ON COMPLETENESS OF HISTORICAL RELATIONAL DATA MODELS

by

Albert Croker

and

James Clifford

Information Systems Department
Leonard N. Stern School of Business
New York University
90 Trinity Place
New York, NY 10006

January 1989

Center for Research on Information Systems
Information Systems Department
Leonard N. Stern School of Business
New York University

Working Paper Series

CRIS #192
GBA #89-2

Abstract

Several proposals for extending the relational data model to incorporate the temporal dimension of data have appeared in the past several years. These proposals have differed considerably in the way that the temporal dimension has been incorporated both into the structure of the extended relations that are defined as part of these extended model, and into the operations of the extended relational algebra or calculus component of the models. Because of these differences it has been difficult to compare the proposed models and to make judgements as to which of them is "better" or indeed, the "best." In this paper we propose a notion of **historical relational completeness**, analogous to Codd's notion of relational completeness, and examine several historical relational proposals in light of this standard.

1 Introduction

In this paper we define a notion of completeness that is applicable to historical relational data models. This **historical relational completeness** provides a basis for determining the "power" of the query languages that have been defined as part of proposed historical relational data models. As such, historical relational completeness serves a role similar to that of the original notion of relational completeness first proposed by Codd [Cod72] and later justified as being reasonable by Bancilhon [Ban78].

Recently various historical relational data models have been proposed [Ari86,Ben82,CC87,Gad86,Sno87]. These data models are intended for those situations where data changes over time, but for which previous values of data items must remain as part of the database. Generally, these data models "extend" the standard relational data model by including a temporal component. This temporal component could be included by simply adding an additional attribute, say *time*, to a relation (the equivalence of time-stamping) [Sno87] or by including it as a more intrinsic part of the structure of a relation [CC87,Gad86]. The latter approach results in **non-first normal form** relations.

Although the structures of the **historical relations** defined in each of the proposed historical relational data models differ from each other to varying degrees, they generally have the same modeling capabilities. However, the query languages defined in these data models differ from each

other, not only in their formulations, but in their capabilities for use in extracting various subsets of a database.

In order to define a basis against which to measure the various query languages we define a **historical relational calculus** as a first-order language with a universe consisting of both objects (values) and times. This calculus is defined with respect to a specific historical relation structure, which we also formally define, that has a modeling capability that is at least as great as those defined in other historical relational data models. The historical relational calculus, viewed as a query language, provides a formal method for denoting the set of historical relations derivable from a given set of historical relations (a **historical database**). The set of derivable relations provide a measure of the **selectivity** of the historical relational calculus.

Each of the proposed query languages has an associated selectivity. This selectivity, which is what we have previously referred to as the power of the language, represents a common basis for comparison. Although the historical relational calculus cannot (and is not intended to) represent all possible queries — for example, like standard relational languages, it too cannot express a transitive closure — its appropriateness as a basis for historical completeness is dependent on its ability to express only "reasonable" queries, and to express those queries expressible by the other proposed historical query languages.

In Section 2 we define a historical relational calculus that serves as the

basis for our notion of historical relational completeness. Included along with the specification of the syntax and semantics of this calculus is a specification of the structure of the historical relation used in defining the calculus. We follow in Section 3 with a discussion of the appropriateness of this relational calculus as a basis for historical completeness. In the next section we examine the completeness of several historical relational languages that have been proposed in order to assess their completeness. We conclude in Section 5.

2 An Historical Relational Calculus

2.1 Preliminaries

In this section we specify the historical relation structure that will be used in the development of our notion of historical relational completeness. In order that this notion of completeness be applicable to the various historical relational data models that have been proposed, it is necessary that this structure have the representational capabilities of the various relation structures defined in these models. These required capabilities can be determined from the intent of a historical relational data model.

Both historical and non-historical databases model situations that are usually viewed as being dynamic. Thus the *state* of the database must also be able to change. Traditionally non-historical databases handle change through operations of the type *INSERT* (a record into the database), and

UPDATE or *DELETE* (an already existing database record). When a record is modified some of the attribute values are replaced by new values, with the old values being lost; when a record is deleted, all trace of the existence of the referent of that record is lost. When a new record is added to the database it is usually not known whether or not the database contained a record with the same key (and therefore modeled the same *object*[1]) at some time in the past. Each of the above operations cause the previous state of the database to be lost.

In a historical database the state of the database as it existed at any point in the past must be retained. Thus, assuming a tuple correlates to an entity, each tuple in a historical relational database represents a history of values associated with each of the attributes over which the tuple is defined. Reflecting this view, historical relations are often depicted as three-dimensional structures (Figure 1). The uneveness and holes in the structure reflect the fact that tuples may be inserted at different times, and that during certain periods the entity modeled by a tuple may no longer be relevant, the tuple being viewed as not existing during those periods.

2.2 A Canonical Historical Relation

In this section we define the structure of a historical relation upon which we will base the calculus that we define in the next section. The structure

[1]We use the term object to refer to both entities and relationships, as defined, for example, in the entity-relationship data model [Che76].

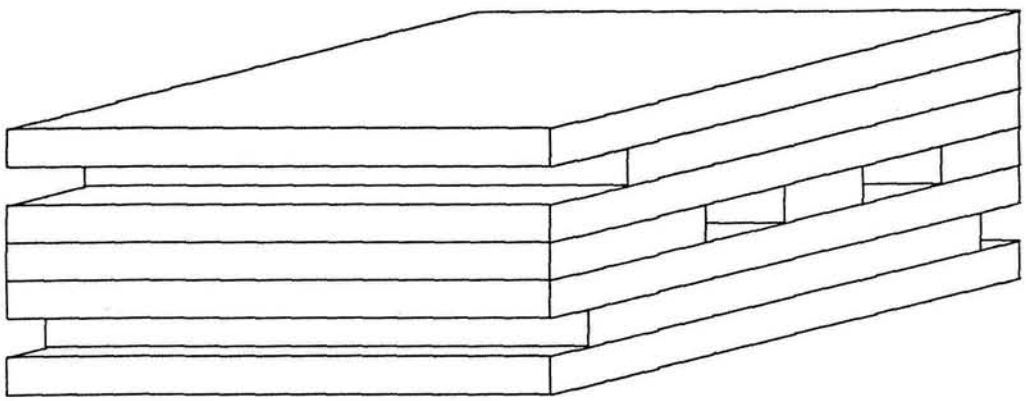

Figure 1: Historical Relation as a 3-D Structure

of this relation is specified in such a way as to capture the intent and requirements of a historical relation, and to be general enough to have the representational capabilities of other proposed historical relations. We refer to this relation as a *canonical historical relation.*

Let $U_D = \{D_1, D_2 \ldots, D_{n_d}\}$ be a (universal) set of **value domains** where for each i, $D_i \neq \emptyset$. 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. Further, let $\mathbf{D} = \bigcup_{i=1}^{n_d} D_i$ be the set of all values.

