2

Semantics of Time-Varying Information

Christian S. Jensen and Richard T. Snodgrass

This paper provides a systematic and comprehensive study of the underlying semantics of temporal databases, summarizing selected results of an intensive collaboration between the two authors over the five-years period from 1991 to 1995. 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 timestamps from exactly one valid-time and one transaction-time dimension. 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.

1 Introduction and Historical Context

This paper summarizes selected results of an intensive collaboration between the two authors over the five-year period from 1991 to 1995 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 [20, 32, 44, 45, 49], and on temporal semantics, specifically characterizing the orthogonality of valid time and transaction time [48]. Christian had previously worked on transaction-time databases, specifically architecture [23, 25], implementation [24], and language support [22].

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 [7], 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., [33]), 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 tem-

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

poral 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 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 [48]? 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?

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, and the focus is on the fundamental question of how to associate time with facts, leading to an understanding of the nature of temporal data. For each 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 [38]. 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 [40]. 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 [21]; 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 [44]. 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 [47, 48]. *Valid time* concerns when a fact was true in the modeled reality [21]. *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

[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, as the first, say, 20 minutes of a “wedding macro event” is not itself a wedding [11, 35].

fact cannot extend beyond the current time (there is no foolproof 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.

3 Temporal Specialization

While valid time and transaction time had been shown to be orthogonal [48], 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” [52, 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 1 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).

`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 *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). □

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 are indeed orthogonal, their *use* in a particular application need 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, 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 supplied when assignments are terminated (more on this in Section 11). 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 [26]. 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

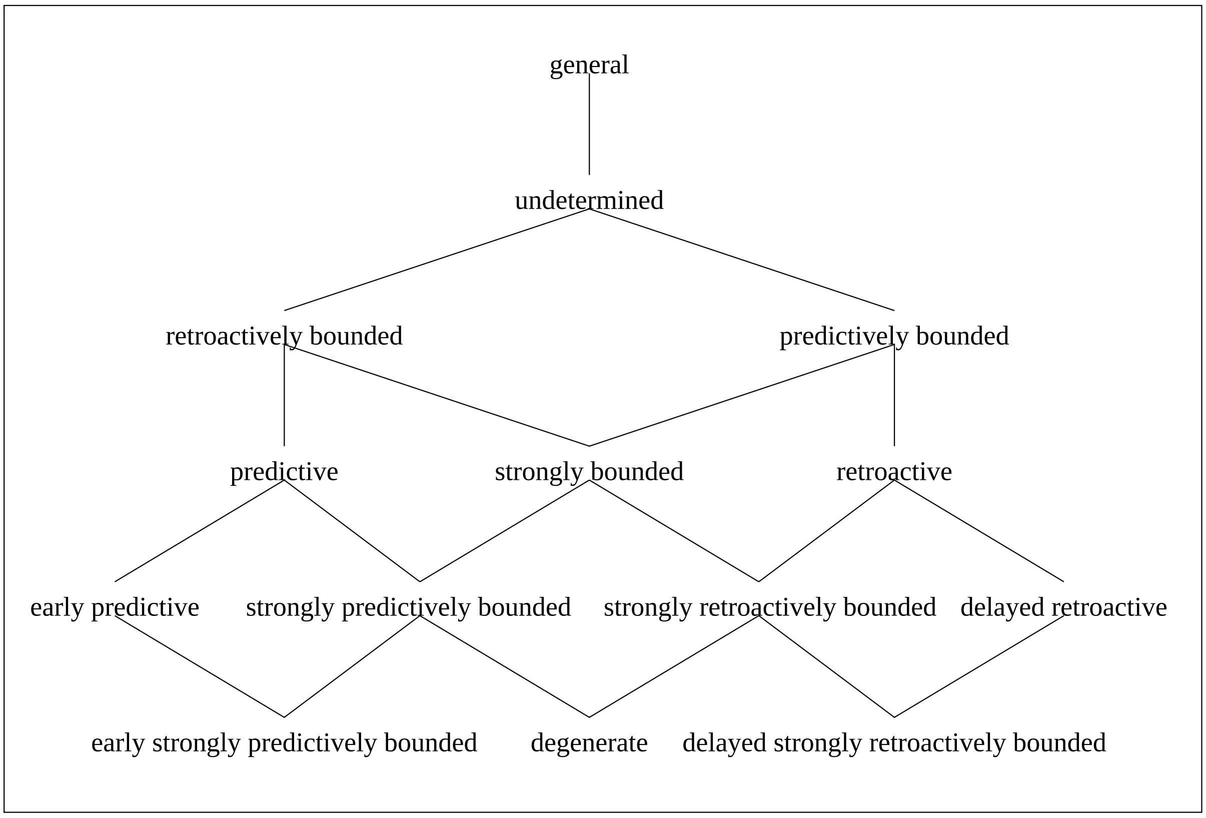


Figure 2: Generalization/Specialization Structure of the Taxonomy for Temporal Specialization

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 daily basis [30], the valid times of the point-of-sales records are between twenty-four hours and a few minutes 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." [1, 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 [16]. 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 August 31, 1997." 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 [14].

