ON COMPLETENESS OF HISTORICAL RELATIONAL QUERY LANGUAGES

by

James Clifford
Information Systems Department
Leonard N. Stern School of Business
New York University

Albert Croker
Statistics and Computer Information Systems
Baruch College
City University of New York

and

Alexander Tuzhilin
Information Systems Department
Leonard N. Stern School of Business
New York University

December, 1991

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

Working Paper Series

STERN IS-91-41

This replaces CRIS #192.

Abstract

Numerous 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 of these temporal models, and consequently into the extended relational *algebra* or *calculus* that they define. Because of these differences it has been difficult to compare the proposed models and to make judgments as to which of them might in some sense be equivalent or even *better*. In this paper we define the notions of **temporally grouped** and **temporally ungrouped** historical data models and propose two notions of **historical relational completeness**, analogous to Codd's notion of relational completeness, one for each type of model. We show that the temporally ungrouped models are less powerful than the grouped models, but demonstrate a technique for extending the ungrouped models with a grouping mechanism to capture the additional semantic power of temporal grouping. For the ungrouped models we define three different languages, a temporal logic, a logic with explicit reference to time, and a temporal algebra, and show that under certain assumptions all three are equivalent in power. For the grouped models we define a many-sorted logic with variables over ordinary values, historical values, and times. Finally, we demonstrate the equivalence of this grouped calculus and the ungrouped calculus extended with the proposed grouping mechanism. We believe the classification of historical data models into grouped and ungrouped provides a useful framework for the comparison of models in the literature, and furthermore the exposition of equivalent languages for each type provides reasonable standards for common, and minimal, notions of historical relational completeness.

1 Introduction

Over the course of the past decade various historical relational data models have been proposed, including [JM80], [BZ82], [CW83], [Ari86], [Tan86], [CC87], [Lor87], [NA87], [Sno87], [Gad88], [Sar90].[1] These data models are intended for those situations where there is a need for managing data as it changes over time. Generally, these data models *extend* the standard relational data model by including a temporal component. This incorporation of the temporal dimension has taken a number of different forms. Chief among these have been the addition of an additional attribute, say *TIME*, to a relation (the equivalence of time-stamping) [Sno87], or the inclusion of time as a more intrinsic part of the *structure* of a relation [CC87, Gad86]. The latter approach results in what have been called **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 it has remained a subject for debate whether they have the same modeling capabilities. Moreover, because the query languages defined in these data models differ from each other in their formulations, it has remained unclear whether they provide the same capabilities for extracting various subsets of a database. So many different languages have appeared in the literature, in fact (e.g., [MS89] refers to no fewer than eleven algebras alone) that it is crucial to have some standard measure against which to compare them.

In this paper we address the issue of *completeness* for historical relational data models. A metric of **historical relational completeness** can provide a basis for determining the *expressive power* of the query languages that have been defined as part of proposed historical relational data models. As such, the notion of historical relational completeness can serve 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] and [CH80].

In Section 2 we first address the issue of the modeling capability of the various historical

[1] This list is not exhaustive; for an overview of the area of time and databases see [AC86] and [Sno90]; for an ongoing bibliography on the subject see [McK86], [SS88] and [Soo91].

data models that have been proposed. In particular we explicate the different modeling capabilities achieved by incorporating the temporal dimension at the tuple level (by time-stamping each tuple) or at the attribute-value level (by including time as part of each value). We introduce the terms **temporally ungrouped** and **temporally grouped** to distinguish between these two approaches, respectively, and discuss the relative power of the two approaches. We then propose two different canonical models to serve as the basis for our analysis of the power of query languages for these two approaches. The distinction between these two different types of models, temporally ungrouped and temporally grouped, serves to structure the remainder of the paper. Essentially, we define two separate notions of completeness – one for each of these two types of models – and then discuss the relationship between the two notions.

In Section 3 we examine the **temporally ungrouped models**, and define three different languages for them: a temporal logic, a logic with explicit reference to time, and a temporal algebra. We show that under certain assumptions about the temporal universe all three are equivalent in power. We propose these languages as a standard for completeness for temporally ungrouped models. In Section 4 we examine the **temporally grouped models**, and define a historical relational calculus for them; this calculus is a many-sorted logic with variables over ordinary values, historical values, and times. We propose this calculus as a standard of completeness for models of this type. We then show in Section 5 how the representation power of the ungrouped models and their languages can be extended to incorporate the *grouping* semantics. Finally, in Section 6 we examine the completeness of several historical relational languages that have been proposed in the literature with respect to these metrics.

It is worth pointing out that there are a number of additional issues which might reasonably be said to be related to the question of *completeness* of query languages but which are necessarily outside of the scope of this paper. We are limiting our attention to models which incorporate a single dimension of time (*historical*, as opposed to *temporal* models, in the terminology of [SA85]), but we believe that these results could be extended to handle additional time dimensions. Furthermore, in the spirit of most of the work on completeness

for standard relational languages, we do not address the issue of temporal aggregates (as, for example, in [SGM87]). Work in the spirit of [Klu82] could extend the results here in that direction if so desired. Finally, we do not incorporate schema evolution over time (as in [CC87]) because this would entail a comprehensive treatment of null values, which is beyond the scope of this paper. For the same reason we limit our attention to *homogeneous* relations ([Gad88]), i.e., relations whose tuples have attributes all defined over the same period of time. In all of these decisions of what to incorporate in our notion of "reasonable" queries, we have been motivated by the desire to choose the common denominator of the various models proposed. In this way we have been be able to apply our metric of completeness fairly against several models in the paper.

We conclude in Section 7 with a summary of our results and some directions for future research.

2 Temporally Grouped and Temporally Ungrouped Data Models

Two different strategies for incorporating a temporal dimension into the relational model have appeared in the literature. In one, the schema of the relation is expanded to include one or more *distinguished* temporal attributes (e.g., *START-TIME* and *END-TIME*) to represent the period of time over which the *fact* represented by the tuple is to be considered valid. This approach has been referred to in the literature as *tuple time-stamping* or as a *first-normal form (1NF)* model. In the other approach, referred to as *attribute time-stamping* or as a *non-first-normal form (N1NF)* model, instead of adding additional attributes to the schema, the domain of each attribute is extended from simple values to complex values (functions, e.g.) which incorporate the temporal dimension Both [CC87] and [Sno90] contrast these two approaches.

Consider, as an example, a relation intended to record the changing departmental and salary histories of employees in an organization[2]. Figures 1 and 2 show typical representa-

[2]Similar examples have appeared in [CW83], [Gad86] and [Sno87]

EMPLOYEE			
NAME	*DEPT*	*SALARY*	*time*
Tom	Sales	20K	0
Tom	Finance	20K	1
Tom	Finance	20K	2
Thomas	MIS	27K	3
Jim	Finance	20K	1
Jim	MIS	30K	2
Jim	MIS	40K	3
Scott	Finance	20K	1
Scott	Sales	20K	2

Figure 1: Prototypical 1NF Historical Employee Relation

tions in these two approaches. While both relations appear to have the same information content, i.e., the same data about three different employees over the same period of time, the models represent this information in quite different ways. In the 1NF approach (Figure 1) each moment of time relevant to each employee is represented by a separate tuple which carries the time stamp. In the N1NF approach (Figure 2), each employee's entire history is represented within a single tuple, within which the time stamps are embedded as *components* of the values of each attribute.

In each of the so-called N1NF models (e.g.,[Tan86, CC87, Gad88]) *all of the information* about each employee is represented in a single tuple; in the 1NF models that have been proposed (e.g.,[Ari86, Sno87, Lor87, TC90]), this property does not hold. Also note, with respect to the N1NF models, that while in general a key field like *NAME* would typically be *constant* over time, there is no requirement that this be the case. For example, in the **EMPLOYEE** relation in Figure 2 the employee *Tom* changes his name to *Thomas* at time 3. There are many applications where the value of a key need not be constant over time, but merely unique in the relation at any given time.

While N1NF models inherently *group* related facts into a single tuple, 1NF models, whether historical or temporal (using the distinction in [SA85]) for models with one or two time dimensions, respectively) are problematic in this regard, as Figure 1 makes clear. Such

EMPLOYEE			
NAME	*DEPT*	*SALARY*	*lifespan*
0 → Tom	0 → Sales	0 → 20K	
1 → Tom	1 → Finance	1 → 20K	
2 → Tom	2 → Finance	2 → 20K	
3 → Thomas	3 → MIS	3 → 27K	{0, 1, 2, 3}
1 → Jim	1 → Finance	1 → 20K	
2 → Jim	2 → MIS	2 → 30K	
3 → Jim	3 → MIS	3 → 40K	{1, 2, 3}
1 → Scott	1 → Finance	1 → 20K	
2 → Scott	2 → Sales	2 → 20K	{1, 2}

Figure 2: Prototypical N1NF Historical Employee Relation

models provide no inherent grouping of the tuples that represent the same *object*[3]; for instance, they do not group the tuples of the same employee (Jim, e.g.) in Figure 1. As we shall see, it is up to the users to know and to maintain that grouping in all of their interactions with the database.

We point out that another technique for time-stamping tuples (or values) that has appeared in the literature (e.g., [Sno87, Lor87]) uses a time-interval rather than a time-point as the time-stamp. For example, the *VALID-TIME (from)* and *(to)* attributes in Figure 3 denote a time interval.It is well-known that if time is *discrete*) , then these two approaches are equivalent[vB83]. Nearly all of the work on historical or temporal databases has assumed a discrete temporal domain (McKenzieSnodgrass89). We will therefore utilize the two representation schemes interchangeably.

Although in this paper we are concerned only with the issue of completeness of query languages for *historical* data models, it is worth pointing out that the same grouping problem occurs in *temporal* models, as the prototypical representation in Figure 3 makes clear. In these models the tuples are stamped, not merely with the time period during which the *fact* that they represent held true in reality (*VALID-TIME*), but also with another time stamp representing the time period during which the database *knows* of the fact (*TRANS-*

[3]We will use the term *object* occasionally in the paper. We use it in a completely neutral sense, and not as a reference to objects in any object-oriented paradigm with all of the implications thereof

EMPLOYEE						
			VALID-TIME		*TRANS-TIME*	
NAME	*DEPT*	*SALARY*	*(from)*	*(to)*	*(start)*	*(stop)*
Tom	Sales	20K	0	1	t_1	∞
Tom	Finance	20K	1	2	t_2	∞
Tom	Finance	20K	2	3	t_3	∞
Thomas	MIS	27K	3	4	t_4	∞
Jim	Finance	20K	1	2	t_5	∞
Jim	MIS	30K	2	3	t_6	∞
Jim	MIS	40K	3	4	t_7	∞
Scott	Finance	20K	1	2	t_8	∞
Scott	Sales	20K	2	3	t_9	∞

Figure 3: Prototypical 1NF Temporal Employee Relation

TIME). We do not treat such relations in this paper, but believe that our results could (and should) be extended to address them.

Because the term N1NF has been used elsewhere to refer to various kinds of relaxations of the 1NF Property including, among other things, models which allow nested relations or set-valued attributes, we prefer to use the terms **Temporally Grouped** and **Temporally Ungrouped** for these two types of models. In the sequel, therefore, we will use the term **Temporally Grouped (TG)** to refer to models which provide built-in support for the grouping of related temporal values, and **Temporally Ungrouped (TU)** for those which do not.

In the rest of this section we will precisely define two canonical models, one ungrouped and the other grouped. These models will first be informally contrasted, and then will be used in the remainder of the paper to provide the basis for our definitions of temporally grouped and temporally ungrouped completeness.

2.1 A Canonical Temporally Ungrouped Relation Structure

The structure for relations in temporally ungrouped data models is essentially a straightforward extension of the standard relational structure.

Let $U_D = \{D_1, D_2 \ldots, D_{n_d}\}$ be a (universal) set of **value domains** (i.e., each D_i is a set of values), where for each i, $D_i \neq \emptyset$. $\mathbf{D} = \bigcup_{i=1}^{n_d} D_i$ is 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 comparators "=" and "$\neq$" to test for the equality and inequality, respectively, of any two elements of the associated value domain.

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. Moreover, there is a distinguished attribute *TIME*, not in U_A, which will be used to represent temporal information.

A **temporally ungrouped historical relation scheme** $\mathbf{R_{TU}}$ is a 4-tuple $\mathbf{R_{TU}} =< \mathbf{A}, \mathbf{T}, \mathbf{K}, \mathbf{DOM} >$ where:

1. $\mathbf{A} \cup \{TIME\}$ is the set of attributes of scheme $\mathbf{R_{TU}}$, where $\mathbf{A} \subseteq U_A$; the attributes in $\mathbf{A}$ are called **value attributes**, and $TIME$ is the **temporal attribute**.
2. $\mathbf{T} = \{t_0, t_1, \ldots, t_i, \ldots\}$ is a non-empty set, the set of **times**, and $<$ is a total order on $\mathbf{T}$. The cardinality of $\mathbf{T}$ is restricted to be at most countably infinite.
3. The set $\mathbf{K} \cup \{TIME\}$, where $\mathbf{K} \subseteq \mathbf{A}$, is the **key** of scheme $\mathbf{R_{TU}}$, i.e. $\mathbf{K} \cup \{TIME\} \rightarrow \mathbf{A}$.
4. $\mathbf{DOM} : \mathbf{A} \cup \{TIME\} \rightarrow U_D \cup \{T\}$ is a function that assigns to each value attribute of scheme $\mathbf{R_{TU}}$ a value domain, denoted $DOM(A_i, \mathbf{R_{TU}})$, and to $TIME$ the temporal domain $\mathbf{T}$.

A **temporally ungrouped historical relational database scheme** $\mathbf{DB_{TU}} = \{R_{1_{TU}}, R_{2_{TU}}, \ldots, R_{n_{TU}}\}$ is a finite set of temporally ungrouped historical relation schemes.

A **temporally ungrouped historical tuple** $\mathbf{t_{TU}}$ on scheme $\mathbf{R_{TU}} =< \mathbf{A}, \mathbf{T}, \mathbf{K}, \mathbf{DOM} >$ is a function that associates with each attribute $A_i \in \mathbf{A}$ a value in $\mathbf{DOM}(A_i, \mathbf{R_{TU}})$ and to $TIME$ a value in $\mathbf{T}$.

A **temporally ungrouped historical relation** r on scheme $\mathbf{R_{TU}} =< \mathbf{A, T, K, DOM} >$ is a finite set of ungrouped historical tuples on scheme $\mathbf{R_{TU}}$ that satisfies the key constraint.

