Temporal Databases

Richard Snodgrass, University of North Carolina at Chapel Hill
Ilsoo Ahn, AT&T Bell Laboratories

Database management systems should provide temporal support directly. Four types of databases are differentiated by their ability to represent temporal information.

Time is an essential part of information about the constantly evolving real world. Facts or data need to be interpreted in the context of time. Causal relationships among events or entities are embedded in the temporal information. Time is a universal attribute in most information management applications and deserves special treatment as such.

Databases supposedly model reality, but conventional database management systems (DBMSs) lack the capability to record and process time-varying aspects of the real world. With increasing sophistication of DBMS applications, the lack of temporal support raises serious problems in many cases. For example, conventional DBMSs cannot support historical queries about past status, let alone trend analysis (essential for applications like decision support systems). There is no way to represent retroactive or proactive changes, while support for error correction or audit trail necessitates costly maintenance of backups, checkpoints, or transaction logs to preserve past states. There is a growing interest in applying database methods for version control and design management in computer-aided design, requiring capabilities to store and process time-dependent data. Without temporal support from the system, many applications have been forced to manage temporal information in an ad-hoc manner.

The need to provide temporal support in DBMSs has been recognized for at least a decade. A bibliographical survey in 1982 contained about 70 articles relating time and information processing,[1] and at least 90 more articles have appeared since then. Recently, the rapid decrease of storage costs, coupled with the emergence of promising new mass storage technologies such as optical disks,[2] has amplified interest in database management systems with version management or temporal support. George Copeland maintained that

> . . . as the price of hardware continues to plummet, thresholds are eventually reached at which these compromises [to achieve hardware efficiency] must be rebalanced in order to minimize the total cost of a system. . . . If the deletion mechanism common to most database systems today is replaced by a non-deletion policy . . . , then these systems will realize significant improvements in functionality, integrity, availability, and simplicity.[3]

Gio Wiederhold also observed, in a review of the present state of database technology and its future, that

> The availability of ever greater and less expensive storage devices has removed the impediment that prevented keeping very detailed or extensive historical information in on-line databases. . . . An immediate effect of these changes will be the retention of past data versions over long periods.[4]

In the past five years, numerous schemes have been proposed to support temporal information processing in database management systems by incorporating one or more time attributes. However, there has been some confusion concerning terminology and the definition of these time attributes. In this article we describe a taxonomy of time consisting of three distinct time concepts for use in databases.[5] Using the taxonomy, we define four types of databases, differentiated by their ability to support these time concepts and processing of temporal information.

0018-9162/86/0900-0035$01.00 © 1986 IEEE

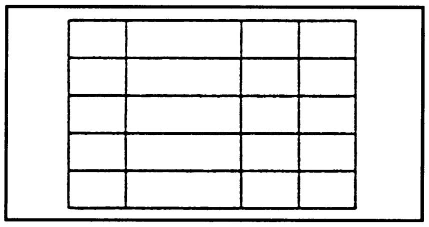


Figure 1. A snapshot relation.

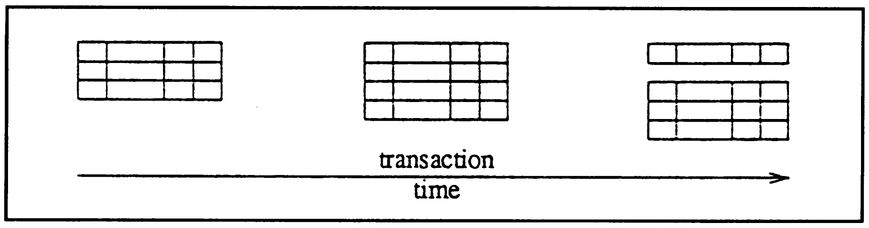


Figure 2. A rollback relation.

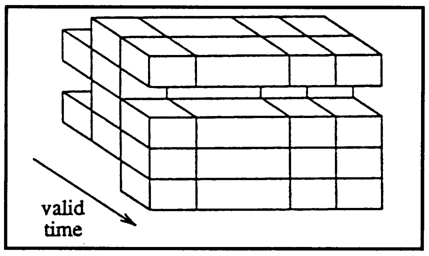


Figure 3. An historical relation.

