On Completeness of Historical Relational Query Languages

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

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

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

March 24, 1993

Working Paper Series
STERN IS-93-8

(replaces: IS-91-41)

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 expressive 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 a 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, NA89, 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 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, the question of whether they have the same modeling capabilities has remained a subject for debate. 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., [MS91a] refers to no fewer than twelve 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 and languages. 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 Chandra and Harel [CH80].

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

[1] This list is not exhaustive. For an overview of the area of time and databases see [AC86], [Sno90], and [TCG+93]; for an ongoing bibliography on the subject see [McK86], [SS88] and [Soo91].

between these two approaches, respectively, and discuss the relative power of the two approaches. We then propose two 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 (*TU*) and temporally grouped (*TG*), serves to structure the remainder of the paper.

In Section 3 we introduce the notions of *weak* and *strong completeness* for comparing the query languages of different data models. Then in Section 4 we apply these concepts of completeness separately to the temporally ungrouped and temporally grouped models. For the **temporally ungrouped models** we define three different languages: 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 strong completeness, which we call *TU-Completeness*, for temporally ungrouped models. In Section 5 we examine the **temporally grouped models**, and define an 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 strong completeness which we call *TG-Completeness*, for models of this type. In Section 6 we show that the ungrouped historical data models are only weakly complete with respect to the grouped historical models. However, we then show how the representation power of the ungrouped models and their languages can be extended to incorporate the *grouping* semantics, and show that this extended ungrouped model is strongly complete with respect to the grouped model. Finally, in Section 7 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 [SGM89]). Work in the spirit of [Klu82] could extend the results here in that direction if so desired. Finally, we limit our attention to *temporally homogeneous* relations ([Gad88]), i.e., relations whose tuples have attributes all defined over the same period of time, and do not incorporate schema evolution over time (as in [CC87]) because treatment

of these additional issues would significantly lengthen the paper, and because they have not been included in most of the proposed historical data models. 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 greatest common denominator of the various models proposed. In this way we have been able to apply our metrics of completeness fairly against several models in the paper. We conclude in Section 8 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., *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 representations 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, and models such as [Ari86, Sno87, Lor87, NA89, TC90]), 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, and models such as [Tan86, CC87, Gad88]) 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. 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

[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

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

Figure 2: Prototypical N1NF Historical Employee Relation

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. Such 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.

[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 the 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

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* these two approaches are formally equivalent ([vB83]), although interval time-stamping may be representationally more compact. Nearly all of the work on historical or temporal databases has assumed a discrete temporal domain ([MS91a]). 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-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 rest of this paper, 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 formally 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 completeness (*TG-Completeness*) and temporally ungrouped completeness (*TU-Completeness*).

2.1 *TU*: 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 ([Mai83]). We refer to the canonical temporally ungrouped historical database model, now to be presented, as *TU*.

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.

$\mathbf{T} = \{t_0, t_1, \ldots, t_i, \ldots\}$ 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.

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. When we need to distinguish between these two types of attributes, we will refer to those attributes in U_A as **value attributes**, and to $TIME$ as a **temporal attribute**.

A *TU* **relation scheme** $\mathbf{R_{TU}}$ is a 3-tuple $\mathbf{R_{TU}} =< \mathbf{A}, \mathbf{K}, \mathbf{DOM} >$ where:

1. $\mathbf{A} \cup \{TIME\}$ is the set of attributes of scheme $\mathbf{R_{TU}}$, where $\mathbf{A} \subseteq U_A$.
2. 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]. We call this the **key constraint.**

3. $\mathbf{DOM} : \mathbf{A} \cup \{TIME\} \rightarrow U_D \cup \{\mathbf{T}\}$ is a function that assigns to each value attribute $A_i \in \mathbf{A}$ from scheme $\mathbf{R_{TU}}$ a value domain, denoted $DOM(A_i, \mathbf{R_{TU}})$, and to $TIME$ the temporal domain $\mathbf{T}$.

A *TU* **relational database scheme** $\mathbf{DB_{TU}} = \{R_{1_{TU}}, R_{2_{TU}}, \ldots, R_{n_{TU}}\}$ is a finite set of *TU* historical relation schemes. A *TU* **tuple** $\mathbf{t_{TU}}$ on scheme $\mathbf{R_{TU}} =< \mathbf{A}, \mathbf{K}, \mathbf{DOM} >$ is a function $\mathbf{t_{TU}} : \mathbf{A} \cup \{TIME\} \rightarrow \mathbf{D} \cup \mathbf{T}$, that associates with each attribute $A_i \in \mathbf{A}$ a value in $\mathbf{DOM}(A_i, \mathbf{R_{TU}})$ and with $TIME$ a value in $\mathbf{T}$. A *TU* **relation** $\mathbf{r_{TU}}$ on scheme $\mathbf{R_{TU}} =< \mathbf{A}, \mathbf{K}, \mathbf{DOM} >$ is a finite set of *TU* historical tuples on scheme $\mathbf{R_{TU}}$ that satisfies the key constraint. A *TU* **database** $\mathbf{d_{TU}} = \{\mathbf{r_{1_{TU}}}, \mathbf{r_{2_{TU}}}, \ldots, \mathbf{r_{n_{TU}}}\}$ is a set of *TU* 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 *TG*: 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. We refer to this canonical temporally grouped historical database model, now to be presented, as *TG*.

