1

Introduction to Temporal Database Research

Christian S. Jensen

A wide range of database applications manage time-varying data. In contrast, existing database technology provides little support for managing such data. The research area of temporal databases aims to change this state of affairs by characterizing the semantics of temporal data and providing expressive and efficient ways to model, store, and query temporal data. This chapter offers a brief introduction to temporal database research. It concisely introduces fundamental temporal database concepts, surveys state-of-the-art solutions to challenging aspects of temporal data management, and also offers a look into the future of temporal database research.

1 Introduction

Most applications of database technology are temporal in nature. Examples include financial applications such as portfolio management, accounting, and banking; record-keeping applications such as personnel, medical-record, and inventory management; scheduling applications such as airline, train, and hotel reservations and project management; and scientific applications such as weather monitoring. Applications such as these rely on *temporal databases*, which record time-referenced data.

Temporal database management is a vibrant field of research, with an active community of several hundred researchers who have produced some 2000 papers over the last two decades. Most of these papers are listed in a series of seven cumulative bibliographies (the newest one [63] provides pointers to its predecessors). The field has produced a comprehensive glossary of terminology [29], an edited volume which captures state of the art circa 1993 [56], and three workshop proceedings [16, 21, 49]. The near complete SQL3 standard includes a Part 7, SQL/Temporal [38]. The topic of temporal databases is now covered in textbooks

(e.g., [1, 13, 20, 35, 64]) and an encyclopedia [60]. Most recently, the first book dedicated entirely to temporal databases has appeared [54].

The present chapter's objective is to give a brief introduction, readable by the non-expert, to central concepts and issues in temporal database research. The chapter does not simply survey the author's contributions as documented in the subsequent chapters; nor does it exclusively survey contributions by others, not covered elsewhere in this publication. Rather, it offers an integrated coverage of research contributions by the author, some of which are elaborated upon in subsequent chapters, and contributions by other researchers. With its relatively broad coverage, the chapter sets the stage for the remainder of this publication.

More specifically, the chapter examines in turn a variety of central areas of temporal database research. Each area is first motivated, and then sample contributions are surveyed, to give the reader a feel for the challenges and issues that are faced in each particular area. The chapter concludes with a look into the possible future of temporal database research. The presentation is by no means complete in its coverage of areas, let alone contributions, necessarily omitting some contributions in the interest of brevity.

2 Temporal Data Semantics

Before we proceed to consider temporal data models and query languages, we examine, in data model-independent terms, the association of times and facts, which is at the core of temporal data management.

Initially, a brief description of terminology is in order. A database models and records information about a part of reality, termed either the *modeled reality* or the *mini-world*. Aspects of the mini-world are represented in the database by a variety of structures that we will simply term *database entities*. We will employ the term "fact" for any (logical) statement that can meaningfully be assigned a truth value, i.e., that is either true or false. In general, times are associated with database entities.

The facts recorded by the database entities are of fundamental interest. Several different temporal aspects may be associated with these. Most importantly, the *valid time* of a fact is the collected times—possibly spanning the past, present, and future—when the fact is true in the mini-world [29]. Valid time thus captures the time-varying states of the mini-world. All facts have a valid time by definition. However, the valid time of a fact may not necessarily be recorded in the database, for any of a number of reasons. For example, the valid time may not be known, or recording it may not be relevant for the applications supported by the database. If a database models different possible worlds, the database facts may have several valid times, one for each such world.

Next, the *transaction time* of a database fact is the time when the fact is current in the database. Unlike valid time, transaction time may be associated with any database entity, not only with facts. For example, transaction time may be associated with objects and values that are not facts because they cannot be true or false in isolation. To be more concrete, the value "63" may be stored in a database, but does not denote a logical statement. It is meaningful to associate transaction time with "63," but not valid time. Thus, all database entities have a transaction-time aspect. This aspect may or may not, at the database designer's discretion, be captured in the database. The transaction-time aspect of a database entity has a duration: from insertion to deletion, with multiple insertions and deletions being possible for the same entity. As a consequence of the semantics of transaction time, capturing this aspect of database entities renders deletions purely logical. Deleting an entity does not physically remove the entity from the database; rather, the entity remains in the database, but ceases to be part of the database's current state. Transaction time captures the time-varying states of the database, and applications that demand accountability or "traceability" rely on databases that record transaction time.

Observe that the transaction time of a database fact, say "*F*," is the valid time of the related fact, "*F is current in the database*." This would indicate that supporting transaction time as a separate aspect is redundant. However, both valid and transaction time are aspects of the content of all databases, and recording both of these is essential in a wide range of applications. In addition, transaction time, due to its special semantics, is particularly well-behaved and may be supplied automatically by the DBMS. Specifically, the transaction times of facts stored in the database are bounded by the time the database was created at one end of the time line and by the current time at the other end. This provides the rationale for the focus of most temporal database research on providing improved support for valid time and transaction time as separate aspects.

In addition, some other times have been considered, e.g., decision time [22, 33]. But the desirability of building decision time support into temporal database technologies is unclear, because the number and meaning of "the decision time(s)" of a fact varies from application to application and because decision times, unlike transaction time, generally do not exhibit specialized properties.

The valid and transaction time values of database entities are drawn from some appropriate time domain. There is no single answer to how to perceive time in reality and how to represent time in a database, and different time domains may be distinguished with respect to several orthogonal characteristics. First, the time domain may or may not stretch infinitely into the past and future. Second, time may be perceived as discrete, dense, or continuous. Some feel that time is really continuous; others contend that time is discrete and that continuity is just a convenient abstraction that makes it easier to reason mathematically about discrete phenomena. In databases, a finite and discrete time domain is typically assumed, e.g., in the

SQL standards. Third, a variety of different structures have been imposed on time. Most often, time is assumed to be totally ordered, but various partial orders have also been suggested, as has cyclic time.

An aspect of time that has been intriguing philosophers for centuries and that is difficult to describe fully is the concept of the current time, which we term *now* [15]. This concept is unique to time; indeed, there really does not exist any other notion quite like it. Among its properties, the current time is ever-increasing, all activity is trapped at the current time, and the current time separates the past from the future. The spatial equivalent, *here*, simply fails to enjoy the properties of *now*. As Merrick Furst puts it, "The biggest difference between time and space is that you can't reuse time." The uniqueness of *now* is one of the reasons why techniques from other research areas are not readily, or not at all, applicable to temporal data; *now* offers new data management challenges, which are particular to temporal databases. Specifically, the support for *now* transcends several parts of this publication and is a prominent theme in half a dozen of the chapters.

Much research has been conducted on the semantics and representation of time, from quite theoretical topics such as temporal logic and infinite periodic time sequences [14] to more applied questions such as how to represent time values in minimal space [17, 18]. Substantial research has been conducted that concerns the use of different time granularities and calendars in general [5], as well as the issues surrounding the support for indeterminate time values [19]. Also, there is a significant body of research on time data types, e.g., time points, time intervals (or "periods") [3], and temporal elements (sets of intervals) [24].

