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

Christian S. Jensen Richard T. Snodgrass

Abstract

This paper concerns the design of temporal relational database schemas.

Normal forms play a central role during the design of conventional relational databases, and we have previously extended all existing relational normal forms to apply to temporal relations. However, these normal forms are all atemporal in nature and do not fully take into account the temporal semantics of the attributes of temporal relations. Consequently, additional guidelines for the design of temporal relations are required.

This paper presents a systematic study of important aspects of the temporal semantics of attributes. One such aspect is the observation and update patterns of attributes—when an attribute changes value and when the changes are recorded in the database. A related aspect is when the attributes have values. A third aspect is the values themselves of attributes—how to derive a value for an attribute at any point in time from stored values. Guidelines for the design of the logical schema of a temporal database are introduced, and implications of the temporal-attribute semantics for the design of views and the physical schema are considered. The Bitemporal Conceptual Data Model, the data model of the consensus temporal query language TSQL2, serves as the context for the study.

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 [SA+94], 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. This paper contributes to the development of such a theory.

We have previously shown how conventional relational normalization concepts, e.g., functional dependencies, keys, and Boyce-Codd Normal Form, may be extended to apply to temporal databases [JSS92]. However, being atemporal, these extended normalization concepts are limited in scope, and additional concepts are needed in order to fully address the time-varying nature of data.

In this paper we focus on the semantics of time-varying attributes of temporal relations and develop concepts that capture the temporal semantics of such attributes. We subsequently formulate decomposition guidelines for the design of appropriate temporal relational database schemas that are based on these temporal semantics. This allows the database designer to capture

*C. S. Jensen is with Department of Mathematics and Computer Science, Aalborg University, Fredrik Bajers Vej 7E, DK–9220 Aalborg Ø, DENMARK, csj@iesd.auc.dk. R. T. Snodgrass is with Department of Computer Science, University of Arizona, Tucson, AZ 85721, USA, rts@cs.arizona.edu.

the important temporal characteristics of the time-varying attributes of relation schemas and then decompose the schemas in accordance with the guidelines.

More specifically, the paper proposes to capture the properties of time-varying attributes by describing their lifespans, their time patterns, their interpolation functions, their associated temporal granularities, and the (generalized and strong) temporal functional dependencies they participate in. The implications of these properties for design of the logical schema, the physical schema, and for view design are covered. Concerning the logical schema, the implications are formulated as decomposition rules.

We have previously provided an extensive survey of existing contributions to the design of temporal databases [JSS92]. Here, we briefly indicate how the contributions of this paper complement the most closely related existing contributions. More detailed comparisons are deferred to Section 4, after our proposals have been presented.

As the most closely related research, Navathe and Ahmed [NA89, Ahm92] define a temporal dependency, intended to capture the notion of synchronous attributes, and use it for defining a temporal normal form that states that relations with asynchronous attributes should be decomposed. We point to a deficiency in this normal form and propose an improved version. Wijsen and his colleagues have proposed functional dependencies for data models with and without object identity [WVO93a, WVO93b, WVO93c, WVO94a, WVO94b]. While defined for somewhat different data models, these dependencies parallel and complement our existing temporal functional dependencies [JSS92], which are generalized in this paper. The present dependencies also generalize those provided by Lorentzos in the context of a relational model with interval-valued attributes and special algebraic FOLD and UNFOLD operators for manipulating such attributes [Lor91]. Lorentzos has also defined a P Normal Form and a Q Normal Form [Lor91]. The former is subsumed by our previous normalization concepts [JSS92], and the latter appears to be similar to the normal form by Navathe and Ahmed, addressed above. Temporal normal forms proposed by BenZvi [BZ82] and Segev and Shoshani [SS88b] were defined for quite different purposes than the usual normal forms and the guidelines presented in this paper.

The paper is structured as follows. The next section provides a focused review of the temporal data model that provides the concrete context for the contribution of the paper. That section first discusses the representation of time and then adds time to relations. Next, a few algebraic operations, to be used in subsequent sections, are defined on the resulting relations. Finally, it is argued that the chosen data model is in some sense a conceptual model and that this model may be integrated into a DBMS with other models that may be more suitable for the presentation and storage of temporal data. Section 3 first 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 different aspects of time-varying attributes, e.g., lifespans and time patterns. The section is concluded by a template for capturing the semantics of time-varying attributes. Section 4 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 A Conceptual Data Model

This section sets the context for discussing guidelines for temporal database design. Specifically, we first adopt a particular model of time itself, then add time to conventional relations to yield a temporal data model for which essential algebraic operators are defined. Finally, the context within a temporal DBMS of the resulting data model is considered.

2.1 Modeling and Representing Time

Most physicists perceive the *real* time line as being bounded, the lower bound being the Big Bang (which is believed to have occurred approximately 14 billion years ago) and the and upper bound being the Big Crunch. There is no general agreement as to whether the real time line is continuous or discrete, but there is general agreement in the temporal database community that a discrete *model* of time is adequate.

Consequently, our model of the real time line is that of a finite sequence of chronons. In mathematical terms, this is isomorphic to a finite sequence of natural numbers [JS94b]. The sequence of chronons may be thought of as representing a partitioning of the real time line into equal-sized, indivisible segments. Thus, chronons are thought of as representing time segments such as femtoseconds or seconds, depending on the particular data processing needs. Real-world time instants are assumed to be much smaller than chronons and are represented in the model by the chronons during which they occur. We will use c, possibly indexed, to denote chronons.

A time interval is defined as the time between two instants, a starting and a terminating instant. A time interval is then represented by a sequence of consecutive chronons where each chronon represent all instances that occurred during the chronon. We may also represent a sequence of chronons simply by the pair of the starting and terminating chronon. The restriction that the starting instant must be before the ending instant is necessary for the definition to be meaningful in situations where an interval is represented by, e.g., a pair of identical chronons. Unions of intervals are termed *temporal elements* [Gad88].

2.2 Relations with Time

Two kinds of time are of general relevance to data recorded in a database. Here, we characterize and add these notions of time to relations. The resulting temporal relations are those of the *bitemporal conceptual data model*, or BCDM [JSS94], upon which TSQL2 is based.

To capture the time-varying nature of data, time values from two orthogonal time domains, namely valid time and transaction time, are associated with the tuples in a bitemporal conceptual relation instance. Valid time is used for capturing the time-varying nature of the portion of reality being modeled, and transaction time models the update activity associated with the relation.

For both time domains, we employ the model of time outlined in the previous section. The domain of valid times is given as $\mathcal{D}_{VT} = \{c_1^v, c_2^v, \ldots, c_k^v\}$, and the domain of transaction times may be given as $\mathcal{D}_{TT} = \{c_1^t, c_2^t, \ldots, c_j^t\}$ A valid-time chronon c^v is thus a member of $\mathcal{D}_{VT}$, a transaction-time chronon c^t is a member of $\mathcal{D}_{TT}$, and a bitemporal chronon $c^b = (c^t, c^v)$ is an ordered pair of a transaction-time chronon and a valid-time chronon.

Next, we define a set of names, $\mathcal{D}_A = \{A_1, A_2, \ldots, A_{n_A}\}$, for explicit attributes and a set of domains for these attributes, $\mathcal{D}_D = \{D_1, D_2, \ldots, D_{n_D}\}$. For these domains, we use $\perp_i$, $\perp_u$, and $\perp$ as inapplicable, unknown, and inapplicable-or-unknown null values, respectively (see, e.g., [Zan82, AD93]). We also assume that a domain of surrogates is included among these domains. Surrogates are system-generated unique identifiers the values of which cannot be seen but only compared for identity [HOT76]. Surrogate values are used for representing real-world objects. With the preceding definitions, the schema of a bitemporal conceptual relation, R, consists of an arbitrary number, e.g., n, of explicit attributes from $\mathcal{D}_A$ with domains in $\mathcal{D}_D$, and an implicit timestamp attribute, T, with domain $2^{(\mathcal{D}_{TT} \cup \{UC\}) \times \mathcal{D}_{VT}}$. Here, UC ("until changed") is a special transaction-time marker. A value (UC, c^v) in a timestamp for a tuple indicates that the tuple being valid at time c^v is current in the database. The example below elaborates on this.

A set of bitemporal functional (and multivalued) dependencies on the explicit attributes are part of the schema. For now, we ignore these dependencies—they are treated in detail later.

A tuple $x = (a_1, a_2, \ldots, a_n | \ t^b)$, in a bitemporal conceptual relation instance, $r(R)$, consists of a number of attribute values associated with a bitemporal timestamp value. For convenience, we will employ the term "fact" to denote the information recorded or encoded by a tuple.

An arbitrary subset of the domain of valid times is associated with each tuple, meaning that the fact 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 fact, 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 and including the value UC may be associated with a valid time. Notice that while the definition of a bitemporal chronon is symmetric, this explanation is asymmetric. This asymmetry reflects 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* [JCE+94] and is denoted t^b. Because no two tuples with mutually identical explicit attribute values (termed *value-equivalent*) are allowed in a bitemporal relation instance, the full history of a fact is contained in a single tuple.

