THE HISTORICAL RELATIONAL DATA MODEL (HRDM) AND ALGEBRA BASED ON LIFESPANS

by

James Clifford
and
Albert Croker

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

June 1986

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

Working Paper Series

CRIS #116
GBA #86-19

The Historical Relational Data Model (HRDM) and Algebra Based on Lifespans

James Clifford and Albert Croker
Computer Applications and Information Systems
Graduate School of Business Administration
New York University

March 1986; revised June 1986

Table of Contents

Abstract

Critical to the design of an historical database model is the representation of the "existence" of objects across the temporal dimension -- for example, the "birth," "death," or "rebirth" of an individual, or the establishment or dis-establishment of a relationship. The notion of the "lifespan" of a database object is proposed as a simple framework for epressing these concepts. An object's lifespan is simply those periods of time during which the database models the properties of that object. In this paper we propose the historical relational data model (HRDM) and algebra that is based upon lifespans and that views the values of all attributes as functions from time points to simple domains. The model that we obtain is a consistent extension of the relational data model, and provides a simple mechanism for providing both time-varying data and time-varying schemes.

1. Introduction

In a database modelling information over time, the status of an "object" -- is it interesting to the enterprise or is it not? -- will change over time. For example, in a personnel database, throughout the period during which a particular person is employed by a company, information about that person can be assumed to be of interest and so will be recorded in the database. But in general it can be assumed that the database itself will have existed before and will continue to exist after the employment period of any particular employee.

The "birth" of an object O, with respect to a database, refers to the point in time when the database first records any information about O. Similarly, its "death" occurs when the object ceases to be modelled. Historical databases, however, need also to support the notion of "reincarnation," since a death is not necessarily terminal. For example, employees can be hired, fired, and subsequently re-hired; students may drop in and out of school. For this reason the historical database must be able to support object reincarnation, to allow for tracking such reincarnation events as well as the individuals so reincarnated. But database "objects" model not simply individuals (parts, suppliers, students, courses, etc.) but also relationships among individuals (shipments, enrollments). Unlike the standard Entity-Relationship model ([Chen 76]), which allows for only one instance of a given relationship (one part-supplier pair, e.g.), the historical model must model relationships over time, allow for "re-incarnated" relationships, and enforce referential integrity constraints with respect to the temporal dimension. For example, a student can only take a course at time t if both the student and the course exist in the database at time t.

Early work on historical databases, (e.g. [Klopprogge 81, Klopprogge 83, Clifford 83]) recognized this problem and proposed the incorporation of a time-stamp and a Boolean-valued EXISTS? attribute to each tuple as a solution. The database was seen as a three-dimensional cube, wherein at any time t a tuple with EXISTS? = True was considered to be meaningful, otherwise it was to be ignored. As discussed in a classification scheme proposed in [Snodgrass 85], other subsequent and contemporaneous efforts at defining "historical" database models (e.g. [Ben-Zvi 82, Lum 84, Snodgrass 84, Ariav 84]) continued to examine more succinct or perspicuous representations along this tuple-based line. [Clifford 82] was the first to suggest incorporating the temporal dimension at the attribute level. This idea was further refined in [Clifford 85] and was also the basis for the model proposed in [Gadia 85].

These developments can be seen as efforts in the direction of associating the temporal dimension with a smaller component of the model -- at first with the relation itself (the "cube" metaphor), later with each tuple (e.g., the notion of "tuple homogeneity" in [Gadia 85]), and finally with each attribute value. We believe that the orthogonal notions of tuple and attribute lifespans proposed in this paper provide for the suitable level of uniformity and flexibility in the temporal dimension.

2. Lifespans

In order to address these temporal issues (and also, as we shall see, the related issue of evolving database schemas) we introduce the lifespan notion. For instance, the lifespan of an employee, with respect to the personnel database, would *explicitly* represent the temporal dimension of the information about that employee. Queries or other data operations that refer to that employee outside of that lifespan will be treated specially, because the database is not modelling that employee during those time periods.

The question arises, what is an appropriate "object" with which to associate such a lifespan? In particular, data models distinguish between the schema and the instance, and provide constructs of both types. Most attention in historical database research has focussed on the database instance, since in general it is the *data objects* whose lifespans will be of interest. In the relational model the database instance can be looked at as a hierarchy as in Figure 1, the database being composed of a set of relations, and each relation composed of a set of tuples.

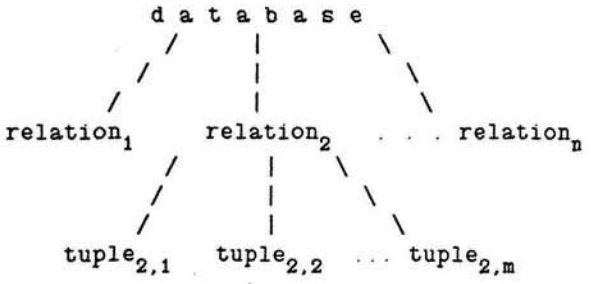

Figure 1: Relational database instance

If we associate a lifespan at the database level in this hierarchy, our database will look like Figure 2, i.e., a collection of relations which are homogeneous in the temporal dimension. (Although in this figure the lifespan is shown as a single, connected interval of time, this is not necessarily the case.) Associating the lifespan at this level commits us to a database in which each relation and each tuple has the same lifespan. Because this is so stringent a constraint, it has not, to our knowledge, been the subject of any serious research.

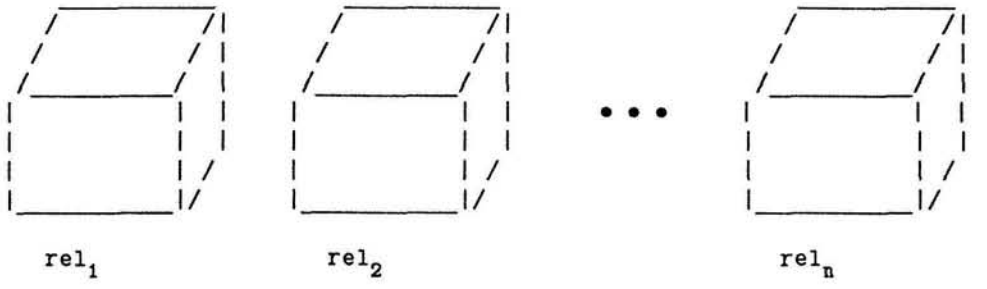

Figure 2: One lifespan associated with entire database

If we instead associate a lifespan with each relation, then we can have a database which looks like Figure 3, where each relation can be defined over different periods of time, but each tuple in a given relation is homogeneous in the temporal dimension, as in [Gadia 85].

