TR85-013

A Temporal Query Language

Richard Snodgrass

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

May, 1985

Abstract

Recently, attention has been focussed on *temporal databases*, representing an enterprise over time. We have developed a new language, *TQuel*, to query a temporal database. TQuel was designed to be a miminal extension to Quel, both syntactically and semantically, of Quel, the query language in the Ingres relational database management system. This paper discusses the language informally, then provides a tuple relational calculus semantics for the TQuel statements that differ from their Quel counterparts, including the modification statements. The three additional temporal constructs defined in TQuel are shown to be direct semantic analogues of Quel's where clause and target list. We also discuss reducibility of the semantics to Quel's semantics when applied to a static database.

A preliminary version of the material in the first third of this paper appeared in the *Proceedings of the Third ACM SIGAct-SIGMod Symposium on Principles of Database Systems* [Snodgrass 1984] and in the *Proceedings of the ACM SIGMod Conference on the Management of Data* [Snodgrass & Ahn 1985]. This paper is a major revision of technical report TR 84-006, July, 1984. This work was supported in part by National Science Foundation grant DCR-8402339 and by an IBM Faculty Development award.

Table of Contents

List of Figures

List of Examples

1. Introduction

Most 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continue to change as new information is added, these changes are viewed as modifications to the state, with the old, out-of-date data being deleted from the database.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.

Recently, attention has been focussed on *temporal databases*, representing the progression of states of an enterprise over an interval of time. In such databases, changes are viewed as additions to the information in the database. Temporal databases (TDBs) are thus generalizations of conventional (termed *static*) databases.

We have developed a new language, ***TQuel*** (*T*emporal *QUE*ry *L*anguage), to query a TDB. TQuel is a derivative of Quel [Held et al. 1975], the query language for the Ingres relational database management system [Stonebraker et al. 1976]. TQuel was designed to be a minimal extension, both syntactically and semantically, of that language. This design decision has three important ramifications: all legal Quel statements are also valid TQuel statements, such statements have an identical semantics in Quel and TQuel when the time domain is fixed, and the additional constructs defined in TQuel to handle time have direct analogues in Quel. TQuel is, then, a natural extension of a static relational query language to a temporal relational query language.

Major portions of the language have been formalized and implemented. This paper will focus on the syntax and semantics of TQuel; the implementation will be described in a later paper. This paper first introduces the concept of temporal databases in Section 2, and provides an overview of the language in the third section. A formal definition and semantics of TQuel is the subject of the next two sections. The final section summarizes the results, compares TQuel to other query languages, and indicates future work. An appendix gives the complete syntax of the augmented TQuel statements.

2. Temporal Databases

Temporal information has been stored in computerized information systems for many years; payroll and accounting systems are typical examples. In these systems, the attributes involving time are manipulated solely by the application programs; the DBMS interprets dates as values in the base data types. For example, the ENFORM database management system encodes dates and times in character arrays [Tandem 1983]; the Query-by-Example system supports both date and time domain types directly [Bontempo 1983]; and Ingres has been extended to convert dates to and from an internal format and to perform comparisons and arithmetic operations on these domains [Overmyer & Stonebraker 1982, Relational Technology 1984]. However, none of these systems interpret temporal domains when deriving new relations.

The need to handle time more comprehensively surfaced in the early 1970's in the area of medical information systems, where a patient's medical history is particularly important. The model supported by the Time Oriented Databank [Wiederhold et al. 1975] and several other medical DBMSs (e.g., CLINFO [Palley et al. 1976]) views the database as a set of entity-attribute-value-time quadruples, where the time portion indicates when the information represented by the tuple became valid. In these systems, the query language is used to select subsets of quadruples from the three dimensional database of entities (i.e., patients), attributes, and times.

In the last five years, interest in the area of TDBs has increased. In a recent, extensive bibliography [Bolour et al. 1982], containing 69 articles from the period 1960 to June, 1982, over half of the referenced articles were published since 1978. This activity may be classified loosely into three emphases: the formulation of a semantics of time at the conceptual level, the development of a model for TDBs analogous to the relational model for static databases, and the design of temporal query languages. However, the problems inherent in the modeling of time are not unique to information processing; a significant literature exists on related issues in artificial intelligence (c.f., [Allen 1983, Allen 1981, Allen 1984, Cheeseman 1983, Fagan 1980, Findler & Chen 1971, Kahn & Gorry 1975, Long & Russ 1983, McDermott 1982, Tsotsos 1981, Vilain 1982]), linguistics (c.f., [Dowty 1972, Hirschman & Story 1981, McCawley 1971, Montague

1973]), logic (c.f., [McArthur 1976, Prior 1967, Rescher & Urquhart 1971]), philosophy (c.f., [Whitrow 1980]), and physics (c.f., [Taylor & Wheeler 1966]).

Bubenko [Bubenko 1976, Bubenko 1977], specified a TDB and examined two possible implementation strategies, in the binary and n-ary relational models. Since the appearance of these papers, various semantic models have been proposed that incorporate the temporal dimension to varying degrees [Anderson 1981, Anderson 1982, Breutmann et al. 1979, Bubenko 1980, Codd 1979, Hammer & McLeod 1981, Klopprogge 1981].

At least two possible approaches to the development of a model for TDBs have been suggested. One is to extend the semantics of the relational model to incorporate time directly. The other is to base TDBs on the static model, with time appearing as additional attribute(s). The first has been applied successfully by Clifford and Warren [Clifford & Warren 1983], with the entity-relationship model used to formulate the intensional logic $\mathbb{IL}_s$. This logic serves as a formalism for the temporal semantics of a TDB much as the first-order logic serves as a formalism for the static relational model. Sernadas has taken the same approach in defining the temporal process specification language DMTLT, which incorporates a special modal tense logic [Sernadas 1980].

In the second approach, the static relational database model [Codd 1970] serves as the underlying model of the TDB. Each temporal relation is *embedded* in a static relation containing additional temporal attribute(s). In this approach, the logic of the model does not incorporate time at all; instead, the query language must translate queries and updates involving time into retrievals and modifications on the underlying static relations. In particular, the query language must provide the appropriate values for these attributes in the relation being derived. In Ben-Zvi's Time Relational Model, for example, five additional attributes are appended to each relation [Ben-Zvi 1982]. Other researchers have also utilized this technique [Ariav & Morgan 1981, Copeland & Maier 1984, Gadia & Vaishnav 1985, Jones & Mason 1980].

Several query languages incorporating time have been designed over the last decade. In Section 7.2, TQuel is compared with these other proposals.

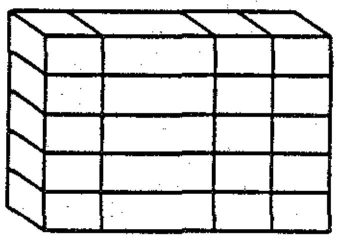
Most databases incorporating time support only one aspect of time—the time when the information is valid. This aspect is termed *valid time*. Two other aspects of time should be supported by a temporal query language: *transaction time* and *user-defined time*. The remainder of this section will characterize these aspects briefly; a more complete discussion may be found in [Snodgrass & Ahn 1985]. The presentation is more of an intuitive nature than a formal characterization of temporal databases; Sectin 5 will show how to embed a temporal relation in a static relation, thereby providing a precise definition. We take the second approach to modelling TDBs: utilizing the static model.

2.1. Static Databases

Conventional databases model the dynamic real world, as 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, since changes to the database will always lag behind changes in the real world. Updating the state of a database is performed 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, and those of the real world, are discarded and forgotten completely. We term this type of database a *static 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 2-dimensional table (see Figure 1). As changes occur in the real world, changes are made in this table.

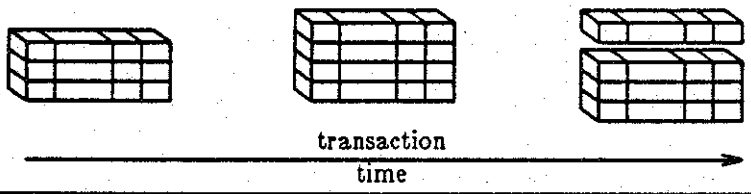
Figure 1: A Static Relation



2.2. Static Rollback Databases

Static databases relying on snapshots are inadequate for many situations. For example, they cannot answer queries on past states. Without system support in this respect, many applications were forced to maintain and handle temporal information in an ad-hoc manner. One approach to resolve these 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 indexed by transaction time. One can get a snapshot of the relation as of some time in the past (a static relation) and make queries upon the static relation by moving along the time axis and selecting this relation. The operation of selecting a static relation is termed ***rollback***, and a database supporting it is termed a ***static rollback database***. A rollback to a time t, where t is between two transaction times t_1 and t_2 represented in a static rollback database, selects the most recent static relation in effect at that time (i.e., the one at t_1). Changes to a static rollback database may only be made to the most recent static state. The (single) 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 a tuple, and (3) the deletion of one tuple (which was entered in the first transaction) and the addition of another tuple. Each transaction results in a new static relation being appended to the right; once a transaction has completed, the static relations in the static rollback relation may not be altered.

Figure 2: A Static Rollback Relation



2.3. Historical Databases

One limitation of supporting transaction time is that the history of database activities is recorded, rather than the history of the real world. A tuple becomes valid as soon as it is entered into the database as in a static database. Retroactive/postactive changes are not recorded and errors in past tuples cannot be corrected. Errors can sometimes be overridden (if they are in the current state) but they cannot be forgotten.

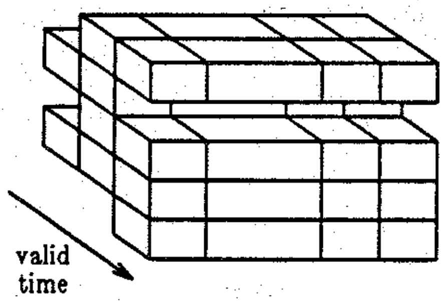
While static rollback databases record a sequence of static states, *historical databases* record a single *historical state* per relation, storing the history as is best known. As errors are discovered, they are corrected by modifying the database. Previous states are *not* retained, so the database may not be viewed as it was in the past. No is record kept of the errors that have been corrected; historical databases are similar to static databases in this respect. Thus historical databases must represent *valid time*, the time that the stored information models reality. Historical databases support *historical queries*, which may utilize information from the past.

Historical databases may also be illustrated in three dimensions (see Figure 3) [Ariav 1984, Ben-Zvi 1982, Clifford & Warren 1983, Lum et al. 1984]. Although the illustration as a series of static relations indexed by a time parameter is similar in some aspects to that for static rollback databases, the label of the time axis has been changed to valid time and the semantics are more closely related to reality, rather than update history. The state of the world being modelled remains unchanged between the individual static relations found in the historical relation; this is termed the *step function continuity assumption* [Clifford & Warren 1983] or the *principle of temporal density* [Ariav 1984]. The information present in the static database slice at one valid time v_1 is assumed to be valid for all times between that valid time and the next one, v_2. Hence, the tuples in the relation are valid for the interval of time $[v_1, v_2)$.

As the model now stands, only states that exist for a finite interval of time may be represented. One representation of events is tuples that exist for exactly one valid time, with the static relations of the previous and next valid times not containing the tuples. This representation is problematic because time is continuous: it is misleading to talk about *the* previous and next time values. Of course, any implementation will encode valid time in some discrete fashion; the proposed representation for events then reduces to an interval of the granularity of the valid time encoding (say, seconds, or microseconds). Static relations, in modelling current reality, cannot represent events at all, precisely because they are instantaneous.

Since an update to an historical relation must specify the valid time it concerns, more sophisticated operations are necessary to manipulate and query valid time adequately, compared to the simple rollback operation, since they apply to the entire historical relation, rather than a single static slice.

Figure 3: An Historical Relation

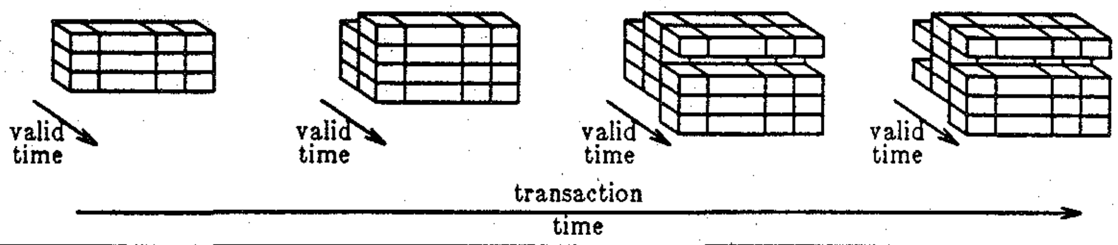


2.4. Temporal Databases

Benefits of both approaches can be combined by supporting both transaction time and valid time. While a static rollback database views tuples as being valid at some time as of that time, and a historical database always views tuples as being valid at some moment as of *now,* a temporal DBMS makes it possible to view tuples as being valid at some moment relative to some other moment, completely capturing the history of retroactive/postactive changes.

We use the term ***temporal database*** to emphasize the need for both valid time and transaction time in handling temporal information. Since two time axes are now involved, four dimensions are required to represent a temporal relation (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 creates a new historical state; hence, temporal relations are append-only. However, the transaction must specify the valid time(s) it concerns, as in an historical database. 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 (with an earlier valid time) was deleted (presumably it should not have been there in the first place). Each update operation involves copying the historical relation, then applying the update to the newly created historical relation.

Figure 4: A Temporal Relation



User-defined time [Jones & Mason 1980] is necessary when additional temporal information, not handled by transaction or valid time, is stored in the database. As an example, consider the Promotion relation, with the three attributes name, rank, and effective_date. The effective date is a user-defined temporal attribute, and values of this attribute would appear in one of the columns of Figures 1-4. This date is the date that the promotion was to take effect, as shown on the promotion letter; the valid time is the moment the promotion letter was signed, i.e., the date the promotion was validated; and the transaction time is the moment the information concerning the promotion was stored in the database. The effective date is application-specific: it is merely a date which appears on the promotion letter. 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. The transaction and valid times are needed in any case in temporal relations.

