Semantics of Time-Varying Information

Christian S. Jensen and Richard T. Snodgrass

TR 96-02

March 7, 1996

Abstract

This paper provides a systematic and comprehensive study of the underlying semantics of temporal databases, summarizing the results of an intensive collaboration between the two authors over the last five years. We first examine how facts may be associated with time, most prominently with one or more dimensions of *valid* time and *transaction* time. One common case is that of a *bitemporal relation*, in which facts are associated with exactly one valid time and one transaction time. These two times may be related in various ways, yielding *temporal specialization*. Multiple transaction times arise when a fact is stored in one database, then later replicated or transferred to another database. By retaining the transaction times, termed *temporal generalization*, the original relation can be effectively queried by referencing only the final relation. We attempt to capture the essence of time-varying information via a very simple data model, the *bitemporal conceptual data model*. Emphasis is placed on the notion of snapshot equivalence of the information content of relations of different data models.

The logical design of temporal databases is a natural next topic. Normal forms play a central role during the design of conventional relational databases. We show how to extend the existing relational dependency theory, including the dependencies themselves, keys, normal forms, and schema decomposition algorithms, to apply to temporal relations. However, this theory does not fully take into account the temporal semantics of the attributes of temporal relations. To address this deficiency, we study the semantics of individual attributes. One aspect is the *observation* and *update patterns* of attributes—when an attribute changes value and when the changes are recorded in the database, respectively. A related aspect is when an attribute has some value, termed its *lifespan*. Yet another aspect is the values themselves of attributes—how to derive a value for an attribute at any point in time from stored values, termed *temporal derivation*. This study of attribute semantics leads to the formulation of temporal guidelines for logical database design.

C. S. Jensen
Department of Mathematics and Computer Science
Aalborg University
Fredrik Bajers Vej 7E
DK–9220 Aalborg Ø
DENMARK
csj@iesd.auc.dk

R. T. Snodgrass
Department of Computer Science
University of Arizona
Tucson, AZ 85721
rts@cs.arizona.edu

Contents

List of Figures

1 Introduction and Historical Context

This paper summarizes the result of an intensive collaboration between the two authors over the last five years into the semantics of time-varying information. A wide variety of topics were investigated, yielding a comprehensive understanding both of this semantics and of why such disparate approaches to temporal data modeling have appeared in the literature.

Christian initially came to Tucson in January, 1991 to start a seven-month sabbatical[1]. Rick had been at the University of Arizona for 16 months. We had each read the other's work, but had met only a few times at conferences. There existed no established joint research stream, nor commonality other than a shared interest in temporal databases. Fortunately, in turned out that we worked very effectively together, and Christian was able to come to Tucson for additional sabbaticals during January–August, 1992 and July, 1994–January, 1995. Rick returned the favor with several shorter visits to Denmark.

Rick had previously worked on temporal query language design and implementation, in the context of his TQuel language [HS93, MS90, Sno87, SGM93, Sno93], and on temporal semantics, specifically characterizing the orthogonality of valid time and transaction time [SA86]. Christian had previously worked on transaction-time databases, specifically architecture [JMRS93, JM93], implementation [JMR91], and language support [JM92].

In our initial discussions once Christian arrived, we identified two areas of common interest: understanding the semantics of temporal data in detail, and developing efficient implementation techniques for bitemporal databases. In large part due to the many projects already underway by Rick's students addressing implementation, we decided to focus instead on the semantics of time-varying information.

At that time, there had already been over a decade of work on temporal databases, principally on temporal query languages and their associated data models. Unlike relational databases, in which a single data model, the relational data model [Cod79], held sway, there were perhaps 20 extant temporal data models described in the literature (that number has since doubled). There was little consensus on the features that a temporal data model should include. Quite the contrary: there was a raging debate over whether the data model should be nested or not (characterized as *first normal form* (1NF) versus *non-1NF* (N1NF) approaches). While there had been some comparisons between the proposals (e.g., [MS91a]), there had been little work to delineate the notions underlying these varied models.

This lack of consensus of even a starting point for work on query language design, query optimization, or temporal access methods was starting to have a constricting effect on temporal database research. Certainly it was complicating temporal semantics and its close relative, temporal database design.

The lack of a single, or at least consensus, temporal data model had less impact on early work on conceptual modeling of time and time-varying information, the latter primarily in the context of the ER model and its temporal extensions. There were also insights from temporal logic, a prominent example being the various models of the time line: dense, continuous, discrete, and branching. Several authors had emphasized the utility of a stepwise constant semantics, in which a fact stored in the database remains true until modified or updated (a kind of Newtonian second law). There had been a few efforts to define *temporal normal forms*; however, all were specific to a particular data model, limiting their applicability. Finally, there occasionally appeared in papers various observations about attribute semantics, anomalies, and normalization.

At the start, we explicitly intended to not produce yet another data model, with its own peculiarities; that would only add to the confusion. Instead, we hoped to discern the underlying

[1] Mention of one author is in the third person; mention of both authors is in the first person.

semantics of temporal data. Our vague intuition was that much of the work on temporal data models was "representational" in nature. It seemed that the model-independent semantics of time-varying information was being forced into specific configurations by existing data models. The resulting structures did capture some of the essence, but were to a large extent artifacts of the data model itself, rather than emphasizing the underlying information content. At the same time, we realized that considering information outside the context of a data model, *some* data model, would have been an aimless and ultimately unsatisfying exercise.

Our early discussions focused on several confusing aspects that we felt might lead us to more fundamental issues.

- Why are there so many temporal data models? Is a single ideal model even possible? As a more specific related question, should data be stored as events (state transitions) or as states?

- Are transaction time and valid time really orthogonal, as Rick had previously claimed [SA86]? More specifically, what is the relationship between POSTGRES' two timestamps, TQuel's four timestamps, and Ben-Zvi's five timestamps? Are there more than two dimensions of time? How does Thompson's taxonomy of four kinds of time relate to valid and transaction time?

- Is first normal form versus non-first normal form a fundamental distinction?

- Which data model aspects are concerned with the information content of the modeled data, which aspects are best justified by their interaction with query language facilities, and which aspects concern only efficiency, and thus are in the domain of physical design? As a specific question, is the problem of NULL values in some temporal data models a logical issue, concerning data semantics, or a physical issue, concerning only performance?

- Can conventional dependency theory be applied in a model-independent fashion to temporal databases? Can existing temporal functional dependencies and normal forms particular to individual data models be recast to apply to a larger subset of data models?

- What are the implications of temporal interpolation on data semantics?

Thinking about these leading questions and following the technical threads that emerged turned out to be a great adventure. This paper gives some of the of the milestones along that journey and provides often surprising answers to the above questions.

The exposition that follows deviates somewhat from a strict chronological order. Rather, we first consider the fundamental question of how to associate time with facts, leading to an understanding of the nature of temporal data. We then address the design of temporal databases. Initially, existing normal forms are generalized to apply to temporal databases. Then, temporal design guidelines are developed based on a study of the semantics of individual attributes of temporal relations. For each new topic introduced, we start with the initial questions that got us thinking about the issue, then follow the investigation as it unfolded.

2 Associating Time with Facts

The past decade of temporal DB research presented a conundrum. Time-varying data seems so simple: rather than one value, there is a value for each instant of time. Yet it seemed that temporal data model design was terribly complicated. There were a plethora of temporal data models, now over 40 discussed in the literature [ÖS95]. There must be something else going on. So we worked hard to get to the essence of temporal data.

Philosophers have long recognized the dichotomy, and the duality, between events and states [RU71]. A *state* is something that has extent over time. Something is true about an object for an interval of time, but was not true before and not after. An *event* is instantaneous [JCE$^+$94]; it is something that "happens," rather than being true over time[2]. Events delimit states. The occurrence of an event results in a fact becoming true; later, the occurrence of another event renders that fact no longer valid. Hence, events and states are duals; states can be represented by their delimiting events, and events are implied by states.

A conventional relation models the reality relevant to an enterprise as a single state [Sno87]. This is often illustrated as a two-dimensional table, with the tuples as rows and the attributes as columns. If nothing changes in reality, the tuples will remain in the relation. Otherwise, some tuples are removed and others are inserted into the relation.

It is well known that database facts have at least two relevant temporal aspects [SA85, SA86]. *Valid time* concerns when a fact was true in the modeled reality [JCE$^+$94]. *Transaction time* concerns when a fact was current in the database. These two aspects are orthogonal, in that each could be independently recorded or not, and each has associated with it specific properties. The valid time of a fact can be in the past or the future and can be changed freely. In contrast, the transaction time of a fact cannot extend beyond the current time (there is no way of knowing whether the fact will be current in the database in the future), and the transaction time cannot be changed (we cannot now change what was stored in the database in the past).

Such was the context for the start of our investigation of temporal database semantics. The simplicity of associating with each fact two times, one valid time, indicating when the fact was true in reality, and one transaction time, indicating when that fact was current in the database, was not adequate to capture the full semantics of time-varying information. We then began a systematic study of the frayed edges of this appealing framework.

2.1 Temporal Specialization

While valid time and transaction time had been shown to be orthogonal [SA86], some papers did not make a distinction between the two. Instead, they seemed to use one time to handle both aspects. For example, the POSTGRES papers mentioned "time travel," terminology strongly suggesting valid time: "For example to find the salary of Sam at time T one would query [...] POSTGRES will automatically find the version of Sam's record valid at the correct time and get the appropriate salary" [SRH90, p. 515]. However, POSTGRES technically supports only transaction time in its data model and query language. Clearly something was going on that was not being captured.

EXAMPLE: Consider a relation recording the assignment of employees to departments, using two attributes, `Name` and `Dept`. On Monday, we observe that employee Tom is in the Shipping department and that Kate is in the Loading department. By end of Tuesday, Tom leaves the Shipping department, and on Wednesday another employee, Sam, starts in Shipping. By end of Thursday, Kate leaves Shipping. This can be represented in a temporal table as illustrated in Figure 1(a).
The *timeslice* at any time yields the conventional relation at that time. For example, the timeslice at time Wednesday yields the relation in Figure 1(b). □

[2] We do not consider the so-called "macro events" that are true, or take place, for an interval of time, but are not true for any subset of their interval. A wedding is an example [Eva90, MMCR92].

Name	Dept	Time
Tom	Shipping	Monday – Tuesday
Kate	Loading	Monday – Thursday
Sam	Shipping	Wednesday – now

(a)

Name	Dept
Kate	Loading
Sam	Shipping

(b)

Figure 1: A Sample Temporal Relation (a) and a Timeslice (b)

The question we asked was, is the temporal relation a valid-time relation or a transaction-time relation? Our eventual answer was: either, or perhaps even both, i.e., a bitemporal relation.

The insight was to consider the interaction between valid and transaction time. While the semantics of these two times is indeed orthogonal, their *use* in a particular application needs not be. In the employee example, the relation is updated precisely (at a granularity of days) when reality changes. On Tuesday, there was no change to the assignment of employees (we will examine this *stepwise constant* assumption more fully in Section 3.5), and so no updates were made to the relation. The relation is assumed to be always up to date; otherwise, the timeslice might not yield the correct result. In this light, the employee relation may be considered to be a transaction-time relation, and the timestamp a transaction time representing when the fact was stored in the database. All modifications are insertions, except that right end points for the timestamps are being are being supplied when assignments are terminated (more on this in Section 2.9). A timeslice at any time in the past yields what the database stored as current at that time.

An equally correct interpretation is that the employee relation is a valid-time relation, with the timestamps indicating when in the past the employee assignments held true. Modifications reflect a change in our understanding of reality; when we learn about a change, we update the relation. A timeslice at a time in the past yields what assignments were valid in reality at that time.

A third, equally correct interpretation is that the employee relation is a bitemporal relation. The transaction time and the valid time, for this application, are synchronized. Hence we could replicate the timestamp, and consider one the valid time and one the transaction time. While a bitemporal relation affords additional query and update capabilities (e.g., retroactive updates), such features are not used by this particular application.

This led us to consider other interactions between valid and transaction time. We term relations with such relationships *specialized* temporal relations [JS94a]. We identified a taxonomy of interrelationships—in between the extremes of identity and no interrelation at all—that are possible between the valid and transaction times of facts, shown in Figure 2.

In this taxonomy, the employee relation would be classified as *degenerate*. As another example, a temporal relation is *retroactive* if the facts stored by the tuples are valid before they are entered into the relation, i.e., the facts became true before they were stored. Retroactive relations are common in monitoring situations, such as process control in a chemical production plant, where variables such as temperature and pressure are periodically sampled and stored in a database for subsequent analysis.

Further, it is often the case that some (non-negative) minimum delay between the actual time of measurement and the time of storage can be determined. For example, a particular set-up for the sampling of temperatures may result in delays that always exceed 30 seconds. This gives rise to a *delayed retroactive relation* if it is retroactive and if there is a bound on the time between when the fact became true in reality and when it was stored in the database.