A **temporally ungrouped historical database** $\mathbf{d_{TU}} = \{\mathbf{r_{1_{TU}}}, \mathbf{r_{2_{TU}}}, \ldots, \mathbf{r_{n_{TU}}}\}$ is a set of temporally ungrouped historical relations where each $\mathbf{r_{i_{TU}}}$ is defined on a (not necessarily unique) ungrouped historical relation scheme $\mathbf{R_{i_{TU}}}$.

The **EMPLOYEE** relation in Figure 1 is a typical relation in the temporally ungrouped historical data model.

2.2 A Canonical Temporally Grouped Relation Structure

As a basis for the specification of our notion of historical completeness for temporally grouped temporal relations, we begin by defining a canonical temporally grouped historical relation upon which we will base the calculus that we define in the next section. The structure of this relation is specified in such a way as to capture the intent and requirements of a temporally grouped historical relation, and to be general enough to have the representational capabilities of other proposed historical relations.

Let U_D, $\mathbf{D}$, Θ_{D_i}, and U_A be as for the canonical temporally ungrouped relation structure (Section 2.1).

$\mathbf{T}$ will designate the set of **times** in the model, and any subset $\mathbf{L} \subseteq \mathbf{T}$ is called a **lifespan.** (Note, therefore, that a lifespan can consist of several, non-contiguous *intervals* of time.) Corresponding to each value domain D_i is a **temporal domain** ${D_i}^T$ of partial temporal-based functions from the set of times $\mathbf{T}$ to the value domain D_i. Each of these partial functions defines an association between each time instance in some lifespan L, and a value in a designated domain, and thus provides a means of modeling the changing of an attribute's value over time.

A **temporally grouped historical relation scheme** $\mathbf{R_{TG}}$ is a 4-tuple $\mathbf{R_{TG}} =< \mathbf{A, T, K, DOM} >$ where:

1. $\mathbf{A} \subseteq U_A$ is the set of **attributes** of scheme $\mathbf{R_{TG}}$

2. $\mathbf{T} = \{t_0, t_1, \ldots, t_i, \ldots\}$ is a non-empty set, the set of **times**, and $<$ is a total order on $\mathbf{T}$. The cardinality of $\mathbf{T}$ is restricted to be at most countably infinite.

3. $\mathbf{K} \subseteq \mathbf{A}$ is the **key** of scheme $\mathbf{R_{TG}}$, i.e., $\mathbf{K} \rightarrow \mathbf{A}$

4. $\mathbf{DOM} : \mathbf{A} \rightarrow U_D \cup \{T\}$ is a function that assigns to each attribute of scheme $\mathbf{R_{TG}}$ a value domain, and, by extension, the corresponding temporal domain. We denote the domain of attribute A_i in scheme $\mathbf{R_{TG}}$ by $\mathbf{DOM}(A_i, \mathbf{R_{TG}})$.

A **temporally grouped historical relational database scheme** $\mathbf{DB_{TG}} = \{\mathbf{R_{1_{TG}}}, \mathbf{R_{2_{TG}}}, \ldots, \mathbf{R_{n_{TG}}}\}$ is a finite set of temporally grouped historical relation schemes.

A **temporally grouped historical tuple** $\mathbf{t_{TG}}$ on scheme $\mathbf{R_{TG}} =< \mathbf{A}, \mathbf{T}, \mathbf{K}, \mathbf{DOM} >$ is a function that associates with each attribute $A_i \in \mathbf{A}$ a temporal-based function from the **tuple lifespan** (any subset of $\mathbf{T}$) common to the tuple, denoted $\mathbf{t_{TG}.l}$, to the domain assigned to attribute A_i. That is, $\mathbf{t_{TG}}(A_i) : \mathbf{L} \rightarrow \mathbf{DOM}(A_i)$. (We note that it is also possible to associate lifespans with attributes [CC87]; a treatment of this is beyond the scope of this paper. Doing so permits historical relation *schemes* to accommodate changes that may occur to them over time.)

A **temporally grouped historical relation** $\mathbf{r_{TG}}$ on scheme $\mathbf{R_{TG}} =< \mathbf{A}, \mathbf{T}, \mathbf{K}, \mathbf{DOM} >$ is a finite set of temporally grouped historical tuples on scheme $\mathbf{R_{TG}}$ such that given any two tuples $\mathbf{t_{1_{TG}}}$ and $\mathbf{t_{2_{TG}}}$ in $\mathbf{r_{TG}}$, $\forall s_1 \in (\mathbf{t_{1_{TG}}.l} \cap \mathbf{t_{2_{TG}}.l})\ \exists A_i \in \mathbf{K}$ such that $\mathbf{t_{1_{TG}}}(A_i)(s_1) \neq \mathbf{t_{2_{TG}}}(A_i)(s_1)$. This notion of a key simply specifies that there can be no time in which two different tuples agree on the key. Although in general we would assume that the temporal-based function associated with each key attribute of a tuple would be constant with respect to the lifespan of that tuple, we do not require it to be so. Note that the **EMPLOYEE** relation in Figure 2, with three tuples, is an example of a temporally grouped historical relation.

A **temporally grouped historical database** $\mathbf{d_{TG}} = \{\mathbf{r_{1_{TG}}}, \mathbf{r_{2_{TG}}}, \ldots, \mathbf{r_{n_{TG}}}\}$ is a set

of temporally grouped historical relations where each $\mathbf{r}_{i_{TG}}$ is defined on a (not necessarily unique) temporally grouped historical relation scheme $\mathbf{R}_{i_{TG}}$.

In the sequel we will generally omit the subscripts **TG** and **TU** when no confusion over the type of model will result.

2.3 Comparison of Grouped and Ungrouped Models

Many researchers have assumed that these two different approaches to incorporating a temporal dimension into the relational data model – the temporally grouped and the temporally ungrouped approaches – were equivalent in power, the differences simply a matter of style[4]. In exploring the issue of *completeness* for historical databases, however, we had to try to reconcile these two different approaches, and in doing so came to the conclusion that they are *not* equivalent. Gadia in [Gad88] addresses this issue of grouping (without using the term), when he discusses the relationship between his homogeneous model, a grouped model, and what he calls a *snapshot valued function* which is essentially a *corresponding* ungrouped model. However, rather than emphasizing the importance of the *differences* between these two approaches, he concludes by showing that they are only *weakly equivalent.* Essentially he shows that you can (trivially) take a grouped relation and ungroup it, but that for an ungrouped relation there is not a unique grouped relation, and hence his equivalence is weak.

What we will argue in the rest of this section is that the differences *are* important, and the modeling capabilities are not the same. In subsequent sections we shall define precisely a notion of completeness for each of the two approaches, and then compare them formally. Finally, we will show how by adding grouping mechanism to the ungrouped model there is a (strong) equivalence between the two models.

The first problem with the ungrouped historical models is that without knowledge of the *key* of the relation there is no way of knowing how to *group* appropriately the facts represented

[4]For example. Snodgrass ([Sno87, pp.264-266]) discusses what he calls the "embedding" of the temporal relation into a flat relation, and informally discusses four techniques for doing so, with the implication that they are all equivalent.

in an arbitrary and unbounded number of tuples. Also, if the key is not required to be *constant* over all times (and there is no reason to require this), there would be no way at all to group related (i.e., describing the same object) tuples! Figure 3 is typical of the figures provided with these models (e.g.,[Sno87, Figure 8]) in that it "begs the question" somewhat by assuming that the key value of an object remains constant over time. Moreover, these figures implicitly sort the tuples by the key field(s). However, since relations are sets, the implicit grouping of the multiple tuples for a given object in these models is in fact being done subliminally for the reader and is *not* supported by these models. A simple listing of the tuples in such a relation is *not* guaranteed to present them in such a nicely ordered fashion.

Another, even more serious problem inherent in these ungrouped models can be seen when we consider the result of the following two queries.

Q1: Give me the salary history of each employee.

Q2: Give me the salary history of each employee, but without identifying the employees to whom they belong.

Q1 poses no additional problems for any of the three models: provided the user knows that *NAME* is the key, the key is constant over time, and the user remembers (or the DBMS is nice enough) to sort the resulting tuples by the key, the interpretation of the tuples in the answer to Q1 is no more problematic than interpreting the tuples in the base relation.

Q2, however, is another matter entirely. First, it is worth noting that this is a very reasonable query and asks simply that the DBMS treat the salary history (temperature history, rainfall history, etc.) as a first-class value that can be queried, manipulated, etc.

The result of the query in the three models is shown in Figure 4. Note that only a temporally grouped model, such as that in Figure 2, respects the integrity of the temporal values of all attributes as first-class objects and therefore yields the answer shown in Figure 4(a). The result of the query in such a model could, for example, be piped to a graphics program to

produce a visual query output such as is shown in Figure 5. Temporally ungrouped models *cannot* support this query, because they do not treat temporal objects as first-class values.

We believe that a model which claims to support the temporal dimension of data should support *temporal values* – i.e., values changing over time. For example, a SALARY should be seen as the *history* of the changing salary values over some time period, and not as a simple scalar value whose time reference is *somewhere else* in the relation. There are two issues here, and they lead to the following definition. A temporal DBMS is said to have **temporal value integrity** if:

1. The integrity of temporal values as first-class objects is inherent in the model, in the sense that the language provides a mechanism (generally, variables and quantification) for direct reference to *value histories* as objects of discourse.

2. Temporal values are considered *identical* only if they are equal for all points in time over which they are defined, and

We categorize models which do not satisfy these properties, such as the so-called 1NF historical data models, as Temporally Ungrouped. In their answer to Q2 (Figures 4(a and b)), Property 1 is violated: instead of showing salary values for three individuals and at nine different moments in time as in Figure 4(a), the TU model incorrectly *equates* Tom, Jim and Scott's salaries between times 1 and 2 and Thomas' and Scott's salaries between times 2 and 3, and discards what it considers *duplicates*, merely because at those particular points in time the salaries happen to have the same values. Property 2 is violated since the tuples in the answer are presented as though they are completely unrelated – which salaries are tied together into which groups? The model does not provide any inherent grouping. The user must therefore always know and demand the *key* in any query, even when, perhaps for security reasons, this is not desired.

In temporally ungrouped models you can never quite take hold of an object like a "salary." You can take pieces of it, but if you try to grab the whole thing and look at it and inspect it, it falls through your fingers moment by moment. Only in a temporally grouped model

SALARY	*lifespan*
$0 \rightarrow$ 20K $1 \rightarrow$ 20K $2 \rightarrow$ 20K $3 \rightarrow$ 27K	$\{0,1,2,3\}$
$1 \rightarrow$ 20K $2 \rightarrow$ 30K $3 \rightarrow$ 40K	$\{1,2,3\}$
$1 \rightarrow$ 20K $2 \rightarrow$ 20K	$\{1,2\}$

(a) Answer in TG Model

SALARY	*time*
20K	0
20K	1
20K	2
27K	3
30K	2
40K	3

(b) Answer in Time Point TU Model

	VALID-TIME	
SALARY	*(from)*	*(to)*
20K	0	1
20K	1	2
20K	2	3
27K	3	4
30K	2	3
40K	3	4

(c) Answer in Time-Interval TU Model

Figure 4: Answers to Q2

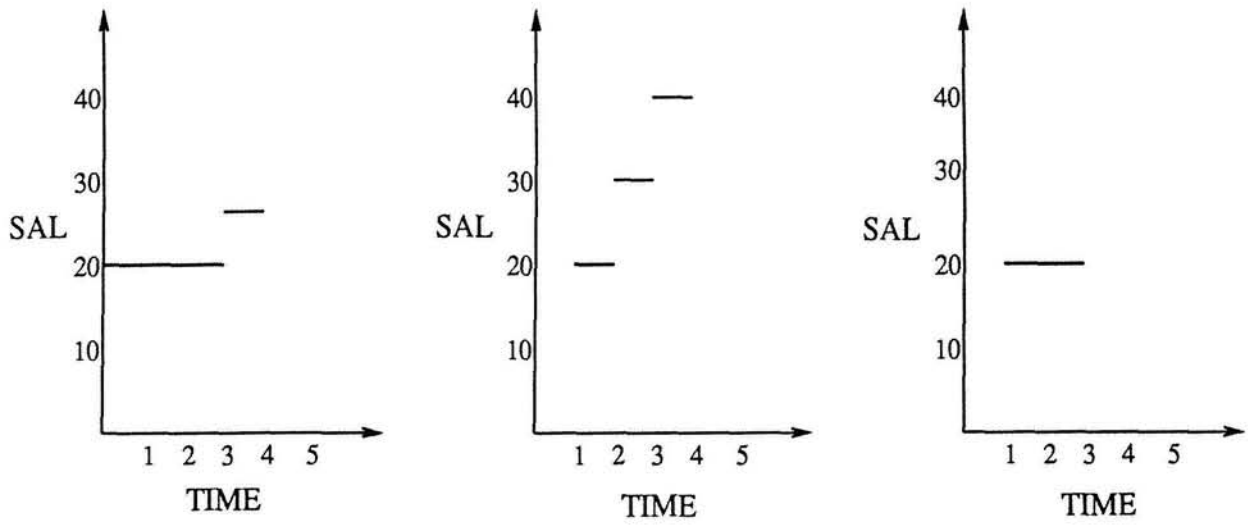

Figure 5: Graph of Employees' Salaries from Figure 4(a?)

is an object like a salary (or the *pricing history* of a stock, or the *average annual rainfall* in Boulder over the last fifty years) a first class object that you can interrogate, examine, dissect, or compare to another salary (or the rainfall in Spokane.) It is really an ontological question of what exists in the model. As Quine put it, "a theory is committed to *those and only those entities* [emphasis ours] to which the bound variables of the theory must be capable of referring" [Qui53, in *On What There Is*]; in temporally ungrouped models temporal entities (like salary histories) *do not exist* because the models and their languages provide no mechanism for referring to them.

We note that the same problem occurs in those ungrouped models (like TQuel [Sno87]) which use two attributes, rather than one, to incorporate the valid time Figure 4(b). Only a Temporally Grouped model, like the one in Figure 2 exhibits **temporal value integrity**, and therefore provides the correct answer to this query, Figure 4(c).

The issue of *completeness* of query languages for a historical relational data model is complicated by the two representation schemes, TG and TU. We are therefore led to define two notions of completeness for historical database query languages, one based on TG models and the other on TU models. We first define TU-completeness and demonstrate the equivalence of 3 different types of query languages for TU models: a temporal logic, a first-order

MANAGEMENT		
MGR	*PROJECT*	*time*
Tom	P1	4
Tom	P2	4
Herb	P2	5
Jim	P3	4
Jim	P3	5
Jim	P3	6

Figure 6: Management TU Historical Employee Relation

logic with explicit temporal variables, and an extended relational algebra ([TC90]). We then define TG-completeness in terms of a calculus which we call L_h. L_h is a natural extension to standard first-order calculus, incorporating two domains (ordinary values and times) and providing each domain with constants, variables, and quantification. Finally, we show how ungrouped models can be extended in a simple way (by adding the grouping mechanism of *group IDs*), so that the two completeness notions, modulo the grouping capacity, are essentially equivalent.

