TR85-027

Temporal Databases

Richard Snodgrass & Ilsoo Ahn

Department of Computer Science
University of North Carolina
Chapel Hill, North Carolina 27514

August, 1985

Abstract

Recognizing the need for providing temporal support in database management systems, and spurred by the rapid decrease of storage cost in recent years, many authors have proposed systems incorporating one or more time attributes in the database management systems. This paper presents a taxonomy of three times, which defines four types of databases differentiated by their ability to represent temporal information. An example highlights the similarities and differences between the types of databases. A prototype implementation is briefly described.

A preliminary version of the material in the first third of this paper appeared in the ***Proceedings of the ACM SIGMOD Conference on the Management of Data*** [Snodgrass & Ahn 1985]. This work was supported by NSF grant DCR-8402339. The first author was also supported by an IBM Faculty Development Award.

Table of Contents

List of Figures

1. Introduction

Time is an essential part of information concerning the real world, which evolves constantly. 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 are supposed to model reality, but conventional database management systems (DBMS's) lack the capability to record and process time-varying aspects of the real world, as will be discussed further in Section 2.1. With increasing sophistication of DBMS applications, the lack of temporal support raises serious problems in many cases. For example, conventional DBMS's cannot support *historical queries* about the past status, let alone *trend analysis* which is essential for applications such as decision support systems [Ariav 1984]. There is no way to represent *retroactive* or *postactive* changes, while support for *error correction* or *audit trail* necessitates costly maintenance of backups, checkpoints, journals 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 the capability to store and process time dependent data [Katz & Lehman 1984]. Without temporal support from the system, many applications have had to maintain and handle temporal information to support such functions in an ad-hoc manner.

The need for providing temporal support in DBMS's has been recognized for at least a decade [Bubenko 1976, Schueler 1977]. A recent bibliography contained about 70 articles relating time and information processing [Bolour et al. 1982]; at least 30 more articles have appeared in the literature since 1982. Recently, the rapid decrease of storage cost, coupled having the emergence of promising new mass storage technologies such as optical disks [Ammon et al. 1985, Hoagland 1985], have amplified interest in database management systems with version management or temporal support. G. 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. [Copeland 1982]

G. 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. [Wiederhold 1984]

In the past five years, numerous schemes have been proposed to support temporal information processing in database management systems by incorporating one or two time attributes in the database management systems. However, there has been some confusion concerning terminology and the definition of these time attributes. In this paper, we describe a taxonomy of time consisting of three distinct time concepts for use in databases [Snodgrass & Ahn 1985]. Using the taxonomy, we define four types of databases, differentiated by their ability to support these time concepts and processing of temporal information. A series of example transactions highlights the similarities and differences among the four types of databases. We also describe a prototype implementation.

2. The Taxonomy

In this section we introduce the taxonomy of time for use in databases. Though the following discussion is based on the relational model, similar arguments also apply to hierarchical or network models. We will first discuss static databases, focusing on their representational inadequacies. We then define three new time concepts and discuss the features associated with particular types of DBMS's supporting various combinations of these time concepts.

2.1. Static Databases

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

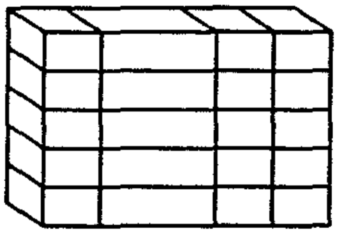


Figure 1: A Static Relation

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 2-dimensional table (see Figure 1). As changes occur in the real world, changes are made in this table. For example, an instance of a relation 'Faculty', with two attributes Name and Rank, at a certain moment may be

Name	Rank
Merrie	Associate Professor
Tom	Assistant Professor

and a query in Quel, a tuple calculus based language for the INGRES database management system [Held et al. 1975], requesting Merrie's rank,

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

yields the rank of Associate Professor.

There are many situations where this static database relying on snapshots is inadequate. For example, it cannot answer queries such as

What was Merrie's rank 2 years ago? (historical query)

How did the number of faculty change over the last 5 years? (trend analysis)

nor record facts like

Merrie was promoted to a full professor starting last month. (retroactive change)

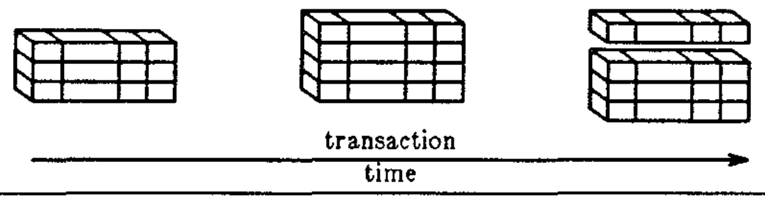
Lee is joining the faculty next month. (postactive change)

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, is implicit. The DBMS provides no facilities for the maintenance, querying, or modification of this information; such operations must be provided by specially-written applications programs that interpret certain attributes (i.e., those encoding the time values) differently. The fact that data varies over time is not an application specific aspect. Hence, this aspect should be supported in a general fashion by the DBMS, rather than by the application programs.