In graphical representations of bitemporal space, we choose the x-axis as the transaction-time dimension, and the y-axis as the valid-time dimension. Hence, the ordered pair (c^t, c^v) represents the bitemporal chronon with transaction time c^t and valid time c^v.

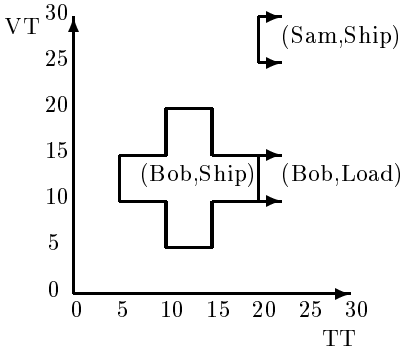
EXAMPLE 1: Consider a relation recording employee/department information, such as "Bob works for the shipping department." We assume that the granularity of chronons is one day for both valid time and transaction time, and the period of interest is some given month in a given year, e.g., January 1995. 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 January 15, 1995. The current time is assumed to be 19 (i.e., $now = 19$).

Figure 1(a) shows an instance, `empDep`, of this relation. A graphical illustration of the `empDep` relation is shown in Figure 1(b). Right-pointing arrows in the graph and the special value UC in the relation signify that the given tuple is still current in the database and that new chronons will be added to the timestamps as time passes and until the tuple is logically deleted.

The relation shows the employment information for two employees, Bob and Sam, contained in three tuples. The first two tuples indicate when Bob worked for the shipping and loading departments, respectively. These two tuples are shown in the graph as the regions labelled "(Bob, Ship)," and "(Bob, Load)," respectively. The last tuple indicates when Sam worked for the shipping department, and corresponds to the region of the graph labelled "(Sam, Ship)." □

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	$\{(UC,10),\ldots,(UC,15)\}$
Sam	Ship	$\{(UC,25),\ldots,(UC,30)\}$

(a)



(b)

Figure 1: A Bitemporal Conceptual Relation

Depending on the extent of decomposition, a tuple in a bitemporal relation may be thought of as encoding an atomic or a composite fact. We simply use the terminology that a tuple encodes a fact and that a bitemporal relation instance is a collection of (bitemporal) facts.

Valid-time relations and transaction-time relations are special cases of bitemporal relations that support only valid time or transaction time, respectively. Thus a valid-time tuple has associated a set of valid-time chronons (termed a *valid-time element* and denoted t^v), and a transaction-time tuple has associated a set of transaction-time chronons (termed a *transaction-time element* and denoted t^t). For clarity, we use the term *snapshot relation* for a conventional relation. Snapshot relations support neither valid time nor transaction time.

2.3 Associated Algebraic Operators

We have so far described the objects in the bitemporal conceptual data model—relations of tuples timestamped with bitemporal elements. We now define some algebraic operators on these objects that will be used later. A complete algebra for the BCDM is defined elsewhere [SJS94].

We first define bitemporal analogues of some of the snapshot relational operators, to be denoted with the superscript "B".

Define a relation schema $R = (A_1, \ldots, A_n|\mathrm{T})$, and let r be an instance of this schema. We will use A as a shorthand for all attributes A_i of R. Let D be an arbitrary set of explicit (i.e., non-timestamp) attributes of relation schema R. The projection on D of r, $\pi^{\mathrm{B}}_D(r)$, is defined as follows.

$$\pi^{\mathrm{B}}_D(r) = \{z^{(|D|+1)} \mid \exists x \in r\ (z[D] = x[D]) \wedge \forall y \in r\ (y[D] = z[D] \Rightarrow y[\mathrm{T}] \subseteq z[\mathrm{T}]) \wedge \forall t \in z[\mathrm{T}]\ \exists y \in r\ (y[D] = z[D] \wedge t \in y[\mathrm{T}])\}$$

The first line ensures that no chronon in any value-equivalent tuple of r is left unaccounted for, and the second line ensures that no spurious chronons are introduced.

Let P be a predicate defined on $A_1, \ldots, A_n$. The selection P on r, $\sigma^{\mathrm{B}}_P(r)$, is defined as follows.

$$\sigma^{\mathrm{B}}_P(r) = \{z \mid z \in r\ \wedge\ P(z[A])\}$$

As can be seen from the definition, $\sigma^{\mathrm{B}}_P(r)$ simply performs the familiar snapshot selection, with the addition that each selected tuple carries along its timestamp T.

Finally, we define two operators that select on valid time and transaction time. They have no counterparts in the snapshot relational algebra. Let c^v denote an arbitrary valid-time chronon and let c^t denote a transaction-time chronon. The *valid-timeslice* operator (τ^{B}) yields a transaction-time relation; the *transaction-timeslice* operator (ρ^{B}) evaluates to a valid-time relation[1].

$$\begin{aligned} \tau^{\mathrm{B}}_{c^v}(r) &= \{z^{(n+1)} \mid \exists x \in r\ (z[A] = x[A] \wedge z[\mathrm{T}] = \{c^t | (c^t, c^v) \in x[\mathrm{T}]\} \wedge z[\mathrm{T}] \neq \emptyset)\} \\ \rho^{\mathrm{B}}_{c^t}(r) &= \{z^{(n+1)} \mid \exists x \in r\ (z[A] = x[A] \wedge z[\mathrm{T}] = \{c^v | (c^t, c^b) \in x[\mathrm{T}]\} \wedge z[\mathrm{T}] \neq \emptyset)\} \end{aligned}$$

Thus, $\tau^{\mathrm{B}}_{c^v}(r)$ simply returns all tuples in r that were valid during the valid-time chronon c^v. The timestamp of a returned tuple is all transaction-time chronons associated with c^v. Next, $\rho^{\mathrm{B}}_{c^t}(r)$ performs the same operation except the selection is performed on the transaction time c^t.

EXAMPLE 2: Consider the `empDep` relation shown in Figure 1(a). The following result is produced by $\tau^{\mathrm{B}}_{12}(\texttt{empDep})$.

[1] Operator ρ was originally termed the *rollback* operator, hence the choice of symbol.

EName	Dept	T
Bob	Ship	$\{5, \ldots, 19\}$
Bob	Load	$\{UC\}$

Using the graphical representation, valid timeslice can be visualized by drawing a horizontal line through the graph at the given valid time. The tuples returned are those that overlap with the drawn line. The timestamps of the returned tuples are set to the segments of transaction time corresponding to the overlapped regions. □

We have only defined operators for bitemporal relations. The similar operators for valid-time and transaction-time relations are simpler special cases and are omitted for brevity. We will use superscripts "T" and "V" for the transaction and valid-time counterparts, respectively.

To extract from r the tuples valid at time c^v and current in the database during c^t (termed a *snapshot* of r), either $\tau^{\mathrm{V}}_{c^v}(\rho^{\mathrm{B}}_{c^t}(r))$ or $\rho^{\mathrm{T}}_{c^t}(\tau^{\mathrm{B}}_{c^v}(r))$ may be used; these two expressions evaluate to the same snapshot relation.

Note that since relations in the data model are *homogeneous*, i.e., all attribute values in a tuple are associated with the same timestamp [Gad88], the valid or transaction timeslice of a relation will not introduce any nulls into the resulting relation.

Later, when additional concepts have been introduced, we will add a derivation operator ξ to the algebra.

2.4 Context

We feel that one reason why more than two dozen temporal data models have been proposed is that attempts have been made to simultaneously retain the simplicity of the relational model, present all the information concerning an object in one tuple, and ensure ease of implementation and query evaluation efficiency. Meeting all of these goals in a single model is a difficult, if not impossible, task, so we advocate a separation of concerns when building a temporal DBMS.

The BCDM data model proposed in this section is intended solely for expressing the time-varying semantics of data and as the basis for a query language, e.g., TSQL2 [JSS94]. Rather than obscure these semantics by concerns of presentation and internal representation, other, more appropriate models are used for those tasks. As a consequence the conceptual database schema is captured using the BCDM.

We have previously shown how the to integrate several temporal data models within the same DBMS [JSS94]. Center to this integration is the concept of *snapshot equivalence*. Snapshot equivalence is a formalization of the notion that two temporal relations have the same information content, and it provides a natural means of comparing rather disparate representations. We have developed mappings, respecting snapshot equivalence, between instances of the BCDM and instances of each of the five existing bitemporal data models. We also showed how the relational algebraic operators defined in the previous section could be mapped to analogous operators in the representational models.

A database designer would design the conceptual schema of the database as a (normalized) collection of BCDM relation schemas. The mappings then make it possible to store and display BCDM relations as snapshot equivalent instances of other data models. This approach yields guidelines for the design of the logical database schema that are independent of any particular representation of a temporal relation.

3 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.

Informally, attributes of an object are time-varying if their values change over time. Time-varying attributes differ with respect to the frequency with which their values change and with respect to how they encode information about the objects they describe. Following an introduction to our use of surrogates as object representatives, we investigate these two aspects of individual time-varying attributes in turn. Then we consider the time-varying nature of the interrelationships among several attributes. The section concludes with a template for capturing the temporal semantics of time-varying attributes of temporal relations.