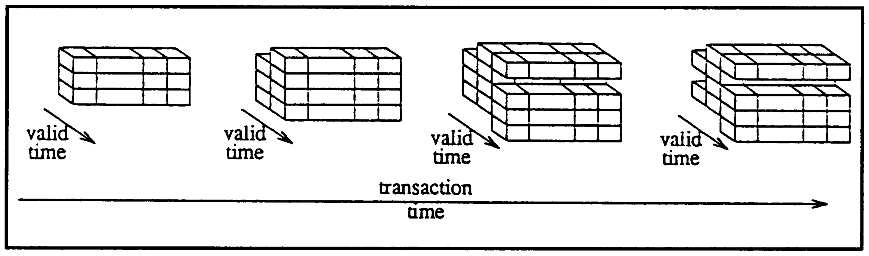


Figure 4. A temporal relation.

The taxonomy

Although the following discussion is based on the relational model, analogous arguments also apply to hierarchical or network models.

Snapshot databases. Conventional databases model an enterprise as it changes dynamically by a snapshot at a particular point in time. A *state* or an *instance* of a database is its current content, which does not necessarily reflect the current status of the enterprise. The state of a database is updated using data manipulation operations such as insertion, deletion, or replacement, taking effect as soon as they are committed. In this process, past states of the database, representing those of the enterprise, are discarded and forgotten. We term this type of database a *snapshot database.*

In the relational model, a database is a collection of *relations.* Each relation consists of a set of *tuples* with the same set of *attributes* and is usually represented as a two-dimensional table (see Figure 1). As changes occur in the enterprise, this table is updated appropriately. For example, at a certain moment an instance of the relation Faculty with the two attributes Name and Rank may be

Name	Rank
Merrie	Associate Professor
Tom	Associate Professor

and a query in Quel (a tuple calculus-based language for the Ingres database management system[6]) as to Merrie's rank,

```
range of f is Faculty
retrieve (f.Rank)
  where f.Name = "Merrie"
```

yields the rank of associate professor.

In many situations this snapshot database is inadequate. For example, it cannot answer queries such as

- What was Merrie's rank two years ago? (historical query)
- How did the number of faculty change over the last five years? (trend analysis)

nor record facts like

- Merrie was promoted to a full professor starting last month. (retroactive change)
- Lee is joining the faculty next month. (proactive change)

Without system support in this respect, many applications have had to maintain and handle temporal information in an ad-hoc manner. For instance, many personnel databases attempt to record the entire employment history of the company's employees. That some of the attributes record time and that only a subset of the employees actually work for the company at any particular point in time do not concern the DBMS itself. The DBMS provides no facility for interpretation or manipulation of this information; such operations must be handled by specially-written application programs. The fact that data vary over time is not an application-specific aspect. This aspect should be supported in a general fashion by the DBMS, rather than by application programs.

Rollback databases. One approach to resolving the above deficiencies is to store all past states, indexed by time, of the snapshot database as it evolves. Such an approach requires a representation of *transaction time*, the time the information was stored in the database. Under this approach a relation can be illustrated conceptually in three dimensions (see Figure 2), with transaction time serving as the third axis. The relation can be regarded as a sequence of snapshot relations (termed *snapshot states*) indexed by time. By moving along the time axis and selecting a particular snapshot state, it is possible to get a snapshot of the relation at some time in the past and make queries about it (a snapshot relation). The operation of selecting a snapshot state is termed *rollback*, and a database supporting it is termed a *rollback database.*

Changes to a rollback database may be made only to the most recent snapshot state. The relation illustrated in Figure 2 had three transactions applied to it, starting from a null relation: (1) addition of three tuples, (2) addition of one tuple, and (3) deletion of one tuple (entered in the first transaction) and addition of another tuple. Each transaction results in a new shapshot state being appended to the front of the time axis. Once a transaction is committed, the snapshot states in the rollback relation may not be altered.

The distinction between snapshot databases and rollback databases is that the latter have the ability to return to any previous state to execute a snapshot query. Any query language can be converted to one that can query a rollback database by adding a clause effecting the rollback operation. TQuel (for Temporal QUEry Language),[7] an extension of Quel for temporal databases, augments the retrieve statement with an as of clause to specify the relevant transaction time. The TQuel query

```
range of f is Faculty
retrieve (f.Rank)
  where f.Name = "Merrie"
  as of "September 1978"
```

on a rollback Faculty relation will find Merrie's rank as of September 1978. The result of a query on a rollback database is a pure snapshot relation.