2.2. Rollback Databases

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

Figure 2: A Rollback Relation



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

The distinction between static databases and rollback databases is the ability to return to any previous state to execute a (static) query. Any query language may be converted to one which may query a rollback database by adding a clause effecting the rollback. TQuel (*T*emporal *QUE*ry *L*anguage) [Snodgrass 1984, Snodgrass 1985], 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 "Sep, 1978"
```

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

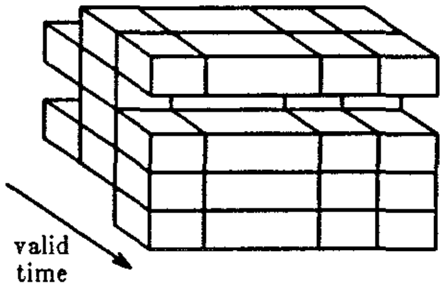
One limitation of supporting transaction time is that the history of database activities, rather than the history of the real world, is recorded. A tuple becomes valid as soon as it is entered into the database, as in a static database. There is no way to record retroactive/postactive changes, nor to 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 was discovered that Merrie's promotion had occurred earlier than previously thought, this error could not be corrected in a rollback database, and the query given above would continue to respond with the incorrect rank.

2.3. Historical Databases

While rollback databases record a sequence of static states, *historical databases* record a single *historical state* per relation, storing the history as it is best known. As errors are discovered, they are corrected by modifying the database. Previous states are *not* retained, so it is not possible to view the database as it was in the past. There is no record kept of the errors that have been corrected. Historical databases are similar to static databases in this respect. Historical databases must represent *valid time*, the time during which the relationship in the real world being modeled was valid.

Historical databases may also be illustrated in three dimensions (see Figure 3), where the label of the time axis indicates the valid time. The semantics are more closely related to reality, instead of update history. Both valid time and user-defined time concern modeling of reality, and so it is appropriate that they should appear together.

Figure 3: Historical Relation



More sophisticated operations are necessary to manipulate the complex semantics of valid time adequately, compared to the simple rollback operation. TQuel supports the expression of 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 *start of*, *precede*, and *overlap*. As an 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 start of f2
```

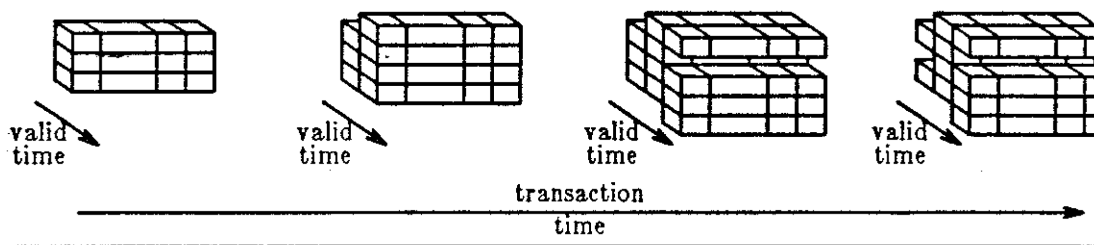
The derived relation is also an historical relation, which may be used in further historical queries.

A second distinction between historical and rollback databases is that historical DBMS's support arbitrary modification, whereas static rollback DBMS's only allow static states to be appended. The same sequence of transactions which resulted in the rollback relation in Figure 2 also results in the historical relation in Figure 3. However, a later transaction (not possible on a rollback relation) has removed an erroneous tuple inserted on the first transaction. Rollback DBMS's can rollback to an incorrect previous static relation; historical DBMS's can record the current knowledge about the past.

2.4. 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 tuples valid at some time as of that time, and a historical database always views tuples valid at some moment as of *now*, a temporal DBMS makes it possible to view tuples valid at some moment seen as of some other moment, completely capturing the history of retroactive/postactive changes.

