Jensen, C. S., R. T. Snodgrass, "Temporally Enhanced Database Design,"
Chapter 7, pp. 163-193, Advances in Object-Oriented Data Modeling,
edited by M. P. Papazoglou, S. Spaccapietra, and Z. Tari

7 Temporally Enhanced Database Design

Christian S. Jensen
Department of Computer Science
Aalborg University
Fredrik Bajers Vej 7E
DK–9220 Aalborg Ø, Denmark
csj@cs.auc.dk

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

The design of appropriate database schemas is critical to the effective use of database technology and the construction of effective information systems that exploit this technology. The temporal aspects of database schemas are often particularly complex and thus difficult and error-prone to design. This chapter focuses on the temporal aspects of database schemas. Its contributions are two-fold. First, a comprehensive set of concepts are presented that capture temporal aspects of schemas. Second, the use of these concepts for database design is explored.

The chapter first generalizes conventional functional dependencies to apply to temporal databases, leading to temporal keys and normal forms. Time patterns identify when attributes change values and when the changes are recorded in the database. Lifespans describe when attributes have values. The temporal support and precision of attributes indicate the temporal aspects that are relevant for the attributes and with what temporal granularity the aspects are to be recorded. And derivation functions describe how the values of an attribute for all times within its lifespan are computed from stored values. The implications of these concepts for database design, of both relational and object-oriented schemas, are explored.

7.1 Introduction

The design of appropriate database schemas is crucial to the effective use of database technology and the construction of information systems that exploit this technology. The process of appropriately capturing the temporal aspects of the modeled reality in the database schema—be it based on, e.g., the relational model or an object-oriented model—is complex and error prone, and the resulting schemas are often overly difficult to understand. With a focus on the temporal aspects of database schemas, this chapter explores the technical foundation for simplifying the conceptual design process.

More than a dozen temporal object-oriented data models have been proposed (38). How to apply database design techniques to schemas in these models is still largely an open problem. The particular data model chosen impacts the manner in which object-oriented schemas are designed. As a natural first approach, this chapter provides a foundation for applying the well-developed relational database design theory to these models. To render the relational design concepts relevant to these various temporal object-oriented data models, this chapter extends these concepts to a temporal relational model, with the application to a particular temporal object-oriented model left as a subsequent task. This application is necessarily highly dependent on the temporal object-oriented model chosen; we exemplify this mapping for a simple object-oriented model, but space limitations prevent more comprehensive coverage. Using the relational model as the basis for this chapter enables the approach to be applicable to the entire spectrum of temporal object-oriented models, while remaining independent of the idiosyncrasies of any particular model.

Specifically, the chapter proposes to separate the design of conceptual database schemas for time-oriented applications into two stages. In the first stage, the underlying temporal aspects are ignored, resulting in the design of simple, single-state (so-called non-temporal) schemas. In the second stage, these initial schemas are annotated with their temporal aspects. These annotations may imply further decomposition of the annotated schemas, leading to the final conceptual schema.

The chapter focuses on the second stage, and begins with a non-temporal database schema. The chapter's contributions are two-fold. First, a comprehensive set of temporal properties that may be used for annotation are defined and illustrated. Second, the use of these properties is explored. Specifically, new guidelines for how the annotations should result in decomposition of the schemas are defined, and their use is explored. The subsequent mapping of annotated, decomposed schemas to implementation platforms is beyond the scope of this chapter.

The chapter is structured as follows. Section 7.2 introduces conceptual temporal relations that may capture the valid time and the transaction time of the stored tuples. These are needed because the non-temporal relation schemas upon annotation may reveal themselves to be temporal. Then, the assumed design process is outlined in order to describe the context of this chapter's topic. At the end, the car rental case that will be used for illustration throughout is introduced.

Section 7.3 reviews how to extend conventional normalization concepts to apply to temporal relations, leading to temporal keys and normal forms. It then argues that the properties of attributes are relative to the objects they describe and thus introduces surrogates for representing real-world objects in the model. The following subsections address in turn different aspects of time-varying attributes, namely lifespans, time patterns, derivation functions, temporal support, and temporal precision. Lifespans describe when attributes have values; time patterns identify when attributes change values and when the changes are recorded in the database; derivation functions describe how the values of an attribute for all times within its lifespan are computed from stored values; and the temporal support and precision of attributes indicate the temporal aspects that are relevant for the attributes and with which temporal granularity the aspects are to be recorded.

Section 7.4, on decomposition guidelines, is devoted to the implications of the temporal properties for conceptual database design. Section 7.5 surveys other approaches to temporally enhanced database design. The final section summarizes and points to opportunities for further research.

7.2 Temporal Database Design—Overview and Context

This section sets the context for discussing temporally enhanced database design. Specifically, we first adopt a particular model of time itself, then add time to conventional relations to yield the conceptual temporal relations employed in the chapter. We also define essential algebraic operators on the temporal relations. A description of the database design process follows, and the section ends with an introduction of the car rental case.

7.2.1 Modeling and Representing Time

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

Consequently, our model of the real time line is that of a finite sequence of *chronons* (19). In mathematical terms, this is isomorphic to a finite sequence of natural numbers (20). The sequence of chronons may be thought of as representing a partitioning of the real time line into equal-sized, indivisible segments. Thus, chronons are thought of as representing time segments such as femtoseconds or seconds, depending on the particular data processing needs. Real-world time instants are assumed to be much smaller than chronons and are represented in the model by the chronons during which they occur. We will use c, possibly indexed, to denote a chronon.

A time interval is defined as the time between two instants, a starting and a terminating instant. A time interval is then represented by a sequence of consecutive chronons where each chronon represent all instances that occurred during the chronon. We may also represent a sequence of chronons simply by the pair of the starting and terminating chronon. The restriction that the starting instant must be before the ending instant is necessary for the definition to be meaningful in situations where an interval is represented by, e.g., a pair of identical chronons. Unions of intervals are termed *temporal elements* (14).

7.2.2 Temporal Database Schemas

Two temporal aspects are of general relevance to data recorded in a database. To capture the time-varying nature of data, time values from two orthogonal time domains,

namely valid time and transaction time, are associated with the tuples in a bitemporal conceptual relation instance. Valid time captures the time-varying nature of the portion of reality being modeled, and transaction time models the update activity associated with the database.

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

Next, we define a set of names, $\mathcal{D}_A = \{A_1, A_2, \ldots, A_{n_A}\}$, for explicit attributes and a set of domains for these attributes, $\mathcal{D}_D = \{D_1, D_2, \ldots, D_{n_D}\}$. For these domains, we use $\perp_i$, $\perp_u$, and $\perp$ as inapplicable, unknown, and inapplicable-or-unknown null values, respectively (see, e.g., (1)). We also assume that a domain of surrogates is included among these domains. Surrogates are system-generated unique identifiers, the values of which cannot be seen, but only compared for identity (17). Surrogate values are used for representing real-world objects. With the preceding definitions, the schema of a bitemporal conceptual relation, R, consists of an arbitrary number, e.g., n, of explicit attributes from $\mathcal{D}_A$ with domains in $\mathcal{D}_D$, and an implicit timestamp attribute, T, with domain $2^{(\mathcal{D}_{TT} \cup \{UC\}) \times \mathcal{D}_{VT}}$. Here, UC ("until changed") is a special transaction-time marker. A value (UC, c^v) in a timestamp for a tuple indicates that the tuple being valid at time c^v is current in the database. The example below elaborates on this.

