The Temporal Query Language TQuel

RICHARD SNODGRASS
University of North Carolina

Recently, attention has been focused 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 minimal extension, 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. TQuel is compared with ten other query languages supporting time.

Categories and Subject Descriptors: H.2.1 [**Database Management**]: Logical Design—*data models*; H.2.3 [**Database Management**]: Languages—*query languages*; H.2.7 [**Database Management**]: Database Administration—*logging and recovery*

General Terms: Languages, Theory

Additional Key Words and Phrases: Historical database, Quel, relational calculus, relational model, rollback database, temporal database, tuple calculus

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continue to change as new information is added, these changes are viewed as modifications to the state, with the old, out-of-date data being deleted from the database.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 focused on *temporal databases* (*TDBs*), 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. TDBs are thus generalizations of conventional (termed *snapshot*) databases and their underlying snapshot relational model.

We have developed a new language, *TQuel* (*T*emporal *QUE*ry *L*anguage), to query a TDB [75]. TQuel is a derivative of Quel [41], the query language for the

This research was sponsored by the National Science Foundation under Grant DCR-8402339, and was supported by an IBM Faculty Development Award and by a UNC-CH Foundation Junior Faculty Development Award.

Author's address: Department of Computer Science, University of North Carolina, Chapel Hill, NC 27514.

Ingres relational database management system [81]. 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 conventional 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 concept of TDBs is introduced in Section 2, and an overview of the language is provided in Section 3. A formal definition, semantics, and the prototype implementation of TQuel are the subjects of Sections 4–6, respectively. The final section summarizes the results, compares TQuel to other query languages, and indicates future work. The 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 [82]; the Query-by-Example system supports both date and time domains directly [18]; and Ingres has been extended to convert dates to and from an internal format and to perform comparisons and arithmetic operations on these domains [66, 70]. However, none of these systems interprets temporal domains when deriving new relations.

The need to handle time more comprehensively surfaced in the early 1970s in the area of medical information systems, where a patient's medical history is particularly important. The model supported by the Time Oriented Databank [93] and several other medical DBMSs (e.g., CLINFO [67]) 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. A recent, extensive bibliography [61] contained 80 articles from 1982 to 1986. At least 25 research groups are studying time in databases [76]. 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 snapshot 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., [4–6, 24, 34, 48, 55, 60, 88, 91]), linguistics (c.f., [33, 42, 59, 64]), logic (c.f., [58, 68, 71]), philosophy (c.f., [92]), and physics (c.f., [87]).

Bubenko [20, 21] 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 [8, 9, 19, 22, 30, 40, 52].

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 snapshot model, with time appearing as additional attribute(s). The first has been applied successfully by Clifford and Warren [27], with the entity-relationship model used to formulate the intensional logic 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 snapshot relational model. Sernadas has taken the same approach in defining the temporal process specification language DMTLT, which incorporates a special modal tense logic [72].

In the second approach, the snapshot relational database model [28] serves as the underlying model of the TDB. Each temporal relation is *embedded* in a snapshot relation containing an 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 snapshot 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 [15]. Other researchers have also utilized this technique [13, 32, 38, 47].

Several query languages incorporating time have been designed over the last decade. In Section 7.1 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 [77], and a comprehensive example may be found in [78]. The presentation is more of an intuitive nature than a formal characterization of TDBs; Section 5.1 will show how to embed a temporal relation in a snapshot relation, thereby providing a precise definition. We take the second approach to modeling TDBs: utilizing the snapshot model.

2.1 Snapshot 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 *snapshot database*.

In the snapshot relational model, a database is a collection of *relations*. Each relation consists of a set of *tuples* with the same set of *attributes* and is usually represented as a two-dimensional table (see Figure 1). As changes occur in the real world, changes are made in this table.

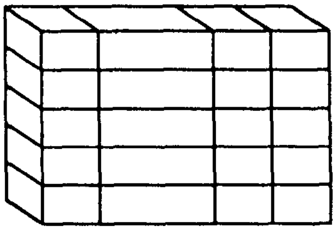
Fig. 1. A snapshot relation.

2.2 Rollback Databases

Snapshot 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 snapshot 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 snapshot relations (termed *snapshot states*) indexed by transaction time. One can get a snapshot of the relation as of some time in the past (a snapshot state) and make queries on that state by moving along the time axis and selecting this relation. The operation of selecting a snapshot state is termed *rollback*, and a database supporting it is termed a *rollback database*. A rollback to a time t, where t is between two transaction times t_1 and t_2 represented in a rollback database, selects the most recent snapshot state in effect at that time (i.e., the one at t_1). Changes to a rollback database may only be made to the most recent snapshot 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 snapshot state being appended to the right; once a transaction has completed, the snapshot states in the rollback relation may not be altered. Transaction time is represented by *transaction identifiers*: monotonically increasing integers generated by the DBMS. We assume that the DBMS maintains information mapping transaction identifiers into the clock time when the transaction was executed, for querying and display purposes.

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 snapshot database. Retroactive/proactive 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.

Whereas rollback databases record a sequence of snapshot states, *historical databases* record a single *historical state* per relation, storing the history as is best

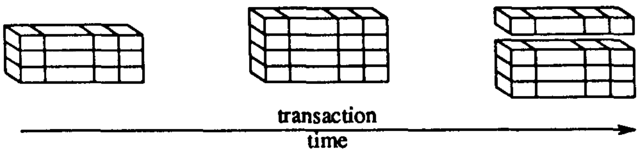

Fig. 2. A rollback relation.

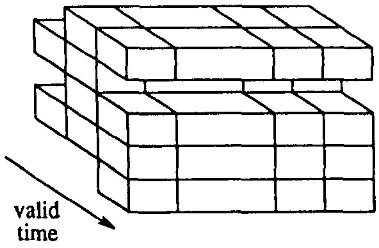

Fig. 3. An historical relation.

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 record is kept of the errors that have been corrected; historical databases are similar to snapshot 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) [12, 15, 27, 56]. 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 modeled remains unchanged between the individual snapshot slices found in the historical relation; this is termed the *step function continuity assumption* [27] or the *principle of temporal density* [11]. The information present in the snapshot database slice at one valid time v_1 is assumed to be valid for all time 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, while *events*, occurring instantaneously, are more difficult to model. Our representation of an event is a tuple that exists for exactly one valid time, with the snapshot slices of the previous and next valid times not containing the tuple. 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 *valid-time granularity* of the encoding (say, seconds, or microseconds). In effect, we are defining *instantaneous* to be any occurrence over a time interval that is less than the valid-time granularity. In the examples that appear later in the paper, the valid-time granularity is one month: Any occurrence over a period of less than a month is considered instantaneous. Snapshot relations as defined in Section 2.1 cannot represent events at all, precisely because they are instantaneous.

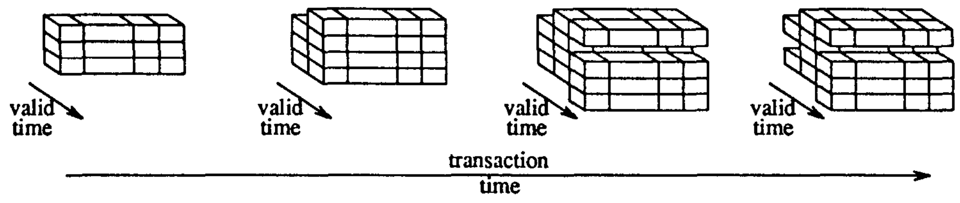

Fig. 4. A temporal relation.

