



What is a Temporal Database?

by Chris Klassen

Time has always been of great interest to mankind. Much has been written about time from enormously complex philosophical manuscripts to farmers' almanacs. The most famous square in New York City is named Time's Square and a worldwide news magazine is titled Time. Not surprisingly, increasing interest in the dimension of time is being expressed by researchers working in the field of databases. The classical database is two-dimensional, and contains only data that is current, a type we will call snapshot data type. While this two-dimensional database model was adequate for older business needs, today it has become out-moded and insufficient. Today's businesses must constantly adapt to an ever-changing business environment, and their databases must support this evolving business framework. The old saying that 'time is money' is now more true than ever. More and more time-management seminars and devices are introduced everyday. As Richard Snodgrass notes "Time varying data is becoming pervasive. It has been estimated that one of every 50 lines of database application code involves a date or time value" [Snodgrass 1998]. The biggest problem in the business computer industry today, the Year 2000 problem (Y2K), is time-related. In the database field, the business solution to the lack of time support is the temporal database.

I. What is a Temporal Database?

We will begin by considering the general question of what a temporal database is. Here is an image that may prove helpful at the outset. The classical database is two-dimensional with columns and rows which intersect each other at cells which contain particular values. Now extend this flat two-dimensional database into a three-dimensional figure such as a cube. When you apply this concept to a database, instead of a flat two-dimensional figure you now have an extended three-dimensional figure with the third dimension being represented as various time intervals [Tansel, et al. 1993].

Roughly speaking, a temporal database is a database that records time-varying information. In my research I have found two different but similar precise definitions of the temporal database. The first comes from Richard Snodgrass. "The official definition of temporal database is 'a database that supports some aspect of time, not counting user-defined time'" [Snodgrass 1998]. Another, more technical definition, though, is given by Shamkant Navathe and Rafi Ahmed. According to them, a temporal database is defined as "a union of two sets of relations R_s and R_t , where R_s is the set of all static relations and R_t is the set of all time-varying relations" [Navathe and Ahmed 1993]. Both definitions are correct, although Navathe and Ahmed's may be more structured to fit their Temporal Relational Model which they have outlined in Temporal Databases: Theory, Design, and Implementation [Navathe and Ahmed 1993]. One caveat with respect to this paper should be noted: due to space limitations, I will focus only on the relational model of the temporal database, to the exclusion of the other well-known types of databases such as object-oriented, network, and hierarchical. Little temporal database research is being done with respect to the last two types of databases, but an increasing amount of research is being spent in the area of temporal object-oriented databases. This corresponds to the general trend of increasing interest in object-oriented databases. The caveat being noted, for the purposes of this paper, we will adopt the simpler definition of the temporal database as proposed by Snodgrass. We might also note that temporal databases have also been referred to as time-oriented databases, time-varying databases, or historical databases. While "time-oriented database" and "time-varying database" are equivalent in meaning to "temporal database", "historical database" is not. As we will see later in this paper, an "historical database" is actually a subset of the temporal database.

The first thing we note about Snodgrass' definition of the temporal database is that the time dimension requirement for a database to be temporal does not include "user-defined" time. What is "user-defined" time, and what is its relationship to the temporal database?

"User-defined time" is some aspect of time which is not recognized by the database management system as being a special data type. "User-defined time is defined as 'an uninterpreted attribute domain of date and time... parallel to domains such as 'money' and integer'" [Snodgrass 1998]. An example of "user-defined" time is data such as birthdates. A classical database essentially treats data such as birthdates as text strings. This treatment of date data does not allow for much manipulation of the data

when performing ad hoc queries on the database. One way of attempting to avoid this problem of treating the date as simply a text type of data is to store the date as a number. One example of such an approach is found in at least some versions of Oracle. The Oracle 7 version converts the date data type into a number which makes it easier to perform calculations on the date. This approach, however, is not without its problems. A problem I've come across in the Oracle version 7.2 is that at least some year 2000 dates are treated as less than dates in the 1900's. Internally converting dates to numbers, though, may be the best way of making possible a wide range of ad hoc queries on temporal data.

The fact that support of "user-defined" time does not merit a database being considered temporal, does not mean that the temporal database cannot or will not include "user-defined" time. Both classical and temporal databases do in fact support "user-defined" time, but whereas in classical databases "user-defined" time is the only "temporal" aspect, in temporal databases "user-defined" time is a small subset of the range of temporal datatypes.

II. Key Concepts for Understanding the Temporal Database

There are various types of time in a temporal database, but before we describe in detail each of the types of time, we must clarify one particular concept regarding the precision of the times stored in a temporal database. The concept denoting the precision in a temporal database is called the granularity of the time. Granularity is the duration of the smallest unit of time stored in the temporal database [Howe 1997]. Examples of different time grains are one day, one hour, or one second.