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

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

An arbitrary subset of the domain of valid times is associated with each tuple, meaning that the fact recorded by the tuple is *true in the modeled reality* during each valid-time chronon in the subset. Each individual valid-time chronon of a single tuple has associated a subset of the domain of transaction times, meaning that the fact, valid during the particular chronon, is *current in the relation* during each of the transaction-time chronons in the subset. Any subset of transaction times less than the current time and including the value UC may be associated with a valid time. Notice that while the definition of a bitemporal chronon is symmetric, this explanation is asymmetric. This asymmetry reflects the different semantics of transaction and valid time.

We have thus seen that a tuple has associated a set of so-called *bitemporal chronons* in the two-dimensional space spanned by transaction time and valid time. Such a set is termed a *bitemporal element* (19) and is denoted t^b. Because no two

CuID	Name	Address	Rating	T
007	Leslie	Birch Street	Preferred	$\{(5,5),\ldots,(5,20),\ldots,(9,5),\ldots,(9,20)\}$
007	Leslie	Elm Street	Preferred	$\{(5,21),\ldots,(5,30),\ldots,(25,21),\ldots,$ $(25,30),(UC,21),\ldots,(UC,30)\}$
007	Leslie	Beech Street	Preferred	$\{(10,5),\ldots,(10,20),\ldots,(25,5),\ldots,$ $(25,20),(UC,5),\ldots,(UC,20)\}$

Figure 7.1 A Bitemporal Conceptual Relation

tuples with mutually identical explicit attribute values (termed *value-equivalent*) are allowed in a bitemporal relation instance, the full history of a fact is contained in a single tuple.

Example 7.1

Consider a bitemporal relation recording information about the customers in a rental car company. The schema has these explicit attributes:

```
Customer = (CuID, Name, Address, Rating)
```

Each customer has a unique customer id, `CuID`, a name, and an address. Also, a rating is maintained that records the value of the customer to the company. This rating is used for preferential customer treatment.

In this example, we assume that the granularity of chronons is one day for both valid time and transaction time, and the period of interest is some given month in a given year, e.g., January 1995. Throughout, we use integers as timestamp components. The reader may informally think of these integers as dates, e.g., the integer 15 in a timestamp represents the date January 15, 1995. The current time is assumed to be 25 (i.e., $now = 25$).

Figure 7.1 shows an instance, `customer`, of this relation schema. The special value *UC* in the relation signify that the given tuple is still current in the database and that new chronons will be added to the timestamps as time passes and until the tuple is logically deleted.

The relation shows the employment information for Leslie, a preferred customer. On time 5, it is recorded that Leslie's address will be Birch Street, from time 5 to time 20, and Elm Street, from time 21 to time 30. Subsequently, it was discovered that Leslie's address was not Birch Street, but rather Beech Street, from time 5 to time 20. As a result, on time 10, the information about Birch Street was (logically) deleted, and the correct information was inserted.

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

Valid-time relations and transaction-time relations are special cases of bitemporal relations that support only valid time or transaction time, respectively. Sets of

valid-time and transaction-time chronons are termed *valid-time* and *transaction-time elements* and are denoted by t^v and t^t, respectively.

The remainder of the chapter will show how to extend existing normalization and decomposition theory to apply to temporal relations such as that shown in Figure 7.1.

Example 7.2

As an aside, we now exemplify how this approach may be further extended to apply to a temporal object-oriented data model, in this example, a very simple one. We use an object-oriented model in which each tuple of Figure 7.1 is modeled as a separate object, so in this case we have three distinct objects, each with its own identity. This `Customer` object type has four attributes, `CuID`, `Name`, `Address`, and `Rating`, as well as an implicit bitemporal element indicating the temporal extent of the object instance. Note that these three objects in the class shown in Figure 7.1 all provide information about the same customer, Leslie.

Now, applying the decomposition approach introduced later in this chapter may replace this object type with new object types, each with a subset of the attributes, and each with an implicit valid-time, transaction-time or bitemporal element. This chapter will give the rules by which these new object types can be configured, to avoid anomalies and redundancy implicit in the original object type.

Note that more complex temporal object-oriented models may also benefit from such decomposition, though the benefits and specifics of applying these decompositions vary from model to model. Hence, the remainder of this chapter will be in terms of the relational model, leaving it to the reader to reformulate these decompositions in terms of the temporal object-oriented model of their choice.

7.2.3 Associated Algebraic Operators

We have so far described the database structures in the bitemporal conceptual data model—relations of tuples timestamped with bitemporal elements. We now define some algebraic operators on these structures that will be used later. A complete algebra is defined elsewhere (36).

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

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

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

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

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

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

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

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

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

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

Example 7.3

Consider the `customer` relation shown in Figure 7.1. The following result is produced by $\tau^{\mathrm{B}}25$(`customer`).

CuID	Name	Address	Rating	T
007	Leslie	Elm Street	Preferred	$\{5, \ldots, 25\}$

This says that at transaction time 5 we stored this information, and this information is still current (at time 25). The valid-timeslice operator selects all tuples with a timestamp that contains a chronon that has the argument chronon as its second component. The timestamp of result tuples contain those transaction-time chronons that were associated with the argument valid-time chronon.

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

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

7.2.4 Overview of the Design Process

The topics considered in this chapter are displayed in their data modeling context in Figure 7.2 and are discussed in the following.

We assume that an atemporal database schema is initially produced. This database schema consists of atemporal versions of the conceptual-relation schemas described

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

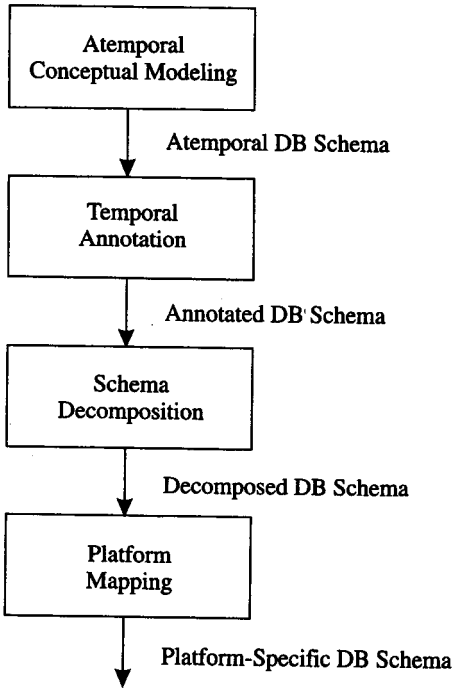

Figure 7.2 Data Modeling Context

earlier in this section. The database schema is atemporal in the sense that all temporal aspects related to valid and transaction time of the relation schemas are simply ignored—or left unspecified. These atemporal relation schemas may also be thought of as primitive object types: Each instance of an object type, i.e., each object (or tuple), has an ID (a surrogate) that is independent of its state, and the state is described solely using single-valued attribute values with domains defined by built-in types.

A wide range of design approaches may be employed to produce the initial atemporal database schema—no assumptions are made.

In the next step, the relation schemas, or primitive object-type schemas, are annotated with temporal properties, to be defined in the next section. Following the annotation, the schema description is complete. The subsequent step is then to apply decomposition guidelines, to be defined in Section 7.4, to the schemas, leading to a decomposed conceptual database schema, with genuine temporal relation schemas as defined earlier. This database schema may subsequently be mapped to various implementation platforms, e.g., SQL–92 (30), SQL3, or TSQL2 (39).