3.1 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. If the salary is with respect to a department, a significantly different pattern of change may be expected, and there will generally be many salaries associated with a department at a single point in time. Hence, it is of essence to identify the reference object when discussing the semantics of an attribute.

This insight, that it is critical to identify the entity (or object) types that the attributes of the database describe, is not new. For example, when using the ER model for conceptual database design, one identifies entity types (or entity sets) at an early stage in the modeling process.

In this paper, the reference-entity types are represented by surrogate attributes, and the entities are represented by surrogates. In this regard, we follow the approach adopted in, e.g., the TEER model by Elmasri [EWK93]. 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. We discuss when this can occur in Section 3.5.

3.2 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.

In this section, 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. In its full generality, the lifespan is a temporal element, but most often, the lifespan is a single time interval. Note that lifespans concern valid time, i.e., are about the times when there exist some valid values. Lifespans are not related to transaction time.

To more precisely define lifespans, we first define an auxiliary function, **vte**.

DEFINITION 1: The function **vte** takes as argument a valid-time relation r and returns the valid-time element defined as follows.

$$\mathbf{vte}(r) = \{c^v \mid \exists s\ (s \in r \wedge c^v \in s[\mathrm{T}])\} \qquad \square$$

This function returns the valid-time element that is the union of all valid timestamps of the tuples in an argument valid-time relation.

DEFINITION 2: 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 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 as follows.

$$\mathbf{ls}(r, A_i, s) = \mathbf{vte}(\sigma_{S=s \wedge A \neq \perp_i}(r)) \qquad \square$$

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 (identified 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. In Section 4.1.1, we will use lifespans to formulate a *Lifespan Decomposition Rule*.

The importance of lifespans in temporal databases has been recognized in the context of data models in the past. In the HRDM model, Clifford [CT85, CC87, CC93] associates explicit lifespans with each attribute of a relation schema and with each tuple of a relation instance. The HRDM goes further than other data models in incorporating lifespans, but it still does not explicitly record the lifespans of attributes of individual tuples/surrogates (HRDM tuples correspond to our object-representing surrogates), as we do here. Rather, the lifespan of an attribute of a particular object is derived as the intersection of the tuple's lifespan and the relation schema's lifespan for the attribute. In the TEER model, Elmasri [EWK93] associates lifespans with individual surrogates, which represent entities in that model. In another extension to the ER model, TERM, Klopprogge [KL83] records lifespans by adding mandatory, boolean-valued valid-time attributes, "**existence**," to entitity and relationship types.

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.

3.3 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. In Section 4.1.2, we will use time patterns for defining a *Synchronous Decomposition Rule* that guides database design.

DEFINITION 3: 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 $j \in \mathcal{N}$ where $j < i$. We term $T(i)$ the i'th time point. $\square$

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^s_A, 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 $O^s_A(0)$ 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. 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. Note that all times in the observation pattern of an attribute belong to its lifespan. This is not necessarily true for times in the update pattern.

EXAMPLE 3: Consider a valid-time relation schema `ExpTemp` = (`ExpS`, `Exp`, `Temp`) that is to be used when monitoring the temperature in chemical experiments. In the schema, `ExpS` is a surrogate-valued attribute, the values of which represent specific experiments. Attributes `Exp` and `Temp` record experiment names and temperatures.

We are given the following information that allows us to characterize the observation pattern for the temperature attribute. In experiments, the temperature is sampled every ten seconds, the sampling is initiated five seconds after the sampling is initiated, and the experiments run for two hours. There is a fixed maximum delay of two seconds from when a temperature is sampled until it is actually stored in the database. In this example, the update pattern is thus closely tied to the observation pattern [JS94a].

Assuming that an experiment x1 starts at 9:00.00 a.m., its time patterns may be given as follows.

$$O_A^{\mathrm{x1}}(0) = \text{9:00.05},\ O_A^{\mathrm{x1}}(1) = \text{9:00.15},\ \ldots,\ O_A^{\mathrm{x1}}(1199) = \text{10:59.55}$$
$$U_A^{\mathrm{x1}}(0) \in [\text{9:00.05},\ \text{9:00.07}),\ \ldots,\ U_A^{\mathrm{x1}}(199) \in [\text{10:59.55},\ \text{10:59.57})$$

Note that it is generally only possible to predict the update pattern within bounds. We will return to this aspect below. □

In some temporal database applications, the observation and update patterns are identical. For example, this is the case in banking where account an balance by definition takes effect when the balance is stored in a database.

To further illustrate the notion of time patterns, we introduce two important types of time patterns, namely regular and constant.

DEFINITION 4: A *regular time pattern* is characterized by a start time t_s, a starting delay Δt_s, a regular frequency Δt_d, and an end time t_e. It is defined as follows.

$$T_{\mathrm{reg}}^s(i) = \begin{cases} t_s + i \cdot \Delta t_s & \text{if } i \in \{0, 1\} \\ t_{i-1} + \Delta t_d & \text{if } i > 1 \wedge t_{i-1} + \Delta t_d \leq t_e \\ \text{undefined} & \text{otherwise} \end{cases}$$

□

Note that the sample observation pattern above is regular with t_s = 9:00.00 a.m., $\Delta t = 10$, $\Delta t_s = 5$, and t_e = 11:00.00 a.m.

DEFINITION 5: A *constant* time pattern is a further specialization.

$$T_{\mathrm{const}}^s(i) = \begin{cases} t_s & \text{if } i = 0 \\ \text{undefined} & \text{otherwise} \end{cases}$$

Initially, an attribute with a constant update pattern has no value. Then, at time t_s, it obtains a value, and the value never changes. □

We may or may not know at schema design time the actual definitions of the observation or update patterns for an attribute. In the example, we were able to calculate $O_A^{\mathrm{x1}}(i)$ for any time

point i, but we were unable to predict the precise value of $U_A^{\mathrm{x1}}(i)$. The best we could do was to indicate bounds for $U_A^{\mathrm{x1}}(i)$. Next, we consider this issue of predictability of time patterns.

DEFINITION 6: A time pattern T is *predictable* if a function f is known that computes T. Time pattern T is *predictable within bounds* if a pair of functions l and u are defined so that for all i for which T is defined, l and u are also defined, and $l(i) \leq T(i) \leq u(i)$. □

EXAMPLE 4: Assuming that f^{x1} predicts the observation pattern for experiment x1, the pair of functions, l^{x1} and u^{x1}, shown next predicts the bounds on the update pattern for the temperature attribute.

$$l^{\mathrm{x1}}(i) = f^{\mathrm{x1}}(i)$$
$$u^{\mathrm{x1}}(i) = f^{\mathrm{x1}}(i) + 2$$ □

Finally, time patterns may be characterized by the bounds that exist between successive times in the patterns. For example, a time in a pattern may be at least some (non-zero) duration Δt^l after and at most some (larger) duration Δt^u after its predecessor time.

EXAMPLE 5: In a company, an agreement may exist between the management and the employees that salaries cannot be renegotiated within six months after they were last negotiated and that they will always be be renegotiated within a year after they were last negotiated. This illustrates a restriction of time patterns where Δt_i^l is six month and Δt_i^u is twelve months. We again emphasize that the new salary can be identical to the old salary, even if it was renegotiated. □

3.4 The Values of Individual Time-Varying Attributes

In the previous section, we introduced the notion of time patterns for describing when attributes change values. Now, we proceed by considering how attributes may encode information about the objects they describe. It is advantageous to first consider only valid-time relations. At the end of this section, we will then consider the effects of including transaction time.

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 6: Consider the two relations in Figure 2. 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.

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., [SS87, Sno87, SA+94]) have considered only the first type of relation. However, note that the relations are similar in the sense that they both record when facts are true. Due to this observation, we make no fundamental distinction between the two types of relations, but instead treat them quite similarly. □

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	$\{20,\ldots,49\}$

(a) `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	$\{55\}$

(b) `expTemp`

Figure 2: Sample Valid-time Relations

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 *interpolation functions* apply to the salary and temperature attributes of the two relations. Interpolation functions preserve the information content of the relations they are applied to and are special cases of derivation functions which are not restricted in this regard. 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.

DEFINITION 7: 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 $\mathcal{D}_D$.

$$f : \mathcal{D}_{VT} \times r(R) \hookrightarrow D$$ □

The importance of interpolation functions in data models has previously been argued convincingly by Klopprogge [KL83], Cliford [CC87, CC93] and Segev [SS87, SS93]. They should thus also be part of a design methodology.

Next, we introduce three important types of derivation functions in turn, namely stepwise-constant, discrete, and nearest-neighbor-interpolation functions.