Figure 4: A Temporal Relation



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 time axes involved now, temporal relations should be illustrated in four dimensions (Figure 4 shows a *single* temporal relation). A temporal relation may be thought of as a sequence of historical states, each of which is a complete historical relation. The rollback operation on a temporal relation selects a particular historical state, on which an historical query may be performed. 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) a previous tuple was deleted (presumably it should not have been there in the first place).

Since TQuel supports both historical queries and rollback, it may 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 start of f2
      as of "Jan, 1979"
```

which determines Merrie's rank when Tom arrived, according to the state of the database as of Jan, 1979. This derived relation is a temporal relation, so further temporal relations can be derived from it. If a similar query is made as of Oct, 1978, the answer may be different if the retroactive promotion was not yet recorded by that time.

2.5. User-defined time

User-defined time [Jones & Mason 1980] 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; all that is needed is an internal representation and input and output functions. Such attributes will then be present 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 Effective-Date. The Effective-Date (a user-defined time) is the date stated in the promotion letter that the promotion was to take effect; the valid date is the date the promotion letter was signed, i.e., the date the promotion was validated; and the transaction date is the date the information concerning the promotion was stored in the database. The effective date is application-specific; it is merely a date which appears in the promotion letter.

2.6. An Analogy

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

First, a static relation can be compared to a 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 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 state of which is a static 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.

An historical relation can be compared to a resume, containing the history of job positions a person went through up to the moment the resume was prepared. If an error is found in the contents of the resume, or one gets a promotion whether retroactively or postactively, a new resume is in order reflecting the change accordingly.

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 check) was issued.

2.7. Summary of the 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 and valid time define the two orthogonal capabilities of rollback and historical queries, thereby differentiating four types of databases: static, rollback, historical and temporal.

Figure 5: Types of Databases

		Transaction Time	
		No Rollback	Rollback
Valid	Static Queries	Static	Rollback
Time	Historical Queries	Historical	Temporal

Support of the rollback capability 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 and to support of rollback. 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 involve *reality* itself, as opposed to transaction time which involves only the *model* of reality (i.e., the database). DBMS's (and their query languages) purporting to fully support temporal information should support all three kinds of time. Section 4 will briefly describe one such DBMS based on TQuel.

Figure 6: Attributes of the New Types of Databases

	Transaction	Valid
Static		
Rollback	✓	
Historical		✓
Temporal	✓	✓

3. An Example

The following example highlights the similarities and differences among the four types of databases. Starting with an empty relation, a series of update operations are performed. Each update is applied to four relations, one of each type discussed previously. Each relation is then displayed: the static relation in a conventional format as a single static state, the rollback relation as a sequence of static 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, perhaps, *is not* stored in each relation.

The schemes for the four relations may be expressed in TQuel as follows:

```
create Static (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 (Static), the collection of all past payroll stubs (Rollback), the most current resume (Historical), and the collection of all past resumes (Temporal).

3.1. Merrie Joins as an Instructor

In September, 1973, the following statement is executed:

```
append to ? (Name = "Merrie", Rank = "Instructor")    [Sep, 1973]
```

In these examples, when a TQuel statement is given, the date the statement was executed by the database management system is shown in brackets to the right of the statement. The "?" would be replaced by the relation's name, i.e., Static, Rollback, Historical, or Temporal.

The four relations resulting from the execution of this statement are almost identical (see Figure 7). The Static relation shows that Merrie is currently an instructor. The Rollback relation contains a single static state that was created on September, 1973 (the transaction time that indexes the static states is shown on the right of the state), indicating that Merrie started receiving payroll checks made out to "Instructor Merrie" in September 1983. The Historical relation indicates that Merrie has been hired as an Instructor (the valid time for each tuple in the historical state is shown on 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.

Figure 7: Merrie joins as an Instructor

Static:

Name	Rank
Merrie	Instructor

Rollback:

Name	Rank	(Transaction Time)
Merrie	Instructor	(Sep, 1973)

Historical:

(Valid Time)	Name	Rank
(Sep, 1973)	Merrie	Instructor

Temporal:

(Valid Time)	Name	Rank	(Transaction Time)
(Sep, 1973)	Merrie	Instructor	(Sep, 1973)

If we asked back in October, 1973 what was Merrie's rank,

```
range of f is ?
retrieve (f.Rank)
    where f.Name ="Merrie"              [Oct, 1973]
```

the same information would be returned from all four relations: "Instructor," but in rather different ways. For the Rollback relation, the current static 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 type [Snodgrass 1985].