Since an update to a 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 snapshot slice.

2.4 Temporal Databases

Benefits of both approaches can be combined by supporting both transaction time and valid time. Whereas a 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/proactive 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 a 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 a 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.

User-defined time [46] is necessary when additional temporal information, not handled by transaction or valid time, is stored in the database. The values of user-defined temporal attributes are not interpreted by the DBMS and are thus the easiest to support; all that is needed is an internal representation and input and output functions. The transaction and valid times are needed in any case in temporal relations.

In this model four types of databases were defined: snapshot, rollback, historical, and temporal. Each may be associated with a class of query languages. A *snapshot query language* supports queries over multiple snapshot relations. A *rollback query language* also supports rollback. A *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 [41], the query language for Ingres [81]. 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 [43], is more complex and has a rather complicated semantics [23, 50]. 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, we assume the reader is familiar with Quel.

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 following query produces the relation shown in Figure 6 when applied to the sample database:

Example 1. List the associate professors.

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

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 snapshot relations. Although 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 snapshot 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*). Although we will argue that events and intervals are quite similar semantically, there are compelling arguments for presenting both to the user [57]. The second temporal

Faculty (Name, Rank):

Name	Rank
Jane	Full
Merrie	Associate
Tom	Associate

Fig. 5. A snapshot database.

Submitted (Author, Journal):

Author	Journal
Jane	*CACM*
Merrie	*CACM*
Merrie	*TODS*
Tom	*JACM*

Associates (Name):

Name
Merrie
Tom

Fig. 6. Results of a query on a snapshot database.

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 are 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 valid-time values is arbitrary; in this section we assume for simplicity a granularity of one month. We also assume that the DBMS has been instructed to display the transaction time to the nearest month, again for simplicity. Tuples are assumed to be *coalesced*, in that tuples with identical values for the explicit attributes (termed *value-equivalent tuples* [62]) neither overlap nor are adjacent in time.

Since TQuel is a strict superset of Quel, the identical query, executed in September, 1983 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.

Since the additional temporal attributes are an artifact of embedding a temporal relation in a snapshot one, users must be constrained in how 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 is 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

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	∞

Fig. 7. A temporal database.

Associates (Name):

Name	Valid time (from)	Valid time (to)	Transaction time (start)	Transaction time (stop)
Jane	12-76	11-80	9-83	∞
Merrie	12-82	∞	9-83	∞
Tom	12-80	∞	9-83	∞

Fig. 8. The same query on a temporal database.

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 [10, 39].

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

Example 2. List the associate professors in September.

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

In this case the query specifies that the interval when the faculty member was an associate professor should include September (of the current year), which is also a time interval (strings, enclosed in double quotation marks, are temporal constants). We mention in passing that the **overlap** operator is also used in another context, as illustrated in Section 3.2 and discussed more deeply in Section 4.3. The result is Merrie and Tom.

Another example of the when clause follows:

Example 3: What papers were written by associate professors?

```
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
```

The time that the paper was submitted must overlap with the time interval when the faculty member was an associate professor. Jane submitted a paper to *CACM*, and Tom submitted a paper to *JACM*.

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 **begin of** and **end of**:

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

```
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 begin of a
```

This query returns Jane.

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

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

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

This example also illustrates the **and** operator; the **or** and **not** operators are allowed as well. Fortunately there are no disgruntled professors.

3.2 The Valid Clause

The *valid clause* serves the same purpose as the target list: specifying the value of an 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.

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

```
range of a is Associates
retrieve into AssociatePromotions (Name = a.Name)
  valid at begin of a
```

Jane was promoted on 12-76, Merrie on 12-82, and Tom on 12-80. 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 **begin 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, **valid from . . . to . . . ,** also contains e-expressions and is used when the derived relation is to be an interval relation:

Example 7. Who got promoted from assistant to full professor while at least one other faculty remained at the associate rank?

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

Tuples in the derived relation Stars indicate the interval of time from joining the faculty as assistant professors to becoming full professors. There are currently no 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 begin of (f1 **overlap** a)

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

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:

Example 8. What stars were known at the end of 1984?

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

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	∞

Fig. 9. An updated temporal 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 temporal DBMS; only the internal representation and the input and output functions 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 temporal DBMS as the current time (recall that temporal databases are append only; hence the modification applies to the current historical state). 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.

Example 9. Tom submitted a paper to *TOPLAS*, not to *JACM*.

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

This results in the relation shown in Figure 9, which should be compared with Figure 7.

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 14 statements; TQuel augments 5 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 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 a historical or temporal relation, and the when and valid clauses may be used. If none of these keywords is used, the relation is a conventional snapshot relation. The four types of relations (snapshot, 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 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 Functions

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 "September 1, 1983" denotes an interval from midnight of September 1, 1983, to midnight of September 2, 1983; "September 1983" denotes the entire month; and "4:00 PM September 1, 1983" denotes a 60-second interval. Events may be approximated with very short intervals. The exact format of these constants is similar to that specified for the time expert [66] or the Ingres system [70]. The constants "beginning," "now," and "forever" are also available, with both transaction and valid time values, distinguished by the context in which they are used.

The implicit temporal attributes are available through the functions "validat," "*validfrom*," and "*validto*" (valid time), and "*transactionstart*" and "*transactionstop*" (transaction time), for use only in the target list and where clauses. These functions, as well as the temporal data type, are provided in part for auditing purposes [17]; a simple example is as follows:

Example 10. When was Tom entered incorrectly as an associate professor?

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

Tom was entered incorrectly on 8-75, and his rank was corrected on 10-75. The MistakeDate and CorrectedDate attributes cannot be used in subsequent when, valid, or as-of-clauses; to the temporal DBMS, 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 [15].

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 that starts and ends at the event's time value. The unary prefix temporal constructors are **begin of** and **end of**, both returning events. The binary infix temporal constructors are **overlap** and **extend**, both returning intevals. **overlap** is undefined if there are no time values that are in both underlying intervals. The **overlap** operator may be thought of as a temporal *intersection* operator, in that it returns the points in time when *both* arguments are valid: 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 the points in time when *either* of the arguments are valid; the predicate

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

is true when the ends of both a and b precede the start of c. The difference between **overlap** and **extend** is illustrated with the time lines in Figure 10.

An *e-expression* is simply an expression containing tuple variables, temporal constants, and temporal constructors, with the constraint that the expression