In this model, four types of databases were defined: static, static rollback, historical, and temporal. Each may be associated with a class of query languages. A *static query language* supports queries over multiple static relations. A ***static rollback query language*** also supports rollback. An ***historical query language*** does not support rollback, but it does support historical queries, which combine information from multiple valid times and possibly multiple relations. A ***temporal query language*** supports both rollback and historical queries. The next section will informally introduce the temporal query language TQuel.

3. Overview of TQuel

TQuel is a superset of Quel [Held et al. 1975], the query language for Ingres [Stonebraker et al. 1976]. Quel was chosen for several reasons: it is well known and implementations are widely available; it is particularly simple but rather powerful; and it has a simple and well defined semantics. The leading contender, SQL [SQL/DS 1981], is more complex and has a rather complicated semantics [Ceri & Gottlob 1985, Kim 1982]. An important goal in the design of TQuel was that it be a minimal extension, both syntactically and semantically, of Quel. This objective has three important ramifications: all legal Quel statements are also valid TQuel statements, such statements have an identical semantics in Quel and TQuel when the time domain is fixed, and the additional constructs defined in TQuel to handle time have direct analogues in Quel.

TQuel will be illustrated using example queries on the database shown in Figure 5. The Faculty relation lists the faculty members and their ranks (one of the values Assistant, Associate, or Full); the Submitted relation lists those papers submitted. In the discussion that follows, the reader is assumed to be familiar with Quel.

Figure 5: A Static Database

Faculty (Name, Rank):

Name	Rank
Jane	Full
Merrie	Associate
Tom	Associate

Submitted (Author, Journal):

Author	Journal
Jane	CACM
Merrie	CACM
Merrie	TODS
Tom	JACM

Figure 6: Result of a Query on a Static Database

Associates (Name):

Name
Merrie
Tom

The Quel retrieve statement consists of two basic components, the *target list*, specifying how the attributes of the relation being derived are computed from the attributes of the underlying relations, and a *where clause*, specifying which tuples participate in the derivation. The query

```
range of f is Faculty
retrieve into Associates (Name = f.Name)
      where f.Rank = "Associate"
```

Example 1: List the associate professors.

produces in the relation shown in Figure 6 when applied to the sample database. The **range** statement associates tuple variables with relations; this binding remains in effect until a new range statement with the same tuple variable is executed.

The relations shown in Figures 5 and 6 are static relations. While the graphical representation of a temporal relation as a sequence of three-dimensional structures is conceptually elegant, it is not convenient for displaying the contents of a temporal relation. For the purposes of this section, the temporal relations will be embedded in a static relation by appending two additional temporal attributes. The value of the first attribute specifies the valid time: when that tuple was valid. For *event relations*, which consist of tuples representing instantaneous occurrences, this attribute contains a single time value (*at*). For *interval relations*, which consist of tuples representing a state valid over a time interval, the attribute contains two time values delimiting the interval (*from, to*). The second temporal attribute specifies the transaction time: when the information was entered into the TDB. Two time values are always associated with the transaction time: the time the tuple was entered into the TDB (*start*), and the time it was removed (*stop*). Hence data is current from the *start* time to just before the *stop* time, when it becomes no longer current. Figure 7 illustrates the Faculty relation extended to become an interval relation, and the Submitted relation extended to become an event relation. Note that Tom was entered into the database as an associate professor in August, 1975; this error was corrected two months later. No errors have been corrected in the Submitted relation, since the *stop* time for all tuples is "∞". Both intervals, for valid and transaction time, are closed on the left and open on the right. The granularity of time values is arbitrary; in this section we assume for simplicity a granularity of one month.

Figure 7: A Temporal Database

Faculty (Name, Rank):

Name	Rank	Valid Time (From)	Valid Time (To)	Transaction Time (Start)	Transaction Time (Stop)
Jane	Assistant	9-71	12-76	9-71	∞
Jane	Associate	12-76	11-80	12-76	∞
Jane	Full	11-80	∞	10-80	∞
Merrie	Assistant	9-77	12-82	8-77	∞
Merrie	Associate	12-82	∞	12-82	∞
Tom	Associate	9-75	∞	8-75	10-75
Tom	Assistant	9-75	12-80	10-75	∞
Tom	Associate	12-80	∞	11-80	∞

Submitted (Author, Journal):

Author	Journal	Valid Time (At)	Transaction Time (Start)	Transaction Time (Stop)
Jane	CACM	11-79	11-79	∞
Merrie	CACM	9-78	9-78	∞
Merrie	TODS	5-79	5-79	∞
Tom	JACM	12-82	12-82	∞

Since TQuel is a strict superset of Quel, the identical query, executed in September, 1985, on this sample TDB, produces the relation shown in Figure 8. The transaction time specifies when the relation was created; subsequent updates will alter the transaction time of individual tuples.

Figure 8: The Same Query on a Temporal Database

Associates (Name):

Name	Valid Time (From)	Valid Time (To)	Transaction Time (Start)	Transaction Time (Stop)
Jane	12-76	11-80	9-85	∞
Merrie	12-82	∞	9-85	∞
Tom	12-80	∞	9-85	∞

Since the additional temporal attributes are an artifact of embedding a temporal relation in a static one, users must be constrained in how which they use these attributes. The query language must be designed so that temporal attributes are used correctly. The approach taken here is to make the temporal attributes implicit in the query language (except in one very restricted case), and to provide facilities in the language for manipulating this implicit attribute. That these additional attributes are implicit is indicated in the figures by a double vertical line and parentheses surrounding the names of the attributes. To manipulate these attributes, TQuel augments the retrieve statement with three components, analogous to the components of the Quel retrieve statement, one specifying how the implicit valid temporal attribute is computed, and two specifying the temporal relationship of the tuples participating in the derivation.

3.1. The When Clause

The *when clause* is the temporal analogue to Quel's where clause. This clause consists of the keyword followed by a *temporal predicate* on the tuple variables, representing the implicit time attributes of the associated relations. The syntax is similar to *path expressions*, which are regular expressions augmented with parallel operators [Andler 1979, Habermann 1975].

The **overlap** operator specifies that the events and/or intervals overlap in time:

```
range of a is Associates
retrieve into FirstDayAssociates (Name = a.Name)
      when a overlap "September"
```

Example 2: List the associate professors in September.

In this case, the query specifies that the interval when the faculty member was an associate professor should include September, which is also a time interval (strings, enclosed in double quotation marks, are temporal constants). As another example,

```
range of a is Associates
range of s is Submitted
retrieve into  AssocPapers (Name = s.Author, Journal = s.Journal)
      where a.Name = s.Author
      when s overlap a
```

Example 3: What papers were written by associate professors?

The time that the paper was submitted must overlap with the time interval when the faculty member was an associate professor.

Intervals include two time values in the implicit attribute; a starting time and a stopping time. These values may be indicated by the unary operators **start of** and **end of**:

```
range of f1 is Faculty
range of a is Associates
retrieve into Full (Name = f1.Name)
        where a.Name = Tom and f1.Rank = "Full"
        when f1 overlap start of a
```

Example 4: Who were the full professors when Tom was promoted to associate?

Sequentiality may be tested with the **precede** operator:

```
range of a is Associates
retrieve into Disgruntled (Name = a.Name)
        when (start of a) precede "January, 1980" and "January, 1985" precede (end of a)
```

Example 5: Who has been an associate professor for the last five years?

This example also illustrates the **and** operator; the **or** and **not** operators are allowed as well.

Given the **precede** operator, the **extend** operator may be introduced. This operator is similar to the **overlap** operator. The **overlap** operator may be thought of as a temporal *intersection* operator, in that it returns true when *both* arguments are true: the predicate

(a **overlap** b) **precede** c

is true when the overlap of the intervals represented by the tuple variables a and b precedes the event or the start of the interval represented by c. However, the **extend** operator is more like a temporal *union*, in that it returns true when *either* of the arguments are true; the predicate

(a **extend** b) **precede** c

is true when the end of both a and b precede the start of c. The difference between **overlap** and **extend** is illustrated with the time lines in Figure 9.

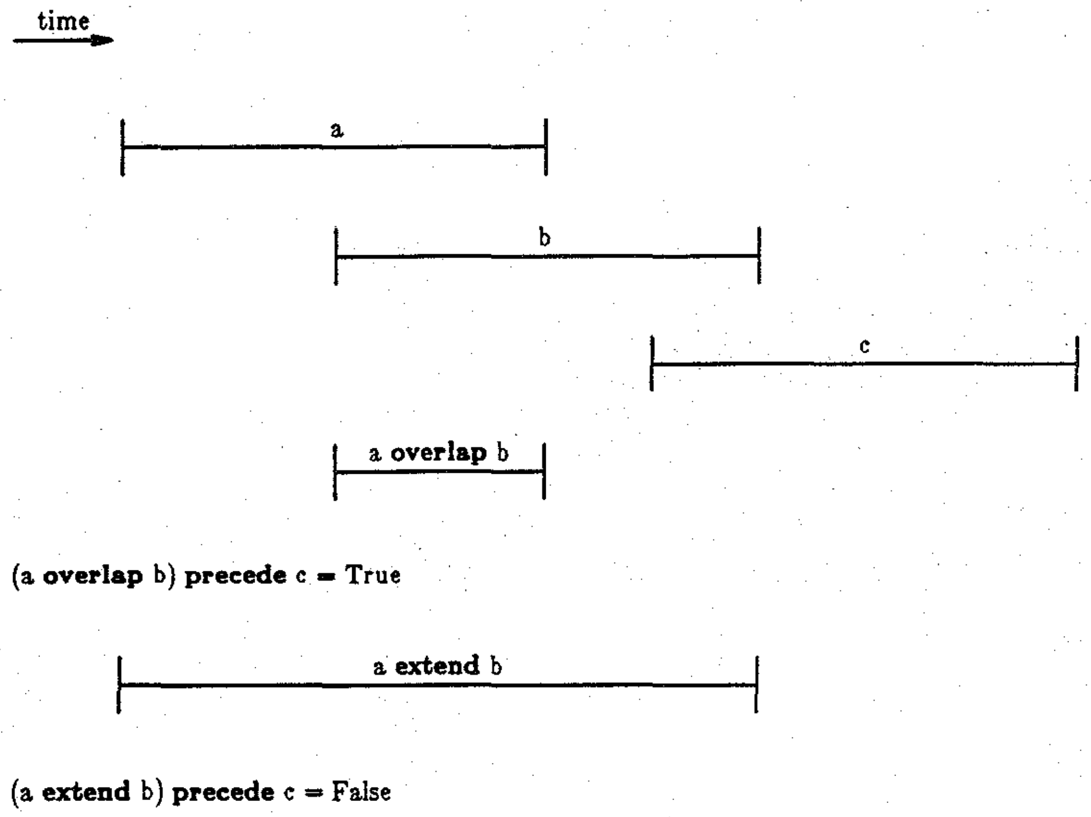

time
a
b
c
a **overlap** b
(a **overlap** b) **precede** c = True
a **extend** b
(a **extend** b) **precede** c = False

Figure 9: The Difference between **overlap** and **extend**

3.2. The Valid Clause

The *valid clause* serves the same purpose as the target list: specifying the value of a attribute in the derived relation. In this case, the attribute in question is the implicit time attribute. There are two variants to this clause. If the derived relation is to be an event relation, the **valid at** variant specifies the value of the single time in the temporal attribute.

> **range of** a **is** Associates
> **retrieve into** AssociatePromotions (Name = a.Name)
>   **valid at start of a**

Example 6: When were the associate professors promoted to this rank?

In this query, the underlying relation, Associates, is an interval relation. One time value, the start time, was selected as the time value in the derived (event) relation. The valid clause contains an *e-expression*, also syntactically similar to path expressions. The operators **start of**, **end of**, **overlap**, **extend**, and

precede may be used in E-expressions. The binary boolean operators **and** and **or** and the unary boolean operator **not** are *not* allowed, since they introduce ambiguity as to which time value is desired.

The second variant of the valid clause, also containing e-expressions, is used when the derived relation is to be an interval relation:

```
range of f1 is faculty
range of f2 is faculty
range of a is Associates
retrieve into Stars (Name = f1.Name)
        valid from start of f1 to start of f2
        where f1.Name = f2.Name and f1.Rank = "Assistant" and f2.Rank = "Full"
        when (f1 overlap a) and (f2 overlap a)
```

Example 7: Who got promoted from assistant to full professor while at least one other faculty remained at the ass

Tuples in the derived relation Stars indicate the interval of time from joining the faculty as assistant professors to becoming full professors.

The operators found in temporal predicates and e-expressions may be applied more generally than shown above; as an example, the e-expression

```
valid at start of (A overlap B)
```

specifies that the time value returned should be the first instant when both tuples are valid. E-expressions must have **start of** or **end of** as top level operators.

As with other languages, there are several ways to write most queries. As an example, the **and** operator in the when clause can considerably simplify matters:

```
range of f1 is Faculty
range of f2 is Faculty
range of a is Associates
retrieve into Stars (Name = f1.Name)
        valid from start of f1 to start of f2
        where f1.Name = f2.Name and f1.Rank = "Assistant" and f2.Rank = "Full"
        when (f1 and f2) overlap a
```

Example 8: A Variant of Example 7.