DEFINITION 8: An attribute A is *snapshot single-valued* in a valid-time relation r if for all chronons c^v in $\mathcal{D}_{VT}$, $|\pi^{\mathrm{V}}_A(\tau^{\mathrm{V}}_{c^v}(r))| \leq 1$ (i.e., at most one A value appears in any timeslice). □

DEFINITION 9: The *stepwise constant* derivation function $f_{A\text{-sc}}$ for a an attribute A is defined for all valid-time relations r with A in its schema and A snapshot single-valued in r.

$$f_{A\text{-sc}}(c^v, r) = \begin{cases} t_1[A] \text{ where } t_1 \in r & \text{if } \exists c^v_1 \in t_1[T]\ (c^v_1 \leq c^v \wedge \\ & \quad \neg(\exists t_2 \in r\ (t_2[A] \notin \{\bot, \bot_u\} \wedge \exists c^v_2 \in t_2[T]\ (c^v_1 < c^v_2 \leq c^v)))) \\ \bot_i & \text{otherwise} \end{cases}$$

Note that the function has a value for all c^v such that there exists a tuple in r with a chronon in its timestamp that is equal to or before c^v. □

EXAMPLE 7: To illustrate this type of derivation function, let relation `empSal1` be populated with the following tuples.

EmpS	EName	Sal	T
e1	Bob	30k	{1}
e1	Bob	32k	{10}
e1	Bob	$\perp_i$	{20}
e1	Bob	36k	{30}
e1	Bob	40k	{40}
e1	Bob	$\perp_i$	{50}

We associate a step-wise constant derivation function with attribute `Sal`. Thus, $f_{\texttt{Sal-sc}}(5, \texttt{empSal1}) =$ 30k, $f_{\texttt{Sal-sc}}(10, \texttt{empSal1}) = 32\text{k}$, $f_{\texttt{Sal-sc}}(15, \texttt{empSal1}) = 32\text{k}$, and $f_{\texttt{Sal-sc}}(25, \texttt{empSal1}) = \perp_i$. For this relation, $f_{\texttt{Sal-sc}}$ is undefined for valid times before 1. Intuitively, `empSal1` with this derivation function encodes the same information as the tuples for Bob in `empSal`. □

In order to properly apply derivation functions to relations, we will add a derivation operator to the algebra. This operator is similar in spirit to Klug's aggregate formation operator [Klu82].

DEFINITION 10: The *derivation operator* ξ is applied to an instance r of a valid-time relation schema R, and is parameterized with three subscripts: a derivation function f_B, an attribute C, and a set of *partitioning attributes* $\{A_{j_1}, A_{j_2}, \ldots, A_{j_k}\}$. Derivation function f_B accepts a pair of a valid-time chronon and a relation instance with schema R as arguments and produces, if defined for the particular argument pair, a value in the domain of attribute B. Attribute B and the A_{j_i}'s are explicit attributes of R, and C is a unique attribute name not already used in R. Further, the domains of attributes B and C are the same.

The result of $\xi(r)$ is a valid-time relation with schema $R' = R \cup \{C\}$. Function f_B is applied in turn to all combinations of a valid-time chronon and a maximal subset of tuples in r with identical values for attributes $A_{j_1}, \ldots, A_{j_k}$. The result of $f_B(c^v, r')$ (r' is a maximal subset), if defined, is stored as a value of attribute C in a tuple valid at time c^v. The remaining values of the tuple are the particular values of $A_{j_1}, \ldots, A_{j_k}$ corresponding to r' and any values of the remaining attributes that are valid at time c^v; if no value is valid at time c^v for an attribute, a null values $\perp$ is used. □

EXAMPLE 8: To exemplify the use of the derivation operator, consider $\xi_{f_{\texttt{Sal-sc}}, \texttt{DSal}, \{\texttt{EName}\}}(\texttt{empSal1})$. The result is as follows.

EmpS	EName	Sal	DSal	T
e1	Bob	30k	30k	$\{1\}$
e1	Bob	$\perp$	30k	$\{2,\ldots,9\}$
e1	Bob	32k	32k	$\{10\}$
e1	Bob	$\perp$	32k	$\{11,\ldots,19\}$
e1	Bob	$\perp_i$	$\perp_i$	$\{20\}$
e1	Bob	$\perp$	$\perp_i$	$\{21,\ldots,29\}$
e1	Bob	36k	36k	$\{30\}$
e1	Bob	$\perp$	36k	$\{31,\ldots,39\}$
e1	Bob	40k	40k	$\{40\}$
e1	Bob	$\perp$	40k	$\{41,\ldots,49\}$
e1	Bob	$\perp_i$	$\perp_i$	$\{50\}$
e1	Bob	$\perp$	$\perp_i$	$\{51,\ldots,c_k^v\}$

In the result, c_k^v is the largest possible valid-time chronon. We can now precisely describe the sense in which `empSal1` with derivation function $f_{\texttt{Sal-sc}}$ and `empSal` record the same information.

$$\delta_{\texttt{DSal}\rightarrow\texttt{Sal}}(\pi^{\mathrm{V}}_{\texttt{EmpS},\texttt{EName},\texttt{DSal}}(\sigma^{\mathrm{V}}_{\texttt{DSal}\neq\perp_i}(\xi_{f_{\texttt{Sal}-\texttt{sc}},\texttt{DSal},\{\texttt{EName}\}}(\texttt{empSal1})))) \equiv \sigma^{\mathrm{V}}_{\texttt{EName}='\texttt{Bob}'}(\texttt{empSal})$$

Here, $\delta_{A\rightarrow B}(r)$ renames an attribute A in the schema of r to B [KS91]. □

Most existing data models implicitly assume that only one (kind of) derivation function is of relevance to the attributes of the base relations representable in the model, namely the *discrete* interpolation function, defined as follows.

Definition 11: The *discrete* derivation function $f_{A\text{-d}}$ for an attribute A takes as arguments a valid-time chronon c^v and a valid-time relation r with A in its schema R and A snapshot single-valued in r.

$$f_{A\text{-d}}(c^v, r) = \begin{cases} t[A] \text{ where } t \in \tau^{\mathrm{V}}_{c^v}(r) & \text{if } |\tau^{\mathrm{V}}_{c^v}(r)| \neq 0 \\ \perp & \text{otherwise} \end{cases}$$ □

Thus, if there exists an A value in r that is valid at c^v, that value is the result; otherwise, the result is $\perp$.

Discrete interpolation functions may be used for precisely characterizing those derivation functions that are also interpolation functions. Intuitively, an interpolation function is a derivation function that does not contradict information in its argument relations.

Definition 12: Let f_A be a derivation function and let $f_{A\text{-d}}$ be the discrete derivation function with the same signature. Then f_A is an *interpolation function* if for all pairs of a valid time c^v and an argument relation r for which $f_{A\text{-d}}(c^v, r) \neq \perp$, the condition $f_A(c^v, r) = f_{A\text{-d}}(c^v, r)$ is satisfied. □

It follows that interpolation functions are special cases of derivation functions. Note also that the discrete and step-wise constant derivation functions are interpolation functions.

Next, we define a simple interpolation function which is appropriate for the `Temp` attribute of `expTemp` in Figure 2(b). As usual, it applies to a particular attribute, e.g., A. When applied to a valid time c^v and a relation r, it returns a value interpolated from the two A values in r that are valid most recently before and after c^v. This may be stated precisely as follows.

Definition 13: The *nearest-neighbor-interpolation* derivation function $f_{A\text{-nn}}$ for a (numeric-valued) attribute A is defined for all valid-time relations r with A in its schema with A snapshot

single-valued in r; it is defined for all valid times c^v for which there exist two tuples in r, one with a valid time in its timestamp that is at or before c^v and one with a valid time that is at or after c^v.

$$f_{A\text{-nn}}(c^v, r) = \ a_x + (c^v - c^v_x)(a_y - a_x)/(c^v_y - c^v_x)$$

where

$$\begin{aligned}
a_x &= \{t[A] \mid t \in r \ \wedge \ t[A] \notin \{\bot_i, \bot_u, \bot\} \ \wedge \ \exists c \in t[\mathrm{T}]\ (c \leq c^v \ \wedge \\
&\qquad \neg\exists t' \in r\ (t'[A] \notin \{\bot_i, \bot_u, \bot\} \ \wedge \ \exists c' \in t'[\mathrm{T}]\ (c < c' \leq c^v)))\} \\
a_y &= \{t[A] \mid t \in r \wedge t[A] \notin \{\bot_i, \bot_u, \bot\} \ \wedge \ \exists c \in t[\mathrm{T}]\ (c \geq c^v \ \wedge \\
&\qquad \neg\exists t' \in r\ (t'[A] \notin \{\bot_i, \bot_u, \bot\} \ \wedge \ \exists c' \in t'[\mathrm{T}]\ (c^v \leq c' < c)))\} \\
c^v_x &= \{\max(t[\mathrm{T}] \cap \{c^v_1, \ldots, c^v\}) \mid t \in r \ \wedge \ t[A] \notin \{\bot_i, \bot_u, \bot\} \ \wedge \ \exists c \in t[\mathrm{T}]\ (c \leq c^v))\} \\
c^v_y &= \{\min(t[\mathrm{T}] \cap \{c^v, \ldots, c^v_k\}) \mid t \in r \ \wedge \ t[A] \notin \{\bot_i, \bot_u, \bot\} \ \wedge \ \exists c \in t[\mathrm{T}]\ (c \geq c^v))\}
\end{aligned}$$

□

Next, we introduce derivation functions with error bounds. This type of derivation function produces upper and lower bounds for each derived value. The bounds may be used for indicating how much the real value is expected to deviate from the derived value.