One additional aspect that we will address is the issue of *safety*: which expressions in the language are guaranteed to yield finite answers, and answers that come from data in the database (see, e.g., the description of *domain independence* in [Ull88]). For instance, consider an additional historical relation modeling managers and their projects, as shown in Figure 6. Without some restrictions on the way that time references can be made in a query, it will be possible to ask questions that in effect *create* arbitrary temporal relationships among data items where such relationships do not exist in the database.

For example, in a query language which does not respect *temporal value integrity* the following query can be asked:

Q3: $\{< x, y, t > | \exists t' \; \exists z(\; \mathbf{EMPLOYEE}(x, y, t') \wedge \mathbf{MANAGEMENT}(x, z, t) \;)\}$

This query, here expressed in a temporal calculus (to be described in Section 3.2), could also be expressed in other ungrouped languages such as TQuel ([Sno87]). The answer, as

NAME	*DEPT*	*time*
Tom	Sales	4
Tom	Finance	4
Jim	Finance	4
Jim	Finance	5
Jim	Finance	6
Jim	MIS	4
Jim	MIS	5
Jim	MIS	6
Jim	MIS	4
Jim	MIS	5
Jim	MIS	6

Figure 7: Answer to Unsafe Query Q3

shown in Figure 7, relates employees and departments at times which are clearly nonsensical because this relationship was *created* by the query rather than *extracted* from the data in the database. While such a query is clearly expressible in a language for a model which treats time as *just another attribute*, it seems to us questionable whether the model is incorporating time into the model in any *meaningful* way. This issue will be addressed by our rules for *safe* expressions in historical query languages, which incorporate the view ([AU79]) that query languages should be used for data extraction only.

Temporally grouped models support temporal values directly – they incorporate the temporal component into the historical model at the appropriate level, and provide a means to refer directly to temporal objects. They also *group* together into a single tuple all of the facts about an object over time. In Sections 3 and Sections 4 we will show that the TG representation is more expressive than the TU representation. Thus we can state that merely time-stamping tuples in the database, as attractive as its simplicity might make it, is not sufficient to adequately incorporate a temporal dimension into the database.

Because the values of many tuples, or their attributes, frequently do not change over long periods of time, it is often convenient to adopt a shorthand notation for temporally grouped relations. Figure 8 represents a TG Historical Database using this shorthand

EMPLOYEE			
NAME	*DEPT*	*SALARY*	*lifespan*
$[0,5) \rightarrow$ Tom	$[0,4) \rightarrow$ Sales	$[0,3) \rightarrow$ 20K	
	$[4,6] \rightarrow$ Mktg	$[3,5) \rightarrow$ 30K	
$[5,6] \rightarrow$ Thomas		$[5,6] \rightarrow$ 27K	$\{0,1,2,3,4,5,6\}$
$[2,6] \rightarrow$ Juni	$[2,6] \rightarrow$ Acctng	$[2,6] \rightarrow$ 28K	$\{2,3,4,5,6\}$
$[1,4) \rightarrow$ Ashley	$[1,3) \rightarrow$ Engrng	$[1,2) \rightarrow$ 27K	
$[5,6] \rightarrow$ Ashley	$[3,4) \rightarrow$ Mktg	$[2,4) \rightarrow$ 30K	
	$[5,6] \rightarrow$ Engrng	$[5,6] \rightarrow$ 35K	$\{1,2,3,5,6\}$

DEPARTMENT		
DEPT	*MGR*	*lifespan*
$[0,6] \rightarrow$ Acctng	$[0,2) \rightarrow$ Paul	
	$[2,6] \rightarrow$ Juni	$\{0,1,2,3,4,5,6\}$
$[0,6] \rightarrow$ Engrng	$[0,5) \rightarrow$ Wanda	
	$[5,6] \rightarrow$ Ashley	$\{0,1,2,3,4,5,6\}$
$[0,6] \rightarrow$ Mktg	$[0,5) \rightarrow$ Tom	
	$[5,6] \rightarrow$ Thomas	$\{0,1,2,3,4,5,6\}$
$[0,6] \rightarrow$ Sales	$[0,6] \rightarrow$ Sue	$\{0,1,2,3,4,5,6\}$

Figure 8: The TG Historical Relations **EMPLOYEE** and **DEPARTMENT**

notation; we will refer to it in the remainder of the paper. Note that the **EMPLOYEE** relation records historical information on three employees in three historical tuples, and the **DEPARTMENT** relation represents the history of four departments in four historical tuples.

3 Historical Relational Completeness for Ungrouped Languages

In this section, we define the concept of **ungrouped temporal relational completeness.** It will be based on two temporal calculi and a temporal algebra. We will define all three formalisms in this section and show their equivalence. However, to make the paper self-contained, we provide a brief overview of temporal logic in the next subsection.

3.1 Overview of Temporal Logic

In this section, we review the basics of temporal logic. Both Kroger [Kro87] and Rescher and Urquhart [RU71] provide a good introduction to the subject.

Since temporal logic deals with time, we have to specify the model of time that we will be working with. The most general model represents time as an arbitrary set with a partial order imposed on it. By specifying additional axioms, we can introduce other models of time, e.g. time can be discrete or dense, bounded or unbounded, linear or branching [vB83]. Although the temporal calculus can be defined for an arbitrary model of time (since it is based on the predicate temporal logic), we consider the *discrete linear* temporal domain in this paper because the algebra *TA* defined in Section 3.3 is based on that domain. We note that this is the model of time generally considered by historical and temporal data models ([MS90]).

The syntax of a predicate temporal logic is obtained from the first-order logic by adding various temporal operators to it. In this paper, we consider the *US logic*, i.e., the temporal logic with **until** and **since** temporal operators, because it is shown in [Kam68] and also in [Gab89] that this logic is equivalent to the first-order temporal logic with explicit references to time[5]. There are several different definitions of **until** and **since** operators proposed in the literature. We will use the definition of these operators from [Kro87] shown in Figure 9.

[5]Kamp [Kam68] used the term *complete* to describe this property. However, we will use this term in a broader sense and abstain from introducing any additional terminology.

A **until** B: is true now if B will be true at some *future* time t and A will be true for *all* the moments of time in the time interval $(now,\ t)$

A **since** B: is true now if B was true at some *past* time t and A was true for *all* the moments of time in the time interval $(t,\ now)$

Figure 9: Temporal Operators **until** and **since**

The semantics of a temporal logic formula is defined with a **temporal structure** [Kro87], which comprises the values of all its predicates at *all* the time instances. Formally, let $P_1, \ldots, P_k$ be a finite set of predicates considered in the predicate temporal language[6]. Then, a **temporal structure** is a mapping $K : T \rightarrow \mathcal{P}_1 \times \ldots \times \mathcal{P}_k$, where T is a set of time instances, and $\mathcal{P}_i$ is the set of all the possible interpretations of predicate P_i. The mapping K assigns to each time instance an interpretation of each of the predicates $P_1, \ldots, P_k$ at that time. We will use K_t instead of $K(t)$ to denote the value of temporal structure K at time t. We make an assumption, natural in the database context, that the domains of predicates *do not change* over time.

From a database perspective, a temporal structure K is most naturally looked at as mapping of each moment of time t into a state of the database, i.e. into instances of each of the database relations at time t. Therefore, each predicate in a temporal structure determines a **historical relation**, i.e. a relation that changes over time.

A historical database represented in a certain historical data model, such as the (historical component of) TQuel data model [Sno87], HRDM [CC87], or the homogeneous data model [Gad88], defines a temporal structure, i.e. the mapping K, although often implicitly. Therefore, a temporal structure represents a common base of comparison for different historical data models. For instance, the temporal structure of the N1NF historical relation **EMPLOYEE** presented in Figure 2 is shown in Figure 10.

Given a temporal structure for temporal logic predicates, we can interpret arbitrary temporal logic formulas in the standard inductive way [Kro87]. For example, the meaning of

[6]Since we are interested in database applications, we consider only a finite number of predicates corresponding to the set of relations in a database.

EMPLOYEE	
time	K_i*(EMPLOYEE)*
$i = 0$	EMPL(Tom, Sales, 20K)
$i = 1$	EMPL(Tom, Finance, 20K) EMPL(Jim, Finance, 20K) EMPL(Scott, Finance, 20K)
$i = 2$	EMPL(Tom, Finance, 20K) EMPL(Jim, MIS, 30K) EMPL(Scott, Sales, 20K)
$i = 3$	EMPL(Thomas, MIS, 27K) EMPL(Jim, MIS, 40K)

Figure 10: Temporal Structure for Relation **EMPLOYEE**.

until and **since** operators in Figure 9 can be defined inductively in terms of the temporal structures for A and B.

Alternatively, assertions about temporal structures can be expressed in a two-sorted first-order logic, where one of the sorts is time. In this logic, arbitrary quantifications are allowed over both temporal and non-temporal variables.

It is clear that **until** and **since** operators can be expressed in this first-order logic. In fact, Figure 9 shows how to do that. Furthermore, Kamp [Kam68] and subsequently Gabbay [Gab89] have shown that the two logics are equivalent when time is modeled either by the real numbers or the integers.

3.2 Temporal Calculi *TL* and *TC*

In Section 3.1 we described the temporal logic *US* and also considered a two-sorted first-order logic with explicit references to time. These two logics give rise to two temporal calculi *TL* and *TC*. In order to define them precisely, we first introduce the concept of **temporal safety** for the two languages.

Intuitively, a temporal formula (both a temporal logic and a first-order formula with explicit

references to time) is safe if it can produce *only* bounded relations at all time instances[7] and if these relations contain *only* data from the database. We basically define the safety of temporal logic formulas as it is done for the snapshot relational case [Ull88], except that, in addition, we assume that the temporal operators **until** and **since** produce safe formulas if operands of these operators constitute safe formulas. It is easy to see that, indeed, these temporal operators cannot produce infinite historical relations if they operate on finite relations. For the first-order logic with explicit references to time, safety is defined exactly as in [Ull88].

With this definition of safety for the two types of logic, we are ready to define the two calculi *TL* and *TC*.

Definition. A temporal calculus query is an expression of the form

$$\{x_1, x_2, \ldots, x_n \mid \phi\}$$

where ϕ is a *safe* temporal logic formula and $x_1, x_2, \ldots, x_n$ are the free variables in ϕ[8]. We denote the temporal calculus based on these queries as *TL*.

Let T be a temporal domain for the predicates in ϕ. The *answer* to this query is a historical relation defined on T, such that for any t in T, its instance is

$$\{x_1, x_2, \ldots, x_n \mid K_t(\phi(x_1, x_2, \ldots, x_n))\}$$

We also define a temporal query expressed in the first-order logic with explicit references to time in a similar way as

$$\{x_1, x_2, \ldots, x_n,\ t \mid \phi\}$$

where ϕ is a *safe* formula from the first-order logic with explicit references to time, x_1, x_2, ..., x_n are the free variables in ϕ, and t is a temporal variable. The answer to this query is defined exactly as in the standard relational case. We denote the temporal calculus based on these queries as *TC*.

[7] "Boundedness" refers to the structural and not to the temporal domain because we have already assumed that the temporal domain is bounded.

[8] As in the standard relational case, we assume that other free variables in ϕ not appearing among the output variables $(x_1, x_2, \ldots, x_n)$ are implicitly existentially quantified.

UNEMPL	
NAME	*YEAR*
Tom Jim	1986
Tom Scott	1987
Scott	1988
$\emptyset$	1989 - 1990

Figure 11: Historical Relation **UNEMPL**

Note that in both calculi a query operates on historical relations and returns a historical relation, i.e. it returns the same type of object as the type of its operands.

Example 1 Assume that time is measured in years. Consider historical relation **UNEMPL**(NAME) specifying that a person is unemployed for most of the year. Figure 11 gives an example of such a relation. Historical relation **TAXES**(NAME, TAX) specifies taxes a person paid in a certain year. Figure 12 gives an example of such a relation. We say that a person is a "good citizen" if he or she always paid taxes during the period of his or her last employment, i.e. since the last time the person was unemployed. The relation **GOOD_CITIZEN**(NAME) can be computed with the following *TL* query:

$$\textbf{GOOD\_CITIZEN} = \{NAME \mid \textbf{TAXES}(NAME, TAX) \textbf{ since } \textbf{UNEMPL}(NAME)\}$$

The same relation can also be computed with the following *TC* query:

$$\begin{aligned}\textbf{GOOD\_CITIZEN} = \{NAME, t \mid (\exists t')(&\textbf{UNEMPL}(NAME, t') \text{ and} \\ &(\forall t'')(t' < t'' < t \Rightarrow \textbf{TAXES}(NAME, TAX, t''))\}\end{aligned}$$

□

The proposal to use *TL* as a query language for historical databases was made in [Tuz89] and in [TC90]. The proposal to use *TC* as a query language for historical databases was made in [KSW90]. Since the *US* temporal logic is equivalent to the first-order logic with

TAXES		
NAME	*TAX*	*YEAR*
Scott	8400	1986
Jim	10400	1987
Jim	10800	1988
Tom	12000	
Jim	11500	1989
Tom	13200	
Jim	12800	1990
Tom	13600	
Scott	9200	

Figure 12: Historical Relation **TAXES**

explicit references to time for the discrete linear model of time considered in the paper [Kam68], it follows that the two calculi *TL* and *TC* are also equivalent.

3.3 Temporal Algebra *TA* and Its Equivalence to Calculi *TL* and *TC*

In this section, we present a temporal algebra *TA* that is equivalent to the two calculi defined in the previous section. This algebra was first introduced in [TC90].

Let $\mathbf{R} = \{R_t\}_{t \in T}$, $\mathbf{S} = \{S_t\}_{t \in T}$ and $\mathbf{Q} = \{Q_t\}_{t \in T}$ be historical relations defined over a temporal domain (lifespan) $T = [t_1, t_n]$. Using the standard relational algebra terminology, we say that two historical relations are **union compatible** if their schemas have the same sets of attributes. Then we consider the following temporal algebra operators:

O1: *Select*: $\mathbf{S} = \sigma_F(\mathbf{R})$ iff $S_t = \sigma_F(R_t)$ for all t in T, where F is the first-order (non-temporal) formula defined as for the standard relational case [Ull88].