It is an underlying rationale that the database is to be managed by a relational, or temporal-relational, DBMS that employs tuple timestamping. Indeed, increasingly many databases are being managed by relational DBMSs; and these systems, in addition to most temporal relational prototype DBMSs, employ tuple timestamping.

```
Branch = (BrID, Name, Location, Manager, AssistantMgr, Capacity)
Car = (CarID, Branch, Model, Make, Category, Year, Mileage, LastServiced)
Customer = (CuID, Name, Address, Rating)
RentalBooking = (RBID, Branch, Category, Customer, Price, CreditCardNo,
                 CardType)
```

Figure 7.3 Car Rental Schema

Hence, the decomposition that maps the atemporal database schema to a decomposed, tuple-timestamped temporal database schema is an important component of a design framework. Specifically, this decomposed temporal database schema provides an attractive starting point for mapping the database schema to the database schema of a specific DBMS (such as CA Ingres, DB2, Informix, Microsoft, Oracle, or Sybase). It is attractive because the "conceptual distance" to the various DBMS schemas is small.

Throughout, we will use the car rental case for exemplification.

7.2.5 Example—Car Rentals

Figure 7.3 describes the car rental database schema that will be annotated with temporal properties and decomposed in the next two sections. The aspects of the schema that are not self-explanatory are described briefly next. Branches have `Manager` and `Location` attributes. The `Capacity` attribute indicates the maximum number of cars a branch is able to manage. A car belongs to a specific branch, so attribute `Branch` is a foreign key referencing table `Branch`. The `Customer` relation was introduced in Example 7.1. When a car rental is booked, the booking is for a specific branch and car category (e.g., Economy, Compact, Mid-size). It is made by a customer who is quoted a certain price, and the customer made the reservation using a credit card.

7.3 Temporal Properties of Data

The schema just presented is atemporal; there is no mention of time-varying values. The following sections will discuss how this schema is elaborated and decomposed when time is considered. But first we examine conventional functional dependencies, which will be subsequently applied to time-varying relations.

7.3.1 Functional Dependencies

Functional dependencies play a central role in conventional database design and should also do so in our framework. In our framework, we initially design a nontemporal database schema and ignore the issues of valid and transaction time. Thus, different attributes in the same initial schema may have different requirements with

respect to temporal support; for example, some attributes may require valid-time support while other attributes do not.

As background for considering temporal functional dependencies, we next state the notion of a functional dependency for conventional (snapshot) relations.

Definition 7.1
Let a relation schema R be defined as $R = (A_1, A_2, \ldots, A_n)$, and let X and Y be sets of attributes of R. The set Y is *functionally dependent* on the set X, denoted $X \rightarrow Y$, if for all meaningful instances r of R,

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

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

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

7.3.2 Temporal Functional Dependencies

Generalizations of conventional dependencies make it meaningful to apply dependencies to the initial schemas, similarly to how dependencies are applied in conventional normalization. The designer's use of the dependencies is not affected by the attributes' particular levels of temporal support.

7.3.2.1 Generalizing Functional Dependencies to Temporal Relations

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

To be specific, consider the restricted case of a transaction-time relation r, with schema $R = (A_1, \ldots, A_n|\mathrm{T})$, and a parallel snapshot relation r' with the same schema (but without the implicit timestamp attribute), i.e., $R' = (A_1, \ldots, A_n)$. The current state of r, denoted by $\rho^{\mathrm{T}}_{now}(r)$, where "*now*" denotes the current time, will faithfully track the current state of r'. Past states of r' will be retained in r, and can be extracted via $\rho^{\mathrm{T}}_{t}(r)$, with "t" being the desired past point in time. A functional dependency on R' will hold for all possible extensions, and hence for all past states of r'. Hence, the same functional dependency must hold for all snapshots of r (this insight first appeared over a decade ago (4)). A similar argument can be applied to valid-time relations and to bitemporal relations, yielding the following characterization (22).

Definition 7.2
Let X and Y be sets of non-timestamp attributes of a bitemporal relation schema R. A *temporal functional dependency*, denoted $X \xrightarrow{\mathrm{T}} Y$, exists on R if for all meaningful

instance r of R,

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

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

The parallel between conventional functional dependencies and temporal functional dependencies means that inference rules such as Armstrong's axioms have close temporal counterparts that play the same role in the temporal context as do the non-temporal rules in the non-temporal context. Next, we can also define temporal keys (22). For example, the explicit attributes X of a temporal relation schema R form a *(temporal) key* if $X \xrightarrow{T} R$. Finally, we can generalize snapshot normal forms in a similar manner.

Example 7.4

These are some of the dependencies that hold in the rental car database schema (this schema was shown in Figure 7.3):

```
In Branch:   Name  --T-->  Location Manager AssistantMgr Capacity
In Car:      CarID --T-->  Branch Model Make Category Year Mileage
                           LastServiced
In Customer: CuID  --T-->  Rating
             CuID Address --T--> Name
```

In table `Customer`, attribute `CuID` determines `Rating`, but because the same customer may have several names and addresses at a time, `CuID` does not determine `Name` and `Address`. Attributes `CuID` and `Address` together determine `Name` because customers have only one `Name` associated with each of their possible several addresses. Note that in table `Branch`, `Name` (and also `BrID`) is a key, as is `CarID` in schema `Car`.

Definition 7.3

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

$$\forall X \xrightarrow{T} Y \in F^+(Y \subseteq X \vee X \xrightarrow{T} R).$$

Definition 7.4

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

In a similar fashion, it is possible to devise temporal variants of other well-known dependencies (e.g., multi-valued and join) and normal forms (e.g., fourth and fifth normal forms). Similarly, the notions of lossless-join and dependency-preserving

decomposition can be naturally extended to temporal relations. Furthermore, one can define temporal variants of conventional integrity constraints involving uniqueness, referential integrity, and subset and cardinality constraints.

To illustrate, we define foreign key constraints. Let bitemporal schemas R and S have explicit attributes $A_1, A_2, \ldots, A_n$ and $B_1, B_2, \ldots, B_m$, respectively. Attributes X of S is a foreign key referencing attributes Y of R if X and Y have the same number of attributes, if the attributes of X and Y are pair-wise compatible, and if for all meaningful instances r of R and s of S,

$$\forall c^v, c^t(\pi^{\mathrm{B}}_X(\tau^{\mathrm{B}}_{c^v}(\rho^{\mathrm{B}}_{c^t}(s))) \subseteq \pi^{\mathrm{B}}_Y(\tau^{\mathrm{B}}_{c^v}(\rho^{\mathrm{B}}_{c^t}(r)))).$$

If both R and S do not support valid or transaction time, the corresponding timeslice operations are simply omitted in the subset condition above. If only one of R and S supports valid time then the valid-timeslice operation with time argument *NOW* is applied to that relation. The same applies to transaction time. For example, if R supports only transaction time and S supports only valid time, the condition becomes $\pi^{\mathrm{B}}_X(\tau^{\mathrm{B}}NOW(s)) \subseteq \pi^{\mathrm{B}}_Y(\rho^{\mathrm{B}}NOW(r))$.

Example 7.5

In the rental car database schema, all relation schemas are in TBCNF, with the exception of schema `Customer` where non-trivial dependency `CuID` $\xrightarrow{\mathrm{T}}$ `Rating` violated the requirement that the left-hand side must be a superkey. To bring the database schema to TBCNF, `Customer` is thus decomposed into two schemas.

```
CustomerAddr = (CuID, Name, Address)
CustomerRating = (CuID, Rating)
```

Here, `CuID` of `CustomerRating` would be declared as a foreign key referencing `CuID` of `CustomerAddr`.