Let U_D, $\mathbf{D}$, Θ_{D_i}, $\mathbf{T}$, and U_A be as for the canonical temporally ungrouped relation structure, *TU*, above. $\mathbf{T}$ 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[5]. Each of

[4]The symbol "$\rightarrow$" is used here to represent a *functional dependency.*

[5]Our use of the term *temporal domain* should not be confused with Gadia's use of the same term in [Gad88]; our use of the term is analogous to Gadia's term *temporal assignment.*

these partial functions defines an association between each time instance in some lifespan **L**, and a value in some value domain D_i, and thus provides a means of modeling the changing of an attribute's value over time.

A *TG* **relation scheme** $\mathbf{R_{TG}}$ is a 3-tuple $\mathbf{R_{TG}} =< \mathbf{A}, \mathbf{K}, \mathbf{DOM} >$ where:

1. $\mathbf{A} \subseteq U_A$ is the set of **attributes** of scheme $\mathbf{R_{TG}}$.
2. $\mathbf{K} \subseteq \mathbf{A}$ is the **key** of scheme $\mathbf{R_{TG}}$, i.e., $\mathbf{K} \rightarrow \mathbf{A}$.
3. $\mathbf{DOM} : \mathbf{A} \rightarrow U_D \cup \{\mathbf{T}\}$ 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 *TG* **database scheme** $\mathbf{DB_{TG}} = \{\mathbf{R_{1_{TG}}}, \mathbf{R_{2_{TG}}}, \ldots, \mathbf{R_{n_{TG}}}\}$ is a finite set of *TG* relation schemes.

A *TG* **tuple** $\mathbf{t_{TG}}$ on scheme $\mathbf{R_{TG}} = < \mathbf{A}, \mathbf{K}, \mathbf{DOM} >$ is a function that associates with each attribute $A_i \in \mathbf{A}$ a temporal-based function from the **tuple lifespan** (any subset of **T**), denoted $\mathbf{t_{TG}}.\mathbf{l}$, to the domain assigned to attribute A_i. That is, $\mathbf{t_{TG}}(A_i) : \mathbf{t_{TG}}.\mathbf{l} \rightarrow \mathbf{DOM}(A_i)$[6].

A *TG* **relation** $\mathbf{r_{TG}}$ on scheme $\mathbf{R_{TG}} =< \mathbf{A}, \mathbf{K}, \mathbf{DOM} >$ is a finite set of *TG* tuples on scheme $\mathbf{R_{TG}}$ such that given any two tuples $\mathbf{t_{1_{TG}}}$ and $\mathbf{t_{2_{TG}}}$ in $\mathbf{r_{TG}}$, $\forall t \in (\mathbf{t_{1_{TG}}}.\mathbf{l} \cap \mathbf{t_{2_{TG}}}.\mathbf{l})$ $\exists A_i \in \mathbf{K}$ such that $\mathbf{t_{1_{TG}}}(A_i)(t) \neq \mathbf{t_{2_{TG}}}(A_i)(t)$. This notion of a key simply specifies that there can be no time in which two different tuples agree on the key. Note that the key is *not* required to be constant over time. The **EMPLOYEE** relation in Figure 2, with three tuples, is an example of a *TG* relation. Its (presumed) key, **NAME**, is an example of a non-constant key.

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

[6]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.

2.3 Comparison of Grouped and Ungrouped Models

Many researchers have assumed that these two different approaches – temporally grouped and temporally ungrouped – were equivalent in power, the differences simply a matter of style[7]. In exploring *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 analogous to our notion of a *corresponding* ungrouped model, to be introduced in Section 6. However, rather than emphasizing the importance of the *differences* between these two approaches, he focuses on their similarity by defining them as being *weakly equivalent.* Essentially he shows that you can 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. (A similar point is made in [TG92] with respect to nested relations.)

What we will argue in the rest of this section is that the differences *are* important, and the modeling and querying 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 grouped model and the ungrouped model with group identifiers.

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 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 tuples (i.e., tuples describing the same object). 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

[7]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 capable of representing identical information.

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 very reasonable query asks simply that the DBMS treat the salary history (temperature history, rainfall history, etc.) as a first-class object that can be queried, manipulated, etc. ([Cli82], [SS87]).

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. These considerations motivate the following definition.