In a data warehousing application where, e.g., point-of-sales records from an operational system are entered into a warehouse relation on a monthly basis, the valid times of the point-of-sales records

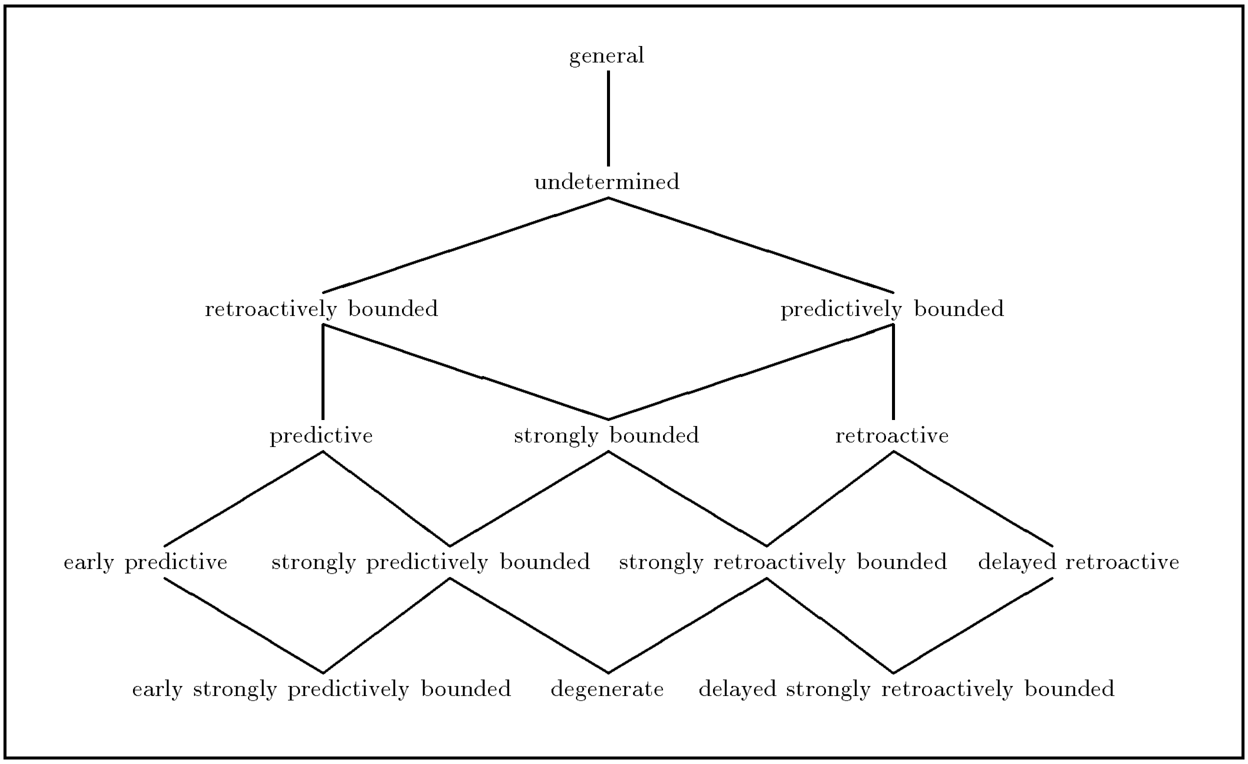


Figure 2: Generalization/Specialization Structure of the Taxonomy for Temporal Specialization

are between a month and a few hours earlier than the corresponding transaction times. Thus, the temporal warehouse relation is *delayed strongly retroactively bounded*.

A temporal relation is *predictive* if the values of an item are not valid until some time after they have been entered into the relation. An example is a relation that records direct-deposit payroll checks. Generally a copy of this relation is made on magnetic tape near the end of the month, and sent to the bank so that the payments can be effective on the first day of the next month. The *early predictive* temporal relation is the specialization of the predictive temporal relation. The direct-deposit payroll check relation is an example if the tape must be received by the bank at least, say, three days before the day the deposits are to be made effective.

The taxonomy of specialized temporal relations provides a coherent framework that allows us to more precisely describe, distinguish, and thus understand temporal relations. The taxonomy may also be used for characterizing the many existing temporal data models. We illustrate this by characterizing several well-known temporal data models.

Ariav's Temporally Oriented Data Model includes the *temporal isomorphism* assumption, in which "there is a tight correspondence between the database and the temporally concurrent reality it is aimed to capture." [Ari86, p. 503]. As the transaction time of a fact can be determined from the stored valid time, under this assumption, this data model supports degenerate bitemporal relations as well as general transaction-time relations.

Gadia presents a multi-dimensional data model which is in turn restricted to a two-dimensional data model with valid and transaction time as the dimensions [GY88]. In this model, however, only data valid in the past may be stored. For example, it is impossible to store on May 11, 1995 the fact that "Employee Kate will be in the Shipping department from September 1, 1995 until February 29, 2000." Therefore, the model does not support fully general bitemporal relations, but supports instead retroactive bitemporal relations. The restriction to retroactive data is inherited from an earlier (retroactive) valid-time data model [Gad88].

Sarda proposes another specialized temporal data model in which current facts may be appended and where so-called retrospective updates (changes to information about the past) are possible [Sar90]. Hence, the transaction time is always equal to or after the valid time, and, like the previous model, this model supports retroactive bitemporal relations.

The POSTGRES data model [Re87, Sto87a] supports degenerate bitemporal relations, in that facts valid now in the real world are stored now, and all past states are retained. The POSTGRES query language [Sto90] supports transaction timeslice. This query language may be viewed alternatively as a transaction-time, valid-time, or even bitemporal query language, with significant restrictions on the expressiveness (query and data) of each. valid-time or bitemporal query language.

Temporal specialization goes down the taxonomy, adding constraints on the interaction of valid and transaction time. Temporal generalization goes up the taxonomy, removing constraints. While considering a different aspect of temporal semantics, we discovered that it made sense to apply generalization above even the top-most point of the hierarchy in Figure 2, yielding temporal relations more general than those termed *general* in the hierarchy, as we will see in the next section.

2.2 Temporal Generalization

A common concern voiced about temporal data models was, why timestamp facts with only one or two timestamps?

EXAMPLE: Consider a promotion decision at a University, which is associated with many dates: the date materials were submitted, the date the departmental committee made its decision, the date the department head decided, the date the college committee decided, the date the Dean decided, the date the Provost's committee decided, the date the Provost decided, the date the President decided, the effective date of the promotion, and the date when each of these decisions was stored in the database (whew!). □

Does it make sense to associate more than one timestamp (valid or transaction) with a fact? Which timestamps are in fact valid and which are transaction? Does it really matter?

The latter two questions are easier to answer. Yes, it does matter, for the simple reason that each kind of time has a particular semantics. The database designer determines the temporal support—valid-time, transaction-time, or bitemporal—of the relations that is appropriate for the applications at hand. The application programmers then exploit that support. Valid and transaction time have precise, crisp definitions. If changes to the past are important, then valid-time support is required. If it is necessary to, e.g., rollback to a previous state of the database, then transaction time support is called for.

Let us examine the promotion decision example more closely. The submission of materials concerns reality, as do the various decisions. These would have dates associated with them regardless of whether they were ever stored in the database. This hints that each of these dates concerns valid time. But which is *the* valid time of the promotion? None of these dates, it turns out. The valid time of the promotion is the time the promotion was valid, that is, its effective date.

The apparent confusion, both in the paragraph above and in some of the research literature, occurs because it makes little sense to reason about what the transaction time and the valid time is abstractly, without reference to a particular fact. We must first identify the fact we are considering! Then it not only makes sense, but also becomes easy to talk about transaction time, valid time, and other times.

So, let us first determine what fact is being timestamped. If the fact is "person X was promoted to Professor," the valid time is the time when the person became a Professor, and the transaction time is when the fact that the person was a Professor was recorded in the database. If the fact is "person X was approved for promotion by the department head," then the valid time is the time when that approval was made (probably when the letter from the department head was signed) and the transaction time is when that fact was recorded in the database. If the fact is "person X is a Professor," the valid time is the interval that started when the promotion decision took effect and is terminated when person X is no longer a Professor. Hence, we see that there are many interrelated facts, each with different valid times and (potentially!) with transaction times.

This discussion provides insight into the relationship between valid time and the so-called *decision time* that has been considered in the literature (e.g., [CK94, KC93, CK93, EGS93, GES94, NE95, ND96]). Assume that we are considering the fact "person X is a Professor." In the example, many decisions took place a different times before person X could become a Professor. These times are decision times of our fact. Different types of facts may have different numbers of different types of decision times. The discussion above reveals that the decision times of a fact are also valid times of facts that are closely related to the "main" fact. What the "closely related" facts are is dependent on the reality to be modeled and on the requirements of the application at hand. Specifically, no general statement can be made about the number and specific meaning of the decision times can be made.

The question is then how to best reflect decision times in a data model. One approach is to store decision times as valid times of the related facts. This permits any number of decision times to be (indirectly) supported, and it clarifies what the individual decision times actually mean. Another approach is to allow for the direct association of an arbitrary number of decision times with all database facts. So far, proposals that take this approach have considered only one decision time per fact. As the meaning of this decision time will vary from application to application, little semantics can be built into the data model for this time. It is not yet known what the benefits of a general solution with this approach are and whether these benefits outweigh the added complexity.

There is, however, an unrelated rationale for storing multiple transaction times in a tuple. This insight followed from considering the four time domains introduced in Thompson's dissertation [Tho91]. When facts flow between temporal relations, several time dimensions may be associated with individual facts.

EXAMPLE: Consider again the promotion decision. This fact has an associated time when the promotion was effective as well as the time when it was entered into a relation on the University's administrative computer. Later, this fact was copied into the departmental personnel relation on a different machine, and is associated with an additional time value, namely the time it was stored there. This personnel relation has three times. Storing both transaction timestamps makes it possible to query the one relation from another relation. In the example, it is possible to query the time-varying relation on the centralized administrative machine indirectly via the personnel relation on the departmental machine. Unlike in the previous discussion, when the multiple times were associated with multiple facts, here we have a *single* fact, "person X is a Professor," with a single valid time and two transaction times: when that fact was stored in the University's database and when it was stored in the departmental database. □

The ability to have multiple transaction times fits in well with temporal specialization. The concepts of specialization and generalization have been used previously within data modeling (e.g., [EN89, HM81, SS77]). A subclass may be created from a class by means of specialization, i.e., by making the defining properties (the intension) of the class more restrictive and thus also restricting the set of examples (the extension) of the class. As the dual, a superclass may be created from a class by means of generalization, i.e., by making the intension of the class less restrictive and thus expanding the extension of the class.

Temporal specialization and generalization are also duals. As we have seen, specialization contracts the space of possible timestamp combinations. Temporal generalization appears in at least four guises, each of which expands the space of possible timestamps. The first is removing restrictions. For example, a strongly predictively bounded relation may be generalized to a predictively bounded relation. This generalization is the opposite of specialization, and it involves moving up the lattice given in Section 2.1.

A second way to define a generalized temporal relation is to simply add completely new, orthogonal time dimensions. In systems where facts flow between multiple temporal relations, facts may accumulate transaction timestamps by retaining their previous timestamps and gaining new transaction timestamps as they are entered into new temporal relations. Consequently, a fact in a generalized temporal relation has several kinds of timestamps: a valid timestamp, which records when the fact was true in reality, a *primary* transaction timestamp, which records when the fact was stored in this relation, and one or more *inherited* transaction timestamps, which record when the fact was stored in previous relations.

A third, more involved, means of defining generalized relations is to have derived relations inherit transaction time-stamps from their underlying relations. For example, consider process control in a chemical manufacturing plant. Values from temperature and pressure sensors may be stored in temporal relations. The sensed data may later be processed further to derive new data, such as the rates at which the reaction is progressing [Ram92]. This derivation typically would depend on past temperature and pressure trends. The derived temporal relation that records the reaction rates would store the transaction time when the rate was recorded, along with one or more inherited transaction times, specifying when the underlying data, the temperature and pressure readings, were originally recorded. These underlying transaction times provide an indication of the relevance of the calculated rates.

A fourth way of generalizing temporal relations occurs when different beliefs about the modeled reality is to be recorded. For example, a database that records the history of some country and is being used by historians may benefit from the inclusion of multiple valid-time dimensions. The different valid-time dimensions may be used for accommodating different, competing perceptions of history.

This elaboration of the original taxonomy of valid and transaction time [SA85] allowed us to better understand Thompson's 4-time model [CT90, Tho91]. Specifically, Thompson's *physical time* is precisely the transaction time of a base financial relation, his *logical time* is the valid time of this relation, his *accounting time* (when that relation is closed out) is the valid time of the relation resulting from the close out process, and his *engineering time* is the inherited transaction time in the close out relation.

Our conclusion is that for facts stored in databases, two kinds of times are fundamental and universal, namely valid time and transaction time, and that these are indeed orthogonal. However, an application's usage of these two time dimensions may introduce interdependencies between the timestamps, multiple valid times, and multiple (inherited) transaction times. In this light, decision time and Thompson's physical, logical, accounting, and engineering times may be seen as valid or transaction times with refined semantics.

From now on, we will assume one valid time (either event or state) and one transaction time.

2.3 Temporal Data Models

At this point, we felt that we had a good handle on the semantics of timestamps. We then turned to the central question of the semantics of time-varying values. How should time be associated with facts? There were at the time some two dozen temporal data models that timestamp facts in some way with valid time. Each proposal came with justifications as to why it was better than the others. Each proposal appeared in a refereed conference or journal, and thus had survived the reviewing process, and was judged to make a contribution.

Rick and a colleague previously analyzed a dozen or so models [MS91a], and had come to the conclusions that (a) there were many desirable criteria for a temporal data model, (b) each model satisfied a substantial subset of the desirable criteria, (c) the design space had been thoroughly explored, in that there generally existed a data model for each combination of relevant aspects, and (d) the desirable criteria were mutually incompatible. So a temporal data model that did everything was simply unattainable.

The implicit mind-set of those developing temporal data models was to find the ideal combination of properties, to come as close to the perfect model as possible. The data models we had individually designed before our collaboration also sought this holy grail [JMR91, MS90, MS91b, Sno87]. We eventually decided that that course of action was inappropriate. The specific design decisions were highly subjective. Because the criteria were incompatible, many design decision necessarily forced useful properties to be unmet. Instead of one design towering over the others by virtue of it satisfying most of the desirable properties, the situation was unavoidably one of a plethora of designs, each with its strong, but also weak, points.

So we decided that the best approach was to alter our goals, instead advocating a separation of concerns. Rather than attempt to define a temporal data model that did everything, we would eliminate those aspects not central to capturing the temporal semantics of the data, which is after all the primary job of a temporal data model. In particular, we would not be concerned with presenting all the information concerning an object in one tuple, or of ensuring ease of implementation and query evaluation efficiency. With a shorter list of requirements, we would then identify a data model that was ideal, in that it did all that was asked of it.

Focusing just on semantics, we found that the existing data models, including our own, were too complicated. These complications arose from the other requirements they were addressing. So we developed a very simple data model, the *Bitemporal Conceptual Data Model*, or BCDM [JSS94], whose sole goal was to capture when facts were valid in reality and when they were stored in the database.