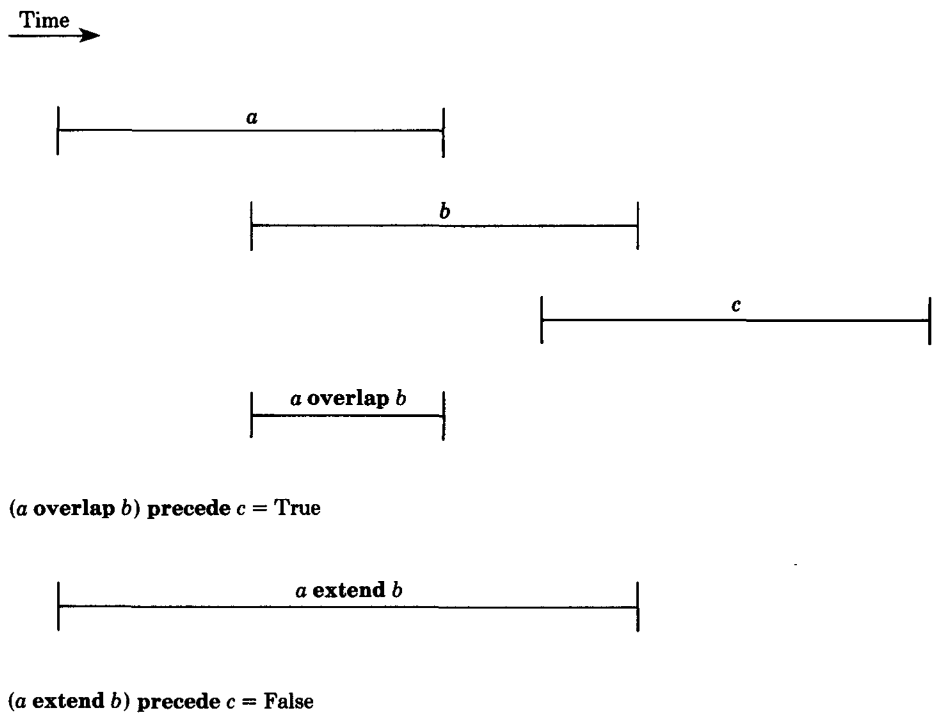
Time
a
b
c
a overlap b
(a overlap b) precede c = True
a extend b
(a extend b) precede c = False

Fig. 10. The difference between **overlap** and **extend.**

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.

An *i-expression* is an expression containing tuple variables, temporal constants, and temporal constructors that evaluates to an interval.

A *temporal predicate operator* is a binary infix operator that takes events or intervals as arguments and returns a Boolean value. The three temporal predicate operators are **precede, overlap,** and **equal.** The **overlap** operator is an overloaded operator, in that it is both a temporal constructor and a temporal predicate 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 (**begin 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 (**begin of** α) is before the event (**end of** β) and the event (**begin of** β) is before the event (**end of** α). An equivalent formulation is (**end of** (**begin of** α **extend begin of** β)) **precede** (**begin of** (**end of** α **extend end of** β)). α **equal** β is true if α and β are two events that occurred at the same time (within the valid-time granularity) or if α

and β are two intervals that began and ended at the same time. An event is never **equal** to an interval.

A *temporal predicate* is an expression containing logical operators (**and**, **or**, **not**) operating on expressions containing a temporal predicate operator (**precede**, **overlap**, or **equal**), operating on e-expressions and i-expressions. This constraint is motivated by consideration of the types in a temporal predicate. In particular, e-expressions evaluate to events, and i-expressions evaluate to intervals. A temporal predicate operator maps pairs of intervals or events to a Boolean value, which may be operated on by the logical operators. Temporal predicates are used only in when clauses.

We envision that additional temporal constructors and temporal predicates would be supported in an implementation.

4.4 Unique

Resulting relations are always coalesced (cf., Section 3) when they are stored. This behavior is analogous to Ingres removing duplicates when storing relations. If the retrieve statement does not name a destination relation, the tuples are not coalesced, and duplicates are not removed, for performance reasons. The user can insist on coalescing and duplicate elimination by specifying **retrieve unique**.

4.5 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. A retrieve statement always generates a new historical state, unless all the underlying relations are snapshot or rollback relations, in which case a new snapshot state is generated.

4.6 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 as follows:

valid from begin of *I* **to end of** *I*
  when *I* **overlap** "now"
  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. The valid from and valid to defaults are distinct; one can be stated explicitly by the user, and the other will be supplied as a default. 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:

Example 11. The first query, with defaults.

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

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 $t_1, t_2, \ldots, t_k$, then the default temporal clauses are the following:

valid from begin of (t_1 **overlap** $\cdots$ **overlap** t_k) **to end of** (t_1 **overlap** $\cdots$ **overlap** t_k)
  when (t_1 **overlap** $\cdots$ **overlap** t_k) **overlap** "now"
  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. Tuple variables associated with event relations are ignored in this case.

For the append statement, the defaults are as follows:

valid from "now" **to** "forever"
  when (t_1 **overlap** $\cdots$ **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. Again, tuple variables associated with event relations are ignored. For the delete statement, the defaults are as follows:

delete t_0
  valid from "now" **to end of** t_0
  when (t_0 **overlap** t_1 **overlap** $\cdots$ **overlap** t_k) **overlap** "now"

The tuple variables $t_1 \cdots t_k$ are from the where, when, and valid clauses. These defaults imply that the deletion only applies to information valid now or in the future. If t_0 was associated with an event relation, the default is as follows:

delete t_0
  valid at t_0
  when (t_1 **overlap** $\cdots$ **overlap** t_k) **overlap** "now"

And, finally, for the replace statement, the defaults are as follows:

replace t_0
  valid from "now" **to end of** t_0
  when (t_0 **overlap** $\cdots$ **overlap** t_k) **overlap** "now"

These defaults follow from the fact that a replace is roughly equivalent to a delete followed by an append. The situation where t_0 is associated with an event relation is handled similarly to the delete statement.

Note that when one or more of these clauses are not provided by the user, it is assumed to be as discussed above. The user should be careful when only a few clauses are defaulted, because the defaulted clause(s) may be inappropriate.

5. FORMAL SEMANTICS

TQuel statements manipulate information in a TDB composed of a sequence of historical states indexed by transaction time, with each historical state consisting of a sequence of snapshot slices 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 snapshot 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 snapshot 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 well-understood 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 TDB based on the relational model can be implemented directly on conventional relational DBMSs. Many of the same advantages resulted from a similar approach in the design of GEM, a query and update language for a (snapshot) semantic data model [94], and in the specification of the semantics of the snapshot query language SQL [23].

5.1 Embedding a Temporal Relation in a Snapshot Relation

The snapshot relational database model is utilized as the underlying model of the TDB by embedding the four-dimensional temporal relation in a two-dimensional snapshot relation. The semantics of operations on four-dimensional temporal relations will be specified by stating their effect on the two-dimensional snapshot 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 Figure 11 the tuples comprising a historical state at a particular transaction time are separated by horizontal lines, and the tuples comprising a snapshot slice at a particular valid time are separated by dots. The snapshot relation in Figure 11 contains a temporal relation consisting of five historical states (each associated with a unique transaction time), each consisting of snapshot slices (each associated with a unique valid time). The last historical state, with a transaction time value of 8–77, consists of four snapshot slices, totaling 8 tuples. That each transaction creates a copy of the most recent

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

Fig. 11. Embedding a temporal relation, version 1.

historical state, mentioned in Section 2.4, can be seen clearly in this representation. The full embedding of Figure 7 would contain eight historical states, since the temporal relation was the result of eight transactions. The last historical state would contain seven snapshot slices and a total of 30 tuples. The entire snapshot relation embedding the temporal relation in Figure 7 would contain 102 tuples! The historical relations of Clifford and Warren are similar to this embedding [27].

Another way to embed a temporal relation in a snapshot relation is to append two attributes, each containing *two* time values, denoting *intervals* of valid and transaction time. Temporal relations in this version were illustrated in Figures 7 and 8. Such a representation was proposed by Ariav in his Temporally Oriented Data Management System [12]. Still a third way is to add a total of five additional attributes: 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

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	—	—

Fig. 12. Embedding a temporal relation, version 3.

Model [15]. 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 (if Tom leaves the department, one tuple would have to be added to Figure 7, whereas only T_d of one existing tuple would have to be changed in Figure 12). The effective time in the time relation model (TRM) is equivalent to valid time in our model; the three registration and deletion times encode the same information as our two transaction times.

A fourth way to embed a temporal relation in a snapshot relation is to associate time values with the attributes themselves [35, 38]. 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 snapshot relation is no longer in first normal form.