7.3.2.2 *Strong Temporal Functional Dependencies*

The temporal dependencies we have seen thus far apply snapshot dependencies to individual snapshots in isolation. Thus, these dependencies are not capable of capturing the relative variation over time of attribute values. So while we were able to capture dependencies such as a salary attribute (at any time) being determined by an employee-name attribute, we cannot capture that a salary of an employee does not change within a month, or never changes. These latter constraints require looking at more than one time point to determine if the constraint is satisfied by a particular relation instance. This distinction has previously been captured more generally with the terms *intrastate* and *interstate* integrity constraints (3).

While a temporal dependency holds if the corresponding conventional dependency holds for each snapshot in isolation, we now "bundle" tuples of certain snapshots and require the corresponding snapshot dependency to hold for each "bundle" in isolation. A "bundle" is defined to contain all tuples in all valid timeslices of the

result obtained from applying a single transaction timeslice operation to a meaningful bitemporal database instance of the schema under consideration. This is stated more precisely below.

Definition 7.5

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

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

Strong temporal dependencies are useful in part because they have a practical and intuitive interpretation. Specifically, if $X \xrightarrow{\text{Str}} Y$ holds on a relation schema, this means that Y does not vary with respect to X.

Example 7.6

In the rental car schema, there are several strong dependencies, e.g., the following.

```
In Car:           CarID --Str--> Model Make Category Year
In RentalBooking: RBID --Str--> Branch Category Customer Price
                                CreditCardNo CardType
```

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

In the strong temporal dependency $X \xrightarrow{\text{Str}} Y$, attributes X may vary more often than attributes Y, but X must change when Y changes.

Definition 7.6

Let X and Y be sets of non-timestamp attributes of a bitemporal relation schema R. A *strong temporal equivalence*, denoted $X \xleftrightarrow{\text{Str}} Y$, exists on R if $X \xrightarrow{\text{Str}} Y$ and $Y \xrightarrow{\text{Str}} X$.

Intuitively, $X \xleftrightarrow{\text{Str}} Y$ means that the sets of attributes X and Y change values simultaneously, and are thus synchronous. We return to this issue in Section 7.4.4.

It is possible to take these notions of dependencies even further, as has subsequently been done by Wang and his colleagues (45) and by Wijsen (46). Wang et al. generalized strong dependencies to dependencies that were along a spectrum between our temporal functional dependencies, which apply to individual timeslices, and strong functional dependencies, which apply to all timeslices at once. Specifically, they define a functional dependency for each available *granularity* (e.g., second, week, year), and require that the equality holds only during a unit of the granularity. Next, Wijsen has recently developed a normalization theory for valid-time databases that includes three types of temporal dependencies. Two correspond to our temporal dependency and strong temporal dependency. The third dependency is in-between the two. This so-called *dynamic dependency* holds if the corresponding snapshot dependency holds on the unions of all pairs of consecutive snapshots.

7.3.3 Using Surrogates

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

We employ surrogates for representing real-world entities in the database. In this regard, we follow the approach adopted in, e.g., the TEER model by Elmasri (11). Surrogates do not vary over time in the sense that two entities identified by identical surrogates are the same entity, and two entities identified by different surrogates are different entities. We assume the presence of surrogate attributes during the design process. Just prior to performing the implementation-platform mapping, surrogate attributes may be either (a) retained, (b) replaced by regular (key) attributes, or (c) eliminated.

Example 7.7
In our database schema, we add a surrogate to each of the (now) six tables. For example, we add a surrogate for branches, `BrSur`, to table `Branch` and a surrogate for cars, `CarSur`, to table `Car`.

Definition 7.7
Let X be a set of non-timestamp attributes of a bitemporal relation schema R with surrogate attribute S. Then X is said to be *time invariant* if $S \xrightarrow{\text{Str}} X$.

Because it is assumed that different entities are represented by different surrogates and the same entity always is represented by the same surrogate, this is a rather natural definition of *time invariant* attributes. By combining standard temporal dependency and strong temporal dependency, the notion of a time-invariant key (which had previously been used with a different meaning (31)) results.

Definition 7.8
Let X be a set of non-timestamp attributes of a bitemporal relation schema R with surrogate attribute S. Then X is termed a *time-invariant key (TIK)* if $S \xrightarrow{\text{Str}} X$ and $X \xrightarrow{\text{T}} R$.

The first requirement to attributes X is that they be time invariant. The second is that they be a temporal key. In combination, the requirements amount to saying that X is a key with values that do not change (with respect to the surrogate attribute).

Example 7.8
For schema Branch, we have seen that Name and BrID are keys. Because BrSur strongly determines only BrID, and not Name, BrID is a time-invariant key. The intuition is that a branch may change name, but not its BrID value. In schema RentalBooking, we have that RBSur $\xrightarrow{\text{Str}}$ RBID, so as we have seen that RBID is a key, RBID is also a time-invariant key. Surrogates such as BrSur and RBSur in relations with a time-invariant key are eliminated from the schema.

7.3.4 Lifespans of Individual Time-Varying Attributes

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

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

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

Definition 7.9
Let a relation schema $R = (S, A_1, \ldots, A_n \mid \mathrm{T})$ be given, where S is surrogate valued, and let r be an instance of R. The *lifespan* for an attribute A_i, $i = 1, \ldots, n$, with respect to a value s of S in r is denoted $\mathbf{ls}(r, A_i, s)$ and is defined by $\mathbf{ls}(r, A_i, s) = \mathbf{vte}(\sigma^{\mathrm{B}}_{S=s \wedge A \neq \perp_i}(r))$.

Lifespans are important because attributes are guaranteed to not have any inapplicable null value during their lifespans.

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

Definition 7.10
Let a relation schema $R = (S, A_1, \ldots, A_n \mid \mathrm{T})$ be given where S is surrogate valued. Two attributes A_i and A_j in R are termed *lifespan equal* with respect to surrogate S, denoted $A_i \overset{\mathrm{LS}}{=}_S A_j$, if for all meaningful instances r of R, $\forall s \in \mathrm{dom}(S)(\mathbf{ls}(r, A_i, s) = \mathbf{ls}(r, A_j, s))$.

Example 7.9
In schema Car, all attributes are mutually lifespan equal: values exist for all attributes when a car is first registered at a branch, and meaningful values persist for all attributes.

All branches have a manager, but small branches have no assistant manager. Thus, some branches only get a meaningful value for attribute AssistantMgr after having

reached a certain capacity. This means that `AssistantMgr` is not lifespan equal to the other attributes, e.g., `Manager` and `Capacity`.

The importance of lifespans in temporal databases has been recognized in the context of data models in the past (cf. (6, 5, 11)). Our use of lifespans for database design differs from the use of lifespans in database instances. In particular, using lifespans during database design does not imply any need for storing lifespans in the database.

7.3.5 Time Patterns of Individual Time-Varying Attributes

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

Definition 7.11
The *time pattern* T is a partial function from the natural numbers $\mathcal{N}$ to a domain $\mathcal{D}_T$ of times: $T : \mathcal{N} \hookrightarrow \mathcal{D}_T$. If $T(i)$ is defined, so is $T(j)$ for all $j < i$. We term $T(i)$ the i'th time point.

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

Note that an attribute may not actually change value at a time point because it may be the case that the existing and new values are the same. The times when changes take place and the resulting values are orthogonal aspects.

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

Definition 7.12
Define relation schema $R = (S, A_1, \ldots, A_n \mid \mathrm{T})$ where S is surrogate valued. Two attributes A_i and A_j in R, with observation patterns $O_{A_i}^S$ and $O_{A_j}^S$, are *synchronous* with respect to S, denoted $A_i \overset{S}{=}_S A_j$, if for all meaningful instances r of R and for all surrogates s,

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

Thus, attributes are synchronous if their lifespans are identical when restricted to the intersection of their lifespans.