The BCDM is termed a *conceptual* model dues to its single-minded focus on semantics. In essence, we advocate moving the distinction between the various existing temporal data models from a semantic basis to a physical, performance-relevant basis, utilizing our proposed conceptual data model to capture the time-varying semantics. The terminology of "conceptual" is used only to emphasize the use of the model for design and as a basis for a query language; otherwise, this new model is similar to other temporal data models in the formalism used to define it.

We rely on existing data model(s) for the other tasks, by exploiting equivalence mappings between the conceptual model and the *representational* models. The equivalence mappings are well-behaved in that they preserve *snapshot equivalence*, which says that two relation instances have the same information content if all their snapshots, taken at all times (valid and transaction), are identical (a precise definition will be provided later). Snapshot equivalence provides a natural means of comparing relation instances in the models considered in this paper. Finally, we feel that the conceptual data model is the appropriate location for database design, as we shall demonstrate in Section 3.

2.4 The Bitemporal Conceptual Data Model

The idea behind the BCDM was to retain the simplicity of the relational model while also allowing for the capturing of the temporal aspects of the facts stored in a database. This was accomplished by associating with each conventional relational database tuple a region in the space spanned by transaction time and valid time that succinctly defines the temporal aspects of the tuple. Below, we describe this in more detail.

The BCDM employs the same model of time for both time domains: that of a finite sequence of chronons. In mathematical terms, this sequence 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 or years, 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 represents 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. Unions of intervals are termed *temporal elements* [Gad88].

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]). 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]. Surrogates 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 different explicit attributes from $\mathcal{D}_A$ with domains in $\mathcal{D}_D$, and an implicit timestamp attribute, T, with domain $2^{(\mathcal{D}_{TT} \cup \{UC\}) \times \mathcal{D}_{VT}} \setminus \emptyset$. 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 tuple $(a_1, a_2, \ldots, a_n |\ t^b)$, in a bitemporal conceptual relation instance, $r(R)$, consists of a number of attribute values associated with a bitemporal timestamp value. Depending on the extent of decomposition, such a tuple may be thought of as encoding an atomic or a composite fact. For convenience, we will simply use the terminology that a tuple encodes or records a fact and that a bitemporal relation instance is a collection of (bitemporal) facts.

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.

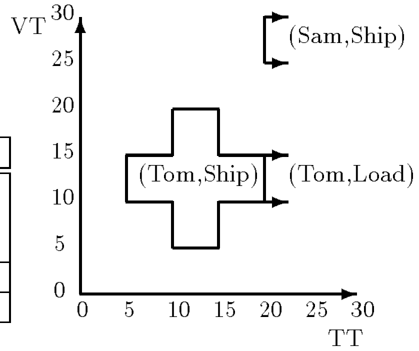
EXAMPLE: Consider a relation recording employee/department information, such as "Tom works for the Shipping department." We assume that the granularity of chronons is one day for both valid time and transaction time, and the interval of interest is some given month in a given year, e.g., January 1995. 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 3(a) shows an instance, `empDep`, of this relation. A graphical illustration of the `empDep` relation is shown in Figure 3(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, Tom and Sam, contained in three tuples. The first two tuples indicate when Tom worked for the Shipping and Loading departments, respectively. These two tuples are shown in the graph as the regions labeled "(Tom, Shipping)," and "(Tom, Loading)," respectively. The last tuple indicates when Sam worked for the Shipping department, and corresponds to the region of the graph labeled "(Sam, Shipping)." □

`EName`	`Dept`	T
Tom	Shipping	$\{(5,10),\ldots,(5,15),\ldots,(9,10),\ldots,(9,15),$ $(10,5),\ldots,(10,20),\ldots,(14,5),\ldots,(14,20),$ $(15,10),\ldots,(15,15)\ldots,(19,10),\ldots,(19,15)\}$
Tom	Loading	$\{(UC,10),\ldots,(UC,15)\}$
Sam	Shipping	$\{(UC,25),\ldots,(UC,30)\}$

(a)



(b)

Figure 3: A Bitemporal Conceptual Relation

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.

As evidence of the simplicity of the relations in the BCDM, it should be noted that, unlike in other models, there is exactly one tuple per fact. We shall also see that BCDM relation instances that are syntactically different have different information content, and vice versa. This conceptual cleanliness is generally not obtained by other bitemporal models where syntactically different instances may record the same information.

2.5 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 [SJS95].

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, $\pi_D^{\mathrm{B}}(r)$, is defined as follows.

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

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, $\sigma_P^{\mathrm{B}}(r)$, is defined as follows.

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

As can be seen from the definition, $\sigma_P^{\mathrm{B}}(r)$ 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[3].

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

Thus, $\tau_{c^v}^{\mathrm{B}}(r)$ 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, $\rho_{c^t}^{\mathrm{B}}(r)$ performs the same operation, except the selection is performed on the transaction time c^t.

EXAMPLE: Consider the `empDep` relation shown in Figure 3(a). The following result is produced by τ_{12}^{B}(`empDep`).

`EName`	`Dept`	T
Tom	Shipping	$\{5, \ldots, 19\}$
Tom	Loading	$\{UC\}$

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. □

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

The operators above apply only to bitemporal relations. Similar operators for valid-time and transaction-time relations are simpler special cases and are omitted for brevity. We will use superscripts "$^{\mathrm{T}}$" and "$^{\mathrm{V}}$" 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 [JSS94]. While other temporal data models often do not provide exact counterparts of these timeslice operators, models generally include functionality that permits this extraction of snapshots.

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.

2.6 Representational Models

A bitemporal conceptual relation is structurally simple—it is a set of facts, each timestamped with a bitemporal element, which is a set of bitemporal chronons. Ostensibly, it is modeling the same time-varying reality that the many other temporal data models capture. How can we characterize this interaction between the models? We need to emphasize the notion of "information content." Specifically, a BCDM database, in a simple and straightforward manner, captures a portion of reality. If a database in another data model captures that same portion, then that database has the same information content as the BCDM database.

Central to this comparison of databases is the concept of snapshot equivalence. Two relation instances with the same non-temporal attributes are *snapshot equivalent* if for all valid and transaction-time pairs, their snapshots are identical. The snapshots are produced using timeslice operators or other language constructs, as described in the previous section. Snapshot equivalence is thus a formalization of the notion that two temporal relations have the same information content. This fundamental insight is due to Gadia, who characterized the information content of individual relations by stating that two relations are *weakly equal* if they are snapshot equivalent [Gad86]. We extended this notion to apply to relations of different data models, thereby providing a natural means of comparing structurally diverse databases.

We developed precise mappings, respecting snapshot equivalence, between instances of the BCDM and instances of each of the existing bitemporal relational data models that have been previously proposed [JSS94]. These data models fall into the class of temporally ungrouped bitemporal models [CCT94] and constitute all such models proposed to date, to our knowledge. We also showed how the relational algebraic operators defined in the previous section induced analogous operators in each of the representational models, and how updates of bitemporal conceptual relations could be mapped into updates on relations in the representation. This provides an explicit homomorphism between the BCDM and the six bitemporal data models, emphasizing their similarities (in terms of information content) and abstracting out their differences, which can be argued concern more efficiency and data presentation than semantics.

This homomorphism has wide-ranging implications, some yet to be explored adequately, for temporal database design and implementation. A database designer could design the conceptual schema of the database as a (normalized) collection of BCDM relation schemas, as will be discussed in Section 3. This approach yields guidelines for the design of the logical database schema, also to be discussed in detail, that are independent of any particular representation of a temporal relation. A temporal DBMS may use any of the existing temporal data models as physical data models. The query language, again focusing on semantics, would be based on the BCDM (an example is the consensual query language TSQL2 [Sno95]). Queries against the BCDM would be mapped into algebraic expressions against the representational data model(s) by the DBMS, to be evaluated in an efficient manner. Physical database design would also be in terms of the representational data model. Snapshot equivalence is the central underpinning of this entire framework.

2.7 Implications of the BCDM

With its accompanying separation of information content and particular encodings of the information content, the BCDM allowed us to answer some of the fundamental questions we began with. Should data be stored as events (state transitions) or as states? Our answer is that at a *logical* level, the natural extension of a conventional relation to a temporal relation, the BCDM relation, encodes states rather than events. An event would be fleeting in a conventional relation: A tuple would appear for a single chronon, then disappear. Only states have persistence in the (conventional) relational model. As events and states are duals, the BCDM relation is sufficient.

Relations capturing events are still useful. A database designer might decide that focusing on the events in a particular corner of the design is more natural than focusing on the states induced by those events.

At a *physical* level, the answer to whether data should be stored as events or states is: It depends on which representational model one feels is most appropriate to achieve good performance for the application at hand. Five of the representational models are state-based; the sixth, Jensen's backlog-based scheme, is event-based. Applications may be identified for which each representation is suitable.

Is 1NF versus N1NF really a fundamental distinction? Our reply becomes: yes, at a representational level, but no at a conceptual level. Two of the representational models are attribute timestamped; the other four are tuple-timestamped. The distinction is not one of semantics. Rather, the distinction may be relevant for performance. We provide more insight into this distinction in Section 3.4.2.

What is the relationship between POSTGRES' two timestamps, TQuel's four timestamps, and Ben-Zvi's five timestamps? We showed in Section 2.1 that POSTGRES was a degenerate bitemporal data model, and thus a tuple's two timestamps *Tmin* and *Tmax* [Sto87a] serve as both valid and transaction time, equal to TQuel's four timestamps, two valid (*begin* = *Tmin* and *end* = *Tmax*) and two transaction (*start* = *Tmin* and *stop* = *Tmax*). Using the BCDM, and in particular its spatial metaphor (cf. Figure 3b), we see that POSTGRES tuples are timestamped with rectangles, with the bottom-left and top-right corners constrained to be on the 45° line of $TT = VT$.

We then considered Ben-Zvi's five tuple timestamps. Again, the question was, was the timestamp format chosen to reflect the semantics of data, or for presentation, or for query language reasons? To review, Ben-Zvi's Temporal Relational Model is a tuple-timestamped model, supporting both valid and transaction time. Let a bitemporal relation schema $\mathcal{R}$ have the attributes $A_1, \ldots, A_n, \mathrm{T}$ where T is the timestamp attribute defined on the domain of bitemporal elements. Then $\mathcal{R}$ is represented by a relation schema R in Ben-Zvi's data model as follows.

$$R \;\; = \;\; (A_1, \ldots, A_n, \mathrm{T}_{es}, \mathrm{T}_{rs}, \mathrm{T}_{ee}, \mathrm{T}_{re}, \mathrm{T}_d)$$

In a tuple, the value of attribute T_{es} (*effective start*) is the time when the explicit attribute values of the tuple start being true. The value for T_{rs} (*registration start*) indicates when the T_{es} value was stored. Similarly, the value for T_{ee} (*effective end*) indicates when the information recorded by the tuple ceased to be true, and T_{re} (*registration end*) contains the time when the T_{ee} value was recorded. The last implicit attribute, T_d (*deletion*), indicates the time when the information in the tuple was logically deleted from the database.

It is not necessary that T_{ee} be recorded when the T_{es} value is recorded (i.e., when a tuple is inserted). The symbol '–' indicates an unrecorded T_{ee} value (and T_{re} value). Also, the symbol '–', when used in the T_d field, indicates that a tuple contains current information.

EXAMPLE: The Ben-Zvi relation corresponding to the conceptual relation in Figure 3 is shown below.

Emp	Dept	T_{es}	T_{rs}	T_{ee}	T_{re}	T_d
Tom	Shipping	10	5	15	5	10
Tom	Shipping	5	10	20	10	15
Tom	Shipping	10	15	15	15	20
Tom	Loading	10	20	15	20	–
Sam	Shipping	25	20	30	20	–

In the example, the timestamps T_{es} and T_{ee} are stored simultaneously, hence the registration timestamps associated with the effective timestamps are identical within each tuple. As facts are corrected, the deletion timestamp T_d is set to the current transaction time, effectively outdating the given fact, and a new tuple without a deletion time is inserted. As only two facts are current when all updates have been performed on the database, only two tuples with no deletion times remain. □

The different updates possible in this model lead to six different types of tuples, as discussed below and illustrated in Figure 4.

1. A tuple is inserted with recorded T_{es} and T_{rs} timestamps.
2. A tuple is inserted with recorded T_{es}, T_{rs}, T_{ee} and T_{re} timestamps. In this case, $T_{rs} = T_{re}$.
3. A tuple with an unrecorded T_{ee} timestamp (1, above) has that timestamp set to a particular time.
4. A tuple with an unrecorded T_{ee} timestamp (1, above) is deleted, setting the T_d timestamp.
5. A tuple with the first four timestamps recorded (2, above) is deleted, setting the T_d timestamp.
6. A tuple with the first four timestamps recorded, with $T_{rs} \neq T_{re}$ (3, above), is deleted, setting the T_d timestamp.

Let's examine each resulting tuple in terms of the BCDM two-dimensional graphical metaphor, cf. Figure 3. Tuple (5) corresponds to a rectangle, with bottom left coordinate (T_{rs}, T_{es}) and top right coordinate (T_d, T_{ee}). The bitemporal elements of the remaining five tuples are open-ended. If T_{ee} is not recorded, the bitemporal element is open-ended at the top; if T_d is not recorded, it is open at the right.