Finally, if we associate the lifespan at the tuple level, we have a database that consists of tuples which, for any given relation, can look like those in Figure 4.

The choice of which level is appropriate is a tradeoff between the cost of maintaining proliferating lifespans, on the one hand, and the flexibility that finer and finer lifespans provide, on the other. In terms of complexity, the overhead for the database or relation approach is quite small, and is

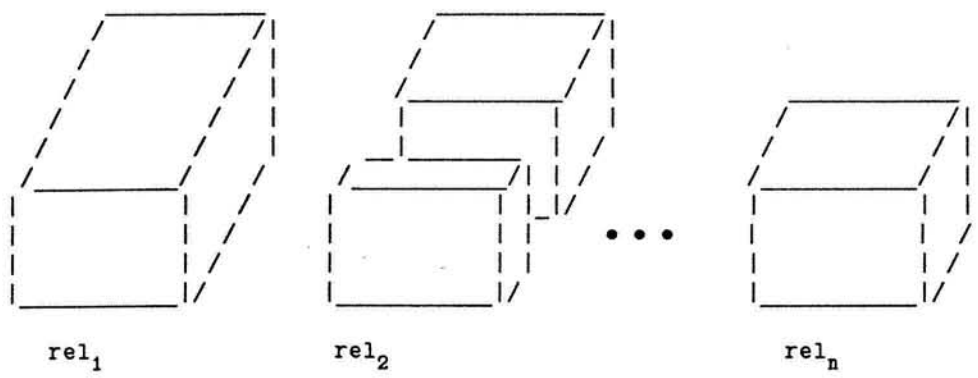

Figure 3: Different lifespans associated with each relation

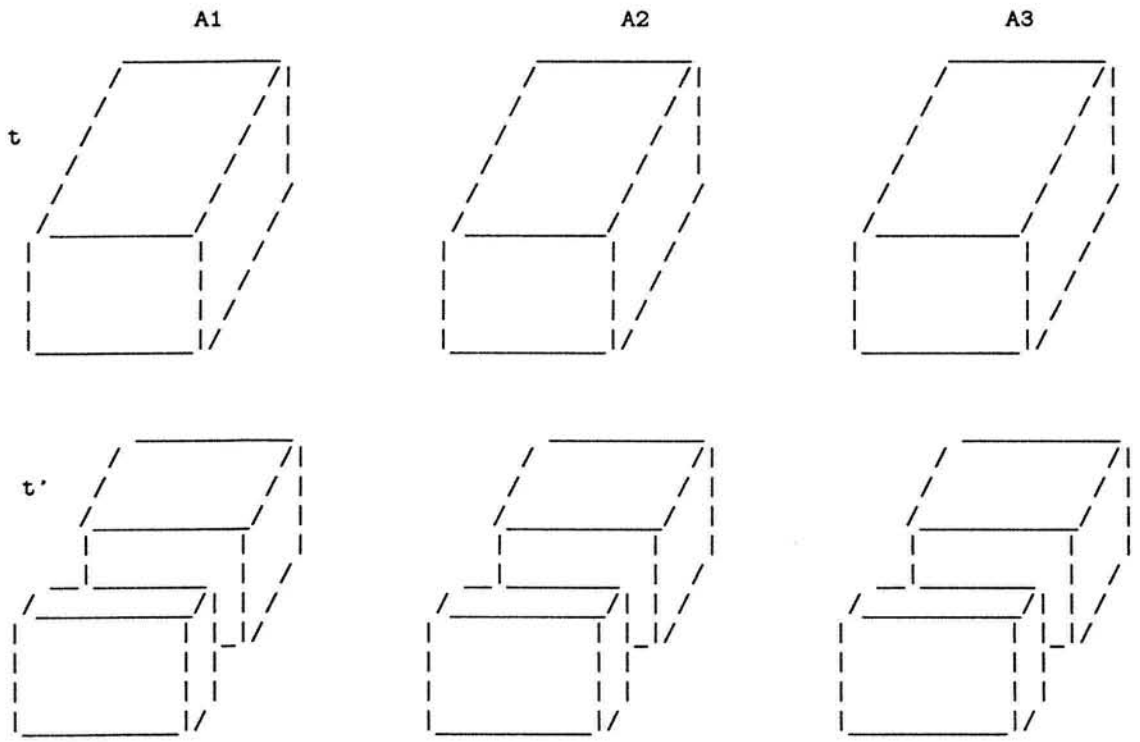

Figure 4: Lifespans associated with each tuple in a relation

proportional to the size of the schema. The cost of the tuple lifespan approach is proportional to the size of the database instance. [Clifford 85] argues that associating the temporal dimension with each attribute provides for more user control of the different temporal properties of individual attributes.

Orthogonal to the database instance and its components is the relational database schema and its components. Some work has been done in considering the schema to be a time-varying component of the database (e.g. [Navathe 80], [Shiftan 85]), but this work has not been done within a single, unified model

for historical databases. The database schema, as illustrated in Figure 5, consists of a set of relation schemas, each of which, ignoring constraints, can be considered to be the set of attributes for that relation.

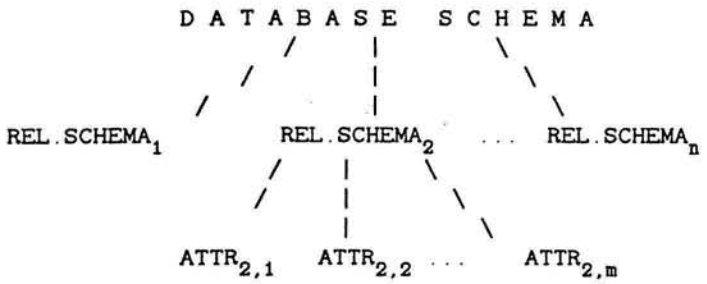

Figure 5: Relational Database Schema

A single lifespan assigned to the database schema (or relation schema) itself would presumably indicate the period of time during which the entire database (or relation) was defined or, in a sense, operational. This does not seem to buy us very much. However, assigning a lifespan to each attribute in a relation scheme, allows the user to explicitly indicate the period of time over which this attribute is defined in that relation, thereby allowing for the possibility of evolving schemes. (Then the lifespan of the relation schema would be the union of the lifespans of all of the attributes in the schema, and we need the constraint that the lifespan of the key attributes must be the same as the lifespan of the entire relation schema.)

