The TSQL2 Data Model[*]

Christian S. Jensen Richard T. Snodgrass Michael D. Soo

1 Introduction

Adding time to the relational model has been a daunting task [BADW82, McK86, SS88b, Soo91]. More than two dozen time-extended relational data models have been proposed over the last fifteen years [Sno92]. Most of these are *valid-time* models. Each fact in a valid-time relation has associated the time when it is true in the modeled reality. Other models support *transaction-time* relations where each fact has associated the time when it is current in the database. A few support both valid and transaction time [BZ82, BG89a, Sno87, SGM93, Tho91]; such models are termed *bitemporal.* As a whole, these data models are referred to as *temporal* data models [JCE$^+$94].

We propose a new data model as a basis for the Temporal Structured Query Language (TSQL) extension to SQL. A data model can be said to consist of a query language, objects manipulated by the query language, an update language for updating the objects, and a mechanism for specifying integrity constraints. In this proposal, we concentrate on the objects, temporal relations. Subsequent proposals will address historical selection and projection, aggregates, and the other aspects necessary to define a comprehensive extension to SQL incorporating time.

While existing data models differ on many dimensions, perhaps the most frequently stated distinction is between tuple timestamping and first normal form (1NF), on one hand, and attribute-value timestamping and non-1NF, on the other. Each of the two approaches has associated difficulties. Remaining within 1NF (an example being the timestamping of tuples with valid and transaction start and end times [Sno87]) may introduce redundancy because attribute values that change at different times are repeated in multiple tuples. The non-1NF models, one being timestamping attribute values with sets of intervals [Gad88], may not be capable of directly using existing relational storage structures or query evaluation techniques that depend on atomic attribute values.

Today there exists a plethora of incompatible data models and query languages, with a corresponding surfeit of model- and language-specific database design and implementation strategies. It is our contention that the simultaneous focus on data *presentation* (how temporal data is displayed to the user), on data *storage*, with its requisite demands of regular structure, and on efficient *query evaluation* is a major reason why such a large number of very diverse data models exists. Further, we find that these simultaneous foci have complicated existing data models and made them less suited for the central task of capturing the time semantics of data.

Consequently, we advocate a very simple *conceptual,* unifying data model that captures the essential semantics of time-varying relations, but has no illusions of being suitable for presentation, storage, or query evaluation. For the other tasks, we are able to use the existing data models. Specifically, we use the notion of *snapshot equivalence* to demonstrate equivalence mappings between the conceptual model and several *representational* models [JSS92b]. Snapshot equivalence

[*]The authors have the following addresses and affiliations. C. S. Jensen is with Aalborg University, Datalogi, Fredrik Bajers Vej 7E, DK–9220 Aalborg Ø, Denmark, `csj@iesd.auc.dk`. R.T.Snodgrass and M. D. Soo are with Department of Computer Science, University of Arizona, Tucson, AZ 85721, USA, `{rts,soo}@cs.arizona.edu`. Copyright ©1994 Christian S. Jensen, Richard T. Snodgrass, and Michael D. Soo. All rights reserved.

formalizes the notion of having the same information contents and is a natural means of comparing rather disparate representations. Two relation instances are snapshot equivalent if all their snapshots, taken at all times (valid and transaction), are identical.

Facts in temporal relations (valid-time, transaction-time, or bitemporal) have associated times. Thus, in the next section, we start by examining the time domain itself. In Section 3, we then review, in turn, how times have previously been associated with facts of valid-time, transaction-time, and bitemporal relations. This review, and subsequent comparison, of 23 existing temporal data models provides context for presenting the new proposal. The basic notion of a bitemporal conceptual relation is presented in Section 5. In Section 6, we examine five bitemporal representational models and show mappings to and from bitemporal conceptual relations, as well as how the conceptual update semantics can be supported in the representational models. The semantic relationship between the conceptual model and the representational models is formalized in Sections 7 and 8. We summarize the proposal in Section 9.

2 The Time Domain

In this section, we focus on time and its semantics. In the next section, we show how previous proposals have combined time with facts to model time-varying information. We initially assume that there is one dimension of time. The distinctions addressed here will apply to each of the several dimensions considered later in this section.

Early work on *temporal logic* centered around two structural models of time, *linear* and *branching* [VB82]. In the linear model, time advances from the past to the future in a totally ordered fashion. In the branching model, also termed the *possible futures* model, time is linear from the past to now, where it then divides into several time lines, each representing a possible sequence of events [Wor90]. Along any future path, additional branches may exist. The structure of branching time is a tree rooted at now. The most general model of time in a temporal logic represents time as an arbitrary set with a partial order imposed on it. Additional axioms introduce other, more refined models of time. For example, linear time can be specified by adding an axiom imposing a total order on this set. Recurrent processes may be associated with a *cyclic* model of time [CI89, LJ88, Lor88].

Axioms may also be added to temporal logics to characterize the *density* of the time line [VB82]. Combined with the linear model, *discrete* models of time are isomorphic to the natural numbers, implying that each point in time has a single successor [CT85]. *Dense* models of time are isomorphic to either the rationals or the reals: between any two moments of time another moment exists. *Continuous* models of time are isomorphic to the reals, i.e., they are both dense and, unlike the rationals, contain no "gaps."

In the continuous model, each real number corresponds to a "point" in time; in the discrete model, each natural number corresponds to a nondecomposable unit of time with some fixed, arbitrary duration. Such a nondecomposable unit of time is referred to as a *chronon* [Ari86, CR87] (other, perhaps less desirable, terms include "time quantum" [And82], "moment" [AH85], "instant" [Gad86a] and "time unit" [NA87, TA86a]). A chronon is the smallest duration of time that can be represented in this model. It is not a point, but a line segment on the time line.

Although time itself is generally perceived to be continuous, most proposals for adding a temporal dimension to the relational data model are based on the discrete time model. Several practical arguments are given in the literature for this preference for the discrete model over the continuous model. First, measures of time are inherently imprecise [And82, CT85]. Clocking instruments invariably report the occurrence of events in terms of chronons, not time "points." Hence, events, even so-called "instantaneous" events, can at best be measured as having occurred during a chronon. Secondly, most natural language references to time are compatible with the discrete time model. For example, when we say that an event occurred at 4:30 p.m., we usually don't mean that the event occurred at the "point" in time associated with 4:30 p.m., but at some time in the chronon (perhaps minute) associated with 4:30 p.m. [And82, CR87, DS92a]. Thirdly, the concepts of chronon and interval allow us to naturally model events that are not instantaneous, but have duration [And82]. Finally, any implementation of a data model with a temporal dimension will of necessity have to have some discrete encoding for time.

Axioms can also be placed on the *boundedness* of time. Time can be bounded orthogonally in the past and in the future.

Models of time may include the concept of *distance* (most temporal logics do not do so, however). Time is a *metric*, in that it has a distance function satisfying four properties: (1) the distance is nonnegative, (2) the distance between any two non-identical elements is non-zero, (3) the distance from time α to time β is identical to the distance from β to α, and (4) the distance from α to γ is equal to or greater than the distance from α to β plus the distance from β to γ (the triangle inequality).

With distance and boundedness, restrictions on range can be applied. The scientific cosmology of the "Big Bang" posits that time begins with the Big Bang, 14 ± 4 billion years ago. There is much debate on when it will end, depending on whether the universe is *open* or *closed*. (Hawking provides a readable introduction to this controversy [Haw88].) If the universe is closed then time will have an end when the universe collapses back onto itself, in what is called the "Big Crunch." If it is open then time will go on forever.

Finally, one can differentiate *relative* time from *absolute* time (more precise terms are *unanchored* and *anchored*). For example, "9 a.m., January 1, 1992" is an absolute time, whereas "9 hours" is a relative time. This distinction, though, is not as crisp as one would hope because absolute time is with respect to another time (in this example, midnight, January 1, A.D. 1). Relative time differs from distance in that the former has a direction, e.g., one could envision a relative time of -9 hours, whereas a distance is unsigned.

Time is multi-dimensional [SA86]. *Valid time* concerns the time when a fact is true in reality. The valid time of an event is the wall clock time at which the event occurred in the modeled reality, independent of the recording of that event in some database. Valid times can be in the future, if it is known that some fact will become true at a specified time in the future. *Transaction time* concerns the time the fact was present in the database as stored data. The transaction time (a set of intervals) of an event identifies the transactions that inserted the information about the event into the database and removed this information from the database. Note that these two time dimensions are orthogonal. A data model supporting neither is termed *snapshot*, as it has no built-in support for any of these notions of time. A data model supporting only valid time is termed *valid-time*; one that supports only transaction time is termed *transaction-time*; and one that supports both valid and transaction time is termed *bitemporal* (*temporal* is a generic term implying some kind of time support [JCE$^+$94]).

While valid time may be bounded or unbounded (as we saw, cosmologists feel that it is at least bounded in the past), transaction time is always bounded on both ends. Specifically, transaction time starts when the database is created (before which time, nothing was stored), and does not extend past now (no facts are known to have been stored in the future). Changes to the database state are required to be stamped with the current transaction time. As the database state evolves, transaction times grow monotonically, and successive transactions have successive transaction times associated. In contrast, successive transactions may mention widely varying valid times.

Unlike the spatial dimensions, the two time dimensions are not homogeneous—transaction time has a different semantics than valid time. Valid and transaction time *are* orthogonal, though there are generally some application-dependent correlations between the two times. As a simple example, consider the situation where a fact is recorded as soon as it becomes valid in reality. In such a *specialized* bitemporal database, termed *degenerate* [JS92], the valid and transaction times of a fact are identical. As another example, if temperature measurements in a chemical experiment are recorded at most two minutes after they were measured, and if it takes at least five seconds from the measurement time to record the measurement, then such a database is *delayed strongly retroactively bounded with bounds five seconds and two minutes.*

3 Previous Data Models

The previous section explored models for the time domain itself. In this section, we discuss the association of facts with times. Specifically, we survey 23 existing data models that have been proposed over the last fifteen years. We consider each model in turn, starting with valid-time models, continuing with transaction-time models, and ending with bitemporal models. As a foundation, we initially define underlying concepts. Following the survey, we compare and categorize the data models with respect to fundamental design decisions.

3.1 Underlying Concepts

It is advantageous to examine several central concepts before each of the proposed data models are considered in turn.

3.1.1 Timestamp Types

We may distinguish between three semantically different types of time values, namely single chronons, sets of consecutive chronons, and arbitrary sets of chronons. These are termed *events*, *intervals*, and *temporal elements*, respectively [JCE^{+}94]. (We use consensus terminology in this commentary. The TSQL2 equivalents for events and intervals, used in the language definition and the remaining commentaries, are *datetimes* and *periods*, respectively. "Temporal element" is defined identically by TSQL2 and the consensus glossary.)

A single event may be represented by a single, atomic, chronon-valued attribute. An interval may be represented by a pair of atomic attribute values, each of which is a chronon or a point in time. If the later representation is adopted, the interval may be defined as open, half-closed, or closed. An interval may also be encoded in a single, atomic, interval-valued attribute. An arbitrary set of chronons may be represented by a non-atomic attribute value. This value may be a set of intervals, each interval defining a set of consecutive chronons, or it may simply be a set of chronons. Finally, sets of multiple chronons, consecutive or not, may be represented via multiple tuples, one tuple per chronon or one per interval.

This discussion applies to both transaction time, valid time, and the combination of valid and transaction time. For example, a *bitemporal element* is a set of *bitemporal chronons* in the transaction-time/valid-time space, and can be represented simply as a set of bitemporal chronons, as a set of contiguous or overlapping rectangles, or via multiple tuples, one tuple per bitemporal chronon or bitemporal rectangle.

3.1.2 Attribute Variability

Attributes are commonly categorized based on how they interact with time. A *time-invariant* attribute [NA89] does not change over time.

The key value in a tuple of a relation instance is commonly used to identify the object, entity or relationship, in the modeled reality. If the key value changes, the tuple represents another object. Thus, the key of a relation schema is time invariant in such models. For example, attribute Name is a time-invariant key in relation schema R = (Name, Course) recording the courses taken by a student population. Time invariance is not restricted to key attributes. The attribute "place of birth" is an example. Note that time invariance generally is applied to valid time. The place of birth might have been in error; in that case, the old tuple would be (logically) deleted and a new tuple with the correct place of birth inserted.

Other models identify the objects that the tuples in a relation instance represent by means of surrogates which are system-generated, unique identifiers that can be referenced and compared for equality, but not displayed to the user [HOT76]. Surrogates are by definition time invariant.

The opposite of time invariant is *time varying.* Examples abound. In the schema R above, the courses taken by a student varies over time, and the attribute Course is time varying.

The *value* of an attribute may be drawn from a temporal domain. Such temporal domains are termed *user-defined time* [SA86]; other than being able to be read in, displayed, and perhaps compared, no special semantics is associated with such domains. Interestingly, most such attributes are time-invariant. The attribute "time of birth" is an example.

3.1.3 Implicit Versus Explicit Timestamps

In some data models, the association of times with facts is implicit; in other models, this association is represented by fully explicit timestamp attributes. We shall now see how this distinction is relevant to three aspects of a data model: update language, display of data, and query language.

The transaction times of facts are supplied by the system itself. Thus, update languages of transaction-time models treat the temporal aspect of facts implicitly. In contrast, the valid times of facts are usually supplied by the user. Thus, update languages of valid-time and bitemporal data models generally must treat time explicitly and are forced to represent a choice as to how the valid times of facts should be specified by the user. At best such data models can allow the the user to choose between several formats.

If, in a data model, it is possible to display directly temporal facts, i.e., facts with associated times, then, as for update, the data model necessarily must treat time explicitly. At best, the model may allow a variety of display formats for temporal facts. Unlike for update, the possibility exists that temporal facts cannot be displayed. This option is especially feasible for the relatively simple transaction time models, and thus the display of facts in these models need not reveal how time is associated with facts.

The query language aspect of the distinction between implicit and explicit timestamps is by far the most complex. If the temporal aspects of facts are represented by attributes, and it is possible in the query language to directly access these attributes then the temporal attributes are just like other attributes—they are explicit. On the other hand, if the timestamp attributes used for associating times with facts are not accessible directly through the query language, but are instead processed internally by queries, then the particular scheme for associating timestamps with facts is invisible to the user of the query language.

3.1.4 Temporal Homogeneity

When several temporal facts pertain to the same object (usually the object is a tuple), the concept of temporal homogeneity surfaces. A tuple is *temporally homogeneous* if each of its facts are defined over the same temporal element [Gad88]. A temporal relation is said to be temporally homogeneous if its tuples are temporally homogeneous [JCE$^+$94]. Further, a temporally homogeneous relation schema is restricted to have only temporally homogeneous relation instances. In addition to being specific to a type of object, homogeneity may be applied to both the valid and the transaction time dimension.

The motivation for homogeneity arises from the fact that the process of deriving a snapshot from of a homogeneous relation does not produce null values.

Certain data models assume temporal homogeneity. Models that employ tuple timestamping rather than attribute value timestamping are necessarily temporally homogeneous—only temporally homogeneous relations are possible.

3.1.5 Value Equivalence and Coalescing

Two tuples are termed *value equivalent* if, when disregarding special timestamp attributes, they are identical. A relation instance is *coalesced* if overlapping or consecutive, value-equivalent tuples are disallowed. Here "overlapping" and "consecutive" are with respect to the timestamp attribute value(s) of the tuples, which must specify a single chronon or a set of consecutive chronons.

When timestamps of tuples have temporal elements as values, the requirement of coalescing is identical to the requirement that there be no value-equivalent tuples present.

3.2 Overview

Over two dozen extensions to the relational model to incorporate time have been proposed over the last 15 years. With a focus on the types of relations they provide, we now review 23 of these temporal data models.

Table 1 lists most of the temporal data models that have been proposed to date. If the model is not given a name, we appropriate the name given the associated query language, where available. Many models are described in several papers; the one referenced is the initial journal paper in which the model was defined. Some models are defined only over valid time or transaction time; others are defined over both. The last column indicates a short identifier which denotes the model; the table is sorted on this column.

We omit a few intermediate data models, specifically Gadia's multihomogeneous model [Gad86a], which was a precursor to his heterogeneous model (Gadia-2), and Gadia's two-dimensional temporal relational database model [BG89b], which is a precursor to Gadia-3. We also do not include the data model used as the basis for defining temporal relational completeness [TC90] because it is a generic data model purposefully designed not to force decisions on most of the aspects to be discussed here.

We first examine the valid-time models that timestamp tuples, then discuss those that timestamp attribute values. We'll proceed chronologically (of course!) We then examine the transaction-time models, and conclude with the bitemporal models that support both valid and transaction time.

3.3 Valid-time Models

Approximately half the proposed temporal data models support only valid time.

Brooks The first academic treatment of time in databases was the dissertation of Frederick Brooks, Jr., which proposes a three-dimensional view of a valid-time database [Bro56]. Subsequent proposals, notably Ahn, Ariav, Clifford-1 and McKenzie, have emphasized this fruitful "cubic" analogy.

Data Model	*Citation*	*Time Dimension(s)*	*Identifier*
—	[SA86]	both	Ahn
Temporally Oriented Data Model	[Ari86]	valid	Ariav
Time Relational Model	[BZ82]	both	Ben-Zvi
—	[Bro56]	valid	Brooks
Historical Data Model	[CW83]	valid	Clifford-1
Historical Relational Data Model	[CC87]	valid	Clifford-2
Homogeneous Relational Model	[Gad88]	valid	Gadia-1
Heterogeneous Relational Model	[GY88]	valid	Gadia-2
TempSQL	[Gad92]	both	Gadia-3
DM/T	[JMR91]	transaction	Jensen
LEGOL 2.0	[JMS79]	valid	Jones
DATA	[Kim78]	transaction	Kimball
Temporal Relational Model	[Lor88]	valid	Lorentzos
—	[MS91]	both	McKenzie
Temporal Relational Model	[NA89]	valid	Navathe
HQL	[Sad87]	valid	Sadeghi
HSQL	[Sar90b]	valid	Sarda
Temporal Data Model	[SS87]	valid	Segev
TQuel	[Sno87]	both	Snodgrass
Postgres	[Sto87]	transaction	Stonebraker
HQuel	[Tan86]	valid	Tansel
Accounting Data Model	[Tho91]	both	Thompson
Time Oriented Data Base Model	[WFW75]	valid	Wiederhold

Table 1: Temporal Data Models

Wiederhold The data model associated with the Time Oriented Data Base (TOD) was developed specifically to support medical applications. In this pioneering model, relations were sets of entity-attribute-time-value quadruples [WFW75] or, for each attribute, sequences of events represented as pairs of visit number and value or intervals represented as sequences of pairs of visit numbers and sequences of values [Blu81]. Timestamping is indirect through the visit number; a separate array associates each visit with a particular date. This was probably done because many measurements are taken each visit. This structure was further elaborated as *time sequences* in Segev's model.

EXAMPLE: For the patient whose record is shown in Figure 1 [Blu81], John Smith's temperature was recorded during visit 1 (July 24, 1970, as recorded in the DATE_ARRAY) as 37.1°. He experienced two episodes of hepatitis, the first from visits 3 to 17, with a maximum of 850 International Units of SGOT during that interval of time. □

Jones LEGOL 2.0 [JMS79] is a language designed to be used in database applications such as legislative rules writing and high-level system specification in which the temporal ordering of events and the valid times for objects are important. It was the first time-oriented algebra defined; it introduced many of the features found in later algebras.

Objects in the LEGOL 2.0 data model are relations as in the relational data model, with one distinction. Tuples in LEGOL 2.0 are assigned two implicit time attributes, Start and Stop. The

$$\begin{tikzcd}
\text{P327} \arrow[d] \\
\text{Name} \arrow[r] \arrow[d] & \text{``John Smith''} \\
\text{Dates} \arrow[r] \arrow[d] & \text{DATE\_ARRAY} \\
\text{Temperatures} \arrow[r] \arrow[d] & (1, 37.1) \arrow[r] & (2, 37.3) \arrow[r] & (3, 37.0) \\
\text{Hepatitis} \arrow[r] & (\text{Intervals } (3, 17), (21, 26)) \arrow[r] & (\text{Interval-Values } (850, 1235))
\end{tikzcd}$$

Figure 1: A Time-Oriented Record for a Hypothetical Patient

values of these two attributes are the chronons corresponding to the (inclusive) end-points of the interval of existence (i.e., valid time) of the entity or relationship in the modeled reality represented by a tuple; these values are specified during data entry by the user.

EXAMPLE: Let R be a relation schema in LEGOL 2.0 that records the courses taken by a student population. The schema has the two explicit attributes, Name and Course. An instance of R is shown in Figure 2. We use 1 to represent the Fall semester 1980, 2 to represent the Spring semester 1981, and so on. Later examples will show the semantically equivalent representation of this instance in other data models. Because the data models all define relations differently and, in some cases, require implicit attributes, we show all relation examples in tabular form for both clarity and consistency of notation. This relation shows that Bill was a student in the English course for the Fall 1980 semester and for the Fall 1981 and Fall 1982 semesters. □

Name	Course	Start	Stop
Bill	English	1	1
Bill	English	3	4
George	English	1	2
George	Math	5	6

Figure 2: An Example Relation with Time

Clifford-1 In the *Historical Database Model*, an additional, chronon-valued attribute, STATE, is part of each relation schema. A boolean attribute, EXISTS, is also added to indicate whether the particular tuple exists for that state [Cli82, CW83].

Ariav In the Temporally Oriented Data Model, a valid-time relation is a sequence of snapshot relation states, indexed by valid time, termed the *data cube* [Ari86]. Associated with this data model is a calculus-based query language, TOSQL.

Navathe The Temporal Relational Model [NA87] and its associated algebra were defined primarily to support TSQL [NA89], a temporal extension to SQL defined in the same paper. This valid-time model allows both non-time-varying and time-varying attributes, but all of a relation's attributes must be of the same type. Objects are classified as: snapshot relations, whose attributes are all non-time-varying, and valid-time relations, whose non-key attributes are all time-varying. Each tuple has associated an interval of validity which is recorded in two mandatory time attributes, Time-start and Time-end. The structure of a valid-time relation in the Temporal Relational Model is the same as that of a valid-time relation in LEGOL 2.0 (Figure 2), with one additional restriction: Value-equivalent tuples, although allowed, are required to be coalesced.

Sadeghi Sadeghi's data model [Sad87] is similar in many ways to Navathe's. It was designed to support the calculus-based valid-time query language HQL [SSD87], which in turn is based on DEAL [Dee85]. In Sadeghi's data model, all objects are valid-time relations. Two implicit attributes, Start and Stop, record the end-points of each tuple's interval of validity. Hence, the structure of a valid-time relation in Sadeghi's model is also the same as that of the valid-time relation in LEGOL 2.0 (Figure 2). Sadeghi's data model requires coalescing.