The different ways the various data models have adopted for timestamping tuples may be explained as the models having adopted different *covering functions* that encode the regions in a bitemporal element using one or more graphical entities. POSTGRES uses two timestamps to encode a rectangle with two corners on the 45° line; TQuel uses four timestamps to encode arbitrary rectangles, as well as open rectangles (regions 1, 2, and 4 of Figure 4); and Ben-Zvi uses five timestamps to encode the six shapes of Figure 4. From a semantic point of view, all can encode (snapshot-) equivalent information. Their differences are more of an issue of data presentation (how

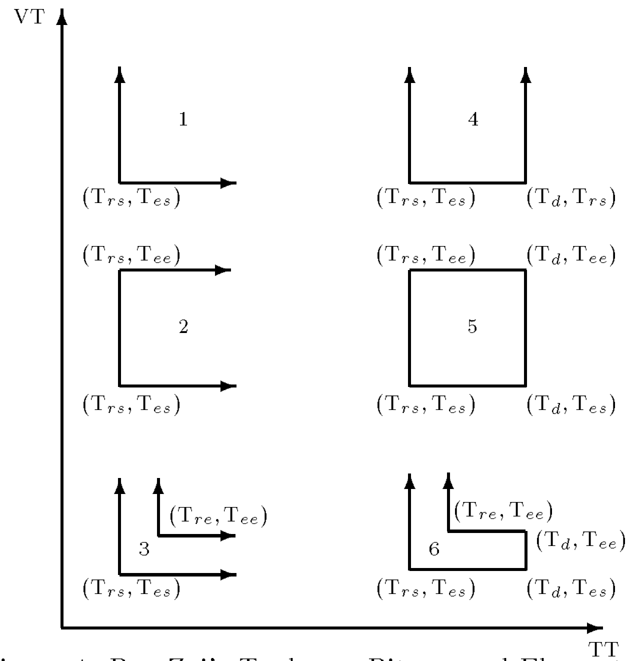


Figure 4: Ben-Zvi's Tuples as Bitemporal Elements

users want to see the temporal information) and storage efficiency. For example, to encode regions 3 and 6 of Figure 4 each require two TQuel tuples. On the other hand, TQuel's data model can encode regions 1, 2, 4, and 5 with one fewer timestamp than Ben-Zvi's data model.

Ben-Zvi's model illustrates another issue, that of what transaction time is. Some authors define the transaction time of a tuple as what is the transaction-time start attribute in TQuel (i.e., *start*) and emphasize that the transaction time of a tuple is a single time instant (e.g., [EGS93, ND96]). This contrasts the definition that we use [JCE+94]. In TQuel terms, the transaction time of a tuple is the time from the *start* to the *stop* attribute value, an interval. With our definition, it is not hard to characterize the transaction time of tuples in Ben-Zvi's model. With the other definition (as a single time instant), we wonder what the transaction time is of each of the six types of tuples (see also [Gad93] where T_{re} is said to be the end of transaction time!).

2.8 Coalescing and Repetition of Information

It turns out that even within a single representational data model, there often is flexibility in representing a bitemporal element. To see this, we use TQuel's four-timestamp rectangles and examine two transformations that can change the covering in a representation without affecting the results of queries, as the transformations preserve snapshot equivalence [JSS94].

The first transformation is termed *coalescing*. Informally, it states that two temporally overlapping or adjacent, value-equivalent tuples may be collapsed into a single tuple [Sno87]. We say that a bitemporal relation instance is *coalesced* if no pair of tuples may be coalesced. Coalescing may reduce the number of tuples necessary for representing a bitemporal relation, and, as such, is a space optimization.

Coalescing of overlapping, value-equivalent tuples is illustrated in Figure 5. The figure shows how rectangles may be combined when overlap or adjacency occurs in transaction time (a) or valid time (b). Note that it is only possible to coalesce rectangles when the result is a bitemporal rectangle. Compared to valid-time relations with only one time dimension, this severely restricts

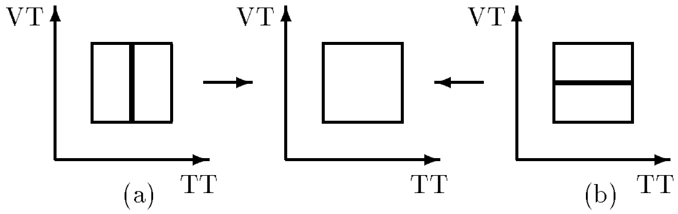


Figure 5: Coalescing

the applicability of coalescing.

As a precursor to explaining the other transformation, we first describe the notion that a relation may have repeated information among its tuples. Specifically, a bitemporal relation instance has *repetition of information* if it contains two distinct tuples that are value-equivalent (i.e., have identical non-temporal attribute values) and have timestamps that encode overlapping regions in bitemporal space. A relation with no such tuples has no repetition of information.

While coalescing may both reduce the number of rectangles and reduce repetition of information, its applicability is restricted. The next transformation may be employed to completely eliminate temporally redundant information, possibly at the expense of adding extra tuples. The transformation maps two value-equivalent tuples with overlapping bitemporal rectangles to three value-equivalent tuples with non-overlapping bitemporal rectangles.

The transformation may partition the regions covered by the argument rectangles on either transaction time or valid time. These two possibilities are illustrated in parts (a) and (b), respectively, of Figure 6.

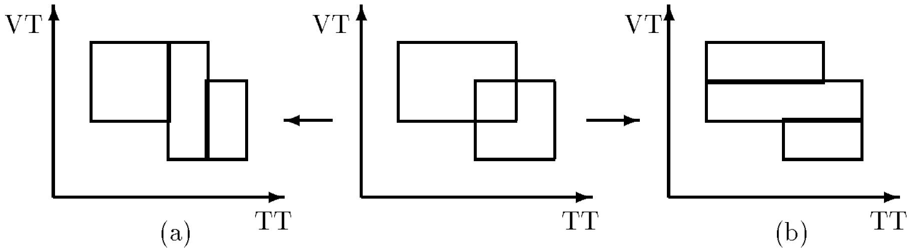


Figure 6: Eliminating Representational Repetition of Information

The transformation is well-behaved. First, it does eliminate repetition among two tuples. Second, the result of an application of the transformation produces at most one additional tuple. Third, repeated application produces a relation instance with no repetition of information. The elimination of repetition of information may thus increase the number of tuples in a representation. The transformation may still be desirable because subsequent coalescing may be possible and, more importantly, because certain modification operations are simplified. (See [JSS94] for a formalization and proofs of these properties.)

2.9 Now and Forever

The next aspect of temporal data that drew our attention was the arrows in Figures 3 and 4. One question was, does the particular semantics of valid time and transaction time imply any differences between upward-pointing arrows and right-pointing arrows? Do open rectangles and L-shaped regions capture the semantics we desire?

Let us return to the employment example. Figure 3 on page 11 contains only right-pointing arrows, indicating information that is still thought to be true, i.e., that has not been logically deleted. Because we cannot know what is stored in the database in the future, the right arrow is always at a transaction time of the current time, or *NOW*.

The figure does not contain upward-pointing arrows because the interval of employment for both Tom and Sam was always known (though not always known correctly, for Tom).

Upward-pointing arrows are illustrated in regions 1, 3, 4, and 6 of Figure 4. These are cases where the terminating time (T_{ee} in Ben-Zvi's model) is not known. We do not know when a fact ceased or ceases being true in reality, so we model it as being forever true. For example, to model the fact that Tom was hired on June 10 in the Loading department, with an unknown termination date, an open rectangle shaped as region 1 of Figure 4 would be used.

If the valid-time domain is bounded, say at some time way in the future, then '–' in Ben-Zvi's model and '∞' in TQuel's model (other data models are similar) are simply shorthands for this maximum valid time. In this sense, in Figure 4, tuples (1) and (2), and (4) and (5), are identical. Tuple (1) is merely a special case of tuple (2) in which T_{ee} is fixed to a particular value; the same applies for tuples (4) and (5).

Modeling Tom's employment as continuing forever is an overly optimistic assumption, and one that is certainly false. We do not know Tom's termination date, but it is certainly within the next 150 years, and probably within the next 10–20 years. In fact, all that we feel that we know for certain is that Tom was in the Loading department from June 10 to June 10 (now), assuming that whenever reality changes (such as Tom resigning), the database is immediately updated. Tomorrow (June 11), if Tom does not resign in the meantime, we will know that Tom was in the Loading department from June 10 to June 11.

In the BCDM, we handled the concept of *NOW* in transaction name by using a special marker, *UC*, that indicated where to add bitemporal chronons to the tuples' timestamps every time the clock advances a tick. This approach is not useful in a practical representation of temporal data. Instead, our solution to being able to model the dependence on the current time is to allow the model to include *variables* as well as ground facts in the stored data [CDI+94]. *NOW* is one such variable that evaluates to the current time. Including such variables increases the fidelity of the data model considerably. To understand its impact, it is useful to consider another kind of time, the *reference time*, which is the time of the database observer's "frame of reference." Reference time is a concept analogous to the *indices* or "points of reference" in intensional logic [Mon73], and discussed more recently in the context of valid-time databases [Fin92]. The reference time facilitates a kind of "time travel" by means of which we may observe the database at times other than the present.

A related time is the *query time*. It is the time at which a query is processed. The reference time and query time are independent concepts. In general, the time when a query is initiated is always the current time, while the reference time is the time at which an observer "observes" the database. In many queries, the user "observes" the database with respect to the frame of reference in which the query was initiated, so the reference time and the query time are the same. But the user may choose to "observe" the database from a previous perspective; for this kind of query, the reference time is earlier than the query time. For example, if today is June 19 and we wish to observe the database from the perspective of a week ago, then the current time (and the query time) is June 19 and the reference time is June 12.

So, how may we visualize a tuple's timestamp when it contains a variable such as *NOW* in transaction and valid time? As the reference time increases, say from June 12 to June 13, the region of the temporal element grows. Only when *NOW* is replaced with a ground value (for valid time, this means that the fact is known to have terminated, and for transaction time, this means that the fact is logically deleted), does the temporal element not grow, in valid time or transaction time, respectively. Illustrating this behavior requires three dimensions: valid time, transaction time, and

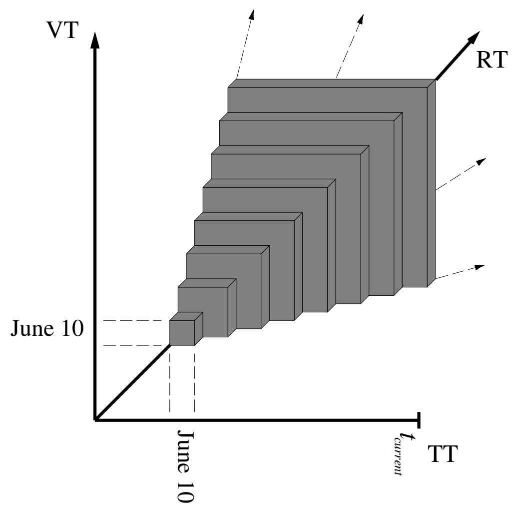


Figure 7: A Rectangle with a Transaction Stop Time and a Valid To Time of *NOW*

reference time. In Figure 7, the dimension that goes into the page illustrates reference time. Here, the fact being recorded is "Tom was in the loading department." Initially, at a reference time of June 10, the database records that the information was valid on June 10.

Finally, we would also like to model facts such as "Tom is definitely employed from June 10 until now, and will probably be employed until the end of the summer, when he will return to school." If we know that changes in reality take two days to make it into the database, we would amend that to "Tom is definitely employed from June 12 until now minus two days, and ..." These facts can be modeled using a refinement of now variables, specifically indeterminate now-relative variables [CDI+94].

2.10 Summary

Up to this point, our focus has been on the association of facts with times. We have seen that while the semantics of valid time and transaction time are orthogonal, their usage within an application may exhibit interactions, such as the valid time always preceding the transaction time in the case of a retroactive relation. A fact may be timestamped with multiple transaction times, if it is copied several times between relations or databases. A fact has typically one valid time, specifying when it was true in reality. The decision time(s) of a fact were seen to be valid times of different, closely related facts.

We have seen that there can be no ideal temporal data model, but that by focusing only on the semantics of time-varying data, and ignoring other possible criteria, a simple data model, the Bitemporal Conceptual Data Model, is quite satisfactory. The BCDM provides insights into the expressiveness of existing temporal data models. Specifically, using snapshot equivalence as the measure, all such models encode the same information; they just break up the bitemporal elements, which are sets of regions in bitemporal chronon space, in different ways, sometimes including a bitemporal chronon in the timestamps of two or more value-equivalent tuples. As another difference, the various models enter the times at different levels (e.g., tuples, attribute values) in the temporal relations.

The right and upward pointing arrows of bitemporal regions represented with timestamps in these models suggested the addition of now-relative variables to the stored data, thereby increasing the modeling power of the model.

3 Design of Relation Schemas

With a considerably increased understanding of temporal data, facts with associated times, we turned to the design of the relations themselves, addressing the question of what constitutes facts (tuples). Existing work on temporal relational design was fragmented, incomplete, and model-specific. We found this state of affairs confusing and unappealing. As there has been a comprehensive dependency theory developed for relational databases, it seemed to us that that theory should be applicable to temporal databases as well.

Previously, an array of temporal normalization concepts had been proposed, including *first temporal normal form* [SS88a] and two different normal forms termed *time normal form* [BZ82, NA89]. Each of these is specific to a particular data model, and thus appropriates the inherent peculiarities of its data model. Furthermore, these normal forms often deviate substantially in nature from conventional normal forms and are in some sense not "true" extensions of these, for a variety of reasons [JSS96].

We adopted a different approach. Since relations in the BCDM can be related to those of other temporal data models, then functional dependencies and normal forms expressed in terms of the BCDM can also be mapped into other data models. We thus chose to apply dependency theory to the BCDM. Furthermore, we wanted our normal forms to be natural extensions of those defined for conventional relations. It turned out that the clean semantics of the BCDM allowed us to do so in a natural fashion.

3.1 Temporal Functional Dependencies

As in design of snapshot relational databases, dependencies are also important during temporal relational database design. As background, we first state the notion of a functional dependency for snapshot relations.

DEFINITION: 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. □

A functional dependency constrains the set of possible extensions of a relation. Which functional dependencies are applicable to a schema reflects the reality being modeled and the intended use of the database. Determining the relevant functional dependencies is a primary task of the database designer.

3.1.1 Generalizing Functional Dependencies to Temporal Databases

In database design, functional dependencies are *intensional*, i.e., they apply to every possible extension. This intuitive notion already encompasses time, for a functional dependency may be interpreted as applying at any time in reality and for any stored state of the relation.

To be more specific, consider the restricted case of a transaction-time relation r, with schema $R = (A_1, \ldots, A_n|\mathrm{T})$, and a parallel snapshot relation r' with the same schema (but without the implicit timestamp attribute): $R' = (A_1, \ldots, A_n)$. The current state of r, denoted by $\rho^{\mathrm{T}}_{now}(r)$, where "*now*" denotes the current time, will faithfully track the current state of r'. Past states of r' will be retained in r, and can be extracted via $\rho^{\mathrm{T}}_{t}(r)$, with "t" being the desired past point in time. A functional dependency on R' will hold for all possible extensions, and hence for all past states of r'. Hence, the same functional dependency must hold for all snapshots of r (this insight first appeared over a decade ago [CW83]). A similar argument can be applied to valid-time relations and to bitemporal relations, yielding the following characterization [JSS96].

