On Temporal Grouping

James Clifford
Information Systems Department, Stern School of Business
New York , NY, USA

Albert Croker
Statistics and Computer Information Systems, Baruch College
New York, NY, USA

Fabio Grandi
Dipartimento di Elettronica Informatica e Sistemistica, Universitá di Bologna
Bologna, ITALY

Alexander Tuzhilin
Information Systems Department, Stern School of Business
New York , NY, USA

On Temporal Grouping

Abstract

In this paper we address some of the concerns that have been expressed regarding the practicality of the temporally-grouped, or history-oriented, data modeling approach. Specifically, we address the concern over the lack of an algebra for this paradigm, by presenting such an algebra. In addition, we examine a number of semantic notions, including FDs, keys, coalescing, and restructuring, from a perspective of comparison between the temporally-grouped and temporally-ungrouped approaches. paradigms.

1 Introduction

A large variety of extensions of the relational data model and query languages to include time have been proposed in recent years (see [McK86, Soo91, SS88] for references to the growing body of literature on temporal databases). According to a fundamental structural property formalized in [CCT94], all the proposals can be classified into two main categories: *temporally ungrouped* and *temporally grouped* models and languages. In temporally ungrouped models, the temporal representation is realized at extensional level, by means of timestamps added to data values as additional attributes to represent their temporal validity. In temporally grouped models, the temporal dimension is implicit in the structure of data representation: attributes are represented as *histories* considered as a whole and without the introduction of distinguished attributes. Attribute histories can be regarded as functions which map time into attribute domains.

The most important property of *temporally grouped* models and languages concerns their formal *expressiveness*, which has been shown to be greater than the expressiveness of ungrouped ones [CCT94]. Moreover, it has also been shown [GST93] that history-oriented temporal query languages, providing a "grouped" point of view on data, can be more "natural" and friendly for human users as they support the concept of *history* as a first-class object of discourse.

A controversy regarding the two approaches was also brought up in the discussions of the *ARPA/NSF International Workshop on an Infrastructure for Temporal Databases*, held in Arlington in 1993. The main objective of the workshop was to establish a common foundation for the discipline of temporal databases and to develop a consensus grounding for further research and development. In a few words, while grouped models appeared more attractive from a logical perspective (being more expressive and also user-friendly), ungrouped models seemed more amenable to implementation at the state-of-the-art of commercial database technology, requiring almost minimal extensions of the 1NF relational model and SQL-92.

After the workshop, the definition of a system supporting temporal grouping has come together into the *history-oriented* entry of the "consensus glossary of temporal database concepts" [JCE$^+$94]. According to the final glossary definition, a DBMS is said to be *history-oriented* if:

1. It supports history unique identification (e.g. via time-invariant keys, surrogates or OIDs);

2. The integrity of histories as first-class objects is inherent in the model, in the sense that history-related integrity constraints might be expressed and enforced, and the data manipulation language provides a mechanism (e.g., history variables and quantification) for direct reference to *object-histories*.

Both terms, temporally grouped and history-oriented, will be freely used when appropriate in the rest of the paper.

As another follow-up of the workshop standardization efforts, a committee was formed with the purpose of designing a temporal extension of the SQL-92 language. The committee, gathering people from both academia and industrial world, released a language specification, named TSQL2, in September 1994 [SAA$^+$94, SAA$^+$94]. In the cradle of the controversy between grouped and ungrouped approaches, TSQL2 was born as a compromise solution. To this end, TSQL2 is provided with a surrogate data type [JeSn94] and special range variables [SnJe94]. Surrogates, which are system-generated identifiers, can be used to simulate the history identity inherent to a temporal grouped data model as shown in [CCT94], but their correct management is completely up to the user. TSQL2 range variables can be used as history and version variables [GST93] in order to simulate history-oriented queries, but their correct use is not straightforward and still is up to the user. Rather than being more clear and readable, history-oriented TSQL2 queries may in fact be very complex and not very natural.

In spite of this, several objections to a temporally grouped approach have been raised. The principal objections will be considered and discussed in this work.

First, a history-oriented DBMS seems too far removed from an SQL-92-based relational system to be put into practice. For instance, even if a 1NF relational model with surrogates could be adopted, the standard relational algebra would be inadequate for defining certain necessary operations (e.g. there are difficulties in dealing with surrogates in Cartesian Products). Also temporally grouped extensions of SQL-92 require the introduction of the notions of history and temporal object at the language level; notions which are beyond the classical perspective of pure relational languages and would rather require features of an object-oriented approach.

Second, perhaps the most severe objection is just the fact that, although logic- or SQL-based languages have been defined, no algebra is available for grouped models yet. The most important consequence thereof is the lack of an effective operational core on which any real language implementation (query processing and optimization strategies) can be based on.

Finally, while grouped models impose a structure on data tables, i.e., the history of stored data objects is the semantic criterion for grouping tuples, other temporal models and languages allow the restructuring of a table on an arbitrary set of columns. In particular, the restructuring operator [Gad86], which has been considered a desirable language feature and has been generalized by the TSQL2 variable mechanism, has seemed to allow a change in the key of a temporal relation.

In this paper we try to make a step towards the resolution of the difficulties or objections that have been raised concerning the grouped approach. We will present a history-oriented extension of SQL language as an alternative to TSQL2. We will introduce an algebra for grouped models, on which our SQL extension can effectively be based, and sketch a temporal completeness proof for it. Finally, we will discuss the notion of regrouping, and some other assorted semantic notions, in the context of our approach.

2 An Algebra for Temporally Grouped Models

It is important to have an equivalent algebra for a query language or calculus because query evaluation is typically done by mapping a query into an equivalent algebraic expression. Therefore, we have developed an algebra for a temporally grouped model that is equivalent to the grouped calculus L_h [CCT94], and we present it in this section. Because of the space limitations, we cannot describe the language L_h and assume that the reader is familiar with this language, as described in [CCT94].

However, before describing a grouped algebra, we would like to point out some difficulties in developing this algebra. To illustrate our points, consider the following safe L_h query:

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

where R and Q are grouped historical relations, e and e' are tuple variables, t and t' are temporal variables, $e.A : t$ is the target list, and $e.l$ is the lifespan of tuple e (the lifespan is the property of a tuple because the data model of L_h is homogeneous). Typically, when a calculus expression is mapped into the equivalent algebraic expression, this is usually done inductively on the number of operators in the calculus expression. However, if we do this with the L_h expression presented above, then the subformula

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

depends on two tuple variables (e and e') and two times (t and t'). This means that there cannot be any equivalent algebraic expression because such an expression would return the relation that has two time attributes in it and, thus, does not adhere to the data model of L_h.

For this reason, it has proven difficult to come up with a grouped algebra that it equivalent to the temporally grouped query language L_h which was proposed in [CCT94] as part of the canonical grouped historical relational model *TG*. To address this issue, we propose here a consistent extension to the data model $M_{TG} = (TG, L_h)$, and make small corresponding adjustments to its query language L_h as described in the next subsection.

2.1 An Inhomogeneous Historical Grouped Calculus L_{hi}

2.1.1 The Data Model TG_{hi}

In [CCT94] the data model $M_{TG} = (TG, L_h)$ was presented, where TG is a canonical temporally grouped data model and L_h its associated query language. The calculus L_h is based on the data model $M_{TG} = (TG, L_h)$ that treats the domain of *all* of the attributes of a historical relation as functions from time to values. The extended data model of $M_{TG_{hi}} = (TG_{hi}, L_{hi})$ considers two other types of domains for attribute values in addition to those of $M_{TG} = (TG, L_h)$, i.e. temporal and atemporal (value-based) attributes. More formally, $M_{TG_{hi}} = (TG_{hi}, L_{hi})$ is a three-sorted data model, with sorts:

- $\mathcal{T}$: a non-empty domain of times.
- $\mathcal{V}$: a non-empty domain of values.
- $\mathcal{F}$: is a set of functions $F : \mathcal{T} \rightarrow \mathcal{V}$.