O2: *Project*: $\mathbf{S} = \pi_{A_1,\ldots,A_k}(\mathbf{R})$ iff $S_t = \pi_{A_1,\ldots,A_k}(R_t)$ for all t in T, where $A_1, \ldots, A_k$ are some attributes in R.

O3: *Cartesian product*: $\mathbf{S} = \mathbf{R} \times \mathbf{Q}$ iff $S_t = R_t \times Q_t$ for all t in T.

O4: *Set difference*: $\mathbf{S} = \mathbf{R} - \mathbf{Q}$ iff $\mathbf{S}$ and $\mathbf{R}$ are union compatible and $S_t = R_t - Q_t$ for all t in T.

O5: *Union*: $\mathbf{S} = \mathbf{R} \cup \mathbf{Q}$ iff $\mathbf{S}$ and $\mathbf{R}$ are union compatible and $S_t = R_t \cup Q_t$ for all t in T.

O6a: *Future linear recursive operator*: $\mathbf{S} = \mathbf{L_F}(\mathbf{R}, \mathbf{Q})$ iff $S_{t+1} = (R_t \cap S_t) \cup Q_t$, $S_{t_1} = \emptyset$.

O6b: *Past linear recursive operator*: $\mathbf{S} = \mathbf{L_P}(\mathbf{R}, \mathbf{Q})$ iff $S_{t-1} = (R_t \cap S_t) \cup Q_t$, $S_{t_n} = \emptyset$.

A temporal join operation $\bowtie$ can be defined exactly as for the snapshot case in terms of the Cartesian product, select and project operators.

Denote the temporal algebra defined by operators **O1** - **O6** as *TA*. Note that the operators **O1** - **O5** correspond to the standard relational operators of the snapshot relational algebra and are reduced to these operators in the degenerate case when $t_1 = t_n$.

Example 2 The relation **GOOD_CITIZEN**(NAME), defined in Example 1, can be computed in *TA* as follows. Set **TAXES1** = π_{NAME} (**TAXES**). Then

GOOD_CITIZEN = L_F(**TAXES1**,**UNEMPL**)

i.e.,

$\text{GOOD\_CITIZEN}_{k+1} = (\text{GOOD\_CITIZEN}_k \cap \text{TAXES1}_k) \cup \text{UNEMPL}_k$

The resulting relation **GOOD_CITIZEN** is shown in Figure 13. The last row constitutes a prediction of who will be a good citizen in 1991.

□

It is shown in [TC90] that the algebra *TA* is equivalent to the calculus *TL* and, therefore, to *TC* for the discrete linear model of time. This means that the three languages, i.e. *TL*, *TC*, and *TA* are equivalent in terms of the expressive power.

GOOD_CITIZEN	
NAME	*YEAR*
$\emptyset$	1986
Tom Jim	1987
Jim Tom Scott	1988
Jim Tom Scott	1989
Jim Tom	1990
Jim Tom	1991

Figure 13: Historical Relation **GOOD_CITIZEN**

3.4 Ungrouped Historical Relational Completeness

We propose to use the three languages considered in this section, the historical relational algebra (*TA*) and the two temporal calculi (*TC* and *TL*), as a basis for ungrouped historical relational completeness because of the following reasons. First, the temporal calculi have a sound and well-studied theoretical basis since they are based on first-order logic and on temporal logic. Second, both the calculi and the algebra are very simple. Essentially, one temporal calculus is based on the first-order logic and another one is obtained from the first-order logic by adding to it the temporal operators **until** and **since**. The temporal algebra is obtained from the relational algebra by a straightforward extension to its five basic operators and by the addition of a single additional temporal operator. Third, the two calculi and the algebra are equivalent for certain models of time, i.e. besides being simple and "natural," the two approaches have the same expressive power. This suggests that they capture an important class of temporal queries. Fourth, the temporal algebra and the two calculi are reduced to the relational algebra and calculus in the degenerate case when the time set consists of only one instance. Therefore, the notion of historical ungrouped completeness is compatible with standard relational completeness when the

temporal dimension is so reduced. Fifth, the temporal calculi are *independent* of a specific historical relational data model, and the temporal algebra is independent of any data model based on the discrete bounded model of time. Some of the query languages and algebras proposed in the literature are *tailored* to a specific historical data model. That is, operators of these languages take into account specific constructs of the underlying historical data model. For example, the constructs **overlap**, **begin of** and **end of** of TQuel [Sno87] assume that the temporal data are grouped into *intervals*. There are no model-specific operators in the temporal calculus and in the algebra considered in this paper. This means that the temporal calculus can be applied to *any* historical relational data model and the temporal algebra to any historical relational data model supporting discrete bounded time.

For all these reasons, we propose to use the two calculi and the algebra presented in this section as a basis of ungrouped historical completeness. We note that our notion of ungrouped historical completeness subsumes the notion proposed in [Sno87, p.287], "the temporal query language, when applied to a snapshot of the database, is at least as powerful as ... Codd's definition." Our notion meets this criterion, but also allows queries (e.g., comparing values across different times) that cannot be reduced to operations on a snapshot of the database. We will return to this issue in Section 6 when we examine the completeness of a number of models proposed in the literature.

4 Historical Relational Completeness for Grouped Languages

In this section we define a *concept* of **grouped historical relational completeness.** The basis for this concept of completeness is a (grouped) tuple-based historical relational calculus, L_h.

4.1 A Grouped Historical Calculus

To begin our development of grouped historical relational completeness we define a tuple-based historical relational calculus, the language L_h. This language is intended for grouped

historical relations that conform to the canonical grouped relations defined in Section 2.2. It is a many-sorted logic with variables over ordinary values, historical values, and times admitting quantification over all three sorts of variables. This distinguishes L_h from, among other languages, Gadia's calculus [Gad88], which does not have temporal variables or quantification over them. To simplify the discussion we will assume that we are defining this calculus relative to a particular relational database $d_{TG} = \{r_1, r_2, \ldots, r_n\}$, with universe of values $\mathbf{D}$, universe of times $\mathbf{T}$ and universe of attributes U_A.

The Syntax of L_h

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

   (a) *Constant Symbols*

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

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

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

   (b) *Variable Symbols*

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

      ii. $V_D = \{x_1, x_2, \ldots\}$ is a denumerably infinite set of domain variables

      iii. $V_{TV} = \{e_1, e_2, \ldots\}$ is a denumerably infinite set of tuple variables

   (c) *Predicate Symbols*

      i. $\theta = \{\theta_1, \theta_2, \ldots, \theta_{n_\theta}\}$ 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). The predicate symbol $<$ must be included for the domain T.

      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 as follows:

   (a) Every individual constant δ is a **value term**

   (b) Every domain variable x is a **value term**

   (c) If e is a tuple variable, A is an attribute name constant, and t is a temporal variable, then $e.A(t)$ is a **value term**

   (d) Every temporal constant τ and temporal variable t is a **temporal term**

   (e) If e is a tuple variable, then $e.l$ is a **lifespan term**, where l is a distinguished symbol of L_h

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

   (a) If α and β are both value terms, and θ is a dyadic predicate, then $\alpha\theta\beta$ is a formula.

   (b) If α is a lifespan term and t is a temporal variable, then $t \in \alpha$ is a formula.

   (c) 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.

   (d) If r is a relation predicate and e is a tuple variable, then r(e) is a formula.

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

   (f) If ϕ is a formula and u is a tuple, temporal, or domain variable, then $\forall u(\phi)$ and $\exists u(\phi)$ are both formulae.

4. The *Query Expressions* of L_h are all expressions of the form:
   $[e_1.A_1, \ldots, e_n.A_n : t]\phi$ where:

   (a) $[e_1.A_1, \ldots, e_n.A_n : t]$ is called a target list, and consists of

      i. A list of terms $e_i.A_i$ where each e_i is a free tuple variable, and

      ii. The free temporal variable t

(b) ϕ is a formula.

As a convenience, for a set of attributes $A = \{A_1, A_2, \ldots, A_n\}$ we use the notation $e.A$ to denote the list $e.A_1, e.A_2, \ldots, e.A_n$ in a target list. Similarly, given a tuple variable e 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 $e.*$ to denote $e.A$.

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 δ denotes an object in some value domain D_i

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

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

   (b) *Variable Symbols*

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

      ii. A domain variable x denotes a value in the universe of domain values **D**

      iii. A tuple variable e denotes a tuple in some grouped historical relation r in the database d

   (c) *Predicate Symbols*

      i. A binary predicate symbol θ denotes some value comparator (e.g., $=, \neq, \leq$) over objects in some 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 individual constant δ denotes an object in some value domain D_i

(b) A domain variable x denotes an object in some value domain D_i

(c) A value term $e.A(t)$ denotes an object in some value domain. In particular, it denotes the value in the tuple denoted by e of the attribute denoted by A at the time denoted by t.

(d) A temporal constant τ denotes a time in the universe of times $\mathbf{T}$

(e) A lifespan term $e.l$ denotes a set of times, in particular, the set of times which is the lifespan of the tuple denoted by e.

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

(a) $\alpha\theta\beta$ is true just in case the denotation of α stands in the relation θ with the denotation of β, and false otherwise

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

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

(d) r(e) is true just in case the tuple denoted by e is in the grouped historical relation denoted by r, and false otherwise.

(e) (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.

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

4. A *Query Expression* $[e_1.A_1, \ldots, e_n.A_n : t]\phi$ 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 value terms $e_i.A_i$. The lifespan of each tuple in the result is the set of values of the temporal variable t, for which all of the $e_i.A_i(t)$ values satisfy the formula ϕ.

4.2 Safety

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. Without these restrictions it is possible to specify formulae that define historical relations that contain an infinite number of tuples, e.g., $[e.* : t]\neg r(e) \wedge t = 12$. It is also possible to specify relations, some of whose tuples have unbounded lifespans or undefined values for certain times within their lifespans, e.g., $[e.* : t]r(e) \wedge \neg(t \in e.l)$.

To avoid such situations we restrict the allowable formulae of L_h to a subset of *safe* formulae. Our the definition of safety derives from [Ull88], and is extended to the temporal domain. For a formula ϕ of L_h to be safe it must satisfy the following conditions:

1. It does not contain any use of the universal quantifier ($\forall$).
2. It contains exactly one free temporal variable, no free domain variables, and for each free tuple variable e, ϕ is of the form $F_e \wedge t \in e.l$ where t is the free temporal variable.
3. For each disjunction $F_1 \vee F_2$ in ϕ, F_1 and F_2 must include the same set of atoms $t_i \in e_j$.
4. In each maximal conjunct $F_1 \wedge \ldots \wedge F_n$ of ϕ the following conditions hold:
   (a) for each F_i of the form $e.A(t) = \alpha$ or $\alpha = e.A(t)$, there is an F_j of the form $t \in e.l$, where α is a term of category value.
   (b) for each F_i of the form $t \in e.l$, there is an F_j of the form $R(e)$;
   (c) for each F_i of the form $\neg F_i'$ the following condition must hold: for all the free temporal variables t in F_i' there is an F_j of the form $t \in e.l$ and for all the free historical variables e in F_i' there is an F_j of the form $R(e)$.
5. The application of the not operator $\neg$ is limited to those terms F_i defined in the rule above for maximal conjuncts

An L_h query

$$[e_1.A_1, \ldots, e_n.A_n : t]\phi$$

is *safe* if the corresponding L_h formula ϕ is safe. We restrict our attention to safe L_h queries in the sequel.

4.3 Examples of L_h Queries

In the following we give several examples of queries expressed in the language L_h for the database consisting of the **EMPLOYEE** and **DEPARTMENT** relations shown in Figure 8.

Example 3 This first query performs a *selection* of historical tuples from **EMPLOYEE**, and projects the results onto the attributes *NAME* and *SALARY*.

What are the names and salaries of those employees in the marketing department at time 6?

$$[e.NAME, e.SALARY : t]\mathbf{EMPLOYEE}(e) \wedge t \in e.l \wedge$$
$$\exists t_1(e.DEPT(t_1) = Mktg \wedge \mathbf{EMPLOYEE}(E) \wedge t_1 \in e.l \wedge t_1 = 6)$$

□

Example 4 This second query returns a set of historical tuples that are derived from the *joining* of two historical relations.

Who are the managers for whom Tom has worked?

$$[e_1.* : t]\mathbf{EMPLOYEE}(e_1) \wedge t \in e_1.l \wedge$$
$$\exists e_2 \exists t_2 \exists d(\mathbf{EMPLOYEE}(e_2) \wedge t_2 \in e_2.l \wedge$$
$$\mathbf{DEPARTMENT}(d) \wedge t_2 \in d.l \wedge$$
$$e_2.NAME(t_2) = Tom \wedge e_2.DEPT(t_2) = d.DEPT(t_2) \wedge$$
$$d.MGR(t_2) = e_1.NAME(t_2))$$

□

Example 5 Finally, we give an example of a query that, although *semantically* safe in that it returns a historical relation having a finite number tuples whose values are extracted from the base relations, is *syntactically* unsafe.

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

$$[e.* : t]\mathbf{EMPLOYEE}(e) \wedge t \in e.l \wedge$$
$$\neg(\exists t_1(t_1 \in e.l \wedge \neg(e.DEPT(t_1) = Acctng)))$$

The query is not safe because the quantified subformula, which is a maximal conjunct, does not include the conjunct $\mathbf{EMPLOYEE}(e)$. If this conjunct were added into the subformula the entire formula would be safe.

□

4.4 Grouped Historical Relational Completeness

We propose to use the language considered in this section, the many-sorted calculus L_h, as the basis for grouped historical relational completeness. The reasons that support the choice of this language as an appropriate metric for completeness are essentially similar to those that motivated out choice of the metric(s) for ungrouped historical relational completeness (Section 3.4). L_h has a sound and well-studied theoretical basis since it is based on a many-sorted first-order logic. The sorts that it uses are the "natural" ones for the task at hand: ordinary values, temporal values, and historical or time-series values. The need for historical values has already been motivated: they provide the linguistic mechanism for direct reference to temporally changing values, and provide for the grouping of these values with the object that they describe. As with our metric for historical ungrouped completeness, L_h reduces to the relational calculus in the degenerate case when the time

set consists of only one instance. It is therefore compatible with standard relational completeness when the temporal dimension is so reduced. Furthermore, in Section 5 we shall see that L_h differs from TC and the other ungrouped languages *only* in this respect, so that it is an extension of the concept of ungrouped historical completeness that is *minimal*: it adds only what is necessary for providing temporal value integrity.

5 Relationship Between Historically Grouped and Historically Ungrouped Completeness

We defined grouped historical completeness based on the calculus L_h in Section 4 and ungrouped historical completeness based on calculi TC, TC, and on algebra TA in Section 3. In this section, we study the relationship between these two concepts.