One limitation of supporting transaction time is that the history of database activities, rather than the history of the real world, is recorded. A tuple becomes effective as soon as it is entered into the database, as in a snapshot database. There is no way to record retroactive/proactive changes, nor to correct errors in past tuples. Errors can sometimes be overridden (if they are in the current state), but they cannot be forgotten. For instance, if it were discovered that Merrie's promotion had occurred earlier than previously recorded, this error could not be corrected in a rollback database; the query given above would continue to respond with the incorrect rank.

Historical databases. While rollback databases record a sequence of snapshot states, *historical databases* record a single historical state per relation. As errors are discovered, they are corrected by modifying the database. Previous states of the database itself are not retained, so it is not possible to view the database as it was in the past. No record is kept of the errors corrected. Historical databases resemble snapshot databases in this respect. Historical databases support *valid time,* the time when the relationship in the enterprise being modeled was valid.

The semantics of valid time are closely related to reality, hence more complex than the semantics of transaction time concerned with database activities. Therefore, historical databases need sophisticated operations to manipulate the complex semantics of valid time adequately. TQuel supports such queries (termed *historical queries*) by augmenting the retrieve statement with a valid clause to specify how the implicit time attribute is computed and a when predicate to specify the temporal relationship of tuples participating in a derivation. These added constructs handle complex temporal relationships such as begin of, precede, and overlap. For example, the TQuel query requesting Merrie's rank when Tom arrived is

```
range of f1 is Faculty
range of f2 is Faculty

retrieve (f1.Rank)
  where f1.Name = "Merrie" and
        f2.Name = "Tom"
  when f1 overlap begin of f2
```

The derived relation is also a historical relation that can be used in further historical queries.

Another distinction between historical and rollback databases is that historical DBMSs support arbitrary modification, whereas rollback DBMSs only allow snapshot states to be appended. The same sequence of transactions that resulted in the rollback relation in Figure 2 will result in the historical relation in Figure 3, where the label of the time axis indicates the valid time. However, the historical relation can show that a later transaction has changed the time when a tuple takes effect in the relation, which is not possible on a rollback relation. Rollback DBMSs can roll back to an incorrect previous snapshot relation; historical DBMSs can represent current knowledge about the past.

Temporal databases. Benefits of both approaches can be combined by supporting both transaction time and valid time in the same relation. While a rollback database views stored tuples, whether valid or not, as of some moment in time, and a historical database always views tuples valid at some moment as of now, a temporal database can view tuples valid at some moment seen as of some other moment, thereby completely capturing the history of retroactive/proactive changes. The user of a temporal DBMS can examine historical information from the viewpoint of a previous state of the database; both kinds of time may be manipulated in a query.

We use the term *temporal database* to emphasize the need for both valid time and transaction time in handling temporal information. Since there are two orthogonal time axes involved now, temporal relations should be illustrated in four dimensions (Figure 4 shows a single temporal relation).

A temporal relation may be regarded as a sequence of historical states, each a complete historical relation. The rollback operation on a temporal relation selects a particular historical state, on which a historical query can be executed. Each transaction causes a new historical state to be created; hence, temporal relations are append-only. The temporal relation in Figure 4 is the result of four transactions, starting from a null relation: (1) three tuples were added, (2) one tuple was added, (3) one tuple was added and an existing one deleted, and (4) one tuple was modified so that it started at a later valid time.

Since TQuel supports both historical queries and rollback operations, it can be used to query temporal databases. An example is the TQuel query

```
range of f1 is Faculty
range of f2 is Faculty
retrieve (f1.Rank)
  where f1.Name = "Merrie" and
        f2.Name = "Tom"
  when f1 overlap begin of f2
  as of "January 1979"
```

that determines Merrie's rank when Tom arrived, according to the state of the database as of January 1979. This derived relation is a temporal relation, so further temporal relations can be derived from it. The answer may differ if a similar query is made as of October 1978 and if her promotion was not yet recorded by that time.

