TEMPORAL DATABASES

Richard Thomas Snodgrass

A temporal database (*see* Temporal Database) contains time-varying data. Time is an important aspect of all real-world phenomena. Events occur at specific points in time; objects and the relationships among objects exist over time. The ability to model this temporal dimension of the real world is essential to many computer applications, such as accounting, banking, econometrics, geographical information systems, inventory control, law, medical records, multi-media, process control, reservation systems, and scientific data analysis.

Conventional databases represent the state of an enterprise at a single moment of time. Although the contents of the database continue to change as new information is added, these changes are viewed as modifications to the state, with the old, out-of-date data being deleted from the database. The current contents of the database may be viewed as a snapshot of the enterprise. When a conventional database is used, the attributes involving time are manipulated solely by the application programs, with little help from the database management system (DBMS).

Alternatively, in a temporal database, queries over previous states are easy to specify. Also, modifications to previous states (if an error is detected, or if more information becomes available) and to future states (for planning purposes) are also easier to express using a temporal DBMS.

Almost all database applications concern time-varying information. In fact, it is difficult to identify applications that do not involve the management of time-varying data. The advantages provided by built-in temporal support include higher-fidelity data modeling, more efficient application development, and a potential increase in performance.

1 Time Data Types

Several temporal data types have proven useful. The most basic is a time instant (*see* Instant), which is a particular chronon on the time line. An evnet (*see* Event) is an instantaneous fact, that is, something occurring at an instant. The event occurrence time (*see* Event Occurrence Time) of an event is the instant at which the event occurs in the real world. The fact in the database is timestamped (*see* Timestamp) with its event occurrence

time.

SQL-92 provides three instant data types: DATE (a particular day, with a year in the range AD 1–9999), TIME (a particular second within a range of 24 hours), and TIMESTAMP (a particular fraction of a second, defaulting to microsecond, of a particular day).

A time period (*see* Time Period) is the time between two instants. (Note: in some of the literature, this notion is called a time interval, but this usage conflicts with the SQL-92 data type INTERVAL, which is a different concept altogether.) SQL-92 does not include periods, but periods are now part of the evolving SQL3 specification.

A time interval (*see* Time Interval) is a directed duration of time, that is, an amount of time with a known length, but not specific starting or ending instants. A positive interval denotes forward motion of time, toward the future; a negative interval denotes motion toward the past. Adding the interval of -3 days to January 15, 1998 would result in January 12, 1998. SQL-92 supports two kinds of intervals, month-year and second-day intervals.

A final temporal data type is temporal element(*see* Temporal Element), which is a finite union of periods.

2 Associating Facts with Time

In the context of databases, two time dimensions are of general interest. Valid time (*see* Valid Time) concerns the time a fact was true in reality (*see* Logical Time). The valid time of an event is the time at which the event occurred in the real world, independent of the recording of that event in some database. Valid times can also be in the future, if it is expected that some fact will be true at a specified time in the future. Transaction time (*see* Transaction Time) concerns the time the fact was present in the database as stored data. The transaction time (a period of time) of a fact identifies the transaction that inserted the fact into the database and the transaction that removed this fact from the database. Transaction times are consistent with the serialization order of the transactions that entered or modified the facts stored in the database (*see* Concurrency Control, Transaction Processing).

These two dimensions are orthogonal. A data model supporting neither is termed snapshot (*see* Snapshot Relation, Distributed Snapshots), as it captures only a single snapshot in time of both the database and the enter-

prise that the database models. A data model supporting only valid time is termed valid-time (*see* Valid-Time Relation); one that supports only transaction time is termed transaction-time (*see* Transaction-Time Relation); and one that supports both valid and transaction time is termed bitemporal (*see* Bitemporal Relation) (*temporal* is a generic term implying some kind of time support). Transaction-time and bitemporal relations are append-only, making them prime candidates for storage on write-once optical disks.

A third kind of time may be included: user-defined time (*see* User-Defined Time). This term refers to the fact that the semantics of these values are known only to the user, and are not interpreted by the DBMS, as differentiated from valid and transaction time, whose semantics are supported by the DBMS.

Information that is temporally indeterminate (*see* Temporal Indeterminacy) can be characterized as "don't know exactly when" information. This kind of information is prevalent; it arises in various situations, including finer system granularity, imperfect dating techniques, uncertainty in planning, and unknown or imprecise event times. An event is determinate (*see* Temporally Determinate) if it is known precisely when it occurred.

3 Temporal Data Models

Support for time in conventional database systems is entirely at the level of user-defined time (i.e., attribute values drawn from a temporal domain).

A temporal data model should simultaneously satisfy many goals. It should capture the semantics of the application to be modeled in a clear and concise fashion. It should be a consistent, minimal extension of an existing data model, such as the relational model. It is best if the temporal data model presents all the time-varying behavior of a fact or object coherently. The data model should be easy to implement, while attaining high performance.

Time has been added to many data models. However, by far the majority of work in temporal databases is based on the relational and object-oriented data models. The experience of the last fifteen years and some forty data models appearing in the literature argues that designing a temporal data model with all of these characteristics is elusive at best, and is probably not possible.

An effort has recently been completed to consolidate approaches to tem-

poral data models and calculus-based query languages, to achieve a consensus extension to SQL-92 and an associated data model upon which future research can be based. This extension is termed the *Temporal Structured Query Language*, or TSQL2.