Associated with each value domain D_i is a set of **value comparators** Θ_{D_i}, each element of which can be used to compare two elements of the domain. At a minimum each set of value comparators contains the com-

parators "=" and "$\neq$" to test for the equality and inequality, respectively, of any two elements of the associated value domain.

Let $T = \{\mathbf{t}_0, \mathbf{t}_1, \ldots, \mathbf{t}_i, \mathbf{t}_{i+1}, \ldots\}$ be a non-empty set, the set of **times**, and let $<$ be a total order on T. The cardinality of T is restricted to be at most countably infinite. We call any subset $L \subseteq T$ a **lifespan**.

Corresponding to each value domain D_i is a **temporal domain** D_i^T of partial temporal-based functions from the set of times to the value domain D_i. Each of these partial functions define an association between each time instance in some lifespan L, and a value in a designated value domain, and thus provides a means of modeling the changing of an attribute's value over time.

Let $U_A = \{A_1, A_2, \ldots, A_{n_a}\}$ be a (universal) set of **attributes**. Each attribute names some property of interest to the application area.

A **historical relation scheme** R is a 3-tuple $R =< A, K, DOM >$ where:

1. $A \subseteq U_A$ is the set of **attributes** of scheme R
2. $K \subseteq A$ is the **key** of scheme R
3. $DOM : A \rightarrow U_D$ is a function that assigns to each attribute of scheme R a value domain, and, by extension, the corresponding temporal domain. We denote the value domain of attribute A_i in scheme R by $DOM(A_i, R)$.

A **historical relational database scheme** $DB = \{R_1, \ldots, R_n\}$ is a finite set of historical relation schemes.

A **historical tuple** t on scheme $R =< A, K, DOM >$ is a function that associates with each attribute $A_i \in A$ a temporal-based function from a common lifespan $L \subseteq T$ to the value domain assigned to attribute A_i; that is, $t(A_i) : L \rightarrow DOM(A_i)$. The subset of times L is called the **tuple lifespan** of t and is denoted $t.l$. We note that it is also possible to associate lifespans with attributes [CC87]. Doing so permits historical relation schemes to accomodate changes that may occur to them over time.

A **historical relation** r on scheme $R =< A, K, DOM, >$ is a finite set of historical tuples on scheme R such that given any two tuples t_1 and t_2 in r, $\forall s_1 \in t_1.l$ and $\forall s_2 \in t_2.l$, $\exists A_i \in K$ such that $t_1(A_i)(s_1) \neq t_2(A_i)(s_2)$. With the concept of a key that we are using here a tuple must throughout its lifespan differ, with respect to its key attributes, from every other tuple, throughout their respective lifespans. Although in general we would assume that the temporal-based function associated with each key attribute of a tuple is constant with respect to the lifespan of that tuple, we do not require it to be so.

In Figure 2 we show two historical relations: **EMPL** and **DEPT**. The tuples in these relations are shown separated by horizontal lines. The temporal based functions assigned to tuple attributes are depicted as a collection of time intervals such that all of the times within an interval are associated with the same value domain value. Thus, for example, a salary

EMPL

NAME	*DEPT*	*SALARY*
$[0, now] \rightarrow$ Tom	$[0, 10) \rightarrow$ Sales $[10, now] \rightarrow$ Mktg	$[0, 7) \rightarrow$ 20K $[7, 11) \rightarrow$ 30K $[11, now] \rightarrow$ 27K
$[5, now] \rightarrow$ Juni	$[5, now] \rightarrow$ Acctng	$[5, now] \rightarrow$ 28K
$[2, 10) \rightarrow$ Ashley $[14, now] \rightarrow$ Ashley	$[2, 6) \rightarrow$ Engrng $[6, 10) \rightarrow$ Mktg $[14, now] \rightarrow$ Engrng	$[2, 5) \rightarrow$ 27K $[5, 10) \rightarrow$ 30K $[14, now] \rightarrow$ 35K

DEPT

DEPT	*MGR*
$[0, now] \rightarrow$ Acctng	$[0, 5) \rightarrow$ Paul $[5, now] \rightarrow$ Juni
$[0, now] \rightarrow$ Engrng	$[0, now) \rightarrow$ Wanda $[14, now] \rightarrow$ Ashley
$[0, now] \rightarrow$ Mktg	$[0, now] \rightarrow$ Tom
$[0, now] \rightarrow$ Sales	$[0, now] \rightarrow$ Sue

Figure 2: The Historical Relations **EMPL** and **DEPT**

of 20K is associated with each of the times in the open interval $[0, 7)$ of the attribute *SALARY* of the first tuple in the relation **EMPL**. We use the symbol *now* to designate that element of the set of times T that corresponds to the current time.

A **historical database** $d = \{r_1, r_2, \ldots, r_n\}$ is a set of historical relations where each r_i is defined on a not necessarily unique historical relation scheme R_i.

2.3 The Calculus

Two types of calculi have been defined for the standard relational data model: domain relational calculi and tuple relational calculi. Although the equivalence of these two types of formulation is well-known, in general, the tuple relational calculus is easier to understand, and has served as the basis for the most popular implementations of relational query languages, SQL and QUEL. We will therefore define a tuple calculus for historical relations. To simplify the discussion we will assume that we are defining an applied relational tuple calculus relative to a particular relational database $d = \{r_1, r_2, \ldots, r_n\}$.