Example 7.10
In schema `Car`, attributes `Branch`, `Model`, `Make`, `Category`, and `Year` are synchronous (if ownerships of cars often shift among branches, `Branch` would not be considered synchronous with the four other attributes). Each of attributes `Mileage` and `LastServiced` not synchronous with other attributes in the schema. `Mileage` is updated when a car is returned, and `LastServiced` is updated when a car is serviced (which occurs less frequently!).

In schema `RentalBooking`, values for all attributes are provided when a booking is made and are not subsequently updated. Thus, all attributes in this schema are synchronous.

7.3.6 The Values of Individual Time-Varying Attributes

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

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

Definition 7.13
A *derivation function* f is a partial function from the domains of valid times $\mathcal{D}_{VT}$ and relation instances r with schema R to a value domain D in the universal set of domains $\mathcal{D}_D$, i.e., $f : \mathcal{D}_{VT} \times r(R) \hookrightarrow D$.

Example 7.11
The `Mileage` attribute of `Car` has associated two derivation functions. One function interpolates recorded mileage values for cars so that a value may be provided for all times. Among other uses, this function is used to project future mileage when scheduling maintenance for the cars. The other derivation function is the discrete derivation function that does not manufacture any information, but only provides mileage values for the times when they are actually recorded.

The importance of derivation functions in data models has previously been argued convincingly by, e.g., Klopprogge and Lockemann (25), Clifford and Crocker (6) and Segev and Shoshani (35).

7.3.7 Temporal Support of Attributes

During database design, a model of a part of reality is created. What aspects of the modeled reality to capture and what to leave out is determined by the functional

requirements to the application being created. The application may require any combination of valid-time and transaction-time support, or no temporal support, for each of the time-varying attributes.

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

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

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

Example 7.12

In schema `CustomerAddr`, support for neither valid nor transaction time is required. In `CustomerRating`, valid-time state support is required for the `Rating` attribute. In schema `RentalBooking`, we require both valid-time-state and transaction-time support for all attributes. The valid time records when the booking is for, and old bookings are to be retained. In schema `Car`, attribute `Mileage` requires valid-time-event support and transaction-time support. The remaining attributes require only transaction-time support.

7.3.8 Temporal Precision of Attributes

Each time-varying attribute has an associated observation pattern, as discussed in Section 7.3.5. A time pattern is a function to a time domain, that has an associated time granularity. The granularity is the precision in which the time-variance is recorded. If a hiring decision occurred sometime during the business day, but it is not known exactly when (i.e., what minute or hour) the decision occurred, then it is inappropriate to store that fact with a timestamp at a minute granularity. The reason is that a particular minute must be chosen, and that minute is probably incorrect, with the implication that the model is incorrect (7).

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

One approach is to use the minimum granularity of the DBMS at the precision of all relations. As just discussed, this results in a low-fidelity model of reality. A better approach is to choose the most appropriate granularity for each relation. We propose a simple strategy. First, each attribute is associated with a set of granularities. The smallest granularity in this set is the granularity in which the time-variance of

the attribute is known. Other, coarser granularities represent granularities which are acceptable to the applications utilizing the relation. Then the relation is decomposed only if there is not a common granularity that is a member of the granularity sets of all attributes.

Example 7.13
In schema `Car`, values for `Mileage` must be recorded with a precision of minutes. The granularity of hours is too coarse because a car may change its mileage within the same hour, which should be possible to record. Bookings of rentals must be recorded by the minute or second. Thus, the attributes of `RentalBooking` have minute and second as their acceptable granularities.

7.3.9 Summary of Attribute Semantics

In summary, the database designer is expected to initially annotate the relation schemas using (regular and strong temporal) dependencies. Then surrogates are used for the modeling of entity types. The notions of lifespans, time patterns, and derivation functions are used for capturing the semantics of attributes, and the temporal support and precision of the attributes are recorded.

Below, we summarize the tasks of the database designer. The designer starts with a set of atemporal conceptual relation schemas in hand. To annotate these schemas with temporal properties, the indicated tasks are performed.

1. *Identify entity types and represent them with surrogate attributes.* The real-world objects (or entities) that the attributes of the database describe are represented with surrogate attributes. Here, time-invariant keys are also identified.
2. *Determine the required temporal support.* For each attribute, indicate the required temporal support for the attribute. Record the interactions (if any) between the valid time and the transaction time implied by the temporal specializations in effect for the attribute.
3. *Describe precisions.* For each time-varying attribute, indicate its set of applicable granularities.
4. *Describe lifespans.* For each relation schema, describe the lifespans of the attributes.
5. *Determine observation and update patterns.* For each relation schema, indicate which attributes are synchronous, i.e., share observation and update patterns.
6. *For each attribute, indicate its appropriate derivation or interpolation function(s).* The functions concern interpolation in valid-time, and there may be several functions per attribute.
7. *Specify temporal functional dependencies on the schemas.* This includes the identification of (primary) keys.
8. *Specify strong temporal functional dependencies.*

7.4 Decomposition Guidelines

In this section, we discuss how the properties of schemas with time-varying attributes as captured in the previous section are used during database design. Emphasis is on the use of the properties for schema decomposition. In addition, issues relevant to the related aspects of view and physical design are touched upon as well.

Database designers are faced with a number of design criteria which are typically conflicting, making database design a challenging task. So, while we discuss certain design criteria in isolation, it is understood that there may be additional criteria that should also be taken into consideration (e.g., good join performance).

Two important goals are to eliminate the use of inapplicable nulls and to avoid the repetition of information. Additionally, the conceptual model employed poses constraints on what attributes that may reside in the same relation schema. We formulate decomposition guidelines that address these concerns.

7.4.1 Normal Form Decomposition

With the introduction of temporal functional dependencies, it is possible to apply conventional normalization theory to our conceptual relations. Thus, dependencies are indicated, satisfaction of normal forms is tested, and relation schemas are decomposed where necessary.

With the introduction of strong temporal functional dependencies and surrogates, it became possible to distinguish between time-varying keys and time-invariant keys, where the latter may serve the purposes of surrogates.

7.4.2 Temporal Support and Precision Decomposition Rules

The characterization of attributes according to the temporal support they require is important for database design because the conceptual data model permits only one type of temporal support in a single relation (as do also temporal implementation data models). We embed this requirement in a simple decomposition rule.

Definition 7.14(Temporal Support Decomposition Rule.)
To achieve the correct temporal support of time-varying attributes, decompose temporal relation schemas to have only attributes with the same temporal support requirements in the same schema, except for the surrogate attribute(s) forming the primary key.

Example 7.14
Schema `Car` must be decomposed. Specifically, attribute `Mileage` is removed from the schema, and a new schema, `CarMileage`, with attributes `CarID` and `Mileage` is introduced.

It may be possible to avoid such decomposition in certain circumstances, but the designer should be aware of the potential drawbacks of doing so. Consider includ-

ing an attribute S requiring snapshot support together with an attribute T requiring transaction-time support, in a transaction-time relation. Because it is embedded in a transaction-time relation, it is given transaction-time support, and past values are automatically retained. Taking the transaction timeslice at *now* produces the correct values for S, but taking a transaction timeslice at a time in the past, at time $c_t < now$, may retrieve an old value of S, which is inconsistent with the requirement that it be a snapshot attribute. Such queries must take this into account, timeslicing the relation as of *now* to get the value of S, then join this with the timeslice of the relation as of c_t to get the value of T, which is quite awkward.