3.2. Merrie is promoted to Full Professor

Later that year (1973), a replace statement is executed:

```
replace f (Rank = "Full Professor")
    where f.Name = "Merrie"             [Dec, 1973]
```

Figure 8 illustrates the changes in the four relations.

Figure 8: Merrie is promoted to Full Professor

Static:

Name	Rank
Merrie	Full Professor

Rollback:

Name	Rank	(Transaction Time)
Merrie	Instructor	(Sep, 1973)
....	....	....
Merrie	Full Professor	(Dec, 1973)

Historical:

(Valid Time)	Name	Rank
(Sep, 1973)	Merrie	Instructor
(Dec, 1973)	Merrie	Full Professor

Temporal:

(Valid Time)	Name	Rank	(Transaction Time)
(Sep, 1973)	Merrie	Instructor	(Sep, 1973)
....	....	....	....
(Sep, 1973)	Merrie	Instructor	(Dec, 1973)
(Dec, 1973)	Merrie	Full Professor	

Since the Static relation always records current information, the one tuple is modified, as is the next payroll check made out to Merrie. A new static 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 pay check is made out to "Full Professor Merrie." Static 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, and an entry is also added to Merrie's resume. The Historical relation always contains only one historical state, so no dotted lines will ever 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 is valid. 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 which was current from September through November, 1973 and a longer one that was created in December, 1973. Merrie's resume file now contains two resumes, one dated

September, 1973 and containing one job position, and the more current one dated December, 1973 and 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 about Merrie's rank;

```
retrieve (f.Rank)
     where f.Name = "Merrie"             [Jan, 1974]
```

we again get the same result from all four relations: "Full Professor." If we ask *in January* what was Merrie's rank last October:

```
retrieve (f.Rank)
     where f.Name = "Merrie"
     when f overlap "Oct, 1973"       [Jan, 1974]
```

we run into some difficulties. This query cannot be executed on a static relation, since information about the past is not stored (looking at Merrie's pay stub from January won't tell us what the pay check read three months prior). 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 "Oct, 1973"             [Jan, 1974]
```

This query effectively turns back the clock to October; all changes after October are not considered. A result is not forthcoming from either the Static 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 currently an Instructor.

3.3. A Correction

However, in the next month we realize we have made a mistake, and we correct it:

```
replace f (Rank = "Assistant Professor")
     valid from "Dec, 1973"
     where f.Name = "Merrie"            [Feb,1974]
```

Last December, Merrie wasn't promoted from Instructor to Full Professor; she was only promoted to Assistant Professor. Figure 9 shows that the next pay check will indicate a new rank, pay checks issued from February, 1974 on bear the correct title, and the current resume is corrected.

Figure 9: A Correction

Static:

Name	Rank
Merrie	Assistant Professor

Rollback:

Name	Rank	(Transaction Time)
Merrie	Instructor	(Sep, 1973)
....	....	....
Merrie	Full Professor	(Dec, 1973)
....	....	....
Merrie	Assistant Professor	(Feb, 1974)

Historical:

(Valid Time)	Name	Rank
(Sep, 1973)	Merrie	Instructor
(Dec, 1973)	Merrie	Assistant Professor

Temporal:

(Valid Time)	Name	Rank	(Transaction Time)
(Sep, 1973)	Merrie	Instructor	(Sep, 1973)
....	....	....	....
(Sep, 1973)	Merrie	Instructor	(Dec, 1973)
(Dec, 1973)	Merrie	Full Professor	
....	....	....	....
(Sep, 1973)	Merrie	Instructor	(Feb, 1974)
(Dec, 1973)	Merrie	Assistant Professor	

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 later that summer. We first ask for Merrie's current rank,

```
retrieve (f.Rank)
     where f.Name = "Merrie"              [Aug, 1974]
```

All four relations respond with "Assistant." When asked what Merrie's rank was in January,

```
retrieve (f.Rank)
     where f.Name = "Merrie"
     when f overlap "Jan, 1974"       [Aug, 1974]
```

the Static and Rollback relations cannot provide an answer. However, the Historical and Temporal Relations both respond with "Assistant Professor," supplying the corrected rank.

If, instead we ask the different question, what was Merrie's current rank as was best known last January,

```
retrieve (f.Rank)
     where f.Name = "Merrie"
     as of "Jan, 1974"            [Aug, 1974]
```

then the Static and Historical relations cannot reply, since they only record information as is best known currently. Both the Rollback and Temporal relations can provide the information we request: "Full Professor."

3.4. Merrie is promoted retroactively

In December, 1978, Merrie is promoted to Associate Professor, retroactive to June, 1978:

```
replace f (Rank = "Associate")
     valid from "Jun, 1978"
     where f.Name = "Merrie"              [Dec, 1978]
```

As shown in Figure 10, the fact that the promotion was retroactive is irrelevant in the Static and Rollback relations. In particular, the pay stubs (i.e., 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.

Figure 10: Merrie is promoted retroactively

Static:

Name	Rank
Merrie	Associate Professor

Rollback:

Name	Rank	(Transaction Time)
Merrie	Instructor	(Sep, 1973)
....	....	....
Merrie	Full Professor	(Dec, 1973)
....	....	....
Merrie	Assistant Professor	(Feb, 1974)
....	....	....
Merrie	Associate Professor	(Dec, 1978)

Historical:

(Valid Time)	Name	Rank
(Sep, 1973)	Merrie	Instructor
(Dec, 1973)	Merrie	Assistant Professor
(Jun, 1978)	Merrie	Associate Professor

Temporal:

(Valid Time)	Name	Rank	(Transaction Time)
(Sep, 1973)	Merrie	Instructor	(Sep, 1973)
....	....	....	....
(Sep, 1973)	Merrie	Instructor	(Dec, 1973)
(Dec, 1973)	Merrie	Full Professor	
....	....	....	....
(Sep, 1973)	Merrie	Instructor	(Feb, 1974)
(Dec, 1973)	Merrie	Assistant Professor	
....	....	....	....
(Sep, 1973)	Merrie	Instructor	(Dec, 1978)
(Dec, 1973)	Merrie	Assistant Professor	
(Jun, 1978)	Merrie	Associate Professor	

When we query the relations 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 Static 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 "Oct, 1978"
     where f.Name = "Merrie"
     as of "Nov, 1978"                    [Mar, 1979]
```

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

In summary, we have examined the information stored by each of the four types of relation, and have shown how each relation responds to various update and retrieval operations. A few tautologies should now be defensible:

- The most recent static state in the Rollback relation is always identical to the entire contents of the Static relation.
- Similarly, the most recent historical state in the Temporal relation is always identical to the entire contents of the Historical relation.
- The results of a query containing only a target list and a where clause will be the same in the explicit attributes (and in the valid and transaction times, when present) when applied to any of the four types of relations.

4. 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. The 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 non-temporal 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 non-temporal queries. There are several issues which need to be addressed for this integrated approach, including tuple *vs.* attribute versioning [Ahn 1985]. This approach clearly involves substantial research and implementation effort, yet holds the promise for addressing deficiencies in performance.

We have adopted an intermediate strategy in implementing our prototype temporal DBMS. This prototype will be used as a comparison point for the fully integrated DBMS we are now developing. We started with the static DBMS *INGRES* [Held et al. 1975], making modifications only when they promised an immediate improvement in efficiency.

One of the most important decisions was determining a means of embedding a four-dimensional temporal relation in a two-dimensional static relation as supported by INGRES. There are at least five ways to embed a temporal relation in a static relation [Snodgrass 1985]. Our prototype adopted the scheme of attaching one or two valid time attributes (depending on whether the relation modeled events or intervals), and two transaction time attributes to each tuple. Each update operation on an existing tuple generates a new version of the tuple, marked with appropriate values of the time attributes indicating the period while the version is active. Though this tuple versioning scheme appears to have a very high degree of redundancy, in that the entire tuple is duplicated, an analysis reveals that this scheme consumes less space than attribute versioning up to a certain degree of volatility [Ahn 1985].

The prototype involved minimal additions to INGRES. The parser 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 *start of, end of, precede, overlap, extend,* and *as of* were added in the interpreter.

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

All three kinds of time are supported, as are all four types of relations. The system relation was modified to support the various combination of implicit temporal attributes according to the type of a relation as specified by its *create* statement. A temporal attribute is represented as a 32 bit integer with a resolution of one second. It is a distinct type, so that input and output 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 prototype was constructed in about 2 person-months over a period of a year; this figure does not include familiarization with the INGRES internals or with TQuel. Most of changes were additions, increasing the source by 2,700 lines, or 4.7 % (our version of INGRES is approximately 56,400 lines long). Measurements of the time and space overhead in the prototype are currently underway, and new access methods for temporal relations are being investigated.