Since the query languages and algebras for the two types of completeness are based on different data models, they are unrelated to each other. This means that the data model for one of the types of completeness must be adjusted to make a comparison possible. In this section, we adjust the language TC so that it can also support grouping. Then we show that the adjusted language, TC_g, is "equivalent" in some sense to the grouped language L_h.

5.1 TC_g: Extending TC to Support Grouping

To support grouping in the temporal calculus TC, we introduce an additional **group identifier** attribute for each relation in TC_g. For example, a TC relation $R(A, B, T)$ is extended with an additional attribute O and becomes $R(O, A, B, T)$, where O is a group identifier attribute. The grouping attribute serves a role similar to that of object identifiers in object-oriented databases.

To define the grouping process, we introduce a temporal logic with *grouping*, TC_g, as a 3-sorted first-order logic, where the first sort is the **domain** sort, the second sort is the **temporal** sort, and the third sort is the **grouping** sort. The domain and the temporal

EMPLOYEE				
gid	*NAME*	*DEPT*	*SALARY*	*time*
100	Tom	Sales	20K	0
100	Tom	Sales	20K	1
100	Tom	Sales	20K	2
100	Tom	Sales	30K	3
100	Tom	Mktg	30K	4
100	Thomas	Mktg	27K	5
100	Thomas	Mktg	27K	6
101	Juni	Acctng	28K	2
101	Juni	Acctng	28K	3
101	Juni	Acctng	28K	4
101	Juni	Acctng	28K	5
101	Juni	Acctng	28K	6
102	Ashley	Engrng	27K	1
102	Ashley	Engrng	30K	2
102	Ashley	Mktg	30K	3
102	Ashley	Engrng	35K	5
102	Ashley	Engrng	35K	6

Figure 14: Relation **EMPLOYEE** in the Grouped TC_g Model

sorts are defined exactly as for TC in Section 3. Intuitively, the grouping sort divides a TC_g relation into groups, each group having the same *group identifier*. Furthermore, tuples are parameterized by time within each group, i.e. the combination of the group-id and time uniquely determines the tuple. Figure 14 shows the **EMPLOYEE** relation of Figure 8 as it would be represented in the TC_g model.

Formally, the grouping sort **O** has countably many constants and variables, and a set of function symbols new_k for $k = 1, 2, \ldots$ that will be defined later. Relational predicates have one and only one attribute with the grouping and temporal sorts, and relational operators ($=$, $>$, $\geq$) are not defined for the grouping sort. Finally, the grouping sort admits quantifiers.

The semantics of grouping is captured with the following **grouping axioms** that specify how TC_g tuples are grouped into "temporal objects."

1. A group-id and time uniquely determine the rest of the tuple *no matter* which relation it belongs to, i.e. if $R(o, x_1, \ldots, x_n, t)$ and $Q(o, y_1, \ldots, y_m, t)$ are true then $m = n$ (i.e., the relations must be union-compatible) and $x_i = y_i$ for $i = 1, \ldots, n$. In other words, OT functionally determines all the attributes in all the relations in which O and T appear.

2. A group-id uniquely determines the group of tuples *independently* of which relation they belong to, i.e., if o appears in relations R and Q, meaning that if both $(\exists u_1) \ldots (\exists u_n)(\exists t')R(o, u_1, \ldots, u_n, t')$ and $(\exists y_1) \ldots (\exists y_n)(\exists t'')Q(o, y_1, \ldots, y_n, t'')$ hold, then, for all $x_1, \ldots, x_n$, t, if $R(o, x_1, \ldots, x_n, t)$ is true then $Q(o, x_1, \ldots, x_n, t)$ is also true, and vice versa.

3. A group of tuples uniquely determines the group-id, i.e. there cannot be two identical groups of tuples with different group-id's. Formally, if there are R, Q, o, and o' such that for all $x_1, \ldots, x_n$, t, if $R(o, x_1, \ldots, x_n, t)$ implies $Q(o', x_1, \ldots, x_n, t)$ and $Q(o', x_1, \ldots, x_n, t)$ implies $R(o, x_1, \ldots, x_n, t)$, then $o = o'$.

The first axiom ensures that a group-id always refers to the groups of tuples of the same arity, and that elements in the same group, defined by a group-id, are parameterized by time. The second and third axioms ensure that a group-id uniquely defines a group of tuples and that a group of tuples is assigned a unique group-id. These axioms ensure that group-ids uniquely identify a group of tuples and vice versa, so that the notion of a group of tuples in the ungrouped model can be made (below) consistent with the notion of a single tuple in the grouped model.

A TC_g **query** is defined as

$$Q(\phi) \equiv \{<< o_1, x_1 >, \ldots, < o_n, x_n >, t > \mid \phi\}$$

where ϕ is a TC_g formula and o_i, x_i and t, for $i = 1, \ldots, n$, are the only free group, domain and temporal variables, respectively, appearing in it.

Example 6 Consider the query of Example 3 in Section 4

What are the names and salaries of those employees in the marketing department at time 6?

It can be expressed in TC_g as

$$\{< o, x >, < o, z >, t \mid \mathbf{EMPLOYEE}(o, x, y, z, t) \wedge y = Mktng \wedge t = 6\}$$

□

However, the definition of a TC_g query, as defined above, has one important drawback. A query does not return an object of the same type as the objects it operates on, i.e. it does not return historically grouped relations. To fix this problem, we slightly change the definition of a TC_g query by "encoding" the tuple of pairs $< o_1, x_1 >, \ldots, < o_n, x_n >$ with a new group-id.

To do this, we divide the set of tuples $S = \{< o_1, x_1 >, \ldots, < o_n, x_n >, t\}$ into groups of tuples

$$G(o_1, \ldots, o_n, S) = \{x_1, \ldots x_n,\ t \mid (< o_1, x_1 >, \ldots, < o_n, x_n >, t) \in S\}$$

i.e. put attributes of tuples with the same group-id's into the same group. Then we encode the group of tuples $G(o_1, \ldots, o_n)$ with an **encoding function**

$$new_k : 2^{\mathbf{D} \times T} \rightarrow O$$

where $\mathbf{D}$ is the universe of all possible values (Section 2.1), O is a set of group-id's, and T is the set of times. An encoding function is a bijection between sets $2^{\mathbf{D} \times T}$ and O. It is well-known that such encoding functions are definable ([End72]).

Then the definition of a TC_g query is changed to

$$\{< new_n(G(o_1, \ldots, o_n, Q(\phi))), x_1, \ldots, x_n, t > \mid \phi\}$$

This definition says that, first, the query is computed according to the previous definition, then tuples in the answer are grouped into sets $G(o_1, \ldots, o_n, Q(\phi))$ and, finally, each set is given a unique group-id.

Although this definition of a TC_g query is technically better than the first one, because it evaluates to objects of the same type, the first definition is easier to use. Therefore, we will often use the first definition of a query in the paper, because it could always be modified to the second form.

Semantics of TC_g Queries. Since TC_g is a 3-sorted first-order logic, the semantics of its formulae is defined as in the standard case of many-sorted logics [End72]. Based on this semantics, a TC_g query

$$\{<< o_1, x_1 >, \ldots, < o_n, x_n >, t > \mid \phi\}$$

returns the set of tuples $<< o_1, x_1 >, \ldots, < o_n, x_n >, t >$ for which the formula ϕ is true.

Safety for TC_g Queries. As is the case with L_h formulae and the standard first-order relational calculus, we have to define *safe* TC_g formulae that return finite answers over a finite time horizon.

A TC_g formula ϕ is **safe** if it satisfies the following conditions.

1. It does not contain any universal quantifiers ($\forall$).

2. There is exactly one free temporal variable t, and for every free group-id variable o_i, $i = 1, \ldots, n$, there is a **range expression** $\psi_i = (\exists x_{ij_1}) \ldots (\exists x_{ij_i}) R_i(o_i, x_{i1}, \ldots, x_{in_i}, t)$ such that $\phi = \psi_1 \wedge \ldots \wedge \psi_n \wedge \phi'$ for some TC_g formula ϕ', and such that all the free domain variables of ϕ and only they appear among the free variables of range expressions ψ_i.

3. If a group-id variable o and a temporal variable t in a TC_g formula ϕ appear in the same predicate $R(o, \ldots, t)$, then we say that there is a **pair** $< o, t >$ of variables in ϕ.

   Then the two disjuncts F_1 and F_2 of each disjunction operator in ϕ, $F_1 \vee F_2$, must have the *same* set of pairs $< o_i, t_j >$.

4. Every maximal conjunct $F_1 \wedge \ldots \wedge F_n$ in ϕ must satisfy the following conditions:

   (a) If some F_i has the form $x = \alpha$ or $\alpha = x$, where α is either a constant or a variable, then there is a conjunct F_j of the form $R(o, \ldots, x, \ldots, t)$ for some variables o and t.

   (b) If some F_i has the form $\neg F_i'$ then for each free temporal variable t in F_i' there must be a conjunct F_j either of the form $Q(o', x_1'', \ldots, x_n'', t)$ or of the form $t = c$, and for each free group-id variable o in F_i' there is a conjunct F_j of the form $Q(o, x_1', \ldots, x_n', t')$.

5. The application of the not operator $\neg$ is limited to those terms F_i defined in the rule above for maximal conjuncts.

This definition of safety mirrors the definition of safety of L_h formulae as defined in Section 4. In particular, Condition 2 ensures that only data from the database can appear in the answer to a TC_g query. This definition is also an extension of the definition of safety from [Ull88] to the temporal domain.

A TC_g query

$$\{<< o_1, x_1 >, \ldots, < o_n, x_n >, t > \mid \phi\}$$

is *safe* if the corresponding TC_g formula ϕ is safe. We restrict our attention to safe TC_g queries in the sequel.

5.2 Equivalence of Languages L_h and TC_g

In this section, we show that the two languages L_h and TC_g are "equivalent" in a sense to be defined precisely below. Since TC_g differs from TC only by supporting the grouping attribute in its relations, we in fact show that the languages L_h and TC are "close enough" to each other and that the concepts of grouped and ungrouped historical completeness are essentially the same "modulo grouping."

The first step in defining and proving equivalences of L_h and TC_g should be establishing the relationship between data models for these two languages. TC_g is based on the relational

data model with group identifiers and L_h on the HRDM data model. In the next section, we explain how one data model can be mapped into another and vice versa.

5.2.1 Relationship Between L_h and TC_g Data Models

In this section, we define mappings between the structures of the ungrouped and grouped historical models. Ω_{UG} maps TC_g relations into L_h relations; intuitively, it groups TC_g tuples with the same group-id into a single group that becomes a historical tuple. Ω_{GU} maps L_h relations to TC_g relations; intuitively, it ungroups a historical tuple into a set of tuples with the same group-id.

Formally, the mapping Ω_{UG} from TC_g to L_h relations is defined as follows. Let R and R' be TC_g and L_h relations, respectively, with the same number of domain attributes $A_1, \ldots, A_k$. Then $\Omega_{UG}(R) = R'$ if and only if the following conditions hold:

1. Each tuple in R appears in some historical tuple in R', i.e. for all the tuples $< o, a_1, \ldots, a_k, t >$ belonging to relation R there is a historical tuple e such that $R'(e)$ is true, $t \in e.l$, and $e.A_1(t) = a_1, \ \ldots, \ e.A_k(t) = a_k$.

2. Each historical tuple e in R' contains all ungrouped tuples from R with the same group-id. Formally, if $R'(e)$ and $R(o, a_1, \ldots, a_k, t)$ are true for some historical tuple e, group-id o, domain values $a_1, \ldots, a_k$ and time t, and if $t \in e.l$ and $e.A_1(t) = a_1, \ \ldots, \ e.A_k(t) = a_k$, then for all $a'_1, \ldots, a'_k, \ t'$, if $R(o, a'_1, \ldots, a'_k, t')$ is true then $t' \in e.l$ and $e.A_1(t') = a'_1, \ \ldots, \ e.A_k(t') = a'_k$.

The mapping Ω_{GU} is defined similarly. It ungroups all the historical tuples into relational tuples with the same group-id. We omit the formal definition of Ω_{GU} because it is very close to the definition of Ω_{UG}.

Clearly, the two mappings Ω_{GU} and Ω_{UG} are inverses of each other, i.e. $\Omega_{GU} \circ \Omega_{UG} = I$ and $\Omega_{UG} \circ \Omega_{GU} = I$ because grouping followed by ungrouping and ungrouping followed by grouping always produce the same relation. This property holds because we introduced

group-id's. Without group-id's, we cannot reconstruct a relation if we first group and then ungroup it and vice versa. (The same problem occurs in all N1NF models ([FVG85, RKS88]. [RKS88, p.409] points out that "in order to avoid problems where [grouping an ungrouped relation is impossible] we assume each database relation, ...their nested relations, and relations created by collecting constants into a limited domain, have an *implicit* [italics ours] keying attribute (or set of attributes) whose value uniquely determines the values of all the other attributes." Our *group-ids* make explicit the need for such a "keying attribute".)

5.2.2 Mapping TC_g Formulae to L_h

In this section, we define the mapping Γ_{UG} that maps safe TC_g formulae into equivalent safe L_h formulae.

To define equivalence, let ϕ be a TC_g formula and ϕ' be an L_h formula, with a set of "similar" relational predicates. That is, there is a bijection between predicates in ϕ and ϕ' such that the corresponding predicates R in ϕ and S in ϕ' have schemas $R(O, A_1, \ldots, A_n, T)$ and $S(A_1, \ldots, A_n)$ respectively. We say that such formulae ϕ and ϕ' are **equivalent** if and only if for any instances $R_1, \ldots, R_n$ of TC_g predicates appearing in ϕ

$$\phi'(\Omega_{UG}(R_1), \ldots, \Omega_{UG}(R_n)) = \Omega_{UG}(\phi(R_1, \ldots, R_n))$$

and for any instances of L_h predicates $S_1, \ldots, S_n$ appearing in ϕ'

$$\phi(\Omega_{GU}(S_1), \ldots, \Omega_{GU}(S_n)) = \Omega_{GU}(\phi'(S_1, \ldots, S_n))$$

It follows from this definition that two formulae ϕ and ϕ' are equivalent if both diagrams in Figure 15 are commutative, i.e. it does not matter if Ω_{UG} is applied first and then ϕ', or ϕ and then Ω_{UG}. Also, it does not matter if Ω_{GU} is applied first and then ϕ, or ϕ' and then Ω_{UG}.