Finally, the most space efficient representation was proposed by Kimball in the DATA system; only the transactions are recorded [51]. Valid time was not considered, but may be added as another attribute (see Figure 14). Determining the tuples valid at a particular time as of another time involves replaying the transctions in order from the beginning (optimizations are of course possible). Updates, on the other hand, are easy to formalize and implement using this representation.

We have chosen the second representation, with each tuple containing four additional time values, upon which to base our semantics. We assume that relations are coalesced, as defined in Section 3. The advantages of this representation include ease of formal manipulation and the promise of rapidly prototyping a temporal DBMS on top of a conventional snapshot DBMS. We emphasize, however, that an equivalent semantics could be generated for the other representations. The semantics of TQuel originates from the model of TDBs developed in Section 2, not from any particular representation.

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

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, ∞)

Fig. 13. Embedding a temporal relation, version 4.

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

Fig. 14. Embedding a temporal relation, version 5.

aggregates [89], and Klug has treated aggregates in the more general case [53]. 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

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

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
. . .
range of t_k **is** R_k
retrieve $(t_{i_1}.D_{j_1}, \ldots, t_{i_r}.D_{j_r})$
  where ψ

is

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

which states that each t_i is in R_i, that each result tuple u is composed of r particular components, that the mth attribute of u is equal to the j_mth attribute (having an attribute name of D_{j_m}) of the tuple variable t_{i_m}, and that the condition ψ' (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 attribute names for the derived relation must be provided in the target list, and attribute values may be expressions. We ignore such details for the remainder of this paper.

The semantics of a query on a TDB will be specified by providing a tuple calculus statement that denotes a snapshot relation embedding a temporal relation that 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 are 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 [53]. This method has been used to express the semantics of SQL statements [23]. 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 [26], [35], and [62].

The next subsection will provide the semantics of e-expressions as functions 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. Similarly, an i-expression selects two time values from those given. 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 [16, 45], or transformations into Petri nets [54], parallel programs [10, 44], or even VLSI circuits [7]. 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 that is slightly more restrictive, yet has several nice properties [9].

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

$$
\begin{aligned}
\mathit{First}(\alpha, \beta) &= \begin{cases} \alpha & \text{if } \mathit{Before}(\alpha, \beta) \\ \beta & \text{otherwise} \end{cases} \\
\mathit{Last}(\alpha, \beta) &= \begin{cases} \beta & \text{if } \mathit{Before}(\alpha, \beta) \\ \alpha & \text{otherwise} \end{cases} \\
\mathit{event}(t) &= \langle t_{at}, t_{at} \rangle \\
\mathit{interval}(t) &= \langle t_{from}, t_{to} \rangle \\
\mathit{beginof}(\langle \alpha, \beta \rangle) &= \langle \alpha, \alpha \rangle \\
\mathit{endof}(\langle \alpha, \beta \rangle) &= \langle \beta, \beta \rangle \\
\mathit{overlap}(\langle \alpha, \beta \rangle, \langle \gamma, \delta \rangle) &= \langle \mathit{Last}(\alpha, \gamma), \mathit{First}(\beta, \delta) \rangle \\
\mathit{extend}(\langle \alpha, \beta \rangle, \langle \gamma, \delta \rangle) &= \langle \mathit{First}(\alpha, \gamma), \mathit{Last}(\beta, \delta) \rangle
\end{aligned}
$$

A few comments are in order. First, if the e-expression is a correct one, that is, 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. Second, as mentioned in Section 4.3, the *Before* predicate is the "≤" 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 [23]. And, finally, the definition of the *overlap* function assumes that the intervals do indeed overlap; if this constraint is satisfied, then the ordered pairs $\langle \alpha, \beta \rangle$ generated by these functions will always represent intervals, that is, the ordered pairs will satisfy $\mathit{Before}(\alpha, \beta)$. 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

begin of (a **overlap** b)

is transformed into

beginof(*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 the following:

$\rightarrow$ *beginof*(*overlap*($\langle a_{from}, a_{to}\rangle$, $\langle b_{from}, b_{to}\rangle$))
$\rightarrow$ *beginof*($\langle$*Last*(a_{from}, b_{from}), First(a_{to}, b_{to})$\rangle$)
$\rightarrow$ $\langle$*Last*(a_{from}, b_{from}), *Last*(a_{from}, b_{from})$\rangle$

Hence the denotation of this expression is *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 attributes of the tuples in the underlying relations. This predicate is generated in three steps: First, the tuple variables and the temporal constructors are replaced by the functions defined in the previous subsection. Second, the **and**, **or**, and **not** operators are replaced by the logical predicates. Finally, the temporal predicate operators are replaced by analogous predicates on ordered pairs of integers as follows:

$$\begin{aligned}
precede(\langle\alpha,\beta\rangle,\langle\gamma,\delta\rangle) &= Before(\beta,\gamma)\\
overlap(\langle\alpha,\beta\rangle,\langle\gamma,\delta\rangle) &= Before(\alpha,\delta)\wedge Before(\gamma,\beta)\\
equal(\langle\alpha,\beta\rangle,\langle\gamma,\delta\rangle) &= Before(\alpha,\gamma)\wedge Before(\gamma,\alpha)\wedge Before(\beta,\delta)\\
&\quad \wedge Before(\delta,\beta)
\end{aligned}$$

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

As an example, applying the first step to the temporal predicate

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

results in

$\rightarrow$ (*beginof*(*overlap*(*interval*(a), *interval*(b))) **precede** *interval*(c))
  or (*interval*(c) **precede** *interval*(a))
$\rightarrow$ (*beginof*(*overlap*($\langle a_{from}, a_{to}\rangle$, $\langle b_{from}, b_{to}\rangle$)) **precede** $\langle c_{from}, c_{to}\rangle$)
  or ($\langle c_{from}, c_{to}\rangle$ **precede** $\langle a_{from}, a_{to}\rangle$)
$\rightarrow$ (*beginof*($\langle$*Last*(a_{from}, b_{from}), *First*(a_{to}, b_{to})$\rangle$) **precede** $\langle c_{from}, c_{to}\rangle$)
  or ($\langle c_{from}, c_{to}\rangle$ **precede** $\langle a_{from}, a_{to}\rangle$)
$\rightarrow$ ($\langle$*Last*(a_{from}, b_{from}), *Last*(a_{from}, b_{from})$\rangle$ **precede** $\langle c_{from}, c_{to}\rangle$)
  or ($\langle c_{from}, c_{to}\rangle$ **precede** $\langle a_{from}, a_{to}\rangle$).

The second step results in

$(\langle \mathit{Last}(\mathrm{a}_{from}, \mathrm{b}_{from}), \mathit{Last}(\mathrm{a}_{from}, \mathrm{b}_{from})\rangle$ **precede** $\langle \mathrm{c}_{from}, \mathrm{c}_{to}\rangle)$
$\vee\ (\langle \mathrm{c}_{from}, \mathrm{c}_{to}\rangle$ **precede** $\langle \mathrm{a}_{from}, \mathrm{a}_{to}\rangle)$,

and third step results in

$\mathit{Before}(\mathit{Last}(\mathrm{a}_{from}, \mathrm{b}_{from}), \mathrm{c}_{from}) \vee \mathit{Before}(\mathrm{c}_{to}, \mathrm{a}_{from})$.