User-defined time. *User-defined time* is necessary when additional temporal information, not handled by transaction or valid time, is stored in the database. The values of user-defined temporal attributes are not interpreted by the DBMS and are thus the easiest to support; only an internal representation and input/output functions are needed. Such attributes will then occur in the relation scheme. Multiple representations are also possible, each associated with input and output functions. As an example of user-defined time, consider a Promotion relation with three attributes: Name, Rank, and Approved-Date. The Approved-Date (a user-defined time) is the date when the promotion committee approved the promotion; the valid time is the date when the promotion will take effect; and the transaction time is the date the information concerning the promotion was stored in the database.

An analogy. A simple analogy will help clarify the subtle differences among the four types of databases categorized above.

	No rollback	Rollback
Current queries	Snapshot	Rollback
Historical queries	Historical	Temporal

Figure 5. Types of databases.

	Transaction	Valid
Snapshot		
Rollback	√	
Historical		√
Temporal	√	√

Figure 6. Attributes of the four types of databases.

First, a snapshot relation can be compared to the latest payroll stub showing the current position of the recipient. If the person gets a promotion, the next stub shows the new rank, but there is no way to find out about a past rank from the latest stub.

If all the payroll stubs are carefully collected without discarding any of them, as some people do, it is possible to determine the rank as of some time in the past from the stub of that period. This collection of payroll stubs can be compared to a rollback relation, a slice of which is a snapshot relation comparable to a payroll stub. These stubs will be printed in indelible ink, so that it will not be possible to make changes to payroll stubs of the past, even when there is a retroactive promotion or an error in last year's salary.

A historical relation can be compared to a resume, containing the history of job positions a person held up through the moment the resume was prepared. If an error is found in the resume, or if the person gets a promotion, a new resume reflecting the change is in order. The current resume should always be up to date.

A temporal relation can be considered as a collection of all such resumes marked by the date when each of them was prepared. It is possible and often interesting to go back to an old resume and read about the personal data as known at some past moment.

An analogy for user-defined time would be the date printed on each payroll stub indicating when the pay period started. Note that this date does not necessarily correspond to the date the check (or the previous one) was issued.

Summary of taxonomy. Three kinds of time—transaction time, valid time, and user-defined time—were introduced, resulting in a categorization of the types of database management systems based on their support for handling temporal information. As shown in Figure 5, transaction time and valid time define the two orthogonal capabilities of rollback operations and historical queries, thereby differentiating four types of databases: snapshot, rollback, historical, and temporal.

Support of rollback operations requires the incorporation of transaction time, which concerns the representation. Support of historical queries requires the incorporation of valid time, which is associated with reality. Support of user-defined time is orthogonal to support of historical queries or rollback operations. Hence, the three kinds of time actually define eight different types of databases. The taxonomy presented here defines four types based on their support of transaction and valid time (see Figure 6). Each of these types may or may not support user-defined time. However, we note that user-defined time is much closer to valid time than to transaction time, in that both valid time and user-defined time are concerned with reality itself, whereas transaction time involves only the model of reality (the database). DBMSs (and their query languages) purporting to provide full temporal support should handle all three kinds of time.

An example

The following example highlights the similarities and differences among the four types of relations, snapshot, rollback, historical, and temporal. We first create an empty relation for each of the four types using TQuel create statements:

```
create Snapshot (Name = c20, Rank =
                 c20)
create persistent Rollback (Name = c20,
                 Rank = c20)
create interval Historical (Name = c20,
                 Rank = c20)
create persistent interval Temporal (Name
                 = c20, Rank = c20)
```

These are the relations alluded to earlier, namely, the latest payroll check (snapshot), the collection of all past payroll stubs (rollback), the most current resume (historical), and the collection of all past resumes (temporal).

Starting with these empty relations, a series of update operations are applied to each of them. After each update operation, the states of four relations are displayed: the snapshot relation in a conventional format as a single snapshot state, the rollback relation as a sequence of snapshot states, the historical relation as a single historical state, and the temporal relation as a sequence of historical states. Several queries on these relations focus on what information is and, more importantly, is not represented in each relation.

Merrie joins as an instructor. In September 1973, the following statement is executed:

```
append to ? (Name = "Merrie", Rank =
    "Instructor")
```

In these examples, the "?" symbol would be replaced by the name of a relation: snapshot, rollback, historical, or temporal.