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

Next, the existence of a strict correlation (a type of temporal specialization (18)) between the valid and transaction time of an attribute can reduce the need for decomposition. Consider an attribute D that requires both valid-time and transaction-time support, but which is degenerate, i.e., the valid and transaction times are exactly correlated. Thus whenever a change occurs in the modeled reality, the new data is immediately recorded in the database. This attribute may reside in a valid-time relation with another attribute V requiring only valid-time support. Transaction-time queries can be recast as valid-time queries on the relation, exploiting the correlation between the two kinds of time. Similarly, D may reside in a transaction-time relation with the attribute T.

Moving on to precisions, the conceptual data model, and indeed all temporal relational data models, support only a single precision per relation for each of transaction and valid time. It then becomes necessary to separate attributes that require different, incompatible precisions.

Definition 7.15(Precision Decomposition Rule.)
To accurately reflect the temporal precisions of time-varying attributes, decompose relation schemas so that all attributes in a schema have a compatible temporal precision, that is, a common granularity.

Example 7.15
The Precision Decomposition Rule does not give rise to decomposition in the car rental schema.

A more general approach was recently proposed by Wang and his colleagues, using their temporal functional dependencies based on granularities (45), discussed briefly in Section 7.3.2.2. Their approach is complex and may generate new granularities, of uncertain comprehensibility by the user. The Precision Decomposition Rule above is very simple and does not generate new granularities, but may decompose relations more than Wang's approach.

7.4.3 Lifespan Decomposition Rule

One important design criterion in conventional relational design is to eliminate the need for inapplicable nulls in tuples of database instances. We introduced in Section 7.3.4 the notion of lifespans in order to capture when attributes are defined for the objects they are introduced in order to describe. Briefly, the lifespan for an attribute—with respect to a particular surrogate representing the object described by the attribute—is all the times when a meaningful attribute value, known or unknown, exists for the object.

The following definition uses the concepts from Section 7.3.4 to characterize temporal database schemas with instances that do not contain inapplicable nulls.

Definition 7.16
A relation schema $R = (S, A_1, \ldots, A_n \mid \mathrm{T})$ where S is surrogate valued is *lifespan homogeneous* if $\forall A, B \in R(A \overset{\mathrm{LS}}{=\!=}_S B)$.

With this definition, we are in a position to formulate the Lifespan Decomposition Rule, which ties the connection of the notion of lifespans of attributes with the occurrence of inapplicable nulls in instances.

Definition 7.17(Lifespan Decomposition Rule.)
To avoid inapplicable nulls in temporal database instances, decompose temporal relation schemas to ensure lifespan homogeneity.

Example 7.16
In schema `Branch`, attribute `AssistantMgr`'s lifespan deviated from those of the other attributes. Thus, `AssistantMgr` is removed from `Branch`, and a new schema, `Assistant`, with attributes `AssistantMgr` and `BrID` is introduced.

It is appropriate to briefly consider the interaction of this rule with the the existing temporal normal forms that also prescribe decomposition of relation schemas. Specifically, while the decomposition that occurs during normalization does, as a side effect, aid in eliminating the need for inapplicable nulls, a database schema that obeys the temporal normal forms may still require inapplicable nulls in its instances. By adjusting the schema, the lifespan decomposition rule attempts to eliminate remaining inapplicable nulls.

7.4.4 Synchronous Decomposition Rule

The synchronous decomposition rule is based on the notion of observation pattern, and its objective is to eliminate a particular kind of redundancy. In Section 7.3.5, we defined the notion of synchronous attributes, which is here employed to define synchronous schemas and the accompanying decomposition rule. Finally, we view synchronism in a larger context, by relating it to existing concepts, and discuss the decomposition rule's positioning with respect to logical versus physical design.

With this definition, we can characterize relations that avoid the redundancy caused by a lack of synchronism and then state the Synchronous Decomposition Rule.

Definition 7.18
Define relation schema $R = (S, A_1, \ldots, A_n \mid \mathrm{T})$ where S is surrogate valued. Relation R is *synchronous* if $\forall A_i, A_j \in R(A_i \stackrel{S}{=}_S A_j)$.

Definition 7.19(Synchronous Decomposition Rule.)
To avoid repetition of attribute values in temporal relations, decompose relation schemas until they are synchronous.

Example 7.17
In the current `Car` schema attribute `LastServiced` is not synchronous with the remaining attributes. In consequence, this `LastServiced` is removed from `Car` and the schema `CarService = (CarID, LastServiced)` is included into the car rental database schema.

Alternative notions of synchronism have previously been proposed for database design by Navathe and Ahmed (31), and by Wijsen (46). While these notions are stated with varying degrees of clarity and precision and are defined in different data-model contexts, they all seem to capture the same basic idea, namely that of *value-based synchronism*, which differs from the synchronism used in this chapter.

It is our contention that in this context, the synchronous decomposition rule is only relevant at the level of the schema of the implementation platform, and depending on the actual implementation platform, the rule may be relevant only to physical database design. Surely, the redundancy that may be detected using the synchronism concept is important when *storing* temporal relations. Next, this type of redundancy is of little consequence for the querying of logical-level relations using the TSQL2 query language (21, 39), a particular implementation platform. Indeed, it will often adversely affect the ease of formulating queries if logical-level relations are decomposed solely based on a lack of synchronism.

Finally, the need for synchronism at the logical level has previously been claimed to make normal forms and dependency theory inapplicable (e.g., (13)). The argument is that few attributes are synchronous, meaning that relation schemas must be maximally decomposed, which leaves other normalization concepts irrelevant. This claim does not apply to the framework put forth here.

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

7.4.5 Implications for View Design

The only concept from Section 7.3 not covered so far is derivation functions. These relate to view design, as outlined next.

For each time-varying attribute, we have captured a set of one or more derivation functions that apply to it. It is often the case that exactly one derivation function applies to an attribute, namely the discrete interpolation function (21), which is a kind of identity function. However, it may also be the case that several nontrivial derivation functions apply to a single attribute.

By using the view mechanism, we maintain the separation between recorded data and data derived via some function. Maintaining this separation makes it possible to later modify existing interpolation functions.

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

Example 7.18
Two derivation functions were associated with attribute `Mileage` of schema `CarMileage`. As the discrete derivation function is the default for event relations, only one view has to be defined, namely one to produce the interpolated `Mileage` values.

7.4.6 Summary

In this section, we have provided a set of guidelines for the decomposition of conceptual relations based on their temporal properties. Here, we briefly review the proposed guidelines.

- With temporal functional dependencies as the formal basis, conventional normalization theory was made applicable to the conceptual relations considered here. In particular, the traditional normal forms, e.g., third normal form, BCNF, and fourth normal form, and their decomposition algorithms are applicable.
- The temporal support decomposition rule ensures that each relation has a temporal support appropriate for the attributes it contains.
- The precision decomposition rule uses the granularity sets to prescribe decomposition of relation schemas and to determine the granularity of the resulting relation schemas.
- The lifespan decomposition rule ensures that inapplicable nulls are not required.
- The synchronous decomposition rule removes redundant attribute values, while being less strict than previous definitions of value synchronism.
- Strong temporal functional dependencies, together with the temporal functional dependencies, allow the designer to identify time-invariant primary keys, which may play the role of surrogates that can then subsequently be eliminated.
- The derivation function associated with attributes induce views computing the derived values.

```
Branch = (BrID, Name, Location, Manager, Capacity)
                • valid-time state and transaction-time support is required
                • BrID is a time-invariant key and Name is a key
Assistant = (AssistantMgr, BrID)
                • valid-time state and transaction-time support is required
                • BrID is a foreign key referencing BrID of Branch
Car = (CarID, Branch, Model, Make, Category, Year)
                • transaction-time support is required
                • CarID is a time-invariant key
CarMileage = (CarID, Mileage)
                • valid-time event and transaction-time support is required
                • the precision for valid time is minutes
                • CarID is a foreign key referencing CarID of Car
CarMileageView = (CarID, S-C(Mileage))
                • view derived from CarMileage
                • S-C is a step-wise constant derivation function
CarService = (CarID, LastServiced)
                • transaction-time support is required
                • CarID is a foreign key referencing CarID of Car
CustomerAddr = (CuID, Name, Address)
                • no temporal support is required
                • (CuID, Name) is a key
CustomerRating = (CuID, Rating)
                • valid-time state support is required
                • CuID is a foreign key referencing CuID of CustomerAddr
RentalBooking = (RBID, Branch, Category, Customer, Price,
                CreditCardNo, CardType)
                • valid-time state and transaction-time support required
                • the precision for valid time is minutes
                • RBID is a time-invariant key
                • Branch is a foreign key referencing BrID of schema Branch
                • Customer is a foreign key referencing CuID of CustomerAddr
```