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

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

For example, the instance `empSal` in Figure 11(a) satisfies the dependency `EName` $\xrightarrow{\mathrm{T}}$ `Sal`.

In the definition of a temporal functional dependency, a temporal relation is perceived as a collection of snapshot relations. Each such snapshot of any extension must satisfy the corresponding functional dependency.

The parallel between conventional functional dependencies and temporal functional dependencies means that inference rules such as Armstrong's axioms have close temporal counterparts that play the same role in the temporal context as do the non-temporal rules in the non-temporal context.

Next, we can also define temporal keys [JSS96]. For example, the explicit attributes X of a temporal relation schema R form a *(temporal) key* if $X \xrightarrow{\mathrm{T}} R$.

Finally, we can generalize snapshot normal forms in a similar manner.

Definition: A pair (R, F) of a temporal relation schema R and a set of associated temporal functional dependencies F is in *temporal Boyce-Codd normal form* (TBCNF) if

$$\forall\, X \xrightarrow{\mathrm{T}} Y \;\in F^+ \; (Y \subseteq X \vee X \xrightarrow{\mathrm{T}} R). \qquad \square$$

Definition: A pair (R, F) of a temporal relation schema R and a set of associated temporal functional dependencies F is in *temporal third normal form* (T3NF) if for all non-trivial temporal functional dependencies $X \xrightarrow{\mathrm{T}} Y$ in F^+, X is a temporal super-key for R or each attribute of Y is part of a minimal temporal key of R. $\square$

One can also define temporal variants of second normal form, multivalued dependencies [Zan76], fourth normal form [DF92], join dependencies [Ris77], fifth normal form (also called project-join normal form) [Fag79], embedded join dependencies [Fag77], inclusion dependencies [CFP84], template dependencies [SU82], domain-key normal form [Fag81], and generalized functional dependencies [Sad80]. This is done by exploiting the intensional quality of these properties (i.e., applying to every extension implies applying over all time), as well as the simplicity of the bitemporal conceptual data model. Similarly, the notions of lossless-join and dependency-preserving decomposition can be naturally extended to temporal relations. Furthermore, one can define temporal variants of conventional integrity constraints involving uniqueness, referential integrity, and subset and cardinality constraints.

Via the mappings that exist between the BCDM and the representational data models, these dependencies, normal forms, and integrity constraints can also be applied to these models. The result is a consistent and wholesale application of existing dependency and normalization theory to valid-time, transaction-time, and bitemporal databases in a wide variety of temporal relational data models.

3.1.2 Parameterized Temporal Functional Dependencies

In the 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 next parameterized 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 dependencies may capture the temporal semantics of a database schema more precisely than the standard temporal dependency.

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

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

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.

(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 less 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 when 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: To see the differences between the sample predicates (1)–(6), consider the sample relation instances in Figure 8.

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

EmpS	EName	Dept	T
e1	Tom	Shipping	$\{(20,1),\ldots,(20,10),\ldots,$ $(30,1),\ldots,(30,10)\}$
e1	Tom	Loading	$\{(20,1),\ldots,(20,10),\ldots,$ $(30,1),\ldots,(30,10)\}$

(a) empDep1

EmpS	EName	Dept	T
e1	Tom	Shipping	$\{(1,20),\ldots,(1,30),\ldots,$ $(10,20),\ldots,(10,30)\}$
e1	Tom	Loading	$\{(1,20),\ldots,(1,30),\ldots,$ $(10,20),\ldots,(10,30)\}$

(b) empDep2

Figure 8: Sample Bitemporal Relations

opposite holds for `empDep2`. □

Parameterization also applies to integrity constraints in general for temporal databases.

3.1.3 Strong Temporal Functional Dependencies

The temporal dependencies we have seen thus 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. So while we were able to capture dependencies such as a salary attribute (at any time) being determined by an employee-name attribute, we cannot capture that a salary of an employee does not change within a month, or never changes. These latter constraints require looking at more than one time point to determine if the constraint is satisfied by a particular relation instance. This distinction has previously been captured more generally with the terms *intrastate* and *interstate* integrity constraints [Böh94].

While a (regular or parameterized) temporal dependency holds if the corresponding conventional dependency holds for each snapshot in isolation, our first step was to "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: Let X and Y be sets of non-timestamp attributes of a bitemporal relation schema R. A *strong temporal functional dependency*, denoted $X \overset{\mathrm{Str}}{\rightarrow} Y$, exists on R if for all meaningful instances r of R,

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

Consider the relation instance `empSal` in Figure 9. While we have seen that it does satisfy the dependency `EName` $\overset{\mathrm{T}}{\rightarrow}$ `Sal`, it does not satisfy the dependency `EName` $\overset{\mathrm{Str}}{\rightarrow}$ `Sal`. For example, (e1, Tom, 30k) and (e1, Tom, 32k) are in valid timeslices at times 5 and 15, respectively, which violates the dependency. The dependency `EmpS` $\overset{\mathrm{Str}}{\rightarrow}$ `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 Tom to Rob, the dependency is not satisfied. Strong temporal dependencies are useful in part because they have a practical and intuitive interpretation. Specifically, if $X \overset{\mathrm{Str}}{\rightarrow} Y$ 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 `EmpS Dept` $\overset{\mathrm{Str}}{\rightarrow}$ `Sal`.

Strong temporal normal forms and integrity constraints can be analogously defined.

Following this work, Wang and his colleagues generalized this notion to dependencies (and

EmpS	EName	Sal	T
e1	Tom	30k	$\{1, \ldots, 9\}$
e1	Tom	32k	$\{10, \ldots, 19\}$
e1	Tom	36k	$\{30, \ldots, 39\}$
e1	Tom	40k	$\{40, \ldots, 49\}$

Figure 9: The `empSal` Relation

normal forms and decomposition algorithms [WBBJ94]) that were along a spectrum between our temporal functional dependencies, which apply to individual timeslices, and strong functional dependencies, which apply to all timeslices at once. Specifically, they define a functional dependency for each available *granularity* (e.g., second, week, year), and require that the equality hold only during a unit of the granularity. Our temporal functional dependency is a Wang dependency on the smallest granularity (that of chronons); our strong functional dependency is a Wang dependency on a granularity in which all of time is contained in a single granule. Next, Wijsen has recently developed a normalization theory for valid-time databases that includes three types of temporal dependencies [Wij95]. Two correspond to our temporal dependency and strong temporal dependency. The third dependency is in-between the two. This so-called *dynamic dependency* holds if the corresponding snapshot dependency holds on the unions of all pairs of consecutive snapshots.

Returning to our dependencies, we subsequently realized that with strong temporal dependencies, we were able to characterize within our general framework a notion of *synchronous attributes* very similar to the one that had previously been defined by Navathe and Ahmed [NA89] in their particular data model. To define this notion, they first define a notion of temporal dependency.

DEFINITION: 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` $t[A_j] = t'[A_j]$.

[NA89, p. 156] and [Ahm92] □

The goal of this definition is to capture a kind of asynchronism among attributes: If at successive times, 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.

Navathe and Ahmed then define two attributes as being *synchronous* if there is no temporal dependency between them. We determined that it was possible to capture and generalize this notion of synchronism of attributes by using strong dependencies. In the 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: 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 $Y \overset{\text{Str}}{\rightarrow} X$. □

Intuitively, $X \overset{\text{Str}}{\leftrightarrow} Y$ means that the sets of attributes X and Y change values simultaneously, and are thus synchronous. We return to this issue in Section 3.4.2.

We then considered the useful and intuitive concept of time-invariant keys and attributes that had been previously used in the context of this particular data model [NA89]. As we were able to define regular (temporal) keys, could a time-invariant key not be defined the same way? On further thought, it became clear that we needed an unambiguous means of tying an attribute value together with the real-world entity that it concerns.

3.2 Using Surrogates

In data modeling, an attribute is about a particular entity in the modeled reality, and we say that the attribute records a property of that entity. As an example, the frequency of change of a salary attribute with respect to a specific employee in a company may be relatively regular, and there will 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 important to identify the reference entity when discussing the semantics of an attribute.

This insight 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 our approach, 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 with no replacements.

DEFINITION: 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 $S \overset{\text{Str}}{\rightarrow} X$. □

Because it is assumed that different entities are represented by different surrogates and the same entity always is represented by the same surrogate, this is a rather natural definition of *time invariant* attributes. In the `empSal` instance in Figure 11, 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 used with a different meaning [NA89]) results.

DEFINITION: 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 $X \overset{\text{T}}{\rightarrow} R$. □

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 situations such as this, where the surrogate attribute `EmpS` was used to determine that `Emp` is a time-invariant key of schema `empSal`, it may be advantageous to remove the surrogate attribute from the schema.

The temporal dependencies and normal forms that arise from conventional dependency theory, along with the extensions just discussed (parameterized and strong temporal functional dependencies), permit characterizations, as we have seen, of some existing temporal dependencies and normal forms, in a model-independent fashion. However, there were other existing normal forms, such as Time Normal Form [NA89], that were not subsumed by our new definitions. So we turned our attention to those normal forms, in an attempt to devise a comprehensive design methodology.

3.3 Lifespans of Individual Time-Varying Attributes

An "anomaly" that had been mentioned in several papers was the necessity of null values, particularly when tuple time-stamping was utilized. Navathe and Ahmed's notion of synchronous attributes, discussed in Section 3.1.3, served in part to identify where null values may or could not occur. In the absence of synchronism, one attribute might have a value when another one did not, thereby requiring a null value. Gadia's homogeneous data model was predicated on snapshots not adding nulls; indeed, this was the source of the homogeneity requirement [Gad88]. In thinking through this issue, we eventually came to see it as a distinction between inapplicable and unknown nulls. While both can be *represented* with NULL values, the former is relevant for logical design; a good design will obviate the need for inapplicable nulls.

As with time-invariance (cf. Section 3.2), we concluded here that the mechanism of functional dependencies—looking at the relationships among attribute values within individual meaningful relations to determine intensional properties—was not adequate. Rather, it was necessary to consider the time-varying semantics of individual attributes, and make design decisions based on that semantics. Hence, to specify when inapplicable nulls could occur, we adapted the concept of *lifespans*, whose importance had previously been recognized in the context of data models. The HRDM model associates explicit lifespans with each attribute of a relation schema and with each tuple of a relation instance [CT85, CC87, CC93]. 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. 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 a recent extension to his data model, Gadia augmented the timestamp to incorporate definite and possible lifespans, both of objects and of attribute values [GNP92]. In TERM, Klopprogge [KL83] records lifespans by adding mandatory, boolean-valued valid-time attributes, "**existence**," to entity and relationship types. In another extension to the ER model, the TEER model associates lifespans with individual surrogates, which represent entities in that model [EWK93].

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.

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.

To more precisely define lifespans, we first define an auxiliary function, **vte**, that takes as argument a valid-time relation r and returns the valid-time element. Specifically, $\mathbf{vte}(r) = \{c^v \mid \exists s\ (s \in r \wedge c^v \in s[\mathrm{T}])\}$.

DEFINITION: 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.

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

Lifespans are important because attributes are guaranteed to not have an 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.

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

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

To exemplify this definition, consider a relation schema Emp with attributes EmpS (the employee's surrogate), 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) exactly when that employee is 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 lifespan and the salary lifespan for individual employees.

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

DEFINITION: A relation schema $R = (S, A_1, \ldots, A_n \mid T)$ where S is surrogate valued is *lifespan homogeneous* if

$$\forall A, B \in R\ (A \overset{\text{LS}}{=}_S B). \qquad \square$$

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 following design rule.

DEFINITION: (*Lifespan Decomposition Rule*) To avoid inapplicable nulls in temporal database instances, decompose temporal relation schemas to ensure lifespan homogeneity. $\square$

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.

Note that decomposition for this reason may not be required, as the temporal normal forms tend to eliminate the need for inapplicable nulls.

3.4 Pattern-Based Synchronism

In the context of their data model where tuples are timestamped with single time intervals, Navathe and Ahmed [NA89] had previously defined a normal form that is related to synchronism and is based on their notion of temporal dependence as discussed in Section 3.1.3. The intuition was that when two attributes were changing independently (i.e., were temporally dependent), tuple timestamping could generate redundancy. Their normal form then states that relation schemas should not contain temporally dependent attributes.

Definition: 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] □

Example: Consider the relation instance `empDepSal` in Figure 10, recording departments and salaries for employees.

`EmpS`	`Dept`	`Salary`	T
e1	Shipping	30k	$\{1,\ldots,5\}$
e1	Loading	30k	$\{6,\ldots,9\}$
e1	Loading	32k	$\{10,\ldots,14\}$
e1	Loading	36k	$\{15,\ldots,27\}$
e1	Loading	40k	$\{28,\ldots,39\}$
e1	Shipping	50k	$\{40,\ldots,49\}$
e1	Loading	50k	$\{50,\ldots,59\}$

Figure 10: The `empDepSal` Relation.

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. This relation 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). □

The type of redundancy identified by TNF has been mentioned in the past by several researchers (e.g., [CT85, GV85, Gad88, GY91]). Most often, it has been used for motivating a non-first normal form data modeling approach where time is associated with attribute values rather than with tuples, because that approach avoids the redundancy. The problem with such data models, as discussed in Section 2.3, is that they jettison other desirable properties in attempting to eliminate this redundancy.

More centrally, though, this "redundancy" should be more properly termed "data replication," as Wijsen has pointed out [Wij95]. It does not share with other identified redundancies the important property that the values in question can be predicted from other information in the relation. In the example above, the salary 30K is indeed replicated, but nevertheless still contributes to the information content of the relation.