3.3. The As of Clause

The when and valid clauses are used to express historical queries. To express rollback, the **as of** clause is used:

```
range of f1 is Faculty
range of f2 is Faculty
range of a is Associates
retrieve into Starsof1984 (Name = f1.Name)
        valid from start of f1 to start of f2
        where f1.Name = f2.Name and f1.Rank = "Assistant" and f2.Rank = "Full"
        when (f1 and f2) overlap a
        as of end of "1984"
```

Example 9: What stars were known at the end of 1984?

The as-of clause ***rolls back*** the database to the state it was at midnight on December 31, 1984, and evaluates the rest of the query using the information known only to that point. Additions and error corrections made after that time would not be included in the resulting relation.

The as-of clause is similar to the where and when clauses, in that it provides an additional constraint on the underlying tuples participating in the query. Most of the time the user will be interested in the most up-to-date information in the database, and will rely on the default for the as-of clause: **as of** "now". To rollback to a previous historical database, the as-of clause as illustrated above would be used. To examine a sequence of transactions occurring over a period of time, a third variant is used:

as of α **through** β

3.4. Temporal Data Type

TQuel provides a temporal data type to support user defined time. As discussed previously, the values of user-defined temporal attributes are not interpreted by the TDBMS; only the internal representation, the input and output functions, and the comparison operators are provided.

3.5. Modification Statements

Quel has three modification statements: append, delete, and replace. These statements in TQuel do not have an as-of clause, because the transaction time is computed automatically by the TDBMS as the current time (recall that TDB's are append-only). However, the valid and when clauses may be employed in these statements. In October, 1985, it was learned that Tom had submitted a paper not to *JACM*, but to *TOPLAS*, a month later than previously thought.

```
range of s is Submitted
replace s (Journal = "TOPLAS")
        where s.Author = "Tom" and s.Journal = "JACM"
        valid at start of "January, 1983"
```

Example 10: Tom submitted a paper to *TOPLAS*, not to *JACM*,

resulting in the relation shown in Figure 10, which should be compared with Figure 7.

Figure 10: An Updated Temporal Relation

Submitted (Author, Journal):

Author	Journal	Valid Time	Transaction Time	
		(At)	(Start)	(Stop)
Jane	CACM	11-79	11-79	∞
Merrie	CACM	9-78	9-78	∞
Merrie	TODS	5-79	5-79	∞
Tom	JACM	12-82	12-82	10-85
Tom	TOPLAS	1-83	10-85	∞

4. Formal Definition

The description of TQuel in the previous section was presented informally to help the reader develop an intuitive understanding of the language. This section and the next will provide a more precise definition and semantics for the language.

Quel has some fourteen statements; TQuel augments five of them: the create, retrieve, append, delete, and replace statements. The statements will be discussed in this order. The syntax for the retrieve statement will be presented in a bottom up fashion, discussing expressions before clauses, in contrast to the top down presentation of the previous section, where the clauses were emphasized. The appendix includes the syntax of the five statements.

4.1. Schema Definition

The create statement defines a new relation and provides a scheme for that relation; the statement

create persistent interval Faculty (Name = c20, Rank = c10)

would define the Faculty relation shown in Figure 7 (the *contents* of this relation would have to be provided through the copy or append statements). The Quel create statement does not include the **persistent, interval,** or **event** keywords. Each of these keywords is optional in TQuel (see the appendix for details on the syntax). If the **persistent** keyword is used, then the relation is either a static rollback or a temporal relation, and the as-of clause may be used in queries. If the **interval** or **event** keyword is used, the relation is either an historical or temporal relation, and the when and valid clauses may be used. If none of these keywords are used, the relation is a conventional static relation. The four types of relations (static, static rollback, historical, temporal) are thereby specified. The domain specifications are similar to those in Quel (integers, floating point numbers, and fixed length character strings, as used above, are supported), with the addition of a temporal data type.

Associated with all static rollback and temporal relations is a pair of transaction time values, *start* and *stop*. Although these values are closely associated with clock time, they are actually transaction identifiers. Tuples created or removed by two different transactions will have different transaction times, even if the transactions started and completed at identical moments in time.

Associated with all historical and temporal event relations is a single valid time value, *at*, and with all historical and temporal interval relations, a pair of valid time values, *from* and *to*. These values are equal to the clock time when the tuple was valid. In contrast to transaction time, two tuples entered into the database at different times may have the same valid times.

4.2. Constants and Predefined Domains

Quel supports numeric and character string constants. TQuel augments these with temporal constants. Strings appearing in the valid, when, and as-of clauses are interpreted as temporal constants denoting a particular time interval. The string "Sept. 1, 1983" denotes an interval from midnight of

9/1/83 to midnight of 9/2/83; "Sept, 1983" denotes the entire month; and "4:00pm September 1, 1983" denotes a sixty second interval. Events may be approximated with very short intervals. The constants "now" and "infinity" are also available. The exact format of these constants is similar to that specified for the time expert [Overmyer & Stonebraker 1982] or the Ingres system [Relational Technology 1984].

The implicit temporal attributes are available as the predeclared attribute names "validat", "validfrom" and "validto" (valid time), and "transactionstart" and "transactionstop" (transaction time), for use only in the target list and where clauses. These special attributes, as well as the temporal data type, are provided in part for auditing purposes [Bjork 1975]; a simple example is

```
range of f is Faculty
retrieve into Mistake (MistakeDate = f.TransactionStart, CorrectedDate = f.TransactionStop)
        where f.Name = "Tom" and f.Rank = "Associate"
        as of "1975"
```

Example 11: When was Tom entered incorrectly as an Associate Professor?

The MistakeDate and CorrectedDate attributes cannot be used in subsequent when, valid, or as-of clauses; to the TDBMS these attributes are just other user-defined attributes. Perhaps the temporal data type's most useful function is to be displayed with the other user-defined attributes (as in the example above). TRM also provides restricted access to the time attributes [Ben-Zvi 1982].

As the other statements, retrieve, append, delete, and replace, all incorporate the when, valid, and as-of clauses, we will first discuss the expressions found in these clauses.

4.3. Temporal Expressions

A *temporal constructor* is a unary or binary operator that takes one or two events or intervals as arguments and returns an event or interval. If either of the arguments to the temporal constructors is an event, then it is coerced into an interval which starts and ends at the event's time value. The unary prefix temporal constructors are **start of** and **end of**, both returning events. The binary infix temporal constructs are **overlap** and **extend**, both returning intervals. **overlap** is undefined if there are no time values which are in both underlying intervals. Figure 9 illustrates the difference between **overlap** and **extend**.

An *e-expression* is simply an expression containing tuple variables, temporal constants, and temporal constructors, with the constraint that the expression must result in an event. E-expressions are used in the valid and as-of clauses. Since the as-of clause specifies rollback to a particular transaction time, the e-expression in an as-of clause must evaluate to a temporal constant. An equivalent constraint is that an e-expression within an as-of clause must not contain a tuple variable.

A *temporal predicate operator* is a binary infix operator that takes events or intervals as arguments and returns a boolean value. The two temporal predicate operators are **precede** and **overlap**. The reader will notice the semantic overloading of the **overlap** operator. This overloading also occurs in English: one may ask whether two intervals *overlap*, or may ask for the *overlap* of two intervals, expecting a yes or no to the first query and an interval for the second request. α **precede** β is true if the event (**end of** α) is before the event (**start of** β). One event is *before* a second event if the time value of the first, expressed as an integer or real value, is less than or equal to ($\leq$) the time value of the second. In this formulation, an event overlaps itself. α **overlap** β is true if the event (**start of** α) is before the event (**end of** β) and the event (**start of** β) is before the event (**end of** α). An equivalent formulation is

(**end of** (**start of** α **extend start of** β)) **precede** (**start of** (**end of** α **extend end of** β))

A *temporal predicate* is an expression containing logical operators (**and**, **or**, **not**) operating on expressions containing a temporal predicate operator (**precede**, **overlap**), operating on e-expressions. As **and** and **or** distribute over all temporal constructors and temporal predicate operators, it is only necessary that the expressions obtained by distributing **and** and **or** operators over the temporal operators obey the above constraint (i.e., logical operators on a temporal predicate operator operating on e-expressions). For example, the temporal predicate

((a **and** b) **overlap** c) **precede** d

by distributing **and** over **overlap**, is equivalent to

((a **overlap** c) **and** (b **overlap** c)) **precede** d

which, by distributing **and** over **precede**, is equivalent to

((a **overlap** c) **precede** d) **and** ((b **overlap** c) **precede** d)

This last expression obeys the constraint if the **overlap** operator is interpreted as a temporal constructor, implying that the original expression was a valid temporal predicate. Temporal predicates are used only in when clauses.

4.4. Augmented Quel Statements

The TQuel retrieve statement and the three TQuel modification statements, append, delete, and replace, augment their Quel counterparts with (optional) valid clauses and when clauses; the retrieve statement also allows an optional as-of clause. See the appendix for details on the syntax.

4.5. Defaults

The defaults assumed in the language will be important for the semantics to be presented shortly. Quel defaults the where clause to where true. The defaults for the additional clauses in TQuel should be natural to the user. The retrieve statement will be handled first. If only one tuple variable (say, I) is used, and it is associated with an interval relation, then the defaults are

valid from start of I **to end of** I
when true
as of "now"

These defaults say that the result tuple is to start when the underlying tuple started and stop when the underlying tuple stopped and that the query is to be executed on the current historical state. When an event relation is associated with the one tuple variable (say, E) the default is

valid at E
when true
as of "now"

specifying simply that the result tuple was valid at the same instant the underlying tuple was valid. The first TQuel query given (Example 1) thus has the following default clauses,

```
range of f is Faculty
retrieve into Associates (Name = f.Name)
        valid from start of f to end of f
        where f.Rank = "Associate"
        when true
        as of "now"
```

Example 12: The previous query, with defaults.

When two or more tuple variables are used, the situation is more complex. If the tuple variables associated with interval relations involved in the query are t_1, t_2, ... , t_k, then the default temporal clauses are

```
valid from start of (t_1 overlap ... overlap t_k) to end of (t_1 overlap ... overlap t_k)
when (t_1 overlap ... overlap t_k)
as of "now"
```

These clauses state that the underlying tuples must be *consistent*, that is, they are all valid for the entire interval the resulting tuple is valid.

For the append statement, the defaults are

```
valid from "now" to "infinity"
when (t_1 overlap ··· overlap t_k) overlap "now"
```

Informally, this means that the tuples used to supply values for the new tuples to be appended should be currently valid, and that the new tuples should be considered to have become valid immediately. For the delete statement, the defaults are

```
delete s
valid from start of s to end of s
when true
```

These defaults imply that the tuple be deleted entirely from the current historical relation. And finally, for the replace statement, the defaults are

```
replace s
valid from start of s to end of s
when (t_1 overlap ··· overlap t_k) overlap "now"
```

These defaults follow from the fact that a replace is equivalent to a delete followed by an append.

Note that when only one of these clauses is provided by the user, the other clause is assumed to be as discussed above. The user should be careful in this situation, because the defaulted clause may be inappropriate.

5. Formal Semantics

TQuel statements manipulate information in a TDB composed of a sequence of historical relations indexed by transaction time, with each historical relation consisting of a sequence of static relations indexed by valid time (i.e., the four dimensional structure). The semantics of TQuel must specify how this relation is modified through an update command or is created through a retrieve command. The semantics of TQuel uses the static relational database model as the underlying model of the TDB (Section 2 discussed one alternative: extending the semantics of the relational model to directly incorporate time). Several benefits accrue from using the static relational model. The relational database model is simple and is based on the well-developed formalisms of set theory and predicate calculus; database models directly incorporating time are significantly more complex, and are based on newer and less developed logics such as Montague, multiple transition, and temporal logics. Extensions involving aggregates and indeterminacy are easier to formulate in the standard model (these extensions will be discussed in a later paper). Finally, a temporal database based on the relational model can be implemented directly on conventional relational database management systems. Many of the same advantages resulted from a similar approach in the design of GEM, a query and update language for a (static) semantic data model [Zaniolo 1983] and in the specification of the semantics of the static query language SQL [Ceri & Gottlob 1985].

5.1. Embedding a Temporal Relation in a Static Relation

The static relational database model is utilized as the underlying model of the TDB by embedding the four dimensional temporal relation in a two dimensional static relation. The semantics of operations on four dimensional temporal relations will be specified by stating their effect on the two dimensional static relations. In this way, the semantics can be expressed in a traditional tuple calculus formalism.

This embedding can be accomplished in several ways. The most straightforward is to append two attributes, each containing a single time value, to the user defined attributes, thereby specifying the valid and transaction times for each tuple. Figure 11 shows a portion of the temporal relation in Figure 7 under this representation. In this figure, the tuples comprising an historical relation at a particular transaction time are separated by horizontal lines, and the tuples comprising a static relation at a particular valid time are separated by dots. The static relation in Figure 11 contains a temporal relation comprised of five historical relations (each associated with a unique transaction time), each comprised of static relations (each associated with a unique valid time). The last historical relation, with a transaction time value of 8-77, is comprised of 4 static relations, totaling of 8 tuples. That each transaction creates a copy of the most recent historical relation, mentioned in Section 2.4, can be seen clearly in this representation. The full embedding of Figure 7 would contain eight historical relations, since the temporal relation was the result of eight transactions. The last historical relation would contain seven static relations and a total of thirty tuples. The entire static relation embedding the temporal relation in Figure 7 would contain 102 tuples(!). The historical relations of Clifford and Warren are similar to this embedding [Clifford & Warren 1983]. Their relations are even more verbose, since all keys must be represented in all static slices.

