Temporal Data Management

Christian S. Jensen and Richard T. Snodgrass

June 9, 1997

TR-17

Title	Temporal Data Management
	Copyright © 1997 Christian S. Jensen and Richard T. Snodgrass. All rights reserved.
Author(s)	Christian S. Jensen and Richard T. Snodgrass
Publication History	June 1997. A TIMECENTER Technical Report.

Abstract

A wide range of database applications manage time-varying information. Existing database technology currently provides little support for managing such data. The research area of temporal databases has made important contributions in characterizing the semantics of such information and in providing expressive and efficient means to model, store, and query temporal data. This paper introduces the reader to temporal data management, surveys state-of-the-art solutions to challenging aspects of temporal data management, and points to research directions.

Keywords: query language, temporal database, temporal data model, time-constrained database, transaction time, user-defined time, valid time

1 Introduction

A *temporal database* records time-varying information. Most database applications are temporal in nature, e.g., 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.

The study of temporal databases is a vibrant research topic, with an active community of several hundred researchers who have produced some 1600 papers over the last two decades. These papers are listed in a series of seven cumulative bibliographies (the last, [21] provides pointers to the previous ones). The field has produced a comprehensive glossary of terminology [7], a book-length survey providing a snapshot circa 1993 [20], and two workshop proceedings [3, 16]. The nascent SQL3 draft standard now includes Part 7, SQL/Temporal [12].

The present paper examines a variety of central areas of temporal database research. For each area, we first present the motivation for the research. Then, we survey sample contributions, to give the reader a feel for the type of challenges and issues that are faced in each particular area.

Given the space limitation, we cannot survey all areas, let alone all contributions, and the presentation must be brief. Thus, we have omitted a wide range of contributions that we consider important. A recent survey [13] and the slightly older book on temporal databases [20] go into more depth. We have also found it useful to focus on relational databases. The relational model is well-known, and its simplicity is conducive to maintaining an emphasis on the temporal essence of past research.

2 Ontological Foundations

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 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 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.

Our focus will be on the facts that databases record. Several different temporal aspects have been 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 [7]. Valid time thus captures the time-varying states of the mini-world. By definition, all facts have a valid time. However, the valid time 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 may be associated with objects and values that are not facts because they cannot be true or false in isolation. 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 (logical) deletion. Transaction time captures the time-varying states of the database, and applications with demands for 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 contents of all databases, and recording both of these is essential in a wide range of applications. In addition, transaction time is, due to its special semantics, particularly well-behaved and may be supplied automatically by the DBMS. Specifically, the transaction time of facts stored in the database marches monotonically forward, and is bounded at both ends, by the time the database was created and the current time. 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. But the desirability of building decision time support into temporal database technologies is limited, because the number and meaning of "the decision times" of a fact varies from application to application and because decision times, unlike transaction time, generally do not exhibit specialized properties.

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 to rather applied questions such as how to represent time values in minimal space and how to utilize calendars. Also, there is a large body of research on time data types, e.g., time points, time intervals (or "periods"), and temporal elements (sets of intervals).

3 Temporal Data Models

Temporal data management is very difficult using conventional (non-temporal) data models and query languages. 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.

The first step to provide support for temporal data management is to extend the database structures of the data model supported by the DBMS to become temporal. More specifically, means must be given for capturing the valid and transaction times of the facts recorded by the relations, leading to temporal relations.

Subsequent steps are to provide support for temporal data modeling and database design, and to design temporal query languages that operate on the databases of the temporal data models. These topics are covered in Sections 4 and 5, respectively.

Adding time to the relational model, then, has been a daunting task, and more than two dozen extended relational data models have been proposed [8]. 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 `CustomerID`s, rent video tapes, identified by `TapeNum`s. We consider a few rentals during May 1997. On the 2nd, customer C101 rents tape T1234 for three days. The tape is subsequently returned on the 5th. Also on the 5th, customer C102 rents tape T1245 with an open-ended return date. The tape is eventually returned on the 8th. On the 9th, customer C102 rents tape T1234 to be returned on the 12th. On the 10th, the rental period is extended to include the 13th, but this tape is not returned until the 16th. The video store keeps a record of these rentals in a relation termed `CheckedOut`.