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, and

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	$\{1,2\}$

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

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

(a) in TG Model (b) in Time Point TU Model (c) in Time-Interval TU Model

Figure 4: Answers to Q2

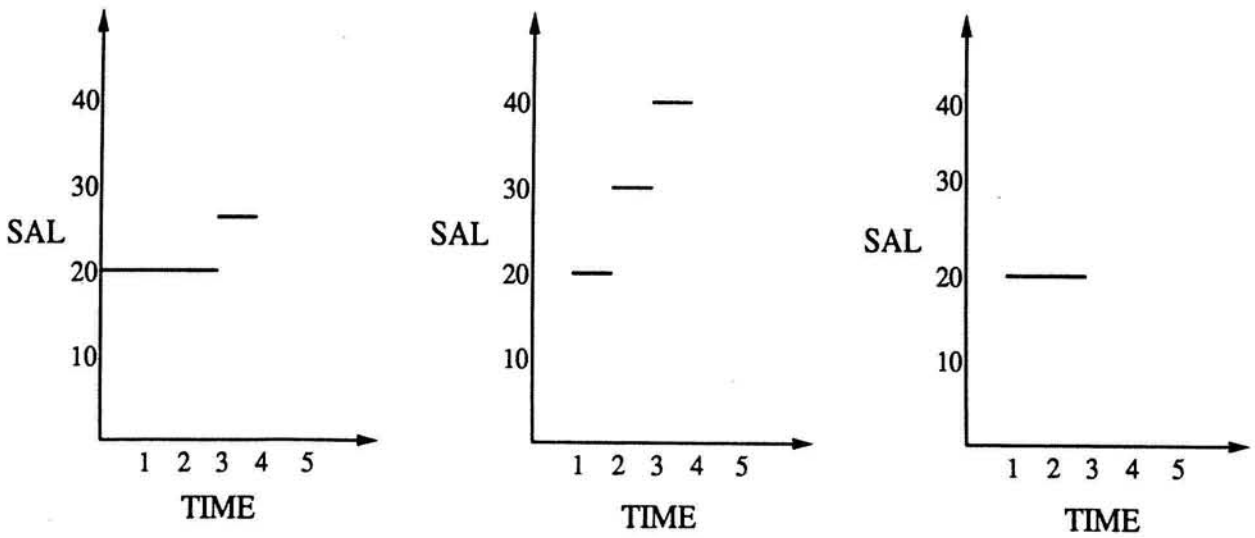

Figure 5: Graph of Employees' Salaries from Figure 4(a)

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

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(b) and 4(c)), 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 take the whole object and look at it and inspect it, it falls through your fingers moment by moment. Only in a temporally grouped model 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(c)). Only a Temporally Grouped model, like *TG*, with relations such as the one in Figure 2, exhibits **temporal value integrity**, and therefore provides the correct answer to this query, as in Figure 4(a).

The two representation schemes, *TG* and *TU*, complicate the question of *completeness* of query languages for an historical relational data model. We are therefore led to define two notions of completeness for historical database query languages, one based on *TG* models and

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

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 logic with explicit temporal variables, and an extended relational algebra ([TC90]). We then define *TG-Completeness* in terms of a calculus that 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 grouping can be simulated explicitly.

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

This query, here expressed in a temporal calculus (to be described in Section 4.2), could also be expressed in other ungrouped languages such as TQuel ([Sno87]). The answer, as 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

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

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 in Section 5.2, which incorporate a data extraction view ([AU79]) of query languages.

Temporally grouped models support temporal values directly – they incorporate the temporal component into the historical model at an 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 Section 6 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 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 a shorthand notation for intervals, where e.g., $[t_1, t_2)$ denotes the set of all points t, $t_1 \leq t < t_2$. 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.

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

3 Completeness

Most of the database literature that has addressed the question of query language *completeness* has done so within the context of a single data model. As with the work of Codd [Cod72] comparing the relational algebra and calculus, two different query languages for the same data model are compared. Informally, one language is said to be complete with respect to another if it can express all of its queries, and two languages are equivalent if they are mutually complete with respect to one another. This notion is made precise in the following definition.

Definition. Given a data model M and two query languages L_1 and L_2 for M, language L_2 *is complete with respect to* L_1 iff

$$\forall db \in M, \forall q \in L1, \exists q' \in L2\ [q'(db) = q(db)]$$

where *db* is a database in M and q is a query in $L1$. Languages $L2$ and $L1$ are *equivalent* iff $L2$ is complete with respect to $L1$ and $L1$ is complete with respect to $L2$.

It is more difficult to compare two *different* data models with different query languages. Let us denote a data model as a 2-tuple $M = (DS, QL)$ consisting of a structural component

(DS) and a query language (QL). For example, we might consider the relational model as $RM = (R, RC)$ where R is the set of all possible relations and RC is the relational calculus. If we are interested in comparing two data models $M_1 = (DS_1, QL_1)$ and $M_2 = (DS_2, QL_2)$, we must consider a mapping between the structures of the different models as well as mappings between the query languages.