DEFINITION 14: A *derivation function with error bounds* f is a partial function from the domains of valid times and relation instances with some fixed schema to a triplet of value domains,

$$f : \mathcal{D}_{VT} \times r(R) \hookrightarrow D \times D \times D,$$

where R is a valid-time relation schema, $r(R)$ is the domain of instances of R, and D is in the universal set of domains $\mathcal{D}_D$. □

With this concept, we may describe the concepts of stability and non-divergence of derivation functions. As these concepts are not essential for database design purposes, we focus solely on the intuition behind the concepts.

EXAMPLE 9: Consider the relation instance `expTemp` in Figure 2(b). This instance stems from the sampling of a temperature sensor in a chemical experiment. For the purpose of this example, assume that there is a maximum delay of five time units between the measurement of temperature values and when they are inserted in `expTemp`. Assume also that the only updates to `expTemp` are such insertions. Let the current time be 66.

Now consider the following application of some derivation function, $f_{\texttt{Temp}}$ for the temperature attribute.

$$\xi_{f_{\texttt{Temp}},\texttt{DTemp},\{\texttt{Exp}\}}(\texttt{expTemp})$$

We know that no temperatures with a valid time before time 61 are deleted or inserted. Thus, we may expect that applications of $f_{\texttt{Temp}}$ to valid times before this time will *from now on always* yield the same value, i.e., are *stable*. This is true in particular for the discrete and step-wise constant interpolation functions. For the nearest-neighbor interpolation function this is also true because the function is undefined for times where earlier and later neighbor temperatures are not available.

For time arguments later than time 61, it is more difficult to provide stability. However, the discrete interpolation function is stable also for times after time 61 because it only returns a value for times where a value is already recorded. The two other interpolation functions are not. □

EXAMPLE 10: In continuation of the previous example, assume that at time 66, a value is derived for time 64 and `expTemp` using $f_{\texttt{Temp-nn}}$. Then, at time 68 a value is again derived for time

64 and `expTemp`. The two values may be different because a temperature valid at time 64 may have been stored at time 67.

However, it may still be the case that derived results improve in accuracy as time progresses. This notion of derivation functions being *non-diverging* may be captured using derivation functions with error bounds. Specifically, if derived values from successive applications of the derivation function have error bounds that do not increase, a derivation function is non-diverging. □

For transaction time, only two interpolation functions appear to be important. The discrete interpolation function is to be used if all the times a fact is current in the database are stored with the fact. The stepwise-constant interpolation function is used if facts are stamped with their insertion times only. Which interpolation function is appropriate generally depends on the adopted temporal data model, and that function is then built into the model.

3.5 Temporal Functional Dependencies

As in design of snapshot relational databases, dependencies are also important during temporal relational database design. As background material, we first state the notion of a functional dependency (see, e.g, [Ull88]) for snapshot relations and then review a previously proposed generalization [JSS92] that extends functional dependencies to temporal relations. Following that, we generalize the notion of temporal functional dependency and discuss some specializations of the general definition, one of which is the existing temporal functional dependency. Finally, we propose the notion of a strong temporal functional dependency and explore its properties.

3.5.1 Generalizing Functional Dependencies to Temporal Databases

Functional dependencies are defined as follows.

DEFINITION 15: Let a relation schema R be defined as $R = (A_1, A_2, \ldots, A_n)$, and let X and Y be sets of attributes of R. The set Y is *functionally dependent* on the set X, denoted $X \rightarrow Y$, if for all meaningful instances r of R,

$$\forall s_1, s_2 \in r(s_1[X] = s_2[X] \Rightarrow s_1[Y] = s_2[Y]).$$

If $X \rightarrow Y$, we say that X *determines* Y. □

The following generalization of a functional dependency to temporal relations perceives a temporal relation as a set of snapshots, and it requires each such snapshot to satisfy the dependency [JSS92].

DEFINITION 16: Let X and Y be sets of non-timestamp attributes of a bitemporal relation schema R. A *temporal functional dependency*, denoted $X \overset{\mathrm{T}}{\rightarrow} Y$, exists on R if for all meaningful instance r of R,

$$\forall c^v, c^t \; \forall s_1, s_2 \in \tau^{\mathrm{V}}_{c^v}(\rho^{\mathrm{B}}_{c^t}(r)) \; (s_1[X] = s_2[X] \Rightarrow s_1[Y] = s_2[Y]).$$ □

For example, the instance `empSal` in Figure 2(a) satisfies the dependency `EName` $\overset{\mathrm{T}}{\rightarrow}$ `Sal`.

DEFINITION 17: Analogously to how keys are defined in the snapshot relational model, the explicit attributes X of a temporal relation schema R is a *(temporal) key* if $X \overset{\mathrm{T}}{\rightarrow} R$. □

Segev and Shoshani define, in their Temporal Data Model, a normal form, 1TNF, for valid-time relations [SS88b] that is related to the notion of temporal keys. To understand this normal

form, we need to first describe their data model and the special variant of the timeslice operator employed there.

Valid-time relation schemas have a distinguished, so-called *surrogate*, attribute. The special timeslice operator relies on the presence of the surrogate attribute. It takes a valid-time relation and a time value as arguments and returns, for each surrogate value, all the values of each time-varying attribute that are valid at the time given as argument. Thus, the result contains precisely one tuple per surrogate, valued with at least one time-varying attribute value, valid at the time argument. As another consequence, time varying attributes may be set-valued, leading to a non-1NF result relation.

DEFINITION 18: For a relation to be in *first temporal normal form* (1TNF), "a time-slice at point t has to result in a standard 1NF-relation." [[SS88b], p. 17] □

This normal form can be expressed in our terminology as "the specified surrogate attribute is a (temporal) key."

We may also use temporal functional dependencies to rephrase the criterion that an attribute of a temporal relation be snapshot single-valued, which was imposed on, e.g., the stepwise constant interpolation function. When a derivation function f_B for an attribute B is applied to a temporal relation r using the ξ operator, this criterion may be stated as r must satisfy $A_1 \ldots A_n \overset{\mathrm{T}}{\rightarrow} B$ where $A_1 \ldots A_n$ are the partioning attributes. For example, for the query $\xi_{f_{\mathtt{Sal-sc}},\mathtt{DSal},\{\mathtt{EName}\}}(\mathtt{empSal1})$ to be defined, the dependency $\mathtt{EName} \overset{\mathrm{T}}{\rightarrow} \mathtt{Sal}$ must be satisfied by `empSal1`.

3.5.2 Generalized Temporal Functional Dependencies

In the previous definition of temporal functional dependency, each constituent snapshot of a temporal relation must satisfy the corresponding snapshot functional dependency for the temporal relation to satisfy the temporal functional dependency. We may generalize that definition by parameterizing the dependency with a *subset* of constituent snapshots that must satisfy the snapshot dependency for the temporal relation to satisfy the parameterized temporal functional dependency.

The resulting generalized dependencies may capture the temporal semantics of a database schema more presicely than may the standard temporal dependency.

DEFINITION 19: Let X and Y be sets of non-timestamp attributes of a temporal relation schema R. A *parameterized temporal functional dependency*, denoted $X \overset{\mathrm{T}}{\rightarrow}^{[P]} Y$, exists on R if for all meaningful instances of r of R,

$$\forall c^v, c^t \; \forall s_1, s_2 \in \tau^{\mathrm{V}}_{c^v}(\rho^{\mathrm{B}}_{c^t}(r)) \; ((P(c^v, c^t) \wedge s_1[X] = s_2[X]) \Rightarrow s_1[Y] = s_2[Y]).$$ □

With this more general definition, it is possible to define a range of different temporal functional dependencies by specifying the predicate P. Examples of the predicate P include the following.

(1) $P_1(c^t, c^v) \equiv$ True. This yields the temporal functional dependency as first defined, and is relevant to *general* temporal relations for which there is no stated relationship between valid and transaction time [JS94a]. Such temporal dependencies have been termed *intraelement integrity constraints* [Böh94].

(2) $P_2(c^t, c^v) \equiv c^v \leq c^t$. With this predicate, only snapshots that concern a past state of reality, relatively to the time the snapshot was current, are required to satisfy the snapshot dependency. For *retroactive* temporal relations, in which the stored information lags the modeled reality, this predicate is equivalent to the generally more restrictive predicate above.

(3) $P_3(c^t, c^v) \equiv c^t = c^v$. Here, only snapshots that are about the current state of reality, relative to the snapshot was current in the database, are considered. This matches *degenerate* relations in which the transaction time always equals the valid time, i.e., updates occur instantly.

(4) $P_4(c^t, c^v) \equiv c^t = now \wedge c^t = now$. Here, only the snapshot about the current state of reality in the current state of the database is considered.