We can now turn to the three major concepts which must be understood if one is grasp what a temporal database is. The first is the concept of valid time. Besides user-defined time, the temporal database supports both transaction time and valid time. Transaction time is the actual time recorded in the database at the time the data is entered. This data is known as the time stamp. "A timestamp is a time value associated with some timestamped object, for example an attribute value or a tuple" [Jensen, et al. 1993]. Time stamps can include the date only or the date and clock time. The transaction time which contains the time stamp is useful for maintaining the data and for allowing different versions of the same objects. By object here I mean a particular thing about which information is being kept. An example of an object is an employee. In a classical database, once a change is made to an employee's record, the original data which was changed is discarded and replaced by the new data. In a temporal database which supports transaction time, however, the transaction time can be attached in the form of a time stamp to both the old data for that employee and to the new data for that employee. In doing so, the database can store both the old data and the new data for the same object, in this case, the employee whose salary was increased on a certain date. One thing to note here is that the transaction time values or time stamps cannot be after the current point in time. In addition, they cannot be changed just as the past cannot be changed.

The other major type of time in a temporal database is called valid time. Valid time is the actual or real world time at which point the data is valid. Let us continue with our employee example taken from the previous paragraph. Whereas transaction time is the point in time in which the data is entered into the database, valid time is when the data which was entered becomes true or takes effect. On January 3rd, our employee is notified that he will receive a raise effective February 1st, 1998. The HR department, after being notified of our employee's raise, must then enter the new salary into the database. Presumably they will do this before the raise goes into effect. The actual time which they enter the raise into the database will then be prior to February 1st, 1998, and that time will be the time stamp for the transaction time. The data, however, is not yet valid. For the rest of January, our employee will continue to receive his old salary. Then, on February 1st, the raise data will now become valid. Thus, February 1st is the valid time. Although the transaction time may be the same as the valid time, in most situations this will not be the case. Also, if an employee receives a raise which is retroactive, the transaction time may be later than the valid time.

The possibility for other types of time besides user-defined, transaction, and valid remains. Another possible type of time might be decision time. Decision time in our example would be the time at which the manager decided to give our employee his raise. This, too, could be stored in the temporal database.

The two major types of time unique to the temporal database, transaction time and valid time, allow for the possibility of three forms of temporal databases: Historical, Rollback, and Bitemporal [Steiner]. An historical database is one which supports valid time, but not transaction time. Recall earlier that we said that historical database was a poor choice as an alternative term for the temporal database. The reason we now see is that an historical database is actually one type of temporal database. An historical database, though, for reasons we will soon make clear, would be a poor choice for anyone wishing to deploy a temporal database.

The second form of a temporal database is the rollback database. This database is the opposite of the historical database. The rollback database only supports transaction and not valid time. As opposed to an historical database, a rollback database is quite useful for data recovery after database failure. The reason, then, that a temporal database which is simply historical would rarely be desirable is that it could not support rollback after DBMS failure. A rollback database is also necessary if the database does not use the

locking technique to ensure data security. Most databases on the market today do support at least some rollback features.

The truly temporal database is the bitemporal database. This database supports both transaction time, and valid time, so it is a combination of the historical and the rollback forms of the temporal database. The bitemporal database supports both of the types of time necessary for storing and querying time-varying data. It is the bitemporal database which will significantly aid in knowledge discovery because only the bitemporal database is able to fully support the time dimension; on the DBMS level with transaction time, on the data level with valid time, and on the user-level with user-defined time.

III. Major Datatypes in a Temporal Database

A temporal database supports three major datatypes: temporal data, static data, and snapshot data. One element of importance, though, is that the datatypes should be transparent to the user. The user should not need to know whether he/she is working with temporal, static, or snapshot data. One goal of the temporal database should be to provide seamless handling of temporal, static, and snapshot data. To this end, the temporal database should treat all data as some form of temporal data [Gadia and Nair 1993].

The temporal datatype is the most important for the temporal database. Shashi Gadia and Sunil S. Nair identify the temporal datatype as "a finite union of intervals" [Gadia and Nair1993]. They call this datatype a "temporal element". Thus, an example of a temporal element is (May 1, 1998-May 10, 1998) U (June 2, 1998- July 3, 1998). Each interval must have a start time and an end time, although the end time does not need to be definite. Until the end time is definite it may be specified by the letters 'INF' or 'NOW'. The temporal datatype or element provides the foundation upon which the temporal database can be built. The temporal element is necessary if one wishes to use time-varying criteria in performing ad hoc queries. Later in this paper, I will give an example of a temporal querying language which uses the temporal element in a new temporal clause which is added to the existing SQL syntax.