Obviously, not all comparisons between different data models are interesting. Informally, we are only interested in comparing two models whose structures can represent *the same information.* If, e.g., data model DM_1 were the relational model and data model DM_2 had structures capable of representing only a single "object", there would be little point in comparing them. Intuitively, DM_2 is not *large enough* to represent all of the possible relations in the relational model, and to distinguish between them.

In order to compare what we may call the *representation power* of the data models themselves, we need to have a mapping $\Omega_{M_1 M_2}$ from structures in one model (M_1) to structures in the other (M_2). Such an $\Omega_{M_1 M_2}$, we contend, is only interesting if it preserves the identity of all of the application-specific base values in M_1, such as the employee names, salaries, etc. At a minimum, if there is to be any possibility that M_2 is going to be able to represent the same information as M_1, there must be a 1-1 mapping from these base value domains in M_1 into M_2. Then some higher-level mapping from the "structures" of M_1 into the "structures" of M_2 must be specified. Once we have two data models that are in this sense *reasonable* to compare, we can examine the issue of whether or not their query languages are in some sense *complete* with respect to one another, or *equivalent* to each other. We will want to talk, not only of the mapping from an "object" in one model to its representation in the other, but also of the mapping applied to an entire database. So we will abuse notation slightly by extending $\Omega_{M_1 M_2}$ in the obvious way. If $D = \{x_1, \ldots x_n\}$ is an instance of data model M_1 (for example, a set of relations), then $\Omega_{M_1 M_2}(D) = \{\Omega_{M_1 M_2}(x_1), \ldots \Omega_{M_1 M_2}(x_n)\}$ is the image of that set of objects (e.g., relations) in M_2 under the mapping.

Let us try to be more precise about what constitutes a "reasonable mapping" between the structures of two different data models. Our definitions will be simpler if we confine the discussion to two "relationally-based" temporal models, i.e., models whose structures are relations consisting of tuples with attributes and values[8]. Accordingly, we will focus on comparing data models which are quite similar in terms of the functionality that they

[8]The entire issue of formal comparisons of different data models is an interesting one which, in its full generality, is necessarily outside of the scope of this paper.

attempt to capture (time-varying values, and temporal grouping) and ignore more general issues of entirely different representation schemes, different normal forms, etc. Therefore, we will focus on comparing two databases which have a comparable set of relations and attributes, but which incorporate time or grouping in different ways.

To this end, we partition the set of attributes of a relation into the set of *distinguished* attributes (e.g., $TIME$) that are required in each relation of the model and the set of *non-distinguished* attributes (e.g., SAL, DEPT, etc.) that are user-specified. We assume that for the models under consideration the following two functions can be specified. $A_{ND}(r)$ denotes the set of all non-distinguished attributes in the relation scheme of r in the model. For each non-distinguished attribute A, the *active domain* of A in a relation r from r denoted $AD(A, r)$, is the set of all the values in the value domain (as defined in Section 2.1) that occur in the A-column of the tuples in r.

Definition. A relation r_2 in data model M_2 *corresponds* to relation r_1 in data model M_1 iff:

1. $A_{ND}(r_1) = A_{ND}(r_2)$
2. $\forall A \in A_{ND}(r_1)\ [AD(A, r_1) = AD(A, r_2)]$

The main purpose of this definition is to eliminate obviously trivial mappings such as a mapping to a model consisting of only a single relation with a single attribute and a single tuple, which could (trivially) be defined to be "equivalent" to any other data model. What this definition requires is that for a relation r_2 to correspond to relation r_1 (1) it must be defined over the same set of non-distinguished attributes, and (2) all of the values of the non-distinguished attributes that appear in relation r_1 also appear in relation r_2. Note that condition 1 only requires the same set of non-distinguished attributes because the distinguished attributes of one model may not be required in a different model which nevertheless can be said to represent "the same information." For example, TU models require the distinguished attribute $TIME$, whereas TG models do not.

Definition. Given data models $M_1 = (DS_1, QL_1)$ and $M_2 = (DS_2, QL_2)$, a mapping

$$\Omega_{M_1 M_2} : DS_1 \rightarrow DS_2$$

from the relations of M_1 to the relations of M_2 is a *correspondence mapping* iff:

$$\forall r_1 \in M_1 \; [\Omega_{M_1 M_2}(r_1) \text{ corresponds to } r_1]$$

For example, the relation in Figure 1 could be obtained from the relation in Figure 2 through the obvious "unnest" correspondence mapping (see Section 6). With these definitions we are prepared to compare different data models and query languages.