3 Temporal Data Models and Query Languages

Temporal data management can be very difficult using conventional (non-temporal) data models and query languages [54]. Accommodating the time-varying nature of the enterprise is largely left to the developers of database applications, leading to ineffective and inefficient ad-hoc solutions that must be reinvented each time a new application is developed. The result is that data management is currently an excessively involved and error-prone activity. Section 3.1 considers temporal data models, and Section 3.2 then covers query languages that are based on these data models. The subsequent step of providing support for temporal data modeling and database design is covered in Section 4.

3.1 Temporal Data Models

The first step in providing support for temporal data management is to extend the database structures of the data models supported by a conventional DBMS. Assuming a relational data model, mechanisms must be provided for capturing the valid

and transaction times of the facts recorded by the relations, leading to temporal relations.

Adding time to the relational model has been a daunting task, and more than two dozen extended relational data models have been proposed [30]. Most of these models support valid time only; some also support transaction time. We will consider three of these latter models and related design issues.

As a simple example, consider a video store where customers, identified by a `CustomerID` attribute, rent video tapes, identified by a `TapeNum` attribute. We consider a few rentals during May 1997. On the 2^{nd} of May, customer C101 rents tape T1234 for three days. The tape is subsequently returned on the 5^{th}. Also on the 5^{th}, customer C102 rents tape T1245 with an open-ended return date. The tape is eventually returned on the 8^{th}. On the 9^{th}, customer C102 rents tape T1234 to be returned on the 12^{th}. On the 10^{th}, the rental period is extended to include the 13^{th}, but this tape is not returned until the 16^{th}. The video store keeps a record of these rentals in a relation termed `CheckedOut`.

Figure 1 gives the relation instance in the Bitemporal Conceptual Data Model (BCDM) [30] that describes the sample rental scenario. This data model timestamps tuples, corresponding to facts, with values that are sets of (*transaction time*, *valid time*) pairs, captured using attribute `T` in the figure. Figure 2 provides a graphical illustration of the three timestamp values, which are termed bitemporal elements. In the general case of infinite and continuous time domains, these are finite unions of rectangles in the two-dimensional space spanned by transaction and valid time.

CustomerID	TapeNum	T
C101	T1234	{(2, 2), (2, 3), (2, 4), (3, 2), (3, 3), (3, 4), . . . , (UC, 2), (UC, 3)(UC, 4)}
C102	T1245	{(5, 5), (6, 5), (6, 6), (7, 5), (7, 6), (7, 7), (8, 5), (8, 6), (8, 7), . . . , (UC, 5), (UC, 6), (UC, 7)}
C102	T1234	{(9, 9), (9, 10), (9, 11), (10, 9), (10, 10), (10, 11), (10, 12), (10, 13), . . . , (13, 9), (13, 10), (13, 11), (13, 12), (13, 13), (14, 9), . . . , (14, 14), (15, 9), . . . , (15, 15), (16, 9), . . . , (16, 15), . . . , (UC, 9), . . . , (UC, 15)}

Figure 1: Bitemporal Conceptual `CheckedOut` Instance

The presence of a pair (*tt*, *vt*) in a timestamp of a tuple means that the current state of the database at time *tt* records that the fact represented by the tuple is valid at time *vt*. The special value UC ("until changed") serves as a marker indicating that its associated facts remain part of the current database state, and the presence of this value results in new time pairs being included into the sets of pairs at each clock tick.

The timestamp of the second tuple is explained as follows. On the 5^{th}, it is

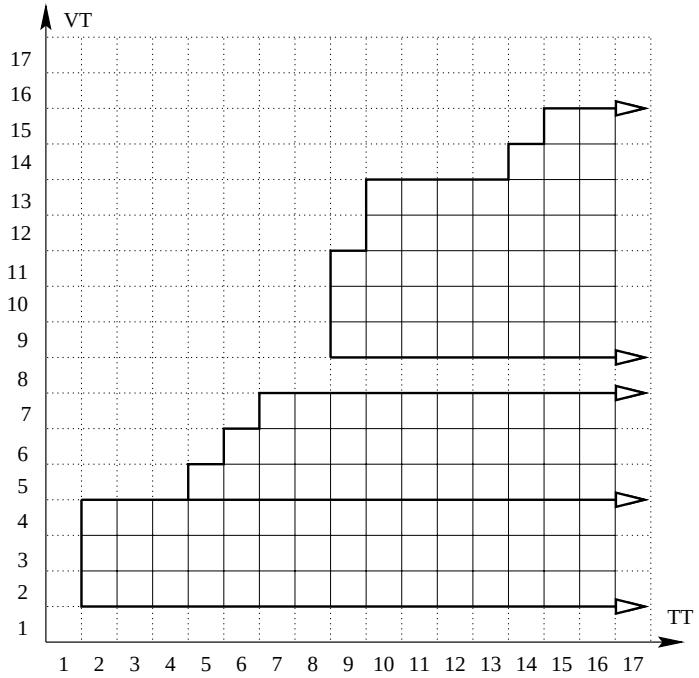

Figure 2: Graphical Illustration of the Timestamp Values in Figure 1

believed that customer C102 has checked out tape T1245 on the 5^{th}. Then, on the 6^{th}, the rental period is believed to include the 5^{th} and the 6^{th}. On the 7^{th}, the rental period extends to also include the 7^{th}. From then on, the rental period remains fixed. The current time is the 17^{th}, and as time passes, the region grows to the right; the arrows indicate this and correspond to the UC values in the textual representation.

The idea behind the BCDM is to retain the simplicity of the relational model while also capturing the temporal aspects of the facts stored in a database. Because no two tuples with mutually identical explicit attribute values (termed *value-equivalent*) are allowed in a BCDM relation instance, the full history of a fact is contained in exactly one tuple. In addition, BCDM relation instances that are syntactically different have different information content, and vice versa. This conceptual cleanliness is generally not obtained by other bitemporal models where syntactically different instances may record the same information.

However, when it comes to the internal representation and the display to users of temporal information, the BCDM falls short. Although it is arguably a first-normal-form relation, the varying length and voluminous timestamps of tuples are impractical to manage directly, and the timestamp values are also hard to comprehend in the BCDM format. Better suited representations of temporal information exist for these purposes.

Figure 3 illustrates the same temporal information as in Figure 1, in two different data models. The model exemplified in part (a) uses a practical and popular (particularly when implementation is considered) fixed-length format for tuples

CustomerID	TapeNum	T_s	T_e	V_s	V_e
C101	T1234	2	*UC*	2	4
C102	T1245	5	7	5	*now*
C102	T1245	8	*UC*	5	7
C102	T1234	9	9	9	11
C102	T1234	10	13	9	13
C102	T1234	14	15	9	*now*
C102	T1234	16	*UC*	9	15

(a)

CustomerID		TapeNum	
[2,*Now*] × [2,4]	C101	[2,*Now*] × [2,4]	T1234
[5,7] × [5,∞]	C102	[5,7] × [5,∞]	T1245
[8,*Now*] × [5,7]		[8,*Now*] × [5,7]	
[9,9] × [9,11]		[9,9] × [9,11]	T1234
[10,13] × [9,13]		[10,13] × [9,13]	
[14,15] × [9,∞]		[14,15] × [9,∞]	
[16,*Now*] × [9,15]		[16,*Now*] × [9,15]	

(b)

Figure 3: Alternative Representations of the CheckedOut Instance

[48]. Attributes T_s and T_e record starting and ending transaction times, and V_s and V_e record starting and ending valid times. In this format, each tuple's timestamp then encodes a rectangular or stair-shaped bitemporal region, and it may take several such tuples to represent a single fact.