Sarda Sarda's data model and associated algebra [Sar90a] were designed to support the calculus-based query language HSQL [Sar90b]. This model associates valid time with tuples. Objects can be either snapshot or valid-time relations. Unlike the data models mentioned previously, Sarda's model represents valid time in a valid-time relation as a single, non-atomic, implicit attribute named Period. Also unlike the previous models, a tuple in Sarda's model is not considered valid at its right-most boundary point, i.e., the interval is closed on the left and open on the right.

Example: The relation in Figure 3 is a valid-time relation instance in Sarda's model. The first two tuples signify that Bill was enrolled in English during the Fall semester 1980 and the Fall semesters 1981 and 1982, but not during the Spring semester 1981. □

Name	Course	Period
Bill	English	$1 \ldots 2$
Bill	English	$3 \ldots 5$
George	English	$1 \ldots 3$
George	Math	$5 \ldots 7$

Figure 3: The Example Relation in Sarda's Data Model

The remaining data models employ distinct non-first-normal form data models, with attribute value timestamping and perhaps with multiple values per attribute. The non-atomicity of attribute values is due to their time-varying nature; any timeslice will usually be in first normal form. Hence, the data models are an extension of the conventional (1NF) relational model; the representation, viewed as a normal relation, is certainly not in 1NF, but then the operators included in the models do not operate on conventional relations—they operate on valid-time relations, which are extensions of conventional relations.

Segev The principal structure of the Temporal Data Model is the *time sequence*, which is a so-called surrogate value identifying the object along with a sequence of time-value pairs [SS87]. There are a variety of time sequences, depending on the assumptions made about the values at points of time intermediate to the points explicitly represented. For a bank account balance, step-wise constant behavior would be assumed; for a time sequence recording the number of copies sold on a day for a particular book, discrete behavior would be assumed; and for measurement of a magnetic field taking at regular intervals, continuous behavior would be assumed. A *time sequence collection* (TSC) is then a set of time sequences.

Clifford-2 The *Historical Relational Data Model* [CC87], a refinement of the model associated with a valid-time algebra [CT85], is unique in that it associates timestamps with both individual tuples and with individual attribute values of the tuples. The data model allows two types of objects: a set of chronons, termed a *lifespan*, and a valid-time relation, where each attribute in the relation schema and each tuple in the relation is assigned a lifespan. A relation schema in the Historical Relational Data Model is an ordered four-tuple containing a set of attributes, a set of key attributes, a function that maps attributes to their lifespans, and a function that maps attributes to their value domains. A tuple is an ordered pair containing the tuple's value and its lifespan. Attributes are not atomic; rather, an attribute's value in a given tuple is a partial function from a domain of chronons onto the attribute's value domain. The domain of chronons is defined as the the intersection of the lifespan for the particular attribute and tuple. Relations have key attributes and no two tuples in a relation are allowed to match on the values of the key attributes at the same chronon.

Example: Figure 4 illustrates the valid-time relation instance in the Historical Relational Data Model, where {Name → {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}, Course → {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}} is the function assigning lifespans to attributes, and the attribute Name is the key.

Because tuple lifespans are sets and because both Bill and George were never enrolled in more than one course at the same time, we are able to record each of their enrollment histories in a single tuple. If one had been enrolled in two or more courses at the same time, however, his total enrollment history could not have been recorded in a single tuple as attribute values are functions from a lifespan onto a value domain. Note also that we have chosen the most straightforward representation for an attribute whose value is a function. Because attribute values in both Clifford's model and Gadia's models, which we describe later, are functions, they have many physical representations. □

Tansel Tansel's model [Tan86, CT85] was designed to support the calculus-based query language HQuel [TA86a] and, later, the Time-by-Example language [TAO89]. The model allows only one type of object: the valid-time relation. However, four types of attributes are supported: Attributes may be either non-time-varying or time-varying, and they may be either atomic-valued or set-valued. The attributes of a relation need not be the same type, and attribute values in a given tuple need not be homogeneous. The value of a time-varying, atomic-valued attribute is represented as a triplet containing an element from the attribute's value domain and the boundary points of its interval of existence while the value of a time-varying, set-valued attribute is simply a set of such triplets.

Example: Figure 5 shows the valid-time relation instance in Tansel's data model, where Name is a non-time-varying, atomic-valued attribute and Course is a time-varying, set-valued attribute.

Tuple Value		*Tuple Lifespan*
Name	Course	
$1 \rightarrow$ Bill $3 \rightarrow$ Bill $4 \rightarrow$ Bill	$1 \rightarrow$ English $3 \rightarrow$ English $4 \rightarrow$ English	{1, 3, 4}
$1 \rightarrow$ George $2 \rightarrow$ George $5 \rightarrow$ George $6 \rightarrow$ George	$1 \rightarrow$ English $2 \rightarrow$ English $5 \rightarrow$ Math $6 \rightarrow$ Math	{1, 2, 5, 6}

Figure 4: The Example Relation in the Clifford-2 Data Model

The enrollment history of a student can be recorded in a single tuple, even if the student was enrolled in two or more courses at some time. Note, however, that each interval of enrollment, even for the same course, must be recorded as a separate element of a time-varying, set-valued attribute. □

Name	Course
Bill	{ ([1, 2), English), ([3, 5), English) }
George	{ ([1, 3), English), ([5, 7), Math) }

Figure 5: The Example Relation in Tansel's Data Model

Gadia-1 Gadia's homogeneous model [Gad88] allows two types of objects: valid-time elements [GV85] and valid-time relations. Valid-time elements are closed under union, difference, and complementation, unlike intervals. The model requires that all attribute values in a given tuple be functions on the same valid-time element, i.e., homogeneity.

EXAMPLE: Figure 6 depicts the relation instance in Gadia's homogeneous model. Here the interval $[t_1, t_2)$ is the set of chronons $\{t_1, \cdots, t_2 - 1\}$. Again, we are able to record the enrollment histories of Bill and George in single tuples only because they were never enrolled in more than one course at the same time (otherwise multiple tuples are required). □

Bhargava's 2-dimensional model [BG90, BG91] is an extension of Gadia's homogeneous model; it supports both valid and transaction time. Many of the criteria concerning transaction time that are satisfied by the data model discussed below are also satisfied by Bhargava's data model.

Name	Course
$[1, 2) \cup [3, 5) \rightarrow$ Bill	$[1, 2) \cup [3, 5) \rightarrow$ English
$[1, 3) \cup [5, 7) \rightarrow$ George	$[1, 3) \rightarrow$ English $[5, 7) \rightarrow$ Math

Figure 6: The Example Relation in the Gadia-1 Data Model

Gadia-2 Gadia's multihomogeneous model [Gad86a] and Yeung's heterogeneous models [Yeu86, GY88] are all extensions of the homogeneous model. They lift the restriction that all attribute values in a tuple be functions on the same temporal element, in part to be able to perform Cartesian product without loss of temporal information caused by merging two timestamps into one. We consider here only the latest [GY88] of these extensions. In this data model (termed Gadia-2), temporal elements may be multi-dimensional to model different aspects of time (e.g., valid time and transaction time). Attribute values are still functions from temporal elements onto attribute value domains, but attribute values need not be functions on the same temporal element. As a result of the lack of temporal homogeneity, some timeslices may produce nulls. Relations are assumed to have key attributes, with the restriction that such attributes be single-valued over their interval of validity. Also, no two tuples may match on the ranges of the functions assigned to the key attributes. Hence, in the previous example, the attribute Name would qualify as a key attribute in the heterogeneous model.

Lorentzos The *Temporal Relational Model* [Lor88, LJ88] was the first to support nested specification of timestamps using values of different granularity and to support periodic events. As with the data models discussed above, this model associates timestamps with individual attribute values rather than with tuples. Although a timestamp is normally associated with each of the attribute values in a tuple, a timestamp may be associated with any non-empty subset of attribute values in a tuple. Furthermore, no implicit or mandatory timestamp attributes are assumed. Timestamps are simply explicit, numeric-valued attributes, to be viewed and updated directly by the user. They represent either the chronon during which one or more attribute values are valid or a *boundary point* of the interval of validity for one or more attribute values. A timestamp in the Temporal Relational Model, like one in Sarda's model, does not include its right-most boundary point. Several timestamp attributes of nested granularity may also be used together in a specification of a chronon.

Example: Let R be a valid-time relation schema in the Temporal Relational Model defined by R = (Name, Course, Semester-start, Semester-stop, Week-start, Week-stop) where all four timestamp attributes are associated with both Name and Course. Assume that the granularity for the timestamp attributes Week-start and Week-stop is a week relative to the first week of a semester. Figure 7 shows the an instance of this relation schema. In this example, we specify the weeks during a semester when a student was enrolled in a course. For example, Bill was enrolled in English during the Fall semester 1980 for only the first 8 weeks of the semester. Note that the meaning of the Week-start and Week-stop attributes is relative to the Semester-start and Semester-stop attributes. □

Name	Course	Semester-start	Semester-stop	Week-start	Week-stop
Bill	English	1	2	1	9
Bill	English	3	5	1	17
George	English	1	3	1	9
George	Math	5	7	9	17

Figure 7: The Example Relation in Lorentzos' Data Model

The data model thus differs from the normal relational model only in that certain columns are given a specific interpretation as representing the period of validity of other column(s) in the relation.

3.4 Transaction-time Models

Transaction-time data models have the valuable property that the objects are *append-only.*

Kimball In the data model termed DATA [Kim78], the association of facts with times is fully implicit. Being a transaction-time model, update operations avoid the explicit mention of time, and do not reveal how times and facts are associated. Next, transaction-time relations cannot be displayed—only snapshot extracted from the transaction-time relations can be displayed. Thus, display does not reveal the particular association of facts and time, either. Finally, the association of facts and times is implicit in the query language—the notion of an explicit timestamp attribute is absent. The consequence is that a user has no way of knowing whether, e.g., timestamps are assigned on the attribute-value level or on the tuple level. Similarly, there is no way to see whether transaction-time event, interval, or element stamping is used.

The DATA data model is implemented using a combination of event-stamped tuples and pointers to predecessor tuples.

Stonebraker The Postgres Data Model [RS87] supports transaction time. As for the previous model, the association of facts with time is implicit with respect to the update language, the query language, and the display of facts. Unlike the previous model, display is not restricted to snapshot states as a relation containing all tuples is a sequence of states may be displayed as well. Such a relation is still a conventional snapshot relation.

In the Postgres system, transaction-time relations are implemented using two timestamp attributes specifying the time when the particular tuple is current in the relation, i.e., when it will appear in a snapshot.

Jensen As in the previous two models, the association of facts with time is invisible in the data model DM/T [JMR91].

As a compensation for the inability to display and directly access timestamped facts, DM/T contains a special system-generated and maintained transaction-time relation, termed a backlog, for each user-defined transaction-time relation. This log-like backlog contains the full, timestamped change history of the associated user-defined relation. Backlog tuples, change requests, are stamped with a single time value and an attribute with values that indicate whether an insertion, deletion, or modification is requested. The timeslice of a backlog is a selection of the portion that existed at the time of the time argument. Thus, the timestamps are present as explicit attributes even after timeslice and may be accessed like any other attribute.

Example: Figure 8 illustrates a backlog, timesliced at transaction time 510, for a user-defined transaction-time relation. At transaction time 423, it was recorded that Bill took the Math course. This entry was then "modified," without changing any values at time, 427. □

Name	Course	Time	Op
Bill	English	423	Ins
Bill	English	427	Mod
George	English	438	Ins
Bill	English	452	Ins
George	Math	487	Ins
George	Math	495	Del

Figure 8: The Example Relation in Jensen's Data Model

3.5 Bitemporal Data Models

Bitemporal data models support both valid time and transaction time.

Ben-Zvi The *Time Relational Model* [BZ82] was the first bitemporal data model. Two types of objects are defined: snapshot relations, as defined in the snapshot model, and bitemporal relations. Bitemporal relations are sets of tuples, with each tuple having five implicit attribute values. The attributes Effective-time-start and Effective-time-stop are the end-points of the interval of validity of the real-world phenomenon being modeled; Registration-time-start is the transaction time of the transaction that stored the Effective-time-start value; Registration-time-stop is the transaction time that stored the Effective-time-stop value; and Deletion-time records the time when erroneously entered tuples are logically deleted. An erroneous attribute value may be corrected by deleting that tuple and inserting a corrected one.

Example: The relation instance in Figure 9 is a bitemporal relation in the Time Relational Model over a relation schema with explicit attributes Name and Course. Note that Georeg's enrollment in the Math course has been (logically) deleted. □

Ahn In differentiating valid and transaction time, a four-dimensional data model was used [SA85, SA86]. Relational instances were illustrated as a sequence, stamped with individual transaction times, of three-dimensional volumes, where one of the dimensions was valid time (tuples were stamped with intervals).

Snodgrass In the data model associated with TQuel, four implicit attributes were added to each relation: the transaction time of the transaction inserting the tuple, the transaction time of the transaction logically deleting the tuple, the time that the tuple started being valid in reality, and the time that the tuple stopped being valid in reality [Sno87, SGM93].

Name	Course	Effective time-start	Effective time-stop	Registration time-start	Registration time-stop	Deletion time
Bill	English	1	1	423	427	—
George	English	1	2	438	438	—
Bill	English	3	4	452	452	—
George	Math	5	6	487	487	495

Figure 9: The Example Relation in Ben-Zvi's Data Model

Example: Figure 10 shows, in the TQuel data model, the bitemporal relation given in Figure 9. □

Name	Course	Valid		Transaction	
		Begin	End	Start	Stop
Bill	English	1	∞	423	427
Bill	English	1	1	427	∞
George	English	1	2	438	∞
Bill	English	3	4	452	∞
George	Math	5	6	487	495

Figure 10: The Example Relation in Snodgrass' Data Model

McKenzie McKenzie's bitemporal model [McK88, MS91] timestamps attribute values but retains the requirement that attributes be single valued. This was done in an effort to achieve the benefits of attribute-value timestamping (e.g., the ability to perform a cartesian product) without the implementation complexities of set-valued attributes. The two types of objects in this model are the snapshot and valid-time relations (a transaction-time relation is a sequence of snapshot relations; a bitemporal relation is a sequence of valid-time relations, both indexed by transaction time). The value of an attribute in a valid-time relation is always an ordered pair whose components are a value from the attribute's domain and a set of chronons. There is no requirement that the timestamps of any of the attribute values in a relation be homogeneous, but relations are not allowed to have value-equivalent tuples.

Example: A valid-time relation instance in McKenzie's data model is shown in Figure 11. In this model, Bill's enrollment in English must be recorded in a single tuple, otherwise the value-equivalence requirement is violated. George's enrollment history, however, cannot be recorded in a single tuple; an attribute may be assigned only one value from its value domain. □

Transaction time was supported by indexing a sequence of valid-time states with transaction time [MS90]. This data model also allowed the schema, and even the class of the relation (i.e., snapshot, valid-time, transaction-time, or bitemporal) to vary.

Name	Course
$\langle$ Bill, {1, 3, 4} $\rangle$	$\langle$ English, {1, 3, 4} $\rangle$
$\langle$ George, {1, 2} $\rangle$	$\langle$ English, {1, 2} $\rangle$
$\langle$ George, {5, 6} $\rangle$	$\langle$ Math, {5, 6} $\rangle$

Figure 11: The Example Relation in McKenzie's Data Model

Gadia-3 In the data model associated with the calculus-based query language TempSQL [Gad92], attributes are timestamped with finite unions of rectangles in valid-time/transaction-time space [BG89b], i.e., effectively bitemporal elements.

EXAMPLE: Figure 12 shows the bitemporal relation given earlier, now as an instance of a relation in the TempSQL data model. □

Name		Course	
$[1,\infty] \times [423,427]$	Bill	$[1,\infty] \times [423,427]$	English
$[1,1] \times [423,\mathrm{NOW}]$	Bill	$[1,1] \times [423,\mathrm{NOW}]$	English
$[3,4] \times [452,\mathrm{NOW}]$	Bill	$[3,4] \times [452,\mathrm{NOW}]$	English
$[1,2] \times [438,\mathrm{NOW}]$	George	$[1,2] \times [438,\mathrm{NOW}]$	English
$[5,6] \times [487,495]$	George	$[5,6] \times [487,495]$	Math

Figure 12: The Example Relation in the Gadia-3 Data Model

Thompson In the Accounting Data Model, tuples have, in addition to the natural key, the static attributes, and the time-varying attributes, four timestamp attributes: accounting start time, accounting finish time, engineering start time, engineering finish time, as well as a boolean timewarp attribute [Tho91]. The accounting time roughly corresponds to valid time, and the engineering time corresponds to transaction time (a more detailed comparison may be found elsewhere [JS93]). The time warp attribute enables attribute values to change historically.

3.6 Summary

The following brief summary oversimplifies the data models in an effort to differentiate them.

- Brooks was the first to consider time in the database (long before the relational model was proposed!).
- Wiederhold was the first temporal model to be implemented.
- Jones was the first to define a time-oriented algebra.
- Clifford-1 attempted to model the semantics of natural language.

- Ariav exploited the three-dimensional analogue, where the third dimension is valid time.
- Navathe defined his data model primarily to support his extension to SQL called TSQL.
- Sadeghi's data model was defined primarily to support his extension to DEAL called HQL.
- Sarda, Lorentzos and Tansel all incorporated operators to switch between an interval representation and a single chronon representation. Lorentzos' data model, closest to the conventional relational data model, supports nested granularity timestamps and periodic time.
- Segev focussed on scientific data, collected generally at regular intervals by multiple sensors.
- Clifford-2, Gadia-1, Gadia-2, Gadia-3, and Tansel all employ non-1NF data models. Clifford-1 emphasizes associating timestamps with both the attribue value and with the tuple; Clifford-2 associates timestamps with both attributes and with tuples; Gadia-1 emphasizes the homogeneity property; Gadia-2 emphasizes the multi-homogeneous property; and Tansel includes four types of attribute values.
- Kimball was the first implemented transaction-time model.
- Stonebraker has the most impressive implementation to date of a temporal data model.
- Jensen used backlog relations to encode the changes made to transaction-time relations.
- Ben-Zvi was the first to incorporate both transaction time and valid time.
- Ahn demonstrated that transaction time and valid time are entirely orthogonal.
- Snodgrass used a particularly simple bitemporal model to support TQuel.
- McKenzie timestamped attribute values but retains the requirement that attributes have only a single value within a tuple.
- Gadia-3 effectively used bitemporal elements.
- Thompson focused on the use of temporal databases in accounting.

3.7 Comparison

The temporal data models just summarized may be compared by asking four basic questions: how is valid time represented, how is transaction time represented, how are attribute values represented, is the model *homogeneous*, and is the model *coalesced.*

3.7.1 Valid Time

Two fairly orthogonal aspects are involved in representing valid time. First, is valid time represented with single chronon identifiers (i.e., event timestamps), with intervals (i.e., as interval timestamps), or as valid-time elements (i.e., as a set of chronon identifiers, or equivalently as a finite set of intervals)? Second, is valid time associated with entire tuples or with individual attribute values? A third alternative, associating valid time with sets of tuples, i.e., relations, has not been incorporated into any of the proposed data models, primarily because it lends itself to high data redundancy. The data models are evaluated on these two aspects in Table 2. Interestingly, only one quadrant, timestamping tuples with an valid-time element, has not been considered.

	Event	Interval	Valid-time Element
timestamped attribute values		Gadia-2 Lorentzos McKenzie Thompson Tansel	Brooks Clifford-2 Gadia-1 Gadia-3
timestamped tuples	Ariav Clifford-2 Segev	Ahn Ben-Zvi Jones Navathe Sadeghi Sarda Snodgrass Wiederhold	

Table 2: Representation of Valid Time

3.7.2 Transaction Time

The same general issues are involved in transaction time, but there are about twice as many alternatives. Transaction time may be associated with

- a single chronon. When stamping a tuple identifying a change to a relation state, the insertion of the tuple signifies the termination (logical deletion) of the most recent tuple (if any) with an identical key value. An additional attribute is required to indicate whether the newly inserted tuple only terminates the previous tuple or also becomes part of the new state (e.g., the attribute Op in Jensen). When an entire evolving state is stamped, no such attribute is necessary. One state is current from its chronon and until it is superceeded by a state with a higher chronon. Note that this alternative results in very high redundancy when compared with the first alternative.

- an interval. A newly inserted tuple would be associated with the interval starting at now and ending at the special value UC, *until-changed*.

- three chronons. Ben-Zvi's model records (1) the transaction time when the valid start time was recorded, (2) the transaction time when the valid stop time was recorded, and (3) the transaction time when the tuple was logically deleted.

- a transaction-time element, which is a set of not-necessarily-contiguous chronons.

Another issue concerns whether transaction time is associated with individual attribute values, with tuples, or with sets of tuples.

The choices made in the various data models are characterized in Table 3. Gadia-3 is the only data model to timestamp attribute values; it is difficult to efficiently implement this alternative directly. Gadia-3 also is the only data model that uses transaction-time elements. Ben-Zvi is the only one to use three transaction-time chronons. All of the rows and columns are represented by at least one data model.

	Single chronon	Interval (pair of chronons)	Three Chronons	Transaction-time element (set of chronons)
timestamped attribute values				Gadia-3
timestamped tuples	Jensen Kimball	Snodgrass Stonebraker	Ben-Zvi	
timestamped sets of tuples	Ahn Thompson	McKenzie		

Table 3: Representation of Transaction Time

3.7.3 Homogeneity and Coalescing

Table 4 compares the models on the last two aspects. The name of the data model is given in the first column. Whether the model is homogeneous in valid time is indicated in the next column (c.f., Section 3.1.4). All the models are homogeneous in transaction time. Tuple-timestamped data models, to be identified shortly, are necessarily temporally homogeneous. All data models that use single chronons as timestamps turn out to be temporally homogeneous as well. For data models that only support transaction time, this aspect is not relevant.

The next column specifies whether the data model requires that tuples be coalesced in valid time (c.f., Section 3.1.5). No model is coalesced on transaction time. Event-stamped data models are by necessity not valid-time coalesced.

3.7.4 Attribute Value Structure

The final major decision to be made in designing a temporal data model is how to represent attribute values. Six basic alternatives are present in the data models. In some models, the timestamp appears as an explicit attribute; we do not consider such attributes in this analysis.