An *relation* in $M_{TG_{hi}} = (TG_{hi}, L_{hi})$ is defined similarly to a relation in $M_{TG} = (TG, L_h)$, except it allows attribute values of all the three sorts, $\mathcal{T}$, $\mathcal{V}$, and $\mathcal{F}$, unlike the $M_{TG} = (TG, L_h)$ relations that allow only attribute values of sort $\mathcal{F}$. Attributes of sort $\mathcal{V}$ are used to model time invariant attributes, and attributes of sort $\mathcal{T}$ are used to model user-defined time.

In addition to the three-sortedness extension, we relax the homogeneity assumption used in M_{TG}, which required that the lifespans of historical attributes in historical tuples must all be the same. In other words, if A and B are attributes of sort $\mathcal{F}$ in relation R then *lifespan(A)* is not necessarily equal to *lifespan(B)*. By relaxing this assumption, we make the data model of $M_{TG_{hi}}$ *inhomogeneous.*

To summarize, $M_{TG_{hi}} = (TG_{hi}, L_{hi})$ is a consistent extension of the data model $M_{TG} = (TG, L_h)$ proposed in [CCT94]. The extensions make $M_{TG_{hi}}$ multi-sorted and temporally inhomogeneous.

Example 1 *Consider the following* **EMPLOYEE** *and* **DEPARTMENT** *relations. The first relation has the schema* **EMPLOYEE***(NAME, DOB, DEPT, SALARY), where attribute NAME has the sort $\mathcal{V}$, DOB has the sort $\mathcal{T}$, and DEPT and SALARY have the sort $\mathcal{F}$. The second relation has the schema* **DEPARTMENT***(DEPT, MGR), where attributes DEPT and MGR have the sort $\mathcal{F}$. Instances of both of these relations are presented in Figure 1. (Note that in these figures we use a shorthand notation for functions; for example, instead of specifying completely the function $\{< 0, Sales >, < 1, Sales >, < 2, Sales >< 3, Sales >\}$ we abbreviate the representation to $\{< [0, 3), Sales >\}$ to save space.)*

□

2.1.2 The Language L_{hi}

The language L_{hi} is a consistent extension of the language L_h; the straightforward language extensions address the extensions to the data model described above. First, L_{hi} is inhomogeneous, and, thus, the lifespan is a property of an individual attribute of sort $\mathcal{F}$. Therefore, a lifespan term in L_{hi} is of the form $e.A.l$, where e is a tuple variable, and A is an attribute of sort $\mathcal{F}$. Second, we extend L_h by making temporal terms to be not only temporal constants and variables but also terms of the form $e.A$, where A is an attribute of sort $\mathcal{T}$. Third, we extend L_h by making value terms to be not only value constants, domain variables, and expressions $e.A(t)$ for attributes A of sort $\mathcal{F}$, but also terms of the form $e.A$, where A is an attribute of sort $\mathcal{V}$.

EMPLOYEE			
NAME	*DOB*	*DEPT*	*SALARY*
Tom	8/15/51	$[0,4) \rightarrow$ Sales $[4,6] \rightarrow$ Mktg	$[2,3) \rightarrow$ 20K $[3,5) \rightarrow$ 30K $[5,6] \rightarrow$ 27K
Juni	2/28/61	$[2,6] \rightarrow$ Acctng	$[4,6] \rightarrow$ 28K
Ashley	5/25/59	$[1,3) \rightarrow$ Engrng $[3,4) \rightarrow$ Mktg $[5,6] \rightarrow$ Engrng	$[1,2) \rightarrow$ 27K $[2,4) \rightarrow$ 30K $[5,6] \rightarrow$ 35K

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

Figure 1: The Historical Grouped Relations **EMPLOYEE** and **DEPARTMENT**.

Finally, the definition of an L_{hi} query is extended from that of L_h. An expression $[\alpha_1, \ldots, \alpha_n]\phi$ is an L_{hi} *query* if ϕ is an L_{hi} formula, and α_i, $i = 1, \ldots, n$, are expressions in one of the following forms. Either α_i is a free variable of sort $\mathcal{T}$ or $\mathcal{V}$ from ϕ, or it is an expression of the form $e.A : t$, where e is a free tuple variable from ϕ, A is an attribute of a relation of sort $\mathcal{F}$ associated with e, and t is a free variable of sort $\mathcal{T}$, such that $t \in e.A.l$. Moreover, only those variables appearing in α_i, $i = 1, \ldots, n$, are the free variables in ϕ. This definition says that the attributes in the answer to an L_{hi} query can be of any of the three sorts $\mathcal{F}$, $\mathcal{T}$, and $\mathcal{V}$. In contrast to this, the attributes in the answer to an L_h query are only of the sort $\mathcal{F}$. Furthermore, the notion of safety, as introduced in L_h, has to be adjusted in a straightforward way to account for these changes to L_h.

To summarize, both the data model and the query language of $M_{TG} = (TG, L_h)$ are special, restricted cases of their consistent extension $M_{TG_{hi}} = (TG_{hi}, L_{hi})$.

We now introduce some examples of L_{hi} queries. These examples are based on relations **EMPLOYEE** and **DEPARTMENT** presented in Figure 1.

Example 2 *The query "Find the names, dates of birth, salary and departmental histories of the people who worked at time 5 in the Accounting department" can be expressed in L_{hi} as*

$$\begin{array}{l}
[e.NAME, e.DOB, e.SALARY : t_1, e.DEPT : t_2] \\
\quad \textbf{EMPLOYEE}(e) \wedge t_1 \in e.SALARY.l \wedge 5 \in e.DEPT.l \wedge \\
\qquad e.DEPT(5) = \text{“}Accntg\text{”} \wedge t_2 \in e.DEPT.l
\end{array}$$

Note that the answer to this query has attributes of all the three sorts $\mathcal{T}$, $\mathcal{V}$, and $\mathcal{F}$.

□

Example 3 *The query "What are the names of the managers for whom Tom has worked?" can be expressed in L_{hi} as*

$$\begin{aligned}
&[e_1.NAME]\ \mathbf{EMPLOYEE}(e_1)\wedge \\
&\quad \exists e_2 \exists t \exists d(\mathbf{EMPLOYEE}(e_2) \wedge t \in e_2.DEPT.l\ \wedge \\
&\qquad\qquad \mathbf{DEPARTMENT}(d) \wedge t \in d.MGR.l \wedge t \in d.DEPT.l\ \wedge \\
&\qquad\qquad e_2.NAME = Tom \wedge e_2.DEPT(t) = d.DEPT(t)\ \wedge \\
&\qquad\qquad d.MGR(t) = e_1.NAME)
\end{aligned}$$

□

2.2 An Inhomogeneous Grouped Algebra A_G

Having described the calculus L_{hi} and its data model, we are ready to present the corresponding algebra.

The grouped relational algebra, A_G, has five standard relational operators union, difference, Cartesian product, selection, and projection extended to the temporal domain. In addition, A_G contains two operators *tdom* and *vdom* that compute, respectively, the active temporal and value domains of a relation, and a timeslice operator. The operators of union, difference, Cartesian product, and projection are defined similarly to the standard relational case, and we use L_{hi} expressions to define them. Let R and Q be two L_{hi} relations.

1. *Union.* If R and Q are union-compatible, then $R \cup Q \ = \ \{e \mid R(e) \vee Q(e)\}$.

2. *Difference.* If R and Q are union-compatible, then $R - Q \ = \ \{e \mid R(e) \wedge \neg Q(e)\}$.

3. *Cartesian product.* $R \times Q \ = \ \{(e, e') \mid R(e) \wedge Q(e')\}$.

4. *Projection.* $\pi_{A_1,\ldots,A_k}(R) \ = \ \{e_1, \ldots, e_k \mid (\exists e)(R(e) \wedge \bigwedge_{i=1}^{k} e.A_i = e_i)\}$[1].

However, definition of selection is more involved in A_G than in the standard relational case.

