Semantics of Time-Varying Attributes and Their Use for Temporal Database Design

Christian S. Jensen[1] Richard T. Snodgrass[2]

[1] Department of Mathematics and Computer Science, Aalborg University, Fredrik Bajers Vej 7E, DK–9220 Aalborg Ø, DENMARK, csj@iesd.auc.dk

[2] Department of Computer Science, University of Arizona, Tucson, AZ 85721, USA, rts@cs.arizona.edu

Abstract. Based on a systematic study of the semantics of temporal attributes of entities, this paper provides new guidelines for the design of temporal relational databases. The notions of observation and update patterns of an attribute capture when the attribute changes value and when the changes are recorded in the database. A lifespan describes when an attribute has a value. And derivation functions describe how the values of an attribute for all times within its lifespan are computed from stored values. The implications for temporal database design of the semantics that may be captured using these concepts are formulated as schema decomposition rules.

1 Introduction

Designing appropriate database schemas is crucial to the effective use of relational database technology, and an extensive theory has been developed that specifies what is a good database schema and how to go about designing such a schema. The relation structures provided by temporal data models, e.g., the recent TSQL2 model [13], provide built-in support for representing the temporal aspects of data. With such new relation structures, the existing theory for relational database design no longer applies. Thus, to make effective use of temporal database technology, a new theory for temporal database design must be developed.

We have previously extended and generalized conventional normalization concepts to temporal databases [7, 8]. But the resulting concepts are still limited in scope and do not fully account for the time-varying nature of data. Thus, additional concepts are needed in order to fully capture and exploit the time-varying nature of data during database design. This paper proposes concepts that capture the time-related semantics of attributes and uses these as a foundation for developing guidelines for the design of temporal databases. The properties of time-varying attributes are captured by describing their lifespans, their time patterns, and their derivation functions. Design rules subsequently allow the database designer to use the properties for (view, logical, and physical) schema design.

The paper is structured as follows. Section 2 first reviews the temporal data model used in the paper. It then argues that the properties of attributes are relative to the objects they describe and then introduces surrogates for representing real-world objects in the model. The following subsections address in turn different aspects of time-varying attributes,namely lifespans, time patterns, and derivation functions. Section 3 is devoted to the implications of the attribute semantics for logical schema, physical schema, and view design. The final section summarizes and points to opportunities for further research.

2 Capturing the Semantics of Time-Varying Attributes

This section provides concepts that allow the database designer to capture more precisely and concisely than hitherto the time-varying nature of attributes in temporal relations. The temporal data model employed in the paper is first described. Then a suite of concepts for capturing the temporal semantics of attributes are introduced.

2.1 A Conceptual Data Model

We describe briefly the relation structures of the Bitemporal Conceptual Data Model (BCDM) (see [9] for a more complete description) that is the data model of TSQL2 and which is used in this paper.

We adopt a linear, discrete, bounded model of time, with a time line composed of chronons. The schema of a bitemporal conceptual relation, R, consists of an arbitrary number, e.g., n, of explicit attributes and an implicit timestamp attribute, T, defined on the domain of sets of bitemporal chronons. A bitemporal chronon $c^b = (c^t, c^v)$ is an ordered pair of a transaction-time chronon c^t and a valid-time chronon c^v. A tuple $x = (a_1, a_2, \dots, a_n |\ t^b)$, in a relation instance r of schema R thus consists of n attribute values associated with a bitemporal timestamp value. An arbitrary subset of the domain of valid times is associated with each tuple, meaning that the information recorded by the tuple is *true in the modeled reality* during each valid-time chronon in the subset. Each individual valid-time chronon of a single tuple has associated a subset of the domain of transaction times, meaning that the information, valid during the particular chronon, is *current in the relation* during each of the transaction-time chronons in the subset. Any subset of transaction times less than the current time may be associated with a valid time. Notice that while the definition of a bitemporal chronon is symmetric, this explanation is asymmetric, reflecting the different semantics of transaction and valid time.