(5) $P_5(c^t, c^v) \equiv c^t \in [1, 10]$. This predicate specifies absolute limits on the states of the database within which all snapshots must satisfy the corresponding snapshot dependency for the temporal relation to satisfy the parameterized temporal functional dependency.

(6) $P_6(c^t, c^v) \equiv c^v \in [1, 10]$. Here, the restriction is that states recording information about reality in the specified interval are the only ones considered.

EXAMPLE 11: To see the differences between the sample predicates (1)–(6), consider the sample relation instances in Figure 3.

EmpS	EName	Dept	T
e1	Bob	Ship	$\{(20,1),\ldots,(20,10),\ldots,(30,1),\ldots,(30,10)\}$
e1	Bob	Load	$\{(20,1),\ldots,(20,10),\ldots,(30,1),\ldots,(30,10)\}$

(a) `empDep1`

EmpS	EName	Dept	T
e1	Bob	Ship	$\{(1,20),\ldots,(1,30),\ldots,(10,20),\ldots,(10,30)\}$
e1	Bob	Load	$\{(1,20),\ldots,(1,30),\ldots,(10,20),\ldots,(10,30)\}$

(b) `empDep2`

Figure 3: Sample Bitemporal Relations

With **(1)** as the predicate of a temporal dependency, neither `empDep1` nor `empDep2` satisfy the dependency `EName` $\overset{\mathrm{T}\,[P_1]}{\rightarrow}$ `Dept`. However, `empDep2` does satisfy the dependency if the predicate of **(2)** is adopted; `empDep1` still does not. If predicate **(3)** (and thus the more restrictive **(4)**) is adopted, both instances satisfy the dependency. Finally, `empDep1` satisfies **(5)**, but not **(6)**. The opposite holds for `empDep2`. □

3.5.3 Strong Temporal Functional Dependency

The temporal dependencies we have seen so far apply snapshot dependencies to individual snapshots in isolation. Thus, these dependencies are not capable of capturing the relative variation over time of attribute values. In order to address this need, we introduce the notion of a strong temporal functional dependency.

So, while a (regular or general) temporal dependency holds if the corresponding conventional dependency holds for each snapshot in isolation, we now "bundle" tuples of certain snapshots and require the corresponding snapshot dependency to hold for each "bundle" in isolation. A "bundle" is defined to contain all tuples in all valid timeslices of the result obtained from applying a single transaction timeslice operation to a meaningful bitemporal database instance of the schema under consideration. This is stated more precisely below.

DEFINITION 20: Let X and Y be sets of non-timestamp attributes of a bitemporal relation

schema R. A *strong temporal functional dependency*, denoted $X \overset{\text{Str}}{\rightarrow} Y$, exists on R if for all meaningful instances r of R,

$$\forall c^t, c_x^v, c_y^v \; \forall s_1 \in \tau_{c_x^v}^{\text{V}}(\rho_{c^t}^{\text{B}}(r)) \; \forall s_2 \in \tau_{c_y^v}^{\text{V}}(\rho_{c^t}^{\text{B}}(r)) \; (s_1[X] = s_2[X] \Rightarrow s_1[Y] = s_2[Y]) \; . \qquad \square$$

Consider the relation instance `empSal` (in Figure 2). While we have seen that it does satisfy the dependency `EName` $\overset{\text{T}}{\rightarrow}$ `Sal`, it does not satisfy the dependency `EName` $\overset{\text{Str}}{\rightarrow}$ `Sal`. For example, (e1, Bob, 30k) and (e1, Bob, 32k) are in valid timeslices at times 5 and 15, respectively, which violates the dependency. The dependency `EmpS` $\overset{\text{Str}}{\rightarrow}$ `EName`, however, holds for `empSal`. It might not hold for the schema of `empSal`. Specifically, if a person may change name, e.g., person e1 may change name from Bob to Rob, the dependency is not satisfied.

Strong temporal dependencies are useful because they have a practical and intuitive interpretation. Specifically, if $X \overset{\text{Str}}{\rightarrow} Y$ holds on a relation schema, this means that Y does not vary with respect to X. For example, the observation that employees never change salary while remaining in a department may be stated as `Dept` $\overset{\text{Str}}{\rightarrow}$ `Sal`.

Strong temporal dependency provides a basis for giving precise definitions to other useful and intuitive concepts, such as time-invariant attributes, time-invariant key, and value-synchronous attributes.

DEFINITION 21: Let X be a set of non-timestamp attributes of a bitemporal relation schema R with surrogate attribute S. Then X is said to be *time invariant* if $S \overset{\text{Str}}{\rightarrow} X$. $\square$

This definition reflects the assumption that surrogates uniquely reflect the identity of the objects modeled by the relation. Since different objects have different surrogates and the same object always has the same surrogate, it is natural to term attributes *time invariant* if their values for an object never change. In the `empSal` instance in Figure 2, attribute `EName` is time invariant, but attribute `Sal` is not. By combining standard temporal dependency and strong temporal dependency, the notion of a time-invariant key (which had previously been defined informally and with a different meaning [NA89]) results.

DEFINITION 22: Let X be a set of non-timestamp attributes of a bitemporal relation schema R with surrogate attribute S. Then X is termed a *time-invariant key (TIK)* if $S \overset{\text{Str}}{\rightarrow} X$ and $X \overset{\text{T}}{\rightarrow} R$. $\square$

The first requirement to attributes X is that they be time invariant. The second is that they be a temporal key. In combination, the requirements amount to saying that X is a key with values that do not change (with respect to the surrogate attribute). In the `empSal` instance, attribute `EName` is a time-invariant key. Indeed, it would be not be inconsistent with our perception of reality to specify `EName` as a time-invariant key for the schema of `empSal`. In such situations, the surrogate attribute `EmpS`, used here to determine that `Emp` is a time-invariant key, may be removed from the schema of `empSal` with no ill effect.

Let's now investigate a few other implications of temporal dependencies. In a strong temporal dependency $X \overset{\text{Str}}{\rightarrow} Y$, attributes X may vary more often than attributes Y, but X must change when Y changes.

DEFINITION 23: Let X and Y be sets of non-timestamp attributes of a bitemporal relation schema R. A *strong temporal equivalence*, denoted $X \overset{\text{Str}}{\leftrightarrow} Y$, exists on R if $X \overset{\text{Str}}{\rightarrow} Y$ and $Y \overset{\text{Str}}{\rightarrow} X$. $\square$

Intuitively, $X \overset{\text{Str}}{\leftrightarrow} Y$ means that the sets of attributes X and Y change values simultaneously. When $X \overset{\text{Str}}{\leftrightarrow} Y$, we also say that X and Y are *mutually value synchronous*. This definition appears to be a formalization of Navathe and Ahmed's [NA89] informal notion of synchronous attributes. We will

return to this issue in Section 4.1.2.

3.6 Attribute Semantics Template

As a summary to the present section and a precursor to the next, we review the concepts just introduced and briefly indicate how they may be used for capturing the semantics of time-varying attributes during database design.

Identify entity types and represent them with surrogate attributes. The real-world objects (or entities) that the attributes of the database describe are represented with surrogate attributes.

Describe lifespans. For each relation schema, desecribe the lifespans of the attributes. The use of the descriptions is covered in the next section where the lifespan decomposition rule is introduced.

Determine observation and update patterns. For each relation schema, indicate which attributes are synchronous, i.e., share observation and update patterns. This information is used in conjunction with the synchronous decomposition rule, also covered in the next section.

For each attribute, indicate its appropriate interpolation function(s). The functions concern interpolation in valid-time, and there is generally one function per attribute, e.g., the discrete interpolation function.

Specify temporal functional dependencies. These provide the basis for applying the temporal extensions of the conventional normal forms for schema decomposition. This is covered by standard database design approaches, all of which apply directly here.

Specify strong temporal functional dependencies. Together with the temporal functional dependencies, these dependencies allow the designer to identify time-invariant keys, which may play the role of surrogates that can then be eliminated.

4 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.

4.1 Logical-Design Guidelines

Two important goals of logical database design are to design a database schema (a) that does not require the use of inapplicable nulls, and (b) that avoids repetition of the same information or facts. 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.

4.1.1 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 24: Let a relation schema $R = (S, A_1, \ldots, A_n \mid T)$ be given where S is surrogate valued. Two attributes A_i and A_j, $i, j = 1, \ldots, n$, 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,

$$\forall s \in \text{dom}(S)\ (\text{ls}(r, A_i, s) = \text{ls}(r, A_j, s)).$$ □

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 25: 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 26: (*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. Initially, observe that a database schema that obeys the temporal normal forms may still require inapplicable nulls in its instances. To exemplify, consider the `Emp` schema. 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.