5. *Selection.* Let $R(A_1, \ldots, A_n)$ be an L_{hi} relation. Then the syntax of selection is $\sigma_{F_1 \wedge \ldots \wedge F_m}(R)$ for $m \geq 1$, where each F_i is defined in one of the following ways:

a. $A_i =_f A_j$, where attributes A_i and A_j are of the sort $\mathcal{F}$, and $=_f$ is the equality operator for functions.

[1]If A_i is of sort $\mathcal{F}$, then the expression $e.A_i = e_i$ is not an L_h expression; rather, it is a macro stating that $e.A_i$ and e_i are equal as functions over time. More precisely, this means that the lifespans of $e.A_i$ and e_i are equal and for any time t in this lifespan $e.A_i(t) = e_i(t)$.

b. $A_i \theta_\tau A_j$ or $A_i \theta_\tau c$, where attributes A_i and A_j are of the sort $\mathcal{T}$, c is a constant of the sort $\mathcal{T}$, and θ_τ is a comparison operator $(=, >, \geq, <, \leq, \neq)$ for the temporal sort.

c. $\alpha_1 \theta_\nu \alpha_2$, where α_i $(i = 1, 2)$ is either a constant of the sort $\mathcal{V}$, or an attribute of that sort, or an expression of the form $A(T)$, where A is an $\mathcal{F}$ attribute and T is a $\mathcal{T}$ attribute of R. Also, θ_ν is a comparison operator $(=, >, \geq, <, \leq, \neq)$ for the sort $\mathcal{V}$.

d. $A_i \in_\tau A_j.\tau$, where attribute A_i is of the sort $\mathcal{T}$, A_j is of the sort $\mathcal{F}$, and $\in_\tau$ is the membership operator for the temporal sort. In this expression, $A_j.\tau$ denotes the *domain* (lifespan) of the temporal attribute A_j.

e. $A_i \in_\nu A_j.\nu$, where attribute A_i is of the sort $\mathcal{V}$, A_j is of the sort $\mathcal{F}$, and $\in_\nu$ is the membership operator for the value sort. In this expression, $A_j.\nu$ denotes the *range* of the temporal attribute A_j.

If no confusion arises, we will drop the subscripts τ and ν in the comparison θ_τ, θ_ν and in the membership $\in_\tau$ and $\in_\nu$ operators and assume that their exact meanings can be judged from the context.

The meaning of $\sigma_F(R)$ is the set of tuples in R for which, when we substitute their values into the formula F replacing attributes appearing in F with the values of the tuples, F becomes true.

Moreover, the algebra A_G contains two additional "twin" operators *tdom* and *vdom* that compute, respectively, the active temporal and value domains of a relation.

6. *Active-domain* operators. Assume that $R(A_1, \ldots, A_n)$ has only attributes of sort $\mathcal{F}$. Then

$$tdom(R) \;=\; \{t \mid (\exists e)(R(e) \wedge \textstyle\bigvee_{i=1}^{n} t \in e.A_i.l)\}$$

If R also contains attributes of sort $\mathcal{T}$, then their values should also be included in $tdom(R)$. $vdom(R)$ is defined similarly to $tdom(R)$: if $R(A_1, \ldots, A_n)$ has only attributes of sort $\mathcal{F}$, then

$$vdom(R) \;=\; \{d \mid (\exists e)(\exists t)(R(e) \wedge \textstyle\bigvee_{i=1}^{n}(t \in e.A_i.l \wedge e.A_i(t) = d))\}$$

If R also contains attributes of sort $\mathcal{V}$, then their values should also be included in $vdom(R)$.

Note that operators $tdom(R)$ and $vdom(R)$ define mappings from relation R into a single attribute relation on the sorts $\mathcal{T}$ and $\mathcal{V}$ respectively.

Finally, the algebra A_G contains an additional *timeslice* operator that restricts the lifespans of attributes of sort $\mathcal{F}$. More specifically,

7. *Timeslice* operator. Let $R(A_1, \ldots, A_i, \ldots, T, \ldots, A_n)$ be an L_{hi} relation, where attribute A_i is of sort $\mathcal{F}$ and attribute T is of sort $\mathcal{T}$. Then

$$\tau_{A_i:T}(R) \;=\; \{e.A_1, \ldots, e.A_{i-1}, e.A_i : t, e.A_{i+1}, \ldots, e.A_n \mid R(e) \wedge t \in e.A_i.l \wedge (\exists e')(R(e') \wedge e'.A_i = e.A_i \wedge t = e'.T)\}$$

This expression says that the timeslice operator $\tau_{A_i:T}(R)$ leaves all $\mathcal{F}$ attributes, except A_i, intact, groups together all the times T corresponding to the same instance of attribute A_i, restricts the lifespan of A_i to these grouped time instances, and projects the temporal attribute T out.

2.3 Equivalence of L_{hi} and A_G

When converting L_{hi} expressions into A_G, we map $R(e) \wedge t \in e.A.l$ into the A_G expression $\sigma_{S \in R.A.\mathcal{T}}(R \times S)$, where $S = tdom(\pi_A(R))$. However, the conversion becomes ambiguous for the L_{hi} expression $R(e) \wedge Q(e) \wedge t \in e.A.l$ because it is not clear if $e.A$ should be mapped into $\pi_A(R)$ or $\pi_A(Q)$. To solve this problem and to make the mapping from L_{hi} to A_G expressions easier, we normalize L_{hi} formulas by replacing L_{hi} expressions of the form $R(e) \wedge Q(e) \wedge t \in e.A.l$ with $(\exists e')(R(e) \wedge Q(e') \wedge e = e' \wedge t \in e.A.l)$. Similarly, we replace expressions of the form $R(e) \wedge Q(e') \wedge t \in e.A.l \wedge t \in e'.B.l$ with $(\exists t')(R(e) \wedge Q(e') \wedge t \in e.A.l \wedge t' \in e'.B.l \wedge t = t')$. This discussion motivates the following definition.

A safe L_{hi} expression ϕ is *normalized* if in every maximal conjunctive subformula of ϕ

- a tuple variable can appear in one and only one relation;
- every temporal variable belongs to one and only one lifespan.

Lemma 1 *Every safe L_{hi} formula can be converted to an equivalent safe normalized L_{hi} formula.*

Sketch of Proof: The proof is based on the observation that if a maximal conjunct of an L_{hi} formula is of the form $R(e) \wedge Q(e) \wedge \ldots$, then it is replaced with the expression $(\exists e')(R(e) \wedge Q(e') \wedge e = e' \wedge \ldots)$. Similarly, the L_{hi} expression $R(e) \wedge Q(e') \wedge t \in e.A.l \wedge t \in e'.B.l \wedge \ldots$ in a maximal conjunct is replaced with the expression $(\exists t')(R(e) \wedge Q(e') \wedge t \in e.A.l \wedge t' \in e'.B.l \wedge t = t' \wedge \ldots)$. $\Box$

Using this lemma, we can prove the main theorem of this section.

Theorem 2 *Safe L_{hi} calculus and the grouped algebra A_G are equivalent*

Sketch of Proof: Since we defined the A_G operators using L_{hi} calculus, then it is clear that any A_G expression can be converted into an equivalent L_{hi} expression. To show that any safe L_{hi} query can be mapped into an equivalent A_G expression, first normalize the L_{hi} query. The crucial part in the proof of this theorem is to show how to map a maximal conjunctive subformula of an L_{hi} expression into A_G. To do this, consider a maximal conjunctive subformula in this query. Consider all the terms of the form $R_i(e_i)$ and of the form $t_j \in e_i.A_i.l$ in it. For each term $t_j \in e_i.A_i.l$ create a single-attribute relation $T_{R_{A_i}j} = tdom(\pi_{A_i.\mathcal{T}}(R_i))$ (note that this is possible because the L_{hi} expression is normalized). Then take the Cartesian product of all the R_i's and $T_{R_{A_i}j}$'s and impose the following restrictions in the select operator on it. Each term $t_j \in e_i.A_i.l$ gives rise to the condition $T_{R_{A_i}j} \in R_i.A_i$. Each term $e_i.A_i = e_j.A_j$ gives rise to the condition $R_i.A_i =_f R_j.A_j$ in the selection. Each term $t_j = t'_j$, where $t_j \in e_i.A_i.l$ and $t'_j \in e'_i.A'_i.l$, gives rise to the condition $T_{R_{A_i}j} = T_{R'_{A'_i}j'}$. Finally, each term $e.A(t) = e'.A'(t')$ gives rise to the selection $R.A(T) = R'.A'(T')$, where T and T' are the single attribute relations of sort $\mathcal{T}$ corresponding to the terms $t \in e.A.l$ and $t' \in e'.A'.l$, and e and e' correspond to relations R and R' respectively. Finally, to convert the resulting relation to the form of the data model, every attribute A_i with an associated lifespan attribute T_{A_i} is timesliced to the set of times equal to the projection on attribute T_{A_i}. $\Box$

To illustrate how the conversion between L_{hi} and A_G works, we provide some examples. Since we expressed algebraic operators in terms of L_{hi} formulas above, we concentrate on mapping L_{hi} expressions to A_G. In the following examples, let $R(A)$ and $R(B)$ be two single attribute relations, where A and B have the sort $\mathcal{F}$.