Figure 11: Embedding a Temporal Relation, Version 1

Faculty (Name, Rank):

Name	Rank	Valid Time	Transaction Time
Jane	Assistant	9-71	9-71
Jane	Assistant	9-71	8-75
Jane	Assistant	9-75	8-75
Tom	Associate	9-75	8-75
Jane	Assistant	9-71	10-75
Jane	Assistant	9-75	10-75
Tom	Assistant	9-75	10-75
Jane	Assistant	9-71	12-76
Jane	Assistant	9-75	12-76
Tom	Assistant	9-75	12-76
Jane	Associate	12-76	12-76
Tom	Assistant	12-76	12-76
Jane	Assistant	9-71	8-77
Jane	Assistant	9-75	8-77
Tom	Assistant	9-75	8-77
Jane	Associate	12-76	8-77
Tom	Assistant	12-76	8-77
Jane	Associate	9-77	8-77
Tom	Assistant	9-77	8-77
Merrie	Assistant	9-77	8-77

Another way to embed a temporal relation in a static relation is to append two attributes, each containing *two* time values, denoting *intervals* of valid and transaction time. This is the way temporal relations were illustrated in Figures 7 and 8. Such a representation was proposed by Ariav in his Temporally Oriented Data Management System [Ariav 1984]. Still a third way is to add a total of five additional domains: the time the tuple became valid (T_{es}, the effective-time-start), the time T_{es} was recorded in the database (T_{rs}, the registration-time-start), the time the tuple became invalid (T_{ee}, the effective-time-end), the time T_{ee} was recorded in the database (T_{re}, the registration-time-end), and the time the entire tuple was removed from the database, as it was no longer correct (T_d, the deletion time). Such a representation was proposed by Ben-Zvi in his Time Relational Model [Ben-Zvi 1982]. Figure 12 illustrates the canonical example in this representation. This example contains the same number of tuples as the representation illustrated in Figure 7; generally it will contain somewhat fewer tuples. The effective time in the TRM is equivalent to valid time in our model; the three registration and deletion times encode the same information as our two transaction times.

Figure 12: Embedding a Temporal Relation, Version 3

Faculty (Name, Rank):

Name	Rank	T_{es}	T_{ee}	T_{rs}	T_{re}	T_d
Jane	Assistant	9-71	12-76	9-71	12-76	–
Jane	Associate	12-76	11-80	12-76	10-80	–
Jane	Full	11-80	–	10-80	–	–
Merrie	Assistant	9-77	12-82	8-77	12-82	–
Merrie	Associate	12-82	–	12-82	–	–
Tom	Associate	9-75	–	8-75	–	10-75
Tom	Assistant	9-75	12-80	10-75	11-80	–
Tom	Associate	12-80	–	11-80	–	–

A fourth way to embed a temporal relation in a static relation is to associate time values with the attributes themselves [Gadia 1985, Gadia & Vaishnav 1985]. Within a tuple, the value of an attribute is no longer restricted to be a single value, but may take on different values at different points in time. Figure 13 illustrates the same temporal relation in this representation, without considering the transaction time. In this representation the static relation is no longer in first normal form.

Figure 13: Embedding a Temporal Relation, Version 4

Faculty (Name, Rank):

Name		Rank	
Jane	[9-71, ∞)	Assistant	[9-71, 12-76)
		Associate	[12-76, 11-80)
		Full	[11-80, ∞)
Merrie	[9-77, ∞)	Assistant	[9-77, 12-82)
		Associate	[12-82, ∞)
Tom	[9-75, ∞)	Assistant	[9-75, 12-80)
		Associate	[12-80, ∞)

Finally, the most space efficient representation was proposed by Kimball in the DATA system; only the transactions are recorded [Kimball 1978]. Valid time was not considered but may be added as another domain (see Figure 14). Determining the tuples valid at a particular time as of another time involves replaying the transactions in order from the beginning (optimizations are of course possible). Updates on the other hand are easy to formalize and to implement using this representation.

Figure 14: Embedding a Temporal Relation, Version 5

Faculty (Name, Rank):

Type	Transaction Time	Name	Rank	Valid Time
Add	9-71	Jane	Assistant	9-71
Add	8-75	Tom	Associate	9-75
Modify	10-75	Tom	Assistant	9-75
Modify	12-76	Jane	Associate	12-76
Add	8-77	Merrie	Assistant	9-77
Modify	10-80	Jane	Full	11-80
Modify	11-80	Tom	Associate	12-80
Modify	12-82	Merrie	Associate	12-82

We have chosen the second representation, with each tuple containing four additional time values, upon which to base our semantics. The advantages of this representation include ease of formal manipulation and the promise of rapid prototyping a TDBMS on top of a conventional static DBMS. We emphasize, however, that an equivalent semantics could be generated for the other representations (this is discussed further in Section 6). The semantics of TQuel originates from the model of temporal databases developed in Section 2, not from any particular representational scheme.

Since TQuel is a superset of Quel, its semantics will be based on the semantics for Quel. We first review how Quel's semantics has been specified, then show how this treatment can be applied to TQuel.

5.2. Quel Semantics

Although no complete formal semantics of Quel has been specified, Ullman has defined a tuple relational calculus semantics for Quel statements without aggregates [Ullman 1982], and Klug has treated aggregates in the more general case [Klug 1982]. The tuple calculus semantics for TQuel associates a tuple calculus statement with each TQuel retrieve statement, ensuring that each construct has a clear and unambiguous meaning.

Tuple relational calculus statements are of the form

$$\left\{ t^{(i)} \mid \psi(t) \right\}$$

where the variable t denotes a tuple of arity i, and $\psi(t)$ is a first order predicate calculus expression containing only one free tuple variable t. $\psi(t)$ defines the tuples contained in the relation specified by the Quel statement. The tuple calculus statement for the skeletal Quel statement

range of t_1 **is** R_1
$\ldots$
range of t_k **is** R_k
retrieve $(t_{i_1}.D_1, \ldots, t_{i_r}.D_r)$
  where ψ

is

$$\left\{ u^{(r)} \mid (\exists\, t_1) \cdots (\exists\, t_k)\, (R_1(t_1) \wedge \cdots \wedge R_k(t_k) \right.$$
$$\wedge\, u[1] = t_{i_1}[j_1] \wedge \cdots \wedge u[r] = t_{i_r}[j_r]$$
$$\left. \wedge\, \psi\,') \right\}$$

which states that t_i is in R_i, that the result tuple u is composed of r particular components, that the $m{-}th$ attribute of u is equal to the $j_m{-}th$ attribute (having an attribute name of D_m) of the tuple variable t_{i_m}, and that the condition $\psi\,'$ (ψ trivially modified for attribute names and Quel syntax conventions) holds for u. The first line corresponds to the relevant range statements, the second to the target list, and the third to the where clause. The skeletal Quel statement is not quite correct syntactically, since domain names for the derived relation must be provided in the target list, and domain values may be expressions. We ignore such details for the remainder of the paper.

The semantics of a query on a TDB will be specified by providing a tuple calculus statement that denotes a static relation embedding a temporal relation which is the result of the query. The tuple calculus statement for a TQuel retrieve statement is very similar to that of a Quel retrieve statement: additional components corresponding to the valid, when, and as-of clauses are also present. Although the expressions appearing in all three clauses are similar syntactically, having their origins in path expressions, their semantics is quite different.

As an alternative, the semantics could have been specified by showing how any TQuel query can be transformed into an equivalent relational algebra expression, for which a semantics has been defined [Klug 1982]. This method has been used to express the semantics of SQL statements [Ceri & Gottlob 1985]. The tuple calculus was used instead for several reasons. The first is pragmatic: since TQuel is a minimal extension of Quel, its semantics should also be a minimal extension of Quel's semantics, which has been partially specified in tuple calculus, as discussed above. The second reason is that the tuple calculus expressions resulting from the transformation can themselves be easily transformed into relational algebra expressions, so no generality has been lost. Third, the tuple calculus statements are closer in form to statements in the query language, making the semantics more comprehensible. Finally, if an algebra is desired, it should probably be a temporal algebra. There is no generally accepted temporal algebra; proposals include [Gadia85B, Clifford85].

The next subsection will provide the semantics of e-expressions as function on time values or pairs of time values, ultimately yielding a time value. The following subsection examines the steps necessary to transform a temporal predicate into a conventional predicate for the when clause; the next subsection will do the same for the as-of clause. Section 5.6 uses these results to provide a tuple calculus semantics for the retrieve statement. The final subsections consider the modification statements and demonstrate a reduction to the Quel semantics.

5.3. The Valid Clause

As discussed previously, the valid clause specifies the time during which the derived tuple is valid. For derived intervals, the valid-from-to variant is used; for derived events, the valid-at variant is used. In both cases, an e-expression is used to specify a time value. The time value returned by the e-expression will in fact be one of the time values contained in one of the tuples associated with the variables involved in that expression. Hence, the e-expression is not actually *deriving* a *new* time value from the given time values; rather, it is *selecting* one of the *given* time values. Of course, the selection criteria can, and indeed usually do, depend on the relative temporal ordering of the original events.

Several researchers have proposed a formal semantics for particular variations on path expressions, involving denotational and axiomatic definitions [Berzins & Kapur 1977, Jayaraman 1982], or transformations into Petri nets [Lauer & Campbell 1975], parallel programs [Andler 1979, Jayaraman 1983], or even VLSI circuits [Anantharaman et al. 1985]. Since these semantics express the active nature of path expressions, that of constraining the occurrence of the relevant events, they are not applicable in the context of TQuel. The approach taken here associates each temporal constructor with a function on one or two intervals, returning an interval. Tuple variables are replaced with their associated valid time values. The result of an e-expression will hence be one of these time values. Individual time values will be represented as integers (a mapping from times and dates to integers is assumed); intervals will be represented as ordered pairs of integers. Anderson has developed a model of time at the conceptual level which is slightly more restrictive yet has several nice properties [Anderson 1982].

We define the temporal constructors after first defining a few auxiliary functions on integers (*First, Last*) or tuple variables (*event, interval*):

$$First(\alpha, \beta) = \begin{cases} \alpha & \text{if } Before(\alpha, \beta) \\ \beta & \text{otherwise} \end{cases}$$

$$Last(\alpha, \beta) = \begin{cases} \beta & \text{if } Before(\alpha, \beta) \\ \alpha & \text{otherwise} \end{cases}$$

$$event(t) = <t_{at}, t_{at}>$$

$$interval(t) = <t_{from}, t_{to}>$$

$$startof(<\alpha, \beta>) = <\alpha, \alpha>$$

$$endof(<\alpha, \beta>) = <\beta, \beta>$$

$$overlap(<\alpha, \beta>, <\gamma, \delta>) = <Last(\alpha, \gamma), First(\beta, \delta)>$$

$$extend(<\alpha, \beta>, <\gamma, \delta>) = <First(\alpha, \delta), Last(\beta, \gamma)>$$

A few comments are in order. First, if the e-expression is a correct one, i.e., if it results in an event, then the denotation of the expression will be defined to be the time value appearing as the first element of the ordered pair resulting from the application of these functions on the underlying tuples. The constraints assure us that the first element will be identical to the second element. The reader should verify that these definitions do indeed result in the correct time value. Secondly, as mentioned in Section 4.3, the *Before* predicate is the "$\leq$" predicate on integer time values. However, we wish to retain the *Before* predicate, because its semantics will be altered when indeterminacy is considered (in a later paper). Third, the translation is ***syntax-directed***: the semantic functions are in correspondence with the productions of the grammar (given in the Appendix) for e-expressions [Ceri & Gottlob 1985]. And finally, the definition of the ***overlap*** function assumes that the intervals do indeed overlap; if this constraint is satisfied, then the ordered pairs $<\alpha, \beta>$ generated by these functions will always represent intervals, i.e., the ordered pairs will satisfy $Before(\alpha, \beta)$. Invalid e-expressions will be handled with an additional clause in the tuple calculus statement presented in Section 5.6.

As an example, the e-expression

start of (a **overlap** b)

is transformed into

startof(***overlap***(***interval***(a), ***interval***(b)))

(we assume that the tuple variables a and b are associated with interval relations). Applying the functions defined above results in

$\rightarrow \textit{startof}(\textit{overlap}(<a_{from}, a_{to}>, <b_{from}, b_{to}>))$
$\rightarrow \textit{startof}(<\textit{Last}(a_{from}, b_{from}), \textit{First}(a_{to}, b_{to})>)$
$\rightarrow <\textit{Last}(a_{from}, b_{from}), \textit{Last}(a_{from}, b_{from})>$

Hence the denotation of this expression is $\textit{Last}(a_{from}, b_{from})$. The use of this time value will be discussed shortly.

5.4. The When Clause

The when clause is the temporal analogue of the where clause. The temporal predicate in the when clause determines whether the tuples may participate in the derivation by examining their relative order. Expressing this formally involves generating a conventional predicate on the temporal domains of the tuples in the underlying relations. This predicate is generated in three steps. First the **and** and **or** logical operators are distributed over the temporal operators. Then the tuple variables and the temporal constructors are replaced by the functions defined in the previous subsection. Finally, the **and**, **or**, and **not** operators are replaced by the logical predicates, and the temporal predicate operators by analogous predicates on ordered pairs of integers:

$$precede(<\alpha, \beta>, <\gamma, \delta>) = Before(\beta, \gamma)$$

$$overlap(<\alpha, \beta>, <\gamma, \delta>) = Before(\alpha, \delta) \wedge Before(\gamma, \beta)$$

The result is a conventional predicate on the valid times of the tuple variables appearing in the when clause.