The major differences between historical relational data models and the standard relational data model arise from the explicit incorporation of a temporal component into the model. This difference is reflected in the definition of the calculus that we specify as the language L_h.

2.3.1 The Language L_h

The Syntax of L_h

1. The Basic Expressions of L_h are of three categories:

   (a) *Constant Symbols*

      i. $C_D = \{\delta_0, \delta_1, \delta_2, \ldots\}$ is a set of individual constants, at most denumerably infinite, one for each value δ in **D**

ii. $C_T = \{\tau_0, \tau_1, \tau_2, \ldots\}$ is a set of temporal constants, at most denumerably infinite, one for each time τ in T

iii. $C_A = \{A_1, A_2, A_3, \ldots\}$ is a finite set of attribute name constants, one for each attribute A in U_A.

(b) *Variable Symbols*

i. $V_T = \{t_0, t_1, t_2, \ldots\}$ is a denumerably infinite set of temporal variables

ii. $V_{TV} = \{x_0, x_1, x_3, \ldots\}$ is a denumerably infinite set of tuple variables

(c) *Predicate Symbols*

i. $\theta = \{\theta_1, \theta_2, \theta_3, \ldots, \theta_{n_\theta}\}$ is a set of binary predicates, one corresponding to each value comparator defined on objects of type γ (e.g., values from a common value domain).

ii. $\mathbf{r} = \{r_1, r_2, \ldots, r_n\}$ is a set of relation predicates, one corresponding to each relation r in the database d.

2. The *Terms* of L_h are of several *Categories*, given as follows:

(a) If x is a tuple variable, A is an attribute name constant, and t is a temporal variable, then

i. $x.A$ is a Term of Category **indexed tuple**

ii. $x.A(t)$ is a Term of Category **indexed tuple value**

(b) If x is a tuple variable, then

i. $x.l$ is a Term of Category **lifespan**

(c) If $L \subseteq C_T$, then

i. L is a Term of Category lifespan

(d) If L_1 and L_2 are Terms of Category lifespan, then so are $L_1 \cup L_2$, $L_1 \cap L_2$, and $L_1 - L_2$.

3. The *Formulae* of L_h are the following:

(a) If α and β are both terms of the same category, then

i. $\alpha = \beta$ and $\alpha \neq \beta$ are formulae.

(b) If α and β are both indexed tuple values, δ is an individual constant, and θ is a dyadic predicate, then

i. $\alpha\theta\beta$ is a formula, and

ii. $\alpha\theta\delta$ and $\delta\theta\alpha$ are formulae.

(c) If α is a lifespan term and t is a temporal variable, then

i. $t \in \alpha$ is a formula.

(d) If t_1 and t_2 are temporal variables and τ is a temporal constant, then

i. $t_1 < t_2$ is a formula,

ii. $\tau < t_1$ and $t_1 < \tau$ are formulae, and

iii. $\tau = t_1$ and $t_1 = \tau$ are formulae.

(e) If r is a relation predicate and x is a tuple variable, then

i. r(x) is a formula.

(f) If ϕ and ψ are formulas, then so are (ϕ), $\neg\phi$, $(\phi \wedge \psi)$, $(\phi \vee \psi)$, $(\phi \rightarrow \psi)$, and $(\phi \leftrightarrow \psi)$.

(g) If ϕ is a formula and u is a variable, then $\forall u\phi$ and $\exists u\phi$ are both formulae.

4. The *Expressions* of L_h are all expressions of the form: $[x_1.A_1, \ldots, x_n.A_n : t]\phi$, where:

   (a) $[x_1.A_1, \ldots, x_n.A_n : t]$ is called a target list, and consists of

      i. A list of indexed tuple terms $x_i.A_i$

      ii. A temporal variable t

   (b) ϕ is a formula.

   As a convenience we use for a set of attributes $A = \{A_1, A_2, \ldots, A_n\}$ the notation $x.A$ to denote the list $x.A_1, x.A_2, \ldots, x.A_n$ in a target list. Similarly, given a tuple variable x that ranges over a set of tuples on a common scheme that consists of the set of attributes $A = \{A_1, A_2, \ldots, A_n\}$, we use the notation $x.*$ to denote $x.A$.

The Semantics of L_h Here we give the intended interpretation of the tuple relational calculus. For convenience the numbering used here correlates directly with that used in the specification of the syntax.

1. The Basic Expressions of L_h denote as follows:

   (a) *Constant Symbols*

i. An individual constant denotes an object in some value domain D_i

ii. A temporal constant denotes a time in the universe of times T

iii. An attribute name constant denotes an attribute in U_A.

(b) *Variable Symbols*

i. A temporal variable denotes a time in the universe of times T

ii. A tuple variable denotes a tuple in some relation r in the database d

(c) *Predicate Symbols*

i. A binary predicate symbol denotes some value comparator (e.g., $=, \neq, \leq, \ldots$) over objects in some value domain

ii. A relation predicate r denotes a relation (set of tuples) in the database

2. The *Terms* of L_h denote as follows:

(a) An indexed tuple denotes a temporal-based function from a lifespan to a value domain

(b) An indexed tuple value denotes an object in some value domain

(c) A lifespan denotes a set of times

3. The *Formulae* of L_h denote as follows:

(a) $\alpha = \beta$ and $\alpha \neq \beta$ are true just in case the object denoted by α is identical (respectively, not identical) with the object denoted by β , and false otherwise.

(b) $\alpha\theta\beta$ is true just in case the object denoted by α stands in the relation θ with the object denoted by β, and false otherwise; similarly for $\alpha\theta\delta$.

(c) $t \in \alpha$ is true just in case the time denoted by t is in the lifespan denoted by α, and false otherwise.

(d) $t_1 < t_2$ is true just in case the time denoted by t_1 occurs before the time denoted by t_2, and false otherwise; similarly for $\tau < t_1$ and $t_1 < \tau$.

