Chapter 19

Time in Database Systems

Jan Chomicki
Department of Comp. Sci. and Eng.
University at Buffalo, U.S.A.
chomicki@cse.buffalo.edu

David Toman
School of Computer Science
University of Waterloo, Canada
david@uwaterloo.ca

19.1 Introduction

Time is ubiquitous in information systems. Almost every enterprise faces the problem of its data becoming out of date. However, such data is often valuable, so it should be archived and some means of accessing it should be provided. Also, some data may be inherently historical, e.g., medical, cadastral, or judicial records. Temporal databases provide a uniform and systematic way of dealing with historical data.

This chapter develops *point-based* data models and query languages for temporal databases in the relational framework. The models provide a separation between the conceptual data (*what* is stored in the database) and the way the data is compactly represented in the temporal relations (*how* it is stored). This approach leads to a clean and elegant data model while still providing an efficient implementation path. The foundations of the approach can be traced to the *constraint database* technology [Kanellakis et al., 1995]: constraint representation is used as the basis for a space-efficient representation of temporal relations.

We first study how logics of time can be used as *query* and *integrity constraint* languages in the above setting and the differences resulting from choosing a particular logic as a query language for temporal data. Consequently, model-theoretic notions, particularly *formula satisfaction in a fixed model*, are of primary interest. This is in sharp contrast with most major application areas of temporal reasoning, where the major issues are *satisfiability* and *validity*. For this reason, the formalisms studied are usually propositional which is insufficient in the database setting. However, decidable fragments of the logics underlying temporal queries have been studied for the purposes of *schema* design and reasoning about integrity constraints.

While considerable effort has been expended on the development of temporal databases and query languages, there is still no universal consensus on how *temporal features* should be added to the standard relational model. On the surface, there appear to be many candidates for an acceptable temporal data model and query language, e.g., TQuel [Snodgrass, 1987] or TSQL2 [Snodgrass, 1995], or one of TSQL2's variants, such as ATSQL [Snodgrass et al., 1995], SQL/Temporal [Snodgrass et al., 1996] (the latest temporal extension of SQL3 proposed to the ISO and ANSI standardization committees). However, none of them has been adopted as the standard language of temporal databases in practice, and none has established the theoretical foundations for management of time-dependent data. This is in stark contrast with the relational model, where the relational calculus (first-order logic) has became the consensus language. In part, the reason for the limited acceptance of earlier temporal models, and their negligible contribution to the development of practical applications, is an extremely (and often unnecessarily) complex syntax *without* comprehensive theoretical foundations.

This chapter provides a formal foundation for temporal data models and query languages based on logics that have been developed over the last ten years [Chomicki, 1994; Chomicki and Toman, 1998; Toman, 1996; Toman, 1997; Toman and Niwinski, 1996; Toman, 2003c]. In our simple *point-based* approach to managing temporal data, temporal attributes naturally range over individual points in time. This approach can serve as an alternative foundation for existing temporal data models and shows that all *well-founded* queries definable in the former approaches can be equivalently and conveniently formulated using a point-based temporal query language. Moreover, the chapter introduces techniques for compact encoding of temporal data and efficient query evaluation procedures with computational properties comparable to standard relational queries.

The chapter is organized as follows: The first part focuses on temporal data models and query languages. Section 19.2 introduces the necessary notions of time ontology and time domain used in the rest of the chapter. Section 19.3 shows several ways to introduce time into the standard relational model and defines the fundamental notions of *temporal databases*. It also shows how such databases naturally arise as *histories* of ordinary relational databases. Section 19.4 discusses issues connected with database design and temporal integrity constraints. Section 19.5 introduces several *query languages* for temporal databases. Section 19.6 describes techniques needed for efficient *query evaluation* over compact representations of temporal databases. Section 19.7 discusses various temporal extensions of SQL, the standard query language of relational databases, that have been proposed over the past 25 years in the framework of abstract and concrete temporal databases and query languages. Section 19.8 outlines issues related to updating temporal databases. The second part of the chapter, Sections 19.9, 19.10, and 19.11, focus on the *limitations* of simple linearly-ordered, first-order temporal data models and queries evalueted in a single model (or, equivalently, under the closed world assumption) and on different ways of overcoming these limitations: Section 19.9 discusses more complex models of time, Section 19.10 discusses *non-first-order* extensions of temporal query languages, and Section 19.11 considers the implications connected with relaxing the closed world assumption. Section 19.12 contains brief conclusions.

19.2 Structure of Time

We first introduce a number of fundamental concepts and distinctions that are used throughout the chapter. First, there is a choice of *temporal ontology*. However, and in contrast to rather complex temporal ontologies commonly used for *reasoning* about time, we use a very simple notion of time in this chapter—a linear ordering of time instants.

Definition 19.1 (Temporal Domain) A single-dimensional linearly ordered temporal domain is a structure $\mathrm{T}_P = (T, <)$, where T is a set of time instants and $<$ is a linear order on T.

The subscript in T_P underlines that this is indeed a domain of time *points* and distinguishes it from the domain of *intervals*, T_I, introduced in Section 19.6.1. In addition to linear ordering, we may consider whether time is discrete or dense and whether it is bounded or unbounded. These choices are orthogonal to the development of this chapter and the majority of the results continue to hold independent of the above choices.

While considering only linear order may seem limiting at first, we shall see that, since the temporal data models and the associated temporal query languages discussed in this chapter are considerably more powerful than those used for *reasoning* about time, we can *model* most of the additional structure often associated with a time ontology in a uniform framework of temporal databases. For example, the question of whether time is single-dimensional or multi-dimensional (i.e., whether truth of facts is associated with a single time instant or with multiple instants) will be a property of the temporal data model rather than of the time ontology. Note that multiple time dimensions can occur naturally if, for example, multiple kinds of time (e.g., transaction time vs. valid time [Snodgrass and Ahn, 1986]) are required in an application. Similarly, other extensions of the simple model time, such as temporal durations, calendars, etc., and their representation in our framework are discussed in Section 19.9.

Finally, there is a choice of *linear* vs. *nonlinear* time, i.e., whether time should be viewed as a single line or rather as a tree [Emerson, 1990; Hodkinson et al., 2002], or even an acyclic graph [Wolper, 1989]. Although the branching-time view is potentially applicable to some database problems like version control or workflows, there has been very little work in this area. Therefore, in this chapter we concentrate on temporal domains that are linearly ordered sets.

19.3 Abstract Data Models and Temporal Databases

It is useful to introduce a distinction between the *abstract*, representation-independent meaning of a temporal database and its *concrete*, finite representation. This section focuses on the abstract databases while Section 19.6 will explore the concrete ones.

A standard relational database is a *first-order structure* built from a *data domain* D, usually equipped with a built-in equality (diagonal) relation. This domain is extended to a *relational database* by adding to it a finite instance $(\mathbf{r}_1, \ldots, \mathbf{r}_k)$ of a user-defined relational database schema $\rho = (r_1, \ldots, r_k)$ over D. Intuitively, a database (instance) D *believes* that a fact $r_i(a_1, \ldots, a_k)$ is *true* whenever the elements $a_1, \ldots, a_k$ are r_i-related (i.e, $a_1, \ldots, a_k \in \mathbf{r}_i^{\mathrm{D}}$) in the instance D and false otherwise. This is equivalent to the *closed world assumption* (CWA).

19.3.1 *1NF Models*

First we consider temporal data models that associate truth of facts with *individual time instants*. This, in database terminology, is equivalent to the *first normal form* requirement [Codd, 1971]: the requirement that relations only relate atomic values. Note that while this requirement may not be fully met by some of the temporal models below at the *syntactic level*, all the models are equivalent to (or subsumed by) such a model.

One obtains an abstract temporal database by linking a standard relational database with a temporal domain. There are several alternative ways of achieving this [Chomicki, 1994] that we discuss next.

19.3.1.1 *The Timestamp Model*

This is defined by augmenting all tuples in relations with an additional temporal attribute.

Definition 19.2 (Abstract Timestamp Temporal Database) A relational symbol R_i is a *temporal extension* of the symbol r_i if it contains all attributes of r_i and a single additional attribute t of sort T_P (w.l.o.g. we assume it is the first attribute). The sort of the attributes of R_i is $\mathrm{T}_P \times \mathrm{D}^{\mathrm{arity}(r_i)}$.

Booking		
Meeting	*Room*	*Time*
DB Group	DC1331	06-Jan-04.10:00
DB Group	DC1331	06-Jan-04.10:01
DB Group	DC1331	06-Jan-04.10:02
...	...	...
DB Group	DC1331	16-Jan-04.11:59
Intro to Databases	MC4042	06-Jan-04.10:00
...	...	...
Intro to Databases	MC4042	06-Jan-04.11:19
Intro to Databases	MC4042	08-Jan-04.10:00
...	...	...
Intro to Databases	MC4042	08-Jan-04.11:19

Fig. 19.1 A Fragment of a Timestamp Instance of the `Booking` relation from Example 19.1.

A *timestamp temporal database* is a first-order structure $\mathrm{D} \cup \mathrm{T}_P \cup \{\mathbf{R}_1 \ldots, \mathbf{R}_k\}$, where $\mathbf{R}_i$ are *temporal relations*— instances of the temporal extensions R_i. In addition we require that the set $\{\mathbf{a} : (t, \mathbf{a}) \in \mathbf{R}_i\}$ be finite for every $t \in \mathrm{T}_P$ and $0 < i \leq k$.

Note that at this point there are no cardinality restrictions imposed on the number of time instants in the instances of abstract temporal relations; we address issues connected with the actual finite representation of these relations in Section 19.6. In the rest of the chapter we use the following example to illustrate various concepts.

Example 19.1 Consider a database recording room bookings for meetings in a university. A relational schema $\texttt{booking}(\mathit{Meeting}, \mathit{Room})$ links meetings to rooms. We assume that rooms are identified by their room numbers and meetings have distinct descriptions (names). Thus our temporal database, assuming the use of the timestamp model, contains a single relational schema with three attributes,

$$\texttt{Booking}(\mathit{Meeting}, \mathit{Room}, \mathit{Time}).$$

A tuple (a, b, t) in an instance of this relation denotes the fact that a meeting a is in a room b at time t. For simplicity in this chapter we assume that time is measured in minutes. An example instance of this schema is shown in Figure 19.1. To distinguish between non-temporal relations and (the derived) timestamp relations we capitalize the name of the later. The granularity of time in our examples is one minute (more on granularities in Section 19.9).

It is important to understand that, e.g., the "DB group" meeting has booked room DC1331 for *every* time instant between 06-Jan-04.10:00 and 06-Jan-04.11:59.

This set of tuples, depending on properties of the time domain, can be infinite (e.g., when dense time domain is considered). There are several things to note about the example: an instance of the `Booking` relation represents *complete information* about meeting schedule; in particular it contains information about meetings that have already finished (e.g., for accounting and evaluation purposes) as well as about meetings scheduled in the future (e.g., to avoid overbooking of rooms). This is necessary, for example, if we want to schedule another meeting in the future, as we need to make sure no other meeting conflicts with it. For this purpose we need to query the database for *empty rooms* at the particular future time we desire and such a query is *only possible* utilizing the closed-world assumption.

Second, we assume that distinct meetings have distinct names. Thus the same meeting (e.g., a class) can meet in several different rooms at different times. Moreover the meeting times may not be continuous (as is common, e.g., for classes). If we wish to distinguish between *instances* of a particular meeting we need to use distinguished names (or an additional attribute).

19.3.1.2 *The Snapshot Model*

The sbstract temporal databases in this model are defined as a mapping of the temporal domain to the class of standard relational databases. This gives a Kripke structure with the temporal domain serving as the accessibility relation.

Definition 19.3 (Abstract Snapshot Temporal Database) A *snapshot temporal database* over D, T_P, and ρ is a map $DB : \mathrm{T}_P \to \mathcal{DB}(\mathrm{D}, \rho)$, where $\mathcal{DB}(\mathrm{D}, \rho)$ is the class of finite relational databases over D and ρ.

It is easy to see that snapshot and timestamp abstract temporal databases are merely different views of the same data and thus can represent the same class of temporal databases. Formally, a snapshot temporal database D corresponds to a timestamp temporal database D' if and only if

$$\forall x_1, \ldots, x_k . \mathbf{r}^{D(t)}(x_1, \ldots, x_k) \iff \mathbf{R}^{D'}(t, x_1, \ldots, x_k),$$

for all r (and R) in the schema of D (D'), where $\mathbf{r}^{D(t)}$ ($\mathbf{R}^{D'}$) are the instances of the relations r (R) in D (D'), respectively, where $k = \mathrm{arity}\, r$. This correspondence allows us to move freely between the two models.

Example 19.2 A snapshot representation of the instance in Figure 19.1 is shown in Figure 19.2.

Note that the relationship between the timestamp and snapshot models is essentially currying and uncurrying [Barendregt, 1984] (the correspondence is exact if the relations are considered to be boolean functions from tuples to the set $\{\mathrm{true}, \mathrm{false}\}$).

Time	booking
06-Jan-04.10:00	{ (DB Group,DC1331), (Intro to Databases,MC4042) }
06-Jan-04.10:01	{ (DB Group,DC1331), (Intro to Databases,MC4042) }
...	
06-Jan-04.11:19	{ (DB Group,DC1331), (Intro to Databases,MC4042) }
06-Jan-04.11:20	{ (DB Group,DC1331) }
...	
06-Jan-04.11:59	{ (DB Group,DC1331) }
06-Jan-04.12:00	{ }
...	
06-Jan-04.12:00	{ }
08-Jan-04.10:00	{ (Intro to Databases,MC4042) }
...	
08-Jan-04.11:19	{ (Intro to Databases,MC4042) }

Fig. 19.2 A Fragment of a Snapshot Instance of the `Booking` relation.

Thus, in the rest of the chapter we use the timestamp abstract temporal databases as the common underlying temporal data model. Also, let us reiterate that the abstract data models are used solely at the *conceptual* level; relations will likely be stored in a different, more space-efficient format, e.g., one that uses time intervals (see Section 19.6).

19.3.1.3 *Relational Database Histories*

Relational databases are updatable and it is natural to consider sequences of database states resulting from the updates.

Definition 19.4 (Finite History) A *history* over a database schema ρ and a data domain D is a sequence $H = (D_0, \ldots, D_n)$ of database instances (called *states*) such that

(1) all the states $D_0, \ldots, D_n$ share the same schema ρ and the same data domain D,
(2) D_0 is the initial instance of the database,
(3) D_i results from applying an update to D_{i-1}, $i \geq 1$,

There is a clear correspondence between histories over D and ρ and snapshot temporal databases over D, **N** (natural numbers), and ρ (see Definitions 19.3 and 19.2). Consequently, any query language for abstract temporal databases can also be used to query database histories. However, there is a difference in the restrictions placed on updates: while there are no a priori limitations placed on snapshot temporal database updates (they can involve any snapshot), histories are *append-only* (the past cannot be modified). This property is often associated with *transaction time* databases—temporal databases in which time instants correspond to *commitment* time of transactions; the *append-only* nature of such databases corresponds to the requirement of *durable transactions*. Indeed, transaction-time temporal databases can be viewed as finite histories of standard relational databases.

19.3.2 *Multiple Temporal Dimensions*

So far we have considered only single-dimensional temporal databases: temporal relations were allowed only a single temporal attribute. To motivate the introduction of multiple temporal dimensions in the context of temporal databases, consider the following examples:

- *Bitemporal* databases: with each tuple in a relation two kinds of time are stored—the valid time (when a particular tuple is true) and the transaction time (when the particular tuple was inserted/deleted in the database) [Jensen et al., 1993].
- *Spatial* databases: multiple dimensions over an interpreted domain can be used for representing *spatial data* where multiple dimensions serve as coordinates of points in a k-dimensional Euclidean space.

Most of the data modeling techniques require only fixed-dimensional data. However, the true need for arbitrarily large dimensionality of data models originates in the requirement of having a first-order complete query language (see Theorem 19.5 in Section 19.5). Thus, there are two cases to consider:

- temporal models with a fixed number of dimensions (> 1), and
- temporal models with a varying number of temporal dimensions without an upper bound.