Example 4 *The L_{hi} query*

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

has an equivalent A_G expression

$$\tau_{R.A:T_{R_A}}(\pi_{R.A,T_{R_A}}(\sigma_{T_{R_A} \in R.A.\tau \wedge T_{Q_B} \in Q.B.\tau \wedge R.A(T_{R_A})=Q.B(T_{Q_B})}(R \times Q \times T_{R_A} \times T_{Q_B})))$$

where $T_{R_A} = tdom(\pi_{A.\tau}(R))$ and $T_{Q_B} = tdom(\pi_{B.\tau}(Q))$ are temporal domains of A and B attributes in relations R and Q respectively.

□

Example 5 *Consider the L_{hi} query*

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

Before mapping it into A_G, we convert it into an equivalent L_{hi} query

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

which is equivalent to

$$\tau_{R.A:T_{R_A}}(\pi_{R.A,T_{R_A}}(\sigma_{T_{R_A} \in R.A.\tau}(R \times T_{R_A}) - \\ \pi_{R,T_{R_A}}(\sigma_{T_{R_A} \in R.A.\tau \wedge T_{Q_B} \in Q.B.\tau \wedge T_{R_A}=T_{Q_B}}(R \times Q \times T_{R_A} \times T_{Q_B}))))$$

where T_{R_A} and T_{Q_B} are defined as in Example 4.

□

Example 6 *The L_{hi} query*

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

has an equivalent A_G expression

$$\tau_{R.A:T_{R_A}}(\pi_{R.A,T_{R_A}}(\sigma_{T_{R_A} \in R.A.\tau \wedge T_{Q_B} \in Q.B.\tau \wedge R.A =_f Q.B}(R \times Q \times T_{R_A} \times T_{Q_B})))$$

where T_{R_A} and T_{Q_B} are defined as in Example 4.

□

3 SQL Extension

Of the many proposals offered to extend the relational model for handling temporal information, the recent appearance of the TSQL2 language specification [SAA$^+$94] is of considerable note. This specification is the result of a major collaborative effort whose goal to present a *standardized* temporal extension to the standard relational data model. Specifically, TSQL2 is presented as a temporally extended SQL-92, the current relational standard.

In the TSQL2 specification several goals were enunciated by the design committee as design guidelines. These guidelines addressed issues relating to the data model: support of a valid-time dimension, and based on homogeneous tuples; to the language: consistent extension of SQL-92, optional temporal support, and operators that do no give special semantics to explicit attributes; and to the implementation: implementable in some first normal form representational model, and have an efficiently implementable algebra that is an extension of the snapshot algebra. The thrust (both explicit and implicit) of these and many of the other features enunciated in the specification is that TSQL2 be a *true* or consistent extension of SQL-92 that defaults to SQL-92 when no temporal semantics are intended.

One consequence of this *upward compatibility* is that legacy applications need not be modified to accommodate a TSQL2-based database management system. This compatibility is achieved in the resulting TSQL2 proposal largely through the appending of additional clauses and sub-clauses to the SQL-92 syntax and by defining default semantics in the absence that is consistent with the SQL-92 clause that results by removing the TSQL2-specific clauses.

Although most of the features of TSQL2 are intended to directly support the formal incorporation of a temporal component to provide capabilities generally associated with database management systems, there is one feature of particular interest. The TSQL2 proposal includes provisions for the specification of a distinguished *surrogate* domain. This domain, and its associated attribute, provides a mechanism by which a user (application) can coindentify (i.e., recognize as forming a cohesive "group") a set of tuples, these having the same surrogate value. Used in this way, the surrogate is used as a grouping mechanism.

TSQL2 provides little in the way of support of a surrogate. It provides a way of associating a surrogate-based attribute with a relation (schema) and a mechanism for comparing for equality two surrogate values. Users are responsible for most other aspects pertaining to their management and administration.

As shown in [CCT94] such a surrogate is necessary if an ungrouped (i.e., 1NF) historical model is to have the expressiveness of a grouped (i.e., N1NF) historical model. However, as is also shown, the effective management of surrogates is considerably complex. This complexity can be embedded in the semantics of the data language, the approach taken in [CCT94], or, alternatively, left as a user task, the effective approach of TSQL2.

The appeal of the grouped approach to representing historical data is due in part to its relationship to the goals, or at least the results of normalization in the standard relational model; that is, the ability to model a single instance of an object (entity or relationship) by a single tuple in a relation. Achieving this correspondence in a historical database where multiple values must be maintained for the attributes of an object requires that a N1NF model be used. Such an approach maintains the correspondence between a semantic "object" and a tuple, although it might increase the complexity required to implement such a model relative to that required to implement a 1NF model.

A grouped historical relational extension represents one example of the desirability of a relationally-oriented model that supports complex objects. Recognition of this fact is evidenced by efforts currently underway to develop a new standard, SQL-3, that will provide a complex object modeling capability. The result of this effort may provide a modeling capability that is easily amenable to being extended to incorporate a historical dimension. More specifically, it may provide a sound basis upon which to build N1NF relations corresponding to the normative group model associated with the group algebra that we describe in the next section.

Any SQL extension that is intended to query, or otherwise manipulate, a grouped historical relational database must be able to accommodate the basic distinction between grouped and ungrouped data models. In an ungrouped model the temporal dimension is incorporated, explicitly or implicitly, through the addition of one or more *distinguished* temporal attributes. For each tuple, the value of these attributes indicates the period of *validity* of the data in the other attributes of the tuple. An SQL extension intended to accommodate such an ungrouped historical model need only provide facilities for dealing with temporal attributes; the semantics of the tuple variables could remain largely unchanged from those of the standard SQL.

In a grouped historical model a tuple represents the history of an entity (or relationship). More specifically, an attribute of a tuple is a history of the values assumed by that attribute. A history can thus be viewed as a pairing of traditional domain values and the times that those values are valid. This aspect of a grouped historical model is what has to be accommodated by a grouped-based historical SQL extension. The following SQL-based SELECT statement provides such a capability:

```
SELECT attribute-target-list
FROM tuple/temporal variable declaration list
WHERE restriction predicate
```

The SELECT construct supports both the traditional tuple variable, and history-oriented temporal variables. As per the standard SQL, tuple variables are bound to relations, and range over the tuples of the relation to which they are bound. Temporal variables are bound to tuple variables, and range over the histories of tuples of the associated relation.

The FROM clause is used to declare both types of variables as well as to perform the necessary bindings. It has the form:

`FROM` $rel_1\ t_1 : T_1,\ rel_2\ t_2 : T_2,\ \ldots$

which declares tuple variable t_i and binds it to relation rel_n, and declares temporal variable T_i and binds it to tuple variable t_i. (Although not indicated, any number of tuple variables may be declared and bound to a single relation. Likewise any number of temporal variables can be declared and bound to a single tuple variable.)