As an example, consider a database that records stock market information, including an attribute Daily Trading Volume.

Figure 6: Lifespan of attribute DAILY_TRADING_VOLUME

Its lifespan may be as indicated in Figure 6, where for the period $[t_1,t_2]$ this information was recorded, after which it became too expensive to collect and so it was dropped from the schema. Subsequently, at time t_3 and continuing through the present, a cheap outside source of this information was discovered and so the schema was expanded to once again incorporate this attribute.

The lifespan of an attribute within a given relation is orthogonal to the notion of the lifespan of a tuple in a relation, as shown in Figure 7.

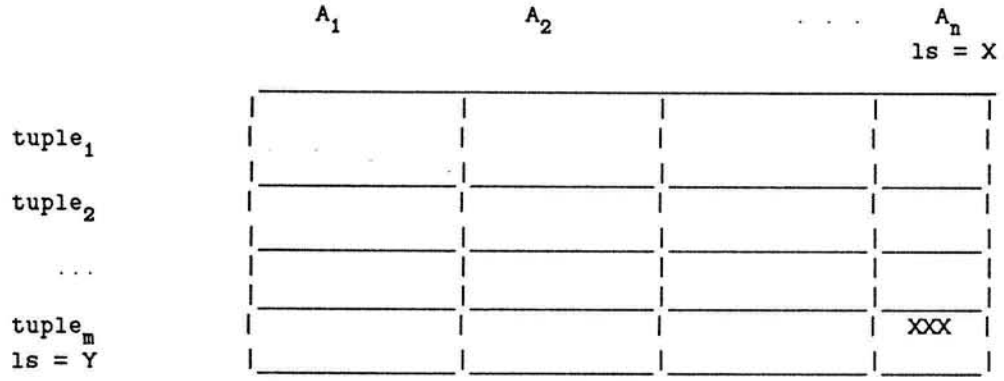

Figure 7: Tuple Lifespan and Attribute Lifespan Interaction

Consider the value of attribute A_n for $tuple_m$, i.e., the XXX'd box in the lower right hand corner of the matrix of Figure 7. Over what period of time is it defined in the database? The tuple provides information about an "object" which is assumed to be defined in the database over the lifespan Y, but the attribute is only defined over the lifespan X. Clearly the "object" can only have a value for this attribute in the database over the intersection $X \cap Y$ of these two lifespans.

Figure 8 represents an example within the model to be presented in the next section; lifespans are associated with tuples and also with attributes, and so tuples are heterogeneous in their temporal dimension. The lifespan of any particular value is limited both by the lifespan of the tuple and the lifespan of the attribute. (It is worth pointing out that in the most general or flexible historical model we would associate a lifespan with each *value* in a relation, and so allow for a completely heterogeneous temporal dimension, but at the cost of maintaining a distinct lifespan for each value.)

3. Historical Relations in HRDM

Let $T = \{ ..., t_0, t_1, ... \}$ be a set of times, at most countably infinite, over which is defined the linear (total) order $<_T$, where $t_i <_T t_j$ means t_i occurs before (is earlier than) t_j. (For the sake of clarity we will assume that $t_i <_T t_j$ if and only if $i < j$.) The set T is used as the basis for incorporating the temporal dimension into the model.

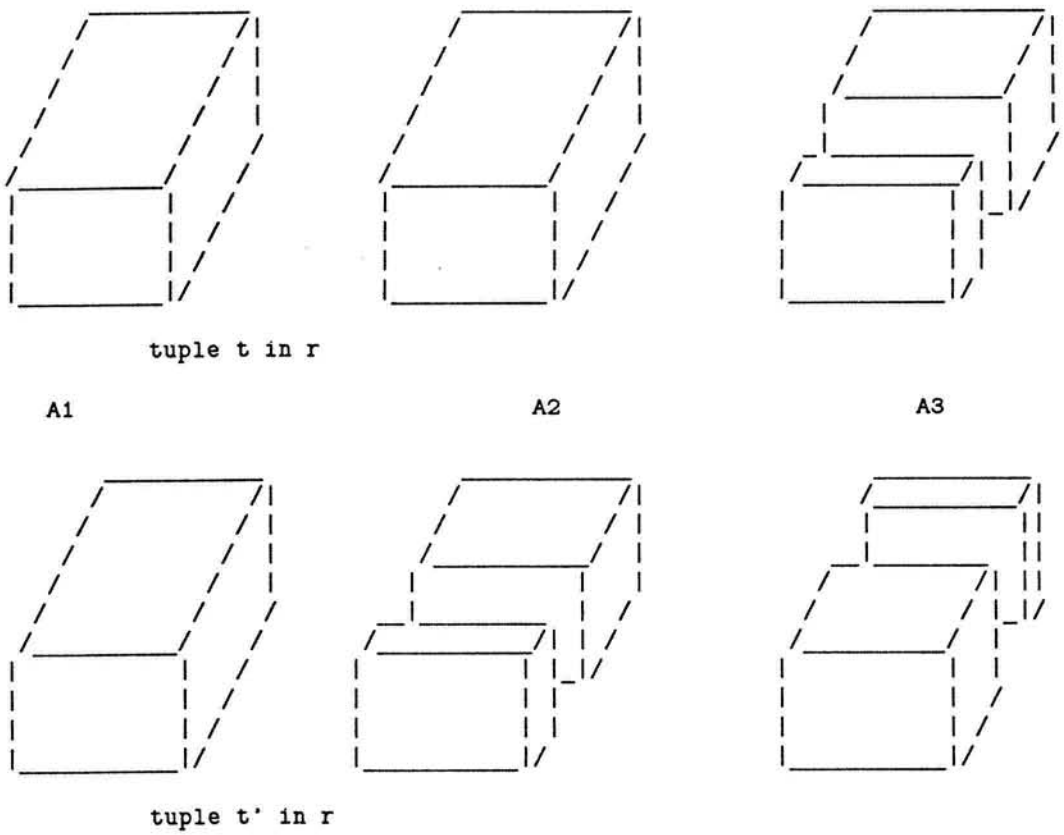

Figure 8: Lifespans associated with tuples and attributes

For the purposes of this paper the reader can assume that T is isomorphic to the natural numbers, and therefore the issue of whether to represent time as intervals or as points is simply a matter of convenience. Using the natural numbers allows us to restrict our attention to closed intervals (a closed interval of T, written $[t_1, t_2]$ is simply the set $\{t_i \mid t_1 <= t_i <= t_2 \}$). In a subsequent paper we will discuss more elaborate structures for the time domain of historical databases.