Figure 7.4 Final Conceptual Car Rental Schema

Example 7.19
Following the steps described here, the car rental schema in Figure 7.3 now appears as shown in Figure 7.4. Sample annotations are included.

While conceptual design is concerned with adequately modeling the *semantics* of the application, physical design is concerned with performance. The concepts concerning synchronism, i.e., time patterns, including observation and update patterns, are relevant for physical design. Their use was discussed in Section 7.4.4. Physical design may also reverse some of the decomposition that is indicated by logical design.

7.5 Other Approaches to Temporally Enhanced Database Design

This section surveys in turn approaches to temporally enhanced database design based on normalization concepts and approaches based on Entity-Relationship (ER) modeling.

7.5.1 Normalization-Based Approaches

For relational databases, a mature and well-formalized normalization theory exists, complete with different types of dependencies, keys, and normal forms. Over the past two decades, a wealth of temporal relational data models have been proposed. Because these temporal models utilize new types of relations, the existing normalization theory is not readily applicable, prompting a need to revisit the issues of database design.

The proposals for temporal normalization concepts, e.g., dependencies, keys, and normal forms, presented in this chapter are based in part on earlier concepts (surveyed in (22)). Space constraints preclude a detailed coverage of these earlier concepts; instead, we briefly survey but a few dependency and normal form concepts.

Some earlier works involving dependencies, e.g., those by Tansel and Garnett (40) and Lorentzos (28), treat nested relations with temporal information and relations with time interval-valued attributes that are unfoldable into relations with time point attributes as snapshot relations with explicit temporal attributes and apply "temporal" dependencies in these contexts. Other dependencies, specifically Vianu's dynamic dependency (44), Navathe and Ahmed's temporal dependency (31), and Wijsen's dynamic and temporal functional dependencies (46), are inter-state dependencies, and thus are more ambitious than the temporal dependency (an intra-state dependency) considered earlier in this chapter. In fact, these dependencies are more closely related to the notion of synchronism defined in Section 7.3.5 and based on observation patterns.

Now considering earlier normal forms, quite a diverse set of proposals exist. Ben-Zvi (2) bases his time normal form on the notion of a contiguous attribute. Informally, an attribute in a temporal relation is contiguous if there exists a value of that attribute for each point in time and for each real-world entity captured in the relation. Segev and Shoshani define, in their Temporal Data Model, a normal form, 1TNF, for valid-time relations (34). In their data model, it is possible for time-slice operations to result in attributes that have multiple values at a single point in time. The 1TNF normal form ensures that this anomaly is avoided. Navathe and Ahmed (31) base their time normal form (TNF) on their value-based notion of synchronous attributes and define 1TNF to ensure that time-varying attributes are synchronous, i.e., change at the same time. This value-based concepts is related to the identity-based notion of synchronous attributes defined earlier in the chapter. Lorentzos (28) defines a P normal form and a Q normal form. P normal form essentially guarantees the relation to be coalesced (37), and Q normal form appears to have similarities with Navathe and Ahmed's concept of synchronism.

As illustrated by the dependencies and normal forms surveyed above, the early proposals for normalization concepts are typically specific to a particular temporal data model. This specificity is a weakness since a given concept inherits the peculiarities of its data model; it is unsatisfactory to have to define each normalization concept anew for each of the more than two dozen existing temporal data models (33). Furthermore, the existing normal forms often deviate substantially in nature from conventional normal forms.

This chapter represents an attempt at lifting the definition of temporal normalization concepts from a representation-dependent, model-specific basis to a semantic, conceptual basis, in the process making the concepts readily applicable to an entire class of temporal relational data models.

Most recently, proposals that are consistent with and refine the approach adopted in this chapter (and in (21, 22)) have been developed. Specifically, Wang et al. (45) and Wijsen (47) have defined dependencies and associated normal forms that extend the normal forms provided here and that are based on temporal granularities and apply to complex objects. These proposals were discussed in Section 7.3.2.2.

7.5.2 ER-Based Design Approaches

The ER model, using varying notations and with some semantic variations, continues to enjoy a remarkable popularity in the research community, the computer science curriculum, and in industry.

As pointed out earlier, it has been widely recognized that temporal aspects of database schemas are prevalent and difficult to model. Because this also holds true when using the ER model, it is not surprising that enabling the ER model to properly capture time-varying information has been an active area of research for the past decade and a half. About a dozen temporally enhanced ER models have resulted. Reference (15) surveys and compares all such models known to its authors at the time of its writing.

Combined, the temporal ER models represent a rich body of insights into the temporal aspects of database design. Table 7.1 provides an overview of the models and contains references to further readings; the reader is encouraged to study the models.

7.6 Summary and Directions

In order to exploit the full potential of database technology—conventional as well as temporal—guidelines for the design of appropriate database schemas are required.

This chapter has presented concepts for capturing the temporal properties of attributes. These concepts include temporal and strong temporal functional dependencies and time-invariant keys. Also included are surrogates that represent the real-world objects described by the attributes, lifespans of attributes, observation and update patterns for time-varying attributes, and derivation functions that compute

Table 7.1 Overview of Temporal ER Models

Name	Main references	Based on
Temporal Entity-relationship Model	(24, 25)	ER
Relationships, Attributes, Keys, and Entities Model	(12)	ER
Model for Objects with Temporal Attributes and Relationships	(32)	ER & OO
Temporal EER model	(10, 11)	EER
Semantic Temporal EER model	(8, 9)	ER
Entity-Relation-Time model	(42, 43, 29)	ER
Temporal ER model	(41)	ER
Temporal EER model	(27)	EER
Kraft's Model	(26)	ER
TERC+	(48)	ERC+
TimeER	(16)	EER

new attribute values from stored ones. We subsequently showed the important roles these concepts play during database design. We were able to formulate four additional decomposition guidelines that supplement normal-form-based decomposition.

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

Finally, the ideas presented here and the methodology that will follow should be transitioned to existing implementation platforms, including non-temporal query languages such as SQL-92 (30). In the short and perhaps even medium term, it is unrealistic to assume that applications will be designed using a temporal data model, implemented using novel temporal query languages, and run on as yet nonexistent temporal DBMSs.

References

1. P. Atzeni and V. De Antonellis. *Relational Database Theory*. Benjamin/Cummings (1993).

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

3. M. H. Böhlen. Valid Time Integrity Constraints. Technical Report 94-30, Department of Computer Science, University of Arizona, Tucson, AZ (1994).

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

5. J. Clifford and A. U. Tansel. On an Algebra for Historical Relational Databases: Two Views. In S. Navathe, editor, *ACM SIGMOD International Conference on the Management of Data*, pp. 247–265 (1985).