The representation of multiple temporal dimensions in abstract temporal databases is quite straightforward: we simply index relational databases by the elements of an appropriate self-product of the temporal domain (in the case of snapshot temporal databases), or add the appropriate number of temporal attributes (in the case of timestamp temporal databases).

19.3.3 *Non 1NF Temporal Models*

Several temporal data models associate relationships between data values—truth of facts recorded in the database— with *sets* of time instants (rather than with a single time instant). These models are no longer in first normal form (N1NF) and are often called *temporally grouped* models [Clifford et al., 1993; Clifford et al., 1994].

Example 19.3 The instance of the `Booking` relation from Figure 19.1 represented in the N1NF (temporally grouped) model is as follows

`Booking`		
Meeting	*Room*	*Time*
DB group	DC1331	{06-Jan-04.10:00, 06-Jan-04.10:01, . . . ,06-Jan-04.11:59}
Intro to Databases	MC4042	{06-Jan-04.10:00, . . . ,06-Jan-04.11:19, 08-Jan-04.10:00, . . . ,08-Jan-04.11:19}

However, the set-based attributes can be *flattened*, perhaps by introducing additional surrogate keys, to obtain a 1NF temporal database containing the same information [Clifford et al., 1993; Wijsen, 1999]. Without introducing additional keys, however, this transformation can be lossy.

Example 19.4 Consider a fragment of a N1NF temporal relation

`booking`(DB group, DC1331, {06-Jan-04.10:00, . . . ,06-Jan-04.11:59})
`booking`(DB group, DC1331, {09-Jan-04.10:00, . . . ,09-Jan-04.11:59})

The meetings in this design are no longer identified by their names, but rather by their name and the set of all meeting times. The same information, however, can be captured by explicitly identifying meetings. Also, such an assumption *prevents* us from representing a situation where a particular meeting takes place in two different rooms at two different times.

Note that the difference between 1NF and N1NF models is intrinsic to these models and can be exhibited *without* introducing temporal aspects into the picture. Also, the differences at the level of abstract databases do not necessarily impact the way the relations are actually stored at the concrete (or physical) level; indeed both of the above examples may be simply two different views of the same physical design.

Another salient point is that a common assumption made by various temporal data models when using the N1NF representation is that facts associated with sets of time instants are also implicitly true at all time instants contained in these sets (as in the above example). This, however, may not be the case in general, as demonstrated by the following example.

Example 19.5 First consider the following two tuples in an instance of a N1NF temporal database:

`Booking`(DB group, DC1331, [06-Jan-04.10:00, 06-Jan-04.11:59])
`Booking`(AI meeting, MC5114, [06-Jan-04.09:00, 06-Jan-04.10:59])

In this case the sets (represented by intervals in this case) serve as encodings of their internal points: the database group indeed meets in the DC1331 room every time instant between 06-Jan-04.10:00 and 06-Jan-04.11:59; similarly for the AI meeting. Thus, a meaningful question is whether these two meetings conflict, i.e., whether there is a time instant related to both meetings. On the other hand, consider another fragment of a temporal database:

`Electricity`(Jones A., 40, [15-May-03.00.00, 14-Jun-03.23:59])
`Electricity`(Smith J., 35, [01-May-03.00.00, 31-May-03.23:59])

The intervals in this example *do not* represent the collections of their internal points, but rather the names of the sets themselves (or points in a 2-dimensional space). Thus applying set-based operations on these sets, e.g., computing their intersection, does not have a clear meaning.

This example also clarifies the difference between two distinct uses of intervals in temporal databases:

(1) intervals as encodings of the extents of convex 1-dimensional sets, or
(2) intervals as (otherwise uninterpreted) names for such sets.

These two approaches assume completely different *meaning* to be assigned to the same construct (often a pair of time instants) in different contexts. Note that in Sections 19.5 and 19.6 we use solely the first paradigm.

An interesting observation at this point is that the keys introduced in the flattening transformation essentially represent *names* of sets of time instants. This idea, however, can be formalized using a 1NF temporal model, e.g., the timestamp model: we simply add an *abstract relation* that links names of sets of time instants with their extents (essentially the *membership relation*). For example, to describe intervals, the relation would look as follows:

$$\mathsf{SETS}(n,t) := \{('[t_1,t_2]',t) : t_1 \leq t \leq t_2\}$$

Note that $'[t_1,t_2]'$ is now an otherwise uninterpreted element of the *data domain*. Similarly we can introduce constants (as singleton sets), calendars, etc. It has to be understood that association of other data values with names of sets does not say anything about the truth of facts with respect to the time instants belonging to these sets. Also, the extents of these sets do not have to be closed under set operations, e.g., an intersection or union of the extents of two such sets, while it always exists, may not have a *name**. Similar approach can be used to introduce names for other sets, e.g., singleton sets for constants, periodic sets for time granularities (see Section 19.9), etc.

19.3.3.1 *The Parametric Model*

This model [Clifford et al., 1994] considers the values stored in individual fields of tuples in the database to be functions of time. It is easy to see that every

*This issue resurfaces when one attempts to define an interval-based temporal data model as a restriction of the N1NF model: since unions of intervals are not necessarily describable as intervals the notion of temporal elements is needed to maintain closure under boolean operations.

instance of a relation r represented in a parametric temporal database D can be represented in the timestamp model as an instance D' as follows:

$$\mathbf{R}^{D'} = \{(t, f_1(t), \ldots, f_k(t)) : (f_1, \ldots, f_k) \in \mathbf{r}^D, t \in \mathrm{T}_P\}$$

Note that this transformation loses the *identity* of the tuples [Clifford et al., 1993]. However, introducing tuple identifiers as outlined in the previous section alleviates this deficiency. Wijsen [Wijsen, 1999] also argues that this transformation indeed simplifies further technical development of integrity constraints and queries.

Moreover, if the functions used in the parametric model are *total*, then there are instances of a timestamp database containing a single unary relation, e.g., $\mathbf{R} = \{(0, a), (1, b), (1, c)\}$, that cannot be represented using the parametric model (since the number of tuples at time 0 differs from the number of tuples at time 1). Thus we need to allow partial functions and/or life-span attributes to regain the expressiveness of the simple 1NF model. We do not consider the parametric model in this chapter any further.

19.4 Temporal Database Design

The equivalence between snapshot and timestamp temporal databases (Definitions 19.2 and 19.3) makes it possible to view the design of temporal database schemas as a special case of the design of relational database schemas.

19.4.1 *Temporal Functional Dependencies*

Jensen et al. [Jensen et al., 1996] propose a formal framework for temporal database design that encompasses and generalizes earlier approaches in this area. We provide here a purely relational reconstruction of that framework, eliminating at the same time its inherent technical limitations. We use the timestamp model and assume first a single temporal dimension with temporal domain T_P.

The cornerstone of the approach of [Jensen et al., 1996] is the notion of *temporal functional dependency* (temporal FD). A temporal FD $X \xrightarrow{T} Y$ holds in a snapshot temporal relation DB if the (classical) FD $X \to Y$ holds in every snapshot of DB. Viewing DB as a timestamp database TDB, this is equivalent to the classical FD $X\,T \to Y$ holding in TDB.

Example 19.6 Assume the relation `booking` with attributes *Meeting* and *Room* from Example 19.1. The temporal FD $Meeting \xrightarrow{T} Room$ expresses the fact that every meeting is held in a single room at any given time. In the corresponding timestamp relation `Booking`, the above condition is captured by the FD $Meeting\ Time \to Room$.

Avoiding the introduction of a new notion of a *temporal FD* has numerous advantages. First, one can use the classical notions of FD inference (Armstrong axioms), dependency closure, keys, normal forms, and lossless decompositions *without any change*. In [Jensen et al., 1996], new notions of temporal keys, temporal normal forms, etc. are derived as temporal versions of their relational analogues. Second, it is no longer necessary to restrict temporal relations to being *finite* (as in [Jensen et al., 1996]) in order to test satisfaction of temporal FDs. It is enough for such relations to be *finitely representable* (in the sense of the constraint databases [Kanellakis et al., 1995]). Every classical FD can be written as a first-order sentence and evaluated as a relational calculus query over any finitely representable relation. Third, one can now mix temporal and non-temporal FDs.

Example 19.7 The dependency $Meeting \rightarrow Room$ in the timestamp relation `Booking` is non-temporal and expresses the property that for every specific meeting the same room is always booked.

With multiple temporal dimensions, the advantages of the relational framework are even more pronounced. For concreteness, we assume two such dimensions: *valid time (VT)* and *transaction time (TT)*. Timestamp relations will now have zero, one (TT or VT), or two ($TT\ VT$) temporal attributes. Now we can have, in addition to non-temporal FDs, three kinds of temporal FDs formulated as classical FDs: transaction-time ($X\ TT \rightarrow Y$), valid-time ($X\ VT \rightarrow Y$), and bitemporal ($X\ TT\ VT \rightarrow Y$).

Example 19.8 The bitemporal dependency captured by the FD

$$Meeting\ TT\ VT \rightarrow Room$$

expresses the constraint that *the record at any time of the room booked for a meeting at any time is uniquely determined*. This is a very weak constraint. If we want to say that *the room booked for a meeting at any time is uniquely determined*, we need to use the FD $Meeting\ VT \rightarrow Room$ which captures a valid-time dependency.

Jensen et al. [Jensen et al., 1996] considered the presence of two temporal dimensions but didn't analyze the consequences of this fact for FDs and other concepts of database design. There are essentially two choices. The first is to limit the attention to bitemporal dependencies. But then valid- and transaction-time FDs become inexpressible, and as a consequence one will not be able to define relational normal forms that truly capture all kinds of FD-related temporal redundancies. For example, the FD $Meeting\ VT \rightarrow Room$ (Example 19.8) identifies a potential redundancy, which should be removed during the database design process.

The second choice is to allow three kinds of temporal FDs: $X \stackrel{TT}{\rightarrow} Y$, $X \stackrel{VT}{\rightarrow} Y$, and $X \stackrel{TT\ VT}{\longrightarrow} Y$. But then one can no longer talk about, e.g., temporal keys, but only about valid-time, transaction-time or bitemporal keys. The framework becomes so complicated that it is unlikely to be of any use.

The relational framework does not suffer from any of those problems. The classical notion of FD is fine enough to capture all the varieties of temporal dependencies. At the same time, the framework does not require any conceptual extensions.

We should mention that temporal functional dependencies have been generalized to multiple temporal granularities [Wang et al., 1997] and to the object-oriented setting [Wijsen, 1999].

19.4.2 *Constraint-generating Dependencies*

If we consider the first-order formulation of temporal functional dependencies in timestamp databases, we notice that the formulas obtained in this way contain *equalities* between temporal variables. It is natural to consider a generalization of such dependencies that allows not only equalities but also arbitrary *constraints* over the given temporal domain. Then we can formulate integrity constraints like "the transaction time of a given tuple should always be greater than or equal to the valid time of this tuple." Note that the constraints over the temporal domain are not used here to represent infinite sets (as in constraint databases [Kanellakis et al., 1995]) but rather to obtain a more expressive language of integrity constraints.

The above idea was first formulated in [Ginsburg and Hull, 1983; Ginsburg and Hull, 1986] and then formalized in [Baudinet et al., 1999] using the notion of a *constraint-generating dependency* (CGD). Baudinet et al. [Baudinet et al., 1999] described a general reduction of the implication problem for such dependencies to the problem of validity of universal formulas in the appropriate constraint theory. Complexity results for restricted classes of CGDs were also given. A similar idea was studied in the temporal database context in [Wijsen, 1998].

19.5 Abstract Temporal Queries

Most logic-based query languages have their semantics defined in terms of abstract temporal databases—they will be termed *abstract* as well. Other languages whose semantics is defined in terms of *concrete databases* will be appropriately called *concrete*. Here we discuss abstract databases and query languages—the concrete ones are discussed in Section 19.6.

Since databases are inherently first-order structures, in this chapter we are primarily interested in temporal extensions of *first-order* logic (relational calculus).

A natural first-order query language over such databases—the relational calculus—coincides with *first-order logic* over the vocabulary $(=, r_1, \ldots, r_k)$ of the extended structure. An answer to a query in relational calculus is the set of valuations (tuples) that make the query true in the given relational database. Domain independent relational calculus queries (those that depend only on the instance of ρ and not on the underlying domain of values D) can be equivalently expressed in *relational algebra* [Codd, 1972]. In this way the relational model provides both a natural declarative paradigm for representing and querying information stored in a relational database and the possibility of efficient implementation of queries through relational algebra.

Following are several temporal queries we may ask over our sample temporal database.

- *find all meetings that always meet in the same room.*
- *find all rooms in which the last meeting was 'DB group'.*
- *find all meetings with a scheduled break (or multi-part meetings, such as classes).*

We discuss two major approaches to introducing time into relational query languages. Both of them are developed in the context of *abstract* temporal databases and thus lead to abstract query languages. The first approach uses modal temporal connectives and implicit temporal contexts; the second adds explicit variables (attributes) and quantifiers over the temporal domain. We report on the relative expressive power of these extensions.

The two different ways of linking time with a relational database (Definitions 19.3 and 19.2) lead to two different temporal extensions of the relational calculus (first-order logic). The snapshot model gives rise to *temporal connectives*, while the timestamp model introduces explicit attributes and quantifiers for handling time. The first approach is appealing because it *encapsulates* all the interaction with the temporal domain inside the temporal connectives. In this way the manipulation of the temporal dimension is completely hidden from the user, as it is performed on implicit temporal attributes.

19.5.1 *First-order Temporal Logic*

Historically, many different variants of temporal logic based on different sets of connectives have been developed [Gabbay et al., 1994]. Some connectives, such as $\diamond$ (*"sometime in the future"*), $\Box$ (*"always in the future"*), or **until** are well-known and have been universally accepted. But in general any appropriate first-order formula in the language of the temporal domain (or, as we will see in Section 19.10, even a second-order one) can be used to define a temporal connective.

Definition 19.5 (First-order Temporal Connectives) Let

$$O ::= t_i < t_j \mid O \wedge O \mid \neg O \mid \exists t_i.O \mid X_i$$

be the first-order language of T_P extended with the propositional variables X_i. We define a *(k-ary) temporal connective* to be an O-formula with exactly one free variable t_0 and k free propositional variables $X_1, \ldots, X_k$. We assume that t_i is the only temporal variable free in the formula to be substituted for X_i.

We define Ω to be a finite set of definitions of temporal connectives: pairs of names $\omega(X_1, \ldots, X_k)$ and (definitional) O-formulas ω^* for temporal connectives.

We call the variables t_i the *temporal contexts*: t_0 defines the outer temporal context of the connective that is made available to the surrounding formula; the variables $t_1, \ldots, t_k$ define the temporal contexts for the subformulas substituted for the propositional variables $X_1, \ldots, X_k$.

The above definition allows only *first-order* temporal connectives. This is sufficient to define the common temporal connectives **since**, **until**, and their derivatives.

Example 19.9 The common temporal connectives are defined as follows:

$$X_1 \,\mathbf{until}\, X_2 \triangleq \exists t_2.t_0 < t_2 \wedge X_2 \wedge \forall t_1(t_0 < t_1 < t_2 \rightarrow X_1)$$
$$X_1 \,\mathbf{since}\, X_2 \triangleq \exists t_2.t_0 > t_2 \wedge X_2 \wedge \forall t_1(t_0 > t_1 > t_2 \rightarrow X_1)$$

Other commonly used temporal connectives, *sometime in the future*, $\Diamond$, *always in the future*, $\Box$, *sometime in the past*, $\blacklozenge$, and *always in the past*, $\blacksquare$, are defined in terms of **since** and **until** as follows:

$$\Diamond X_1 \triangleq \text{true } \mathbf{until}\, X_1 \qquad \Box X_1 \triangleq \neg\Diamond\neg X_1$$
$$\blacklozenge X_1 \triangleq \text{true } \mathbf{since}\, X_1 \qquad \blacksquare X_1 \triangleq \neg\Diamond\neg X_1$$

For a discrete linear order we also define the $\bigcirc$ (next) and $\bullet$ (previous) operators as

$$\bigcirc X_1 \triangleq \exists t_1.t_1 = t_0 + 1 \wedge X_1 \qquad \bullet X_1 \triangleq \exists t_1.t_1 + 1 = t_0 \wedge X_1$$

Clearly, all of the above connectives are definable in the first-order language of T_P (the successor $+1$ and the equality $=$ on the domain T_P are first-order definable in the theory of discrete linear order).