A **lifespan** L is any subset of the set T.

In order to provide for derived lifespans, we allow (similar to [Gadia 85] for the usual set-theoretic operations over lifespans. That is, if L_1 and L_2 are lifespans, then so are

1. $L_1 \cup L_2$
2. $L_1 \cap L_2$
3. $L_1 - L_2$

4. $\sim L_1$

Since lifespans are just sets, defined over a universe T, the semantics of these operators is apparent.

Let $D = \{ D_1, D_2, ..., D_{n_d} \}$ be a set of **value domains** where for each i, $D_i \neq \phi$. Each value domain D_i is analogous to the traditional notion of a domain in that it is a set of atomic (non-decomposable) values.

Using the sets T and D we define two sets of temporal mappings, one from the set T into the set D, TD, and the other from T into itself, TT.

The set $TD = \{ TD_1, TD_2, ..., TD_{n_d} \}$ where for each i, $TD_i = \{ f_i \mid f_i : T \dashrightarrow D_i \}$ is the set of all partial functions from T into the value domain D_i.

The set $TT = \{ g \mid g : T \dashrightarrow T \}$ is the set of all partial functions from T into itself.

The set of temporal functions TT serves a similar role in the model to each of the sets TD_i, but is defined separately to make explicit the distinction in the model between those values representing times, and those that do not.

Let $U = \{ A_1, A_2, ..., A_{n_a} \}$ be a (universal) set of **attributes**.

All attributes in the historical relational data model are defined over sets of partial temporal functions. Specifically, $HD = (TD \cup TT) = \{ TT, TD_1, TD_2, ..., TD_{n_d} \}$ is the set of all **historical domains**.

Among the functions in each of the set of functions in HD are some that are constant-valued, i.e., they associate the *same* value with every time in their domain. Let CD be the set derived from HD by restricting each of the sets of functions in HD to only those functions having a constant image. That is, for each set of functions TD_i (and TT) in HD restrict TD_i (and TT) to only those functions that map their domain to a single value.

We will sometimes want to restrict a function f with domain D to a smaller domain $D' \subseteq D$; we will

denote this restricted function by $f|_{D'}$.

A **relation scheme** R = <A,K,ALS,DOM> is an ordered 4-tuple where:

1. A = { A_{R_1}, A_{R_2}, ..., A_{R_n} } $\subseteq$ U is the set of **attributes of R**. We will sometimes abuse notation and refer to A as *the scheme of R*; no confusion should arise.

2. K = { A_{K_1}, A_{K_2}, ..., A_{K_m} } $\subseteq$ A is the set of **key attributes of R**

3. ALS: A --> 2^T is a function assigning a **lifespan** to each attribute in R. We will refer to the lifespan of attribute A in relation scheme R as ALS(A,R).

4. DOM: A --> HD is a function assigning a **domain** to each attribute in R, with the restrictions that (a) for all key attributes A_i, DOM(A_i) $\in$ CD, i.e., the key attributes must all be constant-valued; and (b) the domain of each of the partial functions in any DOM(A) is contained within ALS(A,R).

We refer to the underlying value set of attribute A (i.e., the ranges of the functions in DOM(A) as the **value-domain of A**, denoted VD(A). The value-domain corresponds to the traditional notion of the domain of an attribute.

A **tuple t on scheme R** is an ordered pair, t = <v,l>, where

1. t.l, the **lifespan of tuple t**, is a lifespan, and

2. t.v, the **value of the tuple** is a mapping such that $\forall$ attributes A $\in$ R, t.v(A) is a mapping in t.l $\cap$ ALS(A,R) --> DOM(A).

Since we associate a lifespan with both a tuple in a relation and an attribute in a scheme, we can derive the lifespan of the value of an attribute A in a tuple t in relation r on scheme R, which we will denote as vls(t,A,R). This lifespan represents the set of times over which the value is defined, and is given by:

$$vls(t,A,R) = t.l \cap ALS(A,R).$$

We can extend this definiton to a set of attributes X = $\{A_1, \ldots, A_n\}$ as follows:

$$vls(t,X,R) = t.l \cap ALS(A_1,R) \cap \ldots \cap ALS(A_n,R)$$

For simplicity we will refer to the value t.v of tuple t as follows. The value of tuple t for attribute A will be denoted by t(A). t(A)(s) is the value of tuple t for attribute A at time s. Similarly, t(X)(s)

represents the value of tuple t for a set of attributes X at time s. Since t(A) is a function with domain t.l $\cap$ ALS(A,R), the value of t(A)(s) is **undefined** for any s not in this time period. In this context undefined means that the attribute is not relevant at such times, and thus does not exist.

A **relation r on R** is a finite set of tuples t on scheme R such that if t_1 and t_2 are in r, $\forall s \in t_1.l$ and $\forall s' \in t_2.l$, $t_1.v(K)(s) \neq t_2.v(K)(s)$. If $r = \{t_1, t_2, ..., t_l\}$ is a relation on R, then LS(r), the lifespan of relation r, is just $LS(r) = t_1.l \cup t_2.l \ ... \cup t_l.l$.

Elsewhere ([Clifford 85], [Clifford 83], e.g.) we have described the need for an interpolation function to deal with the issue of incompletely specified time functions as the values of TV attributes. For the purposes of this paper we assume that t(A)(s) is the value of attribute A at time s for tuple t, regardless of *how* that value is obtained (for example, it might be stored directly in the relation, or computed by means of an interpolation function from stored values.)

Put slightly differently, we can assume that the model consists of three levels, the representation level, the model level, and the physical level, as in Figure 9. At the physical level are the file structures and access methods, at the model level each attribute in a tuple has as its value a *total* function from vls(t,A,R) into some value domain, while at the representation level these functions may be represented more succinctly using intervals and allowing for value interpolation.