We have thus seen that a tuple has associated a set of so-called *bitemporal chronons* in the two-dimensional space spanned by transaction time and valid time. Such a set is termed a *bitemporal element* [4, 6]. We assume that a domain of surrogate values is available for representing real-world objects in the database.

Example 1. Consider the relation instance, `empDep`, shown next.

`EName`	`Dept`	T
Bob	Ship	$\{(5,10),\ldots,(5,15),\ldots,(9,10),\ldots,(9,15),(10,5),\ldots,(10,20),\ldots,$ $(14,5),\ldots,(14,20),(15,10),\ldots,(15,15)\ldots,(19,10),\ldots,(19,15)\}$
Bob	Load	$\{(20,10),\ldots,(20,15),(21,10),\ldots,(21,15)\}$

The relation shows the employment information for an employee, Bob, and two departments, Ship and Load, contained in two tuples. In the timestamps, we assume that the chronons correspond to days and that the period of interest is some given month in a given year, e.g., July 1995. Throughout, we use integers as timestamp components. The reader may informally think of these integers as dates, e.g., the integer 15 in a timestamp represents the date July 15, 1995. The current time is assumed to be 21.

Valid-time relations and transaction-time relations are special cases of bitemporal relations that support only valid time and transaction time, respectively. For clarity, we use the term snapshot relation for a conventional relation, which supports neither valid time nor transaction time.

This completes the description of the objects in the bitemporal conceptual data model—relations of tuples timestamped with temporal elements. An associated algebra and user-level query language are defined elsewhere [13, 14].

2.2 Using Surrogates

An attribute is seen in the context of a particular real-world entity. Thus, when we talk about a property, e.g., the frequency of change, of an attribute, that property is only meaningful when the attribute is associated with a particular entity. As an example, the frequency of change of a salary attribute with respect to a specific employee in a company may reasonably be expected to be relatively regular, and there will only be at most one salary for the employee at each point in time. In contrast, if the salary is with respect to a department, a significantly different pattern of change may be expected. There will generally be many salaries associated with a department at a single point in time. Hence, it is essential to identify the reference object when discussing the semantics of an attribute.

We employ surrogates for representing real-world entities in the database. In this regard, we follow the approach adopted in, e.g., the TEER model by Elmasri [3]. Surrogates do not vary over time in the sense that two entities identified by identical surrogates are the same entity, and two entities identified by different surrogates are different entities. We assume the presence of surrogate attributes throughout logical design. At the conclusion of logical design, surrogate attributes may be either retained, replaced by regular (key) attributes, or eliminated.

2.3 Lifespans of Individual Time-Varying Attributes

In database design, one is interested in the interactions among the attributes of the relation schemas that make up the database.

Here, we provide a basis for relating the lifespans of attributes. Intuitively, the lifespan of an attribute for a specific object is all the times when the object has a value, distinct from $\perp_i$, inapplicable null, for the attribute. Note that lifespans concern valid time, i.e., are about the times when there exist some valid values.

To more precisely define lifespans, we first define an algebraic selection operator on a temporal relation. Define a relation schema $R = (A_1, \ldots, A_n | \mathrm{T})$, and let r be an instance of this schema. Let P be a predicate defined on the A_i. The selection P on r, $\sigma_P^{\mathrm{B}}(r)$, is defined by $\sigma_P^{\mathrm{B}}(r) = \{z \mid z \in r \wedge P(z[A_1, \ldots, A_n])\}$. It follows that $\sigma_P^{\mathrm{B}}(r)$ simply performs the familiar snapshot selection, with the addition that each selected tuple carries along its timestamp, T. Next, we define an auxiliary function **vte** that takes as argument a valid-time relation r and returns the valid-time element defined by $\mathbf{vte}(r) = \{c^v \mid \exists s\ (s \in r \wedge c^v \in s[\mathrm{T}])\}$. The result valid-time element is thus the union of all valid timestamps of the tuples in an argument valid-time relation.