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 [42]. 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 [41, 51] 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 [52] 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.

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.

4 Temporal Generalization

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

Example 2 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., [3, 4, 10, 17, 29, 36, 37]). 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 [53]. When facts flow between temporal relations, several time dimensions may be associated with individual facts.

Example 3 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. In contrast to the previous discussion, where 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., [9, 19, 43]). 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 3.

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 [39]. 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 [47] allowed us to better understand Thompson's 4-time model [8, 53]. 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 the account associated with 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 dimension (either event or state) and one transaction time dimension.

5 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 [33], 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 [24, 31, 32, 44]. 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 [28], 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.

6 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 [27]. 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* [14].

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., [54]). 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 [18]. 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. Note that valid times larger than, as well as smaller than, the current time may be assigned to tuples, making it possible to record facts about the future (i.e., in effect predictions) and the past. 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* [21] 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 4 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).

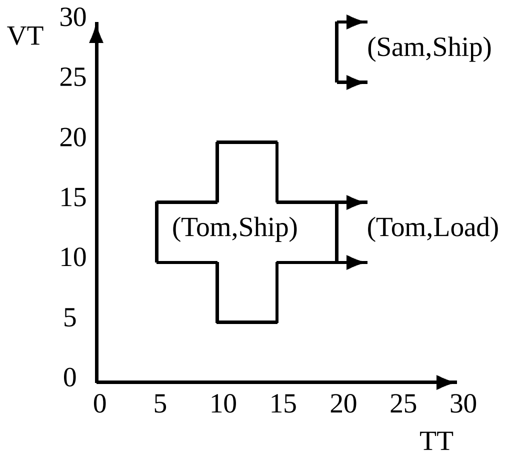
Figure 3(a) shows an instance, `empDep`, of this relation. The timestamp T is a set of bitemporal chronons. For the tuple (Tom, Shipping), the timestamp includes the bitemporal chronons (5, 10), (5, 11), through (5, 15), as well as the chronons (6, 10) through (6, 15), etc., totaling 140 bitemporal chronons. A graphical illustration of the `empDep` relation is shown in Figure 3(b). The axes are transaction time (horizontal) and valid time (vertical), abbreviated in the figure as TT and VT, respectively. Right-pointing arrows in the graph and the special value *UC* in the relation signify that the containing 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, Ship)," and "(Tom, Load)," respectively. The last tuple indicates when Sam worked for the Shipping department, and corresponds to the region of the graph labeled "(Sam, Ship)." □

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-*

`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

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.

7 Algebraic Operators in the BCDM

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 [50].

We first define bitemporal analogues of some of the snapshot relational operators, to be denoted with the superscript “$^{\text{B}}$”.

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

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

A new tuple variable, z, is used for holding the projected result tuples. 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 in z's timestamp. A bitemporal relation results.

Example 5 Consider the `empDep` relation shown in Figure 3(a). The following result is produced by $\pi^{\mathrm{B}}_{\texttt{Dept}}(\texttt{empDep})$.

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

All of the bitemporal chronons associated with a tuple with a `Dept` of Shipping are merged into one bitemporal element. Each chronon of this bitemporal element must be in the timestamp of at least one of the Shipping tuples in the underlying relation. □

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

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

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

Finally, we define two operators that select on valid time and transaction time. They have no counterparts in the snapshot relational algebra. Let c^v denote an arbitrary valid-time chronon and let c^t denote a transaction-time chronon. The *valid-timeslice* operator (τ^{B}) yields a transaction-time relation; the *transaction-timeslice* operator (ρ^{B}) evaluates to a valid-time relation[3].

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

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

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

from the type of the argument relation. Here, a time dimension has been removed while for projection, a number of explicit attributes were removed. Thus, the definitions must again employ a separate tuple variable (i.e., variable z) in order to construct the result tuples.

Example 6 Consider the `empDep` relation shown in Figure 3(a). The following result is produced by $\tau^{\mathrm{B}}_{12}(\texttt{empDep})$.

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. □

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 [28]. 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 [14], the valid or transaction timeslice of a relation will not introduce any nulls into the resulting relation.