- *Atomic valued*—values do not have any internal structure.
- *Set valued*—values are sets of atomic values.
- *Functional, atomic valued*—values are functions from the (generally valid) time domain to the attribute domain.
- *Ordered pairs*—values are an ordered pair of a value and a (valid-time element) timestamp.
- *Triplet valued*—values are a triple of attribute values, valid-from time, and valid-to time. This is similar to the ordered pairs representation, except that only one interval may be represented.
- *Set-triplet valued*—values are a set of triplets. This is more general than ordered pairs, in that more than one value can be represented, and more general than functional valued, since more than one attribute value can exist at a single valid time [Tan86].

The last column of Table 4 specifies the attribute value structure associated with each temporal data model.

In the conventional relational model, if attributes are atomic-valued, they are considered to be in *first normal form* [Cod72]. Hence, only the data models placed in the first category may be considered to be strictly in first normal form. However, in several of the other models, the non-atomicity of attribute values comes about because time is added.

Data Model	*Valid-time Homogeneous*	*Valid-time Coalesced*	*Attribute Values*
Ahn	yes	yes	atomic
Ariav	yes	no	atomic
Ben-Zvi	yes	no	atomic
Brooks	no	?	atomic
Clifford-1	yes	no	atomic
Clifford-2	no	no	functional
Gadia-1	yes	no	functional
Gadia-2	no	yes	functional
Gadia-3	yes	no	functional
Jensen	N/A	N/A	atomic
Jones	yes	no	atomic
Kimball	N/A	N/A	atomic
Lorentzos	no	no	atomic
McKenzie	no	yes	ordered pairs
Navathe	yes	yes	atomic
Sadeghi	yes	yes	atomic
Sarda	yes	no	atomic
Segev	yes	no	atomic
Snodgrass	yes	yes	atomic
Stonebraker	N/A	N/A	atomic
Tansel	no	no	atomic, set-valued, triplet, set-triplet
Thompson	yes	no	atomic
Wiederhold	yes	no	atomic, ordered pairs

Table 4: Comparison of Temporal Data Models

4 Context

The previously proposed data models arose from several considerations. They were all extensions of the conventional relational model that attempted to capture the time-varying semantics of either the reality being modeled, the state of the database, or both. They attempted to retain the simplicity of the relational model; the tuple timestamping models were perhaps most successful in this regard. They attempted to present all the information concerning an object in one tuple; the attribute value timestamped models were perhaps best at that. And they attempted to ensure ease of implementation and query evaluation efficiency; the backlog representation may be advantageous here.

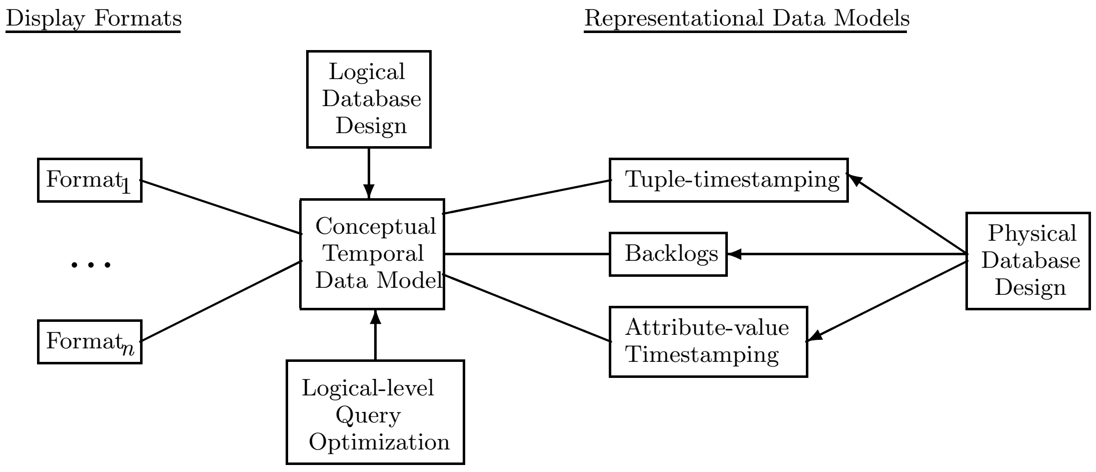
Display Formats
Representational Data Models
Logical Database Design
Format$_1$
...
Format$_n$
Conceptual Temporal Data Model
Tuple-timestamping
Backlogs
Attribute-value Timestamping
Physical Database Design
Logical-level Query Optimization

Figure 13: Interaction of Conceptual and Representational Data Models

It is clear from the number of proposed representations that meeting all of these goals simultaneously is a difficult, if not impossible task. It is our contention that focusing on data *presentation* (how temporal data is displayed to the user), on data *storage*, with its requisite demands of regular structure, and on efficient *query evaluation* has complicated the central task of capturing the time-varying semantics of data. The result has been, as we have seen, a plethora of incompatible data models, with many query languages, and a corresponding surfeit of database design and implementation strategies that may be employed across these models.

We therefore advocate a separation concerns. The time-varying semantics is obscured in the representation schemes by other considerations of presentation and implementation. We feel that the *conceptual* data model to be proposed shortly is the most appropriate basis for expressing this semantics. This data model is generally not the most appropriate way to present the stored data to users, nor is it the best way to physically store the data. However, there are mappings to several *representational* data models that, in many situations, may be more amenable to presentation and storage, those representations can be employed for those purposes, while retaining the semantics of the conceptual data model. Figure 13 shows the placement of the proposed data model with respect to the tasks of logical and physical database design, storage representation, query optimization, and display. As the figure shows, logical database design produces the conceptual relation schemas, which are then refined into relation schemas in some representational data model(s). Query optimization may be performed on the logical algebra, parameterized by the cost models of the representation(s) chosen for the stored data. Finally, display presentation should be decoupled from the storage representation.

Note that this arrangement hinges on the semantic equivalence of the various data models. It must be possible to map between the conceptual model and the various representational models, as will be discussed in Section 6.

5 A New Proposal

We now present a new model, termed the *bitemporal conceptual data model*, or BCDM. This data model supports both valid and transaction time. It is designed to be a conceptual data model, as opposed to a representational data model, in the sense just described.

We begin by specifying the structural aspects of the time domain assumed by the data model. In Section 5.2, we describe the objects (temporal relations) of the model and consider how these objects may be updated.

5.1 The Time Domain

For both valid and transaction time domains, we assume the linear, discrete, bounded structural model of time. We utilize chronons, as discussed in detail in a separate proposal on timestamp representation [DS92b]. We assume that chronons have length (some multiple or fraction of a "second"). We assume that valid and transaction time are absolute. Relative times may be stored in relations as values of attributes (termed *spans* [SS92]); such user-defined times are not discussed further here. As we can number the chronons, the domains are isomorphic to the domain of natural numbers.

5.2 Objects in the Model

Tuples in a bitemporal conceptual relation instance are associated with time values from both valid time and transaction time. For both domains, we assume that the database system has limited precision; the smallest time unit is termed a *chronon* [JCE+94]. The time domains have total orders and both are isomorphic to subsets of the domain of natural numbers. The domain of valid times may be given as $D_{VT} = \{t_1, t_2, \ldots, t_k\}$ and the domain of transaction times may be given as $D_{TT} = \{t'_1, t'_2, \ldots, t'_j\} \cup \{UC\}$ where UC is a distinguished value which is used during update as will be explained later in this section. We expect that the valid time domain is chosen so that some times are before the current time and some times are after the current time.

We also define a set of attribute names $\mathcal{D}_A = \{A_1, A_2, \ldots, A_{n_A}\}$ and a set of attribute domains $\mathcal{D}_D = \{D_1, D_2, \ldots, D_{n_D}\}$. In general, the schema of a bitemporal conceptual relation, $\mathcal{R}$, consists of an arbitrary number of explicit attributes from $\mathcal{D}_A$, A_1, A_2, $\ldots$, A_n, with domains in $\mathcal{D}_D$, encoding some fact (possibly composite) and an implicit timestamp attribute, T, with domain $D_{TT} \times D_{VT}$. Thus, a tuple, $x = (a_1, a_2, \ldots, a_n |\ t_b)$, in a bitemporal conceptual relation instance, $r(\mathcal{R})$, consists of a number of attribute values associated with a timestamp value.

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, the explanation is asymmetric. This assymmetry is also present in the the update operations to be defined shortly, and it reflects the different semantics of transaction and valid time.

Thus, associated with a tuple is a bitemporal element, denoted t_b, consisting of bitemporal chronons ("tiny rectangles") in the two-dimensional space spanned by valid time and transaction time. Because no two tuples with mutually identical explicit attribute values (termed *value-equivalent*) are allowed in a bitemporal relation instance, the full time 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 $(t,\ v)$ represents the bitemporal chronon with transaction time t and valid time v.

EXAMPLE: Consider a relation recording employee/department information, such as "Jake works for the shipping department." We assume that the granularity of chronons is one day for both valid time and transaction time, and the period of interest is some given month in a given year, e.g., June 1992. 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 June 15th.

Figure 14 shows how the bitemporal element in an employee's department tuple changes. Employee Jake was hired by the company as temporary help in the shipping department for the interval from time 10 to time 15, and this fact became current in the database at time 5. This is shown in Figure 14(a). The arrows pointing to the right signify that the tuple has not been logically deleted; it continues through to the transaction time UC(*until_changed*).

Figure 14(b) shows a correction. The personnel department discovers that Jake had really been hired from time 5 to time 20, and the database is corrected beginning at time 10. Later, the personnel department is informed that the correction was itself incorrect; Jake really was hired for the original time interval, time 10 to time 15, and the correction took effect in the database at time 15. This is shown in Figure 14(c). Lastly, Figure 14(d) shows the result of three updates to the relation, all of which become current starting at time 20. These three updates could have been entered in a single transaction, or as separate transactions occurring during the same chronon. While the period of validity was correct, it was discovered that Jake was not in the shipping department, but in the loading department. Consequently, the fact (Jake, Ship) is removed from the current state and the fact (Jake, Load) is inserted. A new employee, Kate, is hired for the shipping department for the interval from time 25 to time 30.

We note that the number of bitemporal chronons in a given bitemporal element is the area enclosed by the bitemporal element. The bitemporal element for (Jake, Ship) contains 140 bitemporal chronons.

The example illustrates how transaction time and valid time are handled. As time passes, i.e., as the computer's internal clock advances, the bitemporal elements associated with current facts are updated. For example, consider when the fact (Jake, Ship) was first inserted into the database. Due to the semantics of insertion as described in the next section, facts are inserted to the relation during the chronon prior to when they first become current. Thus (Jake, Ship) is physically inserted into the relation at time 4, with six valid time chronons (10 to 15) each with the associated transaction time chronon UC.

At this time, the fact is not yet current in the database since no bitemporal chronons with a transaction time other than UC are associated with the tuple. At time 5, the fact logically becomes current in the database, and the six new bitemporal chronons, $(5, 10), \ldots, (5, 15)$, are appended. This continues until time 9, when a correction to the fact's valid time is made. Thus, starting at time 10, 16 bitemporal chronons are added at every clock tick.

The actual bitemporal relation corresponding to the graphical representation in Figure 14(d) is shown in Figure 15 below. This relation contains three facts. The timestamp attribute T shows each transaction-time chronon associated with each valid-time chronon as a set of ordered pairs.

□

Valid-time relations and transaction-time relations are special cases of bitemporal relations that

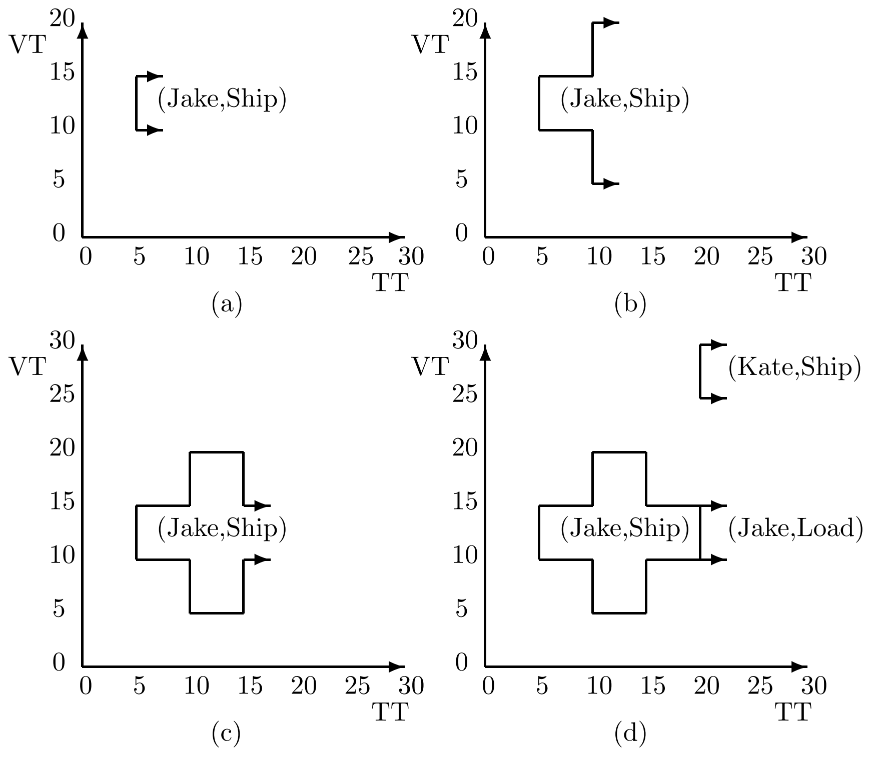

Figure 14: Bitemporal Elements

Emp	Dept	T
Jake	Ship	$\{(5,10),\ldots,(5,15),\ldots,(9,10),\ldots,(9,15),$ $(10,5),\ldots,(10,20),\ldots,(14,5),\ldots,(14,20),$ $(15,10),\ldots,(15,15)\ldots,(19,10),\ldots,(19,15)\}$
Jake	Load	$\{(UC,10),\ldots,(UC,15)\}$
Kate	Ship	$\{(UC,25),\ldots,(UC,30)\}$

Figure 15: Bitemporal Relation Instance

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.

5.3 Update

In this section, we describe the semantics of the three forms of update, insertion, deletion, and modification. This description is pedagogical, meant only to illustrate the semantics of the operations, and not intended for implementation. Possible techniques for efficiently supporting these semantics are discussed in Section 6.

An insertion is issued when we want to record in bitemporal relation instance r that a currently unrecorded fact $(a_1,\ldots,a_n)$ is true for some period(s) of time. These periods of time are represented by a valid-time element. When the fact is stored, its valid-time element stamp is transformed into a bitemporal-element stamp to capture that, until its explicit attribute values are changed, the fact is current in the relation. This is indicated with the special transaction time value UC.

The arguments to the `insert` routine are the relation into which a fact is to be inserted, the explicit values of the fact, and the set of valid-time chronons, t_v, during which the fact was true in reality. The `insert` routine returns the new, updated version of the relation. There are three cases to consider. First, if $(a_1, \ldots, a_n)$ was never recorded in the relation, a completely new tuple is appended. Second, if $(a_1, \ldots, a_n)$ was part of some previously current state, the tuple recording this is updated with the new valid time information. Third, if $(a_1, \ldots, a_n)$ is already current in the relation, a modification is required, and the insertion is rejected (in this case, a `modify` operation should have been used). In the following, we denote valid-time chronons with c_v and transaction-time chronons with c_t.

$$\texttt{insert}(r, (a_1, \ldots, a_n), t_v) =$$
$$\begin{cases} r \cup \{(a_1, \ldots, a_n | \{UC\} \times t_v)\} & \text{if } \neg\exists\, t_b\; ((a_1, \ldots, a_n |\, t_b) \in r) \\ r - \{(a_1, \ldots, a_n |\, t_b)\} & \\ \quad \cup \{(a_1, \ldots, a_n |\, t_b \cup \{\{UC\} \times t_v\})\} & \text{if } \exists\, t_b\; ((a_1, \ldots, a_n |\, t_b) \in r \,\wedge\, \neg\exists\, (UC, c_v) \in t_b) \\ r & \text{otherwise} \end{cases}$$

The `insert` routine adds bitemporal chronons with a transaction time of UC.

As transaction time passes, new chronons must be added. Logically, this is performed by a special routine `ts_update` which is applied to all bitemporal relations at each clock tick. This function simply updates the timestamps to include the new transaction-time value. The timestamp of each tuple is examined in turn. When a bitemporal chronon of the type (UC, c_v) is encountered in the timestamp, a new bitemporal chronon (c_t, c_v), where time c_t is the new transaction-time value, is made part of the timestamp.

```
ts_update($(r, c_t)$) :
    for each $x \in r$
        for each $(UC, c_v) \in x[T]$
            $x[\mathrm{T}] \leftarrow x[\mathrm{T}] \cup \{(c_t, c_v)\}$;
```

We note again that `ts_update` is part of the logical semantics of the conceptual model, and that direct implemention would be prohibitively expensive. In Section 6, we discuss efficient ways to support these semantics.

Deletion concerns the logical removal of a tuple from the current valid-time state of a bitemporal relation. To logically remove a qualifying tuple from the current state, we delete all chronons (UC, c_v), where c_v is some valid-time chronon, from the timestamp of the tuple. As a result, the timestamp is not expanded by subsequent invocations of `ts_update`, and the tuple will not appear in future valid-time states. If there is no qualifying tuple in the relation, or if a qualifying tuple exists but has no chronons with a transaction time of UC, then the deletion has no effect.

$$\texttt{delete}(r, (a_1, \ldots, a_n)) =$$
$$\begin{cases} r - \{(a_1, \ldots, a_n |\, t_b)\} \cup \{(a_1, \ldots, a_n |\, t_b - \texttt{uc\_ts}(t_b))\} & \text{if } \exists\, t_b\; ((a_1, \ldots, a_n |\, t_b) \in r) \\ r & \text{otherwise} \end{cases}$$

where $\texttt{uc\_ts}(t_b) = \{(UC, c_v) \mid (UC, c_v) \in t_b\}$.

Finally, a modification of an existing tuple is defined by a deletion followed by an insertion as follows.

$$\texttt{modify}(r, (a_1, \ldots, a_n), t_v) = \texttt{insert}(\texttt{delete}(r, (a_1, \ldots, a_n)), (a_1, \ldots, a_n), t_v)$$

EXAMPLE: The conceptual relation in Figure 15 is created by the following sequence of commands, invoked at the indicated transaction time.

Command	*Transaction Time*
`insert(dept,("Jake","Ship"),[10,15])`	5
`modify(dept,("Jake","Ship"),[5,20])`	10
`modify(dept,("Jake","Ship"),[10,15])`	15
`delete(dept,("Jake","Ship"))`	20
`insert(dept,("Jake","Load"),[10,15])`	20
`insert(dept,("Kate","Ship"),[25,30])`	20

□

We have given a definition of a bitemporal conceptual relation. As part of the definition, we used the special value *UC* in conjunction with the routine `ts_update` to allow timestamps of tuples to grow as time passes. It should be emphasized that users will not see the value *UC*. Query results are static, and there is no need to display this value.

5.4 Logical Design

A confusing array of normal forms for temporal relations, including *First Temporal Normal Form* [SS88a], *Time Normal Form* [NA89], and *P* and *Q Normal Forms* [LK89], have been proposed. None of these definitions is truly an extension of conventional normal forms, for a variety of reasons that we detail elsewhere [JSS92a]. Also, each definition is restricted to a specific data model, and inherits the peculiarities inherent in that model. It is not satisfactory to have to define all the normal forms anew for each of the two dozen existing temporal data models.

Elsewhere we present a consistent framework of temporal equivalents of all the important conventional database design concepts: functional and multivalued dependencies, primary keys, and third, Boyce-Codd, and fourth normal forms [JSS92a]. This framework is enabled by making a clear distinction between the logical concept of a temporal relation and its physical representation. As a result, the role played by temporal normal forms during temporal database design in the BCDM closely parallels that of normal forms during conventional database design.

5.5 Evaluation

We briefly evaluate the bitemporal conceptual data model using the same criteria by which existing temporal data models were compared in Section 3.7.

The BCDM timestamps tuples, as does Ben-Zvi, Clifford-1, Jones, Navathe, Sadeghi, Sarda, Segev and Snodgrass. The timestamps are temporal elements, as in Clifford-2, Gadia-1 and Gadia-3. In Table 2, the BCDM occupies the unfilled entry corresponding to timestamping tuples with valid-time elements. In Table 3, the BCDM occupies the unfilled entry corresponding to timestamping tuples with transaction-time elements. Hence, the BCDM is unique in that it timestamps tuples with bitemporal elements. The BCDM is inherently valid-time homogeneous—about half of the temporal data models are homogeneous. The BCDM is also inherently valid-time coalesced;

Ahn, Gadia-2, McKenzie, Navathe, Sadeghi and Snodgrass are coalesced. Attributes are atomic in the BCDM, as in most of the temporal data models proposed to date.

In the next section, we shall see how the temporal relations defined thus far may be mapped to other formats, some of which may be better for display or storage of temporal data.

6 Representation Schemes

A BCDM relation is structurally simple—it is a set of facts, each timestamped with a bitemporal element which is a set of bitemporal chronons. In this section, we more closely examine five representations of bitemporal relations that have been previously proposed. These representations fall into the class of temporally ungrouped models [CCT94], and constitute all such models proposed to date, to our knowledge. For each, we briefly specify the objects defined in the representation, provide the mapping to and from conceptual bitemporal relations to demonstrate that the same information is being stored, and show how updates of bitemporal conceptual relations may be mapped into updates on relations in the representation. We progress from a simple model to ones associated with more complex mappings.

In the following, we will use R and S to denote relation schemas. Relation instances are denoted by r, s, and t, and $r(R)$ means that r is an instance of R. For brevity, we use A to denote the set of all attributes A_i, $1 \leq i \leq n$. For tuples we use x, y, and z, possibly indexed, and the notation $x[A_i]$ denotes the A_i^{th} attribute of x. Similarly, $x[\mathrm{T}]$ denotes the timestamp associated with x. Often, when discussing representational models, we will use $x[\mathrm{V}]$ and $x[\mathrm{T}]$ to denote the valid-time and transaction-time intervals, respectively, associated with a representational tuple x. The differing use associated with conceptual tuples should be clear from context.

6.1 Snodgrass' Tuple Timestamped Representation Scheme

In the conceptual model, the timestamp associated with a tuple is an arbitrary set of bitemporal chronons. As such, a relation schema in the conceptual model is non-1NF, which represents difficulties if directly implemented. We describe here how to represent conceptual relations by 1NF snapshot relations [Sno87], allowing the use of existing, well-understood implementation techniques.