Definition. Given data models $M_1 = (DS_1, QL_1)$ and $M_2 = (DS_2, QL_2)$, we say that M_2 *is weakly complete with respect to* M_1 iff there exist two mappings:

$$\Omega_{M_1 M_2} \; : \; DS_1 \rightarrow DS_2 \;\; \text{and}$$

$$\Gamma_{M_1 M_2} \; : \; QL_1 \rightarrow QL_2 \;\; \text{such that}$$

1. $\Omega_{M_1 M_2}$ is a correspondence mapping, and
2. for all ϕ in QL_1 and for all instances D of the structures DS_1 :

$$\Omega_{M_1 M_2}(\phi(D)) = \Gamma_{M_1 M_2}(\phi)(\Omega_{M_1 M_2}(D))$$

Definition. Given data models $M_1 = (DS_1, QL_1)$ and $M_2 = (DS_2, QL_2)$, we say that M_2 *is strongly complete with respect to* M_1 iff there exist two mappings:

$$\Omega_{M_1 M_2} \; : \; DS_1 \rightarrow DS_2 \;\; \text{and}$$

$$\Gamma_{M_1 M_2} \; : \; QL_1 \rightarrow QL_2 \;\; \text{such that}$$

1. $\Omega_{M_1 M_2}$ is a correspondence mapping, and is 1-1
2. for all ϕ in QL_1 and for all instances D of the structures DS_1 :

$$\Omega_{M_1 M_2}(\phi(D)) = \Gamma_{M_1 M_2}(\phi)(\Omega_{M_1 M_2}(D))$$

Figure 9 illustrates these two possible completeness relationships between M_2 and M_1. In the case of *weak completeness*, $\Omega_{M_1 M_2}$ is not required to be 1-1, and so several relations in M_1 could map to the *same* relation in M_2. Strong completeness requires $\Omega_{M_1 M_2}$ to be 1-1. Clearly, *strong completeness* implies *weak completeness.* Finally, we can define a derivative notion of *equivalence* based upon these two notions of completeness:

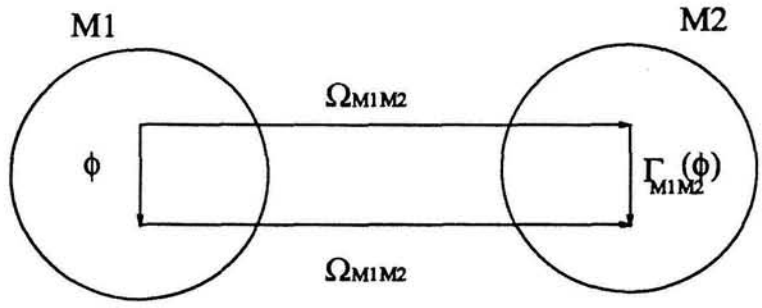

Figure 9: M_2 complete with respect to M_1

Definition. Given data models $M_1 = (DS_1, QL_1)$ and $M_2 = (DS_2, QL_2)$, we say that M_1 *is weakly (strongly) equivalent to* M_2 iff M_1 is weakly (strongly) complete with respect to M_2 and M_2 is weakly (strongly) complete with respect to M_1 .

Note that our definition of weak equivalence is different from the related concept of weak equivalence in [Gad88]. Our notion compares two different data models. Gadia's notion compares two (grouped) relations which map to the same ungrouped relation via an "unnest" operation.

4 Historical Relational Completeness for Ungrouped Languages

In this section, we define the concept of **ungrouped temporal relational completeness**, *TU-Completeness*. It will be based on the canonical temporally ungrouped model *TU*, and 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 (conventional) temporal logic in the next subsection.

4.1 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. With additional axioms, we can introduce other models of time, e.g.,

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 10: Temporal Operators **until** and **since**

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 4.3 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[9]. 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 10.

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[10]. 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 an **historical relation**, i.e. a relation that changes over time. An instance of an historical

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

[10]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 11: Temporal Structure for Relation **EMPLOYEE**.

relation R at time t, $K_t(R)$, will be denoted R_t.

An historical database represented in a certain ungrouped historical data model, such as the (historical component of) TQuel data model [Sno87], or the **TRA** model of [Lor87], defines a temporal, 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 1 is shown in Figure 11.

Given a temporal structure for temporal logic predicates, we can interpret arbitrary temporal logic formulae in the standard inductive way [Kro87]. For example, the meaning of **until** and **since** operators in Figure 10 can be defined inductively in terms of the temporal structures for the arguments to the operator.

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 10 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.

4.2 Temporal Calculi *TL* and *TC*

In the previous section 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 finite relations at all time instances, and if these relations contain *only* data from the database. Our definition of the safety of temporal logic formulae is analogous to that of the snapshot relational case [Ull88] with the added assertion that the temporal operators **until** and **since** produce safe formulae if operands of these operators constitute safe formulae. 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 all the free variables in ϕ. 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 an 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

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

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

Figure 12: Historical Relations **UNEMPL** and **TAXES**

defined exactly as in the standard relational case. We denote the temporal calculus based on these queries as *TC*.