The connectives **since**, $\blacklozenge$, $\blacksquare$, and $\bullet$ are called the *past temporal connectives* (as they refer to the past) and **until**, $\Diamond$, $\Box$, and $\bigcirc$ the *future temporal connectives*.

We discuss the use of more expressive language in the definition of temporal connectives, e.g., monadic second-order logic over the signature of T_P, to define a richer class of temporal connectives in Section 19.10.

The modal query language—first-order temporal logic—is defined to be the original single-sorted first-order logic (relational calculus) extended with a finite set of temporal connectives.

Definition 19.6 (First-order Temporal Logic: syntax) Let Ω be a *finite* set of (names of) temporal connectives. *First-order Temporal Logic (FOTL)* L^{Ω} over a schema ρ is defined as:

$$F ::= r(x_{i_1}, \ldots, x_{i_k}) \mid x_i = x_j \mid F \wedge F \mid \neg F \mid \omega(F_1, \ldots, F_k) \mid \exists x.F$$

where $r \in \rho$ and $\omega \in \Omega$.

A standard linear-time temporal logic can be obtained from this definition using the temporal connectives from Example 19.9:

Example 19.10 The standard FOTL language $L^{\{\mathbf{since},\mathbf{until}\}}$ is defined as

$$F ::= r(x_{i_1}, \ldots, x_{i_k}) \mid x_i = x_j \mid F \wedge F \mid \neg F \mid F_1 \textbf{ since } F_2 \mid F_1 \textbf{ until } F_2 \mid \exists x.F$$

where **since** and **until** are the names for the connectives defined in Example 19.9.

Example 19.11 We show here how various temporal connectives are used to formulate the queries over the temporal database introduced in Example 19.1.

- *find all meetings that always meet in the same room.*

$$\blacklozenge\Diamond\exists y(\texttt{booking}(x, y) \wedge \blacksquare\Box\forall z(\texttt{booking}(x, z) \implies y = z))$$

- *find all rooms in which the last meeting was 'DB group'.*

$$(\neg\exists y.\texttt{booking}(y, x)) \textbf{ since } \texttt{booking}(\texttt{DB group}, x)$$

  Note that this query returns all time instants at which the above statement is true for room x.
- *find all meetings with a scheduled break.*

$$\blacklozenge\exists y.\texttt{booking}(x, y) \wedge \neg\exists y.\texttt{booking}(x, y) \wedge \Diamond\exists y.\texttt{booking}(x, y).$$

The standard way of giving semantics to such a language is as follows.

Definition 19.7 (FOTL: semantics) Let DB be a snapshot temporal database over D, T_P, and ρ, φ a formula of L^{Ω}, $t \in \mathrm{T}_P$, and θ a valuation. We define a relation $DB, \theta, t \models \varphi$ by induction on the structure of φ:

$$\begin{array}{ll}
DB, \theta, t \models r_j(x_{i_1}, \ldots, x_{i_k}) & \text{if } r_j \in \rho, (\theta(x_{i_1}), \ldots, \theta(x_{i_k})) \in \mathbf{r}_j^{DB(t)} \\
DB, \theta, t \models x_i = x_j & \text{if } \theta(x_i) = \theta(x_j) \\
DB, \theta, t \models \varphi \wedge \psi & \text{if } DB, \theta, t \models \varphi \text{ and } DB, \theta, t \models \psi \\
DB, \theta, t \models \neg\varphi & \text{if not } DB, \theta, t \models \varphi \\
DB, \theta, t \models \exists x_i.\varphi & \text{if there is } a \in D \text{ such that } DB, \theta[x_i \mapsto a], t \models \varphi \\
DB, \theta, t \models \omega(F_1, \ldots, F_k) & \text{if } \mathrm{T}_P, [t_0 \mapsto t] \models \omega^* \text{ where} \\
 & \quad \mathrm{T}_P, \delta \models X_i \text{ is interpreted as } DB, \theta, \delta(t_i) \models F_i
\end{array}$$

where $\mathbf{r}_i^{DB(t)}$ is the interpretation of the predicate symbol r_i in DB at time t. We assume the rigid interpretation of constants (they do not change over time). The *answer to a query φ over DB* is the set of tuples $\varphi(DB) := \{(t, \theta_{|FV(\varphi)}) : DB, \theta, t \models \varphi\}$ where $\theta_{|FV(\varphi)}$ is the restriction of θ to the free variables of φ.

Example 19.12 The above definition can be applied to the standard language $L^{\{\mathbf{since},\mathbf{until}\}}$ for which it gives the usual semantics of the **since** and **until** connectives:

$$D, \theta, t_0 \models \varphi \textbf{ until } \psi \text{ if } \exists t_2.t_2 > t_0 \wedge D, \theta, t_2 \models \psi \wedge \forall t_1.t_2 > t_1 > t_0 \rightarrow D, \theta, t_1 \models \varphi.$$

There are even more restricted versions of FOTL. Gabbay, et al. [Gabbay et al., 1994] introduce first-order temporal logics where the temporal connectives are always outside of the first-order quantifiers. While such logics may provide sufficient expressive power for some applications, they are generally weaker than L^Ω (for the same set of temporal connectives Ω).

19.5.2 *Two-sorted First-order Logic*

The second natural extension of the relational calculus to a temporal query language is based on explicit variables and quantification over the temporal domain T_P. It is just the two-sorted version (variables are temporal or non-temporal) of first-order logic (2-FOL) over D and T_P, with the limitation that the predicates can have only one temporal argument [Bacchus et al., 1991].

Definition 19.8 (2-FOL: syntax) The two-sorted first-order language L^P over a database schema ρ is defined by:

$$M ::= R(t_i, x_{i_1}, \ldots, x_{i_k}) \mid t_i < t_j \mid x_i = x_j \mid M \wedge M \mid \neg M \mid \exists x_i.M \mid \exists t_i.M$$

where R is the temporal extension of r for $r \in \rho$. We use t_i to denote temporal variables and x_i to denote data (non-temporal) variables.

Similarly to FOTL we can use 2-FOL to formulate temporal queries:

Example 19.13 The query *find all meetings with a scheduled break* can be formulated in 2-FOL using the following formula:

$$\exists t_1, t_2 . t_1 < t < t_2 \wedge \exists y.\texttt{Booking}(x, y, t_1) \wedge \neg\exists y.\texttt{Booking}(x, y, t) \\ \wedge \exists y.\texttt{Booking}(x, y, t_2).$$

Note that, similarly to the FOTL query in Example 19.11, the query returns names of meetings with a break together with the time of the break; should we require the names alone we would need to use an additional existential quantifier for t.

The semantics for this language is defined in the standard way, similarly to the semantics of relational calculus [Abiteboul et al., 1995].

Definition 19.9 (2-FOL: semantics) Let DB be a timestamp temporal database over D, T_P, and ρ, φ a formula in L^P, and θ a two-sorted valuation. We define the satisfaction relation $DB, \theta \models \varphi$ as follows:

$$\begin{array}{ll}
DB, \theta \models R_j(t_i, x_{i_1}, \ldots, x_{i_k}) & \text{if } R_j \in \rho, (\theta(t_i), \theta(x_{i_1}), \ldots, \theta(x_{i_k})) \in \mathbf{R}_j^{DB} \\
DB, \theta \models t_i < t_j & \text{if } \theta(t_i) < \theta(t_j) \\
DB, \theta \models x_i = x_j & \text{if } \theta(x_i) = \theta(x_j) \\
DB, \theta \models \varphi \wedge \psi & \text{if } DB, \theta \models \varphi \text{ and } DB, \theta \models \psi \\
DB, \theta \models \neg\varphi & \text{if not } DB, \theta \models \varphi \\
DB, \theta \models \exists t_i.\varphi & \text{if there is } s \in \mathrm{T}_P \text{ such that } DB, \theta[t_i \mapsto s] \models \varphi \\
DB, \theta \models \exists x_i.\varphi & \text{if there is } a \in \mathrm{D} \text{ such that } DB, \theta[x_i \mapsto a] \models \varphi
\end{array}$$

where $\mathbf{R}_j^{DB}$ is the interpretation of the predicate symbol R_j in the database DB.

An L^P query is an L^P formula with exactly one free temporal variable.

An *answer to an L^P query φ over DB* is the set $\varphi(DB) := \{\theta_{|FV(\varphi)} : DB, \theta \models \varphi\}$ where $\theta_{|FV(\varphi)}$ is the restriction of the valuation θ to free variables of φ.

The restriction to a single temporal attribute in the signature of queries guarantees closure over the universe of single-dimensional temporal relations. Note that this restriction applies only to queries, not to subformulas of queries.

19.5.2.1 *Expressive Power*

In the remainder of this section we compare the expressive power of FOTL and 2-FOL. First we define a mapping $\mathrm{Embed} : L^\Omega \to L^P$ to show that the L^Ω formulas can be expressed in the L^P language:

Definition 19.10 (Translation) Let Embed be a mapping of L^Ω formulas to

L^P formulas defined as follows:

$$\begin{array}{lcl}
\mathrm{Embed}(r_i(x_1,\ldots,x_{v_i})) & = & R_i(t_0,x_1,\ldots,x_{v_i}) \\
\mathrm{Embed}(x_i = x_j) & = & x_i = x_j \\
\mathrm{Embed}(F_1 \wedge F_2) & = & \mathrm{Embed}(F_1) \wedge \mathrm{Embed}(F_2) \\
\mathrm{Embed}(\neg F) & = & \neg\,\mathrm{Embed}(F) \\
\mathrm{Embed}(\exists x.F) & = & \exists x.\,\mathrm{Embed}(F) \\
\mathrm{Embed}(\omega(F_1,\ldots,F_k)) & = & \omega^*(\mathrm{Embed}(F_1)[t_0/t_1],\ldots,\mathrm{Embed}(F_k)[t_0/t_k])
\end{array}$$

where $\omega(X_1,\ldots,X_k)$ is the name of ω^* in Ω and $F[t_0/t_i]$ is a substitution of t_i for t_0 in F.

We know that we can freely move between snapshot and timestamp representations (see Definitions 19.3 and 19.2). Definition 19.10 allows us to translate queries in L^Ω to queries in L^P while preserving their semantics.

Theorem 19.1 *Let D_1 be a snapshot temporal database and D_2 an equivalent timestamp database. Then $D_1,\theta,s \models \varphi \iff D_2,\theta[t_0 \mapsto s] \models \mathrm{Embed}(\varphi)$ for all $\varphi \in L^\Omega$.*

Therefore Definition 19.10 can also be used to define the semantics of L^Ω queries over timestamp temporal databases. Also, it shows that L^P is at least as expressive as L^Ω (denoted by $L^\Omega \sqsubseteq L^P$). What is the relationship in the other direction? While both snapshot and timestamp temporal models are equivalent in their ability to represent temporal databases equivalently, the derived query languages differ in expressive power[†]. The separation results are as follows:

Theorem 19.2 ([Kamp, 1971]) *$L^{\{\mathbf{since},\mathbf{until}\}} \sqsubset L^{\{\mathbf{since},\mathbf{until},\mathbf{now}\}} \sqsubseteq L^P$ for dense linearly ordered time ($\sqsubset$ denotes the "strictly weaker than" relationship of languages).*

The proof of this fact uses structures that cannot be modeled as abstract temporal databases because they are infinite in both the data and temporal dimensions. Moreover, the proof technique does not consider arbitrary temporal connectives and discrete linear orders. The following results show that $L^\Omega \sqsubset L^P$ holds in general:

Theorem 19.3 ([Abiteboul et al., 1996]) *$L^{\{\mathbf{since},\mathbf{until}\}} \sqsubset L^P$ over the class of finite timestamp temporal databases.*

Theorem 19.4 ([Toman and Niwinski, 1996; Bidoit et al., 2004; Toman, 2003c]) *$L^\Omega \sqsubset L^P$ over the class of timestamp temporal databases for an arbitrary finite set of first-order temporal connectives Ω.*

[†]This is a major difference from the propositional case where linear-time temporal logic has the same expressive power as the monadic first-order logic over linear orders [Kamp, 1968].

In both cases L^Ω is shown not to be able to express the query "are there two distinct time instants at which a unary relation R contains exactly the same values?" On the other hand, this query can be easily expressed in L^P using the formula

$$\exists t_1, t_2. t_1 < t_2 \wedge \forall x. R(t_1, x) \iff R(t_2, x).$$

This formula can be also expressed in a temporal logic in which connectives are allowed to refer to two temporal contexts simultaneously, $L^{\Omega(2)}$ (see Section 19.5.4).

19.5.3 *Temporal Relational Algebras*

The separation results (Theorems 19.3 and 19.4) have several *unpleasant* consequences. In particular, a single-dimensional first-order complete temporal query language cannot be *subquery closed*. This means that in general we cannot define all queries to be combinations of simpler single-dimensional queries. This fact also prevents us from decomposing large queries into views (virtual relations defined by queries). An even more serious problem is that there is no relational algebra defined over the universe of single-dimensional temporal relations that is able to express all first-order temporal queries.

Similarly to relational algebra, a *Temporal Relational Algebra* is a (finite) set of (first-order definable) operators of the form

$$\mathrm{Op} : \mathcal{R} \times \ldots \times \mathcal{R} \longrightarrow \mathcal{R}$$

defined on the universe of single-dimensional temporal relations $\mathcal{R}$ that conform to the data model of temporal databases.

Example 19.14 ([Tuzhilin and Clifford, 1990]) A temporal relational algebra (TRA) is a set of algebraic operators $\pi_V, \sigma_F, \bowtie, \cup, -, \mathcal{S}, \mathcal{U}$ over the universe of single dimensional temporal relations defined by:

$$\begin{array}{ll}
\pi_V(R) & = \{t, \theta_{|V} : DB, \theta, t \models R\} \\
\sigma_F(R) & = \{t, \theta_{|FV(R)} : DB, \theta, t \models R \wedge F\} \\
R \bowtie S & = \{t, \theta_{|FV(R) \cup FV(S)} : DB, \theta, t \models R \wedge S\} \\
R \cup S & = \{t, \theta_{|FV(R) \cup FV(S)} : DB, \theta, t \models R \vee S\} \\
R - S & = \{t, \theta_{|FV(R) \cup FV(S)} : DB, \theta, t \models R \wedge \neg S\} \\
\mathcal{S}(R, S) & = \{t, \theta_{|FV(R) \cup FV(S)} : DB, \theta, t \models R \,\mathbf{since}\, S\} \\
\mathcal{U}(R, S) & = \{t, \theta_{|FV(R) \cup FV(S)} : DB, \theta, t \models R \,\mathbf{until}\, S\}
\end{array}$$

Additional TRA operators, $\blacklozenge$, $\Diamond$, $\blacksquare$, and $\Box$ can be derived from the above operators similarly to Example 19.9.

The above definition allows us to translate (range restricted [Chomicki et al., 2001]) formulas in $L^{\{\mathbf{since},\mathbf{until}\}}$ to TRA.

Example 19.15 The query *find all rooms in which the last meeting was 'DB group'* is expressed in TRA as follows:

$$\mathcal{S}(\blacklozenge(\pi_{\text{Room}}(\sigma_{\text{Meeting=}\texttt{DB group}}(\texttt{Booking}))) - \pi_{\text{Room}}(\texttt{Booking})), \\ \pi_{\text{Room}}(\sigma_{\text{Meeting=}\texttt{DB group}}(\texttt{Booking})))$$

Note that to guarantee the range-restrictions of attributes, we had to rewrite the original formula. Full account of such rewrites was developed by Chomicki et al. [Chomicki et al., 2001].