It remains for us to define the mapping Γ_{UG} from TC_g to L_h formulae. Let ϕ be a safe TC_g formula. The formula $\Gamma_{UG}(\phi)$ is obtained from ϕ by replacing all the atomic formulae in

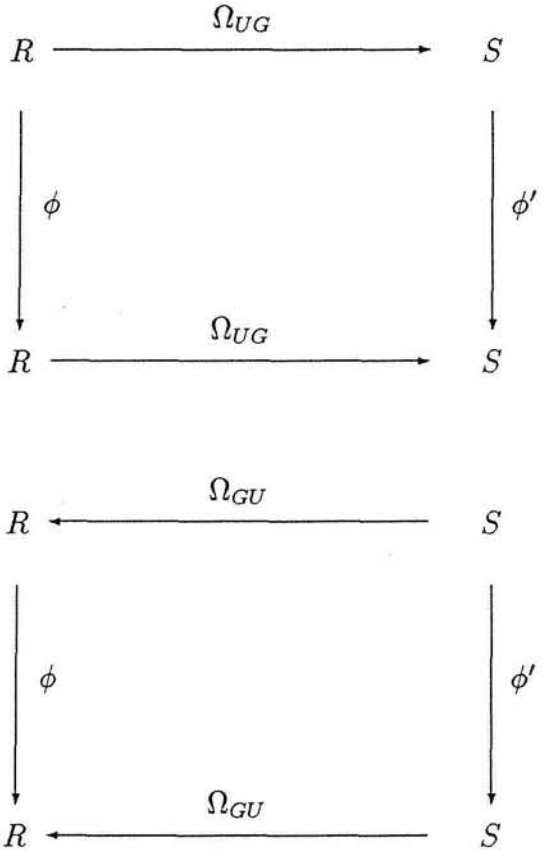

Figure 15: Definition of equivalence of TC_g formula ϕ and L_h formula ϕ'.

ϕ together with quantified variables in the manner described below, and leaving all other connectives (e.g. $\wedge$, $\vee$, $\neg$) intact. The replacement of atomic formulae and quantified variables in ϕ is done as follows:

1. Replace TC_g predicate $R(o, x_1, \ldots, x_n, t)$ with

$$R(e) \wedge t \in e.l \wedge e.A_1(t) = x_1 \wedge \ldots \wedge e.A_n(t) = x_n$$

   where A_i is the attribute in R corresponding to variable x_i[9]. A group-id variable o defines a unique historic variable *across* different relations, i.e. if several predicates in ϕ have the same group-id variable o then this variable o is replaced with the same historic variable e.

2. Replace each quantifier $(\exists o)$ in ϕ with the quantifier $(\exists e)$ in $\Gamma_{UG}(\phi)$, where o is a group-id variable appearing in some TC_g relation $R_i(o, x_1, \ldots, x_n, t)$, and e is the corresponding historic variable defined in Step 1 that replaces o.

3. If a domain variable x is bound in ϕ then do not change its quantifier $(\exists x)$ in $\Gamma_{UG}(\phi)$. If x is unbound in ϕ then $\Gamma_{UG}(\phi)$ is of the form $(\exists x)\psi$, where $\psi \in L_h$.

4. Each range expression $\psi_i = (\exists x_{ij_1}) \ldots (\exists x_{ij_i}) R_i(o_i, x_{i1}, \ldots, x_{in_i}, t)$[10] is replaced with the expression $R_i(e_i) \wedge t \in e_i.l$.

We defined the mapping Γ_{UG} on the set of safe TC_g *formulae*. This mapping can be extended to TC_g *queries* as follows. If $\mathbf{Q}$ is a TC_g query

$$\{< o_1, x_1 >, ..., < o_n, x_n >, t \mid \phi\}$$

where ϕ is a safe TC_g formula of the form:

$$\phi = \phi' \wedge \bigwedge_{i=1}^{n} (\exists x_{ij_1}) \ldots (\exists x_{ij_i}) R_i(o_i, x_{i1}, \ldots, x_{in_i}, t)$$

[9] Actually, there is no need to add expressions $e.A_i(t) = x_i$ for all $i = 1, \ldots, n$ as some examples below will show, but only for those x_i's that appear in other expressions. However, it is acceptable to do it for all terms, i.e. it simplifies the presentation, and the transformation is still correct.

[10] Range expressions were introduced in Section 5.1 when safe TC_g queries were introduced.

then $\Gamma_{UG}(\mathbf{Q})$ is

$$[e_1.A_1, \ldots, e_n.A_n : t]\ \Gamma_{UG}(\phi') \wedge \bigwedge_{i=1}^{n}(R_i(e_i) \wedge t \in e_i.l)$$

where historic variables e_i correspond to the group-id variables o_i appearing in predicates R_i in ϕ, and attributes A_j correspond to variables x_j in these predicates.

Examples illustrating the mapping Γ_{UG} follow. In these examples we assume that the schemas of TC_g relations R and Q are $R(O, A, T)$ and $Q(O, A, T)$ respectively, where O is a group-id, A is an attribute, and T is a temporal attribute.

Example 7 Consider the TC_g query $\mathbf{Q}$

$$\{<< o, x >, < o', x' >, t > \mid R(o, x, t) \wedge Q(o', x', t)\}$$

The first step of the conversion algorithm replaces $R(o, x, t)$ with $R(e) \wedge t \in e.l \wedge e.A(t) = x$, and $Q(o', x', t)$ with $Q(e') \wedge t \in e'.l \wedge e'.A(t) = x'$. The second step is not applicable in this case. The third step results in

$$(\exists x)(\exists x')((R(e) \wedge t \in e.l \wedge e.A(t) = x) \wedge (Q(e') \wedge t \in e'.l \wedge e'.A(t) = x'))$$

Finally, the query $\Gamma_{UG}(\mathbf{Q})$ is

$$[e.A, e'.A' : t]\ (\exists x)(\exists x')((R(e) \wedge t \in e.l \wedge e.A(t) = x) \wedge (Q(e') \wedge t \in e'.l \wedge e'.A(t) = x'))$$
$$\wedge R(e) \wedge t \in e.l \wedge Q(e') \wedge t' \in e'.l$$

This expression for $\Gamma_{UG}(\mathbf{Q})$ could be simplified (using standard techniques of logical transformation) to

$$[e.A, e'.A' : t]\ R(e) \wedge t \in e.l \wedge Q(e') \wedge t \in e'.l$$

However, this simplification is not always possible as the following example shows.

□

Example 8 The TC_g query

$$\{< o, x >, t \mid R(o, x, t) \wedge (\exists o')Q(o', x, t)\}$$

is mapped into

$$[e.A : t]\ R(e) \wedge t \in e.l \wedge e.A(t) = x \wedge (\exists x)(\exists e')(Q(e') \wedge t \in e'.l \wedge e'.A(t) = x)$$

Note that in this case the variable x serves to equate the terms $e.A(t)$ and $e'.A(t)$, via transitivity. In certain cases, the two values $e.A(t)$ and $e'.A(t)$ cannot be equated directly because one of the terms $e.A(t)$ or $e'.A(t)$ is ill-defined. In this case, this use of an intermediate, such as x above, solves the problem. Also note that the quantified group-id variable $(\exists o')$ was replaced with the historic variable $(\exists e')$ in the L_h formula.

□

Example 9 The TC_g query

$$\{< o, x >, t \mid R(o, x, t) \wedge (\exists x')(\exists t')Q(o, x', t')\}$$

is mapped with some additional simplifications into

$$[e.A : t]\ (\exists x')(\exists t')(Q(e) \wedge t' \in e.l \wedge e.A(t') = x') \wedge R(e) \wedge t \in e.l$$

Note that there is the same historic variable in both $R(e)$ and $Q(e)$ because there is the same group-id variable o in the corresponding TC_g formula. Also note that the domain variable x' in the TC_g formula remained unchanged in the L_h formula.

□

Example 10 The TC_g query

$$\{< o, x >, t \mid R(o, x, t) \wedge (\exists x')(\exists t')(R(o, x, t) \wedge Q(o, x', t') \wedge x = x')\}$$

is replaced with

$$[e.A : t]\ (\exists x)((R(e) \wedge t \in e.l \wedge e.A(t) = x) \wedge (\exists x')(\exists t')((R(e) \wedge t \in e.l \wedge e.A(t) = x) \wedge (Q(e) \wedge t' \in e.l \wedge e.A(t') = x') \wedge x = x')) \wedge R(e) \wedge t \in e.l$$

This expression can be simplified to

$$[e.A : t]\ (\exists x)(\exists x')(\exists t')((R(e) \wedge t \in e.l \wedge e.A(t) = x) \wedge (Q(e) \wedge t' \in e.l \wedge e.A(t') = x') \wedge x = x') \wedge R(e) \wedge t \in e.l$$

Note that the equality $x = x'$ did not change in the conversion process. However, it follows from the facts that $e.A(t) = x$, $e.A(t') = x'$ and $x = x'$ that the terms $e.A(t)$ and $e.A(t')$ are equal.

□

Example 11 The TC_g formula

$$\{< o, x >, t \mid R(o, x, t) \wedge \neg Q(o, x, t)\}$$

is converted to

$$[e.A : t]\ (\exists x)(R(e) \wedge t \in e.l \wedge e.A(t) = x \wedge \neg(Q(e) \wedge t \in e.l \wedge e.A(t) = x)) \wedge R(e) \wedge t \in e.l$$

□

Note that in the previous examples, Γ_{UG} maps safe TC_g formulae into safe L_h formulae. We generalize these observations in the following proposition.

Proposition 1 *Γ_{UG} maps safe TC_g formulae into safe L_h formulae.*

Sketch of Proof: Let ϕ be a safe TC_g formula. We will prove that $\Gamma_{UG}(\phi)$ is safe by verifying all the conditions in the definition of safety for L_h formulae. First, $\Gamma_{UG}(\phi)$ does not have universal quantifiers since ϕ does not have them.

Second, the range expression $\psi_i = (\exists x_{ij_1}) \ldots (\exists x_{ij_i}) R_i(o_i, x_{i1}, \ldots, x_{in_i}, t)$ is mapped into the expression $(\exists x_{ij_1}) \ldots (\exists x_{ij_i})(R_i(e_i) \wedge t \in e_i.l \wedge e_i.A_{ij_1}(t) = x_{i1} \wedge \ldots \wedge e_i.A_{ij_i}(t) = x_{ij_i})$ and also the expression $R_i(e_i) \wedge t \in e_i.l$ is added at the "outermost" level of $\Gamma_{UG}(\phi)$ because of condition 4 in the definition of the mapping Γ_{UG}. Clearly, the two expressions are semantically equivalent. But the second condition was added to make $\Gamma_{UG}(\phi)$ syntactically safe. Since $\Gamma_{UG}(\phi)$ has the formula $R_i(e_i) \wedge t \in e_i.l$ for each range expression at the outermost level, the second condition of safety for L_h formulae is satisfied.

Third, subformula $F_1 \vee F_2$ in ϕ is mapped into $\Gamma_{UG}(F_1) \vee \Gamma_{UG}(F_2)$ so that $\Gamma_{UG}(F_1)$ and $\Gamma_{UG}(F_2)$ have the same set of atoms $t_i \in e_j$ because the formulae F_1 and F_2 have the same set of pairs $< o_j, t_i >$ and because the mapping Γ_{UG} translates them into expressions $t_i \in e_j$.

Finally, the mapping Γ_{UG} is defined so that all the three items in the definition of safety related to maximal conjuncts are satisfied. □

Theorem 2 *For any safe TC_g formula ϕ, ϕ and $\Gamma_{UG}(\phi)$ are equivalent.*

Sketch of Proof: Intuitively, the two formulae are equivalent because the predicate $R(o, x_1, \ldots, x_n, t)$ is mapped into the expression $R(e) \wedge t \in e.l$, so that the historical variable e corresponds to the group-id o and t is in the lifespan of e. Furthermore, group-id's are defined so that the variables $x_1, \ldots, x_n$ are uniquely determined by values of o and t and are irrelevant in the translation process. Also, the expressions $R(o, x_1, \ldots, x_n, t)$ and $R(e) \wedge t \in e.l$ are equivalent. In addition, the mapping Γ_{UG} preserves the structure of the formula ϕ, i.e. it leaves conjunctions, disjunctions and negations of ϕ in their places in $\Gamma_{UG}(\phi)$. □

5.2.3 Mapping L_h Formulae to TC_g

In this section, we define the mapping Γ_{GU} that maps safe L_h formulae into equivalent safe TC_g formulae. Let ϕ be a safe L_h formula. As for the Γ_{UG} mapping, the formula $\Gamma_{GU}(\phi)$ is obtained from ϕ by replacing all the atomic formulae in ϕ together with quantified variables and leaving the structure of ϕ intact (operators $\wedge$, $\vee$, $\neg$ remain unchanged). The replacement of atomic formulae and quantified variables is done in the following manner:

1. Replace quantified variables in L_h as follows.

   (a) Do not change any quantified domain and temporal variables, i.e. $(\exists x)$ and $(\exists t)$ in L_h will remain in $\Gamma_{GU}(\phi)$.

(b) Replace quantified historic variables $(\exists e_i)$ with $(\exists o_i)$, where o_i is a *unique* group-id variable.

(c) Consider all pairs of historic and temporal variables e and t such that ϕ contains an expression $t \in e.l$. Depending on the relationship between the scopes of these variables, we add the expression $(\exists x_1) \ldots (\exists x_n)$ to $\Gamma_{GU}(\phi)$, where x_i is a domain variable associated with historic variable e of arity n, as follows.

i. if t is a free and e is a bound variable, then place the expression $(\exists x_1) \ldots (\exists x_n)$ before the expression $(\exists o)$ obtained in Step 1b;

ii. if t and e are bound variables, and the scope of e is contained within the scope of t then also place $(\exists x_1) \ldots (\exists x_n)$ before $(\exists o)$;

iii. if t and e are bound and the scope of t is contained within the scope of e then place $(\exists x_1) \ldots (\exists x_n)$ before $(\exists t)$;

iv. in all other cases, do not add anything to the formula.

2. Replace each occurrence of L_h expression $R(e)$ with $(\exists x_1) \ldots (\exists x_n)(\exists t)R(o, x_1, \ldots, x_n, t)$. If e is a bound variable in ϕ, then the group-id variable o is the same as the one that replaced e in the expression $(\exists e)$ in Step 1b. If e is free, then all the occurrences of e are replaced with the same group-id variable o.