Figure 1 illustrates a relation instance in the Bitemporal Conceptual Data Model (BCDM) [8] that corresponds to the sample rental scenario. To the right, there is a graphical illustration of the three timestamps. The tuples correspond to facts and are timestamped with bitemporal elements, which are finite unions of intervals or, equivalently, sets of time points in the (finite and discrete) two-dimensional space spanned by valid and transaction time.

The timestamp of the second tuple is explained as follows. On the 5th, it is believed that customer C102 has checked out tape T1245 on the 5th. Then, on the 6th, the rental period is believed to include the 5th and the 6th. On the 7th, the rental period extends to also include the 7th. From then on, the rental period remains fixed. The current time is the 17th, and as this time increases, 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 allowing for the capture of 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 single 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.

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

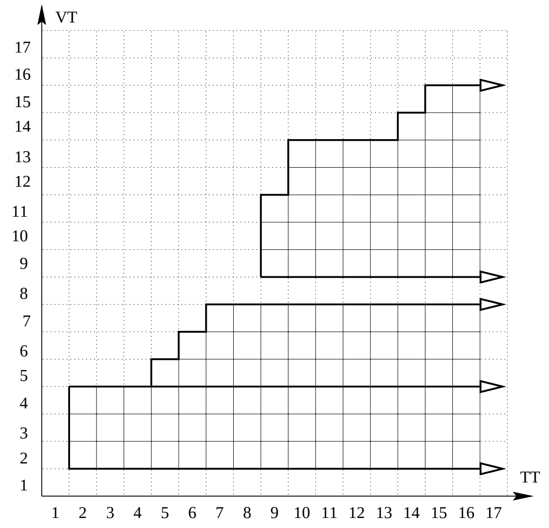

Figure 1: Bitemporal Conceptual `CheckedOut` Instance

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 non-fixed-length and voluminous timestamps of tuples are impractical to manage directly, and the timestamp values are also hard to comprehend in the BCDM format. Alternative representations of temporal information may be better suited for these purposes.

Figure 2 illustrates the same temporal information as in Figure 1, in two different data models. The model exemplified to the left uses a practical and popular (particularly when implementation is considered) fixed-length format for tuples. In this format, each tuple's timestamp records a rectangular or stair-shaped region of times, and it may take several tuples to represent a single fact. The relation format to the right in Figure 2 is a typical non-1NF format. In this format, a relation is thought of as recording information about some type 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. For 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 the 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. 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.

4 Designing Temporal Databases

Database design is typically considered in two contexts. In conceptual design, a database is modeled using a high-level design model that is independent of the particular (implementation) data model of the DBMS that is eventually to be used for managing the database. The second context of database design is the implementation data model,

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

CustomerID		TapeNum	
$[2,\textit{Now}] \times [2,4]$	C101	$[2,\textit{Now}] \times [2,4]$	T1234
$[5,7] \times [5,\infty]$	C102	$[5,7] \times [5,\infty]$	T1245
$[8,\textit{Now}] \times [5,7]$		$[8,\textit{Now}] \times [5,7]$	
$[9,9] \times [9,11]$		$[9,9] \times [9,11]$	T1234
$[10,13] \times [9,13]$		$[10,13] \times [9,13]$	
$[14,15] \times [9,\infty]$		$[14,15] \times [9,\infty]$	
$[16,\textit{Now}] \times [9,15]$		$[16,\textit{Now}] \times [9,15]$	

Figure 2: Alternative Representations of the `CheckedOut` Instance

which is assumed to conform to the ANSI/X3/SPARC three-level architecture. In this context, database design must thus be considered at the view level, the logical level (originally termed "conceptual"), and the physical (or, "internal") level. We proceed to consider conceptual and logical design of temporal databases.

4.1 Conceptual Design

By far, most research on 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, Figure 3 illustrates a conventional ER diagram for video rentals.

