International Journal of Multidisciplinary and Current Research ISSN: 2321-3124
Research Article Available at: http://ijmcr.com

Temporal Data Management

Sameer*

Shah Satnam Ji P.G Boys College, Sirsa, India

Received 05 June 2017, Accepted 20 Aug 2017, Available online 25 Aug 2017, **Vol.5 (July/Aug 2017 issue)**

Abstract

A temporal database records time-varying information. Most database applications are temporal in nature, e.g., scheduling applications such as airline, train, and hotel reservations and project management, and scientific applications such as weather monitoring, financial applications such as portfolio management, accounting, and banking, record-keeping applications such as personnel, medical-record, and inventory management,. Recently there has been a surge of interest in temporal data, because memory and magnetic disk storage costs are rapidly decreasing, and the advances in optical disk technology. In the past, temporal data was mostly delegated to archival storage or discarded altogether because it was too expensive or impractical to access them on-line. While it was recognized that historical data are of great importance to applications such as data analysis for policy decisions, such applications were not viewed as essential for a day-to-day operation. As a result, existing data management systems are designed to support the view of the most current version of the database. The dominant approach is one of data being updated, deleted, and inserted in order to maintain the current version. 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.

Keywords: *Temporal database records etc.*

1. Introduction

A temporal database records information that varies with time. Most database applications are temporal in nature. Time is one of the main aspects characterizing several real world facets and phenomena. The ability to model the temporal dimension of the real world and to respond within time constraints to changes in the real world as well as to application-dependent operations is essential to many computer applications.

In the context of database research, the management of time has been studied extensively in the last decades. In particular, many efforts have been devoted to add time support to database models and system functionalities. Temporal database systems provide special facilities for storing, querying, and updating historical and/or future data.

In this context, two time dimensions are usually considered: legal time and dealing time. Legal time (sometimes referred to as valid time) is the real world time. It refers the time a fact is true in the real world. Dealing time (also known as transaction time) is the system time and it denotes the time during when the fact is stored in the database.

*Corresponding author's ORCID ID: 0000-0000-0000-0000
DOI: https://doi.org/10.14741/ijmcr/v.5.4.21

In general a temporal value is actually a triplet (s, t, v) where s, t, and v stand for surrogate, time, and value respectively. Thus, the triplet (Rahul, April, 23) may represent Rahul's age in April drawn from the space (name X month X age). Surrogates may be either taken from a defined domain, such as "name" or may be assigned automatically by the system. The important thing is that they uniquely identify each element. A collection of values for Rahul over time will have the same surrogate, and thus it can be represented as (s, (t, v)*). (t, v)* ,that represents an ordered-sequence of pairs of times and their associated values. Thus, we may think of temporal data as time ordered sequences of pairs (t, v) for each surrogate. Each sequence for a single surrogate is known as a time sequence (TS).

2. Temporal Data Semantics

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.

In this context, two time dimensions are usually considered: legal time and dealing time. The legal time also known as 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 . 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 Dealing time also known as Transaction time of a database fact is the time when the fact is current in the database. Unlike legal time, dealing time may be associated with any database entity, not only with facts. For example, dealing 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 dealing time with "63," but not legal time. Thus, all database entities have a dealing-time aspect. This aspect may or may not, at the database designer's discretion, be captured in the database. The dealing-time aspect of a database entity has a duration: from insertion to deletion, with multiple insertions and deletions being possible for the same entity.

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.

3. Temporal data model

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

As a simple example, consider a book library where customers, identified by CustomerIDs, rent books identified by BookIds. We consider a few rents during May 2007. On the 6th, customer C101 rents book B1001 for three days. The book is subsequently returned on the 9th. Also on the 10th, customer C102 rents book B1002 with an open-ended return date. The book is eventually returned on the 14th. On the 15th, customer C103 rents book B1005 to be returned on the 18th. On the 16th, the rental period is extended to include the 19th, but this book is not returned until the 21st. The library keeps a record of these rentals in a relation termed Rented.

Figure 1 gives the relation instance in the Bitemporal Conceptual Data Model (BCDM) that describes the sample rental scenario. This data model timestamps tuples, corresponding to facts, with values that are sets of (dealing time, legal time) pairs, captured using attribute T in the figure.

The presence of a pair (dt, lt) in a timestamp of a tuple means that the current state of the database at time dt records that the fact represented by the tuple is valid at time lt.