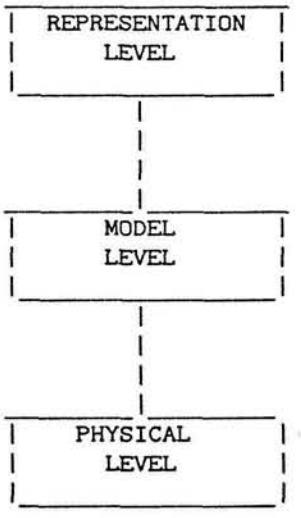

Figure 9. Levels in Historical Relational Data Model.

For example, assume that the lifespan of a particular value for some t(A) is S = vls(t,A,R). We can imagine a situation in which, for some S' $\subseteq$ S, at the representation level t(A) is a function from S' to the value-domain of A. Then the mapping from the representation level to the model level must include, for any such attribute, an **interpolation function** I:

$$I: VD(A)^{2^S} \dashrightarrow VD(A)^S$$

which maps each such "partially-represented function" into a total function from S. As another example, we might imagine that values constrained to be constant-valued functions might, at the representation level, be represented as simple <Lifespan,value> pairs (e.g., $<[t_i,t_j]$,Codd>).

These two types of lifespans, attribute and tuple, constrain the value of every attribute in every tuple as follows. The tuple lifespan indicates the periods of time during which the tuple bears information; a tuple has no value at points in time other than those in its lifespan. Moreover, each attribute in a relation has an associated lifespan, and so attributes in a tuple are further restricted to have no value outside of their own lifespan. Taken together, these two conditions imply that there is no value for an attribute in a tuple for any moment in time not in the intersection of the lifespans of the tuple and the attribute.

4. The Historical Relational Algebra of HRDM

We have expanded the allowable structures of the relational model in two significant ways. We have added a new type of *object* into the model's ontology, namely the set T of times, and have defined attributes to take on values which are functions from points in time (T) into some simple value domain (one of the D_i's or T). Secondly, we have defined the orthogonal concepts of *tuple lifespan* and *attribute lifespan* within a relation, to indicate when the value of an attribute in a tuple is defined. We now proceed to define an algebra over these structures.

The temporal component of the historical database model can in some sense be viewed as a third dimension to the relational model, as seen in Figure 10. Relational algebra provides a unary operator for each of its *two* dimensions (Select for the value dimension, Project for the Attribute dimension). The historical algebra will extend the definition of these two operators to operate historical relations, and add

a third operation (Time-Slice) for the added temporal dimension. The binary Join operation will be extended to join two historical relations. Finally, a When operator will be added to "extract" purely temporal information.

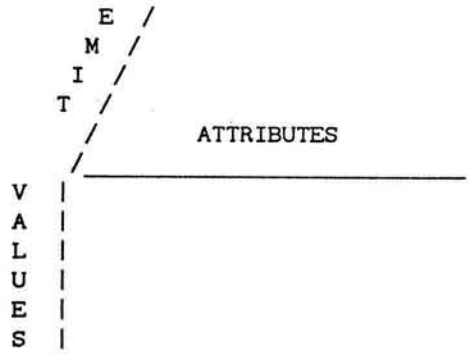

Figure 10: Three Dimensions of Historical Data Model

4.1. Set-Theoretic Operations

Historical relations, like regular relations, are set of tuples; therefore the standard set-theoretic operations of union, intersection, set difference, and Cartesian product **can** be defined over them.

As for standard relations, two historical relations r1 and r2 on $R_1 = <A_1,K_1,ALS_1,DOM_1>$ and $R_2 = <A_2,K_2,ALS_2,DOM_2>$ are said to be **union-compatible** if $A_1 = A_2$ and $DOM_1 = DOM_2$, i.e. they have the same attributes, with the same domains.

If r1 on R1 and r2 on R2 are union-compatible, then

1. $r1 \cup r2 = \{ t \text{ on } R3 \mid t \in r1 \text{ or } t \in r2 \wedge R3 = <A_1,K_1,ALS_1 \cup ALS_2,DOM_1> \}$
2. $r1 \cap r2 = \{ t \text{ on } R3 \mid t \in r1 \text{ and } t \in r2 \wedge R3 = <A_1,K_1,ALS_1 \cap ALS_2,DOM_1> \}$
3. $r1 - r2 = \{ t \text{ on } R1 \mid t \in r1 \text{ and } t \notin r2\}$

Given r1 on R1 and r2 on R2, where the attributes of R1 and R2 are disjoint, the Cartesian product is given as:

$$r1 \times r2 = \{ t \text{ on } R3 \mid \exists\, t1 \in r1,\ t2 \in r2\ [t.l = t1.l \cup t2.l \wedge \forall A \in R1\ \forall s \in t1.l\ [t.v(A)(s) = t1.v(A)(s)] \wedge \forall A \in R2\ \forall s \in t2.l\ [t.v(A)(s) = t2.v(A)(s)] \wedge \forall A \in R1\ \forall s \in t.l - t1.l\ [t.v(A)(s) = \lfloor\rfloor \wedge \forall A \in R2\ \forall s \in t.l - t2.l\ [t.v(A)(s) = \lfloor\rfloor \wedge R3 = <A_1 \cup A_2,K_1 \cup K_2,ALS_1 \cup ALS_2,DOM_1 \cup DOM_2>\}$$

However, a simple example shows that these operations produce counter-intuitive results for historical relations. If r1 and r2 are as in Figure 11, then the result of their union (r1 ∪ r2) is counter-intuitive; a more complex operation, which "merges" tuples of corresponding "objects" (producing r1 + r2 in Figure 11) is more in the spirit of the union operation respecting the semantics of objects. Similar remarks apply to differnce and intersection, and motivate three "object-based" versions of union, intersection, and difference, all of which rely on a preliminary definition of **mergable tuples.**

r1

A1	A2
t1 -> a	t1 --> b
t2 -> a	t2 --> b

r2

A1	A2
t2 -> a	t2 --> b
t3 -> a	t3 --> b

r1 ∪ r2

A1	A2
t1 -> a	t1 --> b
t2 -> a	t2 --> b
t2 -> a	t2 --> b
t3 -> a	t3 --> b

r1 + r2

A1	A2
t1 -> a	t1 --> b
t2 -> a	t2 --> b
t3 -> a	t3 --> b

Figure 11

Two relations r_1 and r_2 on schemes $R_1 = <A_1,K_1,ALS_1,DOM_1>$ and $R_2 = <A_2,K_2,ALS_2,DOM_2>$ are **merge-compatible** if and only if $A_1 = A_2$, $K_1 = K_2$, and $DOM_1 = DOM_2$

Merge-compatibility is therefore stricter than union-compatibility, by requiring the same key.