(e) $\tau = t_1$ is true just in case the time denoted by τ is the same time as that denoted by t_1, and false otherwise.

(f) r(x) is true just in case the tuple denoted by x is in the relation denoted by r, and false otherwise.

(g) (ϕ), $\neg\phi$, $(\phi \wedge \psi)$, $(\phi \vee \psi)$, $(\phi \rightarrow \psi)$, and $(\phi \leftrightarrow \psi)$. are true just in case the obvious conditions on ϕ and ψ hold.

(h) $\forall u\phi$ and $\exists u\phi$ are true just in case the obvious conditions on ϕ and u hold.

4. An *Expression* $[x_1.A_1, \ldots, x_n.A_n : t]\phi$ of L_h denotes a historical relation, each n-tuple of which is derived from a satisfying assignment to the variables of the formula ϕ. The components of the n-tuples are

denoted by the indexed tuple terms x_i. The lifespan of each derived tuple is the set of values of the temporal variable t that together with the values of the tuple values satisfies the formula ϕ.

In order to ensure that the relations denoted by expressions of the calculus are well-defined we include along with the syntax given earlier several additional restrictions. First, we require that each tuple variable specified in the target list, either in the specification of an indexed tuple term, or in denoting a tuple lifespan, must also be specified within the associated formula. Second, we require that tuple variables range only over tuples in relations in the database, and temporal variables, when included in an indexed tuple value term, range only over times in the lifespan of the tuple denoted by the tuple variable component of that term. These restrictions on the ranges of variables characterize a concept of **safe** formula analogous to that defined for the standard relational calculus [Mai83]. Finally we require that a tuple variable can range only over tuples in relations of the same type; that is, in the terminology of the standard relational data model, relations that are **union-compatible**.

In the following we give several examples of queries expressed in the language L_h for the database shown in Figure 2. In these examples the symbols used to identify each of the relation predicates are the names of the relations (i.e., **EMPL** or **DEPT**) that corresponds to that symbol.

Q1. Who are the employees currently in the marketing department?

$$[e.* : t]\mathbf{EMPL}(e) \wedge e.DEPT = Mktg \wedge t \in e.l$$

Q2. Who are the managers for whom Tom has worked?

$$[e_1.* : t]\mathbf{EMPL}(e_1) \wedge \mathbf{EMPL}(e_2) \wedge \mathbf{DEPT}(d) \wedge t \in e.l \wedge$$
$$\exists t_1(e_2.NAME(t_1) = Tom \wedge e_2.DEPT(t_1) = d.DEPT(t_1) \wedge$$
$$d.MGR = e_1.NAME)$$

Q3. Name and salary of each employee?

$$[e.NAME, e.SALARY : t]\mathbf{EMPL}(e) \wedge t \in e.l$$

Q4. Name and salary of each employee at time 12?

$$[e.NAME, e.SALARY : t]\mathbf{EMPL}(e) \wedge t \in e.l$$

Q5. Who are the employees who have only worked in the accounting department?

$$[e.* : t]\mathbf{EMPL}(e) \wedge t \in e.l \wedge \forall t_1(t_1 \in e.l \rightarrow e.DEPT(t_1) = Acctng)$$

3 An L_h-Based Notion of Completeness

The appropriateness of the language L_h as a standard for historical relational query language completeness derives from that of the relational calculus as a standard for relational query language completeness. For both languages this appropriateness can be viewed in terms of the set of relations that can be defined by expressions in the languages.

The relational calculus of the standard relational data model has been shown to be equivalent to its relational algebra. This equivalence is with respect to the set of relations that can derived from a given set of (*base*) relations. A relational calculus formula denotes a subset of (i.e., selects tuples from) the Cartesian product of the sets of tuples over which each of the tuple variables in the formula ranges. (A target list is then used to project out the desired attributes values from the selected tuples.)

For example, let ϕ' be a relational calculus formula that is defined over the tuple variables $x_1, x_2, ..., x_n$. Each tuple variable x_i ranges over a set of tuples r_i that is derived from the set of database relations using the set-theoretic operators $\cup$, $\cap$, and $-$. (In the query languages SQL and Quel each tuple variable is restricted to range over the tuples in a single relation.) Let the relation $r^* = r_1 \times r_2 \times \cdots \times r_n$ be defined as the Cartesian product of these sets. The formula ϕ' specifies a selection criteria on r^* that allows one or more attribute values of each of its tuples to be compared with other attribute values in the same tuple, or, through the use of existential and

universal quantifiers, with attribute values of other tuples in r^*.

In the standard relational data model an attribute value is an element of the value domain associated with that attribute. By definition, all such values are atomic. Thus, the relational calculus indexed tuple term $x_1.A_i$ denotes the value of attribute A_i in the tuple designated by tuple variable x_1. Likewise, the formula (or component of a formula) $x_1.A_i\theta x_2.A_j$ expresses the relationship, denoted by the comparator θ, between the values denoted by the terms $x_1.A_i$ and $x_2.A_j$.

Relational calculus expressions differ from those in the historical relational calculus specified by the language L_h by the absence of temporal constants and variables. This lack of temporal components reflects the view that, in some sense, the state of a standard relational database reflects a single point in time.

A formula in the language L_h also denotes a selection of tuples from the Cartesian product of the sets of tuples over which its tuple variables range. The similarity is such, that a formula in L_h that contains no temporal variables or constants, and thus can contain only the comparators equality "=" and inequality "$\neq$", is also a formula in the standard relational calculus, and has the same interpretation. (Each formula that is specifiable in the standard relational calculus, with the exception of the comparator operators used, is also a formula in the historical relational calculus.)

In the historical relational data model value domains are also associ-

ated with each attribute. However, unlike the standard relational model, the value of an attribute in a tuple denotes a time-varying sequence of values from the associated value domain. We have chosen to represent this sequence as a temporal-based function that maps each relevant point in time (i.e., elements of the tuple's lifespan) to the appropriate value in the value domain. We believe that this is a natural way of viewing time-varying attribute values. Other researchers have chosen different, though essentially equivalent, representation schemes as their metaphor. Thus the historical relational calculus indexed tuple term $x.A_i$ denotes a partial function from time into the value domain of attribute A_i, and the term $x.A_i(t)$ denotes the value of that attribute at the time denoted by t.