As an example, the temporal predicate

(start of (a **overlap** b)) **precede** c **or** (c **precede** a)

remains unchanged after the first step. The second step results in

$\rightarrow$ (*startof*(*overlap*(*interval*(a), *interval*(b))) **precede** *interval*(c)) **or** (*interval*(c) **precede** *interval*(a))

$\rightarrow$ (*startof*(*overlap*($<a_{from}, a_{to}>, <b_{from}, b_{to}>$)) **precede** $<c_{from}, c_{to}>$)
or ($<c_{from}, c_{to}>$ **precede** $<a_{from}, a_{to}>$)

$\rightarrow$ (*startof*($<Last(a_{from}, b_{from}), First(a_{to}, b_{to})>$) **precede** $<c_{from}, c_{to}>$)
or ($<c_{from}, c_{to}>$ **precede** $<a_{from}, a_{to}>$)

$\rightarrow$ ($<Last(a_{from}, b_{from}), Last(a_{from}, b_{from})>$ **precede** $<c_{from}, c_{to}>$) **or** ($<c_{from}, c_{to}>$ **precede** $<a_{from}, a_{to}>$)

The third step results in

$$Before(Last(a_{from}, b_{from}), c_{from}) \vee Before(c_{to}, a_{from})$$

5.5. The As Of Clause

The temporal constructors appearing in the as-of clause can be replaced with their functions on ordered pairs of integers and the temporal constants (strings) can be replaced by their corresponding ordered pairs of integers. The result can be evaluated at "compile-time", resulting in a single integer, for the **as of** variant, and two integers in the **as of through** variant. For convenience, these times will be converted into an interval by interpreting **through** as **extend**.

as of start of "1984" **through** "October, 1984"

will, by using the functions defined in Section 5.3, be converted to

$$
\begin{aligned}
&\mathit{extend}(\mathit{startof}(<1009, 1021>), <1018, 1019>)\\
&\rightarrow \mathit{extend}(<1009, 1009>, <1018, 1019>)\\
&\rightarrow <\mathit{First}(<1009, 1018>), \mathit{Last}(<1009, 1019>)>\\
&\rightarrow <1009, 1019>
\end{aligned}
$$

Here, we have quite arbitrarily mapped months into the integer specifying the number of months since January, 1900.

5.6. The TQuel Retrieve Statement

A formal semantics for the TQuel retrieve statement can now be specified. Let Φ_ϵ be the function corresponding to the e-expression ϵ as generated in the process discussed in Section 5.3. Let Π_τ be the predicate corresponding to the temporal predicate τ as generated by the process discussed in Section 5.4. Note that Φ_ϵ and Π_τ will contain only the functions *First* and *Last* and the predicates *Before*, $\wedge$, $\vee$, $\neg$; the rest of the functions, and Φ_α entirely (where α appears in an as-of clause), can be evaluated at "compile-time". Of course, the defaults provide the appropriate expressions when a clause is not present in the query. Given the query

```
range of $t_1$ is $R_1$
 . . .
range of $t_k$ is $R_k$
retrieve $(t_{i_1}.D_1, \ldots, t_{i_r}.D_r)$
        valid from $\upsilon$ to $\chi$
        where $\psi$
        when $\tau$
        as of $\alpha$ through $\beta$
```

the tuple calculus statement has the following form

$$\Big\{u^{(r+4)} \mid (\exists\, t_1) \cdots (\exists\, t_k)\,(R_1(t_1) \wedge \cdots \wedge R_k(t_k)$$

$$\wedge\; u[1] = t_{i_1}[j_1] \wedge \cdots \wedge u[r] = t_{i_r}[j_r]$$

$$\wedge\; u[r+1] = \Phi_\upsilon \wedge u[r+2] = \Phi_\chi \wedge Before(u[r+1],\, u[r+2])$$

$$\wedge\; u[r+3] = \# \wedge u[r+4] = \infty$$

$$\wedge\; \psi\,'$$

$$\wedge\; \Pi_\tau$$

$$\wedge\; (\forall l)(1 \le l \le k.(Before(\Phi_\alpha,\, t_l[stop]) \wedge Before(t_l[start],\, \Phi_\beta)))$$

$$)\Big\}$$

The first line states that each tuple variable ranges over the correct relation, and is from the Quel semantics. The resulting tuple is of arity $r+4$, and is comprised of r explicit domains and four implicit domains (*from, to, start*, and *stop*). The second line, also from the Quel semantics, states the origin of the values in the explicit domains of the derived relation. The third line originates in the valid clause, and specifies the values of the *from* and *to* valid times. Notice that these times must obey the specified ordering. The fourth line specifies the values of the *start* and *stop* transaction times. "#" is replaced with the integer corresponding to the current transaction; this integer must be monotonically increasing. "∞" is replaced with a distinguished integer, say 0, which must not correspond to a valid transaction. The next line originates in the where clause, and is from the Quel semantics. The fifth line is the predicate from the when clause. The last line originates in the as-of clause, and states that the tuple associated with each tuple variable must have a transaction interval that overlaps the interval specified in the as-of clause (Φ_α and Φ_β will be constant time values, i.e., specific integers).

Note that Φ_υ, Φ_χ, ψ', and Π_τ are functions over the *from*, *to*, and explicit domains of a subset of the tuple variables. If t is a tuple variable associated with an interval relation and appears in an e-expression or temporal predicate, then the *from* and *to* time values are passed to the relevant function; if t is associated with an event relation, then only the *at* time value is used. The superscript $(r+4)$ indicates that the tuple u has r explicit domains and 4 implicit domains, the starting and stopping time values for the valid and transaction intervals; events will have only three implicit domains. The entire transformation from a TQuel query to a tuple calculus expression may be considered to be syntax-directed, as discussed briefly in Section 5.3.

We complete the discussion of the semantics of the retrieve statement with two examples, one realistic but somewhat simple; the other contrived yet more comprehensive. The first is the semantics of the query shown in Example 9.

$$\Big\{u^{(1+4)} \mid (\exists\, \mathrm{f1})\, (\exists\, \mathrm{f2})\, (\exists\, \mathrm{a})\, (\mathrm{Faculty(f1)} \wedge \mathrm{Faculty(f2)} \wedge \mathrm{Associates(a)}$$

$$\wedge\, u[1] = \mathrm{f1}[1]$$

$$\wedge\, u[2] = \mathrm{f1}[3] \wedge u[3] = \mathrm{f2}[3] \wedge \mathit{Before}(u[2], u[3])$$

$$\wedge\, u[4] = 123 \wedge u[5] = 0$$

$$\wedge\, \mathrm{f1}[1] = \mathrm{f2}[1] \wedge \mathrm{f1}[2] = \text{"Assistant"} \wedge \mathrm{f2}[2] = \text{"Full"}$$

$$\wedge\, \mathit{Before}(\mathrm{f1}[3], a[3]) \wedge \mathit{Before}(\mathrm{a}[2], \mathrm{f1}[4]) \wedge \mathit{Before}(\mathrm{f2}[3], \mathrm{a}[3])$$

$$\wedge\, \mathit{Before}(\mathrm{a}[2], \mathrm{f2}[4])$$

$$\wedge\, \mathit{Before}(1020, \mathrm{f1}[6]) \wedge (\mathit{Before}(\mathrm{f1}[5], 1020)$$

$$\wedge\, \mathit{Before}(1020, \mathrm{f2}[6]) \wedge (\mathit{Before}(\mathrm{f2}[5], 1020)$$

$$\wedge\, \mathit{Before}(1020, \mathrm{a}[5]) \wedge (\mathit{Before}(\mathrm{a}[4], 1020)$$

$$)\Big\}$$

The second example, which includes several temporal expressions used as examples in previous sections, is given below.

range of a **is** A
range of b **is** B
range of c **is** C
retrieve (a.M, b.O, c.Q)
    valid from start of (a **overlap** b) **to end of** (a **overlap** b)
    where a.N = b.P **and** b.P = c.R
    when (**start of** (a **overlap** b)) **precede** c **or** (c **precede** a)
    as of start of "1984" **through** "October, 1984"

Example 13: A Contrived Example

This query references relations containing the following domains:

A [M N (from to start stop)]
B [O P (from to start stop)]
C [Q R (from to start stop)]

The implicit temporal domains are in parentheses (A, B, and C are all interval relations). The query then has the following semantics,

$$\Big\{ u^{(3+2)} \mid (\exists a)(\exists b)(\exists c)(A(a) \wedge B(b) \wedge C(c)$$

$$\wedge\ u[1] = a[1] \wedge u[2] = b[1] \wedge u[3] = c[1]$$
$$\wedge\ u[4] = Last(a[3], b[3]) \wedge u[5] = First(a[4], b[4]) \wedge Before(u[4], u[5])$$
$$\wedge\ u[6] = 124 \wedge u[7] = 0$$
$$\wedge\ a[2] = b[2] \wedge b[2] = c[2]$$
$$\wedge\ (Before(Last(a[3], b[3]), c[3]) \vee \mathrm{Before}(c[4], a[3]))$$
$$\wedge\ Before(1009, a[6]) \wedge Before(a[5], 1019)$$
$$\wedge\ Before(1009, b[6]) \wedge Before(b[5], 1019)$$
$$\wedge\ Before(1009, c[6]) \wedge Before(c[5], 1019)$$

$$)\Big\}$$

The correspondence between the Quel and TQuel tuple calculus semantics is striking. The tuple calculus statement for the Quel retrieve statement consists of a component associated with the tuple variables appearing in the query (the first line), a component associated with the target list (the second line), and a component associated with the where clause (the fifth line). The tuple calculus statement for the TQuel retrieve statement adds four additional lines, one each associated with the valid clause (the third line), the when clause (the sixth line), the as-of clause (the last line), and one specifying the transaction time for the derived tuples (the fourth line). The additional lines in the tuple calculus statement are also similar in form to those associated with the analogous Quel statements: the where, when, and as-of clauses all generate predicates, and the target list and valid clause generate equalities.

5.7. Modification Statements

In specifying the semantics of the TQuel modification statements, we will again proceed by examining the tuple calculus semantics of the analogous Quel statements. These have never appeared in the literature; fortunately, they are easy to derive (such is not the case for the other major static relational query language SQL [Ceri & Gottlob 1985]). The skeletal Quel append statement,

append to R $(t_{i_1}.D_1, \ldots, t_{i_r}.D_r)$
where ψ

has the tuple calculus semantics

$$\mathrm{R}' = \mathrm{R} \bigcup \Big\{ u^{(r)} \mid (\exists\, t_1) \cdots (\exists\, t_k)\, (R_1(t_1) \wedge \cdots \wedge R_k(t_k)$$
$$\wedge\, u[1] = t_{i_1}[j_1] \wedge \cdots \wedge u[r] = t_{i_r}[j_r]$$
$$\wedge\, \psi') \Big\}$$

The set being appended is identical to that for the Quel retrieve statement (see Section 5.2). Note that the set being appended may contain tuples already in R. We assume that the integrity constraints, particularly those relating to keys, have already been checked and that the resulting relation R $'$ will satisfy these constraints.

The semantics for the skeletal TQuel append statement,

range of t_1 **is** R_1
. . .
range of t_k **is** R_k
append to R $(t_{i_1}.D_1, \ldots, t_{i_r}.D_r)$
    valid from υ **to** χ
    where ψ
    when τ

is somewhat complicated, because the set to be unioned with the existing relation should only contain tuples that are not valid in the existing relation. We cannot depend on the union working correctly when the tuples being appended are identical to tuples in the current historical relation. For example, if on 9-85 we execute

```
range of f is Faculty
append to Faculty (Name = "Merrie", Rank = "Assistant")
      valid from "8-77" to "12-82"
```

Example 14: Merrie actually joined the department a month earlier.

then we will have to append the following tuple:

Name	Rank	Valid Time		Transaction Time	
		(From)	(To)	(Start)	(Stop)
Merrie	Assistant	8-77	9-77	9-85	∞

Note that the *to* time is 9-77, since a tuple already exists in the relation valid from 9-77 to 12-82 (c.f., Figure 7).

We now give the tuple calculus statement for the skeletal TQuel append statement. As before, we assume that the integrity constraints have been checked previously. In the case of a temporal relation, this implies that for a tuple T with *stop* $= \infty$, if T matches an appended tuple A on the key attributes and is valid at any point in the interval specified in the valid clause, then T will also match A on the non-key attributes, and the union will not affect the presence of A.

$$R' = R \bigcup \Big\{ u^{(r+4)} \mid (\exists t_1) \cdots (\exists t_k) (\exists s) (R_1(t_1) \wedge \cdots \wedge R_k(t_k)$$

$$\wedge\, u[1] = t_{i_1}[j_1] \wedge \cdots \wedge u[r] = t_{i_r}[j_r]$$

$$\wedge\, u[r+3] = \# \wedge u[r+4] = \infty$$

$$\wedge\, \psi'$$

$$\wedge\, \Pi_\tau$$

$$\wedge\, (\forall l)\,(1 \leq l \leq k . t_l[stop] = \infty)$$

$$\wedge\, ((\exists s)(R(s) \wedge (\forall l(1 \leq l \leq r) . s[l] = u[l]) \wedge (C_1 \vee C_2 \vee C_3 \vee C_4))$$

$$\vee\, (\neg(\exists s)(R(s) \wedge (\forall l(1 \leq l \leq r) . s[l] = u[l])) \wedge u[r+1] = \Phi_\upsilon \wedge u[r+2] = \Phi_\chi))$$

$$) \Big\}$$

where

$$C_1 = (Before(s[r+1], \Phi_\upsilon) \wedge Before(\Phi_\upsilon, s[r+2]) \wedge Before(s[r+2], \Phi_\chi) \wedge u[r+1] = s[r+1] \wedge u[r+2] = \Phi_\upsilon)$$