Additionally, this normal form is a very restrictive one. Specifically, it appears that the imposition of TNF effectively leads to a binary data model, in which all relations have just two attributes, an entity-identifying attribute and one time-varying attribute. To see this, observe that if two time-varying attributes are to reside in the same relation schema, we must guarantee that when one attribute changes value, the other also changes its value. If it is at all possible that the value of one attribute (not both) may at some point "change" to its existing value, there is a temporal dependency between the two attributes, and they cannot reside in the same schema.

We thus started looking for less restrictive decomposition guidelines that are based on synchronism. Our first idea was that we did not want to decompose a schema just because an attribute at rare occasions may "change" its value to its existing value. Decomposition guidelines that accomplish this cannot be based solely on the attribute values, as is TNF; and we can use neither strong temporal dependencies nor Navathe and Ahmed's temporal dependency. Rather, we had to focus not on the values themselves, but on the times that the values were observed. As we shall see, this led to a pattern-based notion of synchronism that is quite different from the value-based notions of synchronism that have been considered by others.

3.4.1 Time Patterns of Individual Time-Varying Attributes

Informally, a time pattern is simply a sequence of times. We will use time patterns to define a database design rule that serves a purpose similar to Time Normal Form.

DEFINITION: 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 $j < i$. $T(i)$ denotes the i'th time point. □

In the context of databases, two distinct types of time patterns are of particular interest. The *observation pattern* O_A^s, for an attribute A relative to a particular surrogate s, is the times when the attribute is given a particular value, perhaps as a result of an observation (e.g., if the attribute is sampled), a prediction, or an estimation. Observation patterns relate to 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 relate to transaction time.

Note that an attribute may not actually change value at a time point. It may very well be that the existing and new values are the same (an example will be given shortly). 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, in particular for non-degenerate relations (cf. Section 2.1).

EXAMPLE: 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 experiment is initiated, and each experiment runs 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.

Assuming that an experiment x1 starts at 9:00:00 a.m., its time patterns may be given as follows.

$$O_A^{\mathrm{x1}}(0) = \text{9:00:05},\ O_A^{\mathrm{x1}}(1) = \text{9:00:15},\ \ldots,\ O_A^{\mathrm{x1}}(199) = \text{10:59:55}$$
$$U_A^{\mathrm{x1}}(0) \in [\text{9:00:05},\ \text{9:00:07}),\ \ldots,\ U_A^{\mathrm{x1}}(199) \in [\text{10:59:55},\ \text{10:59:57})$$

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 applications where an account balance by definition takes effect when the balance is stored in a database. As we will show shortly, while the concepts of observation and change patterns are highly useful in database design, it is usually *not* necessary to know the specific time patterns of individual surrogates (indeed, the time patterns are generally not a priori known).

To further illustrate the notion of time patterns, we introduce two important types of time patterns. 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.

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

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.

A *constant* time pattern is a further specialization.

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

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: 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 $l(i) \le T(i) \le u(i)$. □

Example: 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.

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 months 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.

3.4.2 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.

EXAMPLE: Consider the `empDepSal` relation in Figure 10. We recall 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.

$$O^{\mathtt{e1}}_{\mathtt{Dept}} =< [0 \mapsto 1], [1 \mapsto 6], [2 \mapsto 40], [3 \mapsto 50], [4 \mapsto 60] >$$
$$O^{\mathtt{e1}}_{\mathtt{Salary}} =< [0 \mapsto 1], [1 \mapsto 10], [2 \mapsto 15], [3 \mapsto 28], [4 \mapsto 40], [5 \mapsto 60] >$$

With these observation patterns, there is redundancy in the sample instance—we can predict the repeated values. Thus, capturing during database design what attributes of the same relation schema have different observation patterns is a means of identifying a 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. □

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: 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 observation patterns are identical when restricted to the intersection of their lifespans. Often, it is possible to deduce asynchronism, even when the specific observation patterns are not known.

With this definition, we can characterize relations that avoid the redundancy caused by a lack of synchronism.

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

Above, we defined pattern-based synchronism among attributes. In Section 3.1.3 we introduced strong temporal functional dependencies that could capture a notion of value-based synchronism among attributes. The following example will illustrate the relationship between valued-based synchronism and the pattern-based synchronism just introduced.

EXAMPLE: We consider two relations instances. In the first, `Dept` and `Salary` are pattern-synchronous but not value-synchronous.

`empS`	`Dept`	`Salary`	T
e1	Shipping	30k	$\{1, \ldots, 5\}$
e1	Shipping	40k	$\{6, \ldots, 10\}$

We assume the following observation patterns.

$$O^{e1}_{\texttt{Dept}} =< [0 \mapsto 1], [1 \mapsto 6], [2 \mapsto 11] >$$
$$O^{e1}_{\texttt{Salary}} =< [0 \mapsto 1], [1 \mapsto 6], [2 \mapsto 11] >$$

When, at time 6, `Dept` and `Salary` were observed, their values were Shipping and 40k, respectively. These observation patterns, which are consistent with the above instance, imply that $\texttt{Dept} \stackrel{\text{S}}{=}_S \texttt{Salary}$. However, in the relation instant `Dept` and `Salary` are not value-synchronous, because the value of `Salary` changed at time 6, while the value of `Dept` did not.

The next relation instance is value-synchronous, but is not pattern-synchronous.

`empS`	`Dept`	`Salary`	T
e1	Shipping	30k	$\{1, \ldots, 5\}$
e1	Loading	40k	$\{6, \ldots, 10\}$

Since the values of the two attributes track each other, $\texttt{Dept} \stackrel{\text{Str}}{\leftrightarrow} \texttt{Salary}$ for this instance. This instance, in turn, is consistent with the following observation patterns, which are not identical.

$$O^{e1}_{\texttt{Dept}} =< [0 \mapsto 1], [1 \mapsto 6], [2 \mapsto 11] >$$
$$O^{e1}_{\texttt{Salary}} =< [0 \mapsto 1], [1 \mapsto 6], [2 \mapsto 8], [3 \mapsto 11] >$$

The pattern for $O^{s}_{\texttt{Salary}}$ indicates that `Salary` was observed at time 8, and the instance indicates that `Salary` did not change value at that time, so the value observed at time 8 was the same as the existing value. □

Put together, the above example shows that neither of the two notions, $\stackrel{\text{Str}}{\leftrightarrow}$ and $\stackrel{\text{S}}{=}$, 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 independent.

However, note that instances that are consistent with TNF but not with the synchronous decomposition rule (e.g., the latter instance above), only occur when it is guaranteed that no value changes for any time not in both observation patterns. In practice, we expect this to be quite rare. The synchronous decomposition rule is thus typically less restrictive than TNF.

We now address 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 2.6).

Unlike time normal form, the synchronous decomposition rule does eliminate a kind of redundancy.

EXAMPLE: Consider the following relation instance, under the observation patterns specified.

empS	Dept	Salary	T
e1	Shipping	30k	$\{1,\ldots,5\}$
e1	Loading	30k	$\{6,\ldots,10\}$

$$O^{e1}_{\texttt{Dept}} =< [0 \mapsto 1], [1 \mapsto 6], [2 \mapsto 11] >$$
$$O^{e1}_{\texttt{Salary}} =< [0 \mapsto 1], [1 \mapsto 11] >$$

The 30k value in the second tuple is redundant: it can be guessed by using the first tuple and the observation patterns. □

Decomposing relation schemas to eliminate this redundancy may be important when *storing* temporal relations. The specifics depend on the amount of redundancy and the query and update patterns. However, we feel that the redundancy is of little consequence at a logical level, considering query anomalies and modification anomalies.

For the querying of logical-level relations, we use the query language associated with those relations, namely TSQL2. In TSQL2, it is possible to declare variables in the from clause that range over groups of tuples [Sno95].