Temporal variables and constants provide a means of extending the selection criteria specified on the Cartesian product beyond that of simply comparing for equality and inequality attribute values that are, in the case of the historical relational data model, temporal-based functions.

Using temporal variables and constants, and the dyadic predicates used to denote comparators, it is possible to express with a formula a selection criteria for historical tuples in the Cartesian products of the sets of tuples r_i over which the tuple variables of the formula range. This selection criteria allows the value of one or more attributes at specified points in time to be compared with the values of other attributes at other specified points in time. These other attributes may be in the same or different tuples. In effect, this selection criteria is defined over a Cartesian product of the

sequence of values assigned to each attribute in a tuple (extending the Cartesian product into the temporal dimension of the historical relation).

As an example, let x_1 and x_2 be two tuple variables, t_1 and t_2 denote two points in time, and θ be a dyadic predicate. Further, let A_1 and A_2 be two attributes having temporal function domains that correspond to value domains that are θ comparable. The formula $x_1.A_1(t_1)\theta x_2.A_2(t_2)$ then asserts that the value of attribute A_1 in tuple x_1 at time t_1 satisfies the relationship specified by θ with the value of attribute A_2 in tuple x_2 at time t_2. Both $x_1.A_1$ and $x_2.A_2$ denote temporal functions. Existential and universal quantifiers can be used to further specify (indeed, to completely specify) the times for which values are to be compared.

Although a formula such as $x_1.A_1(t_1)\theta x_2.A_2(t_2)$ has been presented as denoting a comparison between two attribute values at the indicated times, it can also be viewed as denoting a comparison between attribute values that, as in the standard relational data model, are viewed as being atomic. That is, θ, t_1, and t_2 can be viewed as parameters that select a function comparator $\tilde{\theta} = \mathcal{F}(\theta, t_1, t_2)$ where $x_1.A_1\tilde{\theta}x_2.A_2$ is true exactly when $x_1.A_1(t_1)\theta x_2.A_2(t_2)$ is true. Similarly, other function comparators can be defined for formulae that contain temporal constants and quantifiers over the temporal variables.

4 Historical Models and Completeness

All of the historical relational data models that have been proposed differ from one another in the set of query operators that they provide. In addition, they often differ in the structure of the historical relations that they specify, that is, the way in which the temporal component is incorporated into the structure. In this section we describe several of these models, and discuss their completeness with respect to the historical calculus characterized by the language L_h.

For each of the data models discussed in the following, we are interested in two aspects of its query language relative to the language L_h: its **expressiveness**, that is, its ability to express every relation that can be denoted by expressions of the language L_h, and its **boundedness**, its ability to express only those relations that can be expressed in L_h. The standard relational calculus satisfies both of these criteria with respect to the standard relational calculus.

We define the completeness of a language soley in terms of its relative expressiveness. That is, a language is complete with respect to the language L_h if it is as expressiveness as L_h. Allowing that all reasonable queries can be expressed in L_h, we also consider the boundedness of each of the query languages discussed here. If L_h is an appropriate basis for a notion of completness, then it must be the case that each language considered is bounded by L_h, or if not, then those queries that are not expressible in L_h are in

some sense "not reasonable". For each of the historical query languages discussed in the following we consider first its boundedness, translating various of its operators into equivalent expression in the language L_h, and then its expressiveness.

We begin with a discussion of the completeness of the historical relational algebra specified by the historical relational data model **HRDM** [CC87]. We discuss this language first both because the canonical historical relation defined in Section 2.1 is derived directly from the structure of the historical relation defined by **HRDM**, (thus we assume the same relational structure in the following), and because the set of operators specified by this model were intended initially to provide all the functionality thought useful and desirable.

4.1 HRDM

In order to establish the historical relational completeness of the **HRDM** algebra or any other historical query language, it is sufficient to provide a translation from each relation defining expression in L_h to a semantically equivalent expression of that query language. Similarly, a query language is *incomplete* if it can be shown that for some relation defining expression in L_h no such translation exists.

We categorize the operators of the **HRDM** as follows:

Set-Theoretic These operators are defined in terms of the set characteristics of relations, and include the standard set operators union ($\cup$), intersection ($\cap$), set difference (-), and Cartesian product ($\times$).

Attribute-Based This category includes those operators that are defined in terms of the attributes (or their values) of a relation. Some of these operators, as suggested by their names, are derived from similar operators that exist in the standard relational algebra. As shown below, often the original definition of these operators has been modified to exploit the temporal component of the historical model. For each of these operators we give both its set-theoretic definition, and then an equivalent L_h-based expression.

1. **Project (π):** This operator is equivalent in definition to its standard relational counterpart, and has the affect of reducing the set of attributes over which each of the tuples x in its operand, a relation r, is defined, to those attributes contained in a set of attributes X.

$$\begin{aligned} \pi_X(r) &= \{x(X) | x \in r\} \\ &\equiv [x.X : t] r(x) \wedge t \in x.l \end{aligned}$$

2. **Select-If ($\sigma - IF$):** This variant of the select operator selects from a relation r those tuples x each of which for some period within its lifespan has a value for a specified attribute A that satisfies a specified

selection criterion. The period of time within the lifespan is specified by a lifespan parameter L. The selection criterion is specified as $A\theta a$, where θ is a comparator and a is a constant. (It is also possible to compare one attribute with another in the same tuple.) A parameter, Q, of the select-if operator is used to denote a quantifier that specifies whether the selection criterion must be satisfied for all ($\forall$) times in the specified subset of the tuple's lifespan, or that there exists ($\exists$) at least one such time.