Definition 1. Let a relation schema $R = (S, A_1, \ldots, A_n \mid \mathrm{T})$ be given, where S is surrogate valued, and let r be an instance of R. The *lifespan* for an attribute A_i, $i = 1, \ldots, n$, with respect to a value s of S in r is denoted $\mathbf{ls}(r, A_i, s)$ and is defined by $\mathbf{ls}(r, A_i, s) = \mathbf{vte}(\sigma^{\mathrm{B}}_{S=s \wedge A \neq \perp_i}(r))$.

Lifespans are important because attributes are guaranteed to not have any inapplicable null value during their lifespans. Assume that we are given a relation schema `empDep` = (`EmpS`, `EName`, `Dept`) that records the names and departments of employees (represented by the surrogate attribute `EmpS`). If employees always have a name when they have a department, and vice versa, this means that inapplicable nulls are not present in instances of the schema. With lifespans, this property may be stated by saying that for all meaningful instances of `EmpSal` and for all `EmpS` surrogates, attributes `EName` and `Dept` have the same lifespans.

The importance of lifespans in temporal databases has been recognized in the context of data models in the past (c.f. [1, 2, 3]). Our use of lifespans for database design differs from the use of lifespans in database instances. In particular, using lifespans during database design does not imply any need for storing lifespans in the database.

2.4 Time Patterns of Individual Time-Varying Attributes

In order to capture how an attribute varies over time, we introduce the concept of a *time pattern*. Informally, a time pattern is simply a sequence of times.

Definition 2. The *time pattern* T is a partial function from the natural numbers $\mathcal{N}$ to a domain $\mathcal{D}_T$ of times: $T : \mathcal{N} \hookrightarrow \mathcal{D}_T$. If $T(i)$ is defined, so is $T(j)$ for all $j < i$. We term $T(i)$ the i'th time point.

In the context of databases, two distinct types of time patterns are of particular interest, namely observation patterns and update patterns. The *observation pattern* O_A^s, for an attribute A relative to a particular surrogate s, is the times when the attribute is given a particular value, perhaps as a result of an observation (e.g., if the attribute is sampled), a prediction, or an estimation. We adopt the convention that $O_A^s(0)$ is the time when it was first meaningful for attribute A to have a value for the surrogate s. Observation patterns concern valid time. The observation pattern may be expected to be closely related to, but distinct from, the actual (possibly unknown) pattern of change of the attribute in the modeled reality. The *update pattern* U_A^s is the times when the value of the attribute is updated in the database. Thus, update patterns concern transaction time.

Note that an attribute may not actually change value at a time point because it may be the case that the existing and new values are the same. The times when changes take place and the resulting values are orthogonal aspects. In the latter half of Section 3.1, we will return to this distinction.

2.5 The Values of Individual Time-Varying Attributes

We proceed by considering how attributes may encode information about the objects they describe. As the encoding of the transaction time of attributes is typically built into the data model, we consider only valid-time relations.

A relation may record directly when a particular attribute value is valid. Alternatively, what value is true at a certain point in time may be computed from the recorded values. In either case, the relation is considered a valid-time relation. An example clarifies the distinction between the two cases.

Example 2. Consider the two relations shown below. The first, `empSal`, records names and salaries of employees, and the second, `expTemp`, records names and temperature measurements for experiments. Attributes `EmpS` and `ExpS` record surrogates representing employees and experiments, respectively.

EmpS	EName	Sal	T
e1	Bob	30k	$\{1,\ldots,9\}$
e1	Bob	32k	$\{10,\ldots,19\}$
e1	Bob	36k	$\{30,\ldots,39\}$
e1	Bob	40k	$\{40,\ldots,49\}$
e2	Sam	25k	$\{1,\ldots,19\}$
e2	Sam	30k	$\{20,\ldots,49\}$

`empSal`

ExpS	Exp	Temp	T
x1	Exp1	75	$\{5,65\}$
x1	Exp1	89	$\{15\}$
x1	Exp1	98	$\{25\}$
x1	Exp1	90	$\{35\}$
x1	Exp1	84	$\{45\}$
x1	Exp1	79	$\{55\}$

`expTemp`

Relation `empSal` records Bob's and Sam's salaries at all the times they have salaries. This is clearly consistent with what a valid-time relation is. At first sight, relation `expTemp` is more problematic. It does not appear to record temperatures for all the times when there exists a temperature for experiment x1. Specifically, we may envision that the temperature of x1 is sampled regularly and that we may later want to compute x1 temperature values for times with no explicitly recorded value.