The relation format in Figure 3(b) is a typical non-first-normal-form representation, in which a relation is thought of as recording information about some types of objects. The present relation records information about customers and thus holds one tuple for each customer in the example, with a tuple containing all information about a customer. In this way, a single tuple records multiple facts. In the example, the second tuple records two facts: rental information for customer C102 for the two tapes, T1245 and T1234.

Unlike in the BCDM, where relations must be updated at every clock tick, relations in these two other formats stay up-to-date; this is achieved by introducing variables (e.g., *now*) as database values that assume the (changing) current time value. The sample relations illustrate the two predominant choices for where to enter time values into relations, namely at the level of tuples (tuple timestamping) and at the level of attribute values ("attribute" timestamping).

It should be noted that all of the three types of bitemporal relations are equally expressive in that they may record the same facts. Put more formally (and briefly),

the relation instances that these models may record are snapshot equivalent.

The notion of *snapshot equivalence* corresponds to a point-based view of data. This view is pervasive in temporal database research, but it is not the only notion of equivalence. Consider the relations in Figure 4, which for simplicity record only valid time and for brevity have only a single non-timestamp attribute. These rela-

A	V_s	V_e
a	2	8
b	2	8

(a)

A	V_s	V_e
a	2	4
a	5	8
b	2	8

(b)

A	V_s	V_e
a	2	8
b	2	4
b	5	8

(c)

Figure 4: Different But Snapshot-Equivalent Relations

tions are clearly different. Specifically, the first relation is a coalesced version of the latter two relations. (The *coalescing* operation merges value equivalent tuples with the same non-timestamp attributes and adjacent or overlapping time intervals [8].) Because the three relations are different, they can well be taken to mean different things. In temporal data models, however, the relations are typically taken to contain the same information because they all contain the same snapshots: the snapshots computed at times 2 through 8 contain two tuples with values "a" and "b," (denoting some facts) and the snapshots at all other times are empty. This notion of equivalence corresponds to a point-based view of data: the time intervals or temporal elements associated with the database facts are merely convenient representations of sets of time points; and as long as the intervals associated with a fact represent the same set, it does not matter that they are different. In the figure, the different relations are associating interval [2, 8] versus intervals [2, 4] and [5, 8] with each of the two facts encoded by "a" and "b."

Stepping a little bit ahead, a query language that conforms to a point-based view of data should treat the three relations in Figure 4 identically. Any query language put on top of the BCDM is guaranteed to do so. Because the BCDM employs temporal elements and only permits coalesced relation instances, relations are only different if they are not snapshot equivalent, as mentioned earlier. In the figure, the last two relations are not legal in the BCDM.

As a contrast to the point-based view, it is also possible to adopt an interval-based view where additional semantics are given to the intervals associated with the facts in the relations. In the example, values "a" and "b" could be taken to mean that a specific customer has checked out two different tapes; the first relation would then indicate that two tapes were checked out once and for seven days each; and the second relation would indicate that the customer initially checked out the first tape for three days, and then for four more days. Some work exists that attempts to

accommodate an interval-based view of data within the confines of a point-based view [9].

3.2 Adding Time to Query Languages

Given the prevalence of applications that currently manage time-varying data, one might ask, why is a temporal query language even needed? Is the existence of all this SQL code of ostensibly temporal applications not proof that SQL is sufficient for writing such applications? The reality is that in conventional query languages like SQL, temporal queries *can* be expressed, but with great difficulty.

To illustrate the issue, consider the two relations `S-CheckedOut` and `V-CheckedOut` in Figure 5. The first is a snapshot relation that records which customers have currently checked out which video tapes; the second, a valid-time relation, records the check-out periods for rentals. The current time is 17, making the former relation a snapshot at the current time of the latter relation. Using SQL, it is

CustomerID	TapeNum
C101	T1234
C102	T1425
C102	T1324
C103	T1243

(a)

CustomerID	TapeNum	V_s	V_e
C101	T1234	2	*now*
C101	T1245	5	10
C102	T1245	22	25
C102	T1425	9	19
C102	T1434	4	14
C102	T1324	9	*now*
C103	T1243	7	21

(b)

Figure 5: Relations (a) `S-CheckedOut` and (b) `V-CheckedOut`

straightforward to express the number of current checkouts from `S-CheckedOut`. For example, this can be expressed as follows.

```
SELECT COUNT(TapeNum) AS Cnt FROM S-CheckedOut
```

We proceed to consider the temporal generalization of this query, asking now for the time-varying count of tapes checked out as recorded in relation `V-CheckedOut`. The reader may verify that the result given in Figure 6 correctly gives the count of tapes checked out at each point in time were a tape is checked out (assuming value 17 has been used for *now*). Expressing this query in SQL is exceedingly difficult, but possible if *now* is replaced with a fixed time value. The author is aware of one solution that consists of six steps and takes up 35 lines of complex SQL code.

As another example, specifying a key constraint on the non-temporal relation `S-CheckedOut` is trivial in SQL.

Cnt	V_s	V_e
1	2	3
1	20	25
2	4	4
2	18	19
3	5	6
4	7	8
4	15	17
5	11	14
6	9	10

Figure 6: A Time-Varying Count on the `TapeNum` Attribute of `V- CheckedOut`

```
ALTER TABLE S-CheckedOut ADD PRIMARY KEY (TapeNum)
```

This key constraint may be generalized to apply to a valid-time relation, now meaning that `TapeNum` is a key at each point in time or, equivalently, in each snapshot that may be produced from the valid-time table. Specifying this constraint on relation `V-CheckedOut` in SQL is again difficult. In one formulation, it takes a twelve line long and rather complicated SQL assertion to express this constraint [54, Ch. 5.3].

The lesson learned is that ordinary queries on non-temporal relations become extremely challenging when timestamp attributes are added. Even SQL experts would be hard pressed to express the examples above in SQL.

Some 40 temporal query languages have been defined [39, 53], most with their own data model. One of the more recent is TSQL2 [51], developed as a second-generation language by many of the designers of first-generation temporal query languages. The goal of TSQL2 was to consolidate approaches to temporal calculus-based query languages, to achieve a consensus extension to SQL-92 [37].

With a temporal query language, simple queries should remain simple when time is added. The temporal count query and the temporal key constraint can be expressed in the variant of TSQL2 being proposed for inclusion into SQL3 [52] as follows.

```
VALIDTIME
   SELECT COUNT(TapeNum) AS Cnt FROM S-CheckedOut

CONSTRAINT temporalKey VALIDTIME UNIQUE TapeNum
```

Early query languages were based on the relational algebra [36]. Calculus-based, Datalog-based, and object-oriented temporal query languages [50] appeared later. Much of the recent work involves extensions to SQL.

As query languages are strongly influenced by the underlying data model,

many of the issues raised in Section 3.1 have analogues in temporal query languages. For example, whether the data model timestamps tuples or attribute values influences the language.