8 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 [13]. 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 [28]. These data models fall into the class of temporally ungrouped bitemporal models [5] 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. 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 [46]). 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.

9 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.

What is the relationship between POSTGRES' two timestamps, TQuel's four timestamps, and Ben-Zvi's five timestamps? We showed in Section 3 that POSTGRES was a degenerate bitemporal data model, and thus a tuple's two timestamps *Tmin* and *Tmax* [51] 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 [2]. 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 $A_1, \ldots, A_n$, 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.

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

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.

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

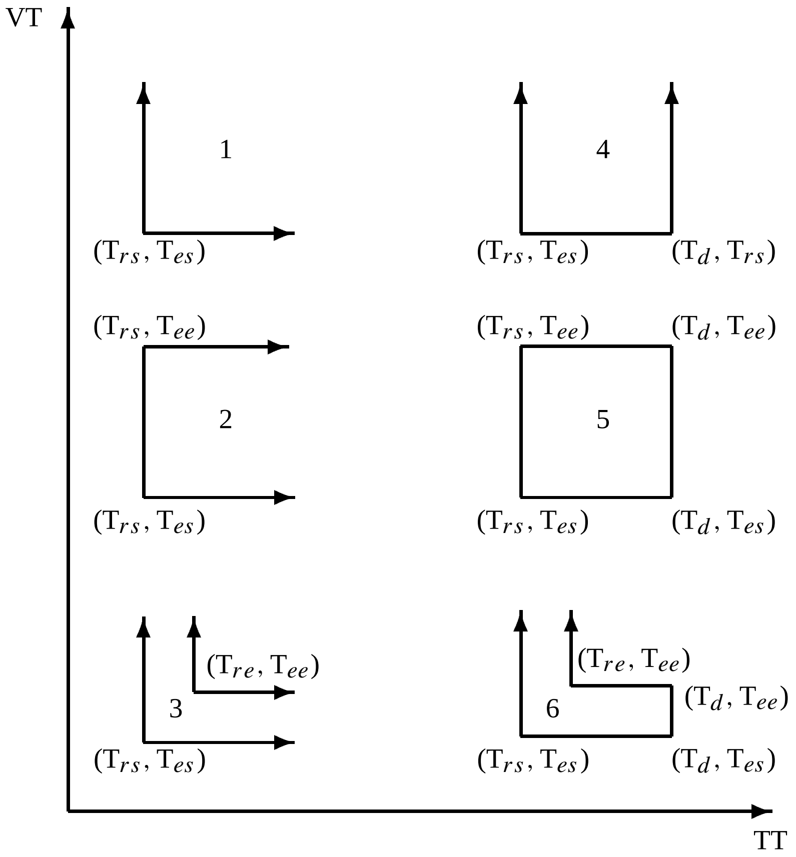


Figure 4: Ben-Zvi's Tuples as Bitemporal Elements

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 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., [10, 36]). This contrasts the definition that we use [21]. 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 [15] where T_{re} is said to be the end of transaction time!).

10 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 [28].

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 [44]. 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.

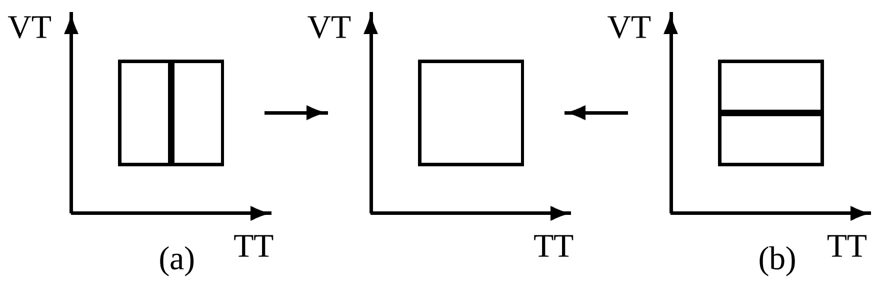


Figure 5: Coalescing

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 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.