The four relations resulting from the execution of this statement are almost identical (see Figure 7). The snapshot relation shows that Merrie is currently an instructor. The rollback relation contains a single snapshot state created on September 1973 (the transaction time that indexes the snapshot states is shown following the state), indicating that Merrie started receiving payroll checks made out to "Instructor Merrie" from September 1973. The historical relation indicates that Merrie has been hired as an instructor (the valid time for each tuple in the historical state is shown to the left of the tuple); there is currently one line on Merrie's resume. The temporal relation contains one historical state, containing one tuple; analogously, there is one resume with one entry in Merrie's resume file.

If we asked back in October 1973 "What was Merrie's rank?",

```
range of f is ?
retrieve (f.Rank)
  where f.Name = "Merrie"
```

the same information would be returned from all four relations ("Instructor"), but in rather different ways. For the rollback relation, the current snapshot state is used; for the historical relation, the tuples currently valid are searched; and for the temporal relation, the tuples currently valid in the current historical state are searched. The defaults defined in TQuel ensure that queries containing only where clauses will return the same information regardless of relation types.[7]

Snapshot:

Name	Rank
Merrie	Instructor

Rollback:

Name	Rank	(Transaction Time)
Merrie	Instructor	(September 1973)

Historical:

(Valid Time)	Name	Rank
(September 1973)	Merrie	Instructor

Temporal:

(Valid Time)	Name	Rank	(Transaction Time)
(September 1973)	Merrie	Instructor	(September 1973)

Snapshot:

Name	Rank
Merrie	Full Professor

Rollback:

Name	Rank	(Transaction Time)
Merrie	Instructor	(September 1973)
....	....	....
Merrie	Full Professor	(December 1973)

Historical:

(Valid time)	Name	Rank
(September 1973)	Merrie	Instructor
(December 1973)	Merrie	Full Professor

Temporal:

(Valid Time)	Name	Rank	(Transaction Time)
(September 1973)	Merrie	Instructor	(September 1973)
....	....	....	....
(September 1973)	Merrie	Instructor	(December 1973)
(December 1973)	Merrie	Full Professor	

Figure 8. Merrie is promoted to full professor.

Merrie is promoted to full professor. In December of that year, a replace statement is executed:

```
replace f (Rank = "Full Professor")
  where f.Name = "Merrie"
```

Figure 8 illustrates the changes in the four relations. Since the snapshot relation records only the current information, the existing tuple is modified, as is the next payroll check made out to Merrie. A new snapshot state is appended to the rollback relation; Merrie's pay stubs for September 1973 through November 1973 still read "Instructor Merrie," but the December 1973 paycheck is made out to "Full Professor Merrie." Snapshot states within the rollback relation are separated by dotted lines. A tuple is added to the historical relation with a valid time of December 1973; an entry is also added to Merrie's resume. The historical relation always contains only one historical state, so no dotted lines will appear in its illustration.

The historical relation is an *interval* relation. In the representation shown in Figure 8, a tuple is valid until the next tuple with the same key becomes effective. Hence, the historical relation in this figure indicates that Merrie was an instructor from September 1973 until December 1973, when she became a full professor. The temporal relation contains two historical states: one current from September through November 1973 and the longer one created in December 1973. Merrie's resume file now contains two resumes, one dated September 1973 containing one job position, and the more recent one dated December 1973 containing two job positions. Only one transaction time is needed for each historical state, even if it contains multiple tuples.

When we ask the next month, January 1974, about Merrie's rank,

```
retrieve (f.Rank)
  where f.Name = "Merrie"
```

we again get the same result from all four relations ("Full Professor"), except that we also get the past rank of Instructor from the historical and the temporal relations. If we ask in January, "What was Merrie's rank last October ?":

```
retrieve (f. Rank)
  where f.Name = "Merrie"
  when f overlap "October, 1973"
```

we run into some difficulties. This query cannot be executed on a snapshot relation, since information about the past is not stored (looking at Merrie's pay stub from January won't tell us what the paycheck read three months earlier). The rollback relation can't give us an answer either. Of course, we could flip through the payroll stubs, but such a search won't tell us if Merrie was given a retroactive promotion (such a situation will be examined shortly). Both the historical and temporal relations can provide the answer "Instructor" by examining the current resume (the historical relation records only the current one anyway).