The SELECT clause has the form:

```
SELECT tup-att-exp₁, tup-att-exp₂, ..., tup-att-expₙ
```

and as in the case of the standard SQL is used to specify the attribute values that are to be included in the resulting relation. `tup-att`$_i$ is of the form $t_i.A_i$ if the domain of A_i is of sort $\mathcal{V}$or $\mathcal{T}$, and $t_i.A_i : T_i$ if its domain is of sort $\mathcal{F}$.

The WHERE clause plays the same role here as it does in the standard SQL; it specifies a predicate that is used to select those values, (tuple values and temporal values) over which the tuple and temporal variables range, that together satisfy the predicate.

Example 7 *Given the grouped historical relation EMPLOYEE the query "Find the name and salary (histories) of all employees who worked in the Toy department at time 4" is expressed in* SQL_{hi} *as:*

```
SELECT E1.NAME, E1.Salary:T1
FROM EMPLOYEE E1:T1,T2
WHERE E1.DEPT(T2) = "Toy" AND T2 = 4
```

□

In this example one tuple variable E1 was declared and bound to the relation EMPLOYEE, and two temporal variables, T1 and T2, were declared and bound to this tuple variable. The first, T1, since not otherwise constrained, was used to extract the complete histories of the SALARY attribute of EMP, and the second, T2, was used to reference the DEPT history value at the specified time.

Example 8 *The query "Find the names, dates of birth, salary and departmental histories of the people who worked at time 5 in the Accounting department" can be expressed in* SQL_{hi} *as:*

```
SELECT E1.NAME, E1.DOB, E1.SALARY:T1, E1.DEPT:T2
FROM EMPLOYEE E1:T1,T2
WHERE E1.DEPT(5) = ‘‘Accntg’’
```

□

Finally,

Example 9 *"What are the names of the managers for whom Tom has worked?" can be expressed in in* SQL_{hi} *as:*

```
SELECT E1.NAME
FROM EMPLOYEE E1, E2:T1, DEPT D:T1
WHERE E2.NAME = ‘‘Tom’’ AND E2.DEPT(T1) = D.DEPT(T1) AND D.MGR(T1) = E1.NAME
```

□

In the absence of temporal components, the grouped SQL SELECT statement described above degrades to a standard SQL-2 SELECT statement, and has the same semantics.

4 Some Semantic Issues Revisited

In this section we look at a number of semantic issues which are familiar and well-understood in the case of the static relational model, but which are more complex and appear to have generated some confusion in the temporal database literature when they have been extended to the case of temporal relations. In particular, we believe that it is worth revisiting the concepts of *relation key*, *functional dependency*, *normalization*, and *coalescing*, in the context of the temporally ungrouped/grouped modeling approaches. In addition, we look at an operation called *regrouping*, related to the concept of normalization, and discuss its utility.

Functional Dependencies

The notion of functional dependency in a snapshot or static relation is well understood.

Let R be a (static) relation scheme on a set of attributes $A = \{A_1, A_2, \ldots, A_n\}$, and let $X, Y \subseteq A$. Then X functionally determines Y, $X \rightarrow Y$, iff for all tuples t_1 and t_2 in a relation r on R, $t_1(X) = t_2(X) \Rightarrow t_1(Y) = t_2(Y)$.

In the Temporal Database Glossary [JCE+94], the notion of a functional dependency is defined for the case of temporal relations. The definition relies on the notion of a "snapshot" of a temporal relation. While this notion is never precisely defined, its meaning should be obvious — the snapshot of a temporal relation at time t is the snapshot (i.e., non-temporal) relation with all and only those tuples which are valid at time t.

Informally, two tuples are *snapshot equivalent* if the snapshots of the tuples at all times are identical. Similarly, two relations are snapshot equivalent if at every instant their snapshots are equal.

A notion of functional dependency with respect to temporal databases, called a *temporal functional dependency*, is defined in the glossary as follows:

> Let X and Y be sets of explicit attributes of a temporal relation schema, R.
> A *temporal functional dependency*, denoted $X \overset{\mathrm{T}}{\rightarrow} Y$, exists on R if, for all instances r of R, all snapshots of r satisfy the functional dependency $X \rightarrow Y$.

Let us explore the notion of "functional dependency" in the context of the distinction between the temporally grouped and ungrouped data modeling paradigms. In the discussion that follows, we will consider three different generic temporal data models: M_g will refer to a temporally grouped model, M_{ug} will refer to a temporally ungrouped model, and M_s will refer to a temporally ungrouped model with surrogates.

There are three different natural extensions to the static notion of an FD to the temporal case:

Let R be a temporally grouped relation scheme on a set of attributes $A = \{A_1, A_2, \ldots, A_n\}$, and let $X, Y \subseteq A$. (Recall that in M_g, the value of an attribute A in a tuple t is a *partial function* from the set of times into some set of values.). Then:

- $X \rightarrow_1 Y$ iff

$$\forall t_1, t_2 \in r[t_1(X) = t_2(X) \Rightarrow t_1(Y) = t_2(Y)]$$

Intuitively, this constraint says that if two tuples have the same *function* value for their X-attribute, then they must have the same *function* value for their Y-attribute.

- $X \rightarrow_2 Y$ iff

$$\forall t_1, t_2 \in r, \forall \tau \in T[t_1(X)(\tau) = t_2(X)(\tau) \Rightarrow t_1(Y)(\tau) = t_2(Y)(\tau)]$$

Intuitively, this constraint says that if two tuples agree at some time τ on their X-attribute, then they must agree at the same time τ on their Y-attribute.

- $X \rightarrow_3 Y$ iff

$$\forall t_1, t_2 \in r, \forall \tau_1, \tau_2 \in T[t_1(X)(\tau_1) = t_2(X)(\tau_2) \Rightarrow t_1(Y)(\tau_1) = t_2(Y)(\tau_2)]$$

Intuitively, this constraint says that if two tuples agree at any times τ_1 and τ_2 on their X-attribute, then they must agree at the same times τ_1 and τ_2 on their Y-attribute.

The following semantic relationships among these three types of dependencies follow immediately from their definitions:

$$X \rightarrow_3 Y \models X \rightarrow_2 Y \models X \rightarrow_1 Y$$

Moreover, these consequences are proper, as the following examples demonstrate. It is also clear that when the universe of times T consists of a single point in time, i.e., in the static case, all three of these notions are equivalent and reduce to the standard relational definition of FD without any change to their definition.

The following relation, for example, satisfies the temporal FD $A \rightarrow_2 B$, but not $A \rightarrow_3 B$:

$R1_g$	
A	B
$\begin{bmatrix} 1 & \rightarrow & a \\ 2 & \rightarrow & a \end{bmatrix}$	$\begin{bmatrix} 1 & \rightarrow & c \\ 2 & \rightarrow & d \end{bmatrix}$
$\begin{bmatrix} 1 & \rightarrow & c \\ 2 & \rightarrow & a \end{bmatrix}$	$\begin{bmatrix} 1 & \rightarrow & x \\ 2 & \rightarrow & d \end{bmatrix}$

and the following relation satisfies the temporal FD $A \rightarrow_1 B$, but not $A \rightarrow_2 B$ (and thus not $A \rightarrow_3 B$):

$R2_g$		
A	B	C
$\begin{bmatrix} 1 & \rightarrow & a \\ 2 & \rightarrow & b \end{bmatrix}$	$\begin{bmatrix} 1 & \rightarrow & x \\ 2 & \rightarrow & y \end{bmatrix}$	$\begin{bmatrix} 1 & \rightarrow & c_1 \\ 2 & \rightarrow & c_2 \end{bmatrix}$
$\begin{bmatrix} 1 & \rightarrow & a \\ 2 & \rightarrow & b \end{bmatrix}$	$\begin{bmatrix} 1 & \rightarrow & x \\ 2 & \rightarrow & y \end{bmatrix}$	$\begin{bmatrix} 1 & \rightarrow & c_3 \\ 2 & \rightarrow & c_4 \end{bmatrix}$
$\begin{bmatrix} 1 & \rightarrow & a \\ 2 & \rightarrow & c \end{bmatrix}$	$\begin{bmatrix} 1 & \rightarrow & z \\ 2 & \rightarrow & y \end{bmatrix}$	$\begin{bmatrix} 1 & \rightarrow & c_5 \\ 2 & \rightarrow & c_6 \end{bmatrix}$

Thus, there are at least three possible extensions to the static notion of FD which are all different, and which are expressible as constraints on valid relations in the model M_g.

