Temporality in Databases

Invited Talk

Abdullah Uz Tansel

Department of Computer Engineering,
Bilkent University,
Ankara, Turkey

Temporality, or the time dimension is an essential aspect of the reality databases attempt to model and keep data about. However, in many database applications temporal data is treated in a rather ad hoc manner in spite of the fact that temporality should be an integral part of any data model. In this presentation we will address issues peculiar to temporal data and explore how we can add a temporal dimension to databases. We will specifically focus our attention on the relational data model and the object relational systems for managing temporal data since these systems are widely available and they are also widely used [1].

A database maintains data about an enterprise and its activities. It addresses organizational information requirements by maintaining accurate, complete, and consistent data from which information is extracted for various applications. Conventional databases are designed to capture the most recent data since they are built to process the transactions efficiently and effectively. As new data values become available through organizational transactions the existing data values are removed from the database and they are discarded or archived. Such databases capture a snapshot of reality, mostly the current snapshot of the reality. Although conventional databases serve many applications well, they are insufficient for those in which past and/or future data are also required. Such a need is very obvious in datawarhouses and OLAP applications for decision support. Thus there is an obvious need for a database that fully supports the storage and querying of data that varies over time. In the broadest sense, a database that maintains past, present, and future data is called a *temporal database*.

There are two common views of time, continuous and discrete time though the time is continuous in nature. Continuous time is considered to be isomorphic to real numbers whereas discrete time is considered to be isomorphic to natural numbers or integers. Both views assume that time is linearly ordered; for the two different time points t1 and t2, either t1 is before t2 or t2 is before t1. Discrete interpretation of time has commonly been adopted by the research community in temporal databases because of its simplicity and relative ease of implementation. Hence, we will interpret time as a set of equally spaced and ordered time points and denote it by T where $T = 0, 1, 2, ...now....$ The symbol 0 is the relative beginning, and *now* is a special variable to represent current time. The value of *now* advances as the clock ticks. Any point beyond *now* is future time. We do not specify any time units and time granularities for the sake of simplicity. Note that between two consecutive time points there is a time

T. Yakhno (Ed.): ADVIS 2004, LNCS 3261, pp. 1–3, 2004.

duration that is invisible unless a smaller time granularity is used. An interval or a time period is any consecutive set of time points and is designated by its boundary points. The closed interval [b, e] contains all the time points including b and e, whereas the half-open interval [b, e) does not include e. Any subset of T is called a *temporalelement*[1], that can also be considered as a disjoint union of time intervals. Any interval or temporal element that includes the special variable *now* expands as the value of *now* advances. Time points, intervals, and temporal elements are essential constructs for modeling and querying temporal data.

Snodgrass developed taxonomy of time in databases [2]. *Valid time* denotes the time when a fact becomes effective in reality. *Transaction time*, on the other hand, refers to the time when a new value is posted to the database. These two times are orthogonal and can be supported separately, or both can be supported in concert. The third variety, user-defined time, is an un-interpreted time domain managed by the user. User-defined time is the easiest to support and many conventional database management systems, as well as the SQL2 standard, include such support.

These kinds of time induce different types of databases. A traditional database supporting neither valid nor transaction time is termed a *snapshot database*. A *valid-time database* contains the entire history of the enterprise, as best known *now*. A *transaction-time database* supports transaction time and hence allows rolling back the database to a previous state. This database records all errors and provides a complete audit trail. As such, it is append-only. A *bitemporal database* records both valid time and transaction time and combines the features of the previous two types. It allows retroactive as well as post active changes; the complete history of these changes and the original values they replaced are all available.

We believe that any temporal database should meet the following fundamental requirements [4]. Let DB_t denote the database state at time t:

1. The data model should be capable of modeling and querying the database at any instance of time, i.e., D_t. The data model should at least provide the modeling and querying power of a 1NF relational data model. Note that when t is *now*, D_t corresponds to traditional database.
2. The data model should be capable of modeling and querying the database at two different time points, i.e., D_{t1} and D_{t2} where $t1 \neq t2$. This should be the case for the time intervals and temporal sets as well.
3. The data model should allow different periods of existence in attributes within a tuple, i.e., non-homogenous (heterogeneous) tuples should be allowed.
4. The data model should handle multi-valued attributes at any time point, i.e., in D_t.
5. A temporal query language should have the capability to return the same type of objects it operates on.
6. A temporal query language should have the capability to regroup the temporal data according to a different temporal attribute.

7. The data model should be capable of expressing set-theoretic operations, as well as set comparison tests, on the timestamps, be it time points, time intervals, or temporal elements.

We will discuss fundamental issues and their implications with respect to the desired features of temporal databases listed above [3, 5, 6]. These issues include temporal data modeling, adding times stamps to tuples or attributes, gluing or not gluing time to values (tuples), temporal integrity constraints, designing temporal relational databases, operations and expressive power of temporal query languages, and adding temporal support to SQL3. We expect the presentation will be beneficial to the researchers as well as the practitioners.

References

1. S.K. Gadia, *A homogeneous relational model and query languages for temporal databases*, ACM Transactions on Database Systems, 13 (4), pp. 418-448, 1988.
2. R. Snodgrass, *The temporal query language TQUEL", ACM Transactions on Database Systems* 12 (2), pp. 247 - 298, 1987.
3. A.U. Tansel, *Adding time dimension to relational model and extending relational algebra*, Information Systems 11 (4), pp. 343 - 355, 1986.
4. A.U. Tansel and E. Tin, *Expressive power of temporal relational query languages*, IEEE Transactions on Knowledge and Database Engineering, 9 (1), pp. 120 - 134, 1997.
5. A. U. Tansel, *Temporal Relational Data Model*, IEEE Transactions on Knowledge and Database Engineering, Vol. 9, No. 3, May 1997, pp. 464 - 479.
6. A. U. Tansel, *On Handling Time-Varying Data in the Relational Data Model*, Journal of Information and Software Technology, Vol. 46, No. 2, February 2004, pp. 119 - 126.