Still interacting with the DBMS in January 1974, we ask "What did we think Merrie's current rank was three months ago?":

```
retrieve (f.Rank)
  where f.Name = "Merrie"
  as of "October, 1973"
```

This query effectively turns back the clock to October; all changes after October are not considered. A result is not forthcoming from either the snapshot or the historical relations, because they both record only current knowledge (in the case of historical relations, current knowledge about the past). In this case, however, flipping through the pay stubs or the stack of resumes (the rollback and temporal relations, respectively) will allow us to determine what we knew in October 1973: that Merrie was then an instructor.

Snapshot:

Name	Rank
Merrie	Assistant Professor

Rollback:

Name	Rank	(Transaction Time)
Merrie	Instructor	(September 1973)
....	....	....
Merrie	Full Professor	(December 1973)
....	....	....
Merrie	Assistant Professor	(February 1974)

Historical:

(Valid Time)	Name	Rank
(September 1973)	Merrie	Instructor
(December 1973)	Merrie	Assistant Professor

Temporal:

(Valid Time)	Name	Rank	(Transaction Time)
(September 1973)	Merrie	Instructor	(September 1973)
....	....	....	....
(September 1973)	Merrie	Instructor	(December 1973)
(December 1973)	Merrie	Full Professor	
....	....	....	....
(September 1973)	Merrie	Instructor	(February 1974)
(December 1973)	Merrie	Assistant Professor	

Figure 9. A correction.

A correction. In the next month, however, we realize that we made a mistake. Last December, Merrie wasn't promoted from instructor to full professor; she was only promoted to assistant professor. We modify the historical and the temporal relations in February 1984 by

```
replace f (Rank = "Assistant Professor")
  valid from begin of "December 1973"
  where f.Name = "Merrie"
```

but we can correct only the current state of the snapshot and rollback relations by

```
replace f (Rank = "Assistant Professor")
  where f.Name = "Merrie"
```

Figure 9 shows that the next paycheck will indicate a new rank, paychecks issued from February 1974 on will bear the correct title, and the current resume is corrected.

Note that the pay stubs for December 1973 and January 1974 still mention "Full Professor Merrie" and that the resume file still contains an old resume with the incorrect promotion rank; both of these relations are by definition append-only.

We perform three queries in August. We first ask about Merrie's current rank:

```
retrieve (f.Rank)
  where f.Name = "Merrie"
```

All four relations respond with "Assistant Professor," except that we also get the past rank of "Instructor" from the historical and temporal relations. When asked what Merrie's rank was in January,

```
retrieve (f.Rank)
  where f.Name = "Merrie"
  when f overlap "January 1974"
```

the snapshot and rollback relations cannot provide an answer. However, the historical and temporal relations both respond with the corrected rank, "Assistant Professor."

If we ask a different question, "What was Merrie's current rank as best known last January?",

```
retrieve (f.Rank)
  where f.Name = "Merrie"
  as of "January 1974"
```

then the snapshot and historical relations cannot reply, since they only record information as currently known. Both the rollback and temporal relations can provide the information we request ("Full Professor"), while the temporal relation also provides the past rank of "Instructor."

Merrie is promoted retroactively. In December 1978, Merrie is promoted to associate professor, retroactive from June 1978. We record the fact in the historical and temporal relations by

```
replace f (Rank = "Associate Professor")
  valid from begin of "June 1978"
  where f.Name = "Merrie"
```

but we can modify only the current state of the snapshot and rollback relations by

```
replace f (Rank = "Associate Professor")
  where f.Name = "Merrie"
```

As shown in Figure 10, the fact that the promotion was retroactive is irrelevant in the snapshot and rollback relations. In particular, the pay stubs (the rollback relation) from February 1974 to November 1978 still specify "Assistant Professor Merrie." However, the current resume (the historical relation) and the most recent resume in the resume file (the temporal relation) will both record the promotion date as June 1978.

When we query the relations in the following March, all four will list Merrie's current rank as "Associate Professor." The historical and temporal relations will list her rank in October 1978 as "Associate Professor." The rollback and temporal relations will list her current rank, as best known in October 1978, as "Assistant Professor," indicating that the promotion had not been made. However, the temporal relation can answer even more involved queries, such as "What was Merrie's rank in October 1978, as best known in November 1978?":

```
retrieve (f.Rank)
  when f overlap "October, 1978"
  where f.Name = "Merrie"
  as of "November 1978"
```

This query can only be answered by the temporal relation, which returns a rank of "Assistant Professor." If the as of clause were omitted, the result would be "Associate Professor."