3. Replace each occurrence of expression $t \in e.l$ with $R(o, x_1, \ldots, x_n, t)$, where predicate R is *one of* the predicates occurring positively in the maximal conjunct containing $t \in e.l$[11]. If e is a bound variable in ϕ, then the group-id variable o is the same as the one that replaced e in the expression $(\exists e)$ in Step 1, and the domain variables $x_1, \ldots, x_n$ are the same as the quantified variables introduced in Step 1 for the combination of $(\exists e)$ and $(\exists t)$ expressions. If e is a free variable in ϕ, then the group-id variable o and the domain variables $x_1, \ldots, x_n$ are free and are different from all other variables in $\Gamma_{GU}(\phi)$.

4. Replace each term $e.A_i(t)$ in ϕ with x_i, where x_i is defined as follows. Since ϕ is safe, the maximal conjunct containing $e.A_i(t)$ must also contain expressions $t \in e.l$

[11] It follows from the grouping axioms in Section 5.1 that it does not matter which positively occurring predicate R is selected. Any selected predicate will produce the same results. In fact, all the qualifying predicates can be selected as well, for a longer but logically equivalent formula.

and $R(e)$ (for some R). In Step 3, $t \in e.l$ is replaced with $R(o, x_1, \ldots, x_n, t)$. Then x_i corresponds to the variable in this expression that corresponds to attribute A_i in R.[12]

Examples illustrating the mapping Γ_{GU} follow. In these examples we assume that the schemas of relations R and Q from L_h are $R(A, B)$ and $Q(A)$ respectively.

Example 12 The L_h query

$$[e.* : t]\ R(e) \wedge t \in e.l \wedge e.B(t) = 5$$

is mapped into the TC_g query as follows. $R(e)$ is replaced with $(\exists x')(\exists y')(\exists t')R(o, x', y', t')$, $t \in e.l$ with $R(o, x, y, t)$, and $e.B(t) = 5$ with $y = 5$.

Putting the pieces together, we get the answer:

$$\{< o, x >, < o, y >, t \mid (\exists x')(\exists y')(\exists t')R(o, x', y', t') \wedge R(o, x, y, t) \wedge y = 5\}$$

Since $(\exists x')(\exists y')(\exists t')R(o, x', y', t') \wedge R(o, x, y, t)$ is equivalent to $R(o, x, y, t)$ we can rewrite the previous query as

$$\{< o, x >, < o, y >, t \mid R(o, x, y, t) \wedge y = 5\}$$

□

Example 13 The L_h query

$$[e.* : t]\ R(e) \wedge t \in e.l \wedge (\exists t')(R(e) \wedge t' \in e.l \wedge (\exists e')(Q(e') \wedge t' \in e'.l \wedge R(e) \wedge t \in e.l \wedge e.B(t) = e'.A(t')))$$

is mapped into the TC_g query

$$\{< o, x >, < o, y >, t \mid R(o, x, y, t) \wedge (\exists x'')(\exists y'')(\exists t')(R(o, x'', y'', t') \wedge (\exists o')(\exists x')(Q(o', x', t') \wedge R(o, x, y, t) \wedge y = x'))\}$$

[12] Remark in the footnote 11 is also applicable here.

□

Note that the domain variable x' in the previous example is quantified in the *same* part of the $\Gamma_{GU}(\phi)$ formula as the group-id variable o'. Also note that the variables x'', y'' are quantified together with temporal variable t'. In general, the domain variables appearing in the same predicate as group-id variable o and temporal variable t are quantified together with the *innermost* scope of variables o and t. The following example illustrates this point further.

Example 14 The L_h query

$$[e.B : t]\ R(e) \wedge t \in e.l \wedge (\exists e')(R(e) \wedge t \in e.l \wedge e.A(t) = 5 \wedge (\exists t')(Q(e') \wedge t' \in e'.l \wedge R(e) \wedge t \in e.l \wedge e.B(t) = e'.A(t')))$$

is mapped into the TC_g query

$$\{< o, y >, t \mid R(o,x,y,t) \wedge (\exists o')(R(o,x,y,t) \wedge x = 5 \wedge (\exists t')(\exists x')(Q(o',x',t') \wedge R(o,x,y,t) \wedge y = x'))\}$$

Note that the variable x' is quantified together with t' and *not* with o', as was done in Example 13.

□

The next two examples show how Γ_{GU} handles negations.

Example 15 The L_h query

$$[e.* : t]\ R(e) \wedge \neg S(e) \wedge t \in e.l$$

is mapped into the TC_g query

$$\{< o, x >, < o, y >, t \mid R(o,x,y,t) \wedge \neg(\exists x')(\exists y')(\exists t')R(o,x',y',t')\}$$

□

Example 16 The L_h query

$$[e'.* : t]\ (\exists e)(Q(e) \wedge \neg(t \in e.l \wedge Q(e)) \wedge R(e') \wedge t \in e'.l) \wedge R(e') \wedge t \in e'.l$$

is converted to

$$\{< o', x' >, < o', y' >, t \mid (\exists o)(\exists x)((\exists x'')(\exists t'')Q(o, x'', t'') \wedge \neg Q(o, x, t) \wedge R(o', x', y', t)) \wedge R(o', x', y', t)\}$$

□

The next example shows that Γ_{GU} does not affect domain variables in ϕ.

Example 17 The L_h query

$$[e.* : t]\ R(e) \wedge t \in e.l \wedge (\exists z)(R(e) \wedge t \in e.l \wedge e.A(t) = z)$$

is translated into

$$\{< o, x >, < o, y >, t \mid R(o, x, y, t) \wedge (\exists z)(R(o, x, y, t) \wedge x = z)\}$$

□

Proposition 3 *Γ_{GU} maps safe L_h formulae into safe TC_g formulae.*

Sketch of Proof: The proof proceeds along the lines of the proof of Proposition 1. □

Theorem 4 *For any safe L_h formula ϕ, ϕ and $\Gamma_{GU}(\phi)$ are equivalent.*

Sketch of Proof: The proof is done by induction on maximal conjuncts in ϕ. At any inductive step the L_h formula $\phi(e_1, \ldots, e_n, x_1, \ldots, x_m, t_1, \ldots, t_k)$ is mapped into the TC_g formula $\Gamma_{GU}(\phi)(o_1, \ldots, o_n, x_1, \ldots, x_m, y_1, \ldots, y_l, t_1, \ldots, t_k)$, where $y_1, \ldots, y_l$ are extra variables

introduced in the translation process (i.e. when $R(e) \wedge t \in e.l$ becomes $R(o, y_1, \ldots, y_s, t)$). Notice that variables $y_1, \ldots, y_l$ are uniquely determined (i.e. functionally depend) by values of variables $o_1, \ldots, o_n, x_1, \ldots, x_m, t_1, \ldots, t_k$. Therefore, these variables are "superfluous" and do not affect the translation process. With this observation in mind, the proof proceeds along the lines of Theorem 2. □

5.2.4 Conclusion

Theorems 2 and 4 show that the languages L_h and TC_g are equivalent. Since TC_g differs from TC only by supporting the grouping attribute in its relations, it shows that the languages L_h and TC are "close" to each other. In fact, it is precisely and only the *inherent* grouping mechanism of temporal values in L_h that makes it more powerful than TC.

6 Historical Models and Completeness

All of the historical relational data models and languages 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 four of these models and discuss the completeness of their languages. Two of the data models we discuss are ungrouped, one with an algebra ([Lor87]) and the other with a calculus ([Sno87]); the other two data models discussed are grouped, one with an algebra ([CC87]), the other with both an algebra and a calculus ([Gad88]).

For each of the data models discussed in the following, we are interested in two aspects of its query language: its **expressiveness**, that is, its ability to express every relation that can be denoted by expressions of the languages L_h or TC defined in the earlier sections, and its **boundedness**, its ability to express only those relations that can be expressed by these languages. It is well known that the standard relational calculus is as expressive as, bounded by, and hence equivalent to the standard relational algebra ([Cod72]).

The completeness of a language can be viewed solely in terms of its relative expressiveness. We have earlier motivated our choice of L_h and TC as appropriate metrics for our notions of completeness. Therefore, in this section a language will be said to be **complete** with respect to either L_h or TC if it is as expressive as that language. We also consider the boundedness of each of the four query languages discussed in this section. If our metric of completeness is reasonable, then it must be the case that each language considered here is bounded by either L_h or TC. 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 one of the previously defined languages, and then its expressiveness. We shall see that all of the languages we consider are bounded by either L_h or TC, but not all are as expressive.

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 is derived directly from the structure of the historical relations in **HRDM**, and because the set of operators specified by this model were intended initially to provide all the functionality thought useful and desirable.

6.1 HRDM

The historical relational data model **HRDM** presented in [CC87] is a temporally grouped model with an algebraic query language which is presented as an extension to the standard relational algebra.

We can categorize the operators of **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 ($\times$). Because these operators do not exploit the *historical* aspects of **HRDM** relations, the standard mappings from these operators in relational algebra to

their counterpart in relational calculus also applies to these operators here. For example,

$$\begin{aligned} r \cup s &= \{x | x \in r \vee x \in s\} \\ &\equiv [e. * | t] r(e) \wedge t \in e.l \vee s(e) \wedge t \in e.l \end{aligned}$$

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.

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

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.

$$\sigma - \mathrm{IF}_{(A\theta a, Q, L)}(r) = \{x \in r | Q(t \in (L \cap x.l))[x.A(t) \theta a]\}$$

$$
\begin{aligned}
(\text{if } Q \text{ is } \exists) \;\equiv\; & [e.* : t]r(e) \wedge t \in e.l \wedge \\
& \exists t_1(t_1 \in L \wedge t_1 \in e.l \wedge e.A(t_1)\theta a) \\
(\text{if } Q \text{ is } \forall) \;\equiv\; & [e.* : t]r(e) \wedge t \in e.l \wedge \\
& \neg\exists t_1(t_1 \in L \wedge t_1 \in e.l \wedge \neg(t_1 \wedge e.A(t)\theta a))
\end{aligned}
$$

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.

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

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.

$$
\begin{aligned}
r_1[A\theta B]r_2 \;&=\; \{e | \exists e_{r_1} \in r_1, \exists e_{r_2} \in r_2 e.l = \{t | e_{r_1}(A)(t)\theta e_{r_2}(B)(t)\} \wedge \\
&\quad e.v(R1) = e_{r_1}.v(R1)|_{e.l} \wedge \\
&\quad e.v(R2) = e_{r_2}.v(R2)|_{e.l}]\} \\
&\equiv\; [e_1.*, e_2.* : t]r_1(e_1) \wedge r_2(e_2) \\
&\quad \wedge t \in e_1.l \wedge t \in e_2.l \wedge e_1.A(t)\theta e_2.B(t)
\end{aligned}
$$

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

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

Other Operators In addition to the above categories of operators, the **HRDM** algebra includes several *grouping* operators that are used to restructure a relation without changing the information content of that relation. These operators, **union-merge** ($\cup_o$), **intersection-merge** ($\cap_o$), and **difference-merge** ($-_o$), first computes the set-theoretic union, intersection, and difference, respectively, and then regroups the tuples in the resulting relation.

The **HRDM** algebra also includes the operators **WHEN** and **Dynamic Time-Slice.** We categorize the **WHEN** 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. The dynamic time-slice is only applicable to relations that include in their scheme an attribute A whose domain consists of partial functions from the set $TIMES$ into itself. We do not treat such attributes in this paper since most of the models considered distinguish between ordinary values and the times at which they hold, and do not allow comparisons between them. Therefore it would be unfair to include such an operator in our comparison. We omit the other operators from our discussion of completeness of **HRDM** and the remaining languages that we will examine. The grouping operators are not treated because they are not intended for querying, and the aggregate operators, because they are outside of the scope of standard relational-based notions of completeness.

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 8 for the name and department of each employee that has at some time received a change in salary, expressible in L_h as

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

The lack of an equivalent algebraic expression is due to the specification of those operators in **HRDM** 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.

6.2 The Historical Homogeneous Model of Gadia

The next historical model that we discuss is one that was proposed by Gadia [Gad88]; it is a model that includes a query language and an algebra. This data model, which we shall label **TDMG**, is the same as that of **HRDM**, and thus of the canonical historical relation defined in Section 2.

In **TDMG** the value of a tuple attribute is a function from a set of times to the value domain of the attribute, and the lifespan is the same for all the attributes (Gadia's *homogeneity assumption*). Therefore the **TDMG** model is *temporally grouped.*

In addition to the data model, Gadia defines a historical algebra and calculus. Although his data model is temporally grouped, the semantics of the algebra is defined in terms of the ungrouped model obtained by ungrouping temporal relations. Gadia calls this a *snapshot interpretation* semantics. The semantics of the historical algebra is defined by ungrouping temporal relations because Gadia considers grouped and ungrouped models "weakly equal" and does not distinguish between them when he proves equivalence of the algebra and the calculus.

Gadia's algebra is defined as follows. He starts with the five standard relational operators of selection, projection, difference, Cartesian product, and union as *TL* does. He also defines derived temporal operators such as join, intersection, negation, and renaming. In addition, he defines temporal expressions for the temporal domain. Finally, he combines relational and temporal expressions by considering relational expressions of the form $e(v)$ where e and v are relational and temporal expressions respectively.

Gadia's algebra is bounded by the temporal calculus *TC* defined in Section 3 for the following reasons. The five standard temporal operators are defined as for *TL* and, therefore, can be expressed in *TC*. Temporal expressions are defined as a closure of a time intervals over the operations of union, intersection, difference and negation. Each of these operators can be expressed in the first-order logic with explicit references to time. For example, the expression $tdom(r(A,B)) \vee tdom(s(A,B))$ can be defined in *TC* as $\{t \mid (\exists x)(\exists y)r(x,y,t) \vee s(x,y,t)\}$. This means that **TDMG** is bounded by *TC*.

Gadia also defines a historical calculus and shows its equivalence to the algebra (modulo temporal grouping). This calculus is expressible in L_h for the same reasons that the ungrouped algebra is expressible in *TC*. A lifespan of a temporal tuple x in **TDMG** can be captured with expression $t \in x.l$ in L_h. Also, the operators of union, intersection, difference and negation for temporal expressions can be expressed in L_h with the same methods that are used to express algebraic expressions in *TC* since L_h explicitly supports time.

However, both Gadia's algebra and calculus are *not* complete for the same reason that the **HRDM** algebra is not complete: it is not possible to compare the value of one attribute at time t_1 with the value of another or the same attribute at some other time t_2. For example, the query that finds the name and department of each employee that has at some time received a cut in salary, i.e.

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

cannot be expressed in **TDMG**.

To summarize, the temporally grouped language L_h has strictly more expressive power than Gadia's calculus, i.e. this calculus is bounded by L_h but not complete. Also, the temporally ungrouped language *TC* is strictly more powerful than Gadia's algebra , i.e. the algebra is bounded by *TC* but not complete.