EXAMPLE: Consider the relation instance in the previous example, containing asynchronous attributes. The following from clause,

```
FROM emp(EmpS, Dept) AS empDept, emp(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	Shipping		$\{1,\ldots,5\}$
e1	Loading		$\{6,\ldots 10\}$

EmpS		Salary	T
e1		30k	$\{1,\ldots,10\}$

Note that the coalescing implied by the from clause isolates the times in which the values change. □

With this facility, we believe that (pattern-based) asynchronous attributes in a relation present no special problems when posing queries.

Assuming that lifespan decomposition has already been performed, there are no insertion nor deletion anomalies, in the sense that no inapplicable nulls are generated by the insertion and deletion operations. Concerning update anomalies, TSQL2 provides facilities that allow the value of an attribute over all time, or over a restricted time, to be updated via a single statement, thereby ensuring that the redundancy does not generate an inconsistency.

In conclusion, the synchronous decomposition rule is clearly relevant to physical-level design. Its relevance to logical-level design seems less obvious and depends on the data model employed. In our conceptual data model, which is a tuple-timestamped first normal form model, it is generally not necessary (and it is probably not even desirable) to separate attributes in logical-level temporal relations on the sole basis of asynchronism.

Several claims have been made in the past about synchronism and database design. The need for synchronism at the logical level has previously been claimed to make normal forms and dependency theory inapplicable (e.g., [GV85]) because it leads to binary relations with no need for further decomposition. 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 databases [...] Our departure from this "hangup" brings temporal databases within the framework of classical relational theory' [GV85, p. 55]). We feel that this statement devolves from a too narrow view of tuple-timestamped data models; it does not apply in our context.

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.

3.5 Temporal Interpolation

The topic of temporal interpolation, how to derive information about times for which no information is stored, has occasionally surfaced in temporal database papers. Here we first consider this background, then present our study of aspects of the topic.

3.5.1 Background

In the early 1980's, several researchers studied independently the notion of temporal interpolation. In the context of a Pascal-based extension of the ER model to include the time dimension, Klopprogge [Klo81, KL83] discussed so-called "derivation functions" (also termed "induction formulas and operators"), to be used for inferring values valid at times when no values were explicitly stored in the database. The "missing" values are inferred from stored ones, e.g., at neighbor points. If a procedure can be given that computes such missing values precisely, the procedure was termed a "derivation;" if the procedure can only approximate the missing values, it was termed an "approximation." Independently, and contemporaneously, Clifford and Warren [CW83] explored the same concepts in a mathematical and less practice-oriented context. In their own terminology, they considered different "continuity assumptions". For example, with the step-function continuity assumption, a value holds until a new value is explicitly recorded, or until the object ceases to exist. More recently, Segev and Shoshani [SS87, SS93] incorporated "interpolation functions" (their term) directly into their time series data model; their papers lists four such functions: step-wise constant, continuous, discrete, and user-defined.

The use of these properties in logical database design was unexplored. In particular, does enumerating these properties impact the logical design? Segev, with collaborators, had used the notions quite effectively in producing efficient representations of time sequence collections [SS88a, SS88b, SC92] (in fact, Illustra's TimeSeries Datablade is based on this); what are the implications for temporal databases in general, and specifically for data semantics? To get a handle on this, we initially examined valid-time derivation. At the end of this section, we consider the effects of including transaction time.

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

EXAMPLE: Consider the two relations in Figure 11. The first, `empSal`, records names and salaries of employees, and the second, `expTemp`, records names and temperature measurements for experiments. Attributes `EmpS` and `ExpS` record surrogates representing employees and experiments, respectively.

EmpS	EName	Sal	T
e1	Tom	30k	$\{1, \ldots, 9\}$
e1	Tom	32k	$\{10, \ldots, 19\}$
e1	Tom	36k	$\{30, \ldots, 39\}$
e1	Tom	40k	$\{40, \ldots, 49\}$

(a) `empSal`

ExpS	Exp	Temp	T
x1	Exp1	25	$\{5, 65\}$
x1	Exp1	27	$\{15\}$
x1	Exp1	31	$\{25\}$
x1	Exp1	29	$\{35\}$
x1	Exp1	27	$\{45\}$
x1	Exp1	26	$\{55\}$

(b) `expTemp`

Figure 11: Sample Valid-time Relations

Relation `empSal` records Tom's salaries at all the times he has a salary. This is clearly consistent with what a valid-time relation is. Relation `expTemp` is different in this regard and is perhaps more problematic. It does not record temperatures for all the times when there exists a temperature for experiment x1. The temperature of x1 is sampled regularly, and we may later want to estimate x1 temperature values for times with no explicitly recorded value.

The difference between relations such as `empSal` and `expTemp` is solely in what additional, or even different, information is implied. In relation `empSal`, no salary is recorded for Tom from time 20 to time 29, and the existing tuples do not imply any salary for Tom in that time interval. However, while no temperature for Exp1 at time 40 is recorded in `expTemp`, such a temperature does exist. Thus, the difference is that different interpolation functions apply to the salary and temperature attributes of the two relations. □

We explore several interesting aspects of interpolation in the following sections.

3.5.2 Derivation Functions

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: 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 $\mathcal{D}_D$.

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

Next, we introduce three important types of derivation functions in turn, namely stepwise-constant, discrete, and nearest-neighbor derivation functions. To do so, we need the following concept.

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

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

$$f_{A\text{-sc}}(c^v, r) = \begin{cases} t_1[A] \text{ where } t_1 \in r & \text{if } \exists c_1^v \in t_1[T]\ (c_1^v \leq c^v \wedge \\ & \quad \neg(\exists t_2 \in r\ (t_2[A] \notin \{\perp, \perp_u\} \wedge \exists c_2^v \in t_2[T]\ (c_1^v < c_2^v \leq c^v)))) \\ \perp_i & \text{otherwise} \end{cases}$$

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

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

We associate a step-wise constant derivation function with attribute `Sal`. Thus, $f_{\texttt{Sal-sc}}(5, \texttt{empSal1}) =$ 30k, $f_{\texttt{Sal-sc}}(10, \texttt{empSal1}) =$ 32k, $f_{\texttt{Sal-sc}}(15, \texttt{empSal1}) =$ 32k, 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 Tom in `empSal`, yet uses instant timestamps. □

One can add a derivation operator to the algebra, similar in spirit to Klug's aggregate formation operator [Klu82]. Such an operator would take as a subscript the derivation operator to be applied. Here we take a more informal approach to emphasize intuition.

Example: To exemplify the derivation operator, we apply it using the stepwise-constant derivation function to the `empSal1` relation above. The following table, `empSal2`, results.

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

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.

$$\delta_{\texttt{DSal}\rightarrow\texttt{Sal}}(\pi^{\mathrm{V}}_{\texttt{EmpS,EName,DSal}}(\sigma^{\mathrm{V}}_{\texttt{DSal}\neq\bot_i}(\texttt{empSal2}))) \equiv \texttt{empSal}$$

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* derivation function, defined as follows.

DEFINITION: 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.

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

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

3.5.3 Interpolation Functions

Interpolation functions preserve the information content of the relations they are applied to and are special cases of derivation functions that are not restricted in this regard. Discrete interpolation functions may be used for precisely characterizing those derivation functions that are also interpolation functions.

DEFINITION: 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 $f_{A\text{-d}}(c^v, r) \neq \bot$, the condition $f_A(c^v, r) = f_{A\text{-d}}(c^v, r)$ is satisfied. □

It follows that the discrete and step-wise constant derivation functions are interpolation functions.

The *nearest-neighbor-interpolation* interpolation function (denoted by $f_{A\text{-nn}}$) is appropriate for the `Temp` attribute of `expTemp` in Figure 11(b). As for the previous two interpolation functions, it applies to a particular (numeric-valued) attribute, e.g., A of a valid-time relation. 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.

3.5.4 Further Properties of Derivation Functions

Next, we introduce derivation functions that produce 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: 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: Consider the relation instance `expTemp` in Figure 11(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 application of some derivation function, $f_{\texttt{Temp}}$ for the temperature attribute on this relation. 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. □

Stability is often too strong a property. In some situations, it is sufficient to ensure only that the interpolated value at a particular time *improves* as more information is utilized by the interpolation function.

EXAMPLE: 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. □

3.5.5 Interpolation in Database Design

In previous work, interpolations such as step-wise constant were specified in the meta-data, then utilized by the query language; or they were hard-wired into the data model. We were uneasy about this, as our work with scientific databases told us that interpolations are subject to change (a good example is an equipment re-calibration); there are also often several different interpolation functions one might want to apply to the same data.

For each time-varying attribute, a set of perhaps several derivation functions may be relevant. 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 accommodate several derivation functions. 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 new derivation functions or remove or modify existing derivation functions.

The view mechanism is an ideal solution that maintains this separation. Thus, the database designer first identifies which sets of derivation functions that should be applied simultaneously to the attributes of a logical relation instance, one function per attribute. The designer then defines a view for each such set. It is feasible, though not easy, to express such view definitions in SQL, assuming the support by the DBMS of user-defined aggregate operators (several commercial DBMS's provide this facility). With the temporal aggregates in TSQL2 [KSL95], and the availability of user-defined aggregate functions, such views are straightforward to express.

3.5.6 Interpolation in Transaction Time

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, e.g., as an interval timestamp. The stepwise-constant interpolation function is used if facts are stamped with their insertion times only, as an instant timestamp. Which single interpolation function is appropriate is determined by the adopted temporal data model. All transaction-time data models known to us employ either discrete or stepwise-constant transaction-time interpolation. We see that transaction-time interpolation is then not relevant for database design.

3.6 Additional Decomposition

To this point, we have seen how most of the aspects associated with attributes enter into the design process, and more fundamentally, into the semantics of the relations in which they participate. There are two aspects that remain to be examined: whether transaction-time (or valid-time) support is needed at all and the granularity of the observation patterns.

3.6.1 Temporal Support of Attributes

During database design, a model of a part of reality is created. What aspects of the modeled reality to capture and what to leave out is determined by the functional requirements to the application being created. The application may require any combination of valid-time and transaction-time support, or no temporal support, for each of the time-varying attributes.

Next, attributes may be either state based or event based. Values of state-based attributes are valid for durations of time while values of event-based attributes are valid only for instants in time.

Combining these alternatives, there are six possibilities for the temporal support required for a time-varying attribute.

$$\left\{ \begin{array}{c} \underline{\textbf{Valid-time:}} \\ \text{no support required} \\ \text{state support required} \\ \text{event support required} \end{array} \right\} \times \left\{ \begin{array}{c} \underline{\textbf{Transaction-time:}} \\ \text{no support required} \\ \text{support required} \end{array} \right\}$$

The characterization of attributes according to the temporal support they require is important for database design because data models generally support only one type of temporal support in a single relation. We embed this requirement in a simple decomposition rule.

Definition: (*Temporal Support Decomposition Rule*) To achieve the correct temporal support of time-varying attributes, decompose temporal relation schemas to have only attributes with the same temporal support requirements in the same schema, except for the surrogate attribute(s) forming the primary key. □

EXAMPLE: Consider a relation schema with four attributes, `Name` (a surrogate attribute which is the primary key), `PromotedBy`, `Salary`, and `Dept`. All four require valid-time support, but the `PromotedBy` attribute is associated with the promotion *event*, while the last two attributes are associated with *states*. The promotion event is the transition between two salary and/or department states. The Temporal Support Decomposition Rule requires that this schema be decomposed into two schemas, (`Name`, `PromotedBy`), a valid-time event relation, and (`Name`, `Salary`, `Dept`), a valid-time state relation. □

It may be possible to avoid such decomposition in certain circumstances, but the designer should be aware of the potential drawbacks of doing so. Consider including an attribute S requiring snapshot support together with an attribute T requiring transaction-time support, in a transaction-time relation. Attribute S should have a single value over all time. However, since it is embedded in a transaction-time relation, past values are automatically retained. Taking the transaction timeslice at *now* produces the correct result, but taking a transaction timeslice at a time in the past, at time $c_t < now$, may retrieve an old value of S, which is inconsistent with the requirement that it be a snapshot attribute. Such queries must take this into account, timeslicing the relation as of *now* to get the value of S, then join this with the timeslice of the relation as of c_t to get the value of T, which is quite awkward. Another drawback is physical in nature: the fact that old values of S are being retained may increase the storage demand.

Including an attribute S along with an attribute V requiring valid-time support is even more problematic. Whereas the system provides the transaction time during modifications, the user must provide the valid time. This raises the issue of what should the valid time be for the snapshot attribute S. All updates have to maintain this semantics, and queries also have to consider the valid time.

Temporal specialization, discussed in Section 2.1, provides flexibility and can reduce the need for decomposition. Consider an attribute D that requires both valid-time and transaction-time support, but which is degenerate, i.e., the valid and transaction times are exactly correlated. Whenever a change occurs in the modeled reality, the new data is immediately recorded in the database. Say this attribute is associated with another attribute V requiring only valid-time support. The two attributes D and V can comfortably co-reside in the relation's schema. In this case, the relation will be a valid-time relation. Transaction-time queries can be recast as valid-time queries on the relation, exploiting the correlation between the two kinds of time. A similar situation holds if D is paired with an attribute T requiring only transaction-time support, with the relation being in this case a transaction-time relation. When the correlation is less precise, as with, for example, a delayed strongly retroactively bounded warehouse relation, pairing a specialized bitemporal attribute with either a valid-time or a transaction-time attribute will eliminate some queries, and so should be justified on performance grounds instead of logical semantic concerns.

3.6.2 Temporal Precision of Attributes

Each time-varying attribute has an associated observation pattern, as discussed in Section 3.4. A time pattern is a function to a time domain, that has an associated time granularity. The granularity is the precision in which the time-variance is recorded. If a hiring decision occurred sometime during the business day, but it is not known exactly when (i.e., what minute or hour) the decision occurred, then it is inappropriate to store that fact with a timestamp at a minute granularity. The reason is that a particular minute must be chosen, and that minute is probably incorrect, with the implication that the model is incorrect [DS93b].

This property of time-varying attributes is important for database design because temporal relational data models and query languages are frequently based on the (sometimes implicit) assumption that all time-varying attributes of a relation may be recorded with the same precision. For example, in tuple timestamped models, the time-variance of all attribute values is recorded with a single timestamp attribute (or the same set of timestamp attributes).

One approach is to use the minimum granularity of the DBMS at the precision of all relations. As just discussed, this results in a low-fidelity model of reality. A better approach is to choose the most appropriate granularity for each relation. We propose a simple strategy. Associate with each attribute a set of granularities. The smallest granularity in this set is the granularity in which the time-variance of the attribute is known. Other, coarser granularities represent granularities which are acceptable to the applications utilizing the relation. Then decompose the relation only if there is not a common granularity that is a member of the granularity sets of all attributes.

DEFINITION: (*Precision Decomposition Rule*) To accurately reflect the temporal precisions of time-varying attributes, decompose relation schemas so that all attributes in a schema have a compatible temporal precision, that is, a common granularity. □

EXAMPLE: Continuing the previous example, the observation pattern for `Salary` is at a granularity of *minute*. However, it is acceptable for applications if the timestamps associated with this attribute are stored at the coarser granularity of *day*, yielding a set of granularities for this attribute of {*minute*, *day*}. The observation pattern for `Dept` is at a granularity of day, and coarser granularities are not acceptable to applications, yielding a set of granularities for this attribute of {*day*}. The Precision Decomposition Rule enables these two attributes to remain together in the relation, which will have a timestamp granularity of *day*. □

