Modeling and Querying Temporal Data

Abdullah Uz Tansel
Bilkent University, Turkey

INTRODUCTION

Databases in general store current data. However, the capability to maintain temporal data is a crucial requirement for many organizations and provides the base for organizational intelligence. A temporal database has a time dimension and maintains time-varying data (i.e., past, present, and future data). In this article, we focus on the relational data model and address the subtle issues in modeling temporal data, such as comparing database states at two different time points, capturing the periods for concurrent events, and accessing to times beyond these periods, handling multivalued attributes, coalescing, and restructuring temporal data (Gadia 1988, Tansel & Tin, 1997). Many extensions to the relational data model have been proposed for handling temporal data.

There is a growing interest in temporal databases in many application domains. The first book dedicated to temporal databases, by Tansel et al. (1993), was followed by others addressing issues in handling time-varying data (Betini, Jajodia, & Wang, 1988; Date, Darwen, & Lorentzos, 2002; Snodgrass 1999).

BACKGROUND

The set \( T \) denotes time values, and it is a total order under the “\( \leq \)” relationship, hence allowing comparisons and calculations. The set \( T \) can be represented by integers \( (I) \) or real numbers \( (R) \). Time is continuous, and real numbers are a better approximation because they can easily accommodate time granularity. Because of its simplicity, time values in a calendar system are commonly implemented by the integers 0, 1, ..., now. The symbol 0 is the relative origin of time, and now is a special symbol that represents the current time. Now advances according to the time granularity used. There are different time granularities, such as seconds, minutes, hours, days, months, years, and so forth (for a formal definition, see Betini, Jajodia, & Wang 1988).

A subset of \( T \) is called a temporal set. A temporal set that contains consecutive time points \( \{ t_1, t_2, ..., t_n \} \) is represented either as a closed interval \( [t_1, t_n] \) or as a half open interval \( [t_1, t_{n+}) \). A temporal element (Gadia, 1988) is a temporal set that is represented by the maximal intervals corresponding to its subsets having consecutive time points. Temporal sets, intervals, and temporal elements can be used as time stamps for modeling temporal data and are essential constructs in temporal query languages. Temporal sets and temporal elements are closed under set-theoretic operations, whereas intervals are not. However, intervals are easier to implement. Time intervals, hence temporal elements and temporal sets, can be compared. The possible predicates are before, after, meet, during, and so forth. An interval or a temporal set (element) that includes now expires in its duration. Other symbols, such as forever or until changed, are also proposed as alternatives to the symbol now for intuitive handling of future data.

There are various aspects of time in databases (Snodgrass, 1987). Valid time indicates when a data value becomes effective. It is also known as logical or intrinsic time. On the other hand, the transaction time (or physical time) indicates when a value is recorded in the database. User-defined time is application-specific and is an attribute whose domain is time. Temporal databases are in general append-only that is, new data values are added to the database instead of replacing the old values. A database that supports valid time keeps historical values and is called a valid time (historical) database. A rollback database supports transaction time and can roll the database back to any time in the past. Valid time and transaction time are orthogonal. However, a temporal database that supports both valid time and transaction time is capable of handling retroactive and post-active changes on temporal data. In the literature, the term temporal database is generically used to mean a database with some kind of time support.

This chapter will focus on the valid time aspect of temporal data in relational databases. However, the discussion herein can easily also be extended to databases that support transaction time.

MODELING TEMPORAL DATA

A temporal atom is a time stamped value, \( \langle t, v \rangle \), and represents a temporal value. It asserts that the value \( v \) is valid over the period of time stamp \( t \) that can be a time point, interval, temporal set, or temporal element. Time points are suitable only for values that are valid at a time point, not over a period. Time can be added to tuples or
attributes and hence, temporal atoms can be incorporated differently into the relational data model. To represent temporal atoms in tuple time stamping, a relation is augmented with two attributes that represent the end points of an interval or a time column whose domain is intervals, temporal sets, or temporal elements. Figure 1 depicts salary (SAL) history of an employee (E1), in which intervals or temporal elements are used as time stamps with a time granularity of month/year. Salary is 20K from 1/01 to 5/02 and from 8/02 to 6/03. The discontinuity is the result of the employee quitting on 6/02 and returning on 8/02. The salary is 30K since 6/03. Figure 2 gives the same salary history in attribute time stamping. An attribute value is a set of temporal atoms. Each relation has only one tuple that carries the entire history. It is also possible to create a separate tuple for each time stamped value (temporal atom) in the history (i.e., three tuples for Figure 2.a, two tuples for Figure 2.b).