Next, the temporal normal forms do tend to aid in eliminating the need for inapplicable nulls, as the following two examples show. First, replace the attribute `MgrSince` in schema `Emp` with an attribute `DBudget` that records the budget for a department. The resulting schema is not lifespan homogeneous—attributes `Salary` and `DBudget` are not lifespan equal (a department could have a budget when a particular employee wasn't in that department). The new schema has a dependency $\texttt{Dept} \xrightarrow{\text{T}} \texttt{DBudget}$, meaning that the schema is not in temporal BCNF. To achieve temporal BCNF, the schema is decomposed by creating a new schema for `Dept` and `Budget` and eliminating `Budget` from the original relation schema. The resulting schemas are in temporal BCNF and are also lifespan homogeneous.

Second, replace the attribute `MgrSince` in schema `Emp` with an attribute `Skill` that records the skill(s) of an employee. Again, the resulting schema is not lifespan homogeneous—attributes `Salary` and `Skill` are not lifespan equal. The new schema has a non-trivial multi-valued dependency $\texttt{EmpS} \xrightarrow{\text{T}}\!\!\!\!\!\rightarrow \texttt{Skill}$, meaning that the schema is not in temporal 4NF. To achieve temporal 4NF, `Skill` is eliminated from the schema, and a new relation schema is added, `Emp3 = (EmpS, SKill)`. The resulting schemas are in temporal 4NF and are also lifespan homogeneous.

4.1.2 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 the type of redundancy the rule addresses. Then we define the notion of synchronous attributes, which leads to a definition of synchronous schemas that avoid this redundancy. Following this, the decomposition rule is stated. 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 12: Consider the relation instance `empDepSal` below, recording departments and salaries for employees.

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	$\{43,\ldots,49\}$
e1	B	50k	$\{50,\ldots,59\}$

The schema for the relation is in temporal BCNF, with the surrogate-valued attribute `EmpS` being the only minimal key and no other dependencies. Yet, it may be observed that the salaries 30k and

50k are repeated once in the instance. Similarly, the departments A and B are repeated once and four times, respectively. These repetitions are due to attributes `Dept` and `Salary` having different observation patterns. Specifically, the instance is consistent with the patterns shown below.

$$O^{\texttt{e1}}_{\texttt{Dept}} =< [0 \mapsto 1], [1 \mapsto 6], [2 \mapsto 43], [3 \mapsto 50], [4 \mapsto 60] >$$
$$O^{\texttt{e1}}_{\texttt{Salary}} =< [0 \mapsto 1], [1 \mapsto 10], [2 \mapsto 15], [3 \mapsto 28], [4 \mapsto 43], [5 \mapsto 60] >$$

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

Note that patterns with additional time points are also consistent with the instance. For example, the salary may have been updated to become 50k at time 55. □

This type of redundancy has been mentioned in the past by several researchers (see, e.g., [CT85, CV85, Gad88, GY91]). Most often, it has been used for motivating a non-first normal form data modeling approach where time is associated with with attribute values rather than with tuples, because that approach avoids the redundancy. The `empDepSal` instance is given next in a typical non-first normal form format.

EmpS		Dept		Salary	
e1	$\{1,\ldots,59\}$	A	$\{1,\ldots,5,43,\ldots,49\}$	30k	$\{1,\ldots,9\}$
		B	$\{6,\ldots,43,50,\ldots,59\}$	32k	$\{10,\ldots,14\}$
				36k	$\{15,\ldots,27\}$
				40k	$\{28,\ldots,42\}$
				50k	$\{43,\ldots,59\}$

To characterize 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 27: 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, $i, j = 1, \ldots, n$, 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,

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

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.

DEFINITION 28: Define relation schema $R = (S, A_1, \ldots, A_n \mid \mathrm{T})$ where S is surrogate valued. Relation R is *synchronous* if

$$\forall A_i, A_j \in R \; (A_i \overset{\mathrm{S}}{=}_S A_j). \qquad \square$$

This definition provides the basis for stating the Synchronous Decomposition Rule.

DEFINITION 29: (*Synchronous Decomposition Rule*) To avoid repetition of attribute values in temporal relations, decompose relation schemas until they are synchronous. □

Above, we defined synchronism among attributes. We have earlier used temporal functional dependencies for defining value synchronism among attributes. While somewhat similar, these two types of synchronism are different. To understand their relationship, assume that $A_i \overset{\text{S}}{=}_S A_j$. This statement concerns the observation patterns of the two attributes—it does not concern the values of the attributes. Thus, this relation instance is possible.

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

When, at time 6, A_i and A_j were observed, their values were a_1 and a_3, respectively.

Next, assume that A_i and A_j are mutually value synchronous, i.e., $A_i \overset{\text{Str}}{\leftrightarrow} A_j$. This means that the values of the two attributes track each other. Thus, if one attribute changes its value from one tuple to another, the other attribute must also change its value from the one tuple to the other. Consequently, the instance above violates value synchronism, and the instance below respects value synchronism.

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

This instance, in turn, is consistent with the following observation patterns, which are not identical.

$$O^s_{A_i} =< [0 \mapsto 1], [1 \mapsto 6], [2 \mapsto 11] >$$
$$O^s_{A_j} =< [0 \mapsto 1], [1 \mapsto 6], [2 \mapsto 8], [3 \mapsto 11] >$$

The pattern for $O^s_{A_j}$ indicates that A_j was observed on time 8, and the instance indicates that A_j did not change value at that time, so the observed value was the same as the existing value. Put together, this means that neither of the two notions, $\overset{\text{Str}}{\leftrightarrow}$ and $\overset{\text{S}}{=}$, is subsumed by the other. The former notion is concerned with the values of attributes while the latter is concerned with the observation patterns, aspects we have previously characterized as orthogonal.

In the context of a data model where tuples are timestamped with single time intervals, Navathe and Ahmed [NA89] have previously defined a temporal dependency and an associated normal form that is related to synchronism. These concepts are defined next.

DEFINITION 30: There exists a *temporal dependency* between two attributes, A_i and A_j, in a relation schema $R = (A_1, A_2, \ldots, A_n, T_s, T_e)$ if there exists an instance r of R containing two distinct tuples, t and t', that satisfy each of the following three properties.

1. $t[K] = t'[K]$ where K is a chosen temporal key.
2. $t[T_e] = t'[T_s] - 1 \vee t'[T_e] = t[T_s] - 1$.
3. $t[A_i] = t'[A_i]$ `XOR` $t[A_j] = t'[A_j]$. □

This definition captures a kind of asynchronism among attributes: If it is possible for two tuples with the same key value (e.g., S value) to have the same A_i values and different A_j values, or

vice versa, then A_i and A_j are temporally dependent. The normal form defined next states that relation schemas should not contain temporally dependent attributes.

DEFINITION 31: A valid-time relation schema "is in *time normal form* (TNF) if and only if it is in [snapshot] BCNF and there exists no temporal dependency among its time varying attributes."[[NA89], p. 157] □

As an example, the `empDepSal` relation shown above is not in TNF, because there exists a temporal dependency between `Dept` and `Salary` (in fact, many tuples represent the value of one attribute changing while the value of the other remained as before).

It may be verified that this concept of temporal dependency (and its inverse) is different from the two notions of synchronism introduced in this paper. Further, we find these temporal dependencies hard to identify during database design. At design time, for what schemas can we guarantee that there will never be an instance with two tuples that satisfy the three properties stated in the definition, and what is the intuition behind this requirement? It appears that, unlike our design rules, the imposition of TNF 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.

We now consider the positioning of the synchronous decomposition rule with respect to logical versus physical database design. In this paper, we have made a clear distinction between logical-level relations and their physical representation in a temporal DBMS (see Section 2.4). Surely, the redundancy that may be detected using the synchronism concepts provided in this paper is important when *storing* temporal relations. At the same time, it is our contention that the type of redundancy captured in this section is of little consequence for the querying of logical-level relations using the query language associated with those relation, namely TSQL2. In TSQL2, it is possible to declare variables in the From clause that range over groups of tuples [SA+94]. To illustrate this, consider the instance `empDepSal` that contains asynchronous attributes. The following From clause,

```
FROM empDepSal(EmpS, Dept) AS empDept, empDepSal(EmpS, Salary) AS empSalary
```

yields variables `empDept` and `empSalary` that range over the following two sets of tuples, respectively (the blank attributes are inaccessible through the variables).

EmpS	Dept		T
e1	A		$\{1,\ldots,5,\ 43,\ldots,49\}$
e1	B		$\{6,\ldots 42,\ 50,\ldots,59\}$

EmpS		Salary	T
e1		30k	$\{1,\ldots,9\}$
e1		32k	$\{10,\ldots,14\}$
e1		36k	$\{15,\ldots,27\}$
e1		40k	$\{28,\ldots,42\}$
e1		50k	$\{43,\ldots,59\}$

Note that the coalescing implied by the From clause isolates the times in which the values change. With this facility, we believe that asynchronous attributes in a relation present no special problems when posing queries. In conclusion, synchronism and the associated type of redundancy as identified in this section is a non-issue at the logical level. Synchronism may affect performance (positively or negatively), but it does not affect correctness. This is in marked contrast to other contexts, e.g., Navathe and Ahmed's, in which synchronism is an issue also for logical design.

Several claims have been made in the past about synchronism and database design.