However, this is also the reason why TRA with an arbitrary finite set of first-order definable operators cannot express all first-order queries (an immediate consequence of Theorems 19.3 and 19.4). This fact causes major problems when implementing query processors for temporal query languages, as the common (and efficient) implementations inherently depend on the equivalence of relational algebra and calculus to be able to execute all queries, [Abiteboul et al., 1995; Ullman, 1989].

19.5.4 *Multiple Temporal Dimensions*

The difficulty with defining a complete temporal relational algebra closed over a single-dimensional temporal data model is probably the most compelling reason for considering temporal data models with multiple temporal dimensions. The question we need to answer here is whether a fixed number of temporal dimensions, e.g., two dimensions used in the bitemporal data model, can lead to a closed algebra. We consider this problem in the following setting: we first define multidimensional temporal query languages by essentially following the development of Section 19.5.

It is easy to see that the language L^P is inherently multi-dimensional: we simply abandon the restriction on the number of free temporal variables in queries. To define the multidimensional counterpart of L^Ω we first define the *multidimensional temporal connectives*.

Definition 19.11 (Multidimensional Temporal Connective) A *k-ary m-dimensional temporal connective* is a formula in the first-order language of the temporal domain T with exactly m free variables $t_0^1, \ldots, t_0^m$ and k free predicate variables $X_1, \ldots, X_k$ (we assume that $t_i^1, \ldots, t_i^m$ are the only temporal variables free in the formula substituted for X_i).

Similarly to Definition 19.5 we define Ω to be a finite set of temporal connectives definitions: pairs of names $\omega(X_1, \ldots, X_k)$ and definitions of temporal connectives ω^*.

The language $L^{\Omega(m)}$ is a first-order logic extended with a finite set $\Omega(m)$ of m-dimensional temporal connectives. The semantics of $L^{\Omega(m)}$ queries is defined using the satisfaction relation

$$DB, \theta, t_1, \ldots, t_m \models \varphi$$

similarly to Definition 19.7: the only difference is that now we use m evaluation points $t_1, \ldots, t_m$ instead of a single evaluation point t. This definition can be used to define most of the common multi-dimensional temporal logics, e.g., the temporal logic with the *now* operator [Kamp, 1971], the Vlach and Åqvist system [Åqvist, 1979], and most of the interval logics [Allen, 1984; van Benthem, 1983]. To compare the expressive power of temporal logics with respect to the dimension of the temporal connectives we use the following observation. The $L^{\Omega(m)}$ language can be used over an n-dimensional temporal database for $n < m$ by modifying the definition of the satisfaction relation as follows:

$$DB, \theta, s_1, \ldots, s_m \models R(t_1, \ldots, t_n, \mathbf{x}) \text{ if } (s_1, \ldots, s_n, \theta(\mathbf{x})) \in \mathbf{R}$$

Similarly we can assume that all temporal formulas from $L^{\Omega(n)}$ can be used as subformulas in $L^{\Omega(m)}$. Thus $L^{\Omega(m)} \sqsubseteq L^{\Omega(m+1)}$ over m-dimensional temporal databases. It is also easy to see that a natural extension of the Embed map to m dimensions, Embed_m, gives us $L^{\Omega(m)} \sqsubseteq L^P$. The following theorem shows that the inclusions are proper:

Theorem 19.5 ([Toman and Niwinski, 1996; Toman, 2003c]) $L^{\Omega(m)} \sqsubset L^{\Omega(m+k)}$ *for* $m > 0$ *and an arbitrary finite set of* m*-dimensional temporal connectives* $\Omega(m)$ *where* k *is the maximal quantifier depth of any connective in* Ω.

As a consequence $L^{\Omega(m)} \sqsubset L^P$ for all $m > 0$. Thus L^P is the only first-order complete temporal query language (among the languages discussed in this chapter). On the other hand, for any fixed query $\varphi \in L^P$ we can find an $m > 0$ such that there is an equivalent query in $L^{\Omega(m)}$. Thus, e.g., the query that was used to separate FOTL from 2-FOL in Section 19.5 can be expressed in $L^{\Omega(2)}$.

19.5.5 *N1NF Data and Queries*

First-order nested query languages (without second-order quantifiers or the power-set constructor) are expressively equivalent to standard first-order queries in the 1NF model [Abiteboul et al., 1995]. Thus, save the possibility of avoiding additional key attributes, the *one-level* nesting in the temporal dimension does not add expressive capabilities to the more natural 1NF temporal models.

19.6 Space-efficient Encoding for Temporal Databases

In the second part we concentrate on *concrete* temporal databases: space efficient encodings of abstract temporal databases necessary from the practical point of view. First we explore in detail the most common encoding of time based on intervals and the associated concrete query languages. We introduce semantics-preserving translations of abstract temporal query languages into their concrete counterparts. We also introduce a generalization of such encodings using *constraints*. We conclude the section with a brief discussion of SQL-derived temporal query languages.

While abstract temporal databases provide a natural semantic domain for interpreting temporal queries, they are not immediately suitable for he implementation, as they are possibly infinite (e.g., when the database contains a fact holding for all time instants). Even for finite abstract temporal databases a direct representation may be extremely space inefficient: tuples are often associated with a large number of time instants (e.g., a validity interval). In addition, changes in the *granularity* of time may affect the size of the stored relations.

Our goal in this section is to develop a compact *encoding* for a subclass of abstract temporal databases that makes it possible to compactly represent such databases in finite space.

19.6.1 *Interval Encoding of Temporal Databases*

The most common approach to such an encoding is to use *intervals* as codes for convex 1-dimensional sets of time instants. The choice of this representation is based on the following empirical observation: Sets of time instants describing the validity of a particular fact in the real world can be often described by an interval or a finite union of intervals. We briefly discuss other encodings at the end of this section. For simplicity from now on we assume a discrete integer-like structure of time. However, dense time can also be accommodated by introducing open and half-open intervals. All the results in this section carry over to the latter setting.

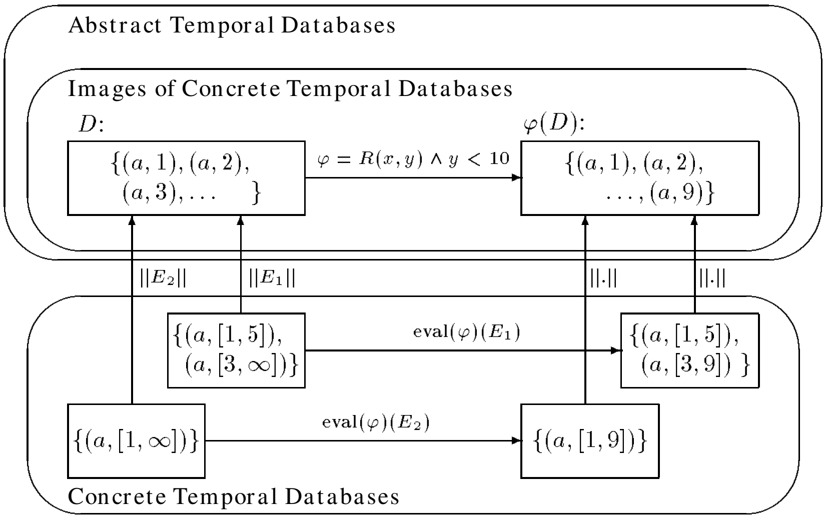
Definition 19.12 (Interval-based Domain $\mathrm{T}_\mathbf{I}$) Let $\mathrm{T}_P = (T, <)$ be a discrete linearly ordered point-based temporal domain. We define the set

$$I(T) = \{(a, b) : a \leq b, a \in T \cup \{-\infty\}, b \in T \cup \{\infty\}\}$$

where $<$ is the order over T_P extended with $\{(-\infty, a), (a, \infty), (-\infty, \infty) : a \in T\}$ (similarly for $\leq$). We denote the elements of $I(T)$ by $[a, b]$ (the usual notation for intervals). We also define four relations on the elements of $I(T)$:

$$\begin{array}{ll} ([a, b] <_{--} [a', b']) \iff a < a' & ([a, b] <_{+-} [a', b']) \iff b < a' \\ ([a, b] <_{-+} [a', b']) \iff a < b' & ([a, b] <_{++} [a', b']) \iff b < b' \end{array}$$

for $[a, b], [a', b'] \in I(T)$. The structure $\mathrm{T}_I = (I(T), <_{--}, <_{+-}, <_{-+}, <_{++})$ is the *Interval-based Temporal Domain (corresponding to* T_P*).*



E_1 and E_2 are concrete temporal databases that represent the abstract temporal database D.

Fig. 19.3 Abstract and Concrete Timestamp Temporal Databases

A concrete (timestamp) temporal database is defined analogously to the abstract (timestamp) temporal database. The only difference is that the temporal attributes range over intervals (T_I) rather than over the individual time instants (T_P).

Definition 19.13 (Concrete Temporal Database) A *concrete temporal database* is a finite first-order structure $\mathrm{D} \cup \mathrm{T}_I \cup \{\mathbf{R}_1 \ldots, \mathbf{R}_k\}$, where $\mathbf{R}_i$ are the *concrete temporal relations* which are finite instances of R_i over D and T_I.

Clearly the values of the interval attributes can be encoded as pairs of their endpoints which are elements of $T \cup \{-\infty, \infty\}$. However, it is important to understand that both T_P and T_I model *the same* structure of time instants, a single-dimensional linearly ordered timeline. This requirement is the crucial difference between the use of intervals in temporal databases and in various interval-based logics (cf. Section 19.5.4). The meaning of concrete temporal databases is defined by a mapping to the class of abstract temporal databases.

Example 19.16 A concrete representation of the instance in Figure 19.1 based on the interval encoding is shown below:

Booking		
Meeting	*Room*	*Time*
DB group	DC1331	[06-Jan-04.10:00,06-Jan-04.11:59]
Intro to Databases	MC4042	[06-Jan-04.10:00,06-Jan-04.11:19]
Intro to Databases	MC4042	[08-Jan-04.10:00,08-Jan-04.11:19]

Definition 19.14 (Semantic Mapping $\|.\|$) Let D_1 be an abstract temporal database and D_2 be a concrete temporal database over the same schema ρ. We say that D_2 *encodes* D_1 if

$$\mathbf{R}_i^{D_1}(t, \mathbf{x}) \iff \exists I \in T_I . \mathbf{R}_i^{D_2}(I, \mathbf{x}) \wedge t \in I$$

for all r_i in ρ, $t \in T_P$, and $\mathbf{x} \in D^{\text{arity}(r_i)}$, where $\mathbf{R}_i^{D}$ is the interpretation of the relation symbol R_i in the database D. This correspondence defines a map $\|.\|$ from the class of the concrete temporal databases to the class of the abstract temporal databases as an extension of the mapping of the relations in D_2 to the relations in D_1.

Note that $\|.\|$ is neither injective nor onto. Therefore there is no unique *canonical* concrete temporal database that encodes a given abstract temporal database. If only a single temporal dimension is allowed, however, we can define a *canonical* form for concrete temporal relations using *coalescing*: A single-dimensional temporal relation is *coalesced* if every fact is associated only with maximal non-overlapping intervals. A concrete temporal database is *coalesced* if all the user-defined relations are coalesced. Unfortunately, such a canonical normal form does *not* generalize to higher dimensions and Theorems 19.3 and 19.4 show that we cannot restrict our attention to the single-dimensional case.

19.6.2 *Concrete Temporal Query Languages*

The simplest query language over concrete temporal databases is the two-sorted first-order logic where variables and quantifiers of the temporal sort range over the domain T_I rather than T_P.

Definition 19.15 (Interval-based Language $\mathbf{L^I}$) Let ρ be a database schema and

$$L ::= R_i(I, \mathbf{x}) \mid L \wedge L \mid \neg L \mid \exists x.L \mid \exists I.L \mid x_1 = x_2 \mid I_1^* < I_2^*$$

where R_i is the temporal extension of $r_i \in \rho$ and $I^* \in \{I^+, I^-\}$.

L^I uses $I_1^* < I_2^*$ instead of the symbols $<_{--}, <_{+-}, <_{-+}, <_{++}$ from the *actual* structure of T_I. However, it is easy to see that, e.g., $I^- < J^-$ can be expressed as $I <_{--} J$, etc., and the new notation is thus merely syntactic sugar. We could also equivalently use Allen's algebra operators [Allen, 1983]. The resulting language is equivalent to L^I.

We assume the usual Tarskian semantics for formulas in L^I. Therefore L^I is fairly easy to implement using standard relational techniques. However, it is crucial to understand that this semantics of L^I is *not* point-based—the elements of T_I correspond to points in the two-dimensional plane (cf. Section 19.5.4). Thus L^I can not be immediately used as a query language over interval-based encodings of *point-based* abstract temporal databases because, among other things, it can easily express representation-dependent queries. Consider the following example:

Example 19.17 Let $\mathrm{D}_1, \mathrm{D}_2$ be two concrete temporal databases over the schema $(r(x))$ defined by $\mathbf{R}^{\mathrm{D}_1} = \{([0,2], a), ([1,3], a)\}$ and $\mathbf{R}^{\mathrm{D}_2} = \{([0,3], a)\}$. Then the formula $\exists I, J.\exists x(R(I,x) \wedge R(J,x) \wedge I \neq J)$ is true in D_1 but false in D_2.

This observation leads to the following definition:

Definition 19.16 (||.||-generic Queries) Let ||.|| be the semantics mapping and $\varphi \in L^I$. Then we say that φ *is* ||.||*-generic* if $||\mathrm{D}_1|| = ||\mathrm{D}_2||$ implies $||\varphi(\mathrm{D}_1)|| = ||\varphi(\mathrm{D}_2)||$ for all concrete temporal databases $\mathrm{D}_1, \mathrm{D}_2$.

In other words, no well-behaved query should distinguish between two equivalent, but differently represented temporal databases. Most interval-based query languages (e.g., TQuel or SQL/Temporal; cf. Section 19.7) are directly based on the language L^I (or one of its variants). This choice inherently leads to the possibility of expressing non ||.||-generic queries.

19.6.2.1 *Compilation of Abstract Query Languages*

A desirable solution is to use one of the abstract query languages for querying the encoded temporal databases. However, the semantics of these languages is defined over the class of abstract temporal databases (and we cannot simply apply the queries to the images of the concrete temporal databases under ||.||, as this would completely defy the purpose of using the concrete encodings and we would have to face the possibility of handling infinite relations). Thus we need to evaluate abstract queries directly over the concrete encodings. This goal is achieved using *compilation techniques* that transform abstract queries to formulas in L^I while *preserving meaning* under ||.||:

Theorem 19.6 ([Toman, 1996]) *There is a (recursive) mapping $F : L^P \to L^I$ such that $\varphi(||\mathrm{D}||) = ||F(\varphi)(\mathrm{D})||$ for all $\varphi \in L^P$ and all concrete temporal databases* D.

Moreover we can show that when using the interval-based encoding L^P can express all ||.||-generic queries in L^I:

Theorem 19.7 ([Toman, 1996]) *For every ||.||-generic $\varphi \in L^I$ there is $\psi \in L^P$ such that $||\varphi(\mathrm{D})|| = \psi(||\mathrm{D}||)$ for all concrete temporal databases* D.

Thus, considering $\|.\|$-generic queries, there is no advantage of basing a temporal query language on L^I.

The mapping from Theorem 19.6 can be also used for L^Ω by composing it with the Embed map from Definition 19.10. However, we may ask, is there is a more direct way from L^Ω to L^I? The following theorem gives a direct mapping of L^Ω to ATSQL (which is essentially a SQL version of L^I; cf. Section 19.7):

Theorem 19.8 ([Böhlen et al., 1996a; Chomicki et al., 2001]) *There is a (recursive) mapping $G : L^\Omega \to L^I$ such that $\varphi(\|\mathrm{D}\|) = \|F(\varphi)(\mathrm{D})\|$ for all $\varphi \in L^\Omega$ and all* coalesced *concrete temporal databases* D.