Static datatype is defined as "a constant defined over the whole universe of time" [Gadia and Nair 1993]. In other words, the validity of a static value is defined as forever. In contrast, a temporal datatype value is valid for a specified time period or interval, and a snapshot datatype value is only valid for the current instant (NOW). Another way of expressing the validity of static data is to say that its domain is forever. Without a finite period of validity, the static datatype does not need a time stamp.

The final major datatype is termed 'snapshot' data. In a conventional or classical database, all data is of the snapshot datatype [Ozsoyoglu and Snodgrass 1995], [Gadia and Nair 1993], [Navathe and Ahmed 1993]. All of the data in a classical database is valid for that instant in time in which it exists. When tuples are updated, the new values replace the old values which are discarded, and a new snapshot is created. Unfortunately, with the snapshot datatype, the history of the changes to the data is lost. Many businesses, though, in order to maintain their competitive advantage, must track the history of their data. Mankind has learned over many millennia that keeping track of history can be highly valuable. We can use the lessons learned from history to avoid repeating past mistakes, and this applies particularly to businesses. Thus, the database should support the history of the data for a given business, and so merely having a snapshot datatype supported by a database is unacceptable; hence the need for the temporal database. However, some data does not need to include time-varying information, and it is this type of data that should be of the snapshot datatype. Therefore, the temporal database will still support the snapshot datatype. Another reason to retain the capability for the snapshot datatype is to provide a smooth transition from the classical database to the temporal database. Since all of the data in a classical database is of the snapshot datatype, migration from the classical database to the temporal database should be seamless [Gadai and Nair 1993].

IV. The Major Purpose for the Temporal Database... Query Capabilities

One of the most important reasons to have a database which supports the temporal dimension, is the ability to perform ad hoc queries on the data. The current standard for conventional (relational) databases is Structured Query Language or SQL. SQL has become the industry standard for Relational Database Management Systems (RDBMS) because of its ease of use due to its English-like syntax. SQL is both feasible and user-friendly. The addition of the temporal element, however, greatly increases the complexity of the queries on temporal data. With the additional element of time, in its current form SQL is no longer able to process ad hoc queries as it did on the (relational) classical database. A new query language or extension to SQL is necessary. Understandably, one of the biggest areas of research today in the field of temporal databases is on this very topic. Many languages and language extensions have been proposed and to examine each in turn or even a few major ones lies outside of the scope of this paper due to the technical nature of the topic. A few remarks on this topic, though, should be noted, and one example of a query language extension will be given.

The new temporal query language should support the current SQL capabilities seamlessly. SQL has greatly assisted the current market dominance of the RDBMS because, as we noted earlier, it is both

suitable to most businesses' needs and is user-friendly. The most promising proposals for a query language which both supports the time dimension and continues to allow users to perform queries without specifying any time criteria (thus avoiding high retraining costs, among other things) are not new query languages per se but are extensions to SQL. One of the most promising examples of temporal-capable SQL is called, appropriately enough, TSQL. TSQL stands for Temporal Structured Query Language. With TSQL ad hoc queries can be performed without specifying any time-varying criteria. Thus, the only essential clauses necessary to perform a query in TSQL remain the same as in SQL: they are the 'SELECT' and 'FROM' clauses. If desired, though, a new criteria clause can be added to the current SQL criteria clauses which are: 'WHERE', 'GROUP BY', 'HAVING', and 'ORDER BY'. The new criteria clause would be either 'WHEN' or 'WHILE'. Either name captures the idea of a time-varying condition. The 'WHEN' clause specifies a particular "time-slice" for which the data is valid or invalid depending on the desired results. Currently, a proposal to add TSQL to the existing ANSI and ISO SQL standards is under consideration by those governing bodies.

V. Business Areas of Application

One of the aims of this paper is to integrate both the theoretical and the practical aspects of temporal database. Up to now almost all of the material has dealt with the theoretical side of the temporal database. In this section of the paper, then, I will present some of the types of businesses who will benefit most from the capabilities of temporal databases.

The first type of business is really not a business per se, it is a tool for nearly any business. This is the data warehouse. Data warehouses are enterprise-wide databases, that is, they are enormous corporate databases which are generally used to store all of the information about a particular company. Data warehouses are proving very useful as repositories of valuable information gathered by data mining tools and used for knowledge discovery. As we have already said, the element of time is so crucial to many businesses, and it is in the data warehouse that the time dimension can be stored and utilized in querying the data. What do data warehouses have to do with temporal databases? According to Ralph Kimball, every data warehouse has a time dimension which makes every data warehouse a temporal database [Kimball 1997], [Snodgrass 1998]. "The time dimension is a unique and powerful dimension in every data mart and enterprise data warehouse" [Kimball 1997].