In M_{ug}, it is clear that both $\rightarrow_2$, and $\rightarrow_3$ are expressible; in fact, it should be clear that $\rightarrow_2$ corresponds to the Glossary's notion of *temporal functional dependency*, above. However, there is no analog to $\rightarrow_1$, since there is no notion of a "group" to quantify over.

The definitions of these dependencies in M_{ug}, however, is not a straightforward extension of the definitions in the static case. The definitions involve a kind of hidden quantification over time; to be more precise, this quantification ought to be "unhidden", and then the differences between these two approaches become more apparent. For M_{ug}, these three notions of FD are:

- $X \rightarrow_1 Y$

  (There is no analog to this.)

- $\{X, Time\} \rightarrow_2 Y$ iff

$$\forall t_1, t_2 \in r[t_1(X) = t_2(X) \wedge t_1(Time) = t_2(Time) \Rightarrow t_1(Y) = t_2(Y)]$$

  This is the notion of temporal functional dependency in the Glossary.

  Intuitively, this constraint says that if two tuples with the same timestamp τ agree on their X-attribute, then they must agree on their Y-attribute.

- $X \rightarrow_3 Y$ iff

$$\forall t_1, t_2 \in r[t_1(X) = t_2(X) \Rightarrow t_1(Y) = t_2(Y)]$$

  Intuitively, this constraint says that if two tuples, regardless of their timestamp, agree on their X-attribute, then they must agree on their Y-attribute.

Finally, in M_s, these three notions of FD are:

- $X \rightarrow_1 Y$
  (There is no analog to this.)
- $\{X, Time\} \rightarrow_2 Y$ iff

$$\forall t_1, t_2 \in r[t_1(X) = t_2(X) \wedge t_1(Time) = t_2(Time) \Rightarrow t_1(Y) = t_2(Y)]$$

  This is the same notion as in M_{ug}; the Surrogate is not involved.
- $X \rightarrow_2 Y$ iff

$$\forall t_1, t_2 \in r[t_1(X) = t_2(X) \Rightarrow t_1(Y) = t_2(Y)]$$

  Again, this is the same notion as in M_{ug}; the Surrogate is not involved.

This information is summarized in the following table:

Constraint	M_g	M_{ug}	M_s	Glossary
$X \rightarrow_1 Y$	Y	N	N	N
$X \rightarrow_2 Y$	Y	Y	Y	$X \overset{\mathrm{T}}{\rightarrow} Y$
$X \rightarrow_3 Y$	Y	Y	Y	N

Keys

The notion of a key in the standard relational model is definable in terms of the more primitive notion of an FD:

Let R be a (static) relation scheme on a set of attributes $A = \{A_1, A_2, \ldots, A_n\}$, and let $X, Y \subseteq A$. Then a set of attributes K, $K \subseteq A$ is a *key* for R if $K \rightarrow A$, and K is minimal.

How does this notion extend to the case of temporal relations in both the temporally grouped and temporally ungrouped paradigms?

In M_g, corresponding to the three kinds of FD, we can derive three notions of key, key_1, key_2, and key_3. The definition of each of these 3 notions is exactly analogous to the standard notion, with the substitution of one of the 3 types of FD. In HRDM ([CC87]) e.g., the notion of key was key_2, which seems to be a natural choice.

In M_{ug}, the notion of key is slightly more complicated, and is not a straightforward extension of the notion of a static key. Moreover, there are only two possible notions of key, key_2, and key_3. For example, consider the following relation in M_g, with attribute A being a key_2:

R_g	
A	B
$\begin{bmatrix} 1 & \to & a \\ 2 & \to & a \end{bmatrix}$	$\begin{bmatrix} 1 & \to & b_1 \\ 2 & \to & b_2 \end{bmatrix}$
$\begin{bmatrix} 2 & \to & c \\ 3 & \to & d \end{bmatrix}$	$\begin{bmatrix} 2 & \to & b_3 \\ 3 & \to & b_2 \end{bmatrix}$

The closest approximation to this relation in$M_{ug}$is the following relation:

$R3_{ug}$		
A	B	*Time*
a	b_1	1
a	b_2	2
c	b_3	2
d	b_2	3

In this relation, A is no longer a key; the key is the set of attributes $\{A, Time\}$. Clearly the notion of a temporal functional dependency is strictly stronger that that of a functional dependency; in effect it involves a "hidden" universal quantifier over all times. This stronger notion, temporal functional dependency, is what motivates the Glossary's notion of the *key* of a temporal relation:

> A set of attribute $K \subseteq A$ is said to be a key for R iff for every valid relation r on scheme R, $K \rightarrow R$, i.e., K *temporally functionally determines* every attribute in R.

Note that this definition tends to obscure the role of the Time attribute, which in fact is a crucial part of the key. Our definition, above, makes this explicit. In fact, in models using two additional timestamping attributes, e.g. Start-Time and End-Time, the explicit definition is not even straightforward.

A final notion that bears comment is that of a constant-valued key. This seems like a perfectly valid constraint that an application may wish to impose. In M_g it can be expressed as follows, for an arbitrary temporally grouped relation scheme R with key K:

$$\forall t \in r, \forall \tau_1, \tau_2 \in T[t(K)(\tau_1) = t(K)(\tau_2)]$$

Thus in M_g it is possible to have some relations where this constraint is imposed on the key, and others where it is not. This constraint is not expressible in M_{ug}, precisely because M_{ug} in the way it implicitly assumes that tuples with the same key values are related, implicitly enforces it. In M_s it can be expressed as follows:

$$\forall t_1, t_2 \in r[t_1(Surrogate) = t_2(Surrogate) \Rightarrow t_1(K) = t_2(K)]$$

Normal Forms

Here is what the glossary ([JCE+94]) says about functional dependencies and normal forms:

> A pair (R, F) of a temporal relation schema R and a set of associated temporal functional dependencies F is in *temporal Boyce-Codd normal form* (TBCNF) if
>
> $$\forall\, X \xrightarrow{\mathrm{T}} Y \;\in F^+ \;(Y \subseteq X \vee X \xrightarrow{\mathrm{T}} R)$$
>
> where F^+ denotes the closure of F and X and Y are sets of attributes of R.
>
> Similarly, (R, F) is in *temporal third normal form* (T3NF) if for all non-trivial temporal functional dependencies $X \xrightarrow{\mathrm{T}} Y$ in F^+, X is a temporal superkey for R or each attribute of Y is part of a minimal temporal key of R.

Because three different notions of FD can be supported in M_g, there are three possible versions of those normal forms which are based on the primitive notion of FD. These need to be further explored.

Moreover, there is the concept of "time normalization" in [NA93]. An ungrouped relation on scheme (ENO SAL MANAGER T_s T_E) is used as the example. Because the non-key attributes do not change "in synch", the recommendation of their "time normalization" design methodology is to split the relation scheme into two schemes:

(ENO SAL T_s T_E) and (ENO MANAGER T_s T_E)

While not uncommon, it is extremely *rare* for the values of attributes to change in a temporally synchronized manner. Therefore this design methodology would tend to spilt every relation scheme $(K\ A_1\ A_2\ \ldots\ A_n)$ into n separate relation scheme $(K\ A_1)$, $(K\ A_2)$, $\ldots (K\ A_n)$.

We believe that this is not best viewed as a normalization issue. Clearly, it is an artifact of the model itself, which associates temporal information with entire tuples. The " design methodology" discovers that this is in fact inappropriate, and then recommends that each individual tuple be split into n separate tuples in n relations. Far better, we believe, would be a model which associated the temporal dimension with each attribute, but kept intact the notion of a normalized tuple as representing all of the information about some semantic object. Thus, in a grouped approach, the temporal relation on scheme $(K\ A_1\ A_2\ \ldots\ A_n)$ is normalized if it satisfies the standard definitions based on functional dependencies. No additional notion of "temporal normalization" is needed.