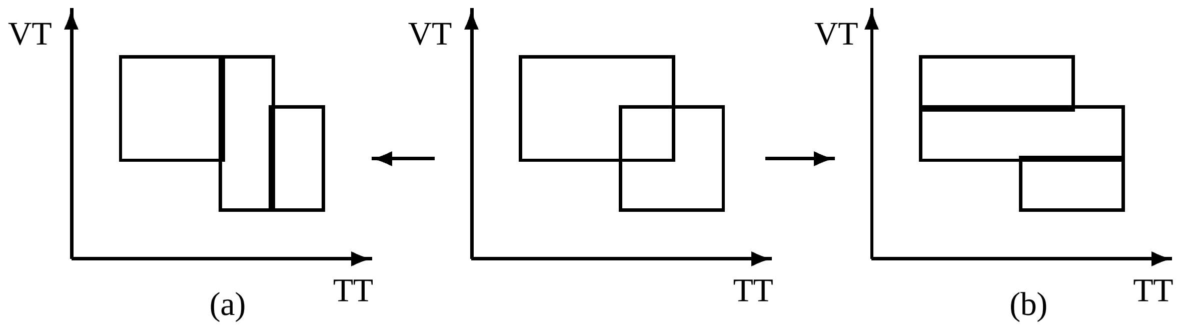


Figure 6: Eliminating Representational Repetition of Information

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 [28] for a formalization and proofs of these properties.)

11 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 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 [6]. *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 [34], and discussed more recently in the context of valid-time databases [12]. 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 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 [6].

12 Summary

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

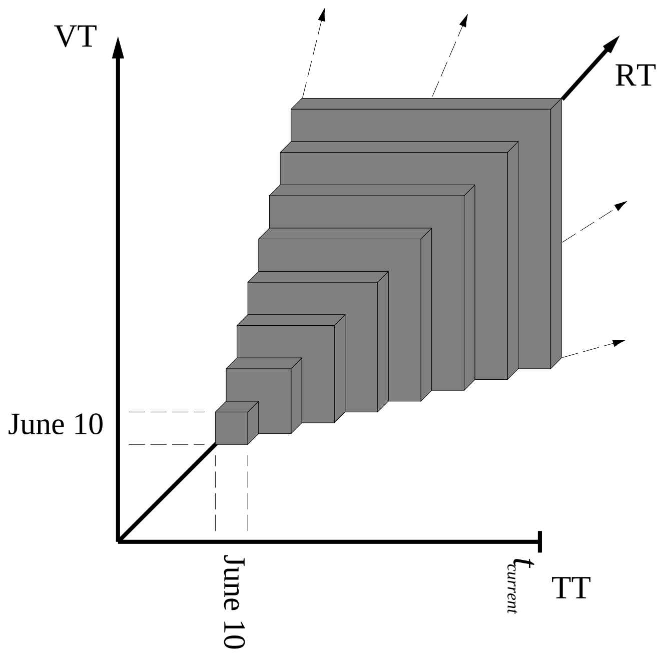


Figure 7: A Rectangle with a Transaction Stop Time and a Valid To Time of *NOW*

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 the bitemporal chronons 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 these data models

Acknowledgements

We would like to thank James Clifford, Curtis Dyreson, Tomás Isakowitz, Nick Kline, Keun Ho Ryu, Arie Segev, and Michael Soo for the many discussions with us over the past five years on the topics addressed in this paper, and Dennis Shasha for perceptive comments on the initial version of 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

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

[2] J. Ben-Zvi. *The Time Relational Model*. Ph.D. dissertation, Computer Science Department, UCLA (1982).

[3] 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 (1993).

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

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

[6] 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 (1994).

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

[8] 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).

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

[10] 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 (1993).

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

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

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

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

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

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

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

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

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

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

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

[22] 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 (1992).

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

[24] 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 (1991).

[25] 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 (1993).

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

[27] C. S. Jensen and R. T. Snodgrass. The Surrogate Data Type. Chapter 9, pp. 153–156. In R. T. Snodgrass, editor, *The TSQL2 Temporal Query Language*, Kluwer Academic Publishers (1995).

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

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

[30] R. Kimball. *The Data Warehouse Toolkit*. John Wiley and Sons. (1996).

[31] E. McKenzie. *An Algebraic Language for Query and Update of Temporal Databases*. Ph.D. dissertation, Department of Computer Science, University of North Carolina at Chapel Hill (September, 1988).

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

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

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

[35] 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*, pp. 702–712, Tokyo, Japan (1992).

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

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

[38] G. Özsoyoğlu and R. T. Snodgrass. Temporal and Real-time Databases: A Survey. *IEEE Transaction on Knowledge and Data Engineering*, **7**(4):513–532 (1995).

[39] K. Ramamritham. Real-time Databases. *International Journal of Distributed and Parallel Databases* (1992).

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

[41] 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 (1987).

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

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

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

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

[46] R. T. Snodgrass, editor. *The Temporal Query Language TSQL2*. Kluwer Academic Pub. (1995).

[47] 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*, pp. 236–246, Austin, TX (1985).

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

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

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

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

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

[53] P. M. Thompson. *A Temporal Data Model Based on Accounting Principles.* Ph.D. dissertation, Department of Computer Science, University of Calgary, Calgary, Alberta, Canada (1991).

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