This mapping is considerably simpler than the indirect way through L^P. However, we pay the price for simplicity by having to maintain coalesced temporal relations, including all intermediate results during the bottom-up evaluation of the query. Note that the use of coalescing is possible due to the inherent single-dimensionality of L^Ω.

The mappings defined in Theorems 19.6 and 19.8 bring up an interesting point: what are the images of the temporal connectives themselves? It turns out that the results of such translations can be considered to be the equivalents of the original connectives that operate on concrete temporal relations, as shown in the example below.

Example 19.18 Let $\mathbf{UNTIL} = \lambda r.\lambda s.F \circ \mathrm{Embed}(r\ \mathbf{until}\ s)$ and ϕ and ψ two queries in L^Ω. Then

$$(\phi\ \mathbf{until}\ \psi)(\|\mathrm{D}\|) = \|(F \circ \mathrm{Embed}(\phi)\ \mathbf{UNTIL}\ F \circ \mathrm{Embed}(\psi))(\mathrm{D})\|$$

for all concrete temporal databases D.

A similar trick can be used to define the remaining temporal connectives. For coalesced databases we can use G in place of $F \circ \mathrm{Embed}$. This definition can be used to define an algebra over concrete relations that preserves the $\|.\|$ mapping and is thus suitable for implementing L^Ω.

19.6.3 *Concrete Multi-dimensional Temporal Databases*

Similarly to the single-dimensional case, storing the abstract multi-dimensional temporal databases directly may induce enormous space requirements. Thus we need to use encodings for multiple temporal dimensions. However, the introduction of multiple dimensions brings new challenges. The choice of encoding for sets of points in the multidimensional space is often much more involved than taking products of the encoding designed for the single-dimensional case. Assume that we attempt to represent the sets of points by hyper-rectangles—the multi-dimensional counterparts of intervals. It is easy to see that we can write first-order queries that do not preserve closure over this encoding:

Example 19.19 Consider the query $\varphi(t_1, t_2) = R(t_1) \wedge R(t_2) \wedge t_1 < t_2$. This query evaluated over the database $R = \{([1, 10])\}$ returns a triangle-like region where, for all the points in the region, the first coordinate is less then the second coordinate.

There are several ways of dealing with this issue:

- We can choose a *multi-dimensional temporal logic* where all the introduced connectives preserve closure over the chosen encoding.
- We can introduce closure restriction for formulas in L^P, [Chomicki et al., 1996; Toman, 1997; Chomicki et al., 2003a]. Such a restriction is designed to guarantee *attribute independence* of the free variables in the query and subsequently closure over an encoding obtained by taking an appropriate number of Cartesian (self-)products of the single-dimensional encoding.
- We can use a more general encoding using constraints in some suitable constraint language [Kanellakis et al., 1995; Libkin et al., 2000].

Another problem with using a multi-dimensional view of time is that it is much harder to define *normal forms* for temporal relations: in the single-dimensional case the coalesced relations provide a unique normal form (for the interval based encoding). However in two or more dimensions, such a normal form does not exist anymore (even when we only use hyper-rectangles).

19.6.4 *Other Encodings*

While the interval-based encoding of temporal databases is the most common in the literature, it is not the only possible approach. Another way to look at this problem is as follows: consider having

- a *finite* relation (with one or more temporal attributes), and
- a view that defines another (abstract) temporal relation in terms of the given relation.

Note that the instance of the relation defined by the view is not necessarily finite. We can think of the given finite relation as the *finite encoding* of an abstract temporal relation defined in terms of the view.

Example 19.20 (Interval Encoding Revisited) Consider a finite instance of a relation $R(t_1, t_2, x)$ where the first two attributes are temporal attributes and the last attribute is a data attribute. In addition consider the view

$$r(t, x) := \{(t, x) : \exists t_1, t_2.R(t_1, t_2, x) \wedge t_1 \le t \le t_2\}$$

It is easy to see that instances of R are essentially the concrete relations based on the interval encodings corresponding to instances the abstract relation r. The view provides an explicit version of the semantic mapping in Definition 19.14.

This approach, however, allows us to define many different mappings between abstract temporal relations and their concrete counterparts. Bettini et al. [Bettini et al., 1998] use this approach to study *temporal semantic assumptions* in temporal databases (in the setting of temporal granularities). Examples of temporal assumptions considered are:

- values of certain attributes *persist* until the value is replaced by another value later (with respect to the flow of time),
- values of a certain attribute are computed as an average, interpolation, etc., of the closest values preceding and following w.r.t. the flow of time;
- values of a certain attribute are computed as the sum of last three values; etc.

Note that the views define *abstract relations* and thus their instances may be infinite in general, even though it is defined on top of a finite relation. Thus the queries that define these views do not have to be range-restricted.

Example 19.21 (Persistence) Consider a relation $R(t, x, y)$ where the first attribute is a temporal attribute and the last two attributes are data attributes. Then the view

$$\begin{aligned} r(t,x,y) := \{(t,x,y) : \exists t_1.R(t_1,x,y) \wedge t_1 \leq t \\ \wedge \forall t_2, y_2.R(t_2,x,y_2) \implies (t_2 < t_1 \vee t_2 > t)\} \end{aligned}$$

defines an abstract temporal relation in which, for a given value x, the value for y *persists* until changed.

The same approach can be applied to define an abstract temporal relation from a log of insertions into and deletions from a temporal relation.

The association of abstract relations with their concrete encodings based on views has been studied extensively in the data integration community under the *global-as-a-view* (GAV) paradigm [Lenzerini, 2002]. Thus, query evaluation is essentially based on view expansion followed by the approach outlined in section 19.6.2. Note however, that to interpret the *results of queries* we need to specify or derive the temporal assumptions associated with the (finite) answer. One option here is to use the interval encoding as the default assumption.

19.7 SQL and Derived Temporal Query Languages

Up to this point we have only discussed temporal query languages based on logic. In this section we focus on the proposals for temporal extensions of more practical query languages, especially SQL [ISO, 1992]. When designing such an extension several obstacles need to be overcome:

(1) The semantics of SQL and other practical languages are commonly based on a bag (duplicate) semantics rather than on a set (Tarskian)

semantics. Therefore we need to design our extension to be consistent with the semantics of the language we started with. This also means that we need to deal with various non first-order features of the original language, e.g., with aggregation (the ability to count the number of tuples in a relation or to compute the sum of values in an attribute of the relation over all tuples).

(2) We need to design the extension in a way that consistently supports the chosen model of time. This point is often not emphasized enough and many of the proposals drift from the intended model of time in order to accommodate extra features. However, such design decisions lead to substantial problems in the long run, especially when a precise semantics of the extension has to be spelled out (this is one of the reasons why only informal semantics exist for many of these languages).

(3) To obtain a feasible solution we need to use a compact encoding of temporal databases introduced in Section 19.6. Therefore we need an *efficient* query evaluation procedure for the chosen class of *concrete databases*.

We would like to point out that vast majority of practical temporal query languages assume a *point-based model of time* (i.e., the truth of facts is associated with single time instants rather than with sets of time instants) [Chomicki, 1994]. Unfortunately (and also in most cases) the *syntax* is based on the syntax of L^I or some of its variants, e.g., languages that use Allen's interval algebra operators [Allen, 1983]. This discrepancy leads to a tension between the syntactic constructs used in the language and the intended semantics of queries. While we focus mostly on temporal extensions of SQL, our observations are general enough to apply to temporal extensions of other query languages, e.g., TQuel [Snodgrass, 1987].

Example 19.22 We demonstrate the differences between the approaches using the following query: *List all meetings with a scheduled break*. This query can be easily formulated in temporal logic as follows:

$$\blacklozenge\exists y.\mathtt{booking}(x,y) \wedge \neg\exists y.\mathtt{booking}(x,y) \wedge \Diamond\exists y.\mathtt{booking}(x,y).$$

This query could be equivalently expressed using future (or past) temporal connectives only.

The temporal extensions of SQL can be divided into two major groups, treated below, based on the *syntactic constructs* added to support temporal queries.

19.7.1 *Abstract Temporal Extensions of SQL*

We first consider extensions of SQL based on abstract temporal query languages.

19.7.1.1 *Extensions based on on L^P*

While query languages based on L^P were often considered to be inherently inefficient, recent results (especially Theorem 19.6, [Toman, 1996]) allow us to define a *point-based* extension of SQL that can be efficiently evaluated over the concrete interval-based temporal databases. The proposed language, SQL/TP, is a clean temporal extension of SQL [Toman, 1997]:

- The syntax and semantics of SQL/TP are defined as a natural extension of SQL with an additional data type based on the point-based temporal domain T_P (i.e., a linearly ordered set of time instants).
- The use of Theorem 19.6 also avoids the problems outlined later in the chapter in Example 19.25: the result of the F map is an ordinary query in L^I (or SQL). Therefore it can be efficiently evaluated over the concrete temporal databases based on interval encoding of timestamps (like any other SQL query).

The SQL/TP proposal also includes a definition of meaningful duplicate semantics and aggregation operations that are compatible with standard SQL [Toman, 1997]. The query from Example 19.22 can be formulated in SQL/TP in the expected way:

```
select r1.Meeting
from   Booking r1, Booking r2
where  r1.Meeting = r2.Meeting
  and  r1.time < r2.time
  and  not exists ( select *
                    from   Booking r3
                    where  r3.Meeting = r1.Meeting
                      and  r1.time < r3.time
                      and  r3.time < r2.time )
```

It is easy to see that the above formulation is is very similar to the declarative formulation of the query in the language L^P or in temporal logic.

19.7.1.2 *Languages based on L^Ω*

Another possible temporal extension of SQL can be based on the language L^Ω for some finite set of temporal connectives Ω. The temporal connectives can be introduced in the language similarly to set operations, e.g., the `union` operation.

Example 19.23 (SQL/$\{$since, until$\}$) The extended language is defined as follows. Every SQL query is also an SQL/$\{$**since**, **until**$\}$ query. Standard SQL queries are evaluated point-wise at every time instant. In addition if `Q1` and `Q2` are two queries (fullselects) then

```
Q1 since Q2          Q1 until Q2
```

are also SQL/{**since**, **until**} queries. The semantics of this language is based on a natural extension of Definition 19.7.

This language is a natural temporal extension of ATSQL's *sequenced semantics* [Snodgrass et al., 1995]. We can use Theorem 19.8 to evaluate queries in this language efficiently over coalesced interval-encoded concrete temporal databases, [Böhlen et al., 1996a; Chomicki et al., 2001]. Note that in this case all temporal relations have only one temporal attribute and therefore we can use coalescing.

Alternatively we can compose the mappings defined in Definition 19.10 with Theorem 19.6 to obtain a query evaluation procedure for $\mathbf{L}^{\Omega}$. This time we do not have to enforce coalescing of the concrete temporal relations as Theorem 19.6 allows evaluation of queries over *all* concrete temporal databases based on interval encoding. Chen et al. [Chen and Zaniolo, 1999] used this approach to define a *universal* way of temporalizing other query languages, such as QBE and Datalog.

19.7.2 *Concrete Temporal Extensions of SQL*

Next, we consider temporal extensions of SQL based on the concrete temporal query languages.

19.7.2.1 *Extensions of SQL based on L^I*

This group contains the majority of the proposals, in particular SQL/Temporal to the ANSI/ISO SQL standardization group [Snodgrass et al., 1996], and ATSQL [Snodgrass et al., 1995], the *applied* version of TSQL2 [Snodgrass, 1993], and the recent temporal extension of Informix (TIP) [Yang et al., 2000]. All these languages are directly based on L^I with Allen's algebra operators expressed in SQL syntax and using bag (duplicate) semantics.

Let us try to formulate the query from Example 19.22 in such a language, e.g., TSQL2 or its successor, SQL/Temporal. The solution that most people come up with is the query below (we use an intuitive and simplified syntax to make our point; for full details on syntax of SQL/Temporal see [Snodgrass et al., 1995; Snodgrass et al., 1996]):

Example 19.24 Query from Example 19.22 in SQL/Temporal:

```
select r1.Meeting
from   Booking r1, Booking r2
where  r1.Meeting = r2.Meeting
  and  r1.time before r2.time
```

Note that the `time` attributes range over intervals and the `before` relationship denotes the before relationship between two intervals. For a similar example in TQuel see [Chomicki, 1994].

Strangely enough, this query accesses the relation `Booking` only twice while the original query in Example 19.22 references the relation three times. This is often considered to be a "feature" of the L^I-based proposals and is attributed to the use of interval-based temporal attributes. It is also appealing due to savings in the query evaluation cost. However, closer scrutiny reveals that the above SQL/Temporal query is incorrect. Indeed, it returns all meetings that were held consecutively in three different rooms without a break. This result is consistent with the *two-dimensional* interval-based semantics of L^I. Similarly we can show many innocent-looking queries to be non-generic (in sense of Definition 19.16) and therefore necessarily incorrect with respect to their intended meaning. On the other hand access to interval endpoints (the *non-sequenced semantics* [Snodgrass et al., 1996]) is essential to write non-trivial temporal queries in SQL/Temporal.

There are two principal approaches that try to avoid this incorrect and unexpected behavior by modifying the semantics of the above languages.

19.7.2.2 *Coalescing*

The first (and historically oldest) approach is based on *coalescing*: an assumption that the timestamps are represented by maximal non-overlapping intervals (see Section 19.6). This is also the assumption commonly made when queries like the one in Example 19.24 are formulated. The coalescing attempts to produce a *normal form* of temporal relations over which the semantics of queries could be (uniquely) defined. The formal justification of this approach lies in realizing that the intended semantics of the language is point-based and therefore we can evaluate queries over any of the $||.||$-equivalent temporal databases (one of which is the coalesced one). For a detailed discussion of coalescing in temporal databases see [Böhlen et al., 1996b].

The most prominent representatives of this approach are TQuel [Snodgrass, 1987; Snodgrass, 1993], and TSQL2 [Snodgrass, 1995; Snodgrass et al., 1994]. However:

- Coalescing does not solve the problem with the query in Example 19.24. The query would only work if the `Booking` relation was coalesced *after* projecting out the attribute *Room*. This is not done in the (informal) semantics of TQuel nor TSQL2. It also means that the performance gain attributed to the use of interval valued attributes does not exist as we need to re-coalesce temporal relations on the fly.
- While coalescing preserves $||.||$-equivalence in the set-based semantics, it is incompatible with the use of duplicate semantics as it inherently

removes duplication. This is the main reason why the newer proposals, e.g., SQL/Temporal or ATSQL, do not use coalescing in order to preserve compatibility with SQL's duplicate semantics.

The most serious problem with coalescing-based approaches is exposed by Theorems 19.3 and 19.4: the theorems show that we cannot evaluate all first-order queries using only one temporal dimension. This result is fatal to the coalescing-based approaches since a canonical representation of temporal relations no longer exists and $\|.\|$-equivalent concrete relations can be distinguished using a first-order query in, for example, SQL/Temporal. We call such queries *representation dependent*. Even very simple queries, e.g., counting the number of regions along the axes, give different results depending on the particular representation.

19.7.2.3 *Folding and Unfolding*

The second approach is based on two additional operations: *fold* and *unfold* [Lorentzos, 1993; Lorentzos and Mitsopoulos, 1997]. These two operations allow us to convert a concrete temporal relation with interval-based timestamps to a temporal relation with point-based timestamps explicitly. An appropriate use of these two operations in in queries, e.g., defining

$$R \textit{ p-diff } S := \textit{fold}(\textit{unfold}(R) - \textit{unfold}(S)),$$

and then using the *p-diff* operator in place of set difference, would make the above query work, as the semantics is now defined essentially on the unfolded temporal relations and therefore is equivalent to the point-based semantics of L_P. However, a direct use of these operations, which is generally allowed in such languages as *unfold* is part of the syntax, is prohibitively expensive as shown in the following example.