5. Conclusion

In this paper, we described a taxonomy of time consisting of three distinct concepts for use in databases: transaction time, valid time, and user-defined time. Using the taxonomy, we defined four types of databases differentiated by their ability to support these time concepts: static, rollback, historical and temporal databases. An analogy was presented to clarify the subtle differences among the four types of databases, and a series of example transactions highlighted their similarities and differences. We also described a prototype implementation of the temporal DBMS for TQuel.

While fifteen years of research has focused on formalizing and implementing static databases, only a few researchers have recently studied the formalization of historical databases (e.g., [Clifford & Warren 1983]) and the implementation of rollback databases (e.g., [Lum et al. 1984]). To the authors' knowledge, there has been nothing published on formalizing rollback or temporal databases, nor implementing historical or temporal databases. The special opportunities promised by temporal databases are, at this time, matched by the challenges in supporting them.

6. Bibliography

[Ahn 1985] Ahn, I. *Towards an Implementation of Database Management Systems with Temporal Support.* 1985. (Submitted for publication.)

[Ammon et al. 1985] Ammon, G., J. Calabria and D. Thomas. *A High-Speed, Large-Capacity, Jukebox Optical Disk System. IEEE Computer,* 18, No. 7, July 1985, pp. 36-46.

[Ariav 1984] Ariav, G. *Preserving The Time Dimension In Information Systems.* PhD. Diss. The Wharton School, University of Pennsylvania, Apr. 1984.

[Bolour et al. 1982] Bolour, A., T.L. Anderson, L.J. Debeyser and H.K.T. Wong. *The Role of Time in Information Processing: A Survey. SigArt Newsletter,* 80, Apr. 1982, pp. 28-48.

[Bubenko 1976] Bubenko, J. A., Jr. *The temporal dimension in information modeling.* Technical Report RC 6187 #26479. IBM Thomal J. Watson Research Center. Nov. 1976.

[Clifford & Warren 1983] Clifford, J. and D. S. Warren. *Formal Semantics for Time in Databases. ACM Transactions on Database Systems,* 8, No. 2, June 1983, pp. 214-254.

[Copeland 1982] Copeland, G. *What If Mass Storage Were Free?. Computer,* 15, No. 7, July 1982, pp. 27-35.

[Held et al. 1975] Held, G.D., M. Stonebraker and E. Wong. *INGRES--A relational data base management system. Proceedings of the 1975 National Computer Conference,* 44 (1975) pp. 409-416.

[Hoagland 1985] Hoagland, A. *Information Storage Technology : A Look at the Future. IEEE Computer,* 18, No. 7, July 1985, pp. 60-67.

[Jones & Mason 1980] Jones, S. and P. J. Mason. *Handling the Time Dimension in a Data Base.* In *Proceedings of the International Conference on Data Bases,* Ed. S. M. Deen and P. Hammersley. British Computer Society. University of Aberdeen: Heyden, July 1980 pp. 65-83.

[Katz & Lehman 1984] Katz, R. and T. Lehman. *Database Support for Versions and Alternatives of Large Design Files. IEEE Transactions on Software Engineering,* SE-10, No. 2, Mar. 1984, pp. 191-200.

[Lum et al. 1984] Lum, V., P. Dadam, R. Erbe, J. Guenauer, P. Pistor, G. Walch, H. Werner and J. Woodfill. *Designing DBMS Support for the Temporal Dimension.* In *Proceedings of the Sigmod '84 Conference,* June 1984 pp. 115-130.

[Schueler 1977] Schueler, B. *Update Reconsidered.* In *Architecture and Models in Data Base Management Systems.* Ed. G. M. NijssenNorth Holland Publishing Co., 1977.

[Snodgrass 1984] Snodgrass, R. *The Temporal Query Language TQuel.* In *Proceedings of the Third ACM SIGAct-SIGMOD Symposium on Principles of Database Systems,* Waterloo, Ontario, Canada: Apr. 1984 pp. 204-212.

[Snodgrass 1985] Snodgrass, R. *A Temporal Query Language.* Technical Report 85-013. Computer Science Department, University of North Carolina at Chapel Hill. May 1985.

[Snodgrass & Ahn 1985] Snodgrass, R. and I. Ahn. *A Taxonomy of Time in Databases.* In *Proceedings of the International Conference on Management of Data,* ACM SIGMod. Austin, TX: May 1985.

[Wiederhold 1984] Wiederhold, G. ***Databases. IEEE Computer,*** 17, No. 10, Oct. 1984, pp. 211-223.