One noteworthy aspect of data presented in Figure 2 is that the time stamps are glued to attribute values. In other words, attribute values are temporal atoms. In forming new relations as a result of query expressions, these time stamps stay with the attribute values. On the other hand, in tuple time stamping, a time stamp may be implicit (glued) or explicit (unglued) to tuples. This is a design choice, and the relations in Figure 1 can be interpreted as having implicit or explicit time stamps. An implicit time stamp is not available to the user as a column of a relation, though the user can refer to it. On the other hand, an explicit time stamp is like any other attribute of a relation and it is defined on a time domain. Implicit time stamps restrict the time of a new tuple created from two constituent tuples, since each tuple may not keep its own time stamp and a new time stamp needs to be assigned to the result tuple. Explicit time stamps allow multiple time stamps in a tuple. In this case, two tuples may be combined to form a new tuple, each carrying its own time reference. However, the user needs to keep track of these separate time references.

**TEMPORAL RELATIONS**

Figure 3 shows some sample employee data for the EMP relation over the scheme E# (Employee number), ENAME (Employee name), DNAME (Department name) and SALARY. E# and ENAME are (possibly) constant attributes, whereas DNAME and SALARY change over time. In EMP relation, temporal elements are used in temporal atoms for representing temporal data. Time stamp of E# represents the life span of an employee that is stored in the database. Note that EMP is a nested (NINF—Non-First Noral Form) relation. It is one of the many possible relational representations of the employee data (Clifford & Tansel, 1985; Gadia, 1988; Tansel, 2004). Figure 4 gives, in tuple time stamping, three 1NF relations, EMP_N, EMP_D, and EMP_S for the EMP relation of Figure 3 (Lorentzos & Johnson, 1987; Navathe & Ahmed 1987; Sarda, 1987; Snodgrass 1987). In Figure 3, temporal sets (elements) can also be used as the time reference. Similarly, in the relations of Figure 4, intervals, or temporal sets (elements), can also be used as the time reference in an attribute that replaces the Start and End columns.

Note that in tuple time stamping, a relation may contain only attributes whose values change at the same time; attributes changing at different times require separate relations. Each particular time stamping method imposes restrictions on the type of base relations allowed as well as the new relations that can be generated from the base relations. The EMP relation in Figure 3 is a unique representation of the employee data, in which each tuple contains the entire history of an employee (Clifford & Tansel, 1985; Gadia, 1988; Tansel, 1997). The E# is a temporal grouping identifier, regardless of the time stamp used (Clifford, Croker, & Tuzhilin, 1993). In the case of tuple time stamping an employee’s data is dispersed into several tuples (i.e., there are three salary tuples for employee 121 in Figure 4.c). These tuples belong to the same employee because their E# values are equal.

For the relations in Figures 3 and 4 there are many other possible representations that can be obtained by taking subsets of temporal elements (intervals) and creating several tuples for the same employee. These relations are called weak relations (Gadia, 1988). Though they contain the same data as the original relation in unique representation, query specification becomes very complex. Weak relations naturally occur in querying a temporal database. Weak relations can be converted to an equivalent unique relation by coalescing tuples that belong to the same object (employee) into one single tuple (Bohlen, Snodgrass & Soo 1996; Sarda, 1987).

Design of relational databases is based on functional and multivalued dependencies. Roughly, a relation is created to represent the data for similar objects (entities),
Related Content

The Expert's Opinion
www.igi-global.com/article/expert-opinion/51100?camid=4v1a

Incremental Discovery of Fuzzy Functional Dependencies
www.igi-global.com/chapter/incremental-discovery-fuzzy-functional-dependencies/20370?camid=4v1a

Data Modeling in UML and ORM: A Comparison
Terry Halpin and Anthony Bloesch (1999). Journal of Database Management (pp. 4-13).
www.igi-global.com/article/data-modeling-uml-orm/51222?camid=4v1a

Fabric Database and Fuzzy Logic Models for Evaluating Fabric Performance
www.igi-global.com/chapter/fabric-database-fuzzy-logic-models/20367?camid=4v1a