Initially, observe that we have argued that the synchronous decomposition rule does not apply to logical-level database design. In our conceptual data model, which is a tuple-timestamped first

normal form model, it is thus not necessary (and it is probably not even desirable) to separate attributes in logical-level temporal relations on the sole basis of asynchronism.

The need for synchronism at the logical level has previously been claimed to make normal forms and dependency theory inapplicable (e.g., [CV85]). The argument is that few attributes are synchronous, meaning that relation schemas must be maximally decomposed, which leaves other normalization concepts irrelevant. This claim then does not apply to our data model.

It has also been claimed that the need for separating asynchronous attributes is inherent to tuple-timestamped data models (e.g., '[...] the notion of first normal form has been applied too literally to temporal datbases [...] Our departure from this "hangup" brings temporal databases within the framework of classical relational theory' [CV85, p. 55]). We do not feel that this claim applies to our data model.

For completeness, it should be mentioned that while the decomposition rule and associated concepts presented in this section have concerned valid time, a similar decomposition rule and associated concepts that concern transaction time, employing update patterns rather than observation patterns, may also be defined. For brevity, we omit these concepts.

4.2 Implications for View Design

The only concept summarized in Section 3.6 and not covered so far is interpolation 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 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 add additional or remove or modify existing interpolation functions.

The view mechanism is an ideal solution that maintains the 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. The designer subsequently defines a view for each such set. This view definition could utilize the derivation operator introduced in Section 3.4. Although interpolation functions have previously been studied, we believe they have never before been associated with the view mechanism.

5 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, derivation functions that compute new attribute values from stored ones, and new temporal functional dependencies.

The paper subsequently showed how surrogates, lifespans, and dependencies play a 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.

In previous work we have extended conventional dependency theory to temporal relations. This led to temporal normals forms that closely track their non-temporal counterparts, but these normal forms were atemporal in nature and did not fully exploit the temporal semantics of data that is captured by temporal relations. This paper complements that work by providing concepts for capturing the temporal semantics and then exploiting it during temporal database 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, an articulate methodology for physical design that takes into account both different specific storage formats and primary and secondary indexing techniques still remains.

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

[Ahm92] R. Ahmed. Personal Communication, March 1992.

[AD93] P. Atzeni and V. De Antonellis. *Relational Database Theory.* The Benjamin/Cummings Publishing Company, Inc., 1993.

[BZ82] J. Ben-Zvi. *The Time Relational Model.* PhD thesis, Computer Science Department, UCLA, 1982.

[Böh94] M. Böhlen. Valid Time Integrity Constraints. Technical Report, University of Arizona, Department of Computer Science, TR 94-30, November, 1994. 22 pages.

[CC87] J. Clifford and A. Croker. The Historical Relational Data Model (HRDM) and Algebra Based on Lifespans. In *Proceedings of the International Conference on Data Engineering*, pages 528–537, Los Angeles, CA, February 1987.

[CC93] J. Clifford and A. Croker. The Historical Relational Data Model (HRDM) Revisited. In *Temporal Databases: Theory, Design, and Implementation*, pages 6–27, Benjamin/Cummings, 1993.

[CT85] J. Clifford and A. U. Tansel. On an Algebra for Historical Relational Databases: Two Views. In *Proceedings of ACM SIGMOD*, pages 247–265, Austin, TX, May 1985.

[EWK93] R. Elmasri, G. Wuu, and V. Kouramajian. A Temporal Model and Query Language for EER Databases. In *Temporal Databases: Theory, Design, and Implementation*, pages 212–229, Benjamin/Cummings, 1993.

[Gad88] S. K. Gadia. A Homogeneous Relational Model and Query Languages for Temporal Databases. *ACM Transactions on Database Systems*, 13(4):418–448, December 1988.

[CV85] S. K. Gadia and J. H. Vaishnav. A Query Language for a Homogeneous Temporal Database. In *Proceedings of ACM PODS'85*, pages 51–56, March 1985.

[GY91] S. K. Gadia. and C.-S. Yeung. Inadequacy of Interval Timestamps in Temporal Databases. *Information Sciences*, 54:1–11, 1991.

[HOT76] P. Hall, J. Owlett, and S. J. P. Todd. Relations and Entities. In G. M. Nijssen, editor, *Modelling in Data Base Management Systems*, pages 201–220. North-Holland, 1976.

[JCE+94] C. S. Jensen, J. Clifford, R. Elmasri, S. K. Gadia, P. Hayes, and S. Jajodia [eds]. A Glossary of Temporal Database Concepts. *ACM SIGMOD Record*, 23(1):52–64, March 1994.

[JS94a] C. S. Jensen and R. Snodgrass. Temporal Specialization and Generalization. *IEEE Transaction on Knowledge and Data Engineering*, 6(6):954–974, 1994.

[JS94b] C. S. Jensen and R. T. Snodgrass. The Surrogate Data Type in TSQL2. Commentary, TSQL2 Design Committee, September 1994.

[JSS92] 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.

[JSS94] C. Jensen, M. Soo, and R. T. Snodgrass. Unifying Temporal Models via a Conceptual Model. *Information Systems*, 19(7):513–547, 1994.

[KL83] M. R. Klopprogge and P. C. Lockemann. Modelling Information Preserving Databases: Consequences of the Concept of Time. In *Proceedings of VLDB'83*, pages 399–416, 1983.

[Klu82] A. Klug. Equivalence of Relational Algebra And Relational Calculus Query Languages Having Aggregate Functions. *Journal of The ACM*, 29(3):699–717, July 1982.

[KS91] H. F. Korth and A. Silberschatz. *Database System Concepts*. McGraw-Hill Advanced Computer Science Series. McGraw-Hill Book Company, second edition, 1991.

[Lor91] N. A. Lorentzos. Management of Intervals and Temporal Data in the Relational Model. Technical Report 49, Agricultural University of Athens, 1991.

[NA89] S. B. Navathe and R. Ahmed. A Temporal Relational Model and a Query Language. *Information Sciences*, 49:147–175, 1989.

[SA+94] R. T. Snodgrass, I. Ahn, G. Ariav, D. S. Batory, J. Clifford, C. E. Dyreson, R. Elmasri, F. Grandi, C. S. Jensen, W. Kafer, N. Kline, K. Kulkanri, T. Y. C. Leung, N. Lorentzos, J. F. Roddick, A. Segev, M. D. Soo, and S. M. Sripada. TSQL2 Language Specification. *SIGMOD Record*, 23(1):65–86, March 1994.

[SJS94] M. D. Soo, C. S. Jensen, and R. T. Snodgrass. An Algebra for TSQL2. Commentary, TSQL2 Design Committe, September 1994.

[Sno87] R. T. Snodgrass. The Temporal Query Language TQUEL. *ACM Transactions on Database Systems*, 12(2):247–298, June 1987.

[SS87] A. Segev and A. Shoshani. Logical Modeling of Temporal Data. In *Proceedings of the ACM SIGMOD '87*, pages 454–466, May 1987.

[SS88a] A. Segev and A. Shoshani. Modeling Temporal Semantics. In *Temporal Aspects in Information Systems*, pages 47–58. North-Holland, 1988.

[SS88b] A. Segev and A. Shoshani. The Representation of a Temporal Data Model in the Relational Environment. In *Proceeding of the 4th International Conference on Statistical and Scientific Database Management*, 1988.

[SS93] A. Segev and A. Shoshani. A Temporal Data Model based on Time Sequences. In *Temporal Databases: Theory, Design, and Implementation*, pages 248–270, Benjamin/Cummings, 1993.

[Ull88] J. D. Ullman. *Database and Knowledge - Base Systems*, Volume I of *Principles of Computer Science*. Computer Science Press, 1803 Research Boulevard, Rockville, MD 20850, 1988.

[WVO93a] J. Wijsen, J. Vandenbulcke, and H. Olivé. Functional Dependencies Generalized for Temporal Databases That Include Object-Identity. In *Proceedings of the International Conference on the Entity-Relationship Approach*, pages 100–114, 1993.

[WVO93b] J. Wijsen, J. Vandenbulcke, and H. Olivé. A Theory of Keys for Temporal Databases. In *Actes 9emes Journees Bases de Donnees*, pages 35–54, 1993.

[WVO93c] J. Wijsen, J. Vandenbulcke, and H. Olivé. Database Design with Temporal Dependencies. Technical Report, Departement Toegepast Economische Wetenschappen, 1993.

[WVO94a] J. Wijsen, J. Vandenbulcke, and H. Olivé. On Time-Invariance and Synchronism in Valid-Time Relational Databases. *Journal of Computing and Information*, 1(1), 1994.

[WVO94b] J. Wijsen, J. Vandenbulcke, and H. Olivé. Temporal Dependencies in Relational Database Design. In *Actes 10èmes Journées Bases de Données*, pages 157–169, Clermont-Ferrand, France, 1994.

[Zan82] C. Zaniolo. Database Relations with Null Values (Extended Abstract). In *Proceedings of the ACM PODS '82*, pages 27–33, March 1982.