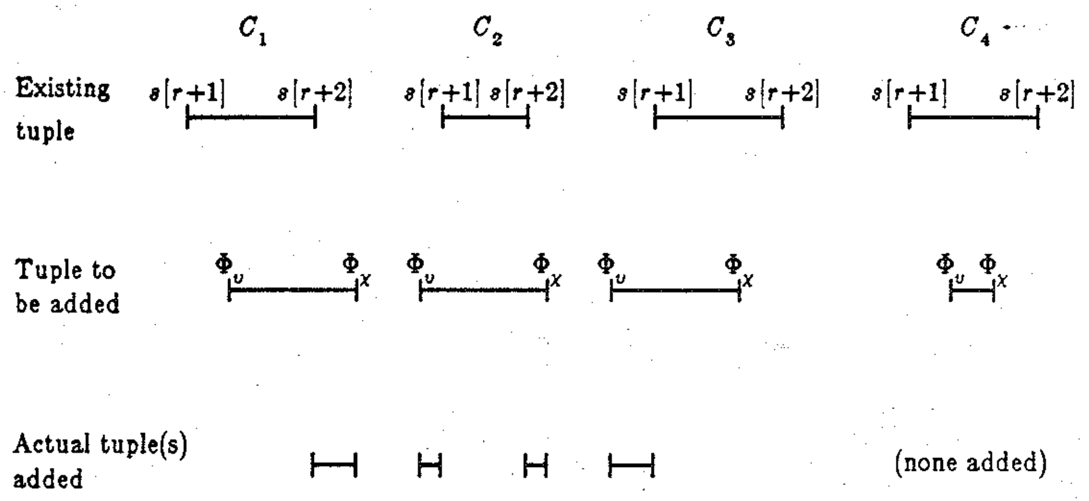
$$C_2 = (Before(\Phi_\upsilon, s[r+1]) \wedge Before(s[r+2], \Phi_\chi) \wedge ((u[r+1] = \Phi_\upsilon \wedge u[r+2] = s[r+1]) \vee (u[r+1] = s[r+2] \wedge u[r+2] = \Phi_\chi)))$$

$$C_3 = (Before(\Phi_\upsilon, s[r+1]) \wedge Before(s[r+1], \Phi_\chi) \wedge Before(\Phi_\chi, s[r+2]) \wedge u[r+1] = \Phi_\upsilon \wedge u[r+2] = s[r+1])$$

$$C_4 = (Before(s[r+1], \Phi_\upsilon) \wedge Before(\Phi_\chi, s[r+2]) \wedge \text{False}$$

Again, the set being appended is similar to the TQuel retrieve statement (see the previous section), with two major changes. The first is that the as-of clause is assumed to be **as of** "now", since the statement should only modify the current historical relation (c.f., the sixth line). The second change is the rather complicated computation of the valid times for the tuples to be added, appearing as the last two lines of the tuple calculus statement, which replace the third line in the tuple calculus statement for the retrieve statement. The four clauses $C_1, \ldots, C_4$ in the seventh line handle the various overlap situations between the tuples to be added and the tuples identical in the explicit domains that already exist during this valid interval. In particular, C_4 states that if the tuple already exists in R over the entire valid time, there is no need to add it. The last line states that the valid times are as specified in the valid clause if no such tuples exist during this valid interval. Figure 15 shows the overlap handled by each clause, and the resulting valid interval(s). Note that one, two, or no tuples are added, depending on the valid clause specified and the tuples already present in the relation.

Figure 15: Calculating the Valid Time in an Append Statement



The semantics of the delete statement shows a similar increase in complexity. The Quel statement

range of r **is** R
delete r
    where ψ

has the tuple calculus semantics

$$R' = \left\{ u^{(r)} \mid R(u) \wedge \neg\psi'(u) \right\}$$

Note that the predicate can only reference the one tuple variable.

We first look at an example of the TQuel delete statement before delving into its semantics.

```
range of f is Faculty
delete f
       where f.Name = "Jane"
       valid from "3-81"
```

Example 15: Jane left the department in March, 1981.

This statement will modify the transaction stop time of the last tuple in Figure 7, and will append an additional tuple (we give both here):

Name	Rank	Valid Time		Transaction Time	
		(From)	(To)	(Start)	(Stop)
Jane	Full	11-80	∞	10-80	9-85
Jane	Full	11-80	3-81	9-85	∞

Hence, the delete statement will perhaps change some transaction stop times from "∞" to "now", and will perhaps also add tuples with a transaction start time of "now". For the skeleton TQuel delete statement

```
range of s is R
delete s
       valid from υ to χ
       where ψ
       when τ
```

the tuple calculus statement is

$$R' = \left\{ s^{(r+4)} \mid (R(s) \wedge (\neg\psi' \vee s[r+4] \neq \infty \vee Before(s[r+2], \Phi_\upsilon) \vee Before(\Phi_\chi, s[r+1]))) \right\}$$

$$\cup \left\{ u^{(r+4)} \mid (\exists\, s)\, (R(s) \wedge \psi' \wedge s[r+4] = \infty \wedge Before(s[r+1], \Phi_\chi) \wedge Before(\Phi_\upsilon, s[r+2]) \right.$$

$$\wedge\, (\forall i. 1 \leq l \leq r. u[i] = s[i])$$

$$\wedge\, (C_1 \vee C_2 \vee C_3 \vee C_4)$$

$$\left.) \right\}$$

where

$$C_1 = (Before(s[r+1], \Phi_\upsilon) \wedge Before(\Phi_\upsilon, s[r+2]) \wedge Before(s[r+2], \Phi_\chi)$$
$$\wedge ((u[r+1] = s[r+1] \wedge u[r+2] = \Phi_\chi \wedge u[r+3] = s[r+3] \wedge u[r+4] = \infty)$$
$$\vee (u[r+1] = \Phi_\upsilon \wedge u[r+2] = s[r+2] \wedge u[r+3] = s[r+3] \wedge u[r+4] = \text{"now"}))$$

$$C_2 = (Before(\Phi_\upsilon, s[r+1]) \wedge Before(s[r+2], \Phi_\chi)$$
$$\wedge u[r+1] = s[r+1] \wedge u[r+2] = s[r+2] \wedge u[r+3] = s[r+3] \wedge u[r+4] = \text{"now"})$$

$$C_3 = (Before(\Phi_\upsilon, s[r+1]) \wedge Before(s[r+1], \Phi_\chi) \wedge Before(\Phi_\chi, s[r+2])$$
$$\wedge ((u[r+1] = s[r+1] \wedge u[r+2] = \Phi_\chi \wedge u[r+3] = s[r+3] \wedge u[r+4] = \text{"now"})$$
$$\vee (u[r+1] = \Phi_\upsilon \wedge u[r+2] = s[r+1] \wedge u[r+3] = s[r+3] \wedge u[r+4] = \infty))$$

$$C_4 = (Before(s[r+1], \Phi_\upsilon) \wedge Before(\Phi_\chi, s[r+2])$$
$$\wedge ((u[r+1] = s[r+1] \wedge u[r+2] = \Phi_\upsilon \wedge u[r+3] = s[r+3] \wedge u[r+4] = \infty)$$
$$\vee (u[r+1] = \Phi_\upsilon \wedge u[r+2] = \Phi_\chi \wedge u[r+3] = s[r+3] \wedge u[r+4] = \text{"now"})$$
$$\vee (u[r+1] = \Phi_\chi \wedge u[r+2] = s[r+2] \wedge u[r+3] = s[r+3] \wedge u[r+4] = \infty))$$

The first set contains all tuples in past historical relations of R and all tuples in the current historical relation of R which do not satisfy the predicate in the where clause or whose valid intervals do not overlap with the specified valid interval. These tuples are not affected by the delete statement. The second set deals with the remaining tuples, in a manner similar to that employed in the semantics of the append statement. In the situation covered by C_1, the tuple to be deleted starts after the existing tuple starts, but still overlaps the existing tuple (see Figure 15). The existing tuple is broken into two intervals, the first which remains (*stop* = ∞) and the second which is removed (*stop* = "now"). This is the situation illustrated in the example above of the delete statement. In the situation covered by C_2, the tuple to be deleted overlaps the existing tuple completely, so the existing tuple is deleted (*stop* = "now"). C_3 is similar to C_1. In the situation covered by C_4, the existing tuple is partitioned into three intervals, and only the middle one is deleted.

The semantics of the replace statement is even more complex. However, since the replace statement has a semantics identical to a delete statement followed by an append statement, the tuple calculus statement follows from the two just presented.

5.8. Reduction to the Quel Semantics

If a TQuel statement does not contain a valid, when, or as-of clause, then it looks identical to the analogous standard Quel retrieve statement; thus it should have an identical semantics. However, an Ingres database is not temporal; it is a static database. Hence, the tuples participating in a Quel statement are in the static relation that is the result of the last transaction performed on the database (i.e., are *current*) and are valid at the time the statement is executed. Note that the statement must not refer to any tuple variables associated with event relations. The tuples in such relations are valid for only an instant, and hence would not ever appear in a static database.

We will show that the TQuel semantics just presented reduces to the standard Quel semantics when applied to a *static database slice* (all current tuples valid at a particular time) of the TDB. A static database slice at time τ is formed by first eliminating the event relations (in Section 2.3 we argued that static relations cannot represent events at all), eliminating all tuples with a *start* time greater than τ and with a *stop* time less than τ, eliminating all tuples not valid at τ, and finally removing the implicit time domains.

The reduction proof will be illustrated on a simple retrieve statement; the interactions are illustrated in Figure 16. Assume that Q is a syntactically correct Quel retrieve statement. (Example 1 is such a statement.) Then Q is also a syntactically correct TQuel statement. Q may be applied to a TDB (for example, the one given in Figure 7) to define a derived temporal relation R_T (the one in Figure 8). In processing the query Q, the defaults for the valid, when, and as-of clauses discussed in Section 4.5 will be applied. A static database slice at time τ of this derived temporal relation results in a conventional relation, R_S. For example, assume that the query Q is executed on January 1, 1984. The database slice at $\tau =$ January 1, 1984 of the Associates relation of Figure 8 is shown in Figure 17. Now, the query Q may also be applied to a static database slice at the same time τ of the entire TDB (shown in Figure 18) to arrive at another static relation, R_S'. To show that the TQuel semantics reduces to the standard Quel semantics when applied to a static database slice, we must show that

$$R_S = R_S'$$

The reduction implies that Figures 6 and 18 are identical.

Figure 16: Outline of the Reduction Proof

$$\begin{tikzcd}
\text{TDB} \arrow[r, "\text{slice at } \tau"] \arrow[d, "Q", "\substack{\text{(TQuel}\\ \text{semantics)}}"'] & \text{static DB} \arrow[d, "\substack{\text{(Quel}\\ \text{semantics)}}", "Q"'] \\
R_T \arrow[r, "\text{slice at } \tau"] & R_S{}' = R_S
\end{tikzcd}$$

Figure 17: Slice of the Associates Relation at January 1, 1984

Associates (Name):

Name
Merrie
Tom

Figure 18: A Database Slice at January 1, 1984

Faculty (Name, Rank):

Name	Rank
Jane	Full
Merrie	Associate
Tom	Associate

The proof of this equality revolves around the defaults for the valid, when, and as-of clauses specified in Section 4.5. The defaults effectively take a database slice at τ = "now", which is the time the query is executed. The default when clause states that all the underlying tuples are valid and the default valid clause states that the underlying tuples are valid. The resulting tuples are guaranteed to be current by the tuple calculus semantics of the retrieve statement. This intuition supports the easily shown equality (actually, identity) of the tuple calculus semantics for R_S and R_S'. Similar reductions can be argued concerning the modification statements, as their defaults were specifically chosen to ensure their reducibility to the standard Quel semantics.

The benefit of this reduction is that the intuition and understanding gained by using Quel on a static database applies to TQuel on a TDB.

6. Implementation

The formulation of the TQuel semantics as tuple relational calculus expressions offers a straightforward means to implement a TDBMS. A TQuel query (or update statement) can be mapped into a tuple calculus statement, which may then be mapped into a Quel statement on the static relations which embed the temporal relations. The TQuel query in Example 9 would be mapped into the equivalent Quel query

```
range of f1 is Faculty
range of f2 is Faculty
range of a is Associates
retrieve into Starsof1984 (Name = f1.Name, validfrom = f1.validfrom,
              validto = f2.validfrom, transactionstart = 123, transactionstop = 0)
    where f1.Name = f2.Name and f1.Rank = "Assistant" and f2.Rank = "Full"
          and f1.validfrom <= a.validto and a.validfrom <= f1.validto
          and f2.validfrom <= a.validto and a.validfrom <= f2.validto
          and 1020 <= f1.transactionstop and f1.transactionstart <= 1020
              and 1020 <= f2.transactionstop and f2.transactionstart <= 1020
              and 1020 <= a.transactionstop and a.transactionstart <= 1020
```

using the formal semantics as given in Section 5.6, on the following static schemas:

```
Faculty (Name, Rank, validfrom, validto, transactionstart, transactionstop)
Associates (Name, validfrom, validto, transactionstart, transactionstop)
```

This conversion can always be done if two functions, First and Last, both taking two integers as arguments, are added to Quel (Example 13 would require the use of these functions). It should be emphasized that the conversion from TQuel to Quel is an entirely separate process from the reduction to the Quel semantics discussed in Section 5.8.

We have extended the Ingres DBMS [Stonebraker et al. 1976] along these lines: temporal relations are stored as static relations, and the TQuel statements are converted to Quel statements and processed normally by Ingres (the implementation differs from this explanation in certain details). We are instrumenting the prototype to gather performance measures. We expect performance to degrade to unacceptable levels as updates are made and the static relations grow monotonically. However, the prototype will serve as a testbed for further study and as a datapoint for comparison on the performance of new access methods and optimization strategies.

7. Conclusion

7.1. Summary

This paper has presented the syntax and formal semantics for the augmented statements in TQuel. The discussion proceeded in an incremental fashion for both the syntax and semantics. First, the Quel syntax was presented informally. Temporal analogues for the where clause and the target list were examined. A more formal presentation, including a digression on constants and defaults, completed the presentation of TQuel's syntax.