The second type of business is the scientific laboratory. Each laboratory runs many tests and many versions of the same test, and each test often involves precise timing sequences. In order that this information not be lost, it must be stored in a database. Since the element of time in these tests cannot simply be discarded, the database in which the information is stored must be a temporal database. One of the benefits of storing scientific test information in a temporal database which support a temporal query language, is that new patterns and knowledge can be mined from the database [Loomis 1997].

The third type of business is one which relies heavily on sales and marketing. Success in sales and marketing depends heavily on good timing. When to approach a customer and when to advertise in certain locations is necessary knowledge for these types of businesses, and the ability to correctly forecast this information is better provided by a temporal database than by a classical database.

The last type of business presented here is multimedia. Multimedia is one of the fastest growing areas of business today, especially over the internet. The time is not far off when a person will be able to order movies over the internet and watch them on his/her PC. The process that will occur when the person orders the movie is that video images which are stored in a multimedia database will be synchronized with the audio files in either the same or else a different database. This synchronization, as the word itself implies, involves a time element. "Because multimedia is time-based, a great deal of synchronization is necessary. If implemented properly, media components will display in a precise synchronized order" [David]. Again, "[Multimedia objects have temporal and spatial relationships that must be maintained in their presentation... some of this data has real-time constraints in its delivery to client stations" [Ozsu]. Thus, the multimedia database, just as the data warehouse, is actually an instance of a temporal database.

In concluding, I would like to say a few words about the future of temporal databases. As the business environment becomes increasingly competitive, the knowledge within an organization becomes an increasingly valuable resource. One of the key elements for knowledge within the business environment is the time dimension. In order to track this information businesses must implement temporal databases. To give businesses this capability to keep track of time-varying data, two things will happen. First, most major database commercial producers will gradually include the ability to store various types of temporal data in their database products. Second, a temporal extension will be added to the existing ANSI and ISO SQL standards. This will enable users to take advantage of the new temporal features in the major database products. At that point, most of the major commercial databases will be temporal databases. This will allow more and better knowledge to be stored in and mined from the newly temporalized databases, which will in turn allow those businesses who adopt the temporal databases to gain a competitive advantage. For an example of the benefits of integrating a temporal database with an existing business system to increase knowledge see Tuzhilin 1993

References

- David, Michael M. "Multimedia Databases Through the Looking Glass," Database Programming & Design Online, [WWW article]. URL <http://www.dbpd.com/9705david.htm>.
- Gadia, Shashi K. and Sunil S. Nair. "Temporal Databases: A Prelude to Parametric Data," Temporal Databases: Theory, Design, and Implementation, eds. A. Tansel, et al., The Benjamin/Cummings Publishing Company, Inc., 1993, pp.28-66.
- Jensen, C., et al. [Glossary], Temporal Databases: Theory, Design, and Implementation, eds. A. Tansel, et al., The Benjamin/Cummings Publishing Company, Inc., 1993, pp.621-633.
- Kimball, R. "It's Time for Time", DBMS Online, July 1997, www.dbmsmag.com/9707d05.html.
- Loomis, Timothy. "Audit History and Time-slice Archiving in an Object DBMS for Laboratory Databases," Hewlett Packard Journal, August 1997.
- Navathe, Shamkant B. and Rafi Ahmed. "Temporal Extensions to the Relational Model and SQL," Temporal Databases: Theory, Design, and Implementation, eds. A. Tansel, et al., The Benjamin/Cummings Publishing Company, Inc., 1993, pp. 92-109.
- Ozsoyoglu, Gultekin and Richard T. Snodgrass. "Temporal and Real-Time Databases: A Survey," IEEE Transactions on Knowledge and Data Engineering, August 1995.
- Ozsu, M. Tamer. "A New Foundation," Database Programming & Design Online, [WWW article]. URL <http://www.dbpd.com/ozsu.htm>
- Snodgrass, R. "Of Duplicates and Septuplets," Database Programming & Design, June 1998, pp. 46-49.
- Steiner, Andreas. What are Temporal Databases?, [WWW article]. URL <http://www.inf.ethz.ch/personal/steiner/TemporalDatabases/TemporalDB.html>
- Tansel, Abdullah, et al. "Part I: Extensions to the Relational Model," Temporal Databases: Theory, Design, and Implementation, eds. A. Tansel, et al., The Benjamin/Cummings Publishing Company, Inc., 1993, pp. 1-5.
- Tuzhilin, Alexander. "Applications of Temporal Databases to Knowledge-based Simulations," Temporal Databases: Theory, Design, and Implementation, eds. A. Tansel, et al., The Benjamin/Cummings Publishing Company, Inc., 1993, pp. 580-593.