Traditionally, `empSal` has been considered a state relation and `expTemp` has been considered an event relation; most data model proposals (with notable exceptions, e.g., [13, 15, 16]) have considered only the first type of relation. However, note that the relations are similar in the sense that they both record when information is true. Due to this observation, we make no fundamental distinction between the two types of relations, but instead treat them quite similarly.

The difference between relations such as `empSal` and `expTemp` in the example above is solely in what additional, or even different, information is implied by each of the relations. At the one extreme, relation `empSal` does not imply any additional information at all. No salary is recorded for Bob from time 20 to time 29, and the existing tuples do not imply any salary for Bob in that time interval. The other sample relation is different. For example, while no temperature for Exp1 at time 40 is recorded, clearly such a temperature exists. Further, we may even have a good idea what the temperature may be (i.e., close to 87).

Thus, the difference is that different *derivation functions* apply to the salary and temperature attributes of the two relations. A derivation function f_A for a specific attribute A of a relation schema R takes as arguments a valid-time chronon c^v and a relation instance r and returns a value in the domain of attribute A. For the salary attribute, a discrete derivation function applies; and for the temperature, a nearest-neighbor derivation function may satisfy some users while other users may need a more sophisticated function.

Definition 3. A *derivation function* f is a partial function from the domains of valid times $\mathcal{D}_{VT}$ and relation instances r with schema R to a value domain D in the universal set of domains $\mathcal{D}_D$, i.e., $f : \mathcal{D}_{VT} \times r(R) \hookrightarrow D$.

The importance of derivation functions in data models has previously been argued convincingly by, e.g., Klopprogge [10], Clifford [1] and Segev [16]. They should thus also be part of a design methodology.

2.6 Summary of Attribute Semantics

In summary, the database designer is expected to initially identify and model entity types using surrogates. Then, the notions of lifespans, time patterns, and derivation functions are used for capturing the semantics of attributes.

Elsewhere, we have generalized conventional functional dependencies to temporal databases [7]. Essentially, a *temporal dependency* holds on a temporal relation if the corresponding snapshot dependency holds on each snapshot relation contained in the temporal relation. With this generalization, conventional relational dependency theory applies wholesale to temporal databases. For example, *temporal keys* may be defined. Such keys are generally time-varying. As a basis for defining time-invariant attributes and keys, we have also defined so-called *strong temporal functional dependencies* and *strong temporal key* [8]. While not discussed here, the designer is also expected to identify temporal and strong temporal functional dependencies.

3 Temporal Relational Database Design Guidelines

In this section, we discuss how the properties of schemas with time-varying attributes as captured in the previous section are used during database design. Emphasis is on the implications of the properties for design of the logical schema, but implications for view design and physical design are touched upon as well.

3.1 Logical-Design Guidelines

Two important goals of logical database design are to design a database schema that does not require the use of inapplicable nulls and avoids representation of the same information. We define two properties that illuminate these aspects of relation schemas and guide the database designer.

Database designers are faced with a number of design criteria which are sometimes conflicting, making database design a challenging task. So, while we discuss certain design criteria in isolation, it is understood that there may be other criteria that should be taken into consideration during database design, such as minimizing the impact of joins required on relations that have been decomposed.

Lifespan Decomposition Rule One important design criterion in conventional relational design is to eliminate the need for inapplicable nulls in tuples of database instances. In the context of temporal databases, we use the notion of lifespans to capture when attributes are defined for the objects they are introduced in order to describe. Briefly, the lifespan for an attribute—with respect to a particular surrogate representing the object described by the attribute—is all the times when a meaningful attribute value, known or unknown, exists for the object.