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 snapshot relation schema R as follows.

$$R \;=\; (A_1, \ldots, A_n, \mathrm{T}_s, \mathrm{T}_e, \mathrm{V}_s, \mathrm{V}_e)$$

The additional attributes T_s, T_e, V_s, V_e are atomic-valued timestamp attributes containing a starting and ending transaction-time chronon and a starting and ending valid-time chronon, respectively. These four values represent the bitemporal chronons in a rectangular region, the idea being to cover the region represented by the bitemporal element of a conceptual tuple into a number of rectangles and then to represent the conceptual tuple by a set of representational tuples, one for each rectangle.

There are many possible ways of covering a bitemporal element. To ensure the representation remains faithful to the semantics of the conceptual relation, we require that any covering function on a bitemporal element $x[\mathrm{T}]$ of a bitemporal tuple x satisfy two properties.

1. Any bitemporal chronon in $x[\mathrm{T}]$ must be contained in at least one rectangle.

2. Each bitemporal chronon in a rectangle must be contained in $x[\mathrm{T}]$.

The first condition ensures that all chronons in the bitemporal element of x are accounted for; the second ensures that no spurious chronons are introduced. Hence, the covering represents the same information as is contained in the original tuple.

Apart from these requirements, the covering function is purposefully left unspecified—an implementation is free to choose a covering with properties it finds desirable. For example, a set of covering rectangles need not be disjoint. Overlapping rectangles may reduce the number of tuples needed in the representation, at the possible expense of additional processing during update.

Example: While the results presented in this paper are independent of particular covering functions, it is still useful to consider some examples to illustrate the range of possibilities.

Figure 16 illustrates three ways of covering the bitemporal element associated with the fact (Jake, Ship) contained in Figure 15, shown graphically in Figure 14(d). We may distinguish between those covering functions that partition the argument set into disjoint rectangles and those that allow overlap between the result rectangles. Figure 16(a) and Figure 16(b) are examples of partitioned coverings while the covering in Figure 16(c) has overlapping rectangles.

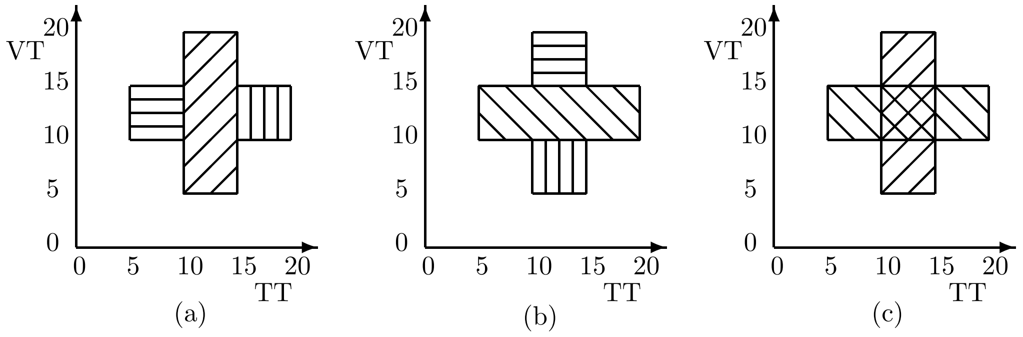

Figure 16: Example Coverings of a Bitemporal Element

Figure 16(a) illustrates a type of covering where regions are partitioned by transaction time. Maximal transaction-time intervals are located so that each transaction time in an interval has the same interval of valid times associated. In the figure, the transaction-time interval (5,9) is maximal, and the associated valid-time interval is (10,15). Thus, the rectangle with corners (5,10) and (9,15) is part of the result. Similarly, the two rectangles with corners ((10,5), (14,20)), and ((15,10), (19,15)) are in the result. Due to the semantics of transaction time [JMRS93], this is perhaps the most natural choice of covering [Sno87]. Indeed, all the examples of representations of the employee bitemporal relation use covering functions that partition by transaction time.

Figure 16(b) illustrates the symmetric partitioning by valid time. Here, three rectangles are created with corners at ((5,10), (19,15)), ((10,5), (14,10)), and ((10,15), (14,20)).

Figure 16(c) exemplifies a type of covering that allows overlaps. The two rectangles in this covering have corners at ((5,10), (19,15)) and ((10,5), (14,20)). The overlap of these rectangles means that two tuples will express the fact that Jake was in the shipping department from time 10 to time 15, recorded as current information from time 10 to time 14.

The last example demonstrates that a covering function that allows overlap may result in a smaller number of covering rectangles, and therefore may yield a more compressed representation than a covering function that partitions. However, this repetition of information makes some updates more time consuming, as more tuples may be affected by a single update. □

We will make use of covering functions throughout this section when representing bitemporal elements of conceptual tuples with rectangles.

EXAMPLE: The 1NF relation corresponding to the conceptual relation in Figure 15 is shown below.

Emp	Dept	T_s	T_e	V_s	V_e
Jake	Ship	5	9	10	15
Jake	Ship	10	14	5	20
Jake	Ship	15	19	10	15
Jake	Load	20	*UC*	10	15
Kate	Ship	20	*UC*	25	30

Here we use a non-overlapping covering function that partitions the bitemporal element by transaction time. □

The following functions convert between a bitemporal conceptual relation instance and a corresponding instance in the representation scheme. The second argument, *cover*, of the routine `conceptual_to_snap` is a covering function. It returns a set of rectangles, each denoted by a set of bitemporal chronons.

```
conceptual_to_snap($r'$, $cover$):
    $s \leftarrow \emptyset$;
    for each $x \in r'$
        $z[A] \leftarrow x[A]$;
        for each $t \in cover(x[\mathrm{T}])$
            $z[\mathrm{T}_s] \leftarrow min\_1(t)$; $z[\mathrm{T}_e] \leftarrow max\_1(t)$;
            $z[\mathrm{V}_s] \leftarrow min\_2(t)$; $z[\mathrm{V}_e] \leftarrow max\_2(t)$;
            $s \leftarrow s \cup \{z\}$;
    return $s$;
```

```
snap_to_conceptual($r$):
    $s \leftarrow \emptyset$;
    for each $z \in r$
        $r \leftarrow r - \{z\}$;
        $x[A] \leftarrow z[A]$;
        $x[\mathrm{T}] \leftarrow bi\_chr(z[\mathrm{T}], z[\mathrm{V}])$;
        for each $y \in r$
            if $z[A] = y[A]$
                $r \leftarrow r - \{y\}$;
                $x[\mathrm{T}] \leftarrow x[\mathrm{T}] \cup bi\_chr(y[\mathrm{T}], y[\mathrm{V}])$;
        $s \leftarrow s \cup \{x\}$;
    return $s$;
```

The functions *min_1* and *min_2* select a minimum first (transaction time) and second (valid time) component, respectively, in a set of bitemporal chronons. The function *max_1* returns the value *UC* if encountered as a first component; otherwise, it returns a maximum first component. The function *max_2* selects a maximum second component. The function *bi_chr* computes the bitemporal chronons covered by the argument rectangular region.

The `conceptual_to_snap` routine generates possibly many representational tuples from each conceptual tuple, each generated tuple corresponding to a rectangle in valid/transaction-time space. The `snap_to_conceptual` routine merges the rectangles associated with a single fact into a single bitemporal element.

Note that the functions are the inverse of each other, i.e., for any conceptual relation instance r',

$$\texttt{snap\_to\_conceptual}(\texttt{conceptual\_to\_snap}(r', cover)) = r'.$$

We sketch an argument around which a formal proof can be constructed. Consider a tuple x in the conceptual relation r'. The function `conceptual_to_snap` produces a set of value-equivalent representational tuples $\{z_1, z_2, \ldots, z_k\}$, $k \geq 1$, from this x, where the bitemporal rectangle associated with z_i is produced by $cover(x[\mathrm{T}])$. We claim that the reverse transformation performed by `snap_to_conceptual` coalesces the set of tuples $\{z_1, z_2, \ldots, z_k\}$ back into the conceptual tuple x. To see this, note that any value-equivalent tuples in `conceptual_to_snap`$(r', cover)$ must have been produced from x, otherwise value-equivalent tuples must have been present in r'. Let y be the conceptual tuple produced by coalescing $\{z_1, z_2, \ldots, z_k\}$. Then $y[\mathrm{T}]$ contains exactly the chronons contained in the union of the rectangles produced by $cover(x[\mathrm{T}])$. By the definition of covering functions, these are exactly the chronons in $x[\mathrm{T}]$. Hence $y = x$. It is easy to see that no spurious tuples can be produced by the transformations. Hence, the same conceptual relation is produced.

For the update routines, the most convenient covering functions partition on either valid or transaction time and do not permit overlaps. The current transaction time is c_t.

```
insert($r, (a_1, \ldots, a_n), t_v, cover_v$):
    $cvr \leftarrow cover_v(t_v)$;
    for each $x \in r$
        if $x[A] = (a_1, \ldots, a_n)$ and $x[\mathrm{T}_e]$ = $UC$
            for each $t \in cvr$
                if $x[\mathrm{V}] \cap t \neq \emptyset$
                    $cvr \leftarrow (cvr - t) \cup (t - x[\mathrm{V}])$;
    for each $t \in cvr$
        $z[A] \leftarrow (a_1, \ldots, a_n)$;
        $z[\mathrm{T}_s] \leftarrow c_t$; $z[\mathrm{T}_e] \leftarrow UC$;
        $z[\mathrm{V}_s] \leftarrow t[s]$; $z[\mathrm{V}_e] \leftarrow t[e]$;
        $r \leftarrow r \cup \{z\}$;
    return $r$
```

```
delete($r, (a_1, \ldots, a_n), c_t$):
    for each $x \in r$
        if $x[A] = (a_1, \ldots, a_n)$ and $x[\mathrm{T}_e] = UC$
            $x[\mathrm{T}_e] \leftarrow c_t$;
    return $r$
```

The function $cover_v$ in the `insert` routine returns a set of valid-time intervals (each a set of contiguous valid-time chronons). The routine first reduces the valid time elements, produced by the covering function, to avoid overlap with the valid times of existing tuples that have a transaction time extending to UC and that are value equivalent to the one to be inserted. Then, one tuple is inserted for each of the remaining valid-time intervals. The `delete` routine simply replaces the transaction end time with the current time, c_t.

As for the conceptual data model, `modify` is simply a combination of `delete` and `insert`.

6.2 Jensen's Backlog-Based Representation Scheme

The previous representation scheme presented a very natural and frequently used way of representing a bitemporal relation by a snapshot relation.

In the backlog-based representation scheme, bitemporal relations are represented by backlogs, which are also 1NF relations [Kim78, JMRS93]. The most important difference between this and the previous schemes is that tuples in backlogs are never updated, i.e., backlogs are append-only. Therefore, this representation scheme is well-suited for log-based storage of bitemporal relations, and it admits the possibility of using cheap write-once optical disk storage devices. This is highly desirable since the information content of bitemporal relations is ever-growing, resulting in very large relations.

A bitemporal relation schema $\mathcal{R} = (A_1, \ldots, A_n \,|\, \mathrm{T})$ is represented by a backlog relation schema R as follows.

$$R \;=\; (A_1, \ldots, A_n, \mathrm{V}_s, \mathrm{V}_e, \mathrm{T}, \mathrm{Op})$$

As in the previous representation scheme, the attributes V_s and V_e store starting and ending valid-time chronons, respectively. Attribute T stores the transaction time when the tuple was inserted into the backlog. Tuples, termed change requests, are either insertion requests or deletion requests, as indicated by the values, I, and D, of attribute Op. The fact in an insertion request is current starting at its transaction timestamp and until a matching deletion request with the same explicit and valid-time attribute values is recorded. Modifications are recorded by a pair of a deletion request and an insertion request, both with the same T value.

Example: The backlog relation corresponding to the conceptual relation in Figure 15 is shown below.

Emp	Dept	V_s	V_e	T	Op
Jake	Ship	10	15	5	*I*
Jake	Ship	10	15	10	*D*
Jake	Ship	5	20	10	*I*
Jake	Ship	5	20	15	*D*
Jake	Ship	10	15	15	*I*
Jake	Ship	10	15	20	*D*
Jake	Load	10	15	20	*I*
Kate	Ship	25	30	20	*I*

□

Next, we consider the conversion between a bitemporal relation and its backlog representation. The first function, `conceptual_to_back`, takes a conceptual relation as its first argument. The second argument is an arbitrary covering function as described in Section 6.1. The result is a backlog relation. Each conceptual tuple, x, is treated in turn. For each rectangle of bitemporal chronons in the cover of the timestamp of x, an insertion request is appended to the result. Further, if the rectangle has an ending transaction time different from UC then a deletion request is inserted.

```
conceptual_to_back($r'$, $cover$):
    $r \leftarrow \emptyset$;
    for each $x \in r'$
        for each $t \in cover(x[T])$
            $z[A] \leftarrow x[A]$;
            $z[V_s] \leftarrow min\_2(t)$; $z[V_e] \leftarrow max\_2(t)$;
            $z[Op] \leftarrow I$; $z[T] \leftarrow min\_1(t)$;
            $r \leftarrow r \cup \{z\}$;
            if $max\_1(t) \neq UC$
                $z[Op] \leftarrow D$; $z[T] \leftarrow max\_1(t)$;
                $r \leftarrow r \cup \{z\}$;
    return $r$;
```

```
back_to_conceptual($r, c_t$):
    $r' \leftarrow \emptyset$;
    for each $z_1 \in r$
        if $z_1[Op] = I$
            $a \leftarrow z_1[V_s]$; $b \leftarrow z_1[V_e]$;
            $c \leftarrow z_1[T]$; $d \leftarrow c_t + 1$;
            $x_1[A] \leftarrow z_1[A]$;
            $r \leftarrow r - \{z_1\}$;
            for each $z_2 \in r$
                if $z_2[A] = z_1[A]$ and $z_2[V] = z_1[V]$ and
                        $z_2[Op] = D$ and $z_1[T] < z_2[T] < d$
                    $d \leftarrow z_2[T]$;
                    $z_3 \leftarrow z_2$;
            if $d \neq c_t + 1$
                $r \leftarrow r - \{z_3\}$;
            $x_1[T] \leftarrow bi\_chr([c, d], [a, b])$;
            if $d = c_t + 1$
                $x_1[T] \leftarrow x_1[T] \cup \{UC\} \times \{a, \ldots, b\}$;
            for each $x_2 \in r'$
                if $x_2[A] = x_1[A]$
                    $x_1[T] \leftarrow x_1[T] \cup x_2[T]$;
                    $r' \leftarrow r' - \{x_2\}$;
            $r' \leftarrow r' \cup \{x_1\}$;
    return $r'$;
```

The second function, `back_to_conceptual`, is the inverse transformation. It is rather complex because not only is information about a single fact spread over a set of update requests, but, depending on the covering function, a single bitemporal chronon may be represented in multiple change requests. The change requests in the argument backlog relation are treated in turn. First, an insertion request is located, and its attribute values are recorded as appropriate. It is initially assumed that the information recorded by the insertion request is still current, indicated by the ending transaction-time value, $c_t + 1$, where, as before, c_t represents the current transaction time. Note that all transaction times in the backlog must be smaller than $c_t + 1$.

In the second loop, the backlog is scanned for a matching deletion request with a larger transaction time. If more than one exists, the earliest is chosen. If no such deletion request exists, denoted when $d = c_t + 1$, then the fact is still current. Now, the correct rectangular region of bitemporal chronons has been computed, and this can be recorded in the bitemporal conceptual relation. If other chronons have already been computed and recorded for the same fact, the two sets of chronons are simply merged.

As before, we claim that the transformation functions are inverses of each other. Briefly, consider a tuple x in the conceptual relation r'. The function `conceptual_to_back` produces a set of value-equivalent change requests, depending on the covering of $x[\mathrm{T}]$. Note that each x must produce at least one change request, and if a change request is value-equivalent to x then it must have been produced from x, otherwise value-equivalent conceptual tuples were present. The reverse transformation, `back_to_conceptual`, produces a single conceptual tuple from each set of value-equivalent change requests in the argument backlog. It can be shown that the same conceptual relation is produced.

As expected, insertion into backlogs, where tuples are never changed, is straightforward. For each set of consecutive valid-time chronons returned by the argument covering function, an insertion request with the appropriate attribute values is created. The current transaction time is assumed to be c_t.

Deletion follows the same pattern, the only complication being that a deletion request can only be inserted if a value-equivalent, previously entered and so far undeleted insertion request is found. First, the backlog is scanned to locate a matching insertion request. Second, it is ensured that the located insertion request has not previously been deleted. For every undeleted, matching insertion request that is found, a deletion request is inserted.

```
insert($r, (a_1, \ldots, a_n), t_v, cover_v, c_t$):
    for each $t \in cover_v(t_v)$
        $r \leftarrow r \cup \{(a_1, \ldots, a_n, min(t), max(t), c_t, I)\}$;
    return $r$;
```

```
delete($r, (a_1, \ldots, a_n), c_t$):
    $r' \leftarrow r$;
    for each $x_1 \in r$
        if $x_1[A] = (a_1, \ldots, a_n)$ and $x_1[\mathrm{Op}] = I$
            $found \leftarrow$ TRUE;
            for each $x_2 \in r$
                if $x_2[A] = x_1[A]$ and $x_2[V] = x_1[V]$ and
                        $x_2[\mathrm{OP}] = D$ and $x_2[T] > x_1[T]$
                    $found \leftarrow$ FALSE;
            if $found$
                $r' \leftarrow r' \cup \{(a_1, \ldots, a_n, x_1[V_s], x_1[V_e], c_t, D)\}$;
    return $r'$;
```

6.3 Gadia's Attribute Value Timestamped Representation Scheme

Non-1NF representations group all information about an object within a single tuple. As such, attribute-value timestamped representations have become popular for their flexibility in data modeling. We describe here how to represent conceptual relations by non-1NF attribute-value timestamped relations [Gad92]. This representational model was denoted Gadia-3 in Section 3. As Clifford, et al. have noted, while the Gadia-3 is temporally grouped [CCT94], Gadia's algebra is defined in terms of a snapshot interpretation semantics [Gad88]. It is that semantics that we capture here.

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 bitemporal relation schema $\mathcal{R}$ is represented by an attribute-value timestamped relation schema R as follows.

$$R \;=\; (\{([\mathrm{T}_s, \mathrm{T}_e] \;\times\; [\mathrm{V}_s, \mathrm{V}_e]\; A_1)\}, \ldots, \{([\mathrm{T}_s, \mathrm{T}_e] \;\times\; [\mathrm{V}_s, \mathrm{V}_e]\; A_n)\})$$

A tuple is composed of n sets. Each set element a is a triple of a transaction-time interval $[\mathrm{T}_s, \mathrm{T}_e]$, a valid-time interval $[\mathrm{V}_s, \mathrm{V}_e]$, representing in concert a rectangle of bitemporal chronons, and an attribute value, denoted $a.val$. As shorthand we will use T to denote the transaction time interval $[\mathrm{T}_s,\mathrm{T}_e]$, and, similarly, V for $[\mathrm{V}_s,\mathrm{V}_e]$, and will refer to them as $a.\mathrm{T}$ and $a.\mathrm{V}$, respectively.

Example: In an attribute value timestamped representation, the grouping of information within a tuple can be based on the value of any attribute or set of attributes. For example, we could represent the conceptual relation in Figure 15 by grouping on the employee attribute. Then all information for an employee is contained within a single tuple, as shown below.

Emp		Dept	
[5,9] × [10,15]	Jake	[5,9] × [10,15]	Ship
[10,14] × [5,20]	Jake	[10,14] × [5,20]	Ship
[15,19] × [10,15]	Jake	[15,19] × [10,15]	Ship
[20,UC] × [10,15]	Jake	[20,UC] × [10,15]	Load
[20,UC] × [25,30]	Kate	[20,UC] × [25,30]	Ship

A tuple in the above relation shows all departments for which a single employee has worked. A different way to view the same information is to perform the grouping by department. A single tuple then contains all information for a department, i.e., the full record of employees who have worked for the department.

Emp		Dept	
[5,9] × [10,15]	Jake	[5,9] × [10,15]	Ship
[10,14] × [5,20]	Jake	[10,14] × [5,20]	Ship
[15,19] × [10,15]	Jake	[15,19] × [10,15]	Ship
[20,UC] × [25,30]	Kate	[20,UC] × [25,30]	Ship
[20,UC] × [10,15]	Jake	[20,UC] × [10,15]	Load

Grouping by both attributes groups together all information for one employee and one department in a single tuple. This yields three tuples, as shown next.

Emp		Dept	
[5,9] × [10,15]	Jake	[5,9] × [10,15]	Ship
[10,14] × [5,20]	Jake	[10,14] × [5,20]	Ship
[15,19] × [10,15]	Jake	[15,19] × [10,15]	Ship
[20,UC] × [10,15]	Jake	[20,UC] × [10,15]	Load
[20,UC] × [25,30]	Kate	[20,UC] × [25,30]	Ship

This notion of restructuring provides flexibility. One user may want to focus on employees and will then use the grouping on employees. Another user may want to investigate departments and then uses the grouping on department. Finally, users may want to study the relationships between employees and departments, in which case the last format above may be advantageous. □

Next we consider the conversion between a conceptual relation and an attribute-value timestamped representation. The first function, `conceptual_to_att`, takes three arguments, r', a conceptual relation, *cover*, a covering function, and *group*, a grouping function. Arguments r' and *cover* are as described for the other representation schemes. Argument *group* partitions r' into disjoint subsets where all tuples in a subset agree on the values of a particular attribute or set of attributes, as illustrated in the above example. Each group of conceptual tuples produces one representation tuple.

```
conceptual_to_att($r'$,$cover$,$group$):
    $s \leftarrow \emptyset$;
    $G \leftarrow group(r')$;
    for each $g \in G$
        $z \leftarrow (\emptyset, \ldots, \emptyset)$;
        for each $x \in g$
            for each $t \in cover(x[\mathrm{T}])$
                for $i \leftarrow$ 1 to $n$
                    $z[A_i] \leftarrow z[A_i] \cup$
                        $\{([min\_1(t), max\_1(t)]$ '$\times$'
                            $[min\_2(t), max\_2(t)]$ $x[A_i])\}$;
        $s \leftarrow s \cup \{z\}$;
    return $s$;
```

```
att_to_conceptual($r$):
    $s \leftarrow \emptyset$;
    for each $z \in r$
        for $i \leftarrow$ 1 to $n$
            $g[i] \leftarrow \emptyset$;
            for each $y \in z[A_i]$
                $t \leftarrow bi\_chr(y.\mathrm{T}, y.\mathrm{V})$;
                $z[A_i] \leftarrow z[A_i] - \{y\}$;
                for each $y' \in z[A_i]$
                    if $y.val = y'.val$
                        $t \leftarrow t \cup bi\_chr(y'.\mathrm{T}, y'.\mathrm{V})$;
                        $z[A_i] \leftarrow z[A_i] - \{y'\}$;
                $g[i] \leftarrow g[i] \cup \{(y.val, t)\}$;
        for each $(a_1, a_2, \ldots, a_n) \in facts(g)$
            $t \leftarrow a_1.t$;
            for $i \leftarrow$ 2 to $n$
                $t \leftarrow t \cap a_i.t$;
            if $t \neq \emptyset$
                for $i \leftarrow$ 1 to $n$
                    $x[A_i] \leftarrow a_i.val$;
                $x[\mathrm{T}] \leftarrow t$;
                $s \leftarrow s \cup \{x\}$;
    return $s$;
```

The second function, `att_to_conceptual`, performs the inverse transformation. Given an attribute-value timestamped representation, it produces the equivalent conceptual relation. If we regard the transaction/valid times associated with an attribute value as rectangles, then the function simply constructs these rectangles for each attribute value in a tuple and then uses intersection semantics to determine the equivalent tuple timestamp. In this transformation, the grouping is ignored.