Coalescing

Another notion that appears in a number of temporal data models is that of the "coalescing" of tuples. This concept is defined in the glossary as follows:

> The *coalesce* operation takes as argument a set of value-equivalent tuples and returns a single tuple which is snapshot equivalent with the argument set of tuples.

This definition relies on the auxiliary notion of value-equivalence:

> Informally, two tuples on the same (temporal) relation schema are *value equivalent* if they have identical non-timestamp attribute values.

This concept appears to have been introduced in [Snod87] and it has reappeared in a number of other papers, including [NA93] (where it is called "compress") and [JSS94].

The concept of coalescing is an artifact of 1NF models, since it's function is to try to "merge" (or "coalesce") into a smaller number of tuples information about "the same object" which is spread across a larger number of tuples. Furthermore, it appears to be relevant only to ungrouped models using two timestamps, such as *StartTime* and *EndTime* as its timestamping mechanism. For example, the first two tuples in relation on the left, below, ought to be "coalesced", yielding the relation on the right.

A	B	*StartTime*	*EndTime*
a	b_1	1	2
a	b_1	3	5
c	b_2	3	6

A	B	*StartTime*	*EndTime*
a	b_1	1	5
c	b_2	3	6

Coalescing is an unnecessary (indeed, meaningless) operation in temporally grouped models. In M_g the information in the above relations would be represented as follows:

A	B
$1 \to$ a $2 \to$ a $3 \to$ a $4 \to$ a $5 \to$ a	$1 \to b_1$ $2 \to b_1$ $3 \to b_1$ $4 \to b_1$ $5 \to b_1$
$3 \to$ c $4 \to$ c $5 \to$ c $6 \to$ c	$3 \to b_2$ $4 \to b_2$ $5 \to b_2$ $6 \to b_2$

Furthermore, it is not clear under what circumstances the *coalesce* operator ought to be performed. Consider the following relation

EMPLOYEE	*SALARY*	*StartTime*	*EndTime*
a	$30,000$	1	2
a	$35,000$	3	5
c	$35,000$	6	7

and consider a query that projected this relation onto the *SALARY* column. Two different results are obtained depending on whether tuples are not coalesced (left) or are coalesced(right):

SALARY	*StartTime*	*EndTime*
$30,000$	1	2
$35,000$	3	5
$35,000$	6	7

SALARY	*StartTime*	*EndTime*
$30,000$	1	2
$35,000$	3	7

In a grouped model, the information would be as shown on the left, and the result of the query as shown on the right. The issue of "coalescing" never arises.

EMPLOYEE	*SALARY*
$[1,2] \to$ a $[3,5] \to$ a	$[1,2] \to 30,000$ $[3,5] \to 35,000$
$[6,7] \to$ c	$[6,7] \to 35,000$

SALARY
$[1,2] \to 30,000$ $[3,5] \to 35,000$
$[6,7] \to 35,000$

Regrouping

A final semantic issue which needs to be clarified is that of the "regrouping" of a temporally grouped relation. This appears to be a virtually meaningless operation, except in certain cases that arise infrequently in practice. Consider the following relation in M_g with the key **EMP** serving as the basis for grouping:

EMPLOYEE		
EMP	*DEPT*	*SALARY*
$\begin{bmatrix} 1 & \to & John \\ 2 & \to & John \\ 3 & \to & John \end{bmatrix}$	$\begin{bmatrix} 1 & \to & Toy \\ 2 & \to & Toy \\ 3 & \to & Clothing \end{bmatrix}$	$\begin{bmatrix} 1 & \to & 20K \\ 2 & \to & 25K \\ 3 & \to & 30K \end{bmatrix}$
$\begin{bmatrix} 1 & \to & Henry \\ 2 & \to & Henry \\ 3 & \to & Henry \end{bmatrix}$	$\begin{bmatrix} 1 & \to & Linen \\ 2 & \to & Linen \\ 3 & \to & Housewares \end{bmatrix}$	$\begin{bmatrix} 1 & \to & 20K \\ 2 & \to & 30K \\ 3 & \to & 35K \end{bmatrix}$

Now, the operation of regrouping is supposed to take a temporally grouped relation, which is grouped by some attribute(s), and regroup it by some different attribute(s). Suppose we try to apply such an operation to the *employees* relation, and try to regroup it by the *SALARY* attribute.

There are several problems with this operation, as this example makes clear. First, if there is to be a regrouping on the attribute *SALARY*, there needs to be some basis for *forming* the new salary groups. Absent any other information, it must be that the basis for the regrouping is the salary *values*, and thus this operation will automatically impose a *time-invariant key constraint* on the resulting relation, which may or may not be appropriate for the application. We assume that this is the intended meaning of the operation, and hence the result is the following:

EMPLOYEE		
SALARY	*EMP*	*DEPT*
$\left[1 \rightarrow 20K \right]$	$\left[1 \rightarrow \{John, Henry\} \right]$	$\left[1 \rightarrow \{Toy, Linen\} \right]$
$\left[2 \rightarrow 25K \right]$	$\left[2 \rightarrow John \right]$	$\left[2 \rightarrow Toy \right]$
$\left[\begin{array}{lll} 2 & \rightarrow & 30K \\ 3 & \rightarrow & 30K \end{array} \right]$	$\left[\begin{array}{lll} 2 & \rightarrow & Henry \\ 3 & \rightarrow & John \end{array} \right]$	$\left[\begin{array}{lll} 2 & \rightarrow & Linen \\ 3 & \rightarrow & Clothing \end{array} \right]$
$\left[3 \rightarrow 35K \right]$	$\left[3 \rightarrow Henry \right]$	$\left[3 \rightarrow Housewares \right]$

A second problem, as this example clearly shows, the resulting object may not be a valid relation. In this case, for example, there are two employees who had the salary $20K$ at time 1, and so the resulting object must have a set of values for the remaining (non-grouping) attributes.

Finally, it is not at all clear what these new tuples represent. It might seem at first that they would represent "temporal portions of actual employee's salary histories." On closer examination, this turns out not to be the case. Consider the third tuple in the result: no employee had any such salary history.

Thus the operation is fraught with problems.

The classic example of this operation is the following *management* relation. According to [GN93], " Its key is *DEPT*. This key is possible because it is the key of the snapshots of the management relation." Thus, the notion of key here appears to be that of snapshot key, and the authors correctly point out that there is a case where such a regrouping operation might be useful.

management	
DEPT	*MANAGER*
$\left[[11,49] \rightarrow Toys \right]$	$\left[\begin{array}{lll} [11,44] & \rightarrow & John \\ [45,49] & \rightarrow & Leu \end{array} \right]$
$\left[\begin{array}{lll} [41,47] & \rightarrow & Clothing \\ [71,\texttt{now}] & \rightarrow & Clothing \end{array} \right]$	$\left[\begin{array}{lll} [41,47] & \rightarrow & Tom \\ [71,\texttt{now}] & \rightarrow & Inga \end{array} \right]$
$\left[[45,60] \rightarrow Shoes \right]$	$\left[[45,60] \rightarrow John \right]$

In [GN93] this relation is restructured (the operation is not formally defined, but it is intuitively obvious and is the one we used, above) into the following "management-2" relation, with key **MANAGER**:

management-2	
MANAGER	*DEPT*
[[11,60] → *John*]	[[11,44] → *Toys* [45,60] → *Shoes*]
[[45,49] → *Leu*]	[[45,49] → *Toys*]
[[41,47] → *Tom* [71,`now`] → *Inga*]	[[41,47] → *Clothing* [41,47] → *Clothing*]

Notice first of all, as we pointed out above, that this restructuring operation, defined in terms of *equality of values*, incorporates the strong assumption that a Manager cannot change names and be considered the same Manager. It is also important to note, as [GN93] does, the underlying assumption at work here, "that *DEPT* and *MANAGER* functionally determine each other. In fact, without this assumption, no such restructuring is possible, as our SALARY example, above, demonstrated. For instance, consider the following relation, only slightly changed from this example:

management-3	
DEPT	*MANAGER*
[11, 49] → Toys	[11, 44] → John [45, 49] → Leu
[41, 47] → Clothing [71, `now`] → Clothing	[41, 47] → Tom [71, `now`] → Inga
[40, 60] → Shoes	[40, 60] → John

Here, $DEPT \rightarrow MANAGER$ but not the inverse. It is clearly not semantically meaningful to restructure this relation, since it is not *about managers*, it is *about departments.* The result of an attempted restructuring would be the following:

management-4	
MANAGER	*DEPT*
[11, 60] → John	[11, 39] → Toys [40, 44] → {Toys,Shoes} [45, 60] → Shoes
[45, 49] → Leu	[45, 49] → Toys
[41, 47] → Tom [71, `now`] → Inga	[41, 47] → Clothing [71, `now`] → Clothing

This is not a valid relation – note the set of values for the DEPT relation. Note that here it is not possible to do the regrouping, since $MANAGER \not\rightarrow DEPT$.