After a short review of tuple calculus, the semantics of e-expressions was described as functions on time values or pairs of time values, ultimately yielding a time value. A transformation system provided the semantics of temporal expressions, yielding a conventional predicate on the tuples participating in the expression. At that point, a tuple calculus expression for TQuel retrieve statements without aggregates was presented. The semantics of the modification statements were discussed. The semantics reduces to the standard Quel semantics when the time domain is fixed at a particular time. To the author's knowledge, the complete semantics of no other query language, static or temporal, has been presented in the literature. Finally, a prototype implementation was described.

7.2. Other Temporal Query Languages

In this section we compare TQuel to other query languages that reference time. Two basic approaches are found in the literature. The most straightforward approach augments a static query language with a construct to project a static database slice of a TDB; examples include TOD, DATA, TRM, and Gemstone. Such languages do not fully support historical queries, though they may support rollback. The Time Oriented Databank (TOD), one of the earliest database systems to incorporate time, records dynamic attributes, whose values may change over time, in a relation in which each tuple has a time attributes specifying when the properties were observed [Blum 1982, Wiederhold et al. 1975]. Querying is accomplished using analysis and reporting procedures written in PL/1; the system supports these procedures by creating auxiliary files, which can optionally be ordered by date. The Dynamic Alerting Transaction System (DATA) extends the relational model to include time by viewing the database as time-ordered lists of transactions, each consisting of a tuple and a time when that tuple became valid [Ariav & Morgan 1981]. The database can be queried at previous points of time, or a sequence of recorded events between two times may be displayed. The Time Relational Model (TRM) is noteworthy in that the query language references both valid and transaction time [Ben-Zvi 1982]. TRM augments SQL [SQL/DS 1981] with a *time-view* construct, identifying the valid and transaction times for the static database slice to which the remainder of the SQL query is to be applied. Hence the rollback operation is provided, but historical queries are not possible. Gemstone is the most recent of the query languages in this category; it supports only transaction time, and hence only the rollback operation [Copeland & Maier 1984].

The second approach is to include in the query language temporal constructs that allow historical queries (queries over tuples containing different valid times). Three relational query languages, in addition to TQuel, have taken this approach. Of course, natural languages also fall into this category [Clifford & Warren 1983].

The first, LEGOL 2.0, involved formalizing legislation, where the history of a case is particularly relevant [Jones et al. 1979, Jones & Mason 1980]. The model supported by this system allows time attributes specifying the period of time each tuple is valid; events may not be stored, and transaction time is not supported. Hence LEGOL 2.0 is an historical query language. LEGOL 2.0 is based on the relational algebra [Codd 1972]. The language was never implemented, although an earlier version of the language was implemented [Stamper 1976] using ISBL [Todd 1976]. In addition, no attempt at a formalization either of the language or of the way the temporal constructs of the language were to be implemented has been made.

The query language associated with the Temporally Oriented Data Management System (TODMS) is similar to that of TRM in that it is an extension of SQL and it supports both valid and transaction time [Ariav 1984]. Unlike TRM, it is a true temporal query language, supporting rollback with an as-of clause quite similar to that of TQuel, and supporting historical queries through the additional AT, WHILE, DURING, BEFORE and AFTER constructs. The major limitation is that only one relation may be referenced in a query; hence the language has less expressive power than most static query languages, which support queries over multiple relations. The query language has not been formalized, and no implementation has been attempted for TODMS.

Finally, the Homogeneous Temporal Query Language (HTQuel) supports historical queries but not rollback [Gadia & Vaishnav 1985]. This relational calculus language is based on the representation of an historical database where the time intervals are associated with attributes (this representation is discussed in Section 5.1). HTQuel is a more substantial extension to Quel than is TQuel. Whereas expressions in Quel and TQuel involve only constants and tuple variables, HTQuel introduces *temporal domains*, which are finite unions of intervals, and *instants* for use in expressions. HTQuel also introduces identifiers that can have temporal domains or instances as values, an assignment statement, a statement to define the type of a tuple variable, and a collection of functions which extract individual intervals or instants from a temporal domain. Such complexity may not be necessary; equivalent TQuel queries exist for all of the HTQuel examples given in [Gadia & Vaishnav 1985] that do not utilize the extraction functions. These functions support *temporal navigation*; we agree with Ariav that such functionality should be supported not by the query language but at the programming language-DBMS interface [Ariav 1984], as navigation in general is not consistent with the non-procedural nature of predicate calculus oriented query languages. Although augmented update operations are defined, they destructively modify the database, so HTQuel supports historical databases; no constructs for manipulating transaction time are provided. No formal semantics is given, and no implementation has been attempted for HTQuel.

In summary, we note that, while the languages examined above differ greatly in their details, a consensus seems to be emerging concerning the basic features of an acceptable temporal query language: it should be based on the relational calculus, support queries over multiple relations, support both rollback and historical queries, incorporate in some manner common temporal concepts such as before, while, and at, handle both instants and time intervals, and have a simple structure. As a final note, we agree with Clifford and Warren that a formal semantics is a necessary requirement for any proposed query language [Clifford & Warren 1983].

7.3. Further Work

Almost every issue raised in the context of static relational database must be addressed anew in the context of temporal databases. This paper has made a start by defining a temporal query language and providing a formal semantics for this language. However, much more research is necessary before a viable TDBMS can be developed.

Many additions are possible to the language itself. The operators available for e-expressions and temporal predicates are certainly not exhaustive, and new ones could be added easily to both the language and its semantics. Another possible addition concerns temporal constants. The temporal constants used in this paper are *absolute*, in that they denote a particular time interval. *Relative* constants would also be quite useful. The following is a variant of Example 5,

```
range of a is Associates
retrieve into Disgruntled (Name = a.Name)
        when (start of a) precede "5 years" precede (end of a)
```

Example 16: Who has been an associate professor for at least 5 years?

The semantics for relative constants is still under study.

Quel supports three domain types, in multiple sizes: integer (1, 2, and 4 bytes long), floating point (4 and 8 bytes long), and character data (1 to 255 bytes long). One necessary extension is a data type with values that vary over the period of time the tuple was valid (this data type is distinct from the temporal data type discussed in Section 3.4, which has a constant value for the entire valid interval). As was stressed in Section 3, caution is needed to ensure that such domains are used correctly. Quel also supports scalar functions such as **abs**, **mod**, and **sin**. Scalar temporal functions, such as **duration**, which compute time varying domains, are needed in the language.

Quel includes the aggregate operators **count**, **sum**, **avg**, **min**, **max**, and **any** to aggregate a computed expression over a set of tuples. Aggregate operators are more complicated in TQuel, due to the time-varying behavior of relations. The standard aggregate operators have been included in TQuel [Snodgrass 1984]. Temporal aggregate functions should also be made available to the user. These functions would select tuples from a set of tuples based on criteria involving the valid or transaction time. Temporal aggregate functions would support temporal navigation [Gadia & Vaishnav 1985] in a limited fashion, yet would be consistent with the declarative nature of TQuel. Various temporal aggregate operators have been proposed in the context of other query languages [Ariav 1984, Ben-Zvi 1982].

The issue of completeness naturally arises whenever a new query language is proposed. A query language is said to be *complete* if it can simulate tuple relational calculus, as defined by Codd [Codd 1972]. TQuel is complete under this definition, since it is an extension of Quel, which has been shown to be complete [Ullman 1982]. However, a more satisfying concept would be that of *temporal completeness*, for which there is no generally accepted definition. Gadia and Vaishnav have proposed a particular temporal relational algebra [Gadia 1985] as a benchmark [Gadia & Vaishnav 1985]; however, the issue of why the is an appropriate benchmark for completeness was never discussed. Two reasons why the algebra is perhaps inappropriate are that it is a multisorted algebra over relations and temporal domains, and that it only concerns valid time. Nevertheless, if this definition of completeness is adopted, then TQuel *is* temporally complete.

A host of other issues must be considered in the design of a temporal query language. How should time granularity (e.g., hour, work week) be handled [Anderson 1982]. Temporal constants, as discussed in Section 4.2, provide only a partial answer. Should valid and transaction time be linear or branching? Branching time, while more complex than linear time, does have some interesting properties [Ariav 1984, Stonebraker & Keller 1980]. How should changes to the schema be incorporated into the language? How should indeterminacy be incorporated? How should temporal relations be displayed? High resolution display devices look quite promising [Ariav 1984, Shannon 1985]. Should periodic or cyclic events and intervals (e.g., fiscal year, monthly payments) or causality be incorporated [Ariav & Morgan 1982, Ariav 1984]? How does TQuel correspond to the user's temporal perception? Further work is necessary in all of these areas.

The prototype described in Section 6 will exhibit unacceptable performance as updates are made to this database. Much more research is needed, particularly in the areas of new access methods, query optimization techniques, and use of novel storage devices such as optical disks.

Temporal database management systems in general are at approximately the same stage as static relational systems were in the early 1970's [Kim 1979]: several high-level, nonprocedural query languages have been designed, at least one query language has been formalized, and a prototype implementations exist. All the questions asked concerning static relational databases, including those that have already been answered, must be asked (and answered) anew in the context of temporal databases.

8. Acknowledgements

The author is grateful to Ilsoo Ahn and Bharat Jayaraman for many helpful discussions on the material in this paper, to Ilsoo Ahn for implementing the prototype described in Section 6, and to the referees for insightful observations that greatly improved the paper.

9. Bibliography

[Allen 1983] Allen, J. F. *Maintaining Knowledge about Temporal Intervals. Communications of the Association of Computing Machinery,* 26, No. 11, Nov. 1983, pp. 832-843.

[Allen 1981] Allen, J.F. *An Interval-Based Representation of Temporal Knowledge.* In *Proceedings of the International Joint Conference on Artificial Intelligence,* Vancouver: 1981 pp. 221-226.

[Allen 1984] Allen, J.F. *Towards a General Theory of Action and Time in Artificial Intelligence. AI Journal,* 23, No. 2, July 1984, pp. 123-154.

[Anantharaman et al. 1985] Anantharaman, T.S., E.M. Clarke, M.J. Foster and B. Mishra. *Compiling Path Expressions into VLSI Circuits.* In *Proceedings of the ACM Symposium on Principals of Programming Languages,* Association for Computing Machinery. New Orleans, LA: acm, Jan.

1985 pp. 191-204.

[Anderson 1981] Anderson, T. L. *The Database Semantics of Time.* PhD. Diss. University of Washington, Jan. 1981.

[Anderson 1982] Anderson, T. L. *Modeling Time at the Conceptual Level.* In *Improving Database Usability and Responsiveness,* Ed. P. Scheuermann. Jerusalem, Israel: Academic Press, 1982 pp. 273-297.

[Andler 1979] Andler, S. A. *Predicate Path Expressions: A High-level Synchronization Mechanism.* PhD. Diss. Computer Science Department, Carnegie-Mellon University, Aug. 1979.

[Ariav & Morgan 1981] Ariav, G. and H. L. Morgan. *MDM: Handling the time dimension in generalized DBMS.* Technical Report. The Wharton School, University of Pennsylvania. May 1981.

[Ariav & Morgan 1982] Ariav, G. and H. L. Morgan. *MDM: Embedding the Time Dimension in Information Systems.* Technical Report 82-03-01. Department of Decision Sciences, The Wharton School, University of Pennsylvania. 1982.

[Ariav 1984] Ariav, G. *Preserving The Time Dimension In Information Systems.* PhD. Diss. New York University, Apr. 1984.

[Ben-Zvi 1982] Ben-Zvi, J. *The Time Relational Model.* PhD. Diss. UCLA, 1982.

[Berzins & Kapur 1977] Berzins, V. and D. Kapur. *Denotational and Axiomatic Definitions for Path Expressions.* Technical Report 153-1. Laboratory for Computer Science, M.I.T.. Nov. 1977.

[Bjork 1975] Bjork, L. A., Jr *Generalized Audit Trail Requirements and Concepts for Data Base Applications. IBM Systems Journal,* 14, No. 3 (1975) pp. 229-245.

[Blum 1982] Blum, R.L. *Discovery, Confirmation, and Incorporation of Casual Relationships from a Large Time-Oriented Database: The RX Project. Computers and Biomedical Research,* 15, No. 2 (1982) pp. 164-187.

[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.

[Bontempo 1983] Bontempo, C. J. *Feature Analysis of Query-By-Example,* in Relational Database Systems. New York: Springer-Verlag, 1983. pp. 409-433.

[Breutmann et al. 1979] Breutmann, B., E. F. Falkenberg and R. Mauer. *CSL: A language of defining conceptual schemas,* in Data Base Architecture. Amsterdam: North Holland, Inc., 1979.

[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.

[Bubenko 1977] Bubenko, J. A., Jr. *The Temporal Dimension in Information Modeling,* in Architecture and Models in Data Base Management Systems. North-Holland Pub. Co., 1977.

[Bubenko 1980] Bubenko, J. A., Jr. *Information modeling in the context of system development.* In *Proceedings of IFIP Congress 80,* Oct. 1980.

[Ceri & Gottlob 1985] Ceri, S. and G. Gottlob. *Translating SQL Into Relational Algebra: Optimization,* *Semantics, and Equivalence of SQL Queries. IEEE Transactions on Software Engineering,* SE-11, No. 4, Apr. 1985, pp. 324-345.

[Cheeseman 1983] Cheeseman, P. *A Representation of Time for Planning.* Technical Report 278. Artificial Intelligence Center. Feb. 1983.

[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.

[Codd 1972] Codd, E. F. *Relational Completeness of Data Base Sublanguages,* in Data Base Systems. Vol. 6 of Courant Computer Symposia Series. Englewood Cliffs, N.J.: Prentice Hall, 1972. pp. 65-98 .

[Codd 1970] Codd, E.F. *A Relational Model of Data for Large Shared Data Bank. Communications of the Association of Computing Machinery,* 13, No. 6, June 1970, pp. 377-387.