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 transaction identifiers, and the temporal constants (strings) can be replaced by their corresponding ordered pairs of transaction identifiers. The result can be evaluated at "compile time," resulting in a single transaction identifier, for the **as of** variant, and two transaction identifiers in the **as of through** variant. For convenience, these times will be converted into an interval by interpreting **through** as **extend**.

as of begin of "1984" **through** "October 1984"

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

$\mathit{extend}(\mathit{beginof}(\langle 1009, 1021\rangle), \langle 1018, 1019\rangle)$
$\rightarrow \mathit{extend}(\langle 1009, 1009\rangle, \langle 1018, 1019\rangle)$
$\rightarrow \langle \mathit{First}(\langle 1009, 1018\rangle), \mathit{Last}(\langle 1009, 1019\rangle)\rangle$
$\rightarrow \langle 1009, 1019\rangle$

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 Γ_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 Γ_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 $(t_{i_1}.D_{j_1}, \ldots, t_{i_r}.D_{j_r})$
  valid from ν to χ
  where ψ
  when τ
  as of α **through** β

the tuple calculus statement has the following form:

$$
\begin{aligned}
\{u^{(r+4)} \mid\ & (\exists t_1) \cdots (\exists t_k)(R_1(t_1) \wedge \cdots \wedge R_k(t_k) \\
& \wedge u[1] = t_{i_1}[j_1] \wedge \cdots \wedge u[r] = t_{i_r}[j_r] \\
& \wedge u[r+1] = \Phi_v \wedge u[r+2] = \Phi_\chi \wedge \mathit{Before}(u[r+1], u[r+2]) \\
& \wedge u[r+3] = \mathit{current\ transaction\ id} \wedge u[r+4] = \infty \\
& \wedge \psi' \\
& \wedge \Gamma_\tau \\
& \wedge (\forall l)(1 \leq l \leq k.(\mathit{Before}(\Phi_\alpha, t_l[\mathit{stop}]) \wedge \mathit{Before}(t_l[\mathit{start}], \Phi_\beta))) \\
&)\}
\end{aligned}
$$

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 consists of r explicit attributes and four implicit attributes (*from*, *to*, *start*, and *stop*). The second line, also from the Quel semantics, states the origin of the values in the explicit attributes 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. "*current transaction id*" is replaced with the integer corresponding to the current transaction; this integer must be monotonically increasing. The transaction time is calculated by the concurrency control mechanism. "∞" 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_v, Φ_χ, ψ', and Γ_τ are functions over the *from*, *to*, and explicit attributes 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 attributes and 4 implicit attributes, the starting and stopping time values for the valid and transaction intervals; events will have only 3 implicit attributes. 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.

The resulting relation is not required by this semantics to be coalesced, although a coalesced result relation is one of the acceptable solutions. Minor changes to the semantics are necessary when changing the type of the relation, for example, deriving a historical relation from a snapshot relation. This is a simple case of *schema evolution*, which is discussed elsewhere [63].

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 8. We assume this retrieve statement was executed on January 1, 1985, yielding a transaction identifier of 12345.

$$
\begin{aligned}
\{u^{(1+4)} \mid\ & (\exists \mathrm{f1})(\exists \mathrm{f2})(\exists \mathrm{a})(\mathrm{Faculty(f1)} \wedge \mathrm{Faculty(f2)} \wedge \mathrm{Associates(a)} \\
& \wedge u[1] = \mathrm{f1}[1] \\
& \wedge u[2] = \mathrm{f1}[3] \wedge u[3] = \mathrm{f2}[3] \wedge \mathit{Before}(u[2], u[3]) \\
& \wedge u[4] = 12345 \wedge u[5] = 0 \\
& \wedge \mathrm{f1}[1] = \mathrm{f2}[1] \wedge \mathrm{f1}[2] = \text{“Assistant”} \wedge \mathrm{f2}[2] = \text{“Full”} \\
& \wedge \mathit{Before}(\mathrm{f1}[3], \mathrm{a}[3]) \wedge \mathit{Before}(\mathrm{a}[2], \mathrm{f1}[4]) \wedge \mathit{Before}(\mathrm{f2}[3], \mathrm{a}[3]) \\
& \qquad \wedge \mathit{Before}(\mathrm{a}[2], \mathrm{f2}[4]) \\
& \wedge \mathit{Before}(1020, \mathrm{f1}[6]) \wedge (\mathit{Before}(\mathrm{f1}[5]), 1020) \\
& \qquad \wedge \mathit{Before}(1020, \mathrm{f2}[6]) \wedge (\mathit{Before}(\mathrm{f2}[5], 1020) \\
& \qquad \wedge \mathit{Before}(1020, \mathrm{a}[5]) \wedge (\mathit{Before}(\mathrm{a}[4], 1020) \\
&)\}
\end{aligned}
$$

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

Example 12. A Contrived Example

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

This query references relations containing the following attributes:

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

The implicit temporal attributes are in parentheses (A, B, and C are all interval relations). The query then has the following semantics (note that the current transaction identifier has been incremented by one):

$$
\begin{aligned}
\{u^{(3+4)} \mid\ & (\exists \mathrm{a})(\exists \mathrm{b})(\exists \mathrm{c})(\mathrm{A(a)} \wedge \mathrm{B(b)} \wedge \mathrm{C(c)} \\
& \wedge u[1] = \mathrm{a}[1] \wedge u[2] = \mathrm{b}[1] \wedge u[3] = \mathrm{c}[1] \\
& \wedge u[4] = \mathit{Last}(\mathrm{a}[3], \mathrm{b}[3]) \wedge u[5] = \mathit{First}(\mathrm{a}[4], \mathrm{b}[4]) \wedge \mathit{Before}(u[4], u[5]) \\
& \wedge u[6] = 12346 \wedge u[7] = 0 \\
& \wedge \mathrm{a}[2] = \mathrm{b}[2] \wedge \mathrm{b}[2] = \mathrm{c}[2] \\
& \wedge (\mathit{Before}(\mathit{Last}(\mathrm{a}[3], \mathrm{b}[3]), \mathrm{c}[3]) \vee \mathit{Before}(\mathrm{c}[4], \mathrm{a}[3])) \\
& \wedge \mathit{Before}(1009, \mathrm{a}[6]) \wedge \mathit{Before}(\mathrm{a}[5], 1019) \\
& \qquad \wedge \mathit{Before}(1009, \mathrm{b}[6]) \wedge \mathit{Before}(\mathrm{b}[5], 1019) \\
& \qquad \wedge \mathit{Before}(1009, \mathrm{c}[6]) \wedge \mathit{Before}(\mathrm{c}[5], 1019) \\
&)\}
\end{aligned}
$$

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), and 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 snapshot relational query language SQL [23]). The skeletal Quel append statement,

append to $R(t_{i_1}.D_{j_1}, \ldots, t_{i_r}.D_{j_r})$
  where ψ

has the following tuple calculus semantics:

$$\begin{aligned} R' = R \cup \{u^{(r)} \mid (\exists t_1) \cdots (\exists t_k)(R_1(t_1) \wedge \cdots \wedge R_k(t_k)\\ \wedge (\forall l)(1 \le l \le r.u[l] = t_{i_l}[j_l])\\ \wedge \psi')\} \end{aligned}$$

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 $R(t_{i_1}.D_{j_1}, \ldots, t_{i_r}.D_{j_r})$
  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:

Example 13. Merrie actually joined the department a month earlier.

append to Faculty (Name = "Merrie", Rank = "Assistant")
  valid from "8-77" **to** "12-82"

then we will have to append the following tuple:

		Valid time		Transaction time	
Name	Rank	(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 (cf., 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.