Inapplicable nulls may occur in a relation schema when two attributes have different lifespans for the same object/surrogate. To identify this type of situation, we introduce the notion of lifespan equal attributes. Examples follow the the definition.

Definition 4. Let a relation schema $R = (S, A_1, \ldots, A_n \,|\, T)$ be given where S is surrogate valued. Two attributes A_i and A_j in R are termed *lifespan equal* with respect to surrogate S, denoted $A_i \overset{\text{LS}}{=}_S A_j$, if for all meaningful instances r of R, $\forall s \in \text{dom}(S)$ $(\mathbf{ls}(r, A_i, s) = \mathbf{ls}(r, A_j, s))$.

To exemplify this definition, consider a relation schema `Emp` with attributes `EmpS` (employee surrogates), `Dept`, `Salary`, and `MgrSince`. The schema is used by a company where each employee is always assigned to some department and has a salary. In addition, the relation records when an employee in a department first became a manager in that department.

For this schema, we have `Dept` $\overset{\text{LS}}{=}_{\texttt{EmpS}}$ `Salary` because an employee has a salary (it might be unknown or zero) exactly when associated with a department. Thus, no instances of `Emp` will have tuples with an inapplicable-null value for one of `Dept` and `Salary` and not for the other. Next, it is not the case that `Dept` $\overset{\text{LS}}{=}_{\texttt{EmpS}}$ `MgrSince` and (by inference) not the case that `Salary` $\overset{\text{LS}}{=}_{\texttt{EmpS}}$ `MgrSince`. This is so because employees often are associated with a department where they have never been a manager. Thus, instances of `Emp` may contain inapplicable nulls. Specifically, the nulls are associated with attribute `MgrSince` as the lifespan of this attribute is shorter than that of `Dept` and `Salary`.

Next, observe that `Dept` and `Salary` being lifespan equal with respect to `EmpS` does not mean that all employees have the same lifespan for their department (or salary) attribute. Employees may have been hired at different times, and the lifespans are thus generally different for different employees. Rather, the equality is between the department and the salary lifespan for individual employees.

The following definition then characterizes temporal database schemas with instances that do not contain inapplicable nulls.

Definition 5. A relation schema $R = (S, A_1, \ldots, A_n \mid T)$ where S is surrogate valued is *lifespan homogeneous* if $\forall A, B \in R \; (A \overset{\text{LS}}{=}_S B)$.

These concepts formally tie the connection between the notion of lifespans of attributes with the occurrence of inapplicable nulls in instances. With them, we are in a position to formulate the Lifespan Decomposition Rule.

Definition 6. *Lifespan Decomposition Rule.* To avoid inapplicable nulls in temporal database instances, decompose temporal relation schemas to ensure lifespan homogeneity.

It is appropriate to briefly consider the interaction of this rule with the the existing temporal normal forms that also prescribe decomposition of relation schemas. Specifically, while the decomposition that occurs during normalization does, as a side effect, aid in eliminating the need for inapplicable nulls, a database schema that obeys the temporal normal forms may still require inapplicable nulls in its instances. To exemplify, consider again the `Emp` schema (and think of the temporal dependencies on temporal relations as regular dependencies on the corresponding snapshot tables). Here, `EmpS` is a temporal key, and there are no other non-trivial dependencies. Thus, the schema is in temporal BCNF. It is also the case that `Emp` has no non-trivial temporal multi-valued dependencies, and it is thus also in temporal fourth normal form. In spite of this, we saw that there are inapplicable nulls. The solution is to decompose `Emp = (EmpS, Dept, Salary, MgrSince)` into `Emp1 = (EmpS, Dept, Salary)` and `Emp2 = (EmpS, MgrSince)`. Both resulting relations are lifespan homogeneous.