$$\begin{aligned}\sigma - \mathrm{IF}_{(A\theta a,Q,L)}(r) &= \{x \in r | Q(t \in (L \cap x.l))[x.A(t)\theta a]\} \\ &\equiv [x.* : t]r(x) \wedge t \in x.l \wedge \\ &\quad Qt_1(t_1 \in (L \cap x.l) \rightarrow x.A(t)\theta a)\end{aligned}$$

3. **Select-When ($\sigma - WHEN$):** This operator is similar to the $\exists$-quantified select-if operator. However, the lifespan of each selected tuple is restricted to those times *when* the selection criterion is satisfied.

$$\begin{aligned}\sigma - \mathrm{WHEN}_{A\theta a}(r) &= \{x | \exists x' \in r[x.l = \{t | x'.A(t)\theta a\} \wedge x.v = x'.v|_{x.l}]\} \\ &\equiv [x.* : t]r(x) \wedge t \in x.l \wedge x.A(t)\theta a\end{aligned}$$

4. **θ-Join:** Like its counterpart in the standard relational data model this operator combines tuples from its two operand relations. With

θ-join two tuples are combined when two attributes, one from each tuple, have values at some time in the intersection of the tuples' lifespans that stand in a θ relationship with each other. The lifespan of the resulting tuple is exactly those times when this relationship is satisfied.

Let r_1 and r_2 be relations on schemes R_1 and R_2, respectively, where $A \in R_1$ and $B \in R_2$ are attributes.

$$\begin{aligned} r_1[A\theta B]r_2 &= \{x | \exists x_{r_1} \in r_1, \exists x_{r_2} \in r_2 [x.l = \{t | x_{r_1}(A)(t) \theta x_{r_2}(B)(t)\} \wedge \\ &\quad x.v(R1) = x_{r_1}.v(R1)|_{x.l} \wedge \\ &\quad x.v(R2) = x_{r_2}.v(R2)|_{x.l}]\} \\ &\equiv [x_1.*, x_2.* : t] r_1(x_1) \wedge r_2(x_2) \\ &\quad \wedge t \in (x_1.l \cap x_2.l) \wedge x_1.A(t) \theta x_2.B(t) \end{aligned}$$

5. **Static Time-Slice ($\mathcal{T}_{@L}$):** This operator reduces a historical relation in the temporal dimension by restricting the lifespan of each tuple x of the operand relation r to those times in the set of times L.

$$\begin{aligned} \mathcal{T}_{@L}(r) &= \{x | \exists x' \in r [l = L \cap x'.l \wedge x.l = l \wedge x.v = x'.v|_l]\} \\ &\equiv [x.* : t] r(x) \wedge t \in (L \cap t.l) \end{aligned}$$

6. **Dynamic Time-Slice ($\mathcal{T}_{@A}$):** The dynamic time-slice also reduces a relation in the temporal dimension, and is applicable to relations

that include in their scheme an attribute A whose domain consists of partial functions from the set of times into itself. Under this operation the lifespan of each tuple t in the operand relation is reduced to those times that also occur in the range of values of its attribute A.

$$\begin{aligned} \mathcal{T}_{@A}(r) &= \{x | \exists x' \in r[\text{for L, the image of } x.A, x.l = L \wedge x = x'|_L]\} \\ &\equiv [x.* : t]r(x) \wedge t \in x.l \wedge \exists t_1(t_1 \in x.l \wedge t = x.A(t_1)) \end{aligned}$$

Other Operators In addition to the above categories of operators, the **HRDM** algebra includes several of what we term *structural* operators because they are used to restructure a relation without changing the information content of that relation. Each of these operators, **union-merge** ($\cup_o$), **intersection-merge** ($\cap_o$), and **difference-merge** ($-_o$), first computes the set-theoretic operator indicated by their prefix, and then in the resulting relation combine into a single tuple several tuples that, based on their key values, denote the same entity. Various types of structural operators are often found in extensions to the relational data model, historical or otherwise.

The **HRDM** algebra also includes an operator **WHEN**. We categorize this operator as an *extra-relational* operator in that it computes a result that is not contained in a database relation, nor given as a constant. Applied to a historical relation, this operator returns a value defined as the union of the lifespans of the tuples in that relation. This operator can be viewed

as a type of temporal-based *aggregate* operator.

We omit such operators from our discussion of completeness of **HRDM** and the remaining languages that we will discuss since in the case of the structural operators they are not intended for querying, and in the case of the extra-relational operator because they generally fall within the realm of other notions of completeness. (We make reference to other such notions in Section 5.)

The translations that we have provided for each of the relation-defining operators of the **HRDM** algebra shows that this algebra is bounded by the language L_h. However, this historical algebra is not complete in that there are queries that are expressible in L_h for which no equivalent algebraic expression (i.e., sequence of algebraic operations) exists. One example is the query on the database in Figure 2 for the name and department of each employee that has at some time received a cut in salary, expressible in L_h as

$$[e.NAME, e.DEPT : t]\mathbf{EMPL}(e) \wedge t \in e.l \wedge$$
$$\exists t_1 \exists t_2 (t_1 < t_2 \wedge e.SAL(t_1) > e.SAL(t_2))$$

The lack of an equivalent algebraic expression is due to the specification of those operators that include the comparison of two values as part of their definition: the join, and the various select operators. In each case only attribute values that occur at the same point in time can be compared. Thus, as required by the above query, it is not possible to compare the

salary of an employee at some time t_1 with that employee's salary at some other point in time, t_2.

4.2 TQuel

TQuel is the query language component of an historical relational data model proposed by Snodgrass [Sno87]. We shall call this model **TRDM**[2] in order to distinguish it from the previously defined **HRDM**.

TRDM provides for two types of historical relations. One, called an **interval** relation (Figure 3) is derived from a standard relation through the addition of two temporal attributes, *valid-from* and *valid-to*, both of whose domains are the set of times T. The values of the other attributes of a tuple in such a relation are considered to be valid during the beginning of the interval of time starting at the *valid-from* value and ending at, but not including, the *valid-to* value. (This interval thus denotes the lifespan of the tuple.)