[Codd 1979] Codd, E.F. *Extending the Database Relational Model to Capture More Meaning. ACM Transactions on Database Systems,* 4, No. 4, Dec. 1979, pp. 397-434.

[Copeland & Maier 1984] Copeland, G. and D. Maier. *Making Smalltalk a Database System.* In *Proceedings of the Sigmod '84 Conference,* June 1984 pp. 316-325.

[Dowty 1972] Dowty, D. R. *Studies in the logic of verb aspect and time reference in English.* Technical Report. Dept. of Linguistics, University of Texas at Austin. 1972.

[Fagan 1980] Fagan, L.M. *VM: Representing Time-Dependent Relations in A Medical Setting.* PhD. Diss. Stanford University, June 1980.

[Findler & Chen 1971] Findler, N. and D. Chen. *On the problems of time retrieval, temporal relations, causality, and coexistence.* In *Proceedings of the International Joint Conference on Artificial Intelligence,* Imperial College: Sep. 1971.

[Gadia 1985] Gadia, S.K. *A Homogeneous Relational Model And Query Languages For Temporal Databases.* 1985. (submitted to TODS.)

[Gadia & Vaishnav 1985] Gadia, S.K. and J.H. Vaishnav. *A Query Language For A Homogeneous Temporal Database.* In *Proceedings of the Conference on Principles of Database Systems,* Apr. 1985.

[Habermann 1975] Habermann, A.N. *Path Expressions.* Technical Report. Computer Science Department, Carnegie-Mellon University. June 1975.

[Hammer & McLeod 1981] Hammer, M. and D. McLeod. *Database Description with SDM: A Semantic Database Model. ACM Transactions on Database Systems,* 6, No. 3, Sep. 1981, pp. 351-386.

[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.

[Hirschman & Story 1981] Hirschman, C. and G. Story. *Representing Implicit and Explicit Time Relations in Narrative.* In *Proceedings of the International Joint Conference on Artificial Intelligence,* Vancouver, BC Canada: Aug. 1981 pp. 289-295.

[SQL/DS 1981] IBM *SQL/Data-System, Concepts and Facilities.* Technical Report GH24-5013-0. IBM. Jan. 1981.

[Jayaraman 1982] Jayaraman, B. and R.M. Keller. *Resource Expressions for Applicative Languages.* In *Proceedings of the 1982 International Conference of Parallel Processing,* IEEE. Aug. 1982 pp. 160-167.

[Jayaraman 1983] Jayaraman, B. *Constructing a Parallel Implementation from High-level Specifications: A Case Study using Resource Expressions.* In *Proceedings of the 1983 International Conference of Parallel Processing,* IEEE. Aug. 1983 pp. 416-420.

[Jones et al. 1979] Jones, S., P. Mason and R. Stamper. *LEGOL 2.0: a relational specification language for complex rules. Information Systems,* 4, No. 4, Nov. 1979, pp. 293-305.

[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.

[Kahn & Gorry 1975] Kahn, K. and G.A. Gorry. *Mechanizing Temporal Knowledge. Artificial Intelligence,* 9, Sep. 1975, pp. 87-108.

[Kim 1979] Kim, W. *Relational Database Systems. ACM Computing Surveys,* 11, No. 3, Sep. 1979, pp. 186-211.

[Kim 1982] Kim, W. *On Optimizing an SQL-like Nested Query. ACM Transactions on Database Systems,* 7, No. 3, sept 1982, pp. 443-469.

[Kimball 1978] Kimball, K.A. *The Data System.* University of Pennsylvania, 1978.

[Klopprogge 1981] Klopprogge, M. R. *TERM: An approach to include the time dimension in the entity-relationship model.* In *Proceedings of the Second International Conference on the Entity Relationship Approach,* Oct. 1981.

[Klug 1982] Klug, A. *Equivalence of Relational Algebra and Relational Calculus Query Languages Having Aggregate Functions. Journal of the Association of Computing Machinery,* 29, No. 3, July 1982, pp. 699-717.

[Lauer & Campbell 1975] Lauer, P.E. and R.H. Campbell. *Formal Semantics of a Class of High-Level Primitives for Coordinating Concurrent Processes. Acta Informatica,* 5 (1975) pp. 297-332.

[Long & Russ 1983] Long, W.J. and T.A. Russ. *A control Structure for Time Dependent Reasoning.* In *Proceedings of the International Joint Conference on Artificial Intelligence,* Karlsruhe, West Germany: Aug. 1983 pp. 230-232.

[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.

[McArthur 1976] McArthur, R. P. *Tense Logic.* Dordrecht, Holland: D. Reidel Publishing Co., 1976.

[McCawley 1971] McCawley, J. *Tense and time reference in English.* Holt Reinhardt and Winston, 1971.

[McDermott 1982] McDermott, D. *A Temporal Logic for Reasoning About Processes and Plans. Cognitive Science,* 6, Dec. 1982, pp. 101-155.

[Montague 1973] Montague, R. *The proper treatment of quantification in ordinary English,* in Approaches to Natural Language. Dordrecht, Holland: D. Reidel Publishing Co., 1973.

[Overmyer & Stonebraker 1982] Overmyer, R. and M. Stonebraker. *Implementation of a Time Expert in a Database System. SIGMod Record,* 12, No. 3, Apr. 1982, pp. 51-59.

[Palley et al. 1976] Palley, N. A. et al. *CLINFO User's Guide: release one.* Technical Report R-1543-1-NIH. Rand Corporation. 1976.

[Prior 1967] Prior, A. *Past, present, future.* Oxford University Press, 1967.

[Relational Technology 1984] Relational *Technology MicroINGRES Reference Manual.* 1984.

[Rescher & Urquhart 1971] Rescher, N.C. and A. Urquhart. *Temporal Logic.* New York: Springer-Verlag, 1971.

[Sernadas 1980] Sernadas, A. *Temporal Aspects of Logical Procedure Definition. Information Systems,* 5 (1980) pp. 167-187.

[Shannon 1985] Shannon, K.P. *The Display of Temporal Information.* Computer Science Department, University of North Carolina at Chapel Hill, 1985. In preparation..

[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 & 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.

[Stamper 1976] Stamper, R. K. *The LEGOL project: a survey.* Technical Report 0081. IBM-UKSC. May 1976.

[Stonebraker et al. 1976] Stonebraker, M., E. Wong, P. Kreps and G. Held. *The Design and Implementation of INGRES. ACM Transactions on Database Systems,* 1, No. 3, Sep. 1976, pp. 189-222.

[Stonebraker & Keller 1980] Stonebraker, M. and K. Keller. *Embedding Experts and Hypothetical Data Bases in a Relational Data Base System.* In *Proceedings of the Conference on Management of Data,* ACM-SIGMOD. Santa Monica, CA: May 1980.

[Tandem 1983] Tandem Computers, Inc. *ENFORM Reference Manual.* Cupertino, CA, 1983.

[Taylor & Wheeler 1966] Taylor, E. F. and J. A. Wheeler. *Space-Time Physics.* San Francisco: W. H. Freeman, 1966.

[Todd 1976] Todd, S. J. P. *The Peterlee relational test vehicle--a system overview. IBM Systems Journal,* 15, No. 4 (1976) pp. 285-308.

[Tsotsos 1981] Tsotsos, J.K. *Temporal Event Recognition: An Application to Left Ventricular Performance.* In *Proceedings of the International Joint Conference on Artificial Intelligence,* Vancouver, BC Canada: Aug. 1981 pp. 900-907.

[Ullman 1982] Ullman, J.D. *Principles of Database Systems, Second Edition.* Potomac, Maryland: Computer Science Press, 1982.

[Vilain 1982] Vilain, M.B. *A System for Reasoning About Time.* In *Proceedings of the American Association for Artificial Intelligence,* Pittsburgh: Aug. 1982 pp. 221-226.

[Whitrow 1980] Whitrow, G. J. *The natural philosophy of time.* New York, NY: Oxford University Press, 1980.

[Wiederhold et al. 1975] Wiederhold, G., J. F. Fries and S. Weyl. *Structured organization of clinical data bases.* In *Proceedings of the National Computer Conference,* AFIPS. 1975.

[Zaniolo 1983] Zaniolo, C. *The Database Language GEM.* In *Proceedings of the 1983 International Conference on Management of Data,* ACM SIGMod. San Jose, CA: May 1983 pp. 207-218.

10. Appendix: Syntax of the Augmented TQuel Statements

This appendix lists the syntax for the statements where Quel and TQuel differ. Since TQuel is a strict superset of Quel, all legal Quel statements are also legal TQuel statements. TQuel augments five Quel statements: create, retrieve, append, delete, and replace. The Quel statements left unaltered are copy (data into/from a relation from/into a Unix file), define (subschema: view, permissions, or integrity constraints), destroy (a relation), help, index, modify (the storage structure of a relation), print, range, and save (a relation until a date). The following non-terminals are not included in the syntax description because they are identical to their Quel counterparts.

<bool expression>	returns a value of type boolean
<expression>	returns a value of type integer, string, floating point, or temporal
<domain>	the name of a domain
<relation>	a relation name
<string>	a string constant
<tuple variable>	the name of a tuple variable
<domain specs>	a list of the names and types for the user specified domains

Also not shown are the additional temporal functions and predefined relations found in TQuel.

```
<TQuel augmented>     ::= <create stmt>
                        | <retrieve stmt>
                        | <append stmt>
                        | <delete stmt>
                        | <replace stmt>

<create stmt>         ::= create <persistent> <history> <domain specs>

<persistent>          ::= ε | persistent

<history>             ::= ε | interval | event
```

```
<retrieve stmt>      ::= <retrieve head> <retrieve tail>

<retrieve head>      ::= retrieve <into> <target list> <valid clause>

<retrieve tail>      ::= <where clause> <when clause> <as of clause>

<into>               ::= ϵ | unique | <relation> | into <relation> | to <relation>

<target list>        ::= ϵ | ( <tuple variable> . all ) | ( <t-list> )

<t-list>             ::= <t-elem> | <t-list> , <t-elem>

<t-elem>             ::= <domain> <is> <expression>

<is>                 ::= is | = | by

<append stmt>        ::= append <to> <target list> <mod stmt tail>

<to>                 ::= <relation> | to <relation>

<delete stmt>        ::= delete <tuple variable> <mod stmt tail>

<replace stmt>       ::= replace <tuple variable> <target list> <mod stmt tail>

<mod stmt tail>      ::= <valid clause> <where clause> <when clause>

<valid clause>       ::= <valid> <from clause> <to clause> | <valid> <at clause>

<valid>              ::= ϵ | valid

<from clause>        ::= ϵ | from <e-expression>

<to clause>          ::= ϵ | to <e-expression>

<at clause>          ::= at <e-expression>

<where clause>       ::= ϵ | where <bool expression>

<when clause>        ::= ϵ | when <temporal pred>

<as-of clause>       ::= ϵ | as of <e-expression> <through clause>

<through clause>     ::= ϵ | through <e-expression>

<e-expression>       ::= <event element>
                     | start of <e-either>
                     | end of <e-either>
                     | ( <e-expression> )

<e-interval>         ::= <interval element>
                     | <e-either> overlap <e-either>
                     | <e-either> extend <e-either>
                     | ( <e-interval> )
```

```
<e-either>            ::= <e-expression> ¦ <e-interval>

<event element>       ::= <tuple variable>   associated with an event relation

<interval element>    ::= <tuple variable>   associated with an interval relation
                       ¦ <temporal constant>

<temporal constant>   ::= <string>

<temporal pred>       ::= <interval element>
                       ¦ <event element>
                       ¦ <temporal pred> precede <temporal pred>
                       ¦ <temporal pred> overlap <temporal pred>
                       ¦ <temporal pred> extend <temporal pred>
                       ¦ <temporal pred> and <temporal pred>
                       ¦ <temporal pred> or <temporal pred>
                       ¦ ( <temporal pred> )
                       ¦ start of <temporal pred>
                       ¦ end of <temporal pred>
                       ¦ not <temporal pred>
```

Event elements are tuple variables associated with event relations. Interval elements are either tuple variables associated with interval relations, or are temporal constants (all temporal constants are intervals).

The where, when, and valid clauses in the delete statement can only refer to one tuple variable, that referenced at the beginning of the statement. The unary operators (**start of**, **end of**, **not**) have the highest precedence, followed in order by the binary temporal constructors (**extend**, **overlap**), the temporal predicate operators (**precede**, **overlap**), and finally the binary logical operators (**and**, **or**). Operators of equal precedence are left associative. **extend** and both variants of **overlap** are commutative; **precede** is not.

Note that the distinction between <interval element> and <event element> makes the grammar context-sensitive. In practice, this distinction is ignored in the LALR parser, and the resulting parse tree is type-checked in the semantic analysis phase. Similarly, checking that the top level operator of a temporal predicate, after the logical operators have been distributed over the temporal ones, must be **precede** or **overlap** is relegated to the semantic analysis phase (this is the reason that no distinction is made in the BNF between temporal predicates and e-expressions).

In keeping with the path expression origins of temporal predicates and e-expressions, the keyword **overlap** may be abbreviated with a comma, **precede** may be abbreviated with a semicolon, and **or** may be abbreviated with a vertical bar. Since non-temporal attributes are designated by "<tuple-variable> . <domain>", the prefix unary operators **start of** and **end of** may be preplaced by the postfix operators "**.from**" and "**.to**". The following is an example.

```
range of f1 is Faculty
range of f2 is Faculty
range of a is Associates
retrieve into Stars (Name = f1.Name)
        valid from f1.start to f2.start
        where f1.Name = f2.Name and f1.Rank = "Assistant" and f2.Rank = "Full"
        when (f1 and f2) , a
```

Example 17: Another Variant of Example 7.