$$
\begin{aligned}
R' = R \cup \{u^{(r+4)} \mid & (\exists t_1) \cdots (\exists t_k)(R_1(t_1) \wedge \cdots \wedge R_k(t_k) \\
& \wedge (\forall l)(1 \le l \le r . u[l] = t_{i_l}[j_l]) \\
& \wedge u[r+3] = \mathit{current\ transaction\ id} \wedge u[r+4] = \infty \\
& \wedge \psi' \\
& \wedge \Gamma_\tau \\
& \wedge (\forall l)(1 \le l \le k . t_l[stop] = \infty) \\
& \wedge ((\exists s)(R(s) \wedge (\forall l)(1 \le l \le r . s[l] = u[l]) \wedge (C_1^a \vee C_2^a \vee C_3^a \vee C_4^a) \\
& \quad \vee (\neg (\exists s)(R(s) \wedge (\forall l)(1 \le l \le r . s[l] = u[l]) \wedge u[r+1] = \Phi_v \\
& \quad \wedge u[r+2] = \Phi_\chi)) \\
&)\}
\end{aligned}
$$

where

$$
\begin{aligned}
C_1^a = & (\mathit{Before}(s[r+1], \Phi_v) \wedge \mathit{Before}(\Phi_v, s[r+2]) \wedge \mathit{Before}(s[r+2], \Phi_\chi) \\
& \wedge u[r+1] = s[r+2] \wedge u[r+2] = \Phi_\chi) \\
C_2^a = & (\mathit{Before}(\Phi_v, s[r+1]) \wedge \mathit{Before}(s[r+2], \Phi_\chi) \\
& \wedge ((u[r+1] = \Phi_v \wedge u[r+2] = s[r+1]) \vee (u[r+1] = s[r+2] \wedge u[r+2] = \Phi_\chi))) \\
C_3^a = & (\mathit{Before}(\Phi_v, s[r+1]) \wedge \mathit{Before}(s[r+1], \Phi_\chi) \wedge \mathit{Before}(\Phi_\chi, s[r+2]) \\
& \wedge u[r+1] = \Phi_v \wedge u[r+2] = s[r+1]) \\
C_4^a = & (\mathit{Before}(s[r+1], \Phi_v) \wedge \mathit{Before}(\Phi_\chi, s[r+2]) \\
& \wedge \mathrm{False})
\end{aligned}
$$

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); recall that an explicit as-of clause is not permitted in any modification statement. The second change is the rather complicated computation of the valid times for the tuples to be added, appearing as the last three lines of the tuple calculus statement, which replace the third

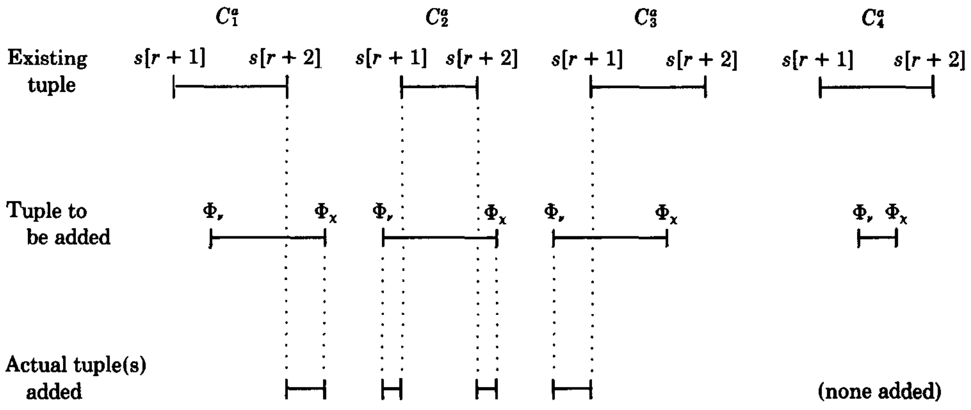

Fig. 15. Calculating the valid time in an append statement.

line in the tuple calculus statement for the retrieve statement. The four clauses $C_1^a, \ldots, C_4^a$ in the seventh line handle the various overlap situations between the tuples to be added and the tuples identical in the explicit attributes that already exist during this valid interval. In particular, C_4^a 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.

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

range of t_1 **is** R_1
. . .
range of t_k **is** R_k
range of s **is** R
delete s
  where ψ

has the following tuple calculus semantics:

$$R' = \{s^{(r)} \mid (\exists t_1) \cdots (\exists t_k)(\exists s)(R(s) \land R_1(t_1) \land \cdots \land R_k(t_k) \land \neg\psi')\}$$

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

Example 14. Jane left the department in March 1981.

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

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 (from)	Valid time (to)	Transaction time (start)	Transaction time (stop)
Jane	Full	11-80	∞	10-80	9-85
Jane	Full	11-80	3-81	9-85	∞

Hence the delete statement will probably change some transaction stop times from ∞ to *now*, if the where and when clauses are true at some point, and will probably also add tuples with a transaction start time of *now*, if any of the tuples to be deleted do not completely cover an existing tuple. For the skeleton TQuel delete statement

range of t_1 **is** R_1
. . .
range of t_k **is** R_k
range of s **is** R
delete s
  valid from v **to** χ
  where ψ
  when τ

the tuple calculus statement is

$$\begin{aligned}R' = \{u^{(r+4)} \mid & (\exists t_1) \cdots (\exists t_k)(\exists s)(R_1(t_1) \wedge \cdots \wedge R_k(t_k)\\ & \wedge (\forall l)(1 \le l \le k.t_l[\mathit{stop}] = \infty)\\ & \wedge (\forall l)(1 \le l \le r.u[l] = s[l]) \wedge u[r+1] = s[r+1]\\ & \quad \wedge u[r+2] = s[r+2] \wedge u[r+3] = s[r+3]\\ & \wedge ((\neg \mathit{Affected} \wedge u[r+4] = s[r+4])\\ & \quad \vee (\mathit{Affected} \wedge u[r+4] = \mathit{current\ transaction\ id}))\}\\ \cup \{u^{(r+4)} \mid & (\exists t_1) \cdots (\exists t_k)(\exists s)(R_1(t_1) \wedge \cdots \wedge R_k(t_k)\\ & \wedge (\forall l)(1 \le l \le k.t_l[\mathit{stop}] = \infty)\\ & \wedge (\forall l)(1 \le l \le r.u[l] = s[l])\\ & \wedge \mathit{Affected} \wedge (C_1^d \vee C_2^d \vee C_3^d \vee C_4^d)\\ & \quad \wedge u[r+3] = \mathit{current\ transaction\ id} \wedge u[r+4] = \infty)\}\end{aligned}$$

where

$$\begin{aligned}\mathit{Affected} = (R(s) \wedge \psi' \wedge \Gamma_\tau \wedge s[r+4] = \infty \wedge \mathit{Before}(s[r+1], \Phi_\chi)\\ \wedge \mathit{Before}(\Phi_v, s[r+2]))\end{aligned}$$

$$\begin{aligned}C_1^d &= (\mathit{Before}(s[r+1], \Phi_v) \wedge \mathit{Before}(\Phi_v, s[r+2]) \wedge \mathit{Before}(s[r+2], \Phi_\chi)\\ &\qquad \wedge u[r+1] = s[r+1] \wedge u[r+2] = \Phi_v)\\ C_2^d &= (\mathit{Before}(\Phi_v, s[r+1]) \wedge \mathit{Before}(s[r+2], \Phi_\chi)\\ &\qquad \wedge \text{False})\\ C_3^d &= (\mathit{Before}(\Phi_v, s[r+1]) \wedge \mathit{Before}(s[r+1], \Phi_\chi) \wedge \mathit{Before}(\Phi_\chi, s[r+2])\\ &\qquad \wedge u[r+1] = \Phi_\chi \wedge u[r+2] = s[r+2])\end{aligned}$$

$$\begin{aligned} C_4^d = {} & (Before(s[r+1], \Phi_v) \wedge Before(\Phi_\chi, s[r+2]) \\ & \wedge ((u[r+1] = s[r+1] \wedge u[r+2] = \Phi_v) \\ & \quad \vee (u[r+1] = \Phi_\chi \wedge u[r+2] = s[r+2])) \end{aligned}$$