6.3 TQuel

TQuel is the query language component of a historical relational data model proposed by Snodgrass [Sno87]. We shall call this model **TRDM**.

TRDM provides for two types of historical relations. One, called an **interval** relation, 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. (An example of such a relation has already been given in Figure 3). As before, we will ignore the two *TRANS-TIME* temporal attributes since we are only considering *historical* data models. The values of the non-temporal 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.

The query language TQuel is an extended relational calculus derived from and defined as a superset of Quel, the query language of the Ingres relational database management system [SWKH76]. TQuel extends Quel by adding temporal-based clauses that accommodate 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 additional 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 among 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.[13] Since standard tuple calculus is bounded by TC, this implies that TQuel is bounded by L_h.

TQuel is bounded by the language TC 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 TC. In particular, Snodgrass shows how 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 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. Therefore (as with Quel) a sequence of TQuel statements, can express any algebraic expression, perhaps by creating temporary relations, and using operators such as *APPEND* and *DELETE*,

Although interval relations and event relations are distinguished by TQuel, they are standard first normal 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 – with the addition of such operators as *APPEND* – it is complete in the sense defined by Codd. By extension, as a result of the use of the temporal attributes, it has *temporally ungrouped completeness.* Therefore we conclude that **TRDM** is complete in the *temporally ungrouped* sense, but does not exhibit *temporal value integrity.*

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

6.4 The Temporal Relational Algebra of Lorentzos

The final historical data model that we discuss is one that was proposed by Lorentzos in [Lor87]. The data model in [Lor87], which is called **TRA**, is essentially the same as that in [Sno87], except that as an *historical* model it is restricted to only one temporal dimension. One of the stated goals of **TRA** is that "no new elementary relational algebra operations are introduced and first normal form is maintained" [Lor87, p. 99]. Typical relations in this model appear basically as in Figure 3 (with the columns *valid-from* and *valid-to* called *Sfrom* and *Sto*, respectively). Although the structures of relations in this model are essentially the same as in the historical version of **TRDM**, we discuss this model here because, unlike [Sno87], the language it proposes is an algebra rather than a calculus.

It is difficult to discuss formally the algebra of **TRA** because it is not specified formally. Rather, it is presented via a series of example queries and discussion. Nevertheless, enough of a picture of the algebra emerges clearly through these examples to make a discussion possible.

Two new operators, *FOLD* and *UNFOLD* are defined. These operators essentially convert between the time interval representation (as in Figure 3) and a time point representation (as in Figure 1). The *FOLD* and *UNFOLD* are clearly expressible in terms of operators in the standard relational algebra, as [Lor87] points out.

The previous sections demonstrated that two other algebras, that of **HRDM** and that of **TDMG** were incomplete because they were not able 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. In **TRA** such comparisons *are* possible. Consider again the query that finds the name and department of each employee that has at some time received a cut in salary:

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

This query can be expressed in **TRA** as follows. First UNFOLD the interval relation

EMPLOYEE into all of its time points:

$$EMPLOYEE_{U_1} = UNFOLD[Time, Start, Stop](TIME, EMPL)$$

Then, Θ-Join this relation with itself, joining tuples with the same name and with a pay cut, and then Select just the names of the employees from the result (here NAME1 and NAME2, etc., refer to the NAME attributes in the first and second operands to the Join):

$$TEMP1 = EMPLOYEE_{U_1} \left[\begin{array}{lcl} NAME1 & = & NAME2, \\ TIME1 & < & TIME2, \\ SAL1 & > & SAL2 \end{array} \right] EMPLOYEE_{U_2}$$

$$TEMP2 = \sigma_{NAME1}(TEMP1)$$

Finally, Join the result with the original relation and Project onto the desired fields:

$$\pi_{\{NAME,DEPT,Sfro,Sto\}}\ (TEMP2 \bowtie EMPLOYEE)$$

Because **TRA** is equivalent to standard relational algebra, the question of its completeness, as in the case of **TRDM**, is reduced to the question of the completeness of relational algebra. Therefore we conclude that **TRA** is complete in the *temporally ungrouped* sense but, like all ungrouped languages, it does not exhibit *temporal value integrity.*

The results of our explorations into the completeness of these five languages is summarized in the table in Figure 16.

7 Summary and Conclusions

In this paper we have explored the question of completeness of languages for historical database models. In this exploration we were led to characterize such models as being of

Language	Reference	Type	Completeness
L_h	Section 4	grouped	Proposed Basis
TL	Section 3	ungrouped	Proposed Basis
TRA algebra	[Lor87]	ungrouped	complete
TRDM calculus	[Sno87]	ungrouped	complete
HRDM algebra	[CC87]	grouped	incomplete
TDMG calculus	[Gad88]	grouped	incomplete
TDMG algebra	[Gad88]	ungrouped	incomplete

Figure 16: Summary of Completeness Results

one of two different types, either **temporally grouped** or **temporally ungrouped**. We first discussed these notions informally by means of example databases and queries, and showed that the two models were not equivalent. The difference between the two models is that in temporally grouped models, historical values are treated as first class objects which can be referred to directly in the query language. In the temporally ungrouped models, no such direct reference is permitted. We characterized this property of the grouped models as **temporal value integrity**.

We then proceeded to propose a notion of **historical relational completeness**, analogous to Codd's notion of relational completeness, for both types of models. We showed that the temporally ungrouped languages are less powerful than the grouped models, because they do not allow for direct reference to temporal objects like salary histories. However, we also demonstrated a technique for extending the ungrouped models, by incorporating a *grouping mechanism*, to capture the additional semantic power of temporal grouping.

Specifically, for the ungrouped models we defined three different languages, TL, TC, and TA: a temporal logic, a logic with explicit reference to time, and a temporal algebra, and showed that under certain assumptions about the model of time employed all three are equivalent in power. For the grouped models we defined the calculus L_h, a many-sorted logic with variables over ordinary values, historical values, and times. We demonstrated a technique for extending the ungrouped model with a grouping mechanism, a *group identifier*. With this mechanism we showed how the ungrouped language TC could be extended to TC_g in

such a way as to make it equivalent to L_h. In this way we demonstrated that the languages are *nearly* equivalent; that is, it is precisely the *grouping* capability which distinguishes them.

Finally, we examined several historical relational proposals in light of these metrics: L_h as the standard for **grouped historical relational completeness** and TC as the standard for **ungrouped historical relational completeness**. We looked at four historical models, two grouped and two ungrouped, offering five different languages. In the ungrouped models we found both a complete algebra (**TRA**) and calculus (TQuel from **TRDM**), while in the grouped models we found (in addition to our metric, the complete calculus L_h) an incomplete algebra (**HRDM**) and an incomplete calculus (**TDMG**) as well as an incomplete ungrouped algebra (**TDMG**). We believe that this classification scheme, and our examination of the completeness of several historical models, should help to explicate the differences and the commonalities between the various models proposed in the literature.

One point bears emphasizing. It has on occasion been said that the issue of adding time to relational databases is an uninteresting one, since the user can always just add whatever extra attributes are desired (e.g., **Start-Time** and **End-Time**) and then use standard SQL (or relational algebra) as the query language. In our discussion of the completeness of the ungrouped temporal languages we, to some extent, relied on the underlying point of this argument. For example, this point underlay our argument that **TRA** (which is equivalent to standard relational algebra) is complete in the ungrouped sense. Two points need to be made in reply to this comment. First, there is a difference between the formal notion of completeness and the informal, but no less important, notion of ease of use. Even though the programming language *C* is formally equivalent to a Turing Machine, it is a lot more convenient to use *C* if you are writing an operating system because of its *built-in* high level features. The *built-in* temporal features of the historical and temporal data models make them easier to use for managing temporal data; without these features a greater burden is placed upon the user. Secondly, this paper has shown that the grouped models and languages are more expressive than the ungrouped systems, unless these models add a surrogate grouping mechanism. This grouping mechanism, itself, is a higher-level construct

that is *implicit* in the grouped systems (and this, we argue, makes them more convenient), but needs to be made *explicit* in the ungrouped systems for them to be equivalent in expressive power.

There are a few interesting areas for future research that this work has clarified. First of all, it is interesting to note that we did not find here, nor are we aware of, *any* complete algebra for grouped historical data models. This is clearly an interesting open question. Another area in which there continues to be interest is in the support of evolving schemas. Our decision not to treat this interesting area here was based largely on the fact that hardly any of the models except [CC87] incorporate this feature, and we wanted to choose the common denominator of all the models in order to make our comparisons fairly. The model in [CC87] addressed this issue, and other work (e.g. [BKKK87, MS90]) continues to be done in this area.

Finally, we would like to address the question of completeness for *temporal* as opposed to *historical* relational models (in the terminology of [SA85]). We believe that our results on **grouped** and **ungrouped historical relational completeness** can be extended in a straightforward way to temporal data models and languages. The extension would involve the addition of another sort (for transaction times). In ungrouped temporal models, relations would be extended with an additional column to stamp every tuple with its transaction time, and the language would have constants, as well as variables, and quantification for this sort. In grouped temporal models, values would be extended to be doubly indexed; they would most likely be better modeled as functions from a transaction time into functions from a data time to a scalar value, but the order of the two temporal indices could be reversed. Preliminary work that we have done on Indexical Databases holds promise for a unified treatment, not only of these two temporal dimensions, but of spatial, or other, dimensions as well.

References

[AC86] G. Ariav and J. Clifford. Temporal data management: Models and systems. In

G. Ariav and J. Clifford, editors, *New Directions for Database Systems*, pages 168–185. Ablex Publishing Corporation, 1986.

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

[AU79] A.V. Aho and J.D. Ullman. Universality of data retrieval languages. In *ACM Symposium on Principles of Programming Languages*, pages 110–120, New York, 1979. ACM.

[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.

[BKKK87] J. Banerjee, W. Kim, H.-J. Kim, and H.F. Korth. Semantics and implementation of schema evolution in object-oriented databases. In *SIGMOD*, pages 311–322, San Francisco, CA, 1987. ACM.

[BZ82] 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, Los Angeles, February 1987. IEEE.

[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.

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

[CW83] J. Clifford and D. S. Warren. Formal semantics for time in databases. *ACM Transactions on Database Systems*, 6(2):214–254, June 1983.

[End72] H. B. Enderton. *A Mathematical Introduction to Logic.* Academic Press, 1972. New York.

[FVG85] P.C. Fischer and D. Van Gucht. Determining when a structure is a nested relation. In *International Conference on Very Large Databases*, pages 171–180, 1985.

[Gab89] D. Gabbay. The declarative past and imperative future: Executable temporal logic for interactive systems. In B. Banieqbal, H. Barringer, and A. Pnueli, editors, *Proceedings of Colloquium on Temporal Logic in Specification*, pages 402–450. Springer-Verlag, 1989. LNCS 398.

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

[Gad88] S. K. Gadia. A homogeneous relational model and query languages for temporal databases. *TODS*, 13(4):418–448, 1988.

[JM80] S. Jones and P.J. Mason. Handling the time dimension in a data base. In *Proc. International Conference on Data Bases*, pages 65–83, Heyden, July 1980. British Computer Society.

[Kam68] H. Kamp. *On the Tense Logic and the Theory of Order.* PhD thesis, UCLA, 1968.

[Klu82] A. Klug. Equivalence of relational algebra and relational calculus query languages having aggregate functions. *Journal of the ACM*, 29(3):699–717, July 1982.

[Kro87] F. Kroger. *Temporal Logic of Programs.* Springer-Verlag, 1987. EATCS Monographs on Theoretical Computer Science.

[KSW90] F. Kabanza, J.-M. Stevenne, and P. Wolper. Handling infinite temporal data. In *Proceedings of PODS Symposium*, pages 392–403, 1990.

[Lor87] R.G. Lorentzos, N.A.; Johnson. TRA: A model for a temporal relational algebra. In *Proceedings of the Conference on Temporal Aspects in Information Systems*, pages 99–112, France, may 1987. AFCET.

[McK86] E. McKenzie. Bibliography: Temporal databases. *ACM SIGMOD Record*, 15(4):40–52, December 1986.

[MS89] E. McKenzie and R. Snodgrass. An evaluation of algebras incorporating time. Technical Report TR89-22, University of Arizona, September 1989.

[MS90] E. McKenzie and R. Snodgrass. Schema evolution and the relational algebra. *Information Systems*, 15(2):207–232, 1990.

[NA87] S.B. Navathe and R. Ahmed. *TSQL* – a language interface for history databases. In *Proceedings of the Conference on Temporal Aspects ,in Information Systems*, pages 113–128, France, may 1987. AFCET.

[Qui53] W.v.o Quine. *From a Logical Point of View.* Harvard University Press, Cambridge, 1953.

[RKS88] M. A. Roth, H. Korth, and A. Silberschatz. Extended algebra and calculus for nested relational databases. *TODS*, 13(4):388–417, 1988.

[RU71] N. Rescher and A. Urquhart. *Temporal Logic.* Springer-Verlag, 1971.

[SA85] R. Snodgrass and I. Ahn. A taxonomy of time in databases. In *Proceedings of ACM SIGMOD*, pages 236–246, New York, 1985. ACM.

[Sar90] N.L. Sarda. Algebra and query language for a historical data model. *The Computer Journal*, 33(1):11–18, February 1990.

[SGM87] R. Snodgrass, S. Gomez, and E. McKenzie. Aggregates in the temporal query language tquel. Technical Report TempIS 16, University of North Carolina at Chapel Hill, July 1987.

[Sno87] R. Snodgrass. The temporal query language TQuel. *ACM Transactions on Database Systems*, 12(2):247–298, June 1987.

[Sno90] R. Snodgrass. Temporal databases: Status and research directions. *ACM SIGMOD Record*, 19(4):83–89, December 1990.

[Soo91] M.D. Soo. Bibliography on temporal databases. *ACM SIGMOD Record*, 20(1):14–23, March 1991.

[SS88] R. Stam and R. Snodgrass. A bibliography on temporal databases. *Database Engineering*, 7(4):231–239, December 1988.

[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.

[Tan86] A.U. Tansel. Adding time dimension to relational model and extending relational algebra. *Information Systems*, 11(4):343–355, 1986.

[TC90] A. Tuzhilin and J. Clifford. A temporal relational algebra as a basis for temporal relational completeness. In *International Conference on Very Large Databases*, pages 13–23, 1990.

[Tuz89] A. Tuzhilin. *Using Relational Discrete Event Systems and Models for Prediction of Future Behavior of Databases.* PhD thesis, New York University, October 1989.

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

[vB83] J.F.A.K. van Benthem. *The Logic of Time.* D. Reidel Publishing Company, 1983.