Language design must consider the impact of the time-varying nature of data on all aspects of the language, including predicates on temporal values, temporal constructors, supporting states or events (or both) in the language, supporting multiple calendars, modification of temporal relations, cursors, views, integrity constraints, temporal indeterminacy, handling *now*, aggregates, schema versioning, vacuuming, and periodic data. Most of these topics have been the sole focus of several papers. However, these aspects interact in subtle ways, requiring consideration of all (or a substantial subset) to ensure that the design makes sense. Adequately documenting the design, rationale, and semantics of a comprehensive attack on the problem is daunting: the complete description of TSQL2 required an entire book [51].

Recently a set of desired properties of temporal query languages has emerged. These include *temporal upward compatibility* [4] (that is, conventional queries and modifications on temporal relations should act on the current state) and pervasive support for *sequenced queries* (that request the history of something, such as the temporal aggregation above) [10]; support for *point-based* and *interval-based* views of data [9]; adequate expressive power; and the ability to be efficiently implemented.

4 Designing Temporal Databases

The design of appropriate database schemas is critical to the effective use of database technology and the construction of effective information systems that exploit this technology. Database schemas capturing time-referenced data are often particularly complex and thus difficult to design.

The first of the two traditional contexts of database design is the data model of the DBMS to be used for managing the data. This data model, generally a variant of the relational model, is assumed to conform to the ANSI/X3/SPARC three-level architecture (e.g., [12]). In this context, database design must thus be considered at each of the view, logical, and physical (or, "internal") levels. In the second context, a database is modeled using a high-level, conceptual design model, typically the Entity-Relationship model. This model is independent of the particular implementation data model that is eventually to be used for managing the database, and it is designed specifically with data modeling as its purpose, rather than implementation or data manipulation, making it more attractive for data modeling than the variants of the relational model. Mappings are assumed available that bring a conceptual design into a schema that conforms to the specific implementation data model of

the DBMS to be used.

We proceed to consider in turn logical and conceptual design of temporal databases.

4.1 Logical Design

A central goal of conventional relational database design is to produce a database schema consisting of a set of *relation schemas*. In normalization theory, normal forms constitute attempts at characterizing "good" relation schemas, and a wide variety of normal forms has been proposed, the most prominent being third normal form and Boyce-Codd normal form. An extensive theory has been developed to provide a solid formal footing for relational database design, and most database textbooks expose their readers to the core of this theory.

In temporal databases, there is an even greater need for database design guidelines. However, the conventional normalization concepts are not applicable to temporal relational data models because these models employ relational structures different from conventional relations. New temporal normal forms and underlying concepts that may serve as guidelines during temporal database design are needed.

In response to this need, a range of temporal normalization concepts have been proposed [32], including temporal dependencies, keys, and normal forms. Consider the `CheckedOut` relation schema from Section 3.1, as exemplified in Figures 1 and 3. Does `CustomerID` (temporally) determine `TapeNum` or vice versa? Looking at the first representation in Figure 3 and applying conventional dependencies directly, the answer to both questions is no. (The possible answers are 'no' and 'perhaps,' as we are considering relation instances.) The second representation is so different from a regular relation that it makes little sense to directly apply conventional dependencies. The relation in Figure 1 also rules out any of the dependencies when we apply regular dependencies directly.

Considering that the different representations of the `CheckedOut` relation model the same miniworld and are capable of recording the same information, it may reasonably be assumed that these different representations would satisfy the same dependencies. At *any point in time*, a customer may have checked out several tapes. In contrast, a tape can only be checked out by a single customer at a single point in time. With this view, `TapeNum` temporally determines `CustomerID`, but the reverse does not hold.

If we consider the information contents of a temporal relation, independent of its actual format, to be the set of conventional snapshot relations it logically comprises, we achieve a means of applying the conventional relational normalization theory that leads to a temporal theory, which naturally generalizes conventional dependencies and may be applied to dependencies other than functional. Specifically, a temporal relation satisfies a temporal dependency if all its snapshots satisfy the

corresponding conventional dependency.

Temporal data models generally define timeslice operators, which may be used to determine the snapshots contained in a temporal relation. Accepting a temporal relation as their argument and a time point as their parameter, these operators return the snapshot of the relation corresponding to the specified time point. For example, a timeslice operator for temporal relations like the one in Figure 1 may take a point (tt, vt) in bitemporal space as its parameter. It returns the tuples of the argument relation that contain this time point, but omitting the timestamp attribute.

With this notion of temporal dependency based on snapshots, a temporal normalization theory may be built that parallels conventional normalization theory and that is independent of any particular representation of a temporal relation. However, the resulting theory, while temporal in that it applies to temporal databases, is actually atemporal, in that it applies to each snapshot of a temporal relation in isolation. This theory therefore fails to account for "temporal" aspects of data.

It is also relevant to consider dependencies and associated normal forms that effectively hold *between* time points [61, 62]. One approach to achieve this is to build the notion of time granularity into the normalization concepts. As a result, it not only is possible to consider snapshots computed at non-decomposable time points, but it is also possible to consider snapshots computed at coarser granularities. In our example relations, we have used day as the finest granularity; with the generalized theory, weeks and months may also be considered.

Another approach to taking the temporal aspects of data into account during database design is to introduce new concepts that capture the temporal aspects of data and may form the basis for new database design guidelines [30, 31].

Perhaps most prominently, *time patterns* may be used for capturing when the values of an attribute for an entity change in the modeled reality and in the database. For example, the set of tapes checked out by a customer may be expected to change substantially more frequently than the customer's address, meaning that the addresses of customers and their checked out video tapes should be stored in separate relations. (In this example, the temporal counterpart of Boyce-Codd normal form may reasonably be assumed to also imply such a decomposition, but this does not apply generally.)

Next, the concept of *lifespan*, that captures when an attribute of an entity has values, also has implications for database design. Specifically, if the lifespan of two attributes differ, null values of the unattractive "do not exist" variety result unless the attributes are stored in separate relations. Assuming that the temporal data model used timestamps tuples, attributes should also be stored separately when different temporal aspects need to be captured for them or when the temporal aspects are captured with differing precisions (resulting in different timestamp granularities).

4.2 Conceptual Design

By far, most research on the conceptual design of temporal databases has been in the context of the Entity-Relationship (ER) model. This model, in its varying forms, is enjoying a remarkable, and increasing, popularity in industry. Building on the example introduced in Section 3.1, Figure 7 illustrates a conventional ER diagram for video rentals.