Synchronous Decomposition Rule The synchronous decomposition rule is based on the notion of observation pattern, and its objective is to eliminate a particular kind of redundancy. We initially exemplify this type of redundancy. Then we define the notion of synchronous attributes, which leads to a definition of synchronous schemas and an accompanying decomposition rule that are aimed at avoiding this redundancy. Finally, we view synchronism in a larger context, by relating it to existing concepts, and discuss the decomposition rule's positioning with respect to logical versus physical design.

Example 3. Consider the relation instance, `empDepSal`, that follows next, recording departments and salaries for employees. The schema for the relation is in temporal BCNF, with the surrogate-valued attribute `EmpS` being the only minimal key and no other non-trivial dependencies. Yet, it may be observed that the salary 30k and the departments A and B are repeated once, once, and four times in the instance, respectively. These repetitions are due to attributes `Dept` and `Salary` having different observation patterns. Specifically, the instance is consistent with the patterns shown to the right of the instance.

EmpS	Dept	Salary	T
e1	A	30k	$\{1,\ldots,5\}$
e1	B	30k	$\{6,\ldots,9\}$
e1	B	32k	$\{10,\ldots,14\}$
e1	B	36k	$\{15,\ldots,27\}$
e1	B	40k	$\{28,\ldots,42\}$
e1	A	50k	$\{43,\ldots,49\}$

$$O^{\mathtt{e1}}_{\mathtt{Dept}} =< [0 \mapsto 1], [1 \mapsto 6], [2 \mapsto 43], [3 \mapsto 50] >$$

$$O^{\mathtt{e1}}_{\mathtt{Salary}} =< [0 \mapsto 1], [1 \mapsto 10], [2 \mapsto 15], [3 \mapsto 28], [4 \mapsto 43], [5 \mapsto 50] >$$

In combination, these observation patterns imply the redundancy that may be observed in the sample instance. Thus, capturing during database design which attributes of the same relation schema have different observation patterns is a means of identifying this type of redundancy.

To capture precisely the synchronism of attributes, define $T|_t$ to be the restriction of time pattern T to the valid-time element t, that is, to include only those times also contained in t.

Definition 7. Define relation schema $R = (S, A_1, \ldots, A_n \mid \mathrm{T})$ where S is surrogate valued. Two attributes A_i and A_j in R, with observation patterns $O^S_{A_i}$ and $O^S_{A_j}$, are *synchronous* with respect to S, denoted $A_i \overset{\mathrm{S}}{=}_S A_j$, if for all meaningful instances r of R and for all surrogates s,

$$O^S_{A_i}|_{\mathbf{ls}(r,A_i,s) \cap \mathbf{ls}(r,A_j,s)} = O^S_{A_j}|_{\mathbf{ls}(r,A_i,s) \cap \mathbf{ls}(r,A_j,s)} .$$

Thus, attributes are synchronous if their lifespans are identical when restricted to the intersection of their lifespans. With this definition, we can characterize relations that avoid the redundancy caused by a lack of synchronism and then state the Synchronous Decomposition Rule.

Definition 8. Define relation schema $R = (S, A_1, \ldots, A_n \mid \mathrm{T})$ where S is surrogate valued. Relation R is *synchronous* if $\forall A_i, A_j \in R\ (A_i \overset{\mathrm{S}}{=}_S A_j)$.

Definition 9. *Synchronous Decomposition Rule.* To avoid repetition of attribute values in temporal relations, decompose relation schemas until they are synchronous.

Alternative notions of synchronism have previously been proposed for database design by Navathe and Ahmed [12], Lorentzos [11], and Wijsen [18]. While these notions are stated with varying degrees of clarity and precision and are defined in different data-model contexts, they all seem to capture the same basic idea, namely that of *value-based synchronism* which is different from the synchronism proposed in this paper.

To explain the difference, consider the relation instance shown next.