We have examined the information represented by each of the four types of relations and have shown how each relation responds to various update and retrieval operations. A few tautologies should now be defensible:

- The most recent snapshot state in the rollback relation is always identical to the entire contents of the snapshot relation.
- Similarly, the most recent historical state in the temporal relation is always identical to the entire contents of the historical relation.
- Queries containing only a target list and a where clause will return the same information when applied to any of the four types of relations.

An implementation

There are several approaches to implementing a DBMS supporting the operations described above. When confronted with the task of adding temporal support to a DBMS, one reasonable initial strategy would be to interpose a layer of code between the user and the database system. This layered approach has the significant advantage of not requiring any change to the complex data structures and algorithms within the DBMS proper. However, the performance of such a system may be inadequate. An immediate concern is the monotonically increasing and potentially enormous storage requirements. In addition, there are multiple versions for some tuples, rendering conventional storage schemes such as indexing or hashing less effective. Performance will deteriorate rapidly not only for temporal queries but also for nontemporal queries.

An alternative is to integrate temporal support into the DBMS itself, developing new query evaluation algorithms and access methods to achieve reasonable performance for a variety of temporal queries without penalizing more frequent nontemporal queries. This approach clearly involves substantial research and implementation effort, yet promises to address deficiencies in performance.

We have adopted an intermediate strategy in implementing our prototype temporal DMBS. We modified portions of the snapshot DBMS Ingres,[6] but still used the conventional access methods and query evaluation algorithms available in it. Such a strategy may also be useful when adding temporal support to a conventional hierarchical or network DBMS. This prototype helps identify problems with conventional access methods and query evaluation algorithms in providing temporal support, and serves as a comparison point for fully integrated DBMSs developed in the future. We also used it to run the example queries above, where it flagged a few subtle errors that had escaped our scrutiny.

One of the most important decisions was determining how to embed a four-dimensional temporal relation into a two-dimensional snapshot relation as supported by Ingres. There are at least five ways to embed a temporal relation in a snapshot relation.[7] Our prototype adopted the scheme of augmenting each tuple with two transaction time attributes for a rollback and a temporal relation, and one or two valid time attributes for a historical and a temporal relation, depending on whether the relation modeled events or intervals. Each update operation on an existing tuple generates a new version of the tuple, marked with appropriate values of

Snapshot:

Name	Rank
Merrie	Associate Professor

Rollback:

Name	Rank	(Transaction Time)
Merrie	Instructor	(September 1973)
Merrie	Full Professor	(December 1973)
Merrie	Assistant Professor	(February 1974)
Merrie	Associate Professor	(December 1978)

Historical:

(Valid Time)	Name	Rank
(September 1973)	Merrie	Instructor
(December 1973)	Merrie	Assistant Professor
(June 1978)	Merrie	Associate Professor

Temporal:

(Valid Time)	Name	Rank	(Transaction Time)
(September 1973)	Merrie	Instructor	(September 1973)
(September 1973)	Merrie	Instructor	(December 1973)
(December 1973)	Merrie	Full Professor	
(September 1973)	Merrie	Instructor	(February 1974)
(December 1973)	Merrie	Assistant Professor	
(September 1973)	Merrie	Instructor	(December 1978)
(December 1973	Merrie	Assistant Professor	
(June 1978)	Merrie	Associate Professor	

Figure 10. Merrie is promoted retroactively.

time attributes indicating the period the version is active. Although this tuple versioning scheme appears to have a high degree of redundancy, in that the entire tuple is duplicated whenever the value of any attribute changes, analysis reveals that this scheme consumes less space than attribute versioning when the database is not volatile.[8]

For the prototype, the parser of Ingres was modified to accept TQuel statements and generate an extended syntax tree with subtrees for valid, when, and as of clauses. Some of the query evaluation modules were changed to handle the newly defined node types and the implicit temporal attributes. Functions to handle the temporal operators begin of, end of, precede, overlap, extend, and as of were added in the interpreter. The system relation was modified to support the various combinations of implicit temporal attributes according to the type of relation as specified by its create statement, an example of which appeared earlier. The temporal attribute has a distinct type, so that input and output of time values can be done in human readable form by automatically converting to and from the internal representation. Various formats of date or time are accepted for input, and resolutions ranging from second to year are selectable for output. The copy statement was also modified to support batch input and output of relations having temporal attributes.