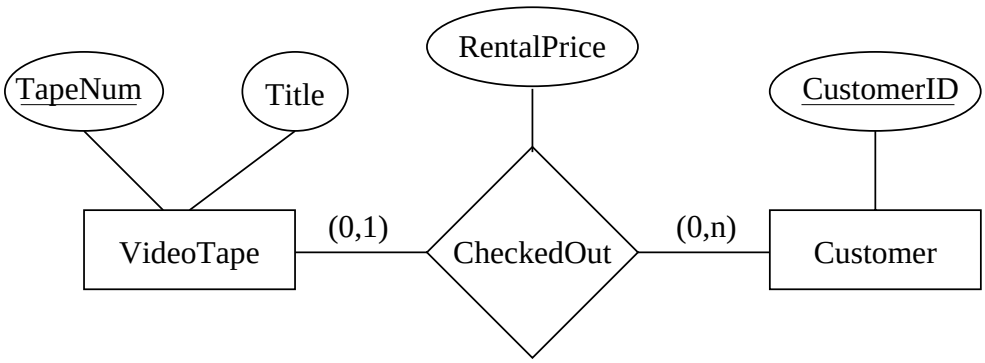

Figure 7: Non-temporal Conventional ER Diagram for Video Rentals

The research on temporal ER modeling is well motivated. It is widely known that the temporal aspects of the mini-world are very important in a broad range of applications, but are also difficult to capture using the ER model. Put simply, diagrams that would be intuitive and easy to comprehend without the temporal aspects become obscure and cluttered when an attempt is made to capture the temporal aspects.

The diagram in Figure 7 is non-temporal, modeling the mini-world at a single point in time. Attempting to capture the temporal aspects that are essential for this application complicates matters. It is necessary to capture the time when a customer has checked out a video tape. And since it is possible for the same customer to have checked out the same tape at different times, the `CustomerID` and `TapeNum` attributes do not identify a single instance of `CheckedOut`. Instead, it is necessary to make `CheckedOut` a ternary relationship type, introducing a new, somewhat artificial, entity type that captures the times of rentals. Including start and end time attributes on this entity type, the `CustomerID`, `TapeNum`, and `StartTime` attributes identify instances of `CheckedOut`. But simply requiring that these three attributes be a key of a relation representing rentals does not ensure the integrity of this relation—it remains possible for the same tape to be checked out more than once at the same point in time. As another issue, rental prices may vary over time, e.g., due to promotions and films getting old. Finally, including transaction time leads to further complications.

As a result, some industrial users simply choose to ignore all temporal aspects in their ER diagrams and supplement the diagrams with textual phrases to indicate

that a temporal dimension to data exists, e.g., "full temporal support." The result is that the mapping of ER diagrams to relations must be performed by hand; and the ER diagrams do not document fully the temporally extended relational database schemas used by the application programmers.

The research community's response to this predicament has been to develop temporally enhanced ER models. Indeed, about a dozen such models have been reported in the research literature [26]. These models represent attempts at modeling the temporal aspects of information more naturally and elegantly. The proposed extensions are based on quite different approaches.

One approach is to give all existing ER model constructs temporal semantics, basically following the "applies to all snapshots" approach used for making conventional normalization concepts "(a)temporal" in the previous section. In its extreme form, this approach does not result in any new syntactical constructs—all the original constructs have simply become temporal. The simplicity of this wholesale approach is attractive. However, this approach rules out databases with non-temporal parts; and legacy diagrams are no longer valid, i.e., while their syntax remains valid, their semantics have changed, and they therefore no longer describe the existing relational databases.

Another approach is to devise new notational shorthands that replace some of the patterns that occur frequently in ER diagrams when temporal aspects are being modeled. One example is the pattern that occurs when modeling a time-varying attribute in the ER model (e.g., the `RentalPrice` in our example). With this approach, it is possible to retain the existing ER-model constructs with their old semantics. This type of model may be more difficult to understand, but it does not invalidate legacy diagrams, and it is also possible to design non-temporal databases as well as databases where some parts are non-temporal while others are temporal.

In brief, the ideal temporal ER model is easy to understand in terms of the ER model; does not invalidate legacy diagrams and database applications; and does not restrict the database to be temporal, but rather permits the designer to mix temporal and non-temporal parts.

The existing models typically assume that their schemas are mapped to schemas in the relational model that serves as the implementation data model. The mapping algorithms are constructed to add appropriate time-valued attributes to the relation schemas. None of the models have one of the many time-extended relational models [39] as their implementation model. These models have data definition and query language capabilities that better support the management of temporal data and would thus constitute natural candidate implementation platforms. Also, mappings to emerging models (e.g., SQL3) are missing. It remains a challenge to design mappings that maximally exploit these and other candidate implementation platforms.

5 Temporal DBMS Implementation

There has been a vast amount of work in storage structures and access methods for temporal data, and a dozen-odd temporal DBMS prototypes have been reported [7]. Two basic approaches may be discerned. Traditionally, an *integrated* approach has been assumed, in which the internal modules of a DBMS are modified or extended to support time-varying data. More recently, a *layered* approach has also received attention [59]. Here, a software layer interposed between the user-applications and a conventional DBMS effectively serves as an advanced application that converts temporal query language statements into conventional statements that are subsequently executed by the underlying DBMS, which is itself not altered. While the former approach ensures maximum efficiency, the latter approach is more realistic in the short and medium term. Consistent with the vast majority of papers on temporal DBMS implementation, this section assumes an integrated approach utilizing timestamping of tuples with time intervals, unless explicitly stated otherwise.

5.1 Query Processing

A query formulated in some high-level, user-oriented query language is typically translated into an equivalent query, formulated in a DBMS-internal, algebraic query language. The DBMS then optimizes this algebraic expression by transforming it into an equivalent expression that is expected to be more efficient to process, the result being better query processing performance.

Optimization of temporal queries offers new challenges over optimization of conventional queries. At the core of the matter, temporal database queries are often large and complex. A recent book offers a wide range of examples of typical temporal database queries that are generally very complex [54]. Because of this added complexity, it is not only more important, but also more challenging, to optimize temporal database queries.

Specifically, the predicates used in temporal queries make these queries difficult to optimize. In non-temporal database applications, predicates are often equality predicates. As a reflection of this, much research in query processing has concentrated on equality predicates, and existing DBMSs are optimized for equality predicates (which occur in, e.g., equi-joins and natural joins). In contrast, temporal queries typically involve numerous inequality predicates. The perhaps most prominent source of such predicates is the test of overlap among two intervals. Inherent in temporal joins, this test occurs frequently in temporal queries and results in two equality predicates. Specifically, two intervals i and j overlap if the begin value of i is less than or equal to the end value of j and the begin value of j is less than or equal to the end value of i. Conventional DBMSs typically resort to nested-loop implementations of joins involving such inequality predicates, with their associated

inefficiency.

Other challenges posed by the complexity of temporal queries concern the coalescing of data [8] and the interactions among coalescing, duplicate removal, and ordering [47].

There are new and unexploited opportunities for query optimization when time is present. The current time advances continuously; and for transaction time, the time value used most recently in updates is the largest value used so far. This implies that a natural clustering or sort order will manifest itself. If relations are partitioned so that current and logically deleted tuples are stored separately, the relation with current tuples will be clustered on their transaction-time start values, while the tuples in the relation with logically deleted tuples will be clustered on their end times. Characteristics such as these can be exploited during query optimization and evaluation.

As another example of an optimization opportunity, the integrity constraint that the begin value of an interval is less than or equal to its end value holds for all intervals in the database. Next, for many relations, the intervals associated with a key value are contiguous in time, with one interval starting exactly when the previous interval ended. Semantic query optimization can exploit these integrity constraints, as well as additional ones that can be inferred.