Two tuples r_1 and r_2 on schemes $R_1 = <A_1,K_1,ALS_1,DOM_1>$ and $R_2 = <A_2,K_2,ALS_2,DOM_2>$ are **mergable** if and only if

1. R_1 and R_2 are merge-compatible
2. $\forall s \in t_1.l \; \forall s' \in t_2.l \; [t_1.v(K_1)(s) = t_2.v(K_2)(s')]$
3. $\forall A \in A_1 \; \forall s \in (t_1.l \cap t_2.l) \; [t_1.v(A)(s) = t_2.v(A)(s)]$

Condition 2 specifies that the tuples have the same key value, and condition 3 that they do not contradict one another at any point in time.

If t_1 and t_2 are mergable, then their merge, denoted $\mathbf{t_1 + t_2}$ is given as follows:

$(t_1 + t_2).l = t_1.l \cup t_2.l$

$(t_1 + t_2).v(A) = t_1.v(A) \cup t_2.v(A)$ for all $A \in A_1$.

Given a tuple t and a set of tuples S, t is **matched in S** if there is some tuple t' in S such that t is mergable with t'; otherwise t is **not matched** in S.

With these preliminary definitions we can define more semantic-based set-theoretic operations $\cup_O$, $\cap_O$, and $-_O$:

If relations r1 on R1 and r2 on R2 are merge-compatible, then

$$r1 \cup_O r2 = \{ t \mid t \in r1 \text{ and } t \text{ is not matched in } r2 \vee t \in r2 \text{ and } t \text{ is not matched in } r2 \vee \exists t_1 \in r_1 \exists t_2 \in r_2 [t = t_1 + t_2] \}$$

$$r1 \cap_O r2 = \{ t \mid \exists t_1 \in r_1 \exists t_2 \in r_2 [t_1 \text{ and } t_2 \text{ are mergable} \wedge t.l = t_1.l \cap t_2.l \wedge \forall A \in R1 \; \forall s \in t.l [t_1.v(A)(s) = t_2.v(A)(s) = t.v(A)(s)]]\}$$

$$r1 -_O r2 = \{ t \mid t \in r1 \text{ and } t \text{ is not matched in } r2 \vee \exists t_1 \in r_1 \exists t_2 \in r_2 [t_1 \text{ and } t_2 \text{ are mergable} \wedge t.l = t_1.l - t_2.l \wedge \forall A \in R1 \; [t.v(A) = t_1.v(A)|_{t.l}]]\}$$

4.2. PROJECT

The project operator π when applied to a relation r removes from r all but a specified set of attributes; as such it reduces a relation along the attribute dimension. It does not change the values of any of the remaining attributes, or the combinations of attribute values in the tuples of the resulting relation. Let r be a relation over the set of attributes R and $X \subseteq R$. Then the projection of r onto X is given by:

$$\pi_X (r) = \{ t(X) \mid t \in r \}$$

4.3. SELECT

In the historical relational data model tuples, and thus the objects represented by those tuples, are viewed as having lifespans. The select operator applied to a relation is intended to select from the tuples of that relation those tuples that satisfy a simple selection criterion. Because of the existence of lifespans

we have the choice of selecting tuples over their entire lifespans, or selecting tuples ignoring all but a "relevant" subset of their lifespans. We define two "flavors" of select to reflect these two choices: SELECT-IF and SELECT-WHEN.

SELECT-IF reduces an historical relation along the value dimension. With it, if the selection criterion is met by a tuple t, then the entire tuple t is returned, and its lifespan is unchanged. The selection criterion, which we specify as Θ, is defined as a simple predicate over the attributes of the tuple. For example, the predicate A Θ a would select only those tuples whose value for attribute A stood in relationship Θ to the value a. (The value a could represent another attribute value or a constant.)

This flavor of select is closest to the definition of the select operator in the relational data model in that if a tuple is taken to represent some object, then in both cases a complete object either is or is not selected. In the historical relational data model a complete object is assumed to exist over its entire lifespan. Since attributes in this model have an associated lifespan it is necessary when specifying a selection predicate to also specify those times in the lifespan of the tuple (and attribute) when the predicate is to be satisfied.

Since values are functions over a set of times, the selection criterion must also specify for *which* times the criterion must me satisfied. This can be done by allowing either existential or universal quantification over a set of times. We use the notation Q(s $\in$ S) (where Q is one of the quantifiers *forall* or *exists*) to represent the *bounded quantification* (universal or existential) over all values in S. Then SELECT-IF can be defined as:

$$\sigma\text{-IF}_{(A\ \Theta\ a,\ Q,\ L)}\ (r) = \{t \in r \mid Q\ (s \in (L \cap t.l))\ [t(A)s\ \Theta\ a]\}$$

where Q is either the existential ($\exists$) or the universal ($\forall$) quantifier, and L is a lifespan. (If L = T, the set of all times, then $s \in (L \cap t.l)$ is equivalent to $s \in t.l$.)

With SELECT-WHEN, if the selection criterion is met by a tuple t at some time in its lifespan, what is returned is a new tuple t' whose lifespan is exactly those points in time WHEN the criterion is met, and whose value is the same as t for those points. The SELECT-WHEN is therefore a *hybrid* operation,

reducing an historical relation in both the value and the temporal dimensions.

$\sigma\text{-WHEN}_{A=a}\ (r) = \{t \mid \exists\ t' \in r\ [t.l = \{s \mid t'(A)(s) = a\} \wedge t.v = t'.v|_{t.l}\ ]\ \}$

For example, $\sigma\text{-WHEN}_{\text{EMP=John, SAL=30K}}$ (emp) would yield a relation (in this case with only 1 tuple, for key John) with a *new lifespan*, namely, just those times when John earned 30K.

4.4. TIME-SLICE

Corresponding to the unary operations SELECT and PROJECT, we define an additional unary operator, called TIME-SLICE, that reduces an historical relation in the temporal dimension. TIME-SLICE can be applied in one of two ways to create two different types of temporal subsets of its operand. We refer to these two applications of TIME-SLICE as *static* and *dynamic*. In a static TIME-SLICE ($T_{@L}$), the desired temporal subset (lifespan) of the operand is specified as a parameter (L) of the operator.