Example 19.25 Consider a temporal relation R containing a single tuple $(a, [-2^n, 2^n])$ for some $n > 0$. Clearly, this relation can be stored in $2n + |a|$ bits. However, unfolding this relation gives us $\{(a, i) : -2^n \leq i \leq 2^n\}$. This relation needs space $2^n \cdot |a|$ which is *exponential* in the size of the original relation R.

Such a cost would clearly disqualify approaches employing these operators as a basis for a practical temporal query language. Note also that while the unfolding can be represented by a first-order query

$$\textit{unfold}(R) = \{(t, \mathbf{a}) : \exists I. R(I, \mathbf{a}) \wedge t \in I\},$$

there cannot be an equivalent *range-restricted* query (i.e., a query in which variables range only over values *present in the concrete database*) that defines this operator: the variable t is not *range-restricted* in the definition.

IXSQL [Lorentzos and Mitsopoulos, 1997] tries to combat the use of the *unfold* operation by defining a *normal form* of temporal relations and introducing an additional efficient *normalization operator* [Lorentzos et al., 1995] into the query language. This operator essentially converts IXSQL's temporal relations to $\|.\|$-equivalent normal forms and reinforces the fact that the meaning of the temporal relations is indeed point-based while intervals serve as a representational tool. The normalization operator is similar to the one used by Toman [Toman, 1996] to prove Theorem 19.6 and later extended to handle duplicates and aggregation in translations of SQL/TP queries to SQL/92 [Toman, 1997].

Similarly to SQL/TP (and unlike the TSQL2 family of languages) IXSQL treats temporal values simply as an additional data type and allows varying numbers of temporal attributes to be used by a relational schema. Date, while using a syntactic variant of IXSQL [Date et al., 2003], considers this approach superior based on the principle of *least departure from the relational foundations* as defined by Codd [Codd, 1972]. However, the true necessity of multiple temporal dimensions (and thus the need for an arbitrary number of temporal attributes) originates from Theorems 19.3, 19.4, and 19.5 and is necessary to guarantee relational completeness.

19.8 Updating Temporal Databases

In addition to storing and retrieving information, most applications of information systems also require the ability to *modify* the stored data. Temporal databases are no different. Here we again take advantage of the representation-independent nature of abstract temporal databases to define *database updates*. Indeed, from the conceptual point of view, updating an abstract temporal database is no different from updating a standard relational database. Thus the standard SQL-style statements for inserting, deleting, and modifying contents of relation instances can be used. There is, however, one small difference: in general, the instances of abstract temporal relations may be infinite and thus cannot be *populated* by inserting single tuples (this is always sufficient in the case of standard relational databases).

Example 19.26 Continuing with our running example, making a new booking of a room for a meeting can be achieved as follows:

```
INSERT into Booking (
   SELECT 'DBgroup', 'DC1331', t
   FROM   unit
   WHERE  '23-Jan-04.14.00' <= t <= '23-Jan-04.16.00' )
```

where `unit` is an auxiliary table that contains a single tuple[‡]. The inner query

[‡]As SQL does not allow `SELECT` blocks without a `FROM` clause.

produces a set

$$\{(\text{DB group}, \text{DC1331}, t) : \text{23-Jan-04.14.00} \leq t \leq \text{23-Jan-04.16.00}\}$$

that is added to the instance of `Booking` as representing another scheduled meeting. Deletion, e.g., creating a 20 minute break in the middle of the above meeting, is achieved analogously by the following statement:

```
DELETE from Booking
   WHERE  Meeting = 'DBgroup'
     AND  Room = 'DC1331'
     AND  '23-Jan-04.14.50' <= t <= '23-Jan-04.15.10'
```

In this case the set

$$\{(\text{DB group}, \text{DC1331}, t) : \text{23-Jan-04.14.50} \leq t \leq \text{23-Jan-04.15.10}\}$$

is removed from the abstract instance of the `Booking` relation.

Similar examples can be shown for SQL's `UPDATE` statement.

19.8.1 *Updates and Concrete Temporal Databases*

In addition to being able to *express* the update requests on the abstract level, and similarly to queries, the effects of the updates must be mapped faithfully into the concrete representation. This is reasonably easy when *interval encoding* is used for concrete databases:

- to make insertions, simply add the appropriate set of concrete tuples to the concrete relation;
- deletions and updates are more complex: we first use techniques similar to those used for mapping L_P queries to the L_I language to identify and remove the affected concrete tuples. However, since a single concrete tuple may represent multiple abstract ones and the deletion may only affect a subset of those tuples, a new tuple(s) may have to be reinserted into the concrete relation to compensate for this situation.

Example 19.27 The insertion in Example 19.26, assuming an underlying concrete representation based on interval encoding, is realized by adding a concrete tuple,

$$(\text{DB group}, \text{DC1331}, [\text{23-Jan-04.14.00}, \text{23-Jan-04.16.00}])$$

to the instance of the concrete representation of the relation `Booking`. While the insertion (save enforcement of integrity constraints) is relatively straightforward, a deletion (and update/modification) is slightly more complex due to the use of the concrete encoding. The deletion in Example 19.26 is performed in two steps:

(1) Tuples $\texttt{Booking}(\text{DB group}, \text{DC1331}, J)$, for $I \cap J \neq \emptyset$ are removed,
(2) Tuples $\texttt{Booking}(\text{DB group}, \text{DC1331}, J')$ for $J' \in (J - I)$ are reinserted[§],

where $I = [\text{23-Jan-04.14.50}, \text{23-Jan-04.15.10}]$. The two steps can be commuted to avoid the need for auxiliary relations. In our example, this leads to the deletion of the tuple

$$(\text{DB group}, \text{DC1331}, [\text{23-Jan-04.14.00}, \text{23-Jan-04.16.00}])$$

and to the insertion of the tuples

$$(\text{DB group}, \text{DC1331}, [\text{23-Jan-04.14.00}, \text{23-Jan-04.14.49}])$$
$$(\text{DB group}, \text{DC1331}, [\text{23-Jan-04.15.11}, \text{23-Jan-04.16.00}]).$$

Also, the situation becomes more complex if the mapping between the abstract and concrete representations is specified by a view. In this case, we are facing the view update problem and, depending on the complexity of the view definition, some of the updates may not be allowed.

19.8.2 *Append-only Databases and Data Expiration*

The situation in the case of append-only (or *transaction-time*) temporal databases is slightly different: here the *updates* are (at least conceptually) realized by adding a *new* state to an already existing (finite) history, yielding an *extended history* which is still finite. Such a history represents an abstract temporal database under the *persistence* assumption (cf. Section 19.6.4). However, in this scenario data accumulates over time and there is no apriori mechanism that allows us to remove/delete no longer needed parts of the history. To combat this problem various *data expiration* techniques have been developed. There are two main approaches to expiring data.

Administrative Approaches. These approaches identify data based on *policies* [Jensen, 1995; Skyt et al., 2003] which can be considered view specifications over the original history: all data not in the view extent are *expired*. Query answering then reduces to answering queries (formulated over the original history) using data in these views only. This problem has been extensively studied in the information integration area and is often referred to as *answering queries over views* [Levy et al., 1995] or the LAV (local as a view) approach [Lenzerini, 2002].

Query-driven Approaches. These approaches base their decisions of what data to expire on identifying parts of database histories that can be safely removed without affecting answers to a given set of queries [Chomicki, 1995; Toman, 2001; Toman, 2003a; Toman, 2003b].

[§]Note that subtracting an interval from another interval may yield a set of intervals, as in our example.

Data expiration techniques can be compared by measuring the size of the residual data (the amount of data retained after the expiration operation completes on a history) in terms of the length of the original history, the size of the active data domain, the queries, etc. Chomicki [Chomicki, 1995] and Toman [Toman, 2001; Toman, 2003b] show that for the past fragment of FOTL and the 2-FOL queries, respectively, the size of the residual data can be made independent of the length of the history, while preserving answers to a fixed set of queries. On the other hand Toman [Toman, 2003b] shows that such techniques cannot exist, e.g., for the future fragment of the fixpoint TL and for various duplicate-preserving temporal query languages.

19.9 Complex Structure of Time

So far we have only considered the simplest temporal domains possible: linearly ordered sets of time instants. In this section we consider relaxing this restriction.

19.9.1 *Complex Temporal Domains*

Often, a temporal domain has also a distinguished element 0 (the beginning of time). The standard temporal domains are: natural numbers $\mathbf{N} = (N, 0, <)$, integers $\mathbf{Z} = (Z, 0, <)$, rationals $\mathbf{Q} = (Q, 0, <)$, and reals $\mathbf{R} = (R, 0, <)$. However, additional structure can be added to the temporal domain; among the more common extensions considered are the

- Durations and Temporal Distances, and
- Periodic Sets.

The first extension can be achieved by introducing a fragment of linear arithmetic into the signature of the temporal domain. Similarly, the later extension adds the *modulo k* predicates to the signature.

19.9.2 *Impact on Integrity Constraints and Database Design*

The additional structure of the temporal domain yields new classes of integrity constraints available to users. Indeed, the linear order of time has already enabled the use of *order dependencies* (see Section 19.4.2). Following that approach, the new interpreted predicates in the signature of the complex temporal domain lead to more complex constraint dependencies.

The additional structure is also essential for specifying *calendars* and *time granularities* [Bettini et al., 2000], for example an *hour* can be defined as

$$hour(t, t') = t \equiv_{60} 0 \wedge t \leq t' < t + 60,$$

where $hour(t, t')$ holds whenever t is the first minute of the hour (which is used to identify hours) and t' is a time instant within the hour t. This also leads to the definition of functional dependencies that take granularity of time into account. Such dependencies constrain attribute values (Y) to depend on another values (X) *within a particular time granule*, e.g., an hour (denoted $R : X \to_{hour} Y$) and can be captured by a formula in the extended signature as follows:

$$\forall \bar{x}, \bar{y_1}, \bar{y_2}, t_1, t_2. \exists t. R(\bar{x}, \bar{y_1}, t_1) \wedge hour(t, t_1) \wedge R(\bar{x}, \bar{y_1}, t_2) \wedge hour(t, t_2) \to (\bar{y_1} = \bar{y_2})$$

Such dependencies, in turn, lead to the definition of *temporal normal forms*, e.g., TBCNF, and the development of decision procedures for logical implication [Wang et al., 1997; Wijsen, 1999].

19.9.3 *Impact on Query Languages*

The impact of such extensions on the abstract query languages is minimal: the new predicate symbols in the signature of the temporal domain are used in exactly the same way as the linear order symbol $<$ has been used so far. This, in the case of temporal logics, leads to the ability to define additional temporal connectives. For example, in temporal domains with constants it is natural to consider bounded versions of such connectives, e.g., $\Box_{[k_1,k_2]} A$, meaning that A is true in the future between time k_1 and time k_2, [Alur and Henzinger, 1992; Koymans, 1989]. Bounded temporal connectives can be defined like the unbounded ones using first-order formulas (Definition 19.5). In fact, for discrete time they can even be directly simulated using the unbounded connectives together with $\bullet$ and $\circ$. However, bounded connectives are quite useful and have been applied to the specification of real-time integrity constraints [Chomicki, 1995], and real-time logic programs [Brzoska, 1993; Brzoska, 1995]. Their advantage is that they are also meaningful in a slightly different semantic model of histories, in which the value of the clock in a state does not have to coincide with the index of the state in a history.

19.9.4 *Impact on Concrete Temporal Databases*

In order to introduce the additional structure of the temporal domain into the *concrete* temporal query languages, we need to consider how the added predicates affect the concrete temporal databases first. A careful analysis of Definition 19.12 reveals that the intervals are essentially quantifier-free formulas in the language of linear order with exactly one free variable. This idea can be generalized to more general structures [Kanellakis et al., 1995]: Let (T, σ) be a point-based temporal domain with the signature σ. Then we can define the set of formulas $C_\sigma = \{\phi(t) : \phi \in L_\sigma \wedge FV(\phi) = \{t\}\}$ where L_σ is the set of finite conjunctions of atomic formulas in the language of σ. This set can serve as the basis of the temporal domain for the class of concrete temporal databases, similar to intervals in Definition 19.12. An example of an alternative encoding is the use of *periodic constraints* [Kabanza et al., 1995; Toman and Chomicki, 1998], or *linear arithmetic constraints* [Kanellakis et al., 1995]. Concrete queries over such complex encodings are increasingly hard to write (cf. the problems we encountered in the case of linear order only). Thus the need for using abstract query languages in this setting is even more crucial.

19.10 Beyond First-order Logic

We survey here a number of temporal query languages whose expressive power goes beyond that of first-order logic. Most of these languages have only recently been proposed and thus their relative expressive power is not completely known and implementation techniques (in particular compilation to concrete query languages) have yet to be developed. In all likelihood such an implementation will require the development of more powerful concrete query languages, as currently available languages like TQuel or TSQL2 are not sufficiently expressive to serve as targets of the compilation.

19.10.1 *Second-order Temporal Connectives*

Definition 19.5 of temporal connectives can be extended with *monadic* second-order quantification over the temporal domain (quantification over subsets of the domain). This gives extra expressive power. For example, the modality "any time at an even distance from now" can be defined as

$$\begin{aligned} \equiv_2^0 X_1 \triangleq \exists t_1.X_1 \wedge \exists S.t_0 \in S \wedge t_1 \in S \wedge closed(S) \\ \wedge \forall S'.(t_0 \in S' \wedge closed(S') \Rightarrow S \subseteq S') \end{aligned}$$

where $closed(S) \triangleq \forall t.(t \in S \Rightarrow t+2 \in S) \wedge (t+2 \in S \Rightarrow t \in S)$. If $\mathbf{N}$ is the temporal domain, the above extension is identical in expressive power to ETL, an extension of temporal logic where temporal connectives are defined using regular languages. ETL was first proposed in [Wolper, 1983], in the propositional case and generalized to the first-order case in [Abiteboul et al., 1996]. The latter paper shows that the expressive power of ETL is incomparable to that of L^P. For other temporal domains, the expressive power of temporal logic with monadic second-order connectives has not yet been studied.

19.10.2 *Fixpoints*

A number of temporal fixpoint query languages have recently been proposed by [Abiteboul et al., 1999]:

- TS-FIXPOINT: the extension of L^P with inflationary fixpoints,
- T-FIXPOINT: the extension of temporal logic with inflationary fixpoints and some additional constructs, such as moves back and forth in time, and local and non-inflationary variables (for details, see [Abiteboul et al., 1999]),

Corresponding non-inflationary versions of those languages have also been proposed. It was shown in [Abiteboul et al., 1999], that TS-FIXPOINT is at least as expressive as T-FIXPOINT and that the relationship in the other direction depends on unresolved questions in complexity theory. On the other hand, T-FIXPOINT is more expressive than L^P. These languages appear to be mainly of theoretical interest. Fixpoint temporal logic μTL [Vardi, 1988], has been extensively used in program verification, although only in the propositional case.

19.11 Beyond the Closed World Assumption

So far we only considered semantics for temporal queries based on the *closed world assumption* (CWA). Under this assumption, temporal databases hold complete information about truth. An alternative that is more commonly considered by AI approaches is to treat the relational structures representing temporal databases as incomplete specifications and use the *open world assumption* (OWA) to answer queries. However, even for closed formulas in any of the abstract query languages we have considered so far, query processing essentially reduces to the satisfiability problem for formulas in these languages which, in all the cases, is highly undecidable.

19.11.1 *Infinite Database Histories and Potential Answers*

Even restricting the scope of the OWA to *append-only* temporal databases does not alleviate the decidability problems. Consider finite histories introduced in Definition 19.4 to be *finite prefixes* of infinite (or complete) histories. Queries, then, are evaluated with respect to the infinite histories (using the same semantic definitions as in Section 19.5, the only difference is in allowing an infinite temporal domain for the history). However, as only a finite portion (a prefix) of the history is available at a particular (finite) point in time, we need to define answers to queries with respect to *possible completions* of the prefix to a complete history.