5.2 Implementing Algebraic Operators

As explained earlier, a user-specified query is translated into an internal, algebraic form, which is then optimized using equivalence-preserving transformations. The DBMS has available a library of algorithms that implement the operations that occur in the resulting algebraic formulation of the query. As the next step, algorithms are chosen from the library for each operation, upon which the query is ready for execution. Good performance is dependent on the availability of good implementations of the operations.

Focus has been on a number of temporal algebraic operators, including selection, joins, aggregates, and duplicate elimination. Conventional approaches to computing these operators typically have poor performance, and new opportunities exist for efficiently implementing these operators. The selection operator is examined in the next section, as its implementation often involves a temporal index.

A wide variety of binary joins have been considered, including *time-join* and *time-equijoin* (TE-join), *event-join* and *TE-outerjoin*, *contain-join*, *contain-semijoin* and *intersect-join*, and *temporal natural join* (e.g., [27, 55]). The various algorithms proposed for these joins have generally been extensions to nested loop or merge joins that exploit sort orders or local workspace, as well as partitioning-based joins, but incremental techniques have also been proposed.

More generally, these latter techniques are particularly attractive for implementing operators on relations capturing transaction time, because these relations retain complete records of their past states. Incremental techniques cache results of previous computations and at later times reuse these results, together with records of the updates to the underlying relations that have occurred since the results were cached, to efficiently compute the up-to-date results. With support for transaction time, the records of updates are already contained in the relations [42].

Next, time-varying aggregates are especially challenging. While there has been much work on the topic in the data warehousing context, only a few papers have considered the more general problem. Finally, *coalescing* is an important operation in temporal databases. Coalescing merges value-equivalent tuples with adjacent intervals (and possibly also value-equivalent tuples with intervals that overlap). This operation may be implemented by first sorting the argument relation on the explicit attribute values as well as the valid time. In a subsequent scan, the merging is then accomplished.

5.3 Indexing Temporal Data

A variety of conventional indexes have long been used to reduce the need to scan an entire relation to access a subset of its tuples, to support the conventional selection algebraic operator and temporal joins. Similarly, a number of temporal indexing strategies are available [46, 58]. Many of the indexes are based on B^+-trees, which index on values of a single key; most of the remainder are based on R-trees, which index on ranges (intervals) of multiple keys. The worst-case performance for most proposals has been evaluated in terms of total space required, updates per change, and several important types of queries. Most of this work is in the context of the selection operator. As also mentioned, indexes may be used to efficiently implement temporal joins and also coalescing and aggregates—this is an area of active investigation.

6 Summary

This chapter has briefly introduced the reader to temporal data management, emphasizing central concepts, surveying important results, and describing the challenges faced. This section briefly summarizes the current state-of-the-art, and Section 7 discusses challenges that remain.

A great amount of research has been conducted on temporal data models and query languages, which has shown itself to be an extraordinarily complex challenge with subtle issues. The (snapshot-based) semantics of standard temporal relational schemas are well understood, as are the implications for database design. The Bitemporal Conceptual Data Model is gaining acceptance as a desirable model

in which to consider data semantics and as a good foundation for a temporal query language.

Many languages have been proposed for querying temporal databases, half of which have a formal basis. The numerous types of temporal queries are fairly well understood. The TSQL2 query language has consolidated many years of research results into a single, comprehensive language. New languages employ so-called statement modifiers, which offer a wholesale approach to giving temporal semantics to query language statements. Constructs from TSQL2, enhanced with statement modifiers, are being incorporated into the part of SQL3 called SQL/Temporal [38].

The semantics of the time domain, including its structure, dimensionality, and indeterminacy, are quite well understood, and representational issues of timestamps have recently been resolved. Operations on timestamps are now well understood, and efficient implementations exist.

Temporal joins, aggregates, and coalescing are well understood, and efficient implementations exist. More than a dozen temporal index structures have been proposed, supporting valid time, transaction time, or both. A handful of prototype temporal DBMS implementations have been developed.

7 Outlook

Although many important insights and results have been reported, many research challenges still remain in temporal database management, some of which are considered here.

The lack of consideration of some of these challenges has reduced the potential of earlier results. In many cases, core concepts have been established, but it remains to be shown how they may be combined and applied, to simplify and automate the management of time-referenced data in practice.

There is a need for increased *legacy-awareness* in a number of areas within temporal databases. Research is needed that takes into account the reality that most databases are in fact legacy temporal databases and that the applications running on them are in fact legacy temporal database applications. In contrast, most research so far has assumed that applications will be designed using a new temporal data model, implemented using novel temporal query languages, and run on as yet nonexistent temporal DBMSs. In the short to medium term, this is an unrealistic assumption. Indeed, perhaps in part because of this and despite the obvious need in the marketplace, as yet no prominent commercial temporal relational DBMS exists.

The recent growth in database architectures, including the various types of middleware, prompts a need for increased *architecture-awareness*. Studies are needed that provide the concepts, approaches, and techniques necessary for third-party developers to efficiently and effectively implement temporal database technol-

ogy while maximally exploiting available architectural infrastructure, as well as the functionality already offered by existing DBMSs. The resulting temporal DBMS architectures will provide a highly relevant alternative to the standard integrated architecture that is generally assumed. As a next step, research is needed on how to exploit existing and novel performance-improving advances, such as temporal algebraic operator implementations and indices, in these architectures. Finally, approaches for transitioning legacy applications will become increasingly sought after as temporal technology moves from research to practice.

Also, there has been little work on adding time to so-called fourth-generation languages that are revolutionizing the user interfaces of commercially available DBMSs. Figures 1 and 3 emphasize the need for ways to *visualize temporal data*. Scrolling down a table with additional timestamp attributes is not an effective means of visualizing the temporal variation in the data.

The results on the *conceptual design* of temporal databases as reported in the literature have potential for finding application in practice, but additional research is needed. When database designers actually understand the core temporal database concepts, perhaps most prominently valid and transaction time, they are able to design better databases using existing models and tools. A central challenge is to provide complete conceptual models, with associated design tools, that cover all aspects of designing a temporal database; empirical evaluation of these by real users is needed to provide essential insights. Reengineering of legacy databases is also a very relevant challenge in this context.

Concerning *performance*, more empirical studies are needed to compare temporal algebraic operator implementations, and to possibly suggest even more efficient implementations. Indexing techniques is an important aspect. While preliminary performance studies have been carried out for all or most of the proposed temporal indexes in isolation, there has been little effort to empirically compare them. More work is also needed on exploiting temporal indexes in algebraic operations other than selection. Finally, there has been little work in refining and validating cost models of temporal operators, or of developing and maintaining database statistics. For example, the cardinality (number of specific values) of an attribute is less useful than the average cardinality at a point in time. Another useful statistic is the number of *long-lived tuples*, the presence of which is the bane of some index structures and temporal algebraic operators.

A number of research areas that are either separate within temporal databases, overlap with this area, or take temporal databases as their point of departure also pose important challenges. Although not mentioned exhaustively in the coverage below, these areas are slated to offer ample challenges to those researchers concerned with effective and efficient implementation of advanced database functionality.