S	A_i	A_j	T
s	a_1	a_2	$\{1, \ldots, 5\}$
s	a_1	a_3	$\{6, \ldots, 10\}$

In value-based synchronism, value-changes of attributes must occur synchronously for attributes to be synchronous. Consequently, the relation instance implies that the attributes A_i and A_j in its schema are not value synchronous, and (value-based) decomposition is thus prescribed. Next, it may be that the attributes in the relation have identical observation patters and that it just (accidentally!) happened that the new value of A_i when both attributes were observed at time 6 was the same as its old value. This means that the relation is consistent with attributes A_i and A_j being (observation-pattern based) synchronous, and the synchronous decomposition rule does then *not* apply. To conclude, the value-based and pattern-based synchronisms are quite unrelated.

Further, it is our contention that using the concept of value-based synchronism during database design is problematic. Specifically, it seems quite rare that the database designer can guarantee that, at *all* times in the future, when two attributes in a tuple of relation are updated, one of them does not get a new value that is identical to its old value. Thus, it appears that decomposition based on value-based synchronism effectively (and unnecessarily) leads to a binary data model, in which all relations have just two attributes, a time invariant attribute and a single time-varying attribute. This is in contrast to the pattern-based decomposition prescribed in this paper.

This study is carried out in the context of TSQL2. It is our contention that in this context, the synchronous decomposition rule is relevant only to physical database design. Surely, the redundancy that may be detected using the synchronism concept is important when *storing* temporal relations. At the same time, this type of redundancy is of little consequence for the querying of logical-level relations using the TSQL2 query language [8, 13]. Indeed, it will often adversely affect the ease of formulating queries if logical-level relations are decomposed solely based on a lack of synchronism. In conclusion, the presence of synchronous attributes in a relation may affect performance (positively or negatively), but it does not negatively affect correctness or the ease of formulating queries, and it is thus a non-issue at the logical level.

The widespread presence of asynchronous attributes in relation schemas has been used for motivating various attribute-value timestamped data models where in relations, time is associated with attribute values rather than with tuples, because these models avoids the redundancy (see, e.g., [2, 4, 5]). Since this redundancy is not a problem in TSQL2, which employs tuple-timestamped relations, asynchronous attributes is not strictly an argument for attribute-value timestamped models.

Finally, the need for synchronism at the logical level has previously been claimed to make normal forms and dependency theory inapplicable (e.g., [5]). The argument is that few attributes are synchronous, meaning that relation schemas must be maximally decomposed, which leaves other normalization concepts irrelevant. This claim does not apply to our data model.

For completeness, it should be mentioned that while the synchronism concepts presented in this section have concerned valid time, similar concepts that concern transaction time and employ update patterns rather than observation patterns, may also be defined.

3.2 Implications for View Design

The only concept from Section 2 not covered so far is derivation functions. These relate to view design, as outlined next.

For each time-varying attribute, we have captured a set of one or more derivation functions that apply to it. It is often the case that exactly one derivation function applies to an attribute, namely the discrete interpolation function [8] that is a kind of identity function. However, it may also be the case that several nontrivial derivation functions apply to a single attribute.

The problem is then how to apply several derivation functions to the base data. We feel that there should be a clear separation between recorded data and data derived from the stored data via some function. Maintaining this separation makes it possible to later modify existing interpolation functions.

The view mechanism provides an ideal solution that maintains this separation. Thus, the database designer first identifies which sets of derivation functions that should be applied simultaneously to the attributes of a logical relation instance and then, subsequently, defines a view for each such set. Although interpolation functions have previously been studied, we believe they have never before been associated with the view mechanism.

4 Summary and Research Directions

In order to exploit the full potential of temporal relational database technology, guidelines for the design of temporal relational databases should be provided.

This paper has presented concepts for capturing the properties of time-varying attributes in temporal databases. These concepts include surrogates that represent the real-world objects described by the attributes, lifespans of attributes, observation and update patterns for time-varying attributes, and derivation functions that compute new attribute values from stored ones. It was subsequently shown how surrogates and lifespans play an role during design of the logical database schema. In particular, the notion of lifespans led to the formulation of a lifespan decomposition rule. The notion of observation (and update) patterns led to a synchronous decomposition rule; it was argued that this rule should ideally apply to physical database design. Finally, it was shown how derivation functions are relevant for view design.