$T_{@L}\ (r) = \{t \mid \exists\ t' \in r\ [\ l = L \cap t'.l \wedge t.l = l \wedge t.v = t'.v|_{l}\ ]\}$

This version of TIME-SLICE defines a relation containing those tuples derived by restricting each tuple in the operand to those times specified as part of the operand, *lifespan*.

The dynamic TIME-SLICE makes use of the distinction in the historical data model between historical domains in TD (mappings from T into D) and those in TT (mappings from T into T). If attribute A is such that **DOM**(A) $\subseteq$ TT, then for any tuple t in a relation defined over A, the *image* of t(A) is the set of times that t(A) maps to. This set of times is used in defining a dynamic TIME-SLICE. Therefore the result of the dynamic TIME-SLICE ($T_{@A}$) is not defined over a fixed, pre-specified lifespan. Rather, the subset of the lifespan that is selected for each tuple is determined by the image of the value of a specified attribute for that tuple.

$T_{@A}\ (r) = \{t \mid \exists\ t' \in r\ [\text{for L, the image of t(A), } t.l = L \wedge t = t'|_{L}\ ]\ \}$

4.5. WHEN

As in [Gadia 85] (and analogous to the logical calculus defined in [Clifford 83]) we provide for a multi-sorted language whose universes are respectively relations and, using the terminology of this paper, lifespans. All of the operators except for WHEN are (unary or binary) operations on historical relations producing historical relations. The unary operator WHEN, denoted Ω, maps relations to lifespans. Its definition follows trivially from the relation lifespan concept:

$\Omega(r) = LS(r)$

Intuitively, the WHEN operator returns the set of times over which the relation is defined. Used in conjunction with other operators, for example SELECT, it provides then answer to *when* particular conditions are satisfied. (Note that since the result of WHEN is a lifespan, it can serve as the "parameter" to those relational operators (such as TIME-SLICE) which require a lifespan as input.)

4.6. JOIN

The binary relational operator join is used to combine two relations by concatenating a tuple of one with a tuple of the other whenever specified attributes of the two tuples stand in a specified relationship with each other. Paralleling the two types of values (*ordinary* and *time*) in the historical database model, we define flavors of join: Θ-JOIN and TIME-JOIN.

Θ-JOIN

The Θ-JOIN is intended to serve the same role in the historical relational data model as in the traditional relational data model; that is, it combines two tuples when the values of specified attributes stand in a Θ relationship with each other. However, the definition of Θ-JOIN must be modified so as to apply to historical tuples. (For all versions of JOIN, if the schemes of the two operands are $R_1 = <A_1,K_1,ALS_1,DOM_1>$ and $R_2 = <A_2,K_2,ALS_2,DOM_2>$ the resulting scheme $R3 = <A_1 \cup A_2, K_1 \cup K_2, ALS_1 \cup ALS_2, DOM_1 \cup DOM_2>$.

$$r1\ \text{JOIN}\ r2\ [A\ \Theta\ B] = \{\ t \mid \exists\ t_{r1} \in r1\ \exists\ t_{r2} \in r2\ [t.1 = \{s \mid t_{r1}(A)(s)\ \Theta\ t_{r2}(B)(s)\ \} \wedge$$
$$t.v(R1 - A) = t_{r1}.v(R1 - A)|_{t.1} \wedge$$
$$t.v(R2 - B) = t_{r2}.v(R2 - B)|_{t.1} \wedge$$
$$t.v(A) = t_{r1}.v(A)|_{t.1} \wedge$$
$$t.v(B) = t_{r2}.v(B)|_{t.1}\ ]\ \}$$

EQUIJOIN

The equijoin is just a special case of the general Θ-JOIN, but its definition can be simplified to the following:

$$r1\ [A = B]\ r2 = \{\ t \mid \exists\ t_{r1} \in r1\ \exists\ t_{r2} \in r2\ [t.1 = vls(t_{r1}, A, R1) \cap vls(t_{r2}, B, R2) \wedge$$
$$t.v(R1 - A) = t_{r1}.v(R1 - A)|_{t.1} \wedge$$
$$t.v(R2 - B) = t_{r2}.v(R2 - B)|_{t.1} \wedge$$
$$t.v(A) = t.v(B) = t_{r1}.v(A) \cap t_{r2}.v(B)\ ]\ \}$$

NATURAL-JOIN

The natural join is just a projection of the equijoin. Given relations r1 on R1 and r2 on R2, let X = A1 $\cap$ A2 be the set of attributes both schemes have in common. Then r1 NATURAL-JOIN r2 is a relation r3 on scheme R3 defined as follows:

$$r1\ \text{NATURAL-JOIN}\ r2 = \{\ t \mid \exists\ t_{r1} \in r1\ \exists\ t_{r2} \in r2\ [t.1 = vls(t_{r1}, X, R1) \cap vls(t_{r2}, X, R2) \wedge$$
$$t.v(R1) = t_{r1}.v(R1)|_{t.1} \wedge$$
$$t.v(R2) = t_{r2}.v(R2)|_{t.1}\ ]\ \}$$

TIME-JOIN

As was the case for the dynamic TIME-SLICE, TIME-JOIN is defined for attributes A with **DOM(A)** $\subseteq$ TT. In such cases it is possible to define a JOIN between a relation containing such a "time-valued" attribute A and some other relation. Essentially such a JOIN serves as a join of dynamic TIME-SLICEs of *both* relations.

Let r1 be a relation on scheme R1 and r2 be a relation on scheme R2, and A an attribute of R1. Then the TIME-JOIN of r1 and r2 at attribute A of r1, denoted r1 [@A] r2, is given as folows:

$$r1\ [@A]\ r2 = \{\ t \mid \exists\ t_1 \in r1\ \exists\ t_2 \in r2\ [t.l = \{t_1.v(A)(s) \mid s \in t_1.l\} \wedge t.v(R_1) = t_1.v(R_1)|_{t.l} \wedge t.v(R_2) = t_2.v(R_2)|_{t.l}\ ]\ \}$$

5. Summary and Future Work

In this paper we have presented the historical relational data model (HRDM) and its algebra. The structures of the traditional relational model were extended by the addition of a new type of *object* into the model's ontology, the set T of times, and the domains of relation attributes were extended to be functions from points in time (T) into some simple value domain. The concepts of tuple lifespan and attribute lifespan were proposed as a simple technique for providing both time-varying data and time-varying schemes, and providing a suitable level of user control over the temporal dimension of the data. Finally, we defined a relational algebra over these structures.