The area of *active databases*, rules responding to database changes and exter-

nal events are a focus. These may be extended to take into account prior history and temporal trends. For example, an absolute temperature reading in a nuclear power plant may be acceptable if it is part of a decreasing trend, but may signal a problem if it represents an increase. Some initial work has been reported in this area [22], but as yet there has been little integration of rule constructs and temporal constructs.

The area of *spatiotemporal databases* will become increasingly important [2, 23, 11]. Providing built-in support for both space and time makes it convenient to manage objects with extents in physical space and time, enabling new database applications.

For example, many "*moving objects*" such as people, animals, cars, aircraft, and ships will be equipped with wireless devices (e.g., GPS) that track their positions and make these available for storage in databases. The continued advances in wireless communications hardware and software constitute powerful drivers for these types of applications. While we are already witnessing the appearance of traffic-related systems (e.g., for tracking taxi's or fleet management), a very substantial growth can be expected in the number, sizes, diffusion, and diversity of these applications.

Next, *multimedia* presentations and *virtual reality* scenarios are in fact special breeds of spatiotemporal databases. And, again, there are powerful enablers of these kind of spatiotemporal database applications. We are witnessing continued advances in data storage, processor, network, and user interaction technologies. In short, the integration of temporal databases with spatial databases offers exciting new challenges and promises to become an important research area in the future. To mention just one example challenge, no index seems to accommodate well the past, current, and anticipated future positions of moving objects such as those mentioned above.

The area of *temporal data mining* [43, 44] is a relatively new one, where exploration has only recently started in earnest. While extracting static associations from a mass of data is an important goal, more effort needs to be focussed on associations that capture time-varying behavior, such as "when stock A goes up, stock B goes up within two weeks."

The fairly recent focus among vendors, users, and researchers alike on *data warehousing* has brought new prominence to temporal databases. W. H. Inmon who is known as the founder of data warehousing cites time variance as one of four salient characteristics of a data warehouse [28], and there is general consensus that a data warehouse is likely to contain several years of time-referenced data.

Being temporal, data warehouses are thus prime candidates to benefit from the advances in temporal databases. But cross-fertilization between temporal databases and data warehousing is largely absent. In fact, some of the original impetus for a separate data model and query language for data warehouses arose from a perceived lack of temporal support in the relational model and SQL. Few attempts

have been made to exploit the advances in temporal databases in the context of data warehousing, although exceptions do exist [6, 41]. The special architecture of a data warehouse and the emphasis on supporting advanced query functionality, e.g., application-specific time-series analysis, bring novel challenges to temporal database researchers. Reconciling the differences between general relational database schemas and specialized star schemas would help enable users and developers to achieve an integrated view of an enterprise.

Another active area of commercial products is that of *time-series* abstract data types (i.e., Informix's datablades, Oracle's cartridges). These data type extensions are highly useful for specialized applications, particularly in the financial sphere, but do not address the general problem of easy expression of the temporal constraints, queries, and modifications discussed in Section 3.2. Rather, a more comprehensive approach along the lines of the extensions being considered for SQL/Temporal appears to be attractive.

Adopting a longer term and more abstract perspective, it is likely that new database management technologies and application areas will continue to emerge that provide 'temporal' challenges. Due to the ubiquity of time and its importance to most database management applications, and because built-in temporal support generally offers many benefits and is challenging to provide, research in the temporal aspects of new database management technologies will continue to flourish for existing as well as new application areas.

Acknowledgments

The understanding of temporal databases as exposed in this chapter was shaped in large part during discussions with my colleagues, most prominently Rick Snodgrass, but also Mike Böhlen, Jim Clifford, and Curtis Dyreson. Although joint writings with Rick served as an outset, the chapter solely reflects my personal views and idiosyncrasies, for which only I should be held accountable.

References

[1] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases.* Addison-Wesley 1995.

[2] T. Abraham and J. F. Roddick. Survey of Spatio-Temporal Databases. *GeoInformatica,* 3(1):61–99, March 1999.

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

[4] J. Bair, M. H. Böhlen, C. S. Jensen, and R. T. Snodgrass. Notions of Upward Compatibility of Temporal Query Languages. *Wirtschaftsinformatik*, 39(1):25–34, February 1997.

[5] C. Bettini, X. S. Wang, and S. Jajodia. A General Framework for Time Granularity and Its Application to Temporal Reasoning. *Annals of Mathematics and Artificial Intelligence*, 22(1–2):29–58, 1998.

[6] R. Bliujūtė, S. Šaltenis, G. Slivinskas, and C. S. Jensen. Systematic Change Management in Dimensional Data Warehousing. In *Proceedings of the Third International Baltic Workshop on DB and IS*, pp. 27–41, Riga, Latvia, April 1998.

[7] M. Böhlen. Temporal Database System Implementations. *ACM SIGMOD Record*, 24(4):53–60, December 1995.

[8] M. H. Böhlen, R. T. Snodgrass, and M. D. Soo. Coalescing in Temporal Databases. In *Proceedings of the 22nd International Conference on Very Large Data Bases*, pp. 180–191, Bombay, India, September 1996.

[9] M. H. Böhlen, R. Busato, and C. S. Jensen. Point-Versus Interval-Based Temporal Data Models. In *Proceedings of the Fourteenth International Conference on Data Engineering*, pp. 192–200, Orlando, Florida, February 1998.

[10] M. H. Böhlen and C. S. Jensen. Temporal Statement Modifiers. Submitted for publication.

[11] M. H. Böhlen, C. S. Jensen, and M. Scholl (eds.). *Spatio-Temporal Database Management. Proceedings of the International Workshop on Spatio-Temporal Database Management.* LNCS 1678. Edinburgh, Scotland, September 1999.

[12] M. L. Brodie and J. W. Schmidt. Final Report of the ANSI/X3/SPARC DBS-SG Relational Database Task Group. *ACM SIGMOD Record*, 12(4):0–62, December 1982.

[13] J. Celko. *Joe Celko's Data and Databases: Concepts in Practice*. Morgan Kaufman Publishers 1999.

[14] J. Chomicki and D. Toman. Temporal Logic in Information Systems. In *Logics for Databases and Information Systems*, Kluwer Academic Publishers 1998, pp. 31–70.

[15] J. Clifford, C. E. Dyreson, T. Isakowitz, C. .S. Jensen, and R. T. Snodgrass. On the Semantics of 'Now' in Databases. *ACM Transactions on Database Systems*, 22(2):171–214, June 1997.

[16] J. Clifford and A. Tuzhilin (eds.). *Recent Advances in Temporal Databases: Proceedings of the International Workshop on Temporal Databases*. Workshops in Computing Series. Springer-Verlag 1995.

[17] C. E. Dyreson and R. T. Snodgrass. Timestamp Semantics and Representation. *Information Systems*, 18(3):143–166, 1993.

[18] C. E. Dyreson and R. T. Snodgrass. Efficient Timestamp Input and Output. *Software—Practice and Experience*, 24(1):89–109, 1994.

[19] C. E. Dyreson and R. T. Snodgrass. Supporting Valid-Time Indeterminacy. *ACM Transaction on Database Systems*, 23(1):1–57, 1998.

[20] R. Elmasri and S. Navathe. *Fundamentals of Database Systems*. Benjamin/Cummings 1994.