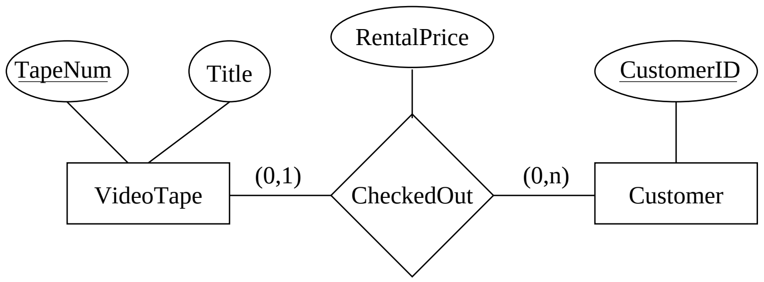

Figure 3: Non-temporal Conventional ER Diagram for Video Rentals

The research on temporal ER modeling is well motivated. It is widely known that the temporal aspects of the mini-world (we use "mini-world" to denote the part of reality that the database stores information about) are very important in a broad range of applications, but are also difficult to capture using the ER model. Put simply, when attempting to capture the temporal aspects, these tend to obscure and clutter otherwise intuitive and easy-to-comprehend diagrams.

The diagram in the figure is non-temporal, capturing the mini-world at a single point in time. Attempting to capture the temporal aspects that are essential for this application clutters up the simple diagram. For example, since the same customer may check 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, with the third entity type capturing start dates of rentals. There is also the issue of where to place the end-time attribute of rentals. Next, rental prices may vary over time, e.g., due to promotions and films getting old. Finally, including also transaction time complicates matters.

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 well the temporally extended relational database schemas used by the application programmers.

The research community's response has been to develop temporally enhanced ER models. Indeed, almost a dozen such models have been reported in the research literature [5]. 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 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. Another approach is to change the semantics of the existing ER model constructs, making them temporal. In its extreme form, this approach does not result in any new syntactical constructs—all the original constructs have simply become temporal. With this approach, it is also possible to add new constructs.

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 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 proposed [13] 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 is a challenge to design mappings that maximally exploit these and other candidate implementation platforms.

4.2 Logical Design

A central goal of conventional relational database design is to produce a database schema, consisting of a set of *relation schemas*. Normal forms constitute an attempt at characterizing "good" relation schemas. 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.

The existing normalization concepts are not applicable to temporal relational data models because these models employ relational structures that are different from conventional relations. There is thus a need for new temporal normal forms and underlying concepts that may serve as important guidelines during temporal database design.

In response to this need, an array of temporal normalization concepts have been proposed [9], including temporal dependencies, keys, and normal forms. Consider the `CheckedOut` relation schema from Section 3, as exemplified in Figures 1 and 2. Does `CustomerID` (temporally) determine `TapeNum` or vice versa? Looking at the first representation in Figure 2 and applying conventional dependencies directly, the answer to both questions is no. 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.

Stepping back, it should be that the same dependencies hold for the `CheckedOut` relation independently of how it is represented. And 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. This notion of dependency naturally generalizes conventional dependencies and may be applied to other dependencies than functional. With this notion of dependency, a temporal normalization theory may be built that parallels conventional normalization theory and that is independent of any particular representation of a temporal relation.

5 Adding Time to Query Languages

Given the prevalence of applications that currently manage time-varying data, one might ask why a temporal query language is even needed. Is the existence of all this SQL code 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.

In addition to the `CheckedOut` relation from Section 3, we assume in this section a `VideoTape` relation with attributes `TapeNum`, `Title`, and `RentalPrice`. Consider first this database with only current information. To determine who has checked out which titles, SQL provides a natural solution.

```
SELECT CustomerID, Title FROM CheckedOut, VideoTape
WHERE CheckedOut.TapeNum = VideoTape.TapeNum
```

We then extend the `VideoTape` and `CheckedOut` relations to record also past and future states by adding to each relation two additional attributes, `StartDate` and `EndDate`, specifying the interval of validity of the tuples. To request the *history* of who checked out which titles requires 25(!) lines of SQL: four `SELECT` statements, `UNION`ed together, performing a case analysis of how the interval of validity of `CheckedOut` overlaps the interval of validity of `VideoTape`.

As another example, referential integrity on the non-temporal relations is trivial in SQL: "`CONSTRAINT TapeNum REFERENCES VideoTape`." When the two relations are time-varying, referential integrity requires a 28-line SQL assertion, with triply-nested `EXISTS`/`NOT EXISTS` subqueries. (Readers are encouraged to try their hand at these two examples.) Ordinary queries on the non-temporal relations become extremely challenging when timestamp attributes are added. Even SQL experts would be hard pressed to express the following in SQL: what is the history of the average rental price for checked out video tapes?