Customer Id	BookId	T
C101	B1001	{(6,6),(6,7),(6,8),(7,6),(7,7),(7,8),- - - - - - - -, (UC,6),(UC,7),(UC,8)}
C102	B1002	{(10,10),(11,10),(11,11),(12,10),(12,11),(12,12),(13,10),(13,11),(13,12),(13,13),(14,10),(14,11),(14,12),- - - - - - - - - - - - - (UC,10),(UC,11),(UC,12),(UC,13)}
C103	B1005	{(15,15),(15,16),(15,17),(16,15),(16,16),(16,17),(16,18),(16,19),- - - - - - - ,(19,15),(19,16),(19,17),(19,18),(19,19),(20,15),(20,16), (20,167,(20,18)- - - - - - , (20,20), (21,15),- - - - - - - - (21,20),- - - - - - - - - - -, (UC,15)- - -(UC,20)}

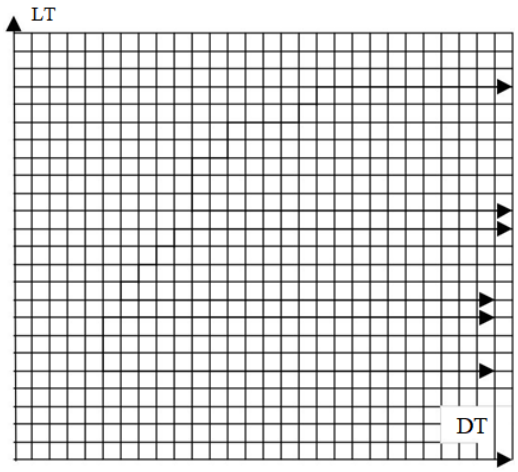
Fig: 1: Bitemporal conceptual Rental Instance

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 5th, it is believed that customer C102 has checked out tape T1002 on the 10th. Then, on the 11th, the rental period is believed to include the 10th and the 11th. On the 12th, the rental period extends to also include the 12th. From then on, the rental period remains fixed. The current time is the 21st, and as time passes, the region grows to the right; the arrows indicate this and correspond to the UC values in the textual representation.

Figure 2 shows a graphical illustration of the three timestamps. The tuples correspond to facts and are times tamped 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 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.



However, in case of 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. Thus we required alternative representations of temporal information that may be better suited for these purposes.

Figure 3 illustrates the same temporal information as in Figure 1, in two different data models. The model illustrated in 3(a) 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 in Figure 3(b) 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. Attributes D*s* and D*e* record starting and ending dealing times, and L*s* and L*e* record starting and ending legal times

Customer Id	BookId	Ds	De	Ls	Le
C101	B1001	6	UC	6	8
C102	B1002	10	13	10	now
C102	B1002	14	UC	10	13
C103	B1005	15	15	15	17
C103	B1005	16	18	15	18
C103	B1005	19	20	15	now
C103	B1005	21	UC	15	20

(a)

CustomerId		BookId	
[6,Now]×[6,8]	C101	[6,Now]×[6,8]	B1001
[10,13]×[10,∞] [14,Now]×[10,13]	C102	[10,13]×[10,∞] [14,Now]×[10,13]	B1002
[15,15]×[15,17] [16,18]×[15,18] [19,20]×[15,∞] [21,Now]×[15,20]	C103	[15,15]×[15,17] [16,18]×[15,18] [19,20]×[15,∞] [21,Now]×[15,20]	B1005

(b)

Fig 3: Alternative Representation of Rented Instance

Unlike in the BCDM, where relations must be updated at every clock tick, relations in this format 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 relation 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).

4. Designing Temporal Database

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.

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, which is assumed to conform to the ANSI 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. Here we consider conceptual and logical design of temporal databases. 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.

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 4 illustrates a conventional ER diagram for video rentals.

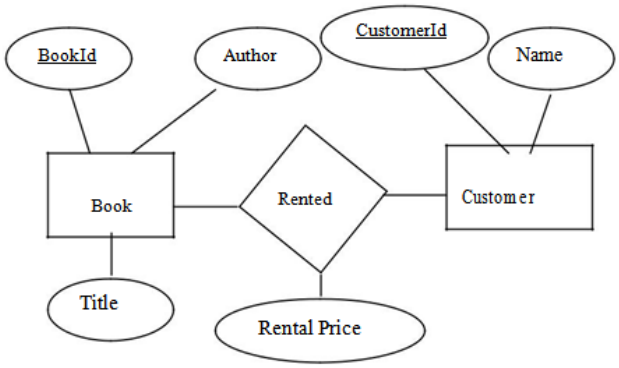


Fig. 4: Non Conventional ER Diagram of Book Rental

The diagram in the figure 4 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 BookId attributes do not identify a single instance of Rented. Instead, it is necessary to make Rented 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 publications and books getting old. Finally, inclusion of dealing 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.