Note that in both calculi a query operates on historical relations and returns an 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, and historical relation **TAXES**(NAME, TAX) specifying taxes a person paid in a certain year. Figure 12 gives examples of such relations. 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) (shown in Figure 13) 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:

$$\textbf{GOOD\_CITIZEN} = \{NAME, t \mid (\exists t')(\textbf{UNEMPL}(NAME, t') \wedge (\forall t'')(t' < t'' < t \Rightarrow \textbf{TAXES}(NAME, TAX, t'')))\}$$

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 explicit

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**

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.

4.3 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, defined over the same domains. 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 $S_{t+1} = (R_t \cap S_t) \cup Q_t$, for $t_1 \leq t < t_n$, and $S_{t_1} = \emptyset$.

O6b: *Past linear recursive operator*: $\mathbf{S} = \mathbf{L_P}(\mathbf{R}, \mathbf{Q})$ iff $S_{t-1} = (R_t \cap S_t) \cup Q_t$, for $t_1 < t \leq t_n$, and $S_{t_n} = \emptyset$.

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 reduce to these operators in the degenerate case when $t_1 = t_n$.

Example 2 The relation **GOOD_CITIZEN**(NAME) in Figure 13 can be computed in *TA* as follows. Set **TAXES1** = π_{NAME} (**TAXES**). Then

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

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

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.

4.4 Ungrouped Historical Relational Completeness

Comparing ungrouped data models is easy because in some sense all of the *structures* of these data models are isomorphic.

Definition. A data model $M = (DS, QL)$ *is a temporally ungrouped historical data model* if DS is isomorphic to the canonical ungrouped historical relational model *TU*.

Definition. A temporally ungrouped historical data model $M = (DS, QL)$ is *TU-Complete* if M is strongly complete with respect to $M_{TU} = (TU, TC)$(or, equivalently, to $M_{TU'} = (TU, TL)$).

Our proposal to use the three languages considered in this section, the historical relational algebra (*TA*) and the two temporal calculi (*TC* and *TL*), as the linguistic basis for ungrouped historical relational completeness is based on a number of 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, in a future and a past version. 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 reduce to the relational algebra and calculus in the degenerate case when the time set consists of only one instance. Therefore, the notion of *TU-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 *TU* with the two calculi and the algebra presented in this section as a basis of ungrouped historical completeness, *TU-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 7 when we examine the completeness of a number of models proposed in the literature.

5 Historical Relational Completeness for Grouped Languages

In this section we define a concept of **grouped historical relational completeness**, *TG-Completeness*. The basis for this concept of completeness is the canonical temporally grouped model *TG*, and a grouped, tuple-based historical relational calculus, L_h.

5.1 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, that conforms to the canonical grouped relations defined in Section 2.2. L_h 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 **D**, universe of times **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 **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 binary 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, θ is a binary predicate on domain $\mathbf{T}$, and τ is a temporal constant, then $t_1\theta t_2$, $\tau\theta t_1$, and $t_1 = \theta\tau$ 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.

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

1. $[e_1.A_1, \ldots, e_n.A_n : t]$, called a *target list*, consists of

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

(b) the free temporal variable t.

2. ϕ is a formula.

As a convenience, when a tuple variable e 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 that 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) $t_1\theta t_2$ is true just in case the time denoted by t_1 stands in the relation θ with t_2, and false otherwise; similarly for $\tau\theta t_1$ and $t_1\theta\tau$.

(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.

A *Query Expression* $[e_1.A_1, \ldots, e_n.A_n : t]\phi$ of L_h denotes an 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.

5.2 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 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 $t_i \in e_j.l$.
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 value term.
   (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 rule (4) above.

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.

5.3 Examples of L_h Queries

We now 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 name and salary histories 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.

What are the names of the managers for whom Tom has worked?

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

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

Give me all the information about 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 **EMPLOYEE**(e). If this conjunct were added into the subformula the entire formula would be safe, and would also be correct.

5.4 Grouped Historical Relational Completeness

We propose to use the canonical temporally grouped model TG, along with the many-sorted calculus L_h considered in this section, as the basis for grouped historical relational completeness. That is, we propose the following definitions.

Definition. A data model $M = (DS, QL)$ *is a temporally grouped historical data model* if DS is isomorphic to the canonical ungrouped historical relational model TG.

Definition. A temporally grouped historical data model $M = (DS, QL)$ is *TG-Complete* if M is strongly complete with respect to $M_{TG} = (TG, L_h)$.

The reasons that support this choice of TG with L_h as an appropriate metric for *TG-Completeness* are essentially similar to those that motivated out choice of the metric(s) for *TU-Completeness* (Section 4.4). 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 *TU-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 6 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.

6 Relationship Between Historically Grouped and Historically Ungrouped Completeness

The question naturally arises as to the relative expressive power of the two approaches, temporally grouped and temporally ungrouped, to representing historical relations. Specifically, we wish to compare the expressive powers of the temporally grouped historical data model $M_{TG} = (TG, L_h)$ and the temporally ungrouped historical data model $M_{TU} = (TU, TC)$.