Both sets are similar in the first two lines, placing conditions on the underlying tuple variables. The only difference is in the manner in which the implicit time attributes are determined. The first set contains all tuples in past historical relations of R and all tuples in the current historical relation of R that are not *Affected*, that is, that do not satisfy the predicate in the where or when clauses or whose valid intervals do not overlap with the specified valid interval. These tuples remain unchanged by the delete statement. This statement also deals with the *Affected* tuples, effectively removing them by setting their *stop* time to *current transaction id.* The *stop* time of these tuples was previously ∞; no other attributes, implicit or explicit, are modified. The second set deals with the existing tuples that only partially should be deleted, in a manner similar to that employed in the semantics of the append statement. Those portions that should not have been deleted are added back in the second set. The clauses $C_1^d, \ldots, C_4^d$ calculate the valid times of the tuples to be added back. In the situation covered by C_1^d, 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 (i.e., is added back by the second step), and the second, which is removed (i.e., is not added back by the second set). This is the situation illustrated in the example above where Jane leaves the department. In the situation covered by C_2^d, the tuple to be deleted overlaps the existing tuple completely, so the existing tuple is deleted (i.e., no tuple is added back). C_3^d is similar to C_1^d. In the situation covered by C_4^d, the existing tuple is partitioned into three intervals, and only the middle one is deleted (i.e., the left and right remaining tuples are added back). In all cases the tuples added back have a *from* time of *now* and a *to* time of ∞.

The semantics of the replace statement is even more complex. The replace statement has a semantics similar to that of a delete statement followed by an append statement. It is not equivalent to a delete followed by an append when the expressions in the target list mention the primary tuple variable. Hence the semantics of the replace statement must be considered separately. The skeletal Quel replace statement

range of t_1 **is** R_1
. . .
range of t_k **is** R_k
range of s **is** R
replace $s(t_{i_1}.D_{j_1}, \ldots, t_{i_r}.D_{j_r})$
  where ψ

has the following tuple calculus semantics:

$$\begin{aligned} R' = \{u^{(r)} \mid {} & (\exists s)(\exists t_1) \cdots (\exists t_k)(R(s) \wedge R_1(t_1) \wedge \cdots \wedge R_k(t_k)) \\ & \wedge ((u = s \wedge \neg\psi') \\ & \quad \vee ((\forall l)(1 \le l \le r.u[l] = t_{i_l}[j_l]) \wedge \psi')) \\ &)\} \end{aligned}$$

Note that the second line is very similar to the tuple calculus semantics of the Quel delete statement, and that the third line is identical to the semantics of the Quel append statement. The same strategy can be used in TQuel. The tuple calculus semantics of the skeleton TQuel replace statement

range of t_1 **is** R_1
. . .
range of t_k **is** R_k
range of s **is** R
replace $s(t_{i_1}.D_{j_1}, \ldots, t_{i_r}.D_{j_r})$
  valid from v **to** χ
  where ψ
  when τ

is the following:

$$\begin{aligned}
R' = \{u^{(r+4)} \mid\, & (\exists t_1) \cdots (\exists t_k)(\exists s)(R_1(t_1) \wedge \cdots R_k(t_k)\\
& \wedge (\forall l)(1 \le l \le k.t_l[stop] = \infty)\\
& \wedge (\forall l)(1 \le l \le r.u[l] = s[l]) \wedge u[r+1] = s[r+1] \wedge u[r+2] = s[r+2]\\
& \quad \wedge u[r+3] = s[r+3]\\
& \wedge (\neg \mathit{Affected} \wedge u[r+4] = s[r+4])\\
& \quad \vee (\mathit{Affected} \wedge u[r+4] = \mathit{current\ transaction\ id}))\}\\
\cup \{u^{(r+4)} \mid\, & (\exists t_1) \cdots (\exists t_k)(\exists s)(R_1(t_1) \wedge \cdots \wedge R_k(t_k)\\
& \wedge (\forall l)(1 \le l \le k.t_l[stop] = \infty)\\
& \wedge (\forall l)(1 \le l \le r.u[l] = s[l])\\
& \wedge \mathit{Affected} \wedge (C_1^d \vee C_2^d \vee C_3^d \vee C_4^d) \wedge u[r+3] = \mathit{current\ transaction\ id}\\
& \quad \wedge u[r+4] = \infty)\}\\
\cup \{u^{(r+4)} \mid\, & (\exists t_1) \cdots (\exists t_k)(R_1(t_1) \wedge \cdots \wedge R_k(t_k)\\
& \wedge (\forall l)(1 \le l \le k.t_l[stop] = \infty)\\
& \wedge (\forall l)(1 \le l \le r.u[l] = t_{i_l}[j_l]) \wedge u[r+3] = \mathit{current\ transaction\ id}\\
& \quad \wedge u[r+4] = \infty\\
& \wedge \psi' \wedge \Gamma_\tau\\
& \wedge ((\exists s)(R(s) \wedge (\forall l(1 \le l \le r).s[l] = u[l]) \wedge (C_1^a \vee C_2^a \vee C_3^a \vee C_4^a))\\
& \quad \vee (\neg(\exists s)(R(s) \wedge (\forall l)(1 \le l \le r.s[l] = u[l])) \wedge u[r+1] = \Phi_v\\
& \qquad \wedge u[r+2] = \Phi_\chi))\\
&)\}
\end{aligned}$$

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 snapshot database. Hence the tuples participating in a Quel statement are in the snapshot 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 snapshot database (this is discussed further in Section 2.3).

$$\begin{tikzcd}
\text{(Figure 7) TDB} \arrow[r, "\text{slice at } \tau"] \arrow[d, "Q[\tau]", "\text{(TQuel semantics)}"'] & \text{snapshot DB (Figure 18)} \arrow[d, "Q"', "\text{(Quel semantics)}"] \\
\text{(Figure 8) } R_T \arrow[r, "\text{slice at } \tau"] & R_S = R'_S \text{ (Figure 17)} \\
& \text{(Figure 6)}
\end{tikzcd}$$

Fig. 16. Outline of the reduction proof.

We will show that the TQuel semantics just presented reduces to the standard Quel semantics when applied to a *snapshot database slice* (all current tuples valid at a particular time) of the TDB. A snapshot database slice at time τ is formed by first eliminating the event relations (since snapshot 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 attributes.

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 (e.g., the one given in Figure 7) at time τ 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 4.6 will be applied. A snapshot 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, on the relation in Figure 7. The database slice at τ = 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 snapshot database slice at the same time τ of the entire TDB (shown in Figure 18) to arrive at another snapshot relation, R'_S. To show that the TQuel semantics reduces to the standard Quel semantics when applied to a snapshot database slice, we must show the following:

$$R_S = R'_S$$

The reduction implies that Figures 6 and 17 are identical.