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 (e.g., the Rental Price in our example). With this approach, it is possible to retain the existing ER-model constructs with their old semantics. 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.

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 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.

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.

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, an array of temporal normalization concepts have been proposed including temporal dependencies, keys, and normal forms. Consider the Rented relation schema from Section 3, as illustrated in Figures 1 and Does Customer ID (temporally) determine BookId or vice versa? Looking at the first representation in Figure 3 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.

Considering that the different representations of the Rented relation model the same mini world 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 books. In contrast, a book can only be checked out by a single customer at a single point in time. With this view, BookId temporally determines CustomerID, but the reverse does not hold.

Temporal data models generally define time slice 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 time slice operator for temporal relations like the one in Figure 1 may take a point (dt, lt) in bitemporal space as its parameter. It returns the tuples of the

argument relation that contain this time point, but omitting the timestamp attribute.

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.

It is also relevant to consider dependencies and associated normal forms that effectively hold between time points. 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. 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.

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

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. To illustrate the issue, consider the two relations S-Rented and V-Rented 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.

CustomerId	BookId
C101	B1001
C102	B1003
C102	B1002
C103	B1005

CustomerId	BookId	Ls	Le
C101	B1001	2	Now
C101	B1003	5	10
C102	B1002	22	25
C102	B1002	9	19
C102	B1005	4	14
C102	B1006	9	Now
C103	B1004	7	21

Fig 5: Realations (a) S-Rented and (b) V-Rented

Using SQL, it is straightforward to express the number of current checkouts from S-Rented. For example, this can be expressed as follows.

SELECT COUNT(bookid) AS CNT From S-Rented

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-Rented. The result given in Figure 6 correctly gives the count of books rented at each point in time were a book is rented (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.

CNT	Ls	Le
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

Fig 6: A Time-Varying Count on the BookId Attribute of V-Rented

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

Alter Table S-Rented Add Primary Key (BOOKID)

This key constraint may be generalized to apply to a legal-time relation, now meaning that BookId is a key at each point in time or, equivalently, in each snapshot that may be produced from the legal-time table. Specifying this constraint on relation V-Rented in SQL is again difficult.

Ordinary queries on non-temporal relations become extremely challenging when timestamp attributes are added.

In an another example in addition to the Rented relation from Section 3, we assume in this section a Book relation with attributes BookId, Title, and RentalPrice. Consider first this database with only current information.

To determine who has rented which titles, SQL provides a natural solution. SELECT CustomerID, Title FROM Rented, Book

WHERE Rented.BookId = Book.BookId

We then extend the Book and Rented 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 huge size of SQL which is very difficult to execute.

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 as follows.

VALIDTIME SELECT CustomerID, Title FROM Rented, Book
WHERE Rented.BookId = Book.BookId

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

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 is useful for such evaluations), and the ability to be efficiently implemented.

Conclusion

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 paper has briefly introduced the concept of 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 conducted on temporal data models and query languages, which has shown itself to be an extraordinarily complex challenge with subtle issues. Many languages have been proposed for querying temporal databases, half of which have a formal basis.

Although many important insights and results have been reported, many research challenges still remain in temporal database management. 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.

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 technology 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. 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.

Reference

[1] Developing Time-Oriented Database Applications In Sql, Richard T. Snodgrass, Morgan Kaufman Publishers.
[2] Introduction to Temporal Database, Christian S. Jensen
[3] Jensen, C. S. ; R. T. Snodgrass(1996), "Semantics of Time-Varying Information," *Information Systems*, Vol. 21, No. 4, pp. 311–352.
[4] Clifford, J., C. Dyreson, ;T. Isakowitz, Jensen, C. S. and R. T. Snodgrass(1997), "On the Semantics of "Now" in Databases," *ACM Transactions on Database Systems*, Vol. 22, No. 2, June pp. 171–214.
[5] 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*,
[6] Jensen, C. S.,M. D. Soo, ;R. T. Snodgrass(1994), "Unifying Temporal DataModels via a Conceptual Model," *Information Systems*, Vol. 19, No. 7, pp. 513–547.
[7] N. Kline ; R. T. Snodgrass.(March 1995) Computing Temporal Aggregates. In *Proceedings of the IEEE International Conference on Database Engineering*, Taipei, Taiwan.
[8] A. Tansel, J. Clifford, S. Gadia, S. Jajodia, A. Segev and R. T. Snodgrass (eds.). *TemporalDatabases: Theory, Design, and Implementation*.
[9] 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*,