Some 40 temporal query languages have been defined [22], most with their own data model. The most recent is TSQL2 [17], 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 [11] upon which future research could be based.

With a temporal query language, simple queries should remain simple when time is added. The temporal join can be expressed in the variant of TSQL2 being proposed for inclusion into SQL3 [18] as follows.

```
VALIDTIME SELECT CustomerID, Title
FROM CheckedOut, VideoTape
WHERE CheckedOut.TapeNum = VideoTape.TapeNum
```

Similarly, referential integrity can be expressed as "`CONSTRAINT TapeNum VALIDTIME REFERENCES VideoTape`." Even with this minimal explanation, the reader should have no difficulty in expressing the average rental price query in this extension to SQL.

Early query languages were based on the relational algebra. Calculus-based, Datalog-based, and object-oriented temporal query languages 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 have analogues in temporal query languages. As one example, whether the data model timestamps tuples or attribute values influences the language. The history of who checked out which titles can be expressed in the TempSQL [4] query language, which utilizes attribute-value timestamps, as shown in Figure 2.

```
SELECT CustomerID, Title
WHILE ⟦CheckedOut.TapeNum = VideoTape.TapeNum⟧ ∩ [Now, Now] × [0, ∞]
FROM CheckedOut, VideoTape
```

Here, the `WHILE` construct restricts the time domain of the resulting attributes to those times when the equality was satisfied. The intersection ensures we only examine the data that is current in the database, that is, data with a transaction time of now.

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 one (or 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 description of TSQL2 required 700 pages [17].

Recently a set of criteria for temporal query languages has emerged. These include *temporal upward compatibility* (that is, conventional queries and modifications on temporal relations should act on the current state), support for *sequenced queries* (that request the history of something, such as the temporal join above), adequate expressive power (a query language-independent test suite [6] is useful for such evaluations), and the ability to be efficiently implemented.

6 Temporal DBMS Implementation

There has been a vast amount of work in storage structures and access methods for temporal data, as well as a dozen-odd temporal DBMS prototypes [1]. There have been two basic approaches. Most authors assume an *integrated* approach, in which the internal modules of a DBMS are modified or extended to support time-varying data. More recently, there has been work using a *stratum* approach, in which a layer converts temporal query language statements into conventional statements executed by an underlying DBMS, which is itself not altered. While the former approach ensures maximum efficiency, the latter approach is more realistic in the medium term. In the following we will, consistent with the vast majority of papers on temporal DBMS implementation, assume an integrated approach, utilizing timestamping of tuples with time intervals.

6.1 Query Processing

Optimization of temporal queries is more involved than that of conventional queries, for several reasons. First, the relations that temporal queries are defined over are often larger; this justifies trying harder to optimize the queries, and spending more execution time to perform the optimization. Second, the predicates used in temporal queries are harder to optimize. In traditional database applications, predicates are usually equality predicates (hence the prevalence of equijoins and natural joins). In temporal queries, conjunctions of inequality predicates appear more frequently. As an example, the TSQL2 temporal join query given in Section 5 determines the overlap between validity intervals from the `CheckedOut` relation and the `VideoTape` relation. This overlap is translated into two "$\leq$" predicates on the underlying timestamps, as follows.

```
BEGIN(CheckedOut) <= END(VideoTape) AND BEGIN(VideoTape) <= END(CheckedOut)
```

Conventional DBMSs focus on equality predicates and often implement inequality joins as Cartesian products, with their associated inefficiency.

There is greater opportunity for query optimization when time is present. Time advances in one direction; the time domain is continuously expanding, and, for transaction time, the most recent time point is the largest value in the domain. This implies that a natural clustering or sort order will manifest itself, which can be exploited during query optimization and evaluation. The integrity constraint `BEGIN(`i`) <= END(`i`)` holds for every interval i. Also, for many relations, the intevals associated with a key are contiguous in time, with one interval starting exactly when the previous interval ended (an example is the `VideoTape` relation). Semantic query optimization can exploit these integrity constraints, as well as additional ones that can be inferred.

6.2 Implementing Algebraic Operators

Attention has been directed at the common (and often expensive) temporal algebraic operators: selection, joins, aggregates, and duplicate elimination. We examine selection in Section 6.3, on temporal indexes.