TSQL2 employs the Bitemporal Conceptual Data Model as its underlying data model, in which temporal databases are designed and queries are expressed. This data model retains the simplicity and generality of the relational model. A separate, representational data model, of equivalent expressive power, is employed for implementation and for ensuring high performance. Other presentational data models may be used to render time-varying behavior to the user or application. A coordinated suite of data models can achieve in concert goals that no single data model could attain.

4 Temporal Query Languages

Some three dozen temporal relational query languages have been proposed to date.

There have been three general approaches to adding valid-time support to a data model and query language. The first approach utilizes the substantial expressive power of the relational or object-oriented data model directly, and thus requires no changes either to the model or to the query language to support time-varying information. The disadvantage is that the user is required to "roll his own" support for time when specifying the schema, and also when specifying queries. This approach also renders query optimization much more difficult, as there are no syntactic constructs provided in the language that are specific to time.

A second approach is to include general extensions to the data model and query language for other reasons, and then show how these extensions may be used to support time-varying information. This approach has been used only with object-oriented query languages.

The third approach is to propose specific data modeling and query language constructs to support information varying over valid time (*see* Temporal Logic). This is the approach adopted by the vast majority of temporal relational query languages. Most add numerous new constructs and temporal operators, yet attempt to retain reducibility to the non-temporal query language on which they are based, which ensures that the user's intuition

about the base language carries over to the temporal extension.

In considering support for transaction time, an important distinction must be made: either the tuples, object instances or attributes are themselves versioned (termed *extension versioning*), or the *definitions* of those objects are versioned (termed schema versioning, *see* Schema Versioning). If extension versioning is adopted, then schema versioning may or may not be supported. If extension versioning is not supported, then schema versioning is not relevant, as only the most recent version of the schema need be retained. In schema evolution (*see* Schema Evolution), the schema can change in response to the varying needs of the application. In schema versioning (*see* Schema Versioning), there are multiple schemas logically in effect. Schema versioning has been examined both in the context of relational databases and in the context of object-oriented databases.

5 Architectural Issues

We now turn to the implementation of the temporal data models and query languages.

Optimization of temporal queries is substantially more involved than that for conventional queries, for several reasons. Optimization of temporal queries is more critical, and thus easier to justify expending effort on, than conventional optimization. The relations that temporal queries are defined over may be larger, and often grow monotonically, implying that unoptimized queries take longer and longer to execute.

On the other hand, there is greater opportunity for query optimization when time is present. Time advances in one direction: the (transaction) time domain is continuous expanding, and the most recent time point is the largest value in the domain. This implies that a natural clustering on sort order will manifest itself, which can be exploited during query optimization and evaluation.

There have been four basic responses to this challenge. The first proposes separating valid-time and transaction-time data, which grows monotonically, from the current data, whose size is fairly stable and whose access is more frequent. This separation, termed temporal partitioning (*see* Temporal Partitioning), was shown to significantly improve performance of some queries, and was later generalized to allow multiple cached states, which further im-

proves performance.

A second approach is the introduction of new query optimization strategies. A single query can be optimized by replacing the algebraic expression with an equivalent one that is more efficient, by changing an access method associated with a particular operator, or by adopting a particular implementation of an operator. To determine which access method is best for each algebraic operator, meta-data, that is, statistics on the stored temporal data, and cost models, that is, predictors of the execution cost for each operator implementation/access method combination, are needed. Temporal data requires additional meta-data, such as the time period over which the relation is defined, the tuple arrival distributions, the distributions of the time-varying attributes, regularity and granularity of temporal data, and the frequency of the null values that are sometimes introduced when attributes within a tuple aren't synchronized.

Introducing new algorithms for expensive operators, such as joining two tables, eliminating duplicates, aggregating over many rows, and coalescing overlapping periods into a single period, is a third approach for gaining efficiency. In particular, a wide variety of temporal joins have been considered. 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.

A fourth approach is to develop new temporal indexes. Conventional indexes have long been used to reduce the need to scan an entire relation to access a subset of its tuples. Indices are even more important in temporal relations that grow monotonically in size. There has been a great deal of research over the past five years in temporal indexing. Most of the indexes are based on B+-Trees, which index on values of a single key; the remainder are based on R-Trees, which index on ranges (periods) of multiple keys. There has been considerable discussion concerning the applicability of point-based schemes for indexing period data. Some argue that structures that explicitly accommodate periods, such as R-Trees and their variants, are preferable; others argue that mapping periods to their endpoints is efficient for search.

Temporal database research has been active for about twenty years. There have been many significant accomplishments.

- The semantics of the time domain, including its structure, dimensionality, indeterminacy, and real-time aspects, is well-understood.

- A great amount of research has been focused on temporal data models, addressing this extraordinarily complex and subtle design problem.
- The semantics of temporal relational schemas and their logical design are well understood. The Bitemporal Conceptual Data Model is gaining acceptance as the appropriate model in which to consider data semantics.
- Many temporal query languages have been proposed.
- Temporal joins, aggregation, duplicate elimination 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.
- Several commercial temporal object-oriented DBMS's are now on the market.

The following research areas need to be addressed.

- 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.
- Little has been done in integrating spatial, temporal, and active data models, query languages, and implementation techniques.
- Achieving adequate performance in a temporal DBMS remains a challenge.
- To date, most research 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. Approaches for transitioning legacy applications will become increasingly sought after as temporal technology moves from research to practice.

6 References

A series of six bibliographies concerning temporal databases culminating in [4] references some 1200 papers. The book edited by Tansel [3] provides a still-current snapshot of temporal databases research.

[1] 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.

[2] R. T. Snodgrass (ed), *The TSQL2 Temporal Query Language*, Kluwer Academic Publishers, 1995.

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

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