In the above, the *facts* function computes, for an array of attribute value/rectangle sets, all combinations of facts that can be constructed from those attribute values.

$$facts(g) = \{((a_1, t_1), (a_2, t_2), \ldots, (a_n, t_n)) \mid \forall i \; 1 \leq i \leq n((a_i, t_i) \in g[i])\}$$

As before the function *bi_chr* computes the bitemporal chronons represented by a given rectangle.

As for the previous representational models, the conversion functions perform inverse transformations. As an outline of a proof, note that `conceptual_to_att` produces, for each set of grouped conceptual tuples, a single attribute-value timestamped tuple. This representational tuple has homogeneous timestamps (identical temporal elements for each attribute), since the conceptual tuples that produced it were trivially homogeneous, being tuple timestamped. In the reverse transformation performed by `att_to_conceptual` this representational tuple is exploded into the group of conceptual tuples that formed it.

Insertion of a fact into an attribute-value timestamped relation can result in either of two actions. Either the new information is merged into an existing tuple $x \in r$ or no such x exists and the creation of an entirely new tuple is required.

The former case occurs when r is grouped so that x matches the explicit attribute values in exactly the grouping attributes, G. (We note again that this grouping is entirely syntactic, and doesn't render the data model grouped [CCT94].) Placing the new information into x preserves the grouped structuring of relation. For any given attribute value $x[A_i]$, some or all of the information being inserted may already be present in $x[A_i]$. A triple y containing such information must match the information being inserted in the explicit attribute value a_i, be current in the database, and overlap in valid-time. We remove all such overlapping valid-times chronons, perform a covering of the remaining chronons, and insert triples into $x[A_i]$ for each element of the covering.

In the latter case, no tuple with matching grouping attributes is found. The new information cannot be merged into an existing tuple without violating the grouped structure of the relation. Therefore, a new tuple containing only the added information is created.

```
insert($r, (a_1, \ldots, a_n), t_v, cover_v, c_t$):
    $found \leftarrow$ FALSE;
    for each $x \in r$
        if $x[G] = (a_1, \ldots, a_n)[G]$;
            $found \leftarrow$ TRUE;
            for $i \leftarrow$ 1 to $n$
                $t' \leftarrow t_v$;
                for each $y \in x[A_i]$
                    if $y.val = a_i$ and $y.\mathrm{T}[e] = UC$
                        $t' \leftarrow t' - \{y.\mathrm{V}\}$;
                for each $t \in cover_v(t')$
                    $x[A_i] \leftarrow x[A_i] \cup$ $\{([c_t, UC]$ '×' $[min(t), max(t)]\ a_i)\}$;
    if $found =$ FALSE
        for each $t \in cover_v(t_v)$
            $r \leftarrow r \cup$ $\{\{([c_t, UC]$ '×' $[min(t), max(t)]\ a_1)\} \ldots \{([c_t, UC]$ '×' $[min(t), max(t)]\ a_n)\}\}$;
    return $r$;
```

Deletion is more complicated. Removing a fact $(a_1, \ldots, a_n)$ from an attribute-valued timestamped relation r involves locating the tuple x containing the fact, if such an x exists, and altering x to reflect that the fact is no longer current. As we are interested only in current information, i.e., when $(a_1, \ldots, a_n)$ is current in the database, the triples in the attribute values of x that can participate in producing the fact must all have an ending transaction time of UC. The function *current* produces tuples from x representing the current information contained in x. It selects triples from each $x[A_i]$, $1 \leq i \leq n$, with an ending transaction time of UC and performs a Cartesian product, resulting in a relation whose tuples have attribute values each containing a single triple.

$$current(x) = \{((t_1v_1a_1), (t_2v_2a_2), \ldots, (t_nv_na_n)) \mid \forall i\ 1 \leq i \leq n((t_iv_ia_i) \in x[A_i] \wedge UC \in t_i)\}$$

Each tuple y potentially has information that must be deleted from the current database state. This is the case if the explicit-attribute values of y match $(a_1, \ldots, a_n)$, and y contains a rectangle in bitemporal space where each of the triples $(t_iv_ia_i)$, $1 \leq i \leq n$, overlap. For each such y, we insert triples indicating that the fact has been deleted from the current database state, and, with the help of a covering function, reinsert unaffected information back into the relation.

```
delete($r, (a_1, \ldots, a_n), cover_v, c_t$):
    for each $x \in r$
        $z[A_i] \leftarrow \emptyset$;  ...  $z[A_n] \leftarrow \emptyset$;
        for each $y \in current(x)$
            if $y[A_1].val = a_1$ and ... and $y[A_n].val = a_n$
                $t_1 \leftarrow bi\_chr(y[A_1].\mathrm{T}, y[A_1].\mathrm{V})$;  ...  $t_n \leftarrow bi\_chr(y[A_n].\mathrm{T}, y[A_n].\mathrm{V})$;
                $t \leftarrow t_1 \cap \ldots \cap t_n$;
                if $t \neq \emptyset$
                    for $i \leftarrow$ 1 to $n$
                        $x[A_i] \leftarrow x[A_i] - \{y[A_i]\}$;
                        $x[A_i] \leftarrow x[A_i] \cup \{([min_1(t), c_t - 1]$ '×' $[min_2(t), max_2(t)]\ y[A_i].val)\}$;
                        for each $t' \in cover_v(t_i - t)$
                            $x[A_i] \leftarrow x[A_i] \cup \{([min_1(t'), max_1(t')]$ '×' $[min_2(t), max_2(t)]\ y[A_i].val)\}$;
    return $r$;
```

As before, `modify` is simply a combination of `insert` and `delete`.

6.4 McKenzie's Attribute Value Timestamped Representation Scheme

Like the representation of the previous section, McKenzie's data model uses non-1NF attribute-value timestamping [McK88, MS91].

In McKenzie's model, a bitemporal relation is a sequence of valid-time states indexed by transaction time. Tuples within a valid-time state are attribute-value timestamped. The timestamps associated with each attribute value are sets of chronons, i.e., valid-time elements. In addition, the model does not assume homogeneity—attributes within the same tuple may have different timestamps.

A bitemporal relation schema $\mathcal{R} = (A_1, \ldots, A_n \mid \mathrm{T})$ is represented by an attribute valued timestamped relation schema R as follows.

$$R \;=\; (\mathrm{T}, \mathrm{VR})$$

where VR is a valid-time relation, and T is the transaction time when VR became current in the database. Stepwise-constant semantics are assumed.

The schema of the valid-time state VR is as follows.

$$\mathrm{VR} \;=\; (A_1\mathrm{V}_1, \ldots, A_n\mathrm{V}_n)$$

Here $A_1, \ldots, A_n$ are explicit attribute values. Associated with each A_i, $1 \leq i \leq n$, is a valid-time element V_i denoting when A_i was true in the modeled reality.

Example: The sequence of valid-time states indexed by transaction time corresponding to the conceptual relation in Figure 15 is shown below.

T	VR
0	$\emptyset$
5	{(Jake {10,...,15}, Ship {10,...,15})}
10	{(Jake {5,...,20}, Ship {5,...,20})}
15	{(Jake {10,...,15}, Ship {10,...,15})}
20	{(Jake {10,...,15}, Load {10,...,15}), (Kate {25,...,30}, Ship {25,...,30})}

Notice that for each tuple in each valid-time state, the timestamps associated with the attribute values in a tuple are identical, i.e., the timestamps are homogeneous. As mentioned above, this is not required by the model, but in our example the values of the attributes Emp and Dept change synchronously, hence the timestamps associated with each are identical. □

Next, we consider the conversion between a bitemporal relation and its representation as a sequence of valid-time states in McKenzie's data model. As before, we exhibit two functions. The first maps conceptual instances into representational instances, and the second performs the inverse transformation.

```
conceptual_to_att2($r', c_t$):
    $r \leftarrow \emptyset$;
    $uc\_present \leftarrow$ FALSE;
    for each $x \in r'$
        for each $(t, v) \in reduce(x[\mathrm{T}])$;
            if $t = UC$
                $uc\_present \leftarrow$ TRUE;
            else
                for $i \leftarrow$ 1 to $n$
                    $z[A_i] \leftarrow x[A_i]$;
                    $z[\mathrm{T}_i] \leftarrow v$;
                $r \leftarrow r \cup \{(t, \{z\})\}$;
    if not $uc\_present$
        $r \leftarrow r \cup \{(c_t, \emptyset)\}$;
    $r \leftarrow r \cup \{(0, \emptyset)\}$;
    $r \leftarrow group(r)$;
    return $r$;
```

```
att2_to_conceptual($r, c_t$):
    for each $(t, vr) \in r$
        $vr \leftarrow homogenize(vr)$;
    $reverse\_sort(r)$;
    $r' \leftarrow \emptyset$;
    $(t, vr) \leftarrow next(r)$;
    for each $y \in vr$
        $z[A] \leftarrow y[A]$;
        $z[\mathrm{T}] \leftarrow bi\_chr(\{t..c_t - 1, UC\}, y[\mathrm{V}])$;
        $r' \leftarrow r' \cup \{z\}$;
    $t_{last} \leftarrow t$;  $(t, vr) \leftarrow next(r)$;
    while $(t, vr) \neq \perp$
        for each $y \in vr$
            $found \leftarrow$ FALSE;
            for each $z' \in r'$
                if $z'[A] = y[A]$
                    $z'[\mathrm{T}] \leftarrow z'[\mathrm{T}] \cup bi\_chr(\{t..t_{last} - 1\}, y[\mathrm{V}])$;
                    $found \leftarrow$ TRUE;
            if not $found$
                $z[A] \leftarrow y[A]$
                $z[\mathrm{T}] \leftarrow bi\_chr(\{t..t_{last} - 1\}, y[\mathrm{V}])$;
                $r' \leftarrow r' \cup \{z\}$;
        $t_{last} \leftarrow t$;  $(t, vr) \leftarrow next(r)$;
    return $r'$;
```

The first function, `conceptual_to_att2`, takes a conceptual relation as its first argument and returns a sequence of valid-time relations, indexed by transaction time, in McKenzie's data model. A conceptual tuple x can contribute possibly many tuples to the result, with the generated tuples residing in possibly many different valid-time states. For example, the first tuple in the conceptual relation of Figure 15 would contribute three tuples, (Jake {10,...,15}, Ship {10,...,15}), (Jake {5,...,20}, S hip {5,...,20}), and (Jake {10,...,15}, Ship {10,...,15}), in the val id-time states associated with transaction times 5, 10 and 15, respectively. Value-equivalent tuples with identical valid-timestamps but at intermediate transaction times, e.g., (Jake {10,...,15}, Ship {10,...,15}) at transaction time 6, are not generated.

We accomplish this by deriving for each conceptual tuple x a set of stepwise constant states from its bitemporal element $x[\mathrm{T}]$. The result is a set of pairs (t,v), the first element being a transaction time and the second being a valid-time element. Effectively, each (t,v) denotes the state of $x[A]$ as being valid during the set v at the transaction time t. Intermediate states are not included in the computed set of pairs, effectively preserving the stepwise constant assumption.

The set of stepwise constant states is computed by the function `reduce` shown below. For the above example, `reduce` returns the set {(5,{10,...,15}), (10,{5,...,20}), (15,{10,...,15})}. The function `next_state` is called by `reduce`; it examines each bitemporal chronon in the timestamp and derives a state (t,v) where t is the earliest transaction time present in the timestamp, and v is the set containing exactly those valid-time chronons associated with t.

```
reduce($T$):
    $T' \leftarrow \emptyset$;
    while $T \neq \emptyset$
        $(t, v) \leftarrow next\_state(T)$;
        $T' \leftarrow T' \cup \{(t, v)\}$;
        $T \leftarrow T - bi\_chr(\{t\}, v)$;
        $t' \leftarrow t + 1$;
        while $(t', v) = next\_state(T)$
                $T \leftarrow T - bi\_chr(\{t'\}, v)$;
                $t' \leftarrow t' + 1$;
    return $T'$;
```

```
next_state($T$):
    $v \leftarrow \emptyset$;
    $t \leftarrow UC$;
    for each $b \in T$
        if $b.\mathrm{T} < t$
            $v \leftarrow \{b.\mathrm{V}\}$;
            $t \leftarrow b.\mathrm{T}$;
        else
            if $b.\mathrm{T} = t$
                $v \leftarrow v \cup \{b.\mathrm{V}\}$;
    return $(t, v)$;
```

For a given pair (t,v), a tuple is generated and placed in a valid-time state indexed by the transaction time t. The end result is a set of pairs of single tuple valid-time states indexed at the given by a transaction time.

Finally, the function `group` collapses all pairs with identical transaction-time components into a single valid-time state, indexed at the given transaction time.

```
group($r$):
    $S \leftarrow \emptyset$;
    for each $(t, vr) \in r$;
        $found \leftarrow$ FALSE;
        for each $(t', vr') \in S$
            if $t = t'$
                $S \leftarrow S - (t', vr')$;
                $S \leftarrow S \cup \{(t', vr' \cup vr)\}$;
                $found \leftarrow$ TRUE;
        if not $found$
            $S \leftarrow S \cup \{(t, vr)\}$;
    return $S$;
```

The second function, `att2_to_conceptual`, performs the inverse transformation. It takes a sequence of valid-time states r, indexed by transaction time, and produces the equivalent conceptual relation.

As the valid-time states of r may contain tuples with non-homogeneous timestamps, we first transform each input valid-time state into an equivalent tuple-timestamped relation. This is the purpose of function `homogenize` shown below. For each tuple $x \in vr$, `homogenize` generates possibly many result tuples, one for each valid-time chronon present in a timestamp associated with an attribute value of x. The function determines the maximal set of attribute values simultaneously valid during that chronon, and generates a result tuple, whose tuple-timestamp contains the single chronon.

```
homogenize($vr$):
    $vr_h \leftarrow \emptyset$;
    for each $x \in vr$
        for $i \leftarrow 1$ to $n$
            for each $v \in x[\mathrm{V}_i]$
                $z[A_1] \leftarrow \perp$; ...   $z[A_n] \leftarrow \perp$;
                $z[A_i] \leftarrow x[A_i]$;
                $z[\mathrm{V}] \leftarrow v$;
                for $j \leftarrow$ 1 to $n$
                    if $j \neq i$ and $v \in x[\mathrm{V}_j]$
                        $z[A_j] \leftarrow x[A_j]$
                $vr_h \leftarrow vr_h \cup \{z\}$;
    return $coalesce(vr_h)$;
```

```
coalesce($vr$):
    $vr' \leftarrow \emptyset$;
    for each $x \in vr$
        $vr \leftarrow vr - \{x\}$;
        for each $y \in vr$
            if $x[A] = y[A]$
                $x[\mathrm{V}] \leftarrow x[\mathrm{V}] \cup y[\mathrm{V}]$;
                $vr \leftarrow vr - \{y\}$;
        $vr' \leftarrow vr' \cup \{x\}$;
    return $vr'$;
```

As many value-equivalent tuples may be produced, function `coalesce` is used to collapse such tuples into a single tuple. The timestamps of matching tuples are unioned into a single result tuple.

The valid-time states of r are then processed from latest to earliest in transaction time order; the pairs $(t, vr) \in r$ are sorted into descending order of t, and a function `next` returns the next (t, vr) in the sorted order. The current valid-time state is treated specially to accommodate the stepwise constant semantics between the time the state was stored, the current transaction time, and UC.

The remaining valid-time states are converted as follows. For a tuple $x \in vr$, its bitemporal timestamp is generated using the appropriate range of transaction-time and valid-time element associated with the tuple. However, since value-equivalent tuples may be present in different valid-time states, we must consolidate the information in such tuples within one resulting conceptual tuple. If a value-equivalent tuple z' is already present in the result, we augment its timestamp with the generated bitemporal element. Otherwise, a new tuple is inserted.

As for the previous representational models, it is possible to construct a proof showing that the functions truly perform the inverse transformations. A possible argument would show that `conceptual_to_att2` explodes each conceptual tuple into value-equivalent tuples in possibly many valid-time states. In the reverse transformation, these value-equivalent tuples are coalesced and any "holes" in the timestamp corresponding to intermediate transaction times are filled in.

We now show how the semantics of bitemporal update are supported within this representation. Insertion of a fact into the database involves the creation of a new current state containing the fact and the time that it was, is, or will be valid. This state is constructed in one of two ways. If the valid-time state current at the time of the insertion contains a value-equivalent tuple, the timestamps of that tuple are augmented to reflect the new information. Otherwise a new tuple is inserted. In both cases, the updated valid-time state is inserted into r indexed by the current transaction time, c_t. The function *rollback* simply returns the valid time state in r current during the argument transaction time. For example, if r is the sequence of valid-time states shown in the previous example then $rollback(r, 11)$ returns the valid-time state {(Jake {5,...,20}, Ship {5,...,20})}.

```
insert($r, (a_1, \dots, a_n), t_v, c_t$):
    $vr \leftarrow rollback(r, c_t)$;
    $found \leftarrow$ FALSE;
    for each $x \in vr$
        if $x[A] = (a_1, \dots, a_n)$
            for $i \leftarrow$ 1 to $n$
                $x[\mathrm{T}_i] \leftarrow x[\mathrm{T}_i] \cup t_v$;
            $found \leftarrow$ TRUE;
    if not $found$
        $vr \leftarrow vr \cup (a_1 t_v, \dots, a_n t_v)$;
    $r \leftarrow r \cup \{(c_t, vr)\}$;
    return $r$;
```

```
delete($r, (a_1, \dots, a_n), c_t$):
    $vr \leftarrow rollback(r, c_t)$;
    for each $x \in vr$
        if $x[A] = (a_1, \dots, a_n)$
            $t \leftarrow x[\mathrm{T}_1] \cap \dots \cap x[\mathrm{T}_n]$;
            if $t \neq \emptyset$
                for $i \leftarrow$ 1 to $n$
                    $x[\mathrm{T}_i] \leftarrow x[\mathrm{T}_i] - t$;
                if $x[\mathrm{T}_1] = \emptyset$ and ... and $x[\mathrm{T}_n] = \emptyset$
                    $vr \leftarrow vr - \{x\}$;
                $r \leftarrow r \cup \{(c_t, vr)\}$;
    return $r$;
```

Deletion of a fact involves the removal of the fact from the current valid-time state if it exists, and no action otherwise. A fact to be deleted is present in a tuple x, if the explicit attribute values of x match $(a_1, \dots, a_n)$ and the intersection of the valid-time elements associated with the attribute values of x is non-empty. We delete from each timestamp the computed intersection, and remove the entire tuple if all resulting timestamps are empty.

6.5 Ben-Zvi's Tuple Timestamped Representation Scheme

Like the representational model in Section 6.1, Ben-Zvi's data model is a 1NF tuple-timestamping model. Appended to each tuple are five timestamp attributes [BZ82].

Let a bitemporal relation schema $\mathcal{R}$ have the attributes $A_1, \dots, A_n$, 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.

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

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 i n 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). Similarly, 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 15 is shown below.

Emp	*Dept*	T_{es}	T_{rs}	T_{ee}	T_{re}	T_d
Jake	Ship	10	5	15	5	10
Jake	Ship	5	10	20	10	15
Jake	Ship	10	15	15	15	20
Jake	Load	10	20	15	20	–
Kate	Ship	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. □

In the conversion functions presented next, the functions *min_1* and *min_2* select a minimum first and second component, respectively, in a set of binary tuples. The function *max_1* returns the symbol '–' if *UC* is encountered as a first component; otherwise, it returns a maximum first component. The function *max_2* selects a maximum second component. The function *bi_chr* may accept the symbol '–' as a transaction-time end value, in which case the symbol is treated as the current time. Bitemporal chronons with *UC* as first component are then generated. When '–' is encountered as a valid-time end, it is treated as the maximum valid-time value, c_{vt}^{∞}. Analogously, when '–' is encountered as a transaction-time value, it is treated as the current transaction time, c_t, as well as the value *UC*.

The first conversion function is very similar to the corresponding function in Section 6.1. The routine `conceptual_to_snap2` constructs an output tuple for each rectangle in a covering of a bitemporal element. The effective-start and effective-end timestamps are set to the minimum and maximum valid-time chronons in the rectangle, respectively. We set the times when the valid timestamps were stored to the minimal transaction time chronon in the rectangle. The deletion time of the tuple is set to the maximal transaction time of the rectangle (possibly *UC*), thereby denoting when the fact was last current in the relation.

```
conceptual_to_snap2($r'$, $cover$):
    $s \leftarrow \emptyset$;
    for each $x \in r'$
        $z[A] \leftarrow x[A]$;
        for each $t \in cover(x[\mathrm{T}])$
            $z[\mathrm{T}_{rs}] \leftarrow min\_1(t)$;
            $z[\mathrm{T}_{re}] \leftarrow z[\mathrm{T}_{rs}]$;
            $z[\mathrm{T}_d] \leftarrow max\_1(t)$;
            $z[\mathrm{T}_{es}] \leftarrow min\_2(t)$;
            $z[\mathrm{T}_{ee}] \leftarrow max\_2(t)$;
            $s \leftarrow s \cup \{z\}$;
    return $s$;
```

```
snap2_to_conceptual($r$):
    $s \leftarrow \emptyset$;
    for each $z \in r$
        $r \leftarrow r - \{z\}$;
        $x[A] \leftarrow z[A]$;
        $x[\mathrm{T}] \leftarrow make\_ts(z[\mathrm{T}_{es}], z[\mathrm{T}_{rs}], z[\mathrm{T}_{ee}], z[\mathrm{T}_{re}], z[\mathrm{T}_d])$;
        $s \leftarrow s \cup \{x\}$;
    return $coalesce(s)$;
```

The function `snap2_to_conceptual` performs the inverse transformation. It constructs one conceptual tuple for each set of value-equivalent tuples in the representation. Initially, each representational tuple is examined, and a conceptual tuple corresponding to that representational tuple is generated.