Exactly what should result from the Cartesian product (and by extension, Join) of three-dimensional relations is not immediately obvious. In this paper we have defined the Cartesian product in such a way that resulting tuples are defined over the union of the lifespans of the participating tuples, and thus potentially contain null values. We have defined the JOIN operations, however, to be equivalent to the appropriate SELECT-WHEN of the Cartesian product, and thus no nulls result; the JOIN of two tuples was defined only over their lifespan intersection. It would also be possible to define JOINs over the union of the tuple lifepans, essentially equivalent to a SELECT-IF of the Cartesian product; a resulting tuple will have null values for times outside of its contributing tuples' lifespans. As we pointed out at the end of Section 2, the most general historical model would associate lifespans with each **value** rather than each tuple; in this case no null values need result from product operation since the lifespans of any two values within a tuple are **completely** independent. It appears to be a tradeoff between the complexity of handling null values versus the complexity of handling additional lifespans.

HRDM is a *consistent extension* of the traditional relational data model. By consistent extension we mean that each component C of the relational model (structural or operational) has a corresponding component C^H in the historical relational model with the property that the defintions of C and C^H

become equivalent in the absence of a temporal dimension. It is beyond the scope of this paper to formally demonstrate this property. We leave that proof to a subsequent paper, along with a discussion of integrity constraints and alegebraic properties within HRDM. We will close with a few examples of these issues.

It is obvious that a traditional relation r is just a special case of an historical relation r^H. One way to view this is to consider the set of times T as the singleton set $\{now\}$, the lifespan of each tuple as T and the values of all tuples as constant functions from T to some value domain.

Considering the "SELECT" operation, both SELECT-IF and SELECT-WHEN reduce to one another and to the traditional SELECT on a static relation r, when $T = \{now\}$. Similar arguments can be made for the set-theoretic operators and for PROJECT and the JOINs. There are no direct analogs to the operations WHEN or TIME-SLICE; however TIME-SLICE can be viewed as the identity function defined only for time *now*, and WHEN maps a relation either to *now* or to the empty set, corresponding to either "always" or "never", respectively.

The strucures and operations presented in this paper represent only two components of the hgistorical relational data model; to further elaborate on HRDM would require a discussion of the extension of the various classes of constraints and the theory of normalization which has been developed for the traditional model. For example, the temporal dimension of historical relations can be used to extend the traditional notion of functional dependency (FD). The "meaning" of the traditional FD X --> A can be captured (as in [Clifford 83]) in a straightforward way. However in HRDM it becomes possible to define dependencies similar to FD's but which make explicit reference to points in time (variously called "intensional" [Clifford 83] or "dynamic" [Casanova 79] constraints.) For example, we can define dependencies that hold not only at each single point in time, but also that hold over all points in time. We can also define constraints over the way that values change over time (as in the familiar "salary must never decrease" example.) These and other types of temporal dependencies can be expected to have a significant impact on design methodologies for historical databases.

Many of the properties of the relational algebra carry over to the historical relational algebra. For

example, the commutativity of select, the distribution of select over the binary set-theoretic operators, and the commutativity of the natural join. The new operators in the model also exhibit properties analogous to these, such as the distribution of TIME-SLICE over the binary set-theoretic operators, commutativity of TIME-SLICE with both flavors of SELECT, etc. These properties follow from the use of functions as the domains of attributes, and the use of the simple concept of lifespans; a full treatment of them will require further elaboration.

Acknowledgements

The authors would like to thank Rick Snodgrass and Ed McKenzie for their valuable comments on a previous version of this paper.

References

[Ariav 84] Ariav, G., Beller, A., and Morgan, H.L.
A Temporal Model.
Technical Report DS-WP 82-12-05, Decision Sciences Dept., Univ. of Penn., December, 1984.

[Ben-Zvi 82] Ben-Zvi, J.
The Time Relational Model.
PhD thesis, Dept. of Computer Science, University of California, Los Angeles, 1982. (Unpublished).

[Casanova 79] Casanova, M.A., and Bernstein P.A.
The Logic of a Relational Data Manipulation Language.
In *Proceedings 6th ACM Symp. on Prog. Lang*, pages ?-?. 1979.

[Chen 76] Chen, P.P.S.
The Entity-Relationship Model: Towards a Unified View of Data.
ACM Trans. on Database Systems 1(1):9-36, March, 1976.

[Clifford 82] Clifford, J.
A Model for Historical Databases.
In *Proceedings of Logical Bases for Data Bases.* Toulouse, France, December, 1982.

[Clifford 83] Clifford, J., and Warren D.S.
Formal Semantics for Time in Databases.
ACM Trans. on Database Systems 6(2):214-254, June, 1983.

[Clifford 85] Clifford, J.
Towards an Algebra of Historical Relational Databases.
In *ACM-SIGMOD International Conference on Management of Data.* Austin, May, 1985.

[Gadia 85] Gadia, Shashi K. and Vaishnav, Jay.
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.

[Klopprogge 81] Klopprogge, M.R.
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.

[Klopprogge 83] Klopprogge, M.R., and Lockemann, P.C.
Modelling Information Preserving Databases: Consequences of the Concept of Time.
In *Proc. of The Ninth International Conference on Very Large Data Bases*, pages 399-416. 1983.

[Lum 84] Lum, V., et al.
Designing DBMS Support for the Temporal Dimension.
In *ACM-SIGMOD International Conference on Management of Data*, pages 115-126. Boston, June, 1984.

[Navathe 80] Navathe, S. B.
Schema Analysis for Database Restructuring.
ACM Trans. on Database Systems 5(2):157-184, June, 1980.

[Shiftan 85] Shiftan, J.
An Assessment of The Temporal Differentiation of Attributes in The Design of Temporally Oriented Relational Databases.
Technical Report, Dept. of Computer Applic. and Info. Sys., New York Univ., February, 1985.
(Ph.D. Dissertation in preparation).

[Snodgrass 84] Snodgrass, R.
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.

[Snodgrass 85] Snodgrass, Richard and Ahn, Ilsoo.
A Taxonomy of Time in Databases.
In *ACM-SIGMOD International Conference on Management of Data*, pages 236-246. Austin, TX, May, 1985.