A database db_{TG} in M_{TG} consists of a set of relations on a set of schemes $\{R_1, \ldots R_n\}$; for our purposes we can view each scheme as simply its set of attributes, i.e. $R_i = \{A_{i_1}, \ldots A_{i_m}\}$. We want to compare such a database with its counterpart in the ungrouped model. By a *corresponding database* to db_{TG} in M_{TU} we mean a database in M_{TU} derived through a correspondence mapping Ω (as defined in Section 3). This corresponding database likewise consists of a set of relations on schemes $\{R'_1, \ldots R'_n\}$, where each relation scheme has the same set of attributes plus the additional attribute $TIME$, i.e. $R'_i = \{A_{i_1}, \ldots A_{i_m}, TIME\}$. The following temporal *unnest* operation (similar to the *snapshot-valued function* defined in [Gad86]) provides an example of the corresponding mapping[11]. Let $r(A_1, \ldots, A_n)$ be a relation in db_{TG}. Then *unnest(R)* is a relation in db_{TU}-consisting of a set of tuples $(a_1, \ldots, a_n, t)$ such that there is an historical tuple e in r satisfying the condition $e.A_1(t) = a_1, \ldots, e.A_n(t) = a_n$.

We want to study the relationship between grouped and the corresponding ungrouped models. To begin with, we establish the relationship between their relational schemes. We start with the following definition.

Definition. A relation scheme R is more expressive than a relation scheme R' if the cardinality of the set of all possible relations on scheme R is greater than the cardinality of the set of all possible relations on scheme R'.

The following theorem compares the expressiveness of the grouped and ungrouped data models.

Theorem 1 *For any database scheme in the data model TG of $M_{TG} = (TG, L_h)$ the corresponding database scheme in the data model TU of M_{TU} is less expressive.*

Sketch of Proof: Let $\mathbf{D}$ be a value domain, and $\mathbf{T}$ be a non-trivial time domain (having more than one value), as defined in Section 2. Let $R_{TG}(A_1, \ldots, A_n)$ be a relation scheme in the TG model and $R_{TU}(A_1, \ldots, A_n, T)$ be the corresponding scheme in the TU model.

By inspection, there can be $(\mathbf{D}^n \cup \{\perp\})^{\mathbf{T}}$ possible historic tuples for the scheme $R_{TG}(A_1, \ldots, A_n)$, where $\perp$ accounts for the situation where data values may not be defined for some values of time. Therefore, there are $\alpha = 2^{(\mathbf{D}^n \cup \{\perp\})^{\mathbf{T}}}$ possible historic relations with schema $R_{TG}(A_1, \ldots, A_n)$. In comparison, there are $\mathbf{D}^n \times \mathbf{T}$ possible ungrouped tuples for the scheme $R_{TU}(A_1, \ldots, A_n, T)$ and $\beta = 2^{\mathbf{D}^n \times \mathbf{T}}$ possible relations.

[11] We omit the details of a discussion of the model $M_{TU_2} = (TU_2, TC_2)$ whose relations represent the temporal dimension with two time attributes, because it is well known that the two representations are equivalent for discrete time.

If we assume that the domains **D** and **T** are infinite, then $\beta < \alpha$ because the cardinality of the power set is greater than the cardinality of the set itself. If we assume that the domains **D** and **T** are finite, then again $\beta < \alpha$. The latter follows from simple combinatorial considerations. □

Corollary 2 *For any database scheme in the data model TG of $M_{TG} = (TG, L_h)$ the corresponding database scheme in the data model TU_2 of M_{TU_2} is less expressive.*

Theorem 1 will help us to establish the relationship between the language L_h and the corresponding language *TC*.

Theorem 3 *For any given database scheme in $M_{TG} = (TG, L_h)$, the corresponding database scheme in $M_{TU} = (TU, TC)$ is weakly equivalent and not strongly equivalent.*

Sketch of Proof: The weak completeness of $M_{TU} = (TU, TC)$ with respect to $M_{TG} = (TG, L_h)$ immediately follows from the choice of *unnest* as the structural correspondence mapping. The weak completeness of $M_{TG} = (TG, L_h)$ with respect to $M_{TU} = (TU, TC)$ immediately follows from the choice of the *identity group mapping* as the structural correspondence mapping. This mapping maps each tuple in $M_{TU} = (TU, TC)$ to a singleton grouped tuple in $M_{TG} = (TG, L_h)$. Lack of strong equivalence immediately follows from Theorem 1 and the definition of strong equivalence. □

We will now demonstrate how ungrouped historical relations can be used to *simulate* grouping through the use of an additional, *distinguished* attribute *O* for keeping track of the groups. Specifically, we will show how the ungrouped model can represent a grouped relation on scheme $R_i = \{A_{i_1}, \ldots A_{i_m}\}$ by a relation on scheme $R_i'' = \{O, A_{i_1}, \ldots A_{i_m}, TIME\}$.