The function `make_ts` constructs a bitemporal element from the five timestamps in the representational tuple. There are three cases to consider. In each case, we construct a bitemporal element representing a rectangle or union of rectangles bounded by the argument time values.

First, if the effective-time start and effective-time end values were stored simultaneously, the associated element corresponds to a rectangular region bounded in valid time and possibly unbounded in transaction time. Similarly, if the values were not stored simultaneously, it may be the case that the effective-end time was never stored. This corresponds to a rectangular region that is unbounded in valid time and possibly bounded in transaction time, depending on if the tuple has been deleted.

Otherwise, both the effective-time start and the effective-time end values have been stored, and are unequal. The resulting region is unbounded in valid time between the times when the effective-time start and effective-time end were stored, and possibly bounded in transaction time, depending on if the tuple has been deleted.

Finally, function `coalesce` collapses each set of value-equivalent tuples in the result into a single tuple.

```
make_ts($t_{es}, t_{rs}, t_{ee}, t_{re}, t_d$):
    if $t_{rs} = t_{re}$
        $t \leftarrow bi\_chr(\{t_{rs}..t_d\}, \{t_{es}..t_{ee}\})$;
    else
        if $t_{re} =$ '–'
            $t \leftarrow bi\_chr(\{t_{rs}..t_d\}, \{t_{es}..c^{\infty}_{vt}\})$;
        else
            $t \leftarrow bi\_chr(\{t_{rs}..t_{re}\}, \{t_{es}..c^{\infty}_{vt}\}) \cup$
                        $bi\_chr(\{t_{re}..t_d\}, \{t_{es}..t_{ee}\})$;
    return $t$;
```

```
coalesce($r$):
    $r' \leftarrow \emptyset$;
    for each $x \in r$
        $r \leftarrow r - \{x\}$;
        for each $y \in r$
            if $x[A] = y[A]$
                $x[\mathrm{T}] \leftarrow x[\mathrm{T}] \cup y[\mathrm{T}]$;
                $r \leftarrow r - \{y\}$;
        $r' \leftarrow r' \cup \{x\}$;
    return $r'$;
```

As for the previous representational models, it is possible to construct a proof showing that the conversion functions truly perform inverse transformations. We outline a proof as follows. In the conversion performed by `snapshot2_to_conceptual`, a single conceptual tuple produces possibly many value-equivalent snapshot tuples, each with an associated rectangle produced by the covering function. In the reverse transformation, these value-equivalent tuples are coalesced back into the original conceptual tuple, and the bitemporal element for the resulting tuple is constructed from the rectangles associated with the representational tuples.

For the update routines, the most convenient covering function partitions on transaction time, and does not permit overlap.

```
insert($r, (a_1, \ldots, a_n), t_v, cover_v, c_t$):
    for each $t \in cover_v(t_v)$
        for each $x \in r$
            if $x[A] = (a_1, \ldots, a_n)$ and $x[\mathrm{T}_d] =$ '–' and
                    $x[\mathrm{T}_{es}, \mathrm{T}_{ee}] \cap t \neq \emptyset$
                $r \leftarrow r - \{x\}$;
                $x[\mathrm{T}_d] \leftarrow c_t$;
                $z[A] \leftarrow x[A]$;
                $z[\mathrm{T}_{es}] \leftarrow min(x[\mathrm{T}_{es}] \cup t)$;
                $z[\mathrm{T}_{ee}] \leftarrow min(x[\mathrm{T}_{ee}] \cup t)$;
                $z[\mathrm{T}_d] \leftarrow$ '–';
                $r \leftarrow r \cup \{x, z\}$;
    return $r$;
```

```
delete($r, (a_1, \ldots, a_n), c_t$):
    for each $x \in r$
        if $x[A] = (a_1, \ldots, a_n)$ and
                $x[\mathrm{T}_d] =$ '–'
            $x[\mathrm{T}_d] \leftarrow c_t$;
    return $r$;
```

6.6 Summary

We examined in detail five representations of bitemporal relations originally introduced in Section 3. For each of these data models, we showed how instances in the BCDM can be mapped to and from representational instances. The established correspondence between representations and the conceptual model is central to this proposal—the BCDM forms a unifying link between disparate relational bitemporal models. The mapping functions assign semantics to instances in the five representations and allows us to meaningfully compare instances of diverse models.

In the next section, we discuss in more detail the role of the BCDM with respect to data model unification, by formalizing the notion of semantic equivalence.

7 Semantic Equivalence

The previous section claimed that many semantically equivalent representations of the same conceptual relation may co-exist. In this and the next section, we explore the nature of this relationship between the conceptual data model and the representational data models. We focus next on the equivalence among the objects in the models; a following section will examine equivalence when operations on these objects is also considered.

7.1 Snapshot Equivalence

We use snapshot equivalence to formalize the notion of relation instances having the same information content.

Snapshot equivalence makes use of transaction and valid timeslice operators. We initially define these operators for BCDM relations, then for each of the five representational models described in Section 3.

The *transaction-timeslice* operator, ρ^{B}, takes two arguments, a bitemporal relation and a time value, the latter appearing as a subscript. The result is a valid-time relation. In order to explain the semantics of ρ^{B}, we describe its operation on a bitemporal conceptual relation. Each tuple is examined in turn. If any of its associated bitemporal chronons have a transaction time matching the argument time, the explicit attribute values, along with each of the valid-time chronons paired to a matching transaction time, become a tuple in the result. The transaction-timeslice operator may also be applied to a transaction-time relation, in which case the result is a snapshot relation.

The *valid timeslice* operator, τ^{B}, is very similar. It also takes two arguments, a bitemporal relation and a time value. The difference is that this operator does the selection on valid time and produces a transaction-time relation. The valid-timeslice operator may also be applied to a valid-time relation, in which case the result is a snapshot relation.

DEFINITION: Define a relation schema $R = (A_1, \ldots, A_n | \text{ T})$, and let r be an instance of this schema. Let t_2 denote an arbitrary time value and let t_1 denote a time not exceeding the current time. Then the transaction-timeslice and valid-timeslice operators may be defined as follows for the conceptual data model.

$$\begin{aligned}
\rho^{\text{B}}_{t_1}(r) &= \{z^{(n+1)} \mid \exists x \in r\ (z[A] = x[A] \wedge z[\text{T}_v] = \{t_2 \mid (t_1, t_2) \in x[\text{T}]\} \wedge z[\text{T}_v] \neq \emptyset)\} \\
\tau^{\text{B}}_{t_2}(r) &= \{z^{(n+1)} \mid \exists x \in r\ (z[A] = x[A] \wedge z[\text{T}_t] = \{t_1 \mid (t_1, t_2) \in x[\text{T}]\} \wedge z[\text{T}_t] \neq \emptyset)\}
\end{aligned}$$

□

The transaction-timeslice operator for transaction-time relations (ρ^{T}) and the valid-timeslice operator for valid-time relations (τ^{V}) are straightforward special cases.

We can now formally define snapshot equivalence so that it applies to each representational data model for which the valid-timeslice and transaction-timeslice operators have been defined.

DEFINITION: Two relation instances, r and s, are *snapshot equivalent*, $r \overset{\text{S}}{\equiv} s$, if for all times t_1 not exceeding the current time and for all times t_2, $\tau^{\text{V}}_{t_2}(\rho^{\text{B}}_{t_1}(r)) = \tau^{\text{V}}_{t_2}(\rho^{\text{B}}_{t_1}(s))$. □

The concept of snapshot equivalence is due to Gadia and was first defined for valid-time relations [Gad86b] and was later generalized to multiple dimensions [GY88]. We have chosen not to use the original term *weakly equivalent* to avoid confusion with the different notion of *weak equivalence* over algebraic expressions (e.g., [Ull82]). In the next section, we will discuss how snapshot equivalence may also be applied to pairs of instances when the instances belong to different models.

The following theorem states that identity and snapshot equivalence coincide for the conceptual model. It is a major source of semantic clarity that two instances have the same information content exactly when they are identical.

THEOREM **1** Let r and s be conceptual relations over the same schema. Then $r \overset{\text{S}}{\equiv} s$ if and only if $r = s$.

PROOF: First assume that $r \overset{\text{S}}{\equiv} s$. We show that for each $x \in r$, $x = (a_1, \ldots, a_n \mid t_x)$ there exists a $y \in s$, $y = (a_1, \ldots, a_n \mid t_y)$, with $t_x = t_y$.

By the definition of snapshot equivalence there exist tuples y_i, $i = 1, \ldots, m$, in s so that for all t_1, t_2, where t_1 does not exceed the current time, $\tau^{\text{V}}_{t_2}(\rho^{\text{B}}_{t_1}(\{x\})) = \tau^{\text{V}}_{t_2}(\rho^{\text{B}}_{t_1}(\{y_1, \ldots, y_m\}))$. The definitions of the involved operators demand that each of the y_i must have $a_1, \ldots, a_n$ as explicit attribute values. Further, the operators demand that $t_x = \cup_i\ t_{y_i}$. By definition of the BCDM, no two tuples with the same explicit attribute values may exist in an instance. Thus, $i = 1$ and $y_1 = y$, proving the claim. As a result, each tuple in r has an exact match in s. By the symmetrical argument, each tuple in s has a match in r, and the two instances are consequently identical.

In the other direction, assuming that $r = s$, clearly $\forall t_1$, t_2 where t_1 does not exceed the current time, $\tau^{\text{V}}_{t_2}(\rho^{\text{B}}_{t_1}(r)) = \tau^{\text{V}}_{t_2}(\rho^{\text{B}}_{t_1}(s))$. □

7.2 Rollback and Timeslice Operators

We now define the timeslice operators for each of the five representational models. These definitions extend the notion of snapshot equivalence to the corresponding representation. In the definitions, let t denote an arbitrary time value and let t' be a time value not exceeding the current time.

DEFINITION: (Snodgrass' Tuple Timestamped Data Model). Define a relation schema $R = (A_1, \ldots, A_n, \mathrm{T}_s, \mathrm{T}_e, \mathrm{V}_s, \mathrm{V}_e)$, and let r be an instance of this schema.

$$\begin{aligned}
\rho^{\mathrm{B}}_{t'}(r) &= \{z^{(n+2)} \mid \exists x \in r\ (z[A] = x[A] \wedge z[\mathrm{V}] = x[\mathrm{V}] \wedge t' \in x[\mathrm{T}])\} \\
\tau^{\mathrm{B}}_{t}(r) &= \{z^{(n+2)} \mid \exists x \in r\ (z[A] = x[A] \wedge z[\mathrm{T}] = x[\mathrm{T}] \wedge t \in x[\mathrm{V}])\}
\end{aligned}$$

□

DEFINITION: (Jensen's Backlog Data Model). Define a relation schema $R = (A_1, \ldots, A_n, \mathrm{V}_s, \mathrm{V}_e, \mathrm{T}, \mathrm{Op})$, and let r be an instance of this schema.

$$\begin{aligned}
\rho^{\mathrm{B}}_{t'}(r) = \{z^{(n+2)} \mid \exists x \in r\ (&z[A] = x[A] \wedge z[\mathrm{V}] = x[\mathrm{V}] \wedge x[\mathrm{T}] \leq t' \wedge x[\mathrm{Op}] = I \wedge \\
&(\neg \exists y \in r\ (y[A] = x[A] \wedge y[\mathrm{V}] = x[\mathrm{V}] \wedge y[\mathrm{Op}] = D \wedge x[\mathrm{T}] \leq y[\mathrm{T}] \leq t')))\} \\
\tau^{\mathrm{B}}_{t}(r) = \{z^{(n+2)} \mid \exists x \in r\ (&z[A] = x[A] \wedge z[\mathrm{T}] = x[\mathrm{T}] \wedge z[\mathrm{Op}] = x[\mathrm{Op}] \wedge t \in x[\mathrm{V}])\}
\end{aligned}$$

In the definition of transaction timeslice, an insertion request contributes to the result if it was entered before the argument transaction time t' and if it was not subsequently countered by a deletion request before t'. The non-symmetry of these two definitions underscores the emphasis accorded transaction time in this model. □

DEFINITION: (Gadia's Attribute Value Timestamped Data Model). Define a relation schema $R = (\{([\mathrm{T}_s, \mathrm{T}_e]\ \times\ [\mathrm{V}_s, \mathrm{V}_e]\ A_1)\}, \ldots, \{([\mathrm{T}_s, \mathrm{T}_e]\ \times\ [\mathrm{V}_s, \mathrm{V}_e]\ A_n)\})$, and let r be an instance of R.

$$\begin{aligned}
\rho^{\mathrm{B}}_{t'}(r) = \{z^{(n)} \mid \exists x \in r\ (\forall i\ (&i \in 1, \ldots, n \wedge \forall a \in x[A_i](t' \in a.\mathrm{T} \Rightarrow (a.\mathrm{V}\ a.val) \in z[A_i]) \wedge \\
&\forall b \in z[A_i](\exists a \in x[A_i](t' \in a.\mathrm{T} \wedge b.val = a.val \wedge b.\mathrm{V} = a.\mathrm{V}))))\} \\
\tau^{\mathrm{B}}_{t}(r) = \{z^{(n)} \mid \exists x \in r\ (\forall i\ (&i \in 1, \ldots, n \wedge \forall a \in x[A_i](t \in a.\mathrm{V} \Rightarrow (a.T\ a.val) \in z[A_i]) \wedge \\
&\forall b \in z[A_i](\exists a \in x[A_i](t \in a.\mathrm{V} \wedge b.val = a.val \wedge b.\mathrm{T} = a.\mathrm{T}))))\}
\end{aligned}$$

For each operator, the first line ensures that no chronon is left unaccounted for, and the second line ensures that no spurious chronons are introduced. □

DEFINITION: (McKenzie's Attribute Value Timestamped Data Model). Define a relation schema $R = (\mathrm{T}, \mathrm{VR})$, with T being a transaction timestamp and $\mathrm{VR} = (A_1\mathrm{V}_1, \ldots, A_n\mathrm{V}_n)$, where the A_i, $1 \leq i \leq n$, are explicit attributes and the corresponding V_i are valid-time elements. An instance of this schema is a sequence of valid-time states indexed by transaction times as. Let r be such an instance.

$$\begin{aligned}
&\rho^{\mathrm{B}}_{t'}(r) = \{z^{(n)} \mid \exists (t, vr) \in r\ (t' \leq t \wedge \neg \exists (t'', vr'') \in r\ (t' \leq t'' < t) \wedge z \in vr)\} \\
&\tau^{\mathrm{B}}_{t}(r) = \{(t'', S) \mid \forall s \in S\ (\exists t''\ ((t'', vr) \in r \wedge \forall x \in vr\ (\forall i\, 1 \leq i \leq n\ ((t \in x[\mathrm{V}_i] \Rightarrow s[A_i] = x[A_i]) \wedge \\
&\qquad (t \notin x[\mathrm{V}_i] \Rightarrow s[A_i] = \bot)) \wedge \exists i\, 1 \leq i \leq n\ (t \in x[\mathrm{V}_i])))))\}
\end{aligned}$$

The first operator extracts the valid time relation with the greatest transaction timestamp before t'. The second returns a rollback relation, a sequence of snapshot states such that each tuple in each snapshot state was valid at valid time t for all attributes. Some, but not all, attribute values in the tuples in the snapshot states may be null values. □

DEFINITION: (Ben-Zvi's Tuple Timestamped Data Model). Define a relation schema $R = (A_1, \ldots, A_n, \mathrm{T}_{es}, \mathrm{T}_{ee}, \mathrm{T}_{rs}, \mathrm{T}_{re}, \mathrm{T}_d)$, and let r be an instance of this schema.

$$\begin{aligned}\rho^{\mathrm{B}}_{t'}(r) = \{z^{(n+2)} \mid \exists x \in r\ (& z[A] = x[A] \wedge z[\mathrm{T}_{es}] = x[\mathrm{T}_{es}] \wedge x[\mathrm{T}_{rs}] \leq t' \wedge (x[\mathrm{T}_d] \neq \text{‘–’} \Rightarrow t' \leq x[\mathrm{T}_d]) \wedge \\ & ((x[\mathrm{T}_{re}] \neq \text{‘–’} \Rightarrow t' \leq x[\mathrm{T}_{re}]) \Rightarrow z[\mathrm{T}_{ee}] = \text{‘–’}) \wedge \\ & ((x[\mathrm{T}_{ee}] \neq \text{‘–’} \wedge x[\mathrm{T}_{re}] \leq t') \Rightarrow z[\mathrm{T}_{ee}] = x[\mathrm{T}_{ee}])\}\end{aligned}$$

$$\begin{aligned}\tau^{\mathrm{B}}_{t}(r) = \{z^{(n+2)} \mid \exists x \in r\ (& z[A] = x[A] \wedge z[\mathrm{T}_{rs}] = x[\mathrm{T}_{rs}] \wedge (\\ & (((x[\mathrm{T}_{es}] \leq t) \wedge (x[\mathrm{T}_{ee}] \neq \text{‘–’} \Rightarrow t \leq x[\mathrm{T}_{ee}])) \Rightarrow z[\mathrm{T}_{re}] = x[\mathrm{T}_d]) \vee \\ & ((x[\mathrm{T}_{ee}] \neq \text{‘–’} \wedge t \geq x[\mathrm{T}_{ee}] \wedge x[\mathrm{T}_{rs}] \neq x[\mathrm{T}_{re}]) \Rightarrow z[\mathrm{T}_{re}] = x[\mathrm{T}_{re}])))\}\end{aligned}$$

In the first operator, the complexity arises in computing T_{ee} for the resulting tuples; the other implicit attribute, T_{es}, is trivial. Two possibilities for T_{ee} exist, ‘–’ and $x[\mathrm{T}_{ee}]$, depending on the value of $x[\mathrm{T}_{re}]$. For the second operator, the complexity is in determining $z[\mathrm{T}_{re}]$, which can also assume two possible values, $x[\mathrm{T}_d]$ and $x[\mathrm{T}_{re}]$, depending primarily on the value of $x[\mathrm{T}_{ee}]$. □

For each of the five schemes, the transaction-timeslice operator for transaction-time relations (ρ^{T}) and the valid-timeslice operator for valid-time relations (τ^{V}) are straightforward special cases of these definitions. Note that the rollback and timeslice operators in the various representations all have the same names, ρ^{B}_t and τ^{B}_t.

The existence of the timeslice operators for the representational models has important implications, as we discuss in the following. Rather than providing theorems and proofs for each representational model, the theorems and proofs in the remainder of this section are limited to a single model only. Specifically, the tuple-timestamped model introduced in Section 6.1 is used due to its straightforward structure. Corresponding results hold for the remaining models; proofs may be similarly obtained.

There is no reason to apply ρ before τ in the definition of snapshot equivalence, as the following theorem states.

Theorem 2 Let r be a temporal relation. Then for all times t_1 not exceeding the current time and for all times t_2,

$$\tau^{\mathrm{V}}_{t_2}(\rho^{\mathrm{B}}_{t_1}(r)) \overset{\mathrm{S}}{\equiv} \rho^{\mathrm{T}}_{t_1}(\tau^{\mathrm{B}}_{t_2}(r)).$$

Proof: Let $x \in \tau^{\mathrm{V}}_{t_2}(\rho^{\mathrm{B}}_{t_1}(r))$; then there is a tuple y in $\rho^{\mathrm{B}}_{t_1}(r)$ with $y[A] = x[A]$ and $t_2 \in y[\mathrm{V}]$. This implies the existence of a tuple z in r so that $z[A] = y[A]$, $z[\mathrm{V}] = y[\mathrm{V}]$, and $t_1 \in z[\mathrm{T}]$. As $t_2 \in z[\mathrm{V}]$, there is a tuple u in $\tau^{\mathrm{B}}_{t_2}(r)$ for which $u[A] = z[A]$ and $u[\mathrm{T}] = z[\mathrm{T}]$. As $t_1 \in u[\mathrm{T}]$, there is a tuple v in $\rho^{\mathrm{T}}_{t_1}(\tau^{\mathrm{B}}_{t_2}(r))$ with $v[A] = u[A]$. By construction, $v = x$. Thus, a tuple on the lhs (left hand side) is also on the rhs (right hand side). Proving the opposite inclusion is similar and omitted. Combining the inclusions proves the equivalence. □

Snapshot equivalence precisely captures the notion that relation instances in the chosen representation scheme have the same information content. More precisely, all representations of the same bitemporal conceptual relation are snapshot equivalent, and two bitemporal relations that are snapshot equivalent represent the same bitemporal conceptual relation.

In the proof of the following theorem, the notion of snapshot subset is utilized.

Definition: A temporal relation instance, r, is a *snapshot subset* of a temporal relation instance, s, $r \overset{\mathrm{S}}{\subseteq} s$, if for all times t_1 not exceeding UC and all times t_2,

$$\tau^{\mathrm{V}}_{t_2}(\rho^{\mathrm{B}}_{t_1}(r)) \subseteq \tau^{\mathrm{V}}_{t_2}(\rho^{\mathrm{B}}_{t_1}(s)).$$

More generally, a temporal query expression Q_1 is a *snapshot subset* of a temporal query expression Q_2, $Q_1 \overset{\mathrm{S}}{\subseteq} Q_2$, if all instantiations of Q_1 are snapshot subsets of the corresponding instantiations of Q_2. □

THEOREM **3** Snapshot equivalent temporal relations represent the same conceptual temporal relation.

1. If `conceptual_to_snap`$(r', cover_1) = r_1$ and `conceptual_to_snap`$(r', cover_2) = r_2$, then $r_1 \overset{\mathrm{S}}{\equiv} r_2$.
2. If $s_1 \overset{\mathrm{S}}{\equiv} s_2$ then `snap_to_conceptual`$(s_1) =$ `snap_to_conceptual`(s_2).