Definition 19.17 Let H be a finite history, Q a query (in an appropriate query language), and θ a substitution. We say that

- θ is *a potential answer for Q with respect to H* if there is an infinite completion H' of H such that $H', \theta \models Q$.

- θ is *a certain answer for Q with respect to H* if for all infinite completions H' of H we have $H', \theta \models Q$.

The notion of *potential answer* is a direct generalization of the notion of *potential constraint satisfaction* [Chomicki, 1995].

Unfortunately, the above definition leads to *undecidable satisfiability problems* even for closed formulas in the temporal query languages introduced in Section 19.5. Indeed, potential/certain satisfaction is closer to the general satisfiability/validity problems that to satisfaction in a fixed model. Therefore potential/certain satisfaction is not useful as a basis for query evaluation. The negative results are as follows:

Proposition 19.1 ([Gabbay et al., 1994]) *The satisfaction problem for two dimensional propositional temporal logic over the natural numbers-based time domain is not decidable.*

This proposition rules out the temporal relational calculus. For weaker query languages based on single-dimensional temporal logic, or its Past and Future fragments, the results are as follows:

Proposition 19.2 ([Chomicki, 1995]) *For past formulas potential constraint satisfaction is undecidable.*

Proposition 19.3 ([Chomicki and Niwinski, 1995]) *For future temporal logic formulas (with a single quantifier in the scope of temporal connectives), potential constraint satisfaction is undecidable.*

19.11.2 *Decidable Fragments*

To regain ability to effectively evaluate queries under the OWA, the only option is to restrict the query languages themselves. Decidable fragments of first-order logic (i.e., languages in which the satisfiability problem is decidable) have been extensively studied. In the temporal setting, Hodkinson et. al. [Hodkinson et al., 2000] have introduced the *monodic temporal extensions* of several decidable fragments of first-order logic. The *monodicity* restriction stipulates that temporal subformulas of formulas in L^{Ω}, i.e., subformulas rooted by a temporal connective, may contain at most one free variable over the data domain (in addition to the requirement that the first-order portion of the formula belongs to an appropriate decidable fragment). Their technique has been successfully applied to a variety of logics, e.g., to the $\mathcal{ALC}$ and $\mathcal{DLR}$ description logics [Artale and Franconi, 2001], to the guarded, packed, and two variable fragments [Hodkinson, 2002]. In addition, the complexity of the decision procedures for these fragments has been studied [Hodkinson et al., 2003]. Decidability and complexity of fixpoint variants of these results have been studied by Franconi and Toman [Franconi and Toman, 2003].

19.11.3 *Temporal Logic Programming*

Another way to escape the limitations of temporal logic is to keep its syntax but use different semantics for its Horn subset. This is analogous to the move from first-order logic to logic programming. Indeed, several proposals have been made by [Abadi and Manna, 1989; Baudinet, 1992; Baudinet, 1995; Brzoska, 1991; Brzoska, 1993; Brzoska, 1995], to extend the language of Horn clauses with temporal connectives in such a way that there is still some notion of least model and resolution-based operational semantics. Not surprisingly, those languages can usually be translated to the standard logic programming languages. For instance, the temporal connectives in Templog, [Abadi and Manna, 1989; Baudinet, 1992; Baudinet, 1995], can be simulated in Prolog using an additional predicate argument that can contain the successor function symbol [Baudinet et al., 1993; Chomicki and Imieliński, 1988]. In this way, there is an exact correspondence between function-free Templog and $Datalog_{1S}$, an extension of Datalog with the successor function symbol in one predicate argument. More sophisticated temporal connectives involving numeric bounds on time, [Brzoska, 1991; Brzoska, 1993; Brzoska, 1995], can be simulated using arithmetic constraints [Jaffar and Lassez, 1987]. One can also study the extensions of the above Horn clause languages with stratified negation [Apt et al., 1988]. Temporal logic programming languages are directly amenable to efficient implementation using the existing logic programming technology. Recently, $Datalog_{1S}$ with negation has been used to define the operational semantics of active database systems [Lausen et al., 1998]. As far as the expressive power is concerned, it is not difficult to see that $Datalog_{1S}$ is subsumed by T-FIXPOINT and is incomparable to ETL. $Datalog_{1S}$ with stratified negation strictly subsumes ETL.

19.12 Conclusion

The chapter has provided mathematical foundations of temporal data management in a uniform framework. This framework allows us to formally compare and evaluate various data models and query languages proposed for managing temporal data. We believe that further work in this area, in addition to solving the remaining open problems, should focus on bridging the gap between logic and practical database systems by developing the necessary software tools and interfaces.

19.12.1 *Issues not Covered in the Chapter*

The chapter, however, does not cover all issues related to management of temporal data. Below we briefly discuss the main topics not covered by the chapter.

Conceptual Modeling of Temporal Data

In Section 19.4 we discuss temporal integrity constraints and the connected issues relating to temporal normal forms. However, the chapter does not cover conceptual design for temporal databases, in particular, various *Temporal ER* models; for a survey see [Gregersen and Jensen, 1999]. A formal treatment of these issues is presented elsewhere in this volume [Artale and Franconi, 2004]. Also, we do not discuss data models not derived from the relational model, such as the *object oriented* (OO) data model, and their temporal variants in this chapter.

Physical Design for Temporal Databases

Another set of issues not covered by this chapter are issues related to data structures and algorithms (query operators) supporting efficient processing of temporal queries and updates. However, we have shown that most of the approaches to querying temporal data essentially end up with first-order queries over concrete temporal databases—queries that depend heavily on the use of *ordering* of time instants. Note that, for example, the translation of temporal equijoin in an abstract query language yields an order-based join on the concrete encoding. A similar situation occurs naturally when using a variant of L^I in which the `WHERE` condition is explicit, e.g., in the form of an interval intersection operator, or when temporal queries are formulated directly in SQL [Snodgrass, 1999]. To facilitate these operations, special-purpose physical access methods (for a survey see [Salzberg and Tsotras, 1999]) and relational operators. For example, [Zhang et al., 2002] consider join methods tailored to processing ordered data.

However, many of these techniques are often limited to single or two-dimensional temporal data model. This is not sufficient for processing of general temporal queries as a consequence of Theorems 19.3, 19.4, and 19.5, and more general techniques such as those proposed by Lorentzos et al. [Lorentzos et al., 1995] are necessary.

Time Series and Temporal Data Mining

Considerable attention has been focused on discovering *interesting patterns* in time series—sequences of values generated over time, such as stock prices. Sequences and time series can be easily modeled as database histories. However, temporal query languages considered in this chapter are not adequate for discovering patterns, correlation, and other statistically interesting phenomena in such histories. Giannotti et al. [Giannotti et al., 2003] consider *logic based* languages for specifying such queries, albeit in a non-temporal setting. A thorough discussion of issues related to temporal data mining and its applications to time series, however, is beyond the scope of this chapter. For a recent overview see [Last et al., 2004].

19.12.2 *Extensions, Related Topics, and Future Directions*

In the remainder of this section we discuss several research directions that are closely related to temporal data management. In particular, we discuss how ideas and results developed for management of temporal data can be applied in those areas.

Spatio-temporal Databases

A very natural extension of the research presented here is to combine time and space in *spatiotemporal* databases. It has already been mentioned here that spatial databases can be treated similarly to multidimensional temporal databases. Spatiotemporal databases also fit in this framework [Geerts et al., 2001]. In principle, one could use both the snapshot and the timestamp models, as well as hybrid models (for example, snapshot databases where the snapshots are spatial timestamp databases). In a pure timestamp model (temporal and spatial timestamps), [Mokhtar et al., 2002] proposed a linear-constraint-based query language for databases of moving objects and [Vazirgiannis and Wolfson, 2001] described an SQL extension with abstract data types that model the trajectories of objects moving on road networks. In an earlier seminal paper in this area [Sistla et al., 1997] presented a a hybrid model query language based on a combination of temporal logic and spatial relationships.

In spatiotemporal databases, it is common to query not only the past states but also the (predicted) future states of the database. It seems fair to say that the design of spatiotemporal query languages is currently at an early stage of development, and the understanding of their formal properties has not yet reached the level of maturity of understanding of the properties of temporal query languages.

Streaming Data Management

The management of *streaming data* [Babcock et al., 2002], that is, query processing over sequences of data items arriving over time (*data streams*), has been the focus of recent research. Several groups are pursuing implementation of streaming data management systems (DSMS) [The STREAM Group, 2003; Chen et al., 2000; Madden et al., 2002]. The issues faced in this area have much incommon with those encountered in temporal databases, in particular when focusing on append-only database histories. For example, the issues related to limiting the space needed to store portions of the stream—called *synopses* in the streaming literature—which are necessary for *contiguous query* processing [Arasu et al., 2002] are essentially the same as those addressed by data expiration techniques for database histories (see Section 19.8.2 or [Toman, 2003b]).

The correspondence between temporal data management and data management for streaming data allows transfer of technology and results: temporal query languages, as surveyed in this chapter, offer mature and well-understood theoretical and practical foundations for the development of query languages for data streams.

Time in Document Management and XML

In contrast to the management of temporal data based on the relational model, handling time in document management systems or in XML repositories is not concerned with representing time-related information *external to the database* but rather with the evolution of a document or of a set of documents over time [Chien et al., 2001; Chien et al., 2002]. Thus the approaches are closer to version control systems used, for example, for managing source code of software systems. The design of temporal extensions of XML itself and of the associated query languages is in its infancy and the understanding of the issues involved is limited.

Model Checking

Model checking techniques were developed to verify temporal properties of (executions of) finite-state concurrent systems. Similarly to temporal databases, the input to a model checker is a finite encoding of all possible executions of the system (often in a form of a finite state-transition system) and a query, usually formulated in a dialect of propositional temporal logic. The techniques for verifying whether the formula is satisfied by the system are commonly based on the correspondence between propositional temporal logics and automata theory. Clarke et al. [Clarke, Jr. et al., 1999] provide an indepth introduction to the field. The main difference between these two approaches is that temporal databases commonly assume a fixed structure of time while model checking approaches tend to represent time explicitly using a transition system. The full understanding of the correspondence between these two fields is, however, remains to be studied.

Acknowledgment

This chapter is an extended and updated version of [Chomicki and Toman, 1998]. The authors gratefully acknowledge the United States National Science Foundation (grants IIS-9110581 and IIS-9632870) and the Natural Sciences and Engineering Research Council of Canada for their support of this research.

Bibliography

[Åqvist, 1979] Åqvist, L. (1979). A conjectured axiomatization of two-dimensional reichenbachian tense logic. *J. Philosophical Logic*, 8:1–45.

[Abadi and Manna, 1989] Abadi, M. and Manna, Z. (1989). Temporal Logic Programming. *Journal of Symbolic Computation*, 8(3).

[Abiteboul et al., 1996] Abiteboul, S., Herr, L., and den Bussche, J. V. (1996). Temporal Versus First-Order Logic to Query Temporal Databases. In *ACM Symposium on Principles of Database Systems*, pages 49–57.

[Abiteboul et al., 1999] Abiteboul, S., Herr, L., and den Bussche, J. V. (1999). Temporal Connectives Versus Explicit Timestamps to Query Temporal Databases. *Journal of Computer and System Sciences*, 58(1):54–68.

[Abiteboul et al., 1995] Abiteboul, S., Hull, R., and Vianu, V. (1995). *Foundations of Databases*. Addison-Wesley.

[Allen, 1983] Allen, J. (1983). Maintaining Knowledge about Temporal Intervals. *Communications of the ACM*, 26(11):832–843.

[Allen, 1984] Allen, J. (1984). Towards a General Theory of Action and Time. *Artificial Intelligence*, 23:123–154.

[Alur and Henzinger, 1992] Alur, R. and Henzinger, T. (1992). Logics and Models of Real-Time: A Survey. In *Real-Time: Theory in Practice*, pages 74–106. Springer-Verlag, LNCS 600.

[Apt et al., 1988] Apt, K., Blair, H., and Walker, A. (1988). Towards a Theory of Declarative Knowledge. In Minker, J., editor, *Foundations of Deductive Databases and Logic Programming*, pages 89–148. Morgan Kaufmann.

[Arasu et al., 2002] Arasu, A., Babcock, B., Babu, S., McAlister, J., and Widom, J. (2002). Characterizing Memory Requirements for Queries over Continuous Data Streams. In *ACM Symposium on Principles of Database Systems*, pages 221–232.

[Artale and Franconi, 2001] Artale, A. and Franconi, E. (2001). A Survey of Temporal Extensions of Description Logics. *Annals of Mathematics and Artificial Intelligence (AMAI)*, 30(1-4).

[Artale and Franconi, 2004] Artale, A. and Franconi, E. (2004). Temporal description logics. In Vila, L., van Beek, P., Boddy, M., Fisher, M., Gabbay, D., Galton, A., and Morris, R., editors, *Handbook of Time and Temporal Reasoning in Artificial Intelligence*. Elsevier.

[Babcock et al., 2002] Babcock, B., Babu, S., Datar, M., Motwani, R., and Widom, J. (2002). Models and Issues in Data Stream Systems. In *ACM Symposium on Principles of Database Systems*, pages 1–16.

[Bacchus et al., 1991] Bacchus, F., Tenenberg, J., and Koomen, J. (1991). A Non-Reified Temporal Logic. *Artificial Intelligence*, 52(1):87–108.

[Barendregt, 1984] Barendregt, H. P. (1984). *The Lambda Calculus: Its Syntax and Semantics*, volume 103 of *Studies in Logic and the Foundations of Mathematics*. North-Holland.

[Baudinet, 1992] Baudinet, M. (1992). A Simple Proof of the Completeness of Temporal Logic Programming. In Fariñas del Cerro, L. and Penttonen, M., editors, *Intensional Logics for Programming*. Oxford University Press. In print.

[Baudinet, 1995] Baudinet, M. (1995). On the Expressiveness of Temporal Logic Programming. *Information and Computation*, 117(2):157–180.

[Baudinet et al., 1993] Baudinet, M., Chomicki, J., and Wolper, P. (1993). Temporal Deductive Databases. In [Tansel et al., 1993], pages 294–320.

[Baudinet et al., 1999] Baudinet, M., Chomicki, J., and Wolper, P. (1999). Constraint-Generating Dependencies. *Journal of Computer and System Sciences*, 59(1):94–115.

[Bettini et al., 2000] Bettini, C., Jajodia, S., and Wang, X. S. (2000). *Time Granularities in Databases, Data Mining, and Temporal Reasoning*. Springer.

[Bettini et al., 1998] Bettini, C., Wang, X., Bertino, E., and Jajodia, S. (1998). Semantic Assumptions and Query Evaluation in Temporal Databases. *IEEE Transactions on Knowledge and Data Engineering*, 10(2):277–296.

[Bidoit et al., 2004] Bidoit, N., de Amo, S., and Segoufin, L. (2004). Order Independent Temporal Properties. *Journal of Logic and Computation*. In press.

[Böhlen et al., 1996a] Böhlen, M., Chomicki, J., Snodgrass, R., and Toman, D. (1996a). Querying TSQL2 Databases with Temporal Logic. In *International Conference on Extending Database Technology*, Avignon, France. Springer Verlag, LNCS 1057.

[Böhlen et al., 1996b] Böhlen, M., Snodgrass, R., and Soo, M. (1996b). Coalescing in Temporal Databases. In *International Conference on Very Large Data Bases*, pages 180–191.

[Brzoska, 1991] Brzoska, C. (1991). Temporal Logic Programming and its Relation to Constraint Logic Programming. In *International Logic Programming Symposium*.

[Brzoska, 1993] Brzoska, C. (1993). Temporal Logic Programming with Bounded Universal Modality Goals. In *International Conference on Logic Programming*.

[Brzoska, 1995] Brzoska, C. (1995). Temporal Logic Programming in Dense Time. In *International Logic Programming Symposium*.