The second type of relation, an **event** relation is defined by extending a standard relation by a single temporal attribute *valid-at*. Since both interval relations and event relations are derived from first normal form relations through the addition of attributes whose values are atomic, they are also in first normal form. Although our discussion of TQuel and **TRDM** con-

[2]The reason for the specific choice of **TRDM** is that this model is part of a larger model that Snodgrass refers to as a **temporal** because of its ability to also accomodate a second temporal component called **transaction time**.

EMPL

NAME	*DEPT*	*SALARY*	*valid-from*	*valid-to*
Tom	Sales	20K	0	7
Tom	Sales	30k	7	10
Tom	Mktg	30k	10	11
Tom	Mktg	27k	11	*now*
Juni	acctng	28k	5	*now*
Ashley	engrng	27k	2	5
Ashley	engrng	30k	5	10
Ashley	engrng	30k	14	*now*

Figure 3: A TRDM Interval Relation

siders only interval relations, it can be extended easily to also cover event relations.

The query language TQuel is a calculus-based language that is derived from and defined as a superset of Quel, the query language of the INGRES relational database management system [SWKH76], through the addition of temporal-based clauses that accomodate the *valid-from* and *valid-to* attributes. (These attributes are not visible to the existing components of the Quel language.)

A *WHEN* clause is added to define an additonal temporal-based selection constraint that must be satisfied in conjunction with the constraint defined by the TQuel (and Quel) *WHERE* clause. This constraint, specified as a temporal predicate over a set of tuple *valid-from-valid-to* intervals (lifespans) defines a restricted set of relationships that must hold amongst

them.

A *VALID* clause is used to define, in terms of temporal expressions, *valid-from* and *valid-to* values for tuples in the relation resulting from the TQuel statement.

As Snodgrass shows [Sno87], both temporal predicates and temporal expressions have a semantics that is expressible in terms of the standard tuple calculus.[3] Since these same temporal predicates and temporal expressions are defined in terms of intervals that denote lifespans they can also be represented using the temporal constants and variables of the more expressive language L_h, implying that TQuel is bounded by L_h.

The completeness, and thus expressiveness, of TQuel can be viewed in terms of that of Quel. In addition, the expressiveness of TQuel is dependent on the type of domains over which non-temporal attributes are defined, and, as we discuss below, by extension, the existence of event relations.

TQuel is bounded by the language L_h since the semantics of TQuel like that of Quel [Ull88] can be expressed in terms of, and is thus bounded by, the standard relational calculus which in turn is bounded by L_h. In particular, any TQuel query can be expressed as a formula of the form $Q \wedge \Gamma \wedge \Phi$ where Q, Γ, and Φ are the calculus formulae of the underlying Quel statement, the TQuel *WHEN* clause and *VALID* clause, respectively, and Γ and Φ contain no quantifiers. Additionally, Γ and Φ are defined only

[3]This specification also includes the use of several auxillary functions that are used to compare times in order to determine which of two times occurs first or last.

over the temporal attributes *valid-from* and *valid-to*, neither of which may be included in Q. The structure of this formula means that, as with Quel, not all algebraic expressions can be expressed as a single TQuel statement (for example, algebraic expressions containing the union operator).

If none of the non-temporal attributes over which a **TRDM** database is defined has a domain whose values are comparable to those in the set of times T, then in no algebraic expression over the relations in this database can such an attribute be compared to either *valid-from* or *valid-to*. For such a database, TQuel statements, as represented by a defining tuple calculus formula, are no more restrictive than Quel statements, and as with Quel a sequence of TQuel statements, by creating temporary relations, and using operators such as *APPEND* and *DELETE*, can express any algebraic expression.

However, if some attribute A is defined over some subset of times, then there exists some algebraic expressions for which no sequence of TQuel statements can be equivalent; while an algebraic expression can compare attribute A to either attribute *valid-from* or *valid-to*, a relational calculus expression derived from a TQuel statement cannot.

This problem is remedied in **TRDM** through the use of event relations. Rather than including a temporal attribute, other than *valid-from* and *valid-to*, in an interval relation, it can be moved to a separate relation, along with the apropriate key attributes, and renamed to the temporal attribute *valid-at*. Such a change will allow it to be included in TQuel

temporal expressions along with the temporal attributes *valid-from* and *valid-to*.

Although interval relations and event relations are distinguished by TQuel, they are standard first nornal form relations that provide a fixed way of encoding temporal data using the temporal attributes. TQuel differs from Quel only in the distinction accorded these attributes. Thus, like Quel it is, with the addition of such operators as *APPEND* complete in the sense defined by Codd, and by extension it is, as a result of the use of the temporal attributes, historically complete. The cost of achieving this historically completeness is the need to define two intrinsically different relation types, and to include more relations in a database than would otherwise be necessary; for example, as with HRDM or the historical model that we describe in the next section. In addition it should be noted that since TQuel relations are in first normal form, it requires several tuples to represent a single entity having attributes whose values vary over time.

4.3 The Historical Model of Gadia

The third historical data model that we discuss is one that was proposed by Gadia [Gad86]. This data model, which we shall label TDMG, defines a historical relation that is the same as that of HRDM, and thus the canonical historical relation defined in Section 2.1. Here too the value of a tuple attribute is a function from a set of times to the value domain of the attribute.

Corresponding to this historical relation Gadia defines two query languages that he shows to be equivalent: one a historical algebra, the other a historical calculus. In the following discussion we consider only the algebra. Again we partition the operators into the three categories: set-theoretic, attribute based, and other. We discuss only the individual attribute based operators; the set-theoretic operators being, similar to those of **HRDM**, are expressible in the language L_h in the obvious way, and the operators in the category other, not being relevant.

Attribute Based The algebra of TDMG defines historical equivalents to the project, select and join operators of the standard relational model. The differences, where they exist, between these operators and their standard relational counterparts arise from the need to accomodate the use of temporal-based functions as tuple attribute values. In addition to these operators TDMG defines a *temporal selection* operator that is a variant of the time-slice operator that we discussed in Section 4.1. Below we define each of these operators in terms of expressions of the language L_h.