PROOF: We prove the two implications in turn. To prove that r_1 and r_2 are snapshot equivalent, we prove that r_1 is a snapshot subset of r_2, and conversely. We need to show that for all times t_1 and t_2 that if $x \in \tau^{\mathrm{V}}_{t_2}(\rho^{\mathrm{B}}_{t_1}(r_1))$ then also $x \in \tau^{\mathrm{V}}_{t_2}\rho^{\mathrm{B}}_{t_1}(r_2))$. Let tuple x be in $\tau^{\mathrm{V}}_{t_2}(\rho^{\mathrm{B}}_{t_1}(r_1))$. By the definitions of transaction and valid timeslice, a set of tuples x_i exist in r_1 with $x_i[A] = x$ and $t_1 \in x_i[\mathrm{T}]$ and $t_2 \in x_i[\mathrm{V}]$. By the premise and the definition of `conceptual_to_snap`, a single tuple x' exists in r' with $x'[A] = x_i[A]$ and so that $x'[\mathrm{T}]$ contains exactly the bitemporal chronons covered by the x_i. Further, the bitemporal chronon (t_2, t_1) must be in $x'[\mathrm{T}]$. Independently of a particular covering function, an application of `conceptual_to_snap` to x' will then result in a set of tuples y_j, each with $y_j[A] = x'[A]$. For at least one of the y_j, it must be true that $t_1 \in y_j[\mathrm{T}]$ and $t_2 \in y_j[\mathrm{V}]$ (the first requirement). Therefore, tuple $y = x'[A]$ must be in $\tau^{\mathrm{V}}_{t_2}(\rho^{\mathrm{B}}_{t_1}(r_2))$. Since $y = x$, r_1 is a snapshot subset of r_2. Due to symmetry, proving the reverse is similar.

To prove the second implication, pick an arbitrary tuple x in some snapshot of s_1 and let (t_i, t_j) be the set of pairs of valid and transaction times so that x is in $\tau^{\mathrm{V}}_{t_i}(\rho^{\mathrm{B}}_{t_j}(s_1))$. (This is simply the bitemporal element in s_1 corresponding to the fact x.) By the premise and the definition of snapshot equivalence, the set of pairs (t'_i, t'_j) such that x is in $\tau^{\mathrm{V}}_{t'_i}(\rho^{\mathrm{B}}_{t'_j}(s_2))$ must be identical to the set (t_i, t_j). In general, these sets of pairs are covered by different sets of rectangles in s_1 and s_2. However, the function `snap_to_conceptual` simply accumulates the covered pairs (corresponding to bitemporal chronons) in sets, rendering the particular covering by rectangles immaterial. □

This theorem has important consequences. For each representation and for any covering function, snapshot equivalence partitions the relation instances into equivalence classes where each instance in an equivalence class maps to the same bitemporal conceptual relation instance. The semantics of the representational instance is thus identical to that of the corresponding conceptual instance. This correspondence provides a way of converting instances between representations: the conversion proceeds through the snapshot equivalent conceptual instance.

Finally, the correspondence provides a way of demonstrating that two instances in different representations are semantically equivalent, again by examining the conceptual instance(s) to which they map. For example, it may be shown that the representation instances given in Sections 6.1 through 6.5 are semantically equivalent to the bitemporal conceptual relation given in Section 5.2, and are thus semantically equivalent to each other.

8 Algebras and Equivalence

We now examine operational aspects of the data models just introduced. A major goal is to demonstrate the existence of the operational counterpart of the structural equivalence established in the previous section.

In Section 7.1, we defined two algebraic operators, the transaction- and valid-timeslice operators, on conceptual relations. We then defined the corresponding operations on the chosen tuple-timestamped representation (see Section 6.1). Each of the remaining four representations could have been used instead. We continue by defining the remaining conceptual algebraic operators. We prove that the operators preserve snapshot equivalence and are natural generalizations of their snapshot counterparts. Finally, we examine two transformations that manipulate coverings in representations of bitemporal-relation instances.

8.1 An Algebra for Bitemporal Conceptual Relations

Define a relation schema $R = (A_1, \ldots, A_n |\ \mathrm{T})$, and let r be an instance of this schema. Let t_2 denote an arbitrary time value and let t_1 denote a time not exceeding the current time.

Let D be an arbitrary set of $|D|$ non-timestamp attributes of relation schema R. The projection on D of r, $\pi^{\mathrm{B}}_D(r)$, is defined as follows.

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

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

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

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

To define the union operator, $\cup^{\mathrm{B}}$, let both r_1 and r_2 be instances of R.

$$\begin{aligned}r_1 \cup^{\mathrm{B}} r_2 = \{z^{(n+1)} \mid\ & (\exists x \in r_1\ \exists y \in r_2\ (z[A] = x[A] = y[A] \wedge z[\mathrm{T}] = x[\mathrm{T}] \cup y[\mathrm{T}])) \vee \\ & (\exists x \in r_1\ (z[A] = x[A] \wedge (\neg \exists y \in r_2 (y[A] = x[A])) \wedge z[\mathrm{T}] = x[\mathrm{T}])) \vee \\ & (\exists y \in r_2\ (z[A] = y[A] \wedge (\neg \exists x \in r_1 (x[A] = y[A])) \wedge z[\mathrm{T}] = y[\mathrm{T}]))\}\end{aligned}$$

The first clause handles value-equivalent tuples found in both r_1 and r_2; the second clause handles those found only in r_1; and the third handles those found only in r_2.

With r_1 and r_2 defined as above, relational difference is defined as follows.

$$\begin{aligned}r_1 -^{\mathrm{B}} r_2 = \{z^{(n+1)} \mid\ & \exists x \in r_1\ ((z[A] = x[A]) \wedge \\ & ((\exists y \in r_2\ (z[A] = y[A] \wedge z[\mathrm{T}] = x[\mathrm{T}] - y[\mathrm{T}])) \vee \\ & (\neg \exists y \in r_2\ (z[A] = y[A]) \wedge z[\mathrm{T}] = x[\mathrm{T}])))\}\end{aligned}$$

The last two lines compute the bitemporal element, depending on whether a value-equivalent tuple may be found in r_2.

In the bitemporal natural join, two tuples join if they match on the join attributes and have overlapping bitemporal-element timestamps. Define r and s to be instances of R and S, respectively, and let R and S be bitemporal relation schemas given as follows.

$$\begin{aligned}R &= (A_1, \ldots, A_n, B_1, \ldots, B_l |\ \mathrm{T}) \\ S &= (A_1, \ldots, A_n, C_1, \ldots, C_m |\ \mathrm{T})\end{aligned}$$

The bitemporal natural join of r and s, $r \bowtie^{\mathrm{B}} s$, is defined below. As can be seen, the timestamp of a tuple in the result is the (bitemporal) intersection of the timestamps of the two tuples that produced it.

$$\begin{aligned}r \bowtie^{\mathrm{B}} s = \{z^{(n+l+m+1)} \mid \exists x \in r\ \exists y \in s\ (& x[A] = y[A] \wedge x[\mathrm{T}] \cap y[\mathrm{T}] \neq \emptyset \wedge \\ & z[A] = x[A] \wedge z[B] = x[B] \wedge z[C] = y[C] \wedge \\ & z[\mathrm{T}] = x[\mathrm{T}] \cap y[\mathrm{T}])\}\end{aligned}$$

Example: To exemplify the join, consider the following relation instance, mgrDep.

Dept	Mgr	T
Ship	Jean	$\{(10,15),\ldots,(10,30),\ldots,(UC,15),\ldots,(UC,30)\}$
Load	Jean	$\{(15,5),\ldots,(15,15),\ldots,(UC,5),\ldots,(UC,15)\}$

Next, assign the name empDep to the relation instance in Figure 15. Then empDep $\bowtie^{\mathrm{B}}$ mgrDep, with the explicit join attribute Dept, shows who managed whom and is given by the following relation.

Emp	Dept	Mgr	T
Jake	Ship	Jean	$\{(10,15),\ldots,(10,20),\ldots,(15,15),\ldots,(15,20)\}$
Jake	Load	Jean	$\{(UC,10),\ldots,(UC,15)\}$
Kate	Ship	Jean	$\{(UC,25),\ldots,(UC,30)\}$

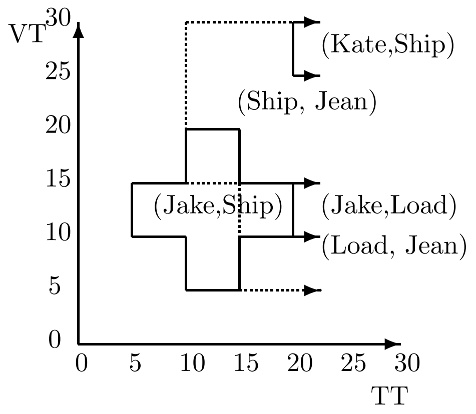

Figure 17: Graph of empDep $\bowtie^{\mathrm{B}}$ mgrDep

Using our graphical representation of bitemporal relations, the bitemporal natural join can be visualized as the overlap of rectangles enclosing regions with matching explicit join attributes. This is easily seen by superimposing the mgrDep relation on top of the empDep relation, as shown in Figure 17. □

We have only defined operators for bitemporal relations. The similar operators for valid-time and transaction-time relations are special cases. The valid and transaction time natural joins are denoted $\bowtie^{\mathrm{V}}$ and $\bowtie^{\mathrm{B}}$, respectively; the conventional snapshot natural join is denoted $\bowtie^{\mathrm{S}}$. The same naming convention is used for the remaining operators.

8.2 An Algebra for Snodgrass' Tuple Timestamped Representation Scheme

For each of the algebraic operators defined in the previous section, we now define counterparts for the first of the five representation schemes. Throughout this section, R and S denote tuple timestamped bitemporal relation schemas, and r and s are instances of these schemas. Initially, R is assumed to have the attributes $A_1, \ldots, A_n, \mathrm{T_s}, \mathrm{T_e}, \mathrm{V_s}$, and $\mathrm{V_e}$.

We define in turn projection, selection, union, difference, and natural join. The timeslice operators were defined in Section 7.2.

To define projection, let D be an arbitrary set of $|D|$ attributes among $A_1, \ldots, A_n$. The projection on D of r, $\pi^{\mathrm{B}}_D(r)$, is defined as follows.

$$\pi^{\mathrm{B}}_D(r) = \{z^{(|D|+4)} \mid \exists x \in r\ (z[D] = x[D] \wedge z[\mathrm{T}] = x[\mathrm{T}] \wedge z[\mathrm{V}] = x[\mathrm{V}])\}$$

Next, 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.

$$\sigma_P^{\mathrm{B}}(r) = \{z^{(n+4)} \mid z \in r \wedge P(z[A]))\}$$

To define the union operator, $\cup^{\mathrm{B}}$, let both r_1 and r_2 be instances of schema R.

$$r_1 \cup^{\mathrm{B}} r_2 = \{z^{(n+4)} \mid \exists x \in r_1\ \exists y \in r_2\ (z = x \vee z = y)\}$$

With r_1 and r_2 defined as above, relational difference is defined using several functions introduced in Section 6.1.

$$\begin{aligned} r_1 -^{\mathrm{B}} r_2 = \{z^{(n+4)} \mid \exists x \in r_1\ (z[A] = x[A] \wedge \\ &\exists t \in \mathit{cover}(\mathit{bi\_chr}(x[\mathrm{T}], x[\mathrm{V}]) - \\ &\qquad \{\mathit{bi\_chr}(y[\mathrm{T}], y[\mathrm{V}]) \mid y \in r_2 \wedge y[A] = x[A]\}) \wedge \\ &z[\mathrm{T}_s] = \mathit{min\_1}(t) \wedge z[\mathrm{T}_e] = \mathit{max\_1}(t) \wedge \\ &z[\mathrm{V}_s] = \mathit{min\_2}(t) \wedge z[\mathrm{V}_e] = \mathit{max\_2}(t))\} \end{aligned}$$

The new timestamp is conveniently determined by set difference on bitemporal elements.

To define the bitemporal natural join, we need two bitemporal relation schemas R and S with overlapping attributes.

$$\begin{aligned} R &= (A_1, \ldots, A_n, B_1, \ldots, B_l, \mathrm{T_s}, \mathrm{T_e}, \mathrm{V_s}, \mathrm{V_e}) \\ S &= (A_1, \ldots, A_n, C_1, \ldots, C_m, , \mathrm{T_s}, \mathrm{T_e}, \mathrm{V_s}, \mathrm{V_e}) \end{aligned}$$

In the bitemporal natural join of r and s, $r \bowtie^{\mathrm{B}} s$, two tuples join if they match on the join attributes and overlap in both valid time and transaction time.

$$\begin{aligned} r \bowtie^{\mathrm{B}} s = \{z^{(n+l+m+4)} \mid \exists x \in r\ \exists y \in s\ (z[A] = x[A] = y[A] \wedge x[\mathrm{T}] \cap y[\mathrm{T}] \neq \emptyset \wedge x[\mathrm{V}] \cap y[\mathrm{V}] \neq \emptyset \wedge \\ &z[B] = x[B] \wedge z[C] = y[C] \wedge \\ &z[\mathrm{T}] = x[\mathrm{T}] \cap y[\mathrm{T}] \wedge z[\mathrm{V}] = x[\mathrm{V}] \cap y[\mathrm{V}])\} \end{aligned}$$

As for the previous model, corresponding operators for valid-time and transaction-time relations may be defined as special cases of the operators already defined.

8.3 Equivalence Properties

We have seen that a bitemporal conceptual relation is represented by a class of snapshot equivalent relations in the representation scheme. We now define the notion of an operator preserving snapshot equivalence.

Definition: An operator α *preserves snapshot equivalence* if, for all parameters X and snapshot relation instances r and r' representing bitemporal relations,

$$r \overset{\mathrm{S}}{\equiv} r' \ \Rightarrow\ \alpha_X(r) \overset{\mathrm{S}}{\equiv} \alpha_X(r').$$

This definition may be trivially extended to operators that accept two or more argument relation instances. □

In the snapshot relational algebra, an operator, e.g., natural join, must return identical results every time it is applied to the same pair of arguments. The same holds for the BCDM. However, in the representational models, for which several relation instances may be snapshot equivalent, only preservation of snapshot equivalence is required. Thus, we add flexibility in implementing the bitemporal operators by accepting that they return different, but snapshot equivalent, results when applied to identical arguments at different times.

We proceed by showing that the operators preserve snapshot equivalence. That is, given snapshot equivalent operands each operator produces snapshot equivalent results. This ensures that the result of an algebraic operation is correct, irrespective of covering. Again, the proof is given only for one representation, though the theorem holds for all five representations considered.

THEOREM **4** The algebraic operators preserve snapshot equivalence. Specifically, let $r \overset{\text{S}}{\equiv} r'$ and $s \overset{\text{S}}{\equiv} s'$. Then

$$\begin{aligned}
r \bowtie^{\text{V}} s &\overset{\text{S}}{\equiv} r' \bowtie^{\text{V}} s' \\
r \bowtie^{\text{B}} s &\overset{\text{S}}{\equiv} r' \bowtie^{\text{B}} s' \\
\sigma^{\text{B}}_{P}(r) &\overset{\text{S}}{\equiv} \sigma^{\text{B}}_{P}(r') \\
\pi^{\text{B}}_{D}(r) &\overset{\text{S}}{\equiv} \pi^{\text{B}}_{D}(r') \\
r \cup^{\text{B}} s &\overset{\text{S}}{\equiv} r' \cup^{\text{B}} s' \\
r -^{\text{B}} s &\overset{\text{S}}{\equiv} r' -^{\text{B}} s'.
\end{aligned}$$

PROOF: As before, we proceed by demonstrating snapshot subsets. To prove the first equivalence, let tuple x be in the lhs. By the definition of $\bowtie^{\text{V}}$ there exists a set of tuples $x_i \in r$ with $x_i[AB] = x[AB]$ and so that $\cup_i x_i[\text{V}] \supseteq x[\text{V}]$. Similarly, there exists a set of tuples $x_j \in s$ with $x_j[AC] = x[AC]$ and so that $\cup_j x_j[\text{V}] \supseteq x[\text{V}]$. Next, by the definition of $\overset{\text{S}}{\equiv}$, for each $x_i \in r$ the exists a set of tuples $x^i_k \in r'$ with $x^i_k[AB] = x_i[AB]$ and so that $\cup_k x^i_k[\text{V}] \supseteq x_i[\text{V}]$. The set x^i_k covers x_i. For each j a similar set x^j_l exists that covers x_j. Applying $\bowtie^{\text{V}}$ to the sets of tuples $x^i_k \in r'$ and $x^j_l \in s'$ yields a set of tuples x_m with $x_m[ABC] = x[ABC]$ and so that $\cup_m x_m[\text{V}] \supseteq x[\text{V}]$. This proves that any tuple in a snapshot made from the lhs will also be present in the same snapshot made from the rhs. By symmetry, the reverse is also true, and the equivalence follows.

The proofs of the other equivalences are similar. $\square$

The next step is to combine the transformation functions with the representation level operators to create corresponding conceptual-level operators. Given a representation level operator, α, its corresponding conceptual-level operator, α^c, is defined as follows.

$$\alpha^c_X(r') = \texttt{snap\_to\_conceptual}(\alpha_X(\texttt{conceptual\_to\_snap}(r')))$$

Theorems 3 and 4 in combination make this meaningful and ensure that the conceptual-level operators behave like the snapshot relational algebra operators—with identical arguments, they always return identical results. This is required because, like snapshot relations, bitemporal conceptual relations are unique, i.e., two conceptual relations have the same information content if and only if they are identical.

Now, we have two sets of operators defined on the bitemporal conceptual relations, namely the directly defined operators in Section 8.1 and the induced operators. In fact, we have constructed the two sets of operators to be identical. Put differently, the operators in Section 8.1 are the explicitly stated conceptual-level operators, induced from the representation level operators (Section 8.2) and the transformation algorithms in Section 6.1. This is formalized in the following theorem.

THEOREM **5** The induced algebraic operators preserve snapshot equivalence.

PROOF: Let α^{c}_X be an induced conceptual operator, and suppose that conceptual relations r and s are snapshot equivalent. By Theorem 1, $r = s$, and therefore, $\texttt{conceptual\_to\_snap}(r) \overset{\mathrm{S}}{\equiv} \texttt{conceptual\_to\_snap}(s)$. By Theorem 4, $\alpha_X(\texttt{conceptual\_to\_snap}(r)) \overset{\mathrm{S}}{\equiv} \alpha_X(\texttt{conceptual\_to\_snap}(s))$. Finally, by Theorem 3, $\texttt{snap\_to\_conceptual}(\alpha_X(\texttt{conceptual\_to\_snap}(r))) \overset{\mathrm{S}}{\equiv} \texttt{snap\_to\_conceptual}(\alpha_X(\texttt{conceptual\_to\_snap}(s)))$. $\square$

Next we show how the operators in the various data models, snapshot, transaction-time, valid-time, and bitemporal, are related. Specifically, we show that the semantics of an operator in a more complex data model reduces to the semantics of the operator in a simpler data model. Reducibility guarantees that the semantics of simpler operators are preserved in their more complex counterparts.

For example, the semantics of the transaction-time natural join reduces to the semantics of the snapshot natural join in that the result of first joining two transaction-time relations and then transforming the result to a snapshot relation yields a result equivalent to that obtained by first transforming the arguments to snapshot relations and then joining the snapshot relations. This is shown in Figure 18 and stated formally in the first equivalence of the following theorem.

$$\begin{tikzcd}
\underline{\text{Transaction-time relations}} & \underline{\text{Snapshot relations}} \\
r,\ r' \arrow[r, "\rho^{\mathrm{T}}_t"] \arrow[d, "\bowtie^{\mathrm{T}}"'] & \rho^{\mathrm{T}}_t(r),\ \rho^{\mathrm{T}}_t(r') \arrow[d, "\bowtie^{\mathrm{S}}"] \\
r \bowtie^{\mathrm{T}} r' \arrow[r, "\rho^{\mathrm{T}}_t"] & \rho^{\mathrm{T}}_t(r \bowtie^{\mathrm{T}} r') \overset{\mathrm{S}}{\equiv} \rho^{\mathrm{T}}_t(r) \bowtie^{\mathrm{S}} \rho^{\mathrm{T}}_t(r')
\end{tikzcd}$$

Figure 18: Reducibility of Transaction-time Natural Join to Snapshot Outer Natural Join.

THEOREM **6** Let t denote an arbitrary time that, when used with a rollback operator, does not exceed the current time. In each equivalence, let r and s be relation instances of the proper types for the given operators. Then the following hold.

$$\begin{aligned}
\rho^{\mathrm{T}}_t(r \bowtie^{\mathrm{T}} s) &\overset{\mathrm{S}}{\equiv} \rho^{\mathrm{T}}_t(r) \bowtie^{\mathrm{S}} \rho^{\mathrm{T}}_t(s) \\
\tau^{\mathrm{V}}_t(r \bowtie^{\mathrm{V}} s) &\overset{\mathrm{S}}{\equiv} \tau^{\mathrm{V}}_t(r) \bowtie^{\mathrm{S}} \tau^{\mathrm{V}}_t(s) \\
\tau^{\mathrm{B}}_t(r \bowtie^{\mathrm{B}} s) &\overset{\mathrm{S}}{\equiv} \tau^{\mathrm{B}}_t(r) \bowtie^{\mathrm{T}} \tau^{\mathrm{B}}_t(s) \\
\rho^{\mathrm{B}}_t(r \bowtie^{\mathrm{B}} s) &\overset{\mathrm{S}}{\equiv} \rho^{\mathrm{B}}_t(r) \bowtie^{\mathrm{V}} \rho^{\mathrm{B}}_t(s)
\end{aligned}$$

PROOF: An equivalence is shown by proving its two inclusions separately. The non-timestamp attributes of r and s are AB and AC, respectively, where A, B, and C are sets of attributes and A denotes the join attribute(s).

We prove the fourth equivalence. The proofs of the remaining equivalences are similar and are omitted. Let $x'' \in$ lhs. Then there is a tuple $x' \in r \bowtie^{\mathrm{B}} s$ such that $x'[ABC] = x''$ and $t \in x'[\mathrm{T}]$. By the definition of $\bowtie^{\mathrm{B}}$, there exists tuples $x_1 \in r$ and $x_2 \in s$ such that $x_1[A] = x_2[A] = x'[A]$, $x_1[B] = x'[B]$, $x_2[C] = x'[C]$, $x'[T] = x_1[T] \cap x_2[T]$, and $x'[V] = x_1[V] \cap x_2[V]$. By the definition of ρ^{B}_t, there exists a tuple $x'_1 \in \rho^{\mathrm{B}}_t(r)$ such that $x'_1 = x'[AB]$ and $x'_1[V] = x'[V]$ and a tuple $x'_2 \in \rho^{\mathrm{B}}_t(s)$

such that $x'_2 = x'[AC]$ and $x'_2[V] = x'[V]$. Then there exists $x''_{12} \in$ rhs such that $x''_{12}[AB] = x'_1$, $x''_{12}[C] = x'_2[C]$, and $x''_{12}[V] = x'_1[V] \cap x'_2[V]$. By construction $x''_{12} \overset{\mathrm{S}}{\equiv} x''$ (in fact, $x''_{12} = x''$).