A more general approach was recently proposed by Wang and his colleagues, using their temporal functional dependencies based on granularities, discussed briefly in Section 3.1.3 [WBBJ94]. Their approach is complex and may generate new granularities, of uncertain comprehensibility by the user. The Precision Decomposition Rule above is very simple and does not generate new granularities.

3.7 Summary

Below, we review the concepts introduced in this section and briefly indicate how they may be used for capturing the semantics of time-varying attributes during database design.

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

We 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.

These concepts are applied during logical database design. The database designer starts with a set of relation schemas in hand. The first task is to refine the semantics of each attribute appearing in the schemas.

- *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, describe the lifespans of the attributes.
- *Determine observation and update patterns.* For each relation schema, indicate which attributes are synchronous, i.e., share observation and update patterns.
- *Describe precisions.* For each time-varying attribute, indicate its set of applicable granularities.
- *For each attribute, indicate its appropriate derivation or interpolation function(s).* The functions concern interpolation in valid-time, and there may be several functions per attribute.
- *Determine the required temporal support.* For each attribute, indicate the required temporal support for the attribute. Record the interactions (if any) between the valid time and the transaction time implied by the temporal specializations in effect for the attribute.
- *Specify temporal functional dependencies.* These provide the basis for applying the temporal extensions of the conventional normal forms for schema decomposition.
- *Specify strong temporal functional dependencies.*

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 redundancy. Logical temporal database design accomplishes this by applying the available decomposition rules.

The information listed above guides the logical and physical design of relation schemas and the design of views.

- Temporal functional dependencies may be used to achieve the temporal analogues of traditional normal forms, e.g., third normal form, BCNF, fourth normal form. All standard database design approaches apply here directly.
- The lifespan decomposition rule ensures that inapplicable nulls are not required.
- The synchronous decomposition rule removes redundant attribute values, while being less strict than previous definitions of value synchronism.
- The temporal support decomposition rule ensures that each relation has a temporal support appropriate for the attributes it contains.
- The precision decomposition rule uses the granularity sets to prescribe decomposition of relation schemas and to determine the granularity of the resulting relation schemas.
- Strong temporal functional dependencies, together with the temporal functional dependencies, allow the designer to identify time-invariant keys, which may play the role of surrogates, which can subsequently be eliminated.
- The derivation function associated with attributes induce views computing the derived values.

While logical design is concerned with adequately modeling the *semantics* of the application, physical design is concerned with performance. The concepts concerning synchronism, i.e., time patterns, including observation and update patterns, are relevant for physical design. Their use was discussed in Section 3.4.2.

Physical design may also reverse some of the decomposition that is indicated by logical design. Database designers are faced with a number of design criteria which are sometimes conflicting, making database design a challenging task. So, while we discussed the 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 need for joins.

4 Status and Outlook

At this point in time, we feel that the semantics of temporal relational schemas and their logical design are well understood. In this paper, we have presented the general concepts of temporal specialization and generalization, the Bitemporal Conceptual Data Model, the impact of temporal coalescing, now and forever on temporal semantics, temporal functional dependencies (including parameterized and strong variants), surrogates, attributes lifespans, pattern-based synchronism, temporal interpolation, and several forms of decomposition based on temporal semantics.

We feel that several aspects merit further study. It is not yet clear how to best integrate the logical and conceptual design of temporal databases, two research areas that have hitherto evolved with minimal interaction. The third design area, physical temporal database design, is still in its infancy, with few results. A coherent design methodology for temporal databases, including conceptual (implementation-data-model independent) design and logical design, is needed. A methodology for physical design that takes into account available storage formats and primary and secondary indexing techniques still remains to be devised. In particular, the role of synchronous decomposition in physical design needs to be explored in greater detail. The methodology should be validated with actual applications. Finally, the ideas presented here and the methodology that will follow should be transitioned to existing implementation platforms, including non-temporal query languages such as SQL-92 [MS93]. In the short and perhaps even medium term, it is unrealistic to assume that applications will be designed using a temporal data model with new design methodologies, implemented using new temporal query languages, and run on new temporal DBMSs.

5 Acknowledgments

We would like to thank Jim Clifford, Curtis Dyreson, Tommy Isakowitz, Nick Kline, Keun Ho Ryu, Arie Segev, and Mike Soo for the many discussions with us over the past five years on the topics addressed in this paper.

The first author was supported in part by the Danish Natural Science Research Council, grants 11–1089–1, 11–0061–1, and 9400911. The second author was supported in part by NSF grants ISI-8902707 and ISI-9202244 and by grants from IBM, the AT&T Foundation, and DuPont.

References

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

[Ari86] G. Ariav. A Temporally Oriented Data Model. *ACM Transactions on Database Systems*, 11(4):499–527, December 1986.

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

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

[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. IEEE Computer Society, IEEE Computer Society Press.

[CC93] J. Clifford and A. Croker. *The Historical Relational Data Model (HRDM) Revisited*, Chapter 1, pages 6–27. In A. Tansel et al., editors, *Temporal Databases,* Benjamin/Cummings, 1993.

[CCT94] J. Clifford, A. Croker, and A. Tuzhilin. On Completeness of Historical Relational Query Languages. *ACM Transactions on Database Systems*, 19(1):64–116, March 1994.

[CDI+94] J. Clifford, C. Dyreson, T. Isakowitz, C. S. Jensen, and R. T. Snodgrass. On the Semantics of "now" in Temporal Databases. Technical Report R-94–2047, Aalborg University, Department of Mathematics and Computer Science, Aalborg, Denmark, November 1994.

[CFP84] M. A. Casanova, R. Fagin, and C. H. Papadimitriou. Inclusion Dependencies and Their Interaction with Functional Dependencies. *Journal of Computer and System Sciences*, 28(1):29–59, 1984.

[CK93] S. Chakravarthy and S. K. Kim. Semantics of Time-varying Information and Resolution of Time Concepts in Temporal Databases. In R. T. Snodgrass, editor, *Proceedings of the International Workshop on an Infrastructure for Temporal Databases*, Arlington, TX, June 1993.

[CK94] S. Chakravarthy and S. K. Kim. Resolution of Time Concepts in Temporal Databases. *Information Sciences*, 80(1/2):91–125, September 1994.

[Cod79] E. F. Codd. Extending the Database Relational Model to Capture More Meaning. *ACM Transactions on Database Systems*, 4(4):397–434, December 1979.

[CT85] J. Clifford and A. U. Tansel. On an Algebra for Historical Relational Databases: Two Views. In S. Navathe, editor, *ACM SIGMOD International Conference on the Management of Data*, pages 247–265, Austin, TX, May 1985.

[CT90] A. Colijin and P. Thompson. A Temporal Data Model Based on Accounting Principles. In *Proceedings of the International Conference on Computing and Information*, Toronto, Canada, 1990.

[CW83] J. Clifford and D. S. Warren. Formal Semantics for Time in Databases. *ACM Transactions on Database Systems*, 8(2):214–254, June 1983.

[DF92] C. J. Date and R. Fagin. Simple Conditions for Guaranteeing Higher Normal Forms in Relational Databases. *ACM Transactions on Database Systems*, 17(3):465–476, September 1992.

[DS93b] C. E. Dyreson and R. T. Snodgrass. Valid-time Indeterminacy. In *Proceedings of the International Conference on Data Engineering*, pages 335–343, Vienna, Austria, April 1993.

[EGS93] O. Etzion, A. Gal, and A. Segev. Temporal Active Databases. In R. T. Snodgrass, editor, *Proceedings of the International Workshop on an Infrastructure for Temporal Databases*, Arlington, TX, June 1993.

[EN89] R. Elmasri and S. B. Navathe. *Fundamentals of Database Systems*. The Benjamin/Cummings Publishing Company, Inc., 1989.

[Eva90] C. Evans. The Macro-Event Calculus: Representing Temporal Granularity. In *Proceedings of PRICAI*, Tokyo, Japan, 1990.

[EWK93] R. Elmasri, G. Wuu, and V. Kouramajian. *A Temporal Model and Query Language for EER Databases*, Chapter 9, pages 212–229. In A. Tansel et al., editors, *Temporal Databases,* Benjamin/Cummings, 1993.

[Fag77] R. Fagin. Multivalued Dependencies and a New Normal Form for Relational Databases. *ACM Transactions on Database Systems*, 2(3):262–278, September 1977.

[Fag79] R. Fagin. Normal Forms and Relational Database Operators. In *ACM SIGMOD International Conference on the Management of Data*, 1979.

[Fag81] R. Fagin. A Normal Form for Relational Databases That is Based on Domains and Keys. *ACM Transactions on Database Systems*, 6(3):387–415, September 1981.

[Fin92] M. Finger. Handling Database Updates in Two-dimensional Temporal Logic. *Journal of Applied Non-Classical Logics*, 2(2), 1992.

[Gad86] S. K. Gadia. Weak Temporal Relations. In *ACM SIGAct-SIGMOD Principles on Database Systems*, Los Angeles, CA, 1986.

[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.

[Gad93] S. K. Gadia. *Ben-Zvi's Pioneering Work in Relational Temporal Databases*, Chapter 8, pages 202–207. In A. Tansel et al., editors, *Temporal Databases,* Benjamin/Cummings, 1993.

[GES94] A. Gal, O. Etzion, and A. Segev. Representation of Highly Complex Knowledge in a Database. *Journal of Intelligent Information Systems*, 3(2):185–203, March 1994.

[GNP92] S. K. Gadia, S. Nair, and Y.-C. Poon. Incomplete Information in Relational Temporal Databases. In *International Conference on Very Large Databases*, Vancouver, Canada, August 1992.

[GV85] S. K. Gadia and J. H. Vaishnav. A query language for a homogeneous temporal database. In *ACM SIGAct-SIGMOD Principles on Database Systems*, pages 51–56, March 1985.

[GY88] S. K. Gadia and C.-S. Yeung. A Generalized Model for a Relational Temporal Database. In *Proceedings of the ACM SIGMOD '88*, pages 251–259, 1988.

[GY91] S. K. Gadia and C.-S. Yeung. Inadequacy of Interval Timestamps in Temporal Databases. *Information Sciences*, 54:1–22, 1991.

[HM81] M. Hammer and D. McLeod. Database Description With SDM: A Semantic Database Model. *ACM Transactions on Database Systems*, 6(3):351–386, September 1981.

[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.

[HS93] S. Hsu and R. T. Snodgrass. Optimal Block Size for Set-valued Attributes. *Information Processing Letters*, 45(3):153–158, March 1993.

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

[JM92] C. S. Jensen and L. Mark. Queries on Change in an Extended Relational Model. *IEEE Transactions on Knowledge and Data Engineering*, 4(2):192–200, April 1992.

[JM93] C. S. Jensen and L. Mark. *Differential Query Processing in Transaction-Time Databases*, chapter 19, pages 457–491. In A. Tansel et al., editors, *Temporal Databases,* Benjamin/Cummings, 1993.

[JMR91] C. S. Jensen, L. Mark, and N. Roussopoulos. Incremental Implementation Model for Relational Databases with Transaction Time. *IEEE Transaction on Knowledge and Data Engineering*, 3(4):461–473, December 1991.

[JMRS93] C. S. Jensen, L. Mark, N. Roussopoulos, and T. Sellis. Using Differential Techniques to Efficiently Support Transaction Time. *The VLDB Journal*, 2(1):75–111, January 1993.

[JS94a] C. S. Jensen and R. T. 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.

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

[JSS96] C. S. Jensen, R. T. Snodgrass, and M. D. Soo. Extending Existing Dependency Theory to Temporal Databases. *IEEE Transaction on Knowledge and Data Engineering*, to appear, 1996.

[KC93] S.-K. Kim and S. Chakravarthy. Modeling Time: Adequacy of Three Distinct Time Concepts for Temporal Databases. In *Conference on Entity-Relationship Approach*, December 1993.

[KL83] M. R. Klopprogge and P. C. Lockemann. Modelling Information Preserving Databases: Consequences of the Concept of Time. In M. Schkolnick and C. Thanos, editors, *International Conference on Very Large Databases*, pages 399–416, Florence, Italy, 1983.

[Klo81] M. R. Klopprogge. Term: An Approach to Include the Time Dimension in the Entity-Relationship Model. In *Proceedings of the Second International Conference on the Entity Relationship Approach*, pages 477–512, Washington, DC, October 1981.

[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.

[KSL95] N. Kline, R. T. Snodgrass, and T. Y. Cliff Leung. Aggregates. Chapter 21, pages 395–425. In R. T. Snodgrass, editor, *The TSQL2 Temporal Query Language*, Kluwer Academic Publishers, September 1995.

[MMCR92] A. Montanari, E. Maim, E. Ciapessoni, and E. Ratto. Dealing with Time Granularity in the Event Calculus. In *Proceedings of the International Conference on Fifth Generation Computer Systems 1992*, volume 2, pages 702–712, Tokyo, Japan, June 1992. ICOT.

[Mon73] R. Montague. *The Proper Treatment of Quantification in Ordinary English*. D. Reidel Publishing Co., Dordrecht, Holland, 1973.

[MS90] E. McKenzie and R. T. Snodgrass. Schema Evolution and the Relational Algebra. *Information Systems*, 15(2):207–232, June 1990.

[MS91a] E. McKenzie and R. T. Snodgrass. An Evaluation of Relational Algebras Incorporating the Time Dimension in Databases. *ACM Computing Surveys*, 23(4):501–543, December 1991.

[MS91b] E. McKenzie and R. T. Snodgrass. Supporting Valid Time in an Historical Relational Algebra: Proofs and Extensions. Technical Report TR–91–15, Department of Computer Science, University of Arizona, Tucson, AZ, August 1991.

[MS93] J. Melton and A. R. Simon. *Understanding the New SQL: A Complete Guide*. Morgan Kaufmann Publishers, Inc., San Mateo, CA, 1993.

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

[ND96] M. A. Nascimento and M. H. Dunham. Indexing a Transaction-decision Time Database. In *Proceedings of the ACM Eleventh Annual Symposium on Applied Computing*, February 1996.

[NE95] M. A. Nascimento and M. Eich. On Decision Time for Temporal Databases. In *Proceedings of the Second International Workshop on Temporal Representation and Reasoning*, pages 157–162, 1995.

[ÖS95] G. Özsoyoğlu and R. T. Snodgrass. Temporal and Real-time Databases: A Survey. *IEEE Transaction on Knowledge and Data Engineering*, 7(4), August 1995.

[Ram92] K. Ramamritham. Real-time Databases. *International Journal of Distributed and Parallel Databases*, 1992.

[Re87] L. A. Rowe and M. R. Stonebraker, editors. The Postgres Papers. Memorandum, UCB/ERL M86/85, University of California, Electronics Research Laboratory, College of Engineering, University of California, Berkeley, CA 94720, June 1987.

[Ris77] J. Rissanen. Independent Components of Relations. *ACM Transactions on Database Systems*, 2(4):317–325, December 1977.

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

[SA85] R. T. Snodgrass and I. Ahn. A Taxonomy of Time in Databases. In S. Navathe, editor, *ACM SIGMOD International Conference on the Management of Data*, pages 236–246, Austin, TX, May 1985.

[SA86] R. T. Snodgrass and I. Ahn. Temporal Databases. *IEEE Computer*, 19(9):35–42, September 1986.

[Sad80] F. Sadri. *Data Dependencies in the Relational Model of Data: A Generalization.* Ph.D. thesis, Princeton University, October 1980.

[Sar90] N. L. Sarda. Extensions to SQL for Historical Databases. *IEEE Transactions on Knowledge and Data Engineering*, 2(2):220–230, June 1990.

[SC92] A. Segev and R. Chandra. *A Data Model for Time-Series Analysis.* Springer Verlag, 1992.

[SGM93] R. T. Snodgrass, S. Gomez, and E. McKenzie. Aggregates in the Temporal Query Language TQuel. *IEEE Transaction on Knowledge and Data Engineering*, 5:826–842, October 1993.

[SJS95] M. D. Soo, C. S. Jensen, and R. T. Snodgrass. An Algebra for TSQL2, Chapter 27, pages 505–546. In R. T. Snodgrass, editor, *The TSQL2 Temporal Query Language*, Kluwer Academic Publishers, September 1995.

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

[Sno93] R. T. Snodgrass. *An Overview of TQuel*, Chapter 6, pages 141–182. In A. Tansel et al., editors, *Temporal Databases,* Benjamin/Cummings, 1993.

[Sno95] R. T. Snodgrass, editor. *The Temporal Query Language TSQL2.* Kluwer Academic Pub., September 1995.

[SRH90] M. Stonebraker, L. Rowe, and M. Hirohama. The implementation of Postgres. *IEEE Transaction on Knowledge and Data Engineering*, 2(1):125–142, March 1990.

[SS77] J. M. Smith and D. C. P. Smith. Database Abstractions: Aggregation and Generalization. *ACM Transactions on Database Systems*, 2(2):105–133, June 1977.

[SS87] A. Segev and A. Shoshani. Logical Modeling of Temporal Data. In U. Dayal and I. Traiger, editors, *Proceedings of the ACM SIGMOD Annual Conference on Management of Data*, pages 454–466, San Francisco, CA, May 1987.

[SS88a] 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.

[SS88b] A. Segev and A. Shoshani. *The Representation of a Temporal Data Model in the Relational Environment*, volume 339, pages 39–61. Springer Verlag, 1988.

[SS93] A. Segev and A. Shoshani. *A Temporal Data Model Based on Time Sequences*, Chapter 11, pages 248–270. In A. Tansel et al., editors, *Temporal Databases,* Benjamin/Cummings, 1993.

[Sto87a] M. Stonebraker. The Design of the Postgres Storage System. In P. Hammersley, editor, *International Conference on Very Large Databases*, pages 289–300, Brighton, England, September 1987.

[Sto90] M. Stonebraker. The Postgres DBMS (video presentation abstract). In *ACM SIGMOD International Conference on the Management of Data*, Atlantic City, New Jersey, May 1990.

[SU82] F. Sadri and J. Ullman. Template Dependencies: A Large Class of Dependencies in Relational Databases and its Complete Axiomatization. *Journal of the ACM*, 29(2), April 1982.

[Tho91] P. M. Thompson. *A Temporal Data Model Based on Accounting Principles.* PhD thesis, Department of Computer Science, University of Calgary, Calgary, Alberta, Canada, March 1991.

[WBBJ94] X. Wang, C. Bettini, A. Brodsky, and S. Jajodia. Logical Design for Temporal Databases with Multiple Granularities. ISSE-TR 94-111, ISSE Department, George Mason University, 1994.

[Wij95] J. Wijsen. Design of Temporal Relational Databases Based on Dynamic and Temporal Functional Dependencies. In S. Clifford and A. Tuzhilin, editors, *Recent Advances in Temporal Databases*, pages 61–76, Zurich, Switzerland, September 1995. Proceedings of the International Workshop on Temporal Databases, Springer Verlag.

[Zan76] C. Zaniolo. *Analysis and Design of Relational Schemata for Database Systems.* Ph.D. thesis, UCLA, July 1976.

[Zan82] C. Zaniolo. Database Relations with Null Values (extended abstract). In *ACM SIGAct-SIGMOD Principles on Database Systems*, pages 27–33, Los Angeles, CA, March 1982.