[21] O. Etzion, S. Jajodia, and S. Sripada (eds.). *Temporal Databases: Research and Practice*. LNCS 1399, Springer-Verlag 1998.

[22] O. Etzion, A. Gal, and A. Segev. Temporal Active Databases. In [49].

[23] A. Frank, R. H. Güting, C. S. Jensen, M. Koubarakis, N. Lorentzos, Y. Manolopoulos, E. Nardelli, B. Pernici, H.-J. Schek, M. Scholl, T. Sellis, B. Theodoulidis, and P. Widmayer. Chorochronos—A Research Network for Spatiotemporal Database Systems. *ACM SIGMOD Record*, 28(3):, September 1999.

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

[25] S. K. Gadia and G. Bargava. SQL-like Seamless Query of Temporal Data. In [49].

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

[27] H. Gunadhi and A. Segev. A Framework for Query Optimization in Temporal Databases. In *Proceedings of the Fifth Conference on Statistical and Scientific and Databases Management*, pp. 131–147, Charlotte, North Carolina, April 1990.

[28] W. H. Inmon. *Building the Data Warehouse*. John Wiley & Sons 1992.

[29] C. S. Jensen and C. E. Dyreson (eds.). A Consensus Glossary of Temporal Database Concepts—February 1998 Version. [21, pp. 367–405].

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

[31] C. S. Jensen and R. T. Snodgrass. Temporally Enhanced Database Design. In [40].

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

[33] S.-K. Kim and S. Chakravarthy. Modeling Time: Adequacy of Three Distinct Time Concepts for Temporal Databases. In *Proceedings of 12th International Conference on Entity-Relationship Approach*, LNCS 823, pp. 475–491, Arlington, TX, December 1993.

[34] W. Kim (ed.) *Modern Database Systems: The Object Model, Interoperability and Beyond*. Addison-Wesley/ACM Press 1995.

[35] M. Levene and G. Loizou. *A Guided Tour of Relational Databases and Beyond*. Springer-Verlag 1999.

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

[37] J. Melton and A. R. Simon. *Understanding the New SQL: A Complete Guide*. Morgan Kaufmann Publishers 1993.

[38] J. Melton (editor). *SQL/Temporal*. July 1996. (ISO/IEC JTC 1/SC 21/WG 3 DBL-MCI-0012.)

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

[40] M. P. Papazoglou, S. Spaccapietra, and Z. Tari (eds.). *Object-Oriented Data Modeling*. MIT Press 2000, to appear.

[41] T. B. Pedersen and C. S. Jensen. Multidimensional Data Modeling for Complex Data. In *Proceedings of the 15th International Conference on Data Engineering* , pp. 336–345, Sydney, Australia, March 1999.

[42] D. Pfoser and C. S. Jensen. Incremental Join of Time-Oriented Data. In *Proceedings of the Eleventh International Conference on Scientific and Statistical Database Management*, pp. 232–243, Cleveland, Ohio, July 1999.

[43] J. F. Roddick and M. Spiliopoulou. A Bibliography of Temporal, Spatial and Spatio-Temporal Data Mining Research. *ACM SIGKDD Explorations*, 1(1):34–38, June 1999.

[44] J. F. Roddick and M. Spiliopoulou. Temporal Data Mining: Survey and Issues. Research Report ACRC-99-007. School of Computer and Information Science, University of South Australia. 1999.

[45] J. F. Roddick and J. D. Patrick. Temporal Semantics in Information Systems—a Survey. *Information Systems*, 17(3):249–267, October 1992.

[46] B. Salzberg and V. J. Tsotras. A Comparison of Access Methods for Time Evolving Data. *ACM Computing Surveys*, to appear.

[47] G. Slivinskas, C. S. Jensen, and R. T. Snodgrass. Query Plans for Conventional and Temporal Queries Involving Duplicates and Ordering. In *Proceedings of the 16th International Conference on Data Engineering*, San Diego, California, February/March 2000, to appear.

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

[49] R. T. Snodgrass (ed.). *Proceedings of the International Workshop on an Infrastructure for Temporal Databases*. Arlington, TX, June 1993.

[50] R. T. Snodgrass. Temporal Object Oriented Databases: A Critical Comparison. [34, Ch. 19, pp. 386–408].

[51] R. T. Snodgrass (ed.), I. Ahn, G. Ariav, D. Batory, J. Clifford, C. E. Dyreson, R. Elmasri, F. Grandi, C. S. Jensen, W. Käfer, N. Kline, K. Kulkarni, T. Y. Leung, N. Lorentzos, J. F. Roddick, A. Segev, M. D. Soo, and S. M. Sripada. *The TSQL2 Temporal Query Language*. Kluwer Academic Publishers 1995. See also <`http://www.cs.arizona.edu/people/rts/sql3.html`>.

[52] R. T. Snodgrass, M. H. Böhlen, C. S. Jensen, and A. Steiner. *Adding Valid Time to SQL/Temporal*. ANSI X3H2-96-501r2, ISO/IEC JTC 1/SC 21/WG 3 DBL-MAD-146r2, November 1996.

[53] R. T. Snodgrass. Temporal Databases. [64, Part II].

[54] R. T. Snodgrass. *Developing Time-Oriented Database Applications in SQL*. Morgan Kaufmann Publishers 2000.

[55] M. D. Soo, R. T. Snodgrass, and C. S. Jensen. Efficient Evaluation of the Valid-Time Natural Join. In *Proceedings of the IEEE International Conference on Data Engineering*, pp. 282–292, Houston, Texas, February 1994.

[56] A. Tansel, J. Clifford, S. Gadia, S. Jajodia, A. Segev, and R. T. Snodgrass (eds.). *Temporal Databases: Theory, Design, and Implementation*. Benjamin/Cummings Publishers 1994.

[57] C. I. Theodoulidis and P. Loucopoulos. The Time Dimension in Conceptual Modelling. *Information Systems*, 16(3):273–300, 1991.

[58] V. J. Tsotras, C. S. Jensen, and R. T. Snodgrass. An Extensible Notation for Spatiotemporal Index Queries. *ACM SIGMOD Record*, 27(1):47–53, March 1998.

[59] K. Torp, C. S. Jensen, and R. T. Snodgrass. Stratum Approaches to Temporal DBMS Implementation. In *Proceedings of the 1998 International Database Engineering and Applications Symposium*, pp. 4–13, Cardiff, Wales, UK, July 1998.

[60] V. J. Tsotras and X. S. Wang. Temporal Databases. *Encyclopedia of Electrical and Electronics Engineering*, John Wiley and Sons, 1999.

[61] X. S. Wang, C. Bettini, A. Brodsky, and S. Jajodia. Logical Design for Temporal Databases with Multiple Granularities. *ACM Transactions on Database Systems*, 22(2):115–171, June 1997.

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

[63] Y. Wu, S. Jajodia, and X. S. Wang. Temporal Database Bibliography Update. [21, pp. 338–366].

[64] C. Zaniolo, S. Ceri, C. Faloutsos, R. T. Snodgrass, V. S. Subrahmanian, and R. Zicari. *Advanced Database Systems*. Morgan Kaufmann Publishers 1997.