6. 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*, pp. 528–537 (1987).

7. C. E. Dyreson and R. T. Snodgrass. Supporting Valid-time Indeterminacy. *ACM Transactions on Database Systems*, **23**(1), to appear (1998).

8. R. Elmasri, I. El-Assal, and V. Kouramajian. Semantics of Temporal Data in an Extended ER Model. In *Ninth International Conference on the Entity-Relationship Approach*, pp. 239–254 (1990).

9. R. Elmasri and V. Kouramajian. A Temporal Query Language for a Conceptual Model. In N. R. Adam and B. K. Bhargava, editors, *Advanced Database Systems*, Volume 759 of *Lecture Notes in Computer Science*, pp. 175–195, Springer-Verlag (1993).

10. R. Elmasri and G. T. J. Wuu. A Temporal Model and Query Language for ER databases. In *Proceedings of the Sixth International Conference on Data Engineering*, pp. 76–83 (1990).

11. R. Elmasri, G. T. J. Wuu, and V. Kouramajian. *A Temporal Model and Query Language for EER Databases*, Chapter 9, pp. 212–229. In A. Tansel et al., editors, *Temporal Databases*, Benjamin/Cummings (1993).

12. S. Ferg. Modeling the Time Dimension in an Entity-Relationship Diagram. In *Fourth International Conference on the Entity-Relationship Approach*, pp. 280–286 (1985).

13. S. K. Gadia and J. H. Vaishnav. A Query Language for a Homogeneous Temporal Database. In *ACM SIGAct-SIGMOD Principles on Database Systems*, pp. 51–56 (1985).

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

15. H. Gregersen and C. S. Jensen. Temporal Entity-Relationship Models—a Survey. *IEEE Transactions on Knowledge and Data Engineering*, to appear (1999).

16. H. Gregersen and C. S. Jensen. Conceptual Modeling of Time-Varying Information. TimeCenter TR-35, Department of Computer Science, Aalborg University (1998).

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

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

19. C. S. Jensen and C. E. Dyreson, editors. A Consensus Glossary of Temporal Database Concepts—February 1998 Version. In O. Etzion, S. Jajodia, and S. Sripada, editors, *Temporal Databases: Research and Practice*, pp. 367–405, LNCS 1399, Springer-Verlag (1998).

20. C. S. Jensen and R. T. Snodgrass. The Surrogate Data Type. Chapter 9, pp. 153–156. In (39).

21. C. S. Jensen and R. T. Snodgrass. Semantics of Time-Varying Attributes and Their Use for Temporal Database Design. In *Fourteenth International Conference on Object-Oriented and Entity Relationship Modeling*, pp. 366–377 (1995).

22. C. S. Jensen, R. T. Snodgrass, and M. D. Soo. Extending Existing Dependency Theory to Temporal Databases. *IEEE Transaction on Knowledge and Data Engineering*, **8**(4):563–582 (1996).

23. C. S. Jensen and R. T. Snodgrass. Semantics of Time-Varying Information. *Information Systems*, **21**(4):311–352 (1996).

24. M. R. Klopprogge. TERM: An Approach to Include the Time Dimension in the Entity-Relationship Model. In *Proceedings of the Second International Conference on the Entity Relationship Approach*, pp. 477–512 (1981).

25. M. R. Klopprogge and P. C. Lockemann. Modelling Information Preserving Databases: Consequences of the Concept of Time. In *International Conference on Very Large Databases*, pp. 399–416 (1983).

26. P. Kraft. Temporal Qualities in ER Models. How? (in Danish) Working paper 93, Department of Information Science, The Aarhus School of Business (1996).

27. V. S. Lai, J-P. Kuilboer, and J. L. Guynes. Temporal Databases: Model Design and Commercialization Prospects. *DATA BASE*, **25**(3):6–18 (1994).

28. N. A. Lorentzos. Management of Intervals and Temporal Data in the Relational Model. TR 49, Agricultural University of Athens (1991).

29. P. McBrien, A. H. Seltveit, and B. Wangler. An Entity-Relationship Model Extended to Describe Historical Information. In *International Conference on Information Systems and Management of Data*, pp. 244–260 (1992).

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

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

32. A. Narasimhalu. A Data Model for Object-Oriented Databases With Temporal Attributes and Relationships. Technical report, National University of Singapore (1988).

33. G. Özsoyoğlu and R. T. Snodgrass. Temporal and Real-Time Databases: A Survey. *IEEE Transactions on Knowledge and Data Engineering*, **7**(4):513–532 (1995).

34. A. Segev and A. Shoshani. The Representation of a Temporal Data Model in the Relational Environment. In *Proceeding of the Fourth International Working Conference on Statistical and Scientific Database Management*, Volume 339 of *Lecture Notes in Computer Science*, pp. 39–61, Springer-Verlag (1989).

35. A. Segev and A. Shoshani. A Temporal Data Model Based on Time Sequences, Chapter 11, pp. 248–270. In A. Tansel et al., editors, *Temporal Databases*, Benjamin/Cummings (1993).

36. M. D. Soo, C. S. Jensen, and R. T. Snodgrass. An Algebra for TSQL2, Chapter 27, pp. 505–546. In (39).

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

38. R. T. Snodgrass. Temporal Object Oriented Databases: A Critical Comparison, Chapter 19, pp. 386–408. In *Modern Database Systems: The Object Model, Interoperability and Beyond*, W. Kim, editor, Addison-Wesley/ACM Press (1995).

39. R. T. Snodgrass (editor), I. Ahn, G. Ariav, D. Batory, J. Clifford, C. E. Dyreson, R. Elmasri, F. Grandi, C. S. Jensen, W. Käfer, N. Kline, K. Kulkarni, T. Y. C. Leung, N. Lorentzos, J. F. Roddick, A. Segev, M. D. Soo, and S. M. Sripada. *The Temporal Query Language TSQL2*. Kluwer Academic Publishers (1995).

40. A. U. Tansel and L. Garnett. Nested Historical Relations. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, pp. 284–293 (1989).

41. B. Tauzovich. Toward Temporal Extensions to the Entity-Relationship Model. In *The Tenth International Conference on the Entity Relationship Approach*, pp. 163–179 (1991).

42. C. I. Theodoulidis, B. Wangler, and P. Loucopoulos. The Entity Relationship Time Model. In *Conceptual Modelling, Databases, and CASE: An Integrated View of Information Systems Development*, pp. 87–115, Wiley (1992).

43. C. I. Theodoulidis, P. Loucopoulos, and B. Wangler. A Conceptual Modelling Formalism for Temporal Database Applications. *Information Systems*, **16**(4):401–416 (1991).

44. V. Vianu. Dynamic Functional Dependencies and Database Aging. *Journal of the ACM*, **34**(1):28–59 (1987).

45. X. Wang, C. Bettini, A. Brodsky, and S. Jajodia. Logical Design for Temporal Databases with Multiple Granularities. *ACM Transactions on Database Systems*, **22**(2):115–170 (1997).

46. J. Wijsen. Design of Temporal Relational Databases Based on Dynamic and Temporal Functional Dependencies. In J. Clifford and A. Tuzhilin, editors, *Recent Advances in Temporal Databases*, pp. 61–76, Zurich, Switzerland (1995).

47. J. Wijsen. Temporal FDs on Complex Objects. *ACM Transactions on Database Systems*, **23**(4), to appear (1998).

48. E. Zimanyi, C. Parent, S. Spaccapietra, and A. Pirotte. TERC+: A Temporal Conceptual Model. In *Proceedings of the International Symposium on Digital Media Information Base* (1997).