/87 & \rightarrow & 500 \end{bmatrix}$	$\begin{bmatrix} 8/5/87 & \rightarrow & 200 \\ 9/10/87 & \rightarrow & 100 \\ 10/10/87 & \rightarrow & 500 \end{bmatrix}$
S2	P1	$\begin{bmatrix} 8/10/87 & \rightarrow & 500 \\ 9/10/87 & \rightarrow & 300 \end{bmatrix}$	$\begin{bmatrix} 8/12/87 & \rightarrow & 500 \\ 9/13/87 & \rightarrow & 300 \end{bmatrix}$
S2	P2	$\begin{bmatrix} 9/1/87 & \rightarrow & 300 \\ 9/10/87 & \rightarrow & 300 \end{bmatrix}$	$\begin{bmatrix} 9/10/87 & \rightarrow & 300 \\ 9/15/87 & \rightarrow & 100 \\ 9/30/87 & \rightarrow & 200 \end{bmatrix}$
S2	P2	$\begin{bmatrix} 8/6/87 & \rightarrow & 400 \\ 9/12/87 & \rightarrow & 50 \\ 10/2/87 & \rightarrow & 600 \end{bmatrix}$	$\begin{bmatrix} 8/9/87 & \rightarrow & 400 \\ 9/16/87 & \rightarrow & 50 \\ 10/20/87 & \rightarrow & 600 \end{bmatrix}$

Figure 8: Example Historical Relation

PROJECT	APPROVER: <Observer, Rec-Time, Data-Time>			
Watergate Break-In	*Halderman* →	*Jan* →	*Dec* →	*Mitchell*
			Jan →	*Mitchell*
			Feb →	*Nixon*
		Feb →	*Dec* →	*Nixon*
			Jan →	*Nixon*
			Feb →	*Nixon*
	Ehrlichman →	*Jan* →	*Dec* →	*Mitchell*
			Jan →	*Mitchell*
			Feb →	*Mitchell*
		Feb →	*Dec* →	*Mitchell*
			Jan →	*Mitchell*
			Feb →	*Mitchell*
Watergate Coverup	*Halderman* →	*Jan* →	*Dec* →	*Mitchell*
			Jan →	*Mitchell*
			Feb →	*Nixon*
		Feb →	*Dec* →	*Nixon*
			Jan →	*Nixon*
			Feb →	*Nixon*
	Ehrlichman →	*Jan* →	*Dec* →	*Mitchell*
			Jan →	*Mitchell*
			Feb →	*Mitchell*
		Feb →	*Dec* →	*Mitchell*
			Jan →	*Mitchell*
			Feb →	*Mitchell*
Ellsberg Break-In	*Halderman* →	*Jan* →	*Dec* →	*Mitchell*
			Jan →	*Mitchell*
			Feb →	*Nixon*
		Feb →	*Dec* →	*Nixon*
			Jan →	*Nixon*
			Feb →	*Nixon*
	Ehrlichman →	*Jan* →	*Dec* →	*Mitchell*
			Jan →	*Mitchell*
			Feb →	*Mitchell*
		Feb →	*Dec* →	*Mitchell*
			Jan →	*Mitchell*
			Feb →	*Mitchell*

Figure 9: Relation With Multiple Indices