So what exactly is this *regrouping* operation, and how general is it?

Note that when normalized, relations typically have the property that they represent information either about some real-world entity, or about an association (relationship) between or among some number of entities. In either case, it is the key attribute(s) which identify the thing being modeled, and its lifespan represents the maximal period of time during which information is known about the object. The other attributes are descriptive attributes which give additional, non-identifying information about the objects; the temporal dimension of this information is some subset (possibly all) of the lifespan of the object being modeled. In some cases these descriptive attributes are foreign keys, i.e., “references” to objects which are modeled in some other relation.

Consider, for example, the EMPLOYEE relation, which models employees with grouped tuples. The notion of regrouping here can only apply to the non-key attributes, i.e., either to the attribute DEPT, which is a foreign key, or the attribute SALARY, which is not. In either case, there are problems. In the case of DEPT, where the operation might at first glance appear to make sense semantically, the operation, as we have seen, is only well-defined for the rare case when the 2 attributes are mutually FD ($DEPT \leftrightarrow EMP$), i.e., when the attribute to be regrouped on is itself a candidate key for the relation.

Thus, a regrouping operator for temporally grouped relations does not appear to be a useful one, and the lack of it is not a valid criticism of the temporally grouped approach to temporal relational databases. Should such an operation be deemed desirable, in the case of multiple candidate keys, its definition is straightforward and can be included in any temporally grouped relational data model.

5 Conclusions

In this paper we have addressed the two principal concerns that have been raised with respect to the temporally grouped, or history-oriented, approach to modeling temporal information in a temporal relational data model. Specifically, we presented an extension to the temporally grouped data model and query language proposed in [CCT94], a data model and query language which is also quite similar to that proposed in [GST93]. These consistent extensions, motivated by the desire to define an equivalent algebraic query language, in fact provide greater modeling capability by allowing for the representation of three different sorts of information: constant values, times, and value histories. In addition, these extensions relax the temporal homogeneity requirement for the attributes of a given tuple. Having extended the data model and calculus in this fashion, we presented an equivalent algebraic query language, A_G,thereby addressing one of the major concerns about this approach.

In addition, we addressed a number of other semantic issues in the context of the temporally grouped/tempo
ungrouped modeling distinction , and argued that in all cases the temporally grouped approach appears to have significant advantages.

Our conclusion is a strong recommendation that the effort to develop a temporal extension to the SQL-3 standard, dubbed TSQL3, be based upon the temporally grouped model if it is to better meet the modeling needs of temporal information.

References

[CC87] J. Clifford, A. Croker. The Historical Relational Data Model (HRDM) and Algebra Based on Lifespans. *Proc. 3rd International Conference on Data Engineering* (Los Angeles, CA), February, 1987.

[CCT94] J. Clifford, A. Croker, A. Tuzhilin. On Completeness of Historical Relational Query Languages. *ACM Transactions on Database Systems*, 19(2):64–116, March 1994.

[Gad86] S. Gadia. Weak Temporal Relations. *Proc. of ACM SIGACT-SIGMOD Symposium on Principles of Database Systems* (Cambridge, MA), March 1986.

[GN93] S.K. Gadia, S.S. Nair. Temporal Databases: A Prelude to Parametric Data. in *Temporal Databases: Theory, Design, and Implementation*, Benjamin/Cummings Publishing Co., Redwood City, CA, 1993.

[GST93] F. Grandi, M.R. Scalas, P. Tiberio, History and Tuple Variables for Temporal Query Languages, in [Snod93].

[JCE$^+$94] C.S. Jensen, J. Clifford, R. Elmasri, S.K. Gadia, P. Hayes, S. Jajodia (eds.), C. Dyreson, F. Grandi, W. Käfer, N. Kline, N. Lorentzos, Y. Mitsopoulos, A. Montanari, D. Nonen, E. Peressi, B. Pernici, J.F. Roddick, N.L. Sarda, M.R. Scalas, A. Segev, R.T. Snodgrass, M.D. Soo, A. Tansel. P. Tiberio, G. Widerhold, A Consensus Glossary of Temporal Database Concepts, *ACM SIGMOD Record* 23:1, 1994.

[JeSn94] C.S. Jensen, R.T. Snodgrass, The Surrogate Data Type in TSQL2, A TSQL2 Commentary, in [TSQL2a].

[JSS94] C.S. Jensen, M.D. Soo, R.T. Snodgrass, Unifying Temporal Data Models via a Conceptual Model. *Information Systems*, 19(7):513–548, October 1994.

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

[NA93] S.B.Navathe, R.Ahmed. Temporal Extensions to the Relational Model and SQL. in *Temporal Databases: Theory, Design, and Implementation*, Benjamin/Cummings Publishing Co., Redwood City, CA, 1993.

[PSE$^+$94] N. Pissinou, R.T. Snodgrass, R. Elmasri, I.S. Mumick, M.T. Özsu, B. Pernici, A. Segev, B. Theodoulidis and U. Dayal, Towards an Infrastructure for Temporal Databases: Report of an Invitational ARPA/NSF Workshop, *ACM SIGMOD Record* 23:1, 1994.

[Snod87] The Temporal Query Language TQuel. *ACM Transactions on Database Systems*, 12(2):247–298, July 1987.

[Snod93] R.T. Snodgrass (ed.), *Proc. of the ARPA/NSF International Workshop on an Infrastructure for Temporal Databases*, Arlington (TX), 1993.

[SAA$^+$94] R.T. Snodgrass, I. Ahn, G. Ariav, D.S. Batory, J. Clifford, C.E. Dyreson, R. Elmasri, F. Grandi, C.S. Jensen, W. Käfer, N. Kline, K. Kulkarni, T.Y.C. Leung, N. Lorentzos, J.F. Roddick, A. Segev, M.D. Soo, S.M. Sripada, TSQL2 Language Specification, *ACM SIGMOD Record* 23:1, 1994.

[SAA$^+$94] R.T. Snodgrass, I. Ahn, G. Ariav, D.S. Batory, J. Clifford, C.E. Dyreson, R. Elmasri, F. Grandi, C.S. Jensen, W. Käfer, N. Kline, K. Kulkarni, T.Y.C. Leung, N. Lorentzos, J.F. Roddick, A. Segev, M.D. Soo, S.M. Sripada, A TSQL2 Tutorial, *ACM SIGMOD Record* 23:3, 1994.

[SnJe94] R.T. Snodgrass, C.S. Jensen, The From Clause in TSQL2, A TSQL2 Commentary, in [TSQL2a].

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

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

[TSQL2a] *The TSQL2 Language Design Committee*, TSQL2 Language Specification, 1994 (available by anonymous ftp at site `cs.arizona.edu`).

[TSQL2b] *The TSQL2 Language Design Committee*, TSQL2 Language Specification, *ACM SIGMOD Record* 23:1, 1994.

[GST95] F. Grandi, M.R. Scalas, P. Tiberio, A History-oriented Temporal SQL Extension, to be presented at the *Second International Workshop on Next Generation Information Technologies and Systems* (Naharia, Israel), 1995.