The resulting prototype supports all the TQuel statements, including the augmented create, append, delete, replace, and retrieve statements. The valid, when, and as of clauses are fully supported. All three kinds of time are supported, as are all four types of relations. Proposed extensions to the language, including indeterminacy and aggregate functions, are not yet supported.

To evaluate the performance of the prototype, we ran a benchmark with a set of queries on the four types of databases using various access methods. We determined the fixed portion, the variable portion, and the growth rate of the disk access cost for various types of queries, as the number of versions in each database increased using a series of replace operations.[9] Performance degraded rapidly for both temporal and nontemporal queries, as expected for conventional access methods such as hashing and indexed sequential access. We are investigating new access methods addressing this problem to enhance the performance.

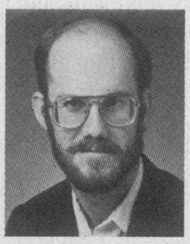
While fifteen years of research have focused on formalizing and implementing snapshot databases, only a few researchers have recently studied the formalization of historical databases and the implementation of rollback databases. Little has been published on formalizing rollback or temporal databases, or on implementing historical or temporal databases. The special opportunities promised by temporal databases are, at this time, matched by the challenges in supporting them.

Acknowledgments

This work was supported by NSF grant DCR-8402339. Richard Snodgrass' work was also supported by an IBM Faculty Development Award.

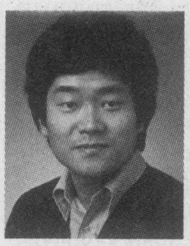
References

1. A. Bolour et al., "The Role of Time in Information Processing: A Survey," *SigArt Newsletter,* Vol. 80, Apr. 1982, pp. 28-48.
2. A. Hoagland, "Information Storage Technology: A Look at the Future," *Computer,* Vol. 18, No. 7, July 1985, pp. 60-67.
3. G. Copeland, "What If Mass Storage Were Free?," *Computer,* Vol. 15, No. 7, July 1982, pp. 27-35.
4. G. Weiderhold, "Databases," *Computer,* Vol. 17, No. 10, Oct. 1984, pp. 211-223.
5. R. Snodgrass and I. Ahn, "A Taxonomy of Time in Databases," *Proc. Int'l Conf. Management of Data,* ACM SIGMOD, Austin, TX, May 1985, pp. 236-246.
6. G. D. Held, M. Stonebraker, and E. Wong, "Ingres—A Relational Data Base Management System," *Proc. 1975 Nat'l Computer Conf.,* Vol. 44, 1975, pp. 409-416.
7. R. Snodgrass, "The Temporal Query Language TQuel," *ACM Trans. Database Systems,* to appear 1986.
8. I. Ahn, "Towards an Implementation of Database Management Systems with Temporal Support," *Second Int'l Conf. Data Engineering,* Los Angeles, CA, Feb. 1986, pp. 374-381.
9. I. Ahn and R. Snodgrass, "Performance Evaluation of a Temporal Database Management System," *Proc. Int'l Conf. Management of Data,* ACM SIGMOD, Washington DC, May 1986, pp. 96-107.



Richard Snodgrass is currently an assistant professor at the Department of Computer Science, University of North Carolina, Chapel Hill. His research interests include temporal databases, programming environments, and software monitoring. He is the director of the SoftLab project, which is concerned with pragmatic issues in the implementation of programming environments, database management systems, and operating systems.

Snodgrass received the BA degree in physics from Carleton College, Northfield, MN, in 1977 and the PhD degree in computer science from Carnegie-Mellon University, Pittsburgh, PA, in 1982.



Ilsoo Ahn is a member of the technical staff at Bell Laboratories, Columbus, Ohio. His research interests include temporal databases, physical database design, performance analysis, distributed systems and computer networks.

Ahn worked as an engineer for telephone switching systems and as an MVS system programmer in the Korea Telecommunications Co. after receiving an MSEE from the Korea Advanced Institute of Science in 1977. He received a PhD in computer science from the University of North Carolina at Chapel Hill in 1986.

Readers may write to Snodgrass at the University of North Carolina at Chapel Hill, Dept. of Computer Science, Chapel Hill, NC 27514.