6.1 Extending *TU* and *TC* to Support Grouping

To support grouping in the ungrouped model *TU*, we introduce an additional **group identifier** attribute for each relation in *TU*. For example, a *TC* relation scheme $R(A, B, T)$ is extended with an additional attribute *O* and becomes $R(O, A, B, T)$, where *O* is a group identifier (group-id) attribute. The role of the grouping attribute – identifying *groups* of tuples which correspond to a single *object* – serves a role similar to that of object identifiers in object-oriented databases, or the *surrogates* in certain extended relational models ([HOT76, Dat83]). We call the model with these relations TU_g.

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

We then introduce a 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 sorts are defined exactly as for TC in Section 4. Intuitively, the grouping sort divides a TU_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 might be represented in the TU_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 (e.g., $=$, $>$, $\geq$) 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 TU_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-ids. 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. We call the resulting model, consisting of the ungrouped relations extended with a group identifier, and the logic extended with the group sort, $M_{TU_g} = (TU_g, TC_g)$.

While these axioms may be stronger than necessary, we shall nevertheless demonstrate that they are sufficient to define the model $M_{TU_g} = (TU_g, TC_g)$ which we show to be strongly equivalent to the model M_{TG} consisting of TG with L_h.

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 5

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

It can be expressed in TC_g as

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

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 can divide a set of tuples $S = \{< < o_1, x_1 >, \ldots, < o_n, x_n >, t \ >\}$ into groups of tuples

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

Then we encode the group of tuples $G(o_1, \ldots, o_n, S)$ with an **encoding function**

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

where $\mathbf{D}$ is the universe of all possible values (Section 2.1), O is a set of group-id's, and $\mathbf{T}$ is the set of times. An encoding function is a bijection between finite sets in $2^{\mathbf{D^k \times T}}$ and O. It is well-known that such encoding functions are definable for finite $\mathbf{D}$ and $\mathbf{T}$ ([End72]). In the case of countably infinite $\mathbf{D}$ and/or $\mathbf{T}$, the definability of such a function follows from the finiteness of the relations in the database and their representation.

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 use of the universal quantifier ($\forall$).

2. It contains 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. In each maximal conjunct $F_1 \wedge \ldots \wedge F_n$ of ϕ the following conditions must hold:

   (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 rule (4) above.

This definition of safety mirrors the definition of safety of L_h formulae as defined in Section 5. 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 remainder of the paper.

6.2 Strong Equivalence of M_{TG} and M_{TU_g}

We begin this section by defining two correspondence mappings Ω_{UG} and Ω_{GU} between the structures of the two data models $M_{TG} = (TG, L_h)$ and $M_{TU_g} = (TU_g, TC_g)$, and show that they are inverses of each other. We then consider their query languages, and define mappings Γ_{UG} and Γ_{GU} between them. Finally, we show that the models M_{TG} and M_{TU_g} are strongly equivalent.

6.2.1 Relationship Between Data Models of TG and TU_g

In this section we define mappings between the structures of the ungrouped and grouped historical models. Ω_{UG} maps TU_g relations into TG relations; intuitively, it groups TU_g tuples with the same group-id into a single group that becomes an historical tuple. Ω_{GU} maps TG relations to TU_g relations; intuitively, it ungroups an historical tuple into a set of tuples with the same group-id. Ω_{GU} can be considered as an extension of Gadia's temporal *unnest* operator described in Section 6 that supports group identifiers.

Formally, the mapping Ω_{UG} from TU_g to TG relations is defined as follows. Let r and r' be TU_g and TG 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 an 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"; [TG92] addresses this issue in detail.)

6.2.2 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.

Let ϕ be a safe TC_g formula. The formula $\Gamma_{UG}(\phi)$ is obtained from ϕ by replacing all the atomic formulae in 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. Each range expression $\psi_i = (\exists x_{ij_1}) \ldots (\exists x_{ij_i}) R_i(o_i, x_{i1}, \ldots, x_{in_i}, t)$[12] is replaced with

[12]Range expressions were introduced in Section 6.1 when safe TC_g queries were introduced.

the expression $R_i(e_i) \wedge t \in e_i.l$.

2. 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[13]. 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.

3. Replace each quantification $(\exists o)$ in ϕ with the quantification $(\exists e)$ 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 2 that replaces o.

4. Each quantification $(\exists x)$ in ϕ remains unchanged in $\Gamma_{UG}(\phi)$. For any free variables x remaining in ϕ , prepend $(\exists x)$ to $\Gamma_{UG}(\phi)$.

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:

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

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 TU_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.

[13]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, as it simplifies the presentation, and the transformation is still correct.