We feel that several aspects merit further study. An integration of the various existing contributions to temporal relational database design into a coherent framework has yet to be attempted. Likewise, a complete design methodology, including conceptual (implementation-data-model independent) design and logical design, for temporal databases is warranted. Finally, a next step is to adopt the concepts provided in this paper in richer, entity-based (or semantic or object-based) data models.

Acknowledgements
This work was supported in part by NSF grant ISI-9202244. In addition, the first author was supported in part by the Danish Natural Science Research Council, grants 11–1089–1, 11–0061–1, and 9400911.

References

1. J. Clifford and A. Croker. The Historical Relational Data Model (HRDM) and Algera Based on Lifespans. In *Proceedings of ICDE*, pp. 528–537, February 1987.
2. J. Clifford and A. U. Tansel. On an Algebra for Historical Relational Databases: Two Views. In *Proceedings of ACM SIGMOD*, pp. 247–265, May 1985.
3. R. Elmasri, G. Wuu, and V. Kouramajian. A Temporal Model and Query Language for EER Databases. In [17], pp. 212–229.
4. S. K. Gadia. A Homogeneous Relational Model and Query Languages for Temporal Databases. *ACM TODS*, 13(4):418–448, December 1988.
5. S. K. Gadia and J. H. Vaishnav. A Query Language for a Homogeneous Temporal Database. In *Proceedings of ACM PODS*, pp. 51–56, March 1985.
6. C. S. Jensen, J. Clifford, R. Elmasri, S. K. Gadia, P. Hayes, and S. Jajodia (eds). A Glossary of Temporal Database Concepts. *SIGMOD Record*, 23(1):52–64, March 1994.
7. C. S. Jensen, R. T. Snodgrass, and M. D. Soo. Extending Normal Forms to Temporal Relations. Technical Report TR-92-17, Department of Computer Science, University of Arizona, Tucson, AZ, July 1992.
8. C. S. Jensen and R. T. Snodgrass. Semantics of Time-Varying Attributes and Their Use for Temporal Database Design. Technical Report R-95–2012, Department of Math. and Computer Science, Aalborg University, Denmark, May 1995.
9. C. S. Jensen, M. D. Soo, and R. T. Snodgrass. Unifying Temporal Models via a Conceptual Model. *Information Systems*, 19(7):513–547, 1994.
10. M. R. Klopprogge and P. C. Lockemann. Modelling Information Preserving Databases: Consequences of the Concept of Time. In *Proceedings of VLDB*, pp. 399–416, 1983.
11. N. A. Lorentzos. Management of Intervals and Temporal Data in the Relational Model. Technical Report 49, Agricultural University of Athens, 1991.
12. S. B. Navathe and R. Ahmed. A Temporal Relational Model and a Query Language. *Information Sciences*, 49:147–175, 1989.
13. R. T. Snodgrass, editor, The TSQL2 Temporal Query Language. Kluwer Academic Publishers, 1995, 674+xxiv pages.
14. M. D. Soo, C. S. Jensen, and R. T. Snodgrass. An Algebra for TSQL2. In [13], chapter 27, pp. 505–546.
15. R. T. Snodgrass. The Temporal Query Language TQuel. *ACM TODS*, 12(2):247–298, June 1987.
16. A. Segev and A. Shoshani. A Temporal Data Model based on Time Sequences. In [17], pp. 248–270.
17. A.U. Tansel, J. Clifford, S.K. Gadia, A. Segev, and R.T. Snodgrass, eds, Temporal Databases: Theory, Design, and Implementation. Benjamin/Cummings, 1993.
18. J. Wijsen. Extending Dependency Theory for Temporal Databases. Ph.D. Thesis. Department Computerwetenschappen, Katholieke Universiteit Leuven, 1995.