[Chen and Zaniolo, 1999] Chen, C. X. and Zaniolo, C. (1999). Universal Temporal Extensions for Database Languages. In *IEEE International Conference on Data Engineering*, pages 428–437.

[Chen et al., 2000] Chen, J., DeWitt, D. J., Tian, F., and Wang, Y. (2000). NiagaraCQ: A Scalable Continuous Query System for Internet Databases. In *ACM SIGMOD International Conference on Management of Data*, pages 379–390.

[Chien et al., 2001] Chien, S.-Y., Tsotras, V. J., and Zaniolo, C. (2001). Version Management of XML Documents. *SIGMOD Record*, 30(3):46–53.

[Chien et al., 2002] Chien, S.-Y., Tsotras, V. J., and Zaniolo, C. (2002). Efficient schemes for managing multiversion XML documents. *The VLDB Journal*, 11(4):332–353.

[Chomicki, 1994] Chomicki, J. (1994). Temporal Query Languages: A Survey. In Gabbay, D. and Ohlbach, H., editors, *Temporal Logic, First International Conference*, pages 506–534. Springer-Verlag, LNAI 827.

[Chomicki, 1995] Chomicki, J. (1995). Efficient Checking of Temporal Integrity Constraints Using Bounded History Encoding. *ACM Transactions on Database Systems*, 20(2):149–186.

[Chomicki et al., 1996] Chomicki, J., Goldin, D., and Kuper, G. (1996). Variable Independence and Aggregation Closure. In *ACM Symposium on Principles of Database Systems*, pages 40–48, Montréal, Canada.

[Chomicki et al., 2003a] Chomicki, J., Goldin, D., Kuper, G., and Toman, D. (2003a). Variable Independence in Constraint Databases. *IEEE Transactions on Knowledge and Data Engineering*, 15(6):1422–1436.

[Chomicki and Imieliński, 1988] Chomicki, J. and Imieliński, T. (1988). Temporal Deductive Databases and Infinite Objects. In *ACM Symposium on Principles of Database Systems*, pages 61–73, Austin, Texas.

[Chomicki and Niwinski, 1995] Chomicki, J. and Niwinski, D. (1995). On the Feasibility of Checking Temporal Integrity Constraints. *Journal of Computer and System Sciences*, 51(3):523–535.

[Chomicki et al., 2003b] Chomicki, J., Saake, G., and van der Meyden, R., editors (2003b). *Logics for Emerging Applications of Databases*. Springer.

[Chomicki and Toman, 1998] Chomicki, J. and Toman, D. (1998). Temporal Logic in Information Systems. In Chomicki, J. and Saake, G., editors, *Logics for Databases and Information Systems*, pages 31–70. Kluwer.

[Chomicki et al., 2001] Chomicki, J., Toman, D., and Böhlen, M. H. (2001). Querying ATSQL Databases with Temporal Logic. *ACM Transactions on Database Systems (TODS)*, 26(2):145–178.

[Clarke, Jr. et al., 1999] Clarke, Jr., E. M., Grumberg, O., and Peled, D. A. (1999). *Model Checking*. MIT Press.

[Clifford et al., 1993] Clifford, J., Croker, A., and Tuzhilin, A. (1993). On the Completeness of Query Languages for Grouped and Ungrouped Historical Data Models. In [Tansel et al., 1993], pages 496–533.

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

[Codd, 1971] Codd, E. F. (1971). Normalized data structure: A brief tutorial. In Codd, E. F. and Dean, A. L., editors, *Proceedings of 1971 ACM-SIGFIDET Workshop on Data Description, Access and Control, San Diego, California*, pages 1–17. ACM.

[Codd, 1972] Codd, E. F. (1972). Relational Completeness of Data Base Sub-Languages. In Rustin, R., editor, *Data Base Systems*, pages 33–64. Prentice-Hall.

[Date et al., 2003] Date, c. J., Darwen, H., and Lorentzos, N. A. (2003). *Temporal Data and the Relational Model*. Morgan Kaufman.

[Emerson, 1990] Emerson, E. (1990). Temporal and Modal Logic. In van Leeuwen, J., editor, *Handbook of Theoretical Computer Science*, volume B, chapter 16, pages 995–1072. Elsevier/MIT Press.

[Franconi and Toman, 2003] Franconi, E. and Toman, D. (2003). Fixpoint Extensions of Temporal Description Logics. In *Description Logics DL2003*, volume 81 of *CEUR-WS*, pages 160–167.

[Gabbay et al., 1994] Gabbay, D., Hodkinson, I., and Reynolds, M. (1994). *Temporal Logic: Mathematical Foundations and Computational Aspects*. Oxford University Press.

[Geerts et al., 2001] Geerts, F., Haesevoets, S., and Kuijpers, B. (2001). A Theory of Spatio-Temporal Database Queries. In *International Workshop on Database Programming Languages*, pages 198–212.

[Giannotti et al., 2003] Giannotti, F., Manco, G., and Wijsen, J. (2003). Logical Languages for Data Mining. In [Chomicki et al., 2003b], chapter 9, pages 325–361.

[Ginsburg and Hull, 1983] Ginsburg, S. and Hull, R. (1983). Order Dependency in the

Relational Model. *Theoretical Computer Science*, 26:149–195.

[Ginsburg and Hull, 1986] Ginsburg, S. and Hull, R. (1986). Sort Sets in the Relational Model. *Journal of the ACM*, 33(3):465–488.

[Gregersen and Jensen, 1999] Gregersen, H. and Jensen, C. S. (1999). Temporal Entity-Relationship Models - A Survey. *Knowledge and Data Engineering*, 11(3):464–497.

[Hodkinson et al., 2003] Hodkinson, I., Kontchakov, R., Kurucz, A., Wolter, F., and Zakharyaschev, M. (2003). On the Computational Complexity of Decidable Fragments of First-order Linear Temporal Logics. In *Proceedings of the Joint 10th International Symposium on Temporal Representation and Reasoning and 4th International Conference on Temporal Logic (TIME-ICTL 2003)*.

[Hodkinson, 2002] Hodkinson, I. M. (2002). Monodic Packed Fragment with Equality is Decidable. *Studia Logica*, 72(2):185–197.

[Hodkinson et al., 2000] Hodkinson, I. M., Wolter, F., and Zakharyaschev, M. (2000). Decidable Fragments of first-order Temporal Logics. *Annals of Pure and Applied Logic*, 106:85–134.

[Hodkinson et al., 2002] Hodkinson, I. M., Wolter, F., and Zakharyaschev, M. (2002). Decidable and Undecidable Fragments of First-Order Branching Temporal Logics. In *IEEE Symposium on Logic in Computer Science*, pages 393–402.

[ISO, 1992] ISO (1992). Database Language SQL. ISO/IEC 9075:1992, International Organization for Standardization.

[Jaffar and Lassez, 1987] Jaffar, J. and Lassez, J.-L. (1987). Constraint Logic Programming. In *ACM Symposium on Principles of Programming Languages*.

[Jensen et al., 1996] Jensen, C., Snodgrass, R., and Soo, M. (1996). Extending Existing Dependency Theory to Temporal Databases. *IEEE Transactions on Knowledge and Data Engineering*, 8(4).

[Jensen et al., 1993] Jensen, C., Soo, M., and Snodgrass, R. (1993). Unification of Temporal Data Models. In *IEEE International Conference on Data Engineering*.

[Jensen, 1995] Jensen, C. S. (1995). Vacuuming. In Snodgrass, R. T., editor, *The TSQL2 Temporal Query Language*, pages 447–460. Kluwer Academic Publishers.

[Kabanza et al., 1995] Kabanza, F., Stevenne, J.-M., and Wolper, P. (1995). Handling Infinite Temporal Data. *Journal of Computer and System Sciences*, 51(1):3–17.

[Kamp, 1971] Kamp, H. (1971). Formal Properties of 'now'. *Theoria*, 37:227–273.

[Kamp, 1968] Kamp, J. (1968). *Tense Logic and the Theory of Linear Order*. PhD thesis, University of California, Los Angeles.

[Kanellakis et al., 1995] Kanellakis, P., Kuper, G., and Revesz, P. (1995). Constraint Query Languages. *Journal of Computer and System Sciences*, 51(1):26–52.

[Koymans, 1989] Koymans, R. (1989). *Specifying Message Passing and Time-Critical Systems with Temporal Logic*. PhD thesis, Technische Universiteit Eindhoven.

[Last et al., 2004] Last, M., Kandel, A., and Bunke, H., editors (2004). *Data Mining in Time Series Databases*, volume 57 of *Machine Perception and Artificial Intelligence*. World Scientific.

[Lausen et al., 1998] Lausen, G., Ludäscher, B., and May, W. (1998). On Logical Foundations of Active Databases. In Chomicki, J. and Saake, G., editors, *Logics for Databases and Information Systems*, pages 389–422. Kluwer.

[Lenzerini, 2002] Lenzerini, M. (2002). Data Integration: a Theoretical Perspective. In *ACM Symposium on Principles of Database Systems*, pages 233–246.

[Levy et al., 1995] Levy, A. Y., Mendelzon, A. O., Sagiv, Y., and Srivastava, D. (1995). Answering Queries Using Views. In *ACM Symposium on Principles of Database Systems*, pages 95–104.

[Libkin et al., 2000] Libkin, L., Kuper, G., and Paredaens, J., editors (2000). *Constraint Databases*. Springer.

[Lorentzos, 1993] Lorentzos, N. A. (1993). The Interval-extended Relational Model and Its Application to Valid-time Databases. In [Tansel et al., 1993], pages 67–91.

[Lorentzos and Mitsopoulos, 1997] Lorentzos, N. A. and Mitsopoulos, Y. G. (1997). SQL Extension for Interval Data. *IEEE Transactions on Knowledge and Data Engineering*, 9(3):480–499.

[Lorentzos et al., 1995] Lorentzos, N. A., Poulovassilis, A., and Small, C. (1995). Manipulation Operations for an Interval-extended Relational Model. *Data and Knowledge Engineering*, 17(1):1–29.

[Madden et al., 2002] Madden, S., Shah, M. A., Hellerstein, J. M., and Raman, V. (2002). Continuously Adaptive Continuous Queries over Streams. In *ACM SIGMOD International Conference on Management of Data*, pages 49–60.

[Mokhtar et al., 2002] Mokhtar, H., Su, J., and Ibarra, O. (2002). On Moving Objects Queries. In *ACM Symposium on Principles of Database Systems*, pages 188–198.

[Salzberg and Tsotras, 1999] Salzberg, B. and Tsotras, V. J. (1999). Comparison of Access Methods for Time-evolving Data. *ACM Computing Surveys (CSUR)*, 31(2):158–221.

[Sistla et al., 1997] Sistla, A. P., Wolfson, O., Chamberlain, S., and Dao, S. (1997). Modeling and Querying Moving Objects. In *IEEE International Conference on Data Engineering*, pages 422–432.

[Skyt et al., 2003] Skyt, J., Jensen, C. S., and Mark, L. (2003). A Foundation for Vacuuming Temporal Databases . *Data and Knowledge Engineering*, 44(1):1–29.

[Snodgrass and Ahn, 1986] Snodgrass, R. and Ahn, I. (1986). Temporal Databases. *IEEE Computer*, 19(9).

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

[Snodgrass, 1993] Snodgrass, R. T. (1993). An Overview of TQuel. In *Temporal Databases: Theory, Design, and Implementation*, chapter 6, pages 141–182. Benjamin/Cummings.

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

[Snodgrass, 1999] Snodgrass, R. T. (1999). *Developing Time-Oriented Database Applications in SQL*. Morgan Kaufmann.

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

[Snodgrass et al., 1996] Snodgrass, R. T., Böhlen, M. H., Jensen, C. S., and Steiner, A. (1996). Adding Valid Time to SQL/Temporal. ISO/IEC JTC1/SC21/WG3 DBL MAD-146r2 21/11/96, (change proposal), International Organization for Standardization.

[Snodgrass et al., 1995] Snodgrass, R. T., Jensen, C. S., and Böhlen, M. H. (1995). Evaluating and Enhancing the Completeness of TSQL2. Technical Report TR 95-5, Computer Science Department, University of Arizona.

[Tansel et al., 1993] Tansel, A., Clifford, J., Gadia, S., Jajodia, S., Segev, A., and Snodgrass, R. T., editors (1993). *Temporal Databases: Theory, Design, and Implementation*. Benjamin/Cummings.

[The STREAM Group, 2003] The STREAM Group (2003). STREAM: The Stanford Stream Data Manager (short overview paper). *IEEE Data Engineering Bulletin*, 26(1).

[Toman, 1996] Toman, D. (1996). Point vs. Interval-based Query Languages for Temporal Databases. In *ACM Symposium on Principles of Database Systems*, Montréal, Canada.

[Toman, 1997] Toman, D. (1997). Point-based Temporal Extensions of SQL. In *International Conference on Deductive and Object-Oriented Databases*.

[Toman, 2001] Toman, D. (2001). Expiration of Historical Databases. In *International Symposium on Temporal Representation and Reasoning*, pages 128–135. IEEE Press.

[Toman, 2003a] Toman, D. (2003a). Logical Data Expiration. In [Chomicki et al., 2003b], chapter 7, pages 203–238.

[Toman, 2003b] Toman, D. (2003b). Logical Data Expiration for Fixpoint Extensions of Temporal Logics. In *International Symposium on Advances in Spatial and Temporal Databases SSTD 2003*, pages 380–393.

[Toman, 2003c] Toman, D. (2003c). On Incompleteness of Multi-dimensional First-order Temporal Logics. In *International Symposium on Temporal Representation and Reasoning and International Conference on Temporal Logic*, pages 99–106.

[Toman and Chomicki, 1998] Toman, D. and Chomicki, J. (1998). Datalog with Integer Periodicity Constraints. *Journal of Logic Programming*, 35(3):263–306.

[Toman and Niwinski, 1996] Toman, D. and Niwinski, D. (1996). First-Order Queries over Temporal Databases Inexpressible in Temporal Logic. In *International Conference on Extending Database Technology*, Avignon, France.

[Tuzhilin and Clifford, 1990] Tuzhilin, A. and Clifford, J. (1990). A Temporal Relational Algebra as a Basis for Temporal Relational Completeness. In *International Conference on Very Large Data Bases*.

[Ullman, 1989] Ullman, J. (1989). *Principles of Database and Knowledge-Base Systems*, volume 2. Computer Science Press.

[van Benthem, 1983] van Benthem, J. (1983). *The Logic of Time*. D.Reidel.

[Vardi, 1988] Vardi, M. Y. (1988). A Temporal Fixpoint Calculus. In *ACM Symposium on Principles of Programming Languages*.

[Vazirgiannis and Wolfson, 2001] Vazirgiannis, M. and Wolfson, O. (2001). A Spatiotemporal Model and Language for Moving Objects on Road Networks. In *International Symposium on Advances in Spatial and Temporal Databases SSTD 2003*, pages 20–35.

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

[Wijsen, 1998] Wijsen, J. (1998). Reasoning about Qualitative Trends in Databases. *Information Systems*, 23(7):463–487.

[Wijsen, 1999] Wijsen, J. (1999). Temporal FDs on Complex Objects. *ACM Transactions on Database Systems*, 24(1):127–176.

[Wolper, 1983] Wolper, P. (1983). Temporal Logic Can Be More Expressive. *Information and Control*, 56:72–99.

[Wolper, 1989] Wolper, P. (1989). On the Relation of Programs and Computations to Models of Temporal Logic. In Banieqbal, B., Barringer, B., and Pnueli, A., editors, *Temporal Logic in Specification*, pages 75–123. Springer-Verlag, LNCS 398.

[Yang et al., 2000] Yang, J., Ying, H. C., and Widom, J. (2000). TIP: A Temporal Extension to Informix. In *ACM SIGMOD International Conference on Management of Data*, page 596.

[Zhang et al., 2002] Zhang, D., Tsotras, V. J., and Seeger, B. (2002). Efficient Temporal

Join Processing using Indices. In *IEEE International Conference on Data Engineering*, pages 103–114.

Contents

Index