1. Project:

$$\pi_X(r) \equiv [x.X : t]r(x) \wedge t \in x.l$$

2. Select: The select operation $\sigma_{A\theta B}(r)$ differs from its standard relational counterpart in the interpretation of the selection criteria. This criteria, where A is an attribute, B is an attribute (or constant), and

θ is a comparator of the appropriate type, is satisfied by a tuple x in r if for all times t in its lifespan $x.A(t)\theta x.B(t)$ (or if B is a constant, $x.A(t)\theta B$).

$$\sigma_{A\theta B}(r) \equiv [x.* : t]r(x) \wedge t \in x.l \wedge \forall t_1(t_1 \in x.l \rightarrow x.A(t)\theta x.B(t))$$

(If B is a constant, then $x.B(t)$ is replaced by B in the above definition.)

3. Join: As expected, the join operation $r_1[A_1\theta A_2]r_2$ defines a relation, each tuple of which is formed by joining a tuple from relation r_1 with a tuple from r_2. Two tuples are joined if and only if they have the same lifespan, and throughout each time in their common lifespan the value of attribute A_1 of the r_1 tuple satisfies the relationship specified by θ with the value of attribute A_2 of the tuple from r_2

$$\begin{aligned} r_1[A_1\theta A_2]r_2 \equiv\ & [x_1.*, x_2.* : t]r_1(x_1) \wedge r_2(x_2) \wedge t \in x.l \wedge \\ & x_1.l = x_2.l \wedge \forall t_1(t_1 \in x_1.l \rightarrow x_1.A_1(t_1)\theta x_2.A_2(t_1)) \end{aligned}$$

A second join operator is also defined in the algebra. However, since it is a more restrictive variant of the above we will not discuss it here.

4. Temporal Selection (Timeslice): The temporal selection $r(e)$ defines a relation, each tuple of which is derived from a tuple in r by restricting its lifespan to those times that are included in the set of times defined

by the temporal expression e. The temporal expressions allowed are all included among those definable in the language L_h.

$$r(e) \equiv [x.* : t]r(x) \wedge t \in (e \cap x.l)$$

As was the case with the historical model HRDM, TDMG is bounded by the language L_h since each of its operators are definable in terms of expressions of L_h. Similarly, as can be seen by the description and definition of the operators, the TDMG algebra lacks the expressiveness of L_h since it is not possible to compare the value of one attribute at a time t_1 with the value of another or the same attribute at some other time t_2. Thus, it too is incomplete with respect to L_h.

5 Summary and Conclusions

In this paper we have defined a concept of relational completeness for historical relational databases. This notion of completeness, analogous to completeness in the standard relational data model provides a standard against which the power of various of historical query languages can be compared.

The basis for our notion of completeness is the language L_h, a historical relational calculus. This language is defined in terms of a database of canonical historical relations that are specified so as to reflect the intent of what is generally meant by a historical database. That is, these relations

are able to model a set of *"things"* (e.g., entities and relationships) that change over time, retaining both currently valid data, as well as data that was valid at some time in the past.

After discussing the reasonableness of the historical relational calculus as a basis for historical relational completeness we then use it in a discussion of the completeness of several proposed query languages: the historical algebras proposed by Gadia, and Clifford and Croker, and the historical calculus proposed by Snodgrass. Each language was found to be bounded by the the language L_h. Only one of the languages, TQuel, was found to be as expressive as L_h. However, two different types of relations, interval relations and event relations, were used to achieve this expressiveness.

In keeping with the analogy with the standard relational calculus, the notion of historical relational completeness that we describe is limited. For example, it too is not sufficient for expressing an historical variant of transitive closure. We expect, therefore, that historical analogs to the notions of completeness defined by Bancilhon [Ban78] and Chandra and Harel [CH80] can also be defined, with the later being sufficient to describe aggregate-type operations on historical databases.

Completeness is only one of a number of criteria that can be used to evaluate and compare historical relational query languages. McKenzie and Snodgrass [MS87] discuss several other criteria that they then use to evaluate various historical relational algebras.

References

[Ari86] G. Ariav. A temporally oriented data model. *ACM Transactions on Database Systems*, 11(4):499–527, December 1986.

[Ban78] F. Bancilhon. On the completeness of query languages for relational databases. In *Proc. Seventh Symposium on Mathematical Foundations of Computing*, pages 112–123, Springer-Verlag, 1978.

[Ben82] J. Ben-Zvi. *The Time Relational Model.* PhD thesis, University of California at Los Angeles, 1982.

[CC87] J. Clifford and A. Croker. The historical relational data model HRDM and algebra based on lifespans. In *Proc. Third International Conference on Data Engineering*, pages 528–537, IEEE, Los Angeles, February 1987.

[CH80] A.K. Chandra and D. Harel. Computable queries for relational data bases. *Journal of Computer and System Sciences*, 21(2):156–178, October 1980.

[Che76] P.P.-S. Chen. The entity-relationship model – toward a unified view of data. *ACM Transactions on Database Systems*, 1(1):9–36, March 1976.

[Cod72] E.F. Codd. Relational completeness of data base sublanguages. In R. Rustin, editor, *Data Base Systems*, Prentice-Hall, 1972.

[Gad86] S.K. Gadia. Toward a mulithomogeneous model for a temporal database. In *Proc. Second International Conference on Data Engineering*, IEEE, Los Angeles, California, February 1986.

[Mai83] D. Maier. *The Theory of Relational Databases.* Computer Science Press, 1983.

[MS87] E. McKenzie and R. Snodgrass. *An Evaluation of Historical Algebras.* Technical Report TR87-020, University of North Carolina at Chapel Hill, October 1987.

[Sno87] R. Snodgrass. The temporal query language tquel. *ACM Transactions on Database Systems*, (12):2, June 1987.

[SWKH76] M. Stonebraker, E. Wong, P. Kreps, and G. Held. The design and implementation of ingres. *ACM Transactions on Database Systems*, 1(3):189–222, September 1976.

[Ull88] J. Ullman. *Principles of Database and Knowledge-Base Systems.* Volume 1, Computer Science Press, 1988.