Now assume $x'' \in$ rhs. Then there exists tuples x'_1 and x'_2 in $\rho^{\mathrm{B}}_t(r)$ and $\rho^{\mathrm{B}}_t(s)$, respectively, such that $x'_1 = x''[AB]$ and $x'_2 = x''[AC]$ and $x''[V] = x'_1[V] \cap x'_2[V]$. This implies the existence of tuples $x_1 \in r$ and $x_2 ins$ and with $x_1[AB] = x'_1[AB]$, $x_1[V] = x_1[V]$, $t \in x_1[\mathrm{T}]$, $x_2[AC] = x'_2[AC]$, $x_2[V] = x'_2[V]$, and $t \in x_2[\mathrm{T}]$. There must exist a tuple $x' \in r \bowtie^{\mathrm{B}} s$ with $x'[AB] = x_1[AB]$, $x'[C] = x_2[C]$, $x'[V] = x_1[V] \cap x_2[V]$, and $t \in x'[\mathrm{T}]$. Consequently, there exists a tuple $x''_{12} \in$ lhs such that $x''_{12} = x'[ABC]$ and $x''_{12}[V] = x'[V]$. By construction, $x''_{12} \overset{\mathrm{S}}{\equiv} x''$. □

8.4 Covering Transformations

When a bitemporal conceptual relation is mapped to a representation scheme, a covering function is employed to represent bitemporal elements by sets of rectangles. The mappings were used in Sections 6.1 to 6.5, and different types of covering functions were discussed in Section 6.1. We now define two transformations that can change the covering in a representation without affecting the results of queries, as the transformations preserve snapshot equivalence. Both are generalizations of simpler transformations used in valid time data models.

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]. Coalescing may reduce the number of tuples necessary for representing a bitemporal relation, and, as such, is a space optimization. We formally define coalescing and show that it preserves snapshot equivalence.

Definition: *Coalescing.* Let $x = (a_1, \ldots, a_n, t_1, t_2, v_1, v_2)$ and $x' = (a_1, \ldots, a_n, t_3, t_4, v_3, v_4)$ be two distinct tuples belonging to the same bitemporal relation instance.

First, if $x[\mathrm{T}] = x'[\mathrm{T}]$ and $x[\mathrm{V}] \cup x'[\mathrm{V}] = [\min(v_1, v_3), \max(v_2, v_4)]$, the two tuples may be *coalesced* into the single tuple $y = (a_1, \ldots, a_n, t_1, t_2, \min(v_1, v_3), \max(v_2, v_4))$. Second, if $x[\mathrm{V}] = x'[\mathrm{V}]$ and $x[\mathrm{T}] \cup x'[\mathrm{T}] = [\min(t_1, t_3), \max(t_2, t_4)]$, the two tuples may be *coalesced* into the single tuple $y' = (a_1, \ldots, a_n, \min(t_1, t_3), \max(t_2, t_4), v_1, v_2)$.

A bitemporal relation instance is *coalesced* if no pair of tuples may be coalesced. □

The proof of the next theorem utilizes a subtle requirement on null values in bitemporal relations. Specifically, we require that null information not conflict with non-null information. If one tuple states that the value of an attribute is null then another, temporally concurrent tuple that contains non-null information for that attribute must not exist. More formally, we define this property as follows.

Definition: *Consistency of null information.* Let two tuples x and x', both belonging to a relation instance r, be given by $x = (a_1, \ldots, a_n, t)$ and $x' = (a'_1, \ldots, a'_n, t')$ where $\exists k_1 \ldots k_m\, (a_{k_1} = \bot \neq a'_{k_1} \wedge \ldots \wedge a_{k_m} = \bot \neq a'_{k_m})$ and $\forall i \notin \{k_1, \ldots, k_m\}(a_i = a'_i)$. The last elements, t and t', of the two tuples denote bitemporal elements. If, for all such tuple pairs in r, it is the case that $t \cap t' = \emptyset$ then the null information in r is consistent. □

Theorem **7** Coalescing preserves snapshot equivalence.
Proof: Let r be a relation instance containing x and x' as given in the definition of coalescing. In the first of the two cases, let relation s be identical to r, but with x and x' replaced by the tuple y as given in the definition. We must prove r and s snapshot equivalent. The tuples x and x' result in exactly the tuple $(a_1, \ldots, a_n)$ being present in all snapshots of r with a transaction time in $[t_1, t_2]$ and a valid time in $[\min(v_1, v_3), \max(v_2, v_4)]$. Similarly, the tuple y results in

$(a_1, \ldots, a_n)$ being part of all snapshots of s with a transaction time in $[t_1, t_2]$ and a valid time in $[\min(v_1, v_3), \max(v_2, v_4)]$. The requirement that null information be genuine ensures this even in the case when there are nulls among the a_i. The proof for the second of the two cases is similar. $\square$

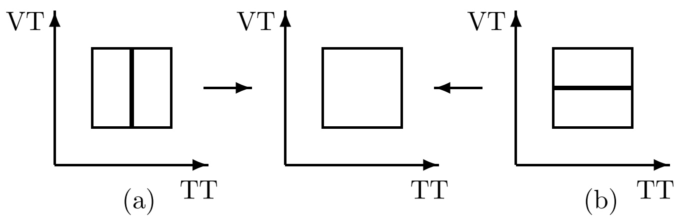

Figure 19: Coalescing

Coalescing of overlapping, value-equivalent tuples is illustrated in Figure 19. 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.

We now formalize the notion that a relation may have repeated information among tuples.

DEFINITION: A bitemporal relation instance r has *repetition of information* if it contains two distinct tuples $x = (a_1, \ldots, a_n, t_1, t_2, v_1, v_2)$ and $x' = (a_1, \ldots, a_n, t_3, t_4, v_3, v_4)$ such that $x[\mathrm{T}] \cap x'[\mathrm{T}] \neq \emptyset \wedge x[\mathrm{V}] \cap x'[\mathrm{V}] \neq \emptyset$. A relation with no such tuples has no repetition of information. $\square$

While coalescing may both reduce the number of rectangles and reduce repetition of information, its applicability is restricted. The next equivalence preserving transformation may be employed to completely eliminate temporally redundant information, possibly at the expense of adding extra tuples. We first define the transformation and then describe its properties.

DEFINITION: *Elimination of repetition.* With x and x' as in the definition above, the information in tuple y, defined below, is contained in both x and x'.

$$y = (a_1, \ldots, a_n, \max(t_1, t_3), \min(t_2, t_4), \max(v_1, v_3), \min(v_2, v_4))$$

The repetition incurred by x and x' may be eliminated by replacing tuples x and x' by the set of tuples, s, defined below.

1 $s = \{z^{(n+4)} \mid z[A] = x[A] \wedge ((z[\mathrm{T}] \in \mathit{cover}_t^{max}(x[\mathrm{T}] - x'[\mathrm{T}]) \wedge z[\mathrm{V}] = x[\mathrm{V}]) \vee$
2 $(z[\mathrm{T}] \in \mathit{cover}_t^{max}(x'[\mathrm{T}] - x[\mathrm{T}]) \wedge z[\mathrm{V}] = x'[\mathrm{V}]) \vee$
3 $(z[\mathrm{T}] = x[\mathrm{T}] \cap x'[\mathrm{T}] \wedge z[\mathrm{V}] = x[\mathrm{V}] \cup x'[\mathrm{V}]))\}$

The function cover_t^{max} transforms an argument set of transaction-time chronons into a set of maximal intervals of consecutive chronons. $\square$

THEOREM **8** The elimination of repetition transformation has the following properties.

1. It eliminates repetition among two argument tuples.

2. The result, s, has at most three tuples.

3. It is equivalence preserving.

4. Repeated application produces a relation instance with no repetition of information.

PROOF: There is no repetition of information between the resulting tuples as they do not overlap in transaction time.

Let x and x' be given as in the definition of elimination of repetition and define $T_x = cover_t^{max}(x[\mathrm{T}] - x'[\mathrm{T}])$ and $T'_x = cover_t^{max}(x'[\mathrm{T}] - x[\mathrm{T}])$. Tuples x and x' are replaced by at most three tuples. Line 3 results in one tuple. Lines 1 and 2 collectively result in two tuples, for the following reasons. The set T_x has two elements when $x'[\mathrm{T}]$ contains no endpoints of x[T]. In this case T'_x is empty. The sets T_x and T'_x have both one element when $x'[\mathrm{T}]$ contains exactly one of the endpoints of $x[\mathrm{T}]$. Lastly, T_x is empty when $x'[\mathrm{T}]$ contains both endpoints of x[T]. In this case T'_x has two elements.

Being similar to that for coalescing, the proof of equivalence preservation is omitted.

The process of eliminating repetition is terminating because the new tuples that result from one transformation step cover strictly smaller intervals in the transaction-time dimension. In addition, two tuples that cover only a single transaction time and have repeated information may be coalesced into a single tuple that would not be further partitioned. □

The transformation partitions the regions covered by the argument rectangles on transaction time. The symmetric transformation, which partitions on valid time, may also be included. These transformations are illustrated in parts (a) and (b), respectively, of Figure 20.

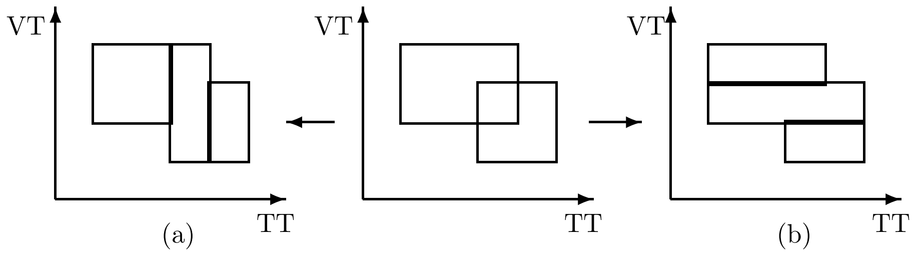

Figure 20: Eliminating Representational Repetition of Information

The elimination of repetition of information may 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 updates are simplified.

9 Summary

This paper has compared the many existing temporal data models and has proposed a new one, the bitemporal conceptual data model (BCDM), as the basis for the Temporal Structured Query Language (TSQL).

The data model proposed in this paper is a conceptual one, meant specifically for the purpose of capturing the semantics of time-varying data. It is based on the observation that different data models are appropriate for different tasks, such as data presentation, storage representation, and modeling the time semantics of data. These separate tasks pose very different requirements for a data model, and they should be considered in isolation, utilizing different data models. It is our contention that the large number of data models existing today is a consequence of trying to do each of the tasks using the same data model. As a result of trying to accommodate presentation and representation, the central task of modeling the time semantics of data has been obscured by data models. Finally, we feel that the BCDM is the appropriate location for database design and logical-level query optimization.

The BCDM is a simple data model, built on the experience gained from previous proposals. A BCDM relation consists simply of a set of ordinary tuples. For each tuple, an implicit attribute value specifies when the (composite) fact represented by the tuple is true in the modeled reality and is current in the stored relation. The implicit attribute has temporal elements (i.e., sets of chronons) as its values. Temporal elements have been chosen because they allow relations to contain one complete fact per tuple—a relation is a *set* of tuples, and value-equivalence and coalescing are easily ensured. Also temporal elements are generalizations of events and intervals and are closed under union, difference, and complementation.

An important property of the conceptual model—shared with the conventional relational model, but not held by the representational models—is that relation instances are semantically unique; distinct instances model different realities and thus have distinct semantics.

We showed that the BCDM is a unifying model in that conceptual instances could be mapped into instances of five existing bitemporal *representational data models*: a first normal form (1NF) tuple-timestamped data model, a data model based on 1NF timestamped change requests recorded in backlog relations, a non-1NF data model in which attribute values were stamped with rectangles in transaction-time/valid-time space, a non-1NF model where valid-time states are indexed by transaction time, and a 1NF model where each tuple is accorded five timestamp values. We also showed how extensions to the conventional relational algebraic operators could be defined in a representational data model and then be meaningfully mapped to analogous operators in the conceptual data model.

That two temporal relations have the same information content was formalized using the notion of snapshot equivalence. Briefly, two relations are snapshot equivalent if all derived snapshots are mutually identical. As all data models provide means of producing snapshots, snapshot equivalence provides a natural way to compare temporal relations in diverse data models. The semantic uniqueness of the relations in the BCDM implies that two BCDM relations are snapshot equivalent if and only if they are identical.

This paper has addressed one aspect of the design of the Temporal Structured Query Language. The closely related tasks of designing an appropriate query language and algebra are under way.

10 Acknowledgements

Portions of this research were conducted while the first author visited the University of Arizona. Support was provided in part by the Danish Natural Science Research Council through grants 11–9675–1 SE, 11–1089–1 SE, 11–9675–1 SE, and 11–0061; the National Science Foundation through grants IRI–8902707 and IRI–9302244; the IBM Corporation through Contract #1124; and the AT&T Foundation. James Clifford was helpful in understanding structural aspects of models of time. Curtis Dyreson and Nick Kline provided helpful comments on a previous draft.

References

[AH85] J. F. Allen and P. J. Hayes. A Common-Sense Theory of Time. In *ijcai*, pages 528–531, Los Angeles, CA, August 1985.

[And82] T. L. Anderson. Modeling Time at the Conceptual Level. In P. Scheuermann, editor, *Proceedings of the International Conference on Databases: Improving Usability and Responsiveness*, pages 273–297, Jerusalem, Israel, June 1982. Academic Press.

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

[BADW82] A. Bolour, T. L. Anderson, L. J. Dekeyser, and H. K. T. Wong. The Role of Time in Information Processing: A Survey. *SigArt Newsletter*, 80:28–48, April 1982.

[BG89a] G. Bhargava and S. Gadia. Achieving Zero Information Loss in a Classical Database Environment. In *Proceedings of the Conference on Very Large Databases*, pages 217–224, Amsterdam, August 1989.

[BG89b] G. Bhargava and S. K. Gadia. A 2-Dimensional Temporal Relational Database Model for Querying Errors and Updates, and for Achieving Zero Information-Loss. Technical Report TR#89-24, Department of Computer Science, Iowa State University, Ames, Iowa, December 1989.

[BG90] G. Bhargava and S. K. Gadia. The Concept of Error in a Database: An Application of Temporal Databases. In N. Prakash, editor, *Proceedings of 1990 COMAD International Conference on Management of Data*, pages 106–121, New Delhi, December 1990. Tata McGraw-Hill.

[BG91] G. Bhargava and S. K. Gadia. Relational Database Systems with Zero Information-Loss. IEEE Transactions on Knowledge and Data Engineering (to appear), 1991.

[Blu81] R. L. Blum. Displaying Clinical Data from a Time-Oriented Database. *Comput. Biol. Med.*, 11(4):197–210, 1981.

[Bro56] F. P. Brooks. *The Analytic Design of Automatic Data Processing Systems.* PhD thesis, Harvard University, Cambridge, MA, May 1956.

[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.

[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.

[CI89] J. Chomicki and T. Imelinski. Relational Specifications of Infinite Query Answers. In *Proceedings of ACM SIGMOD*, pages 174–183, May 1989.

[Cli82] J. Clifford. A Model for Historical Databases. In *Proceedings of Workshop on Logical Bases for Data Bases*, Toulouse, France, December 1982.

[Cod72] E. F. Codd. *Further Normalization of the Data Base Relational Model*, volume 6 of *Courant Computer Symposia Series.* Prentice Hall, Englewood Cliffs, N.J., 1972.

[CR87] J. Clifford and A. Rao. A Simple, General Structure for Temporal Domains. In *Proceedings of the Conference on Temporal Aspects in Information Systems*, pages 23–30, France, May 1987. AFCET.

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

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

[Dee85] S.M. Deen. Deal: A Relational Language with Deductions, Functions and Recursions. *IEEE Transactions on Knowledge and Data Engineering*, 1, 1985.

[DS92a] C. E. Dyreson and R. T. Snodgrass. Timestamp Semantics and Representation. Technical Report TR 92-16a, Department of Computer Science, University of Arizona, July 1992.

[DS92b] C. E. Dyreson and R. T. Snodgrass. Timestamp Semantics and Representation. TempIS Technical Report 33, Department of Computer Science, University of Arizona, February 1992.

[Gad86a] S. K. Gadia. Toward a Multihomogeneous Model for a Temporal Database. In *Proceedings of the International Conference on Data Engineering*, pages 390–397, Los Angeles, CA, February 1986.

[Gad86b] S. K. Gadia. Weak Temporal Relations. In *ACM PODS*, 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.

[Gad92] S. K. Gadia. A Seamless Generic Extension of SQL for Querying Temporal Data. Technical Report TR-92-02, Computer Science Department, Iowa State University, May 1992.

[GV85] S. K. Gadia and J. H. Vaishnav. A Query Language for a Homogeneous Temporal Database. In *ACM PODS*, pages 51–56, Mar 1985.

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

[Haw88] S. Hawking. *A Brief History of Time.* Bantam Books, New York, 1988.

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

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

[JMR91] C. S. Jensen, L. Mark, and N. Roussopoulos. Incremental Implementation Model for Relational Databases with Transaction Time. *IEEE Transactions 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.

[JMS79] S. Jones, P. Mason, and R. Stamper. Legol 2.0: A Relational Specification Language for Complex Rules. *Information Systems*, 4(4):293–305, November 1979.

[JS92] C.S. Jensen and R. Snodgrass. Temporal Specialization. In F. Golshani, editor, *Proceedings of the International Conference on Data Engineering*, pages 594–603, Tempe, AZ, February 1992.

[JS93] C. S. Jensen and R. Snodgrass. Temporal Specialization and Generalization. *IEEE Transactions on Knowledge and Data Engineering*, 1993.

[JSS92a] C. S. Jensen, R. T. Snodgrass, and M. D. Soo. Extending Normal Forms to Temporal Relations. TR 92-17, Department of Computer Science, University of Arizona, July 1992.

[JSS92b] C. S. Jensen, M. D. Soo, and R. T. Snodgrass. Unification of Temporal Relations. Technical Report 92-15, Department of Computer Science, University of Arizona, July 1992.

[Kim78] K. A. Kimball. The Data System. Master's thesis, University of Pennsylvania, 1978.

[LJ88] N. A. Lorentzos and R. G. Johnson. Requirements Specification for a Temporal Extension to the Relational Model. *Data Engineering*, 11(4):26–33, 1988.

[LK89] N. A. Lorentzos and V. Kollias. The Handling of Depth and Time Intervals in Soil-Information Systems. *Computers and Geosciences*, 15(3):395–401, 1989.

[Lor88] N. A. Lorentzos. *A Formal Extension of the Relational Model for the Representation of Generic Intervals*. PhD thesis, Birkbeck College, University of London, 1988.

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

[McK88] E. McKenzie. *An Algebraic Language for Query and Update of Temporal Databases*. PhD thesis, Department of Computer Science, University of North Carolina, September 1988.

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

[MS91] L. 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.

[NA87] S. B. Navathe and R. Ahmed. TSQL—A Language Interface for History Databases. In *Proceedings of the Conference on Temporal Aspects in Information Systems*, pages 113–128, France, May 1987.

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

[RS87] L. Rowe and M. Stonebraker. The Postgres Papers. Technical Report UCB/ERL M86/85, University of California, Berkeley, CA, June 1987.

[SA85] R. Snodgrass and I. Ahn. A Taxonomy of Time in Databases. In S. Navathe, editor, *Proceedings of ACM SIGMOD*, pages 236–246, Austin, TX, May 1985.

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

[Sad87] R. Sadeghi. *A Database Query Language for Operations on Historical Data*. PhD thesis, Dundee College of Technology, Dundee, Scotland, December 1987.

[Sar90a] N. Sarda. Algebra and Query Language for a Historical Data Model. *The Computer Journal*, 33(1):11–18, February 1990.

[Sar90b] N. Sarda. Extensions to SQL for Historical Databases. *IEEE Transactions on Knowledge and Data Engineering*, 2(2):220–230, June 1990.

[SGM93] R. Snodgrass, S. Gomez, and E. McKenzie. Aggregates in the Temporal Query Language TQUEL. *IEEE Transactions on Knowledge and Data Engineering*, 1993.

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

[Sno92] R. T. Snodgrass. *Temporal Databases*. In volume 639 of *Lecture Notes in Computer Science*, pages 22–64. Springer-Verlag, September 1992.

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

[SS87] A. Segev and A. Shoshani. Logical Modeling of Temporal Data. In U. Dayal and I. Traiger, editors, *Proceedings of ACM SIGMOD*, 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] R. Stam and R. Snodgrass. A Bibliography on Temporal Databases. *Database Engineering*, 7(4):231–239, December 1988.

[SS92] M. D. Soo and R. Snodgrass. Mixed Calendar Query Language Support for Temporal Constants. TempIS Technical Report 29, Department of Computer Science, University of Arizona, Revised May 1992.

[SSD87] R. Sadeghi, W. B. Samson, and S. M. Deen. HQL — A Historical Query Language. Technical report, Dundee College of Technology, Dundee, Scotland, September 1987.

[Sto87] M. Stonebraker. The Design of the Postgres Storage System. In P. Hammersley, editor, *Proceedings of the Conference on Very Large Databases*, pages 289–300, Brighton, England, September 1987.

[TA86a] A. U. Tansel and M. E. Arkun. HQUEL, A Query Language for Historical Relational Databases. In *Proceedings of the Third International Workshop on Statistical and Scientific Databases*, July 1986.

[Tan86] A. U. Tansel. Adding Time Dimension to Relational Model and Extending Relational Algebra. *Information Systems*, 11(4):343–355, 1986.

[TAO89] A.U. Tansel, M.E. Arkun, and G. Ozsoyoglu. Time-By-Example Query Language for Historical Databases. *IEEE Transactions on Software Engineering*, 15(4):464–478, April 1989.

[TC90] A. Tuzhilin and J. Clifford. A Temporal Relational Algebra as a Basis for Temporal Relational Completeness. In *Proceedings of the Conference on Very Large Databases*, Brisbane, Australia, August 1990.

[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.

[Ull82] J.D. Ullman. *Principles of Database Systems*, 2nd edition. Computer Science Press, Potomac, Maryland, 1982.

[VB82] J. F. K. A. Van Benthem. *The Logic of Time: A Model-Theoretic Investigation into the Varieties of Temporal Ontology and Temporal Discourse.* Reidel, Hingham, MA, 1982.

[WFW75] G. Wiederhold, J.F. Fries, and S. Weyl. Structured Organization of Clinical Data Bases. In *Proceedings of NCC*, pages 479–485. AFIPS, 1975.

[Wor90] M. F. Worboys. Reasoning about gis using temporal and dynamic logics. In *Temporal GIS Workshop.* University of Maine, October 1990.

[Yeu86] C. S. Yeung. Query Languages for a Heterogeneous Temporal Database. Master's thesis, EE/CS Department, Texas Tech University, 1986.