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*. 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 hash joins ([19] surveys these algorithms). 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 [10]. Finally, *Coalescing* is an important operation in temporal databases [2]. Coalescing merges value-equivalent tuples with intervals that overlap or meet. 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.

6.3 Indexing Temporal Data

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 selection algebraic operator and temporal joins. Indexes are even more important in temporal relations due to their size. Many temporal indexing strategies are available [15]. 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, update per change, and several important 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, coalescing, and aggregates—this is still an area of active investigation.

7 Summary

This paper has briefly introduced the reader to temporal data management, emphasizing what we believe are important concepts and surveying important results produced by the research community. In what remains, we first summarize the current state-of-the-art, then point to issues that remain challenges and which require further attention.

A great amount of research has been expended on temporal data models and query languages, which has shown itself as an extraordinarily complex challenge with subtle issues. We feel that the semantics of standard temporal relational schemas and their logical design are well understood, and the Bitemporal Conceptual Data Model is gaining acceptance as the appropriate model in which to consider data semantics.

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. Constructs from that language are being incorporated into a new part of SQL3, called SQL/Temporal.

The semantics of the time domain, including its structure, dimensionality, and indeterminacy, is 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.

While many important insights and results have been reported, there are still many *research challenges*. First, the conceptual and physical database design of temporal schemas are still in their infancy. In the past, such investigation has been hindered by the plethora of temporal data models. Concerning performance, more empirical studies are needed to compare temporal algebraic operator implementations, and to possibly suggest even more efficient implementations. While preliminary performance studies have been carried out for each 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.

Next, 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, in part because of this and despite the obvious need in the marketplace, there is as yet no prominent commercial temporal relational DBMS.

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 user interfaces for commercially available DBMSs. Finally, little has been done in integrating spatial, temporal, and active data models, query languages, and implementation techniques.

Acknowledgements

This research was supported in part by the National Science Foundation through grants IRI-9632569 and ISI-9202244, the Danish Natural Science and Technical Research Councils through grants 9400911 and 9502695, and by the CHOROCHRONOS project, funded by the European Commission DG XII Science, Research and Development, as a Networks Activity of the Training and Mobility of Researchers Programme, contract no. FMRX-CT96-0056.

References

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

[2] 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.

[3] 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, Berlin, 1995.

[4] S. K. Gadia and G. Bargava. SQL-like Seamless Query of Temporal Data, in [16].

[5] H. Gregersen and C. S. Jensen. *Temporal Entity-Relationship Models—A Survey*. Technical Report R-96-2039, Aalborg University, Department of Mathematics and Computer Science, September 1996.

[6] C. S. Jensen (editor), J. Clifford, S.K. Gadia, F. Grandi, P.P. Kalua, N. Kline, N. Lorentzos, Y. Mitsopoulos, A. Montanari, S.S. Nair, E. Peressi, B. Pernici, E.L. Robertson, J.F. Roddick, N.L. Sarda, M.R. Scalas, A. Segev, R.T. Snodgrass, A. Tansel, R. Tiberio, A. Tuzhilin, and G.T.J. Wuu. *A Consensus Test Suite of Temporal Database Queries*. Technical Report R 93-2034, Dept. of Mathematics and Computer Science, Aalborg University, Denmark, November, 1993, 45 pages.

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

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

[9] 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.

[10] N. Kline and R. T. Snodgrass. Computing Temporal Aggregates. In *Proceedings of the IEEE International Conference on Database Engineering*, Taipei, Taiwan, March 1995.

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

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

[13] 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.

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

[15] B. Salzberg and V. J. Tsotras. A Comparison of Access Methods for Time Evolving Data. *ACM Computing Surveys*, to appear, 1997.

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

[17] 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.

[18] 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.

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

[20] A. Tansel, J. Clifford, S. Gadia, S. Jajodia, A. Segev and R. T. Snodgrass (eds.). *Temporal Databases: Theory, Design, and Implementation*. Database Systems and Applications Series. Benjamin/Cummings, Redwood City, CA, 1994.

[21] V. J. Tsotras and A. Kumar. Temporal Database Bibliography Update. *ACM SIGMOD Record*, 25(1):41–51, March, 1996.

[22] R. T. Snodgrass. Temporal Databases. Part II of *Advanced Database Systems*, Morgan Kaufmann Publishers, Inc., San Francisco, CA, 1997.