The proof of this equality revolves around the defaults for the valid, when, and as-of clauses specified in Section 4.6. The defaults effectively take a database slice at τ = *now*, which is the time the query is executed. The default when and valid clauses state that all the underlying tuples are valid for the entire interval

Associates (Name):

Name
Merrie
Tom

Fig. 17. Slice of the associates relation at January 1, 1984.

Faculty (Name, Rank):

Name	Rank
Jane	Full
Merrie	Associate
Tom	Associate

Fig. 18. A database slice at January 1, 1984.

the resulting tuple was 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.

A similar reduction can be argued concerning the modification statements, as their defaults were specifically chosen to ensure their reducibility to the standard Quel semantics. Figure 16 still applies if Q is interpreted to denote a valid Quel modification statement. The modification statement Q executed at time τ will cause a new historical state to be appended to one of the temporal relations (we assume that *Before*(*start*, τ) for all historical states in the database). A snapshot slice at τ on this historical state will result in the same snapshot state as if the modification statement, with its Quel semantics, had been executed on a snapshot slice of the original temporal database.

The benefit of these reductions is that the intuition and understanding gained by using Quel on a snapshot database applies to TQuel on a TDB. A second benefit is that it forces the definition of the defaults to support this intuition.

5.9 The Nondeletion Property

In Section 2 we argued that rollback and temporal relations were append only: An update to such a relation would result in a new snapshot state or historical state, respectively, being appended to the existing relation. As there are several means of embedding a four-dimensional temporal relation (see Section 5.1) and several ways to define the semantics of the modification statements given a particular embedding, it is not necessary for the *semantics* to be append only, as long as the *model* remains append only. However, the semantics presented above *is* append only, with one qualification: The transaction *stop* time of some existing tuples is changed from ∞ to *now* by a delete or replace statement. This distinction is not important in a practical sense, in that a transition from ∞ to *now* can be effected by a write-once storage device such as an optical disk drive through the considered selection of the encoding for ∞ (an encoding of all zeros is sufficient). However, to be precise, we will term a semantics with this qualification a *nondeletion* semantics, in contrast to an append-only semantics.

It is easy to prove that the semantics just presented has the nondeletion property. The append statement is truly append only, so it trivially has this property. In the deletion semantics, the first set is equivalent to the original relation with the exception of the *Affected* tuples, of which only the *stop* attribute (the $r + 4$ attribute) is changed to *now*. An examination of the definition of *Affected* reveals that the previous value of this attribute was ∞. Hence the deletion semantics has the nondeletion property! The same argument applies to the replace statement.

6. IMPLEMENTATION

The formulation of the TQuel semantics as tuple relational calculus expressions offers a straightforward means to implement a temporal DBMS. 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 snapshot relations that embed the temporal relations. The TQuel query in Example 8 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 snapshot 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 12 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 [81] along somewhat different lines [3]. Our prototype adopts the scheme of augmenting each tuple with two transaction time attributes for a rollback and a temporal relation, and one or two valid time attributes for a historical and temporal relation depending on whether the relation models events or intervals. The parser was modified to accept TQuel statements and generate an extended syntax tree with subtrees for **valid**, **when**, and **as-of** clauses. Some of the query evaluation modules were changed to handle the newly defined node types and implicit time attributes. Functions to handle temporal operators **start of**, **end of**, **precede**, **overlap**, **extend**, and **equal** were added in the one-variable query processing interpreter. The system relation was modified to support various combinations of implicit time attributes, which depend on the type of a relation as specified by its **create** statement. A time attribute is represented as a 32-bit integer with a resolution of 1 second. It has a distinct type, so that imput and output can be done in human-readable form by automatically converting to and from the internal representation. Various formats of date and time are accepted for input, and resolutions ranging from a second to a year are selectable for output.

The prototype supports all the augmented TQuel statements: **retrieve**, **append**, **delete**, **replace**, and **create**. The valid, when, and as-of clauses are fully supported, though default values for these clauses are not yet supplied. The **copy** statement was modified to perform batch input and output of relations having time attributes. It also supports all four types of databases: snapshot, rollback, historical, and temporal. Coalescing, aggregates, and schema evolution are not yet supported.

For a rollback relation, an append operation inserts a tuple with the transaction-start and transaction-stop attributes set to the current time and "forever," respectively. A delete operation on a tuple simply changes the transaction-stop attribute to the current time. A replace operation first executes a delete operation, then inserts a new version with the transaction-start attribute set to the current time. A historical relation follows similar steps for append, delete, and replace operations with the valid-from and valid-to attributes as the counterparts of transaction-start and transaction-stop attributes.

For a temporal relation, an append operation inserts a tuple with the transaction start of the current time, and transaction stop of "forever." Attributes valid from and valid to are set as specified by the valid clause, or defaulted if it is absent. A delete operation on a tuple sets the transaction-stop attribute to the current time indicating that the tuple was virtually deleted from the relation. Next, a new version with the updated valid-to attribute is inserted indicating that the version has been valid until that time. A replace operation first executes a delete operation as above, then appends a new version marked with appropriate time attributes. Therefore, each replace operation in a temporal relation inserts two new versions. This scheme has a high overhead in terms of space, but completely captures the history of retroactive and proactive changes. In addition, all modification operations for rollback and temporal relations in this scheme have the nondeletion property, so write-once optical disks can be utilized.

The prototype was constructed in about three person-months over a period of a year; this figure does not include familiarization with the Ingres internals or with TQuel. Most changes were additions, increasing the source by 2,900 lines, or 4.9 percent (our version of Ingres is approximately 58,800 lines long).

A benchmark set of queries was run to study the performance of the prototype [3]. As expected, the performance rapidly deteriorated as information was added to the database. Access methods such as sequential scan, hashing, and ISAM all suffered. In addition, reorganization did not help shorten overflow chains, because all versions of a tuple share the same key. These results indicate that new storage structures are needed for TDBs to obtain adequate performance.

7. CONCLUSION

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 attribute is fixed at a particular time. Finally, a prototype implementation was described.

7.1 Other Query Languages Incorporating Time

In order to compare TQuel with the other query languages supporting time, we introduce 17 properties and rate each query language on these criteria. Each property is defined below; a summary of the analysis appears in Table I. Four of the properties are *essential*; we contend that no query language not having all 4 essential properties should be considered to be a well-defined temporal query language. The remaining 13 properties are *desirable*; the ideal temporal query language would also possess all of these properties. Many criteria have been suggested to evaluate and compare temporal query languages; we have chosen these because they are well defined, are independent of any specific query language, are not logically implied by other criteria, and are demonstrably beneficial. Other criteria not sharing these properties will be discussed later.

We evaluate the following query languages on these criteria:

—Tansel's algebra, also operating on non-first-normal-form (N1NF) relations, but not requiring homogeneity [26]; and HQuel, his extension to Quel along similar lines [83, 84];

—the homogeneous temporal query language (HTQuel), an extension of Quel operating on N1NF relations consisting of attributes containing one or more (value, interval) pairs (this representation was discussed in Section 5.1); the intervals within a tuple must be identical for all attributes (termed *homogeneity*) [38];

—the query language for the multihomogeneous model (MHM), an extension of the homogeneous model where homogeneity is required only for subsets of the attributes within a tuple [36];

—natural language as formalized in the intensional logic IL_s [27];

—Legol 2.0 [46, 47], a language based on relational algebra used to formalize legislation;

—Ariav's time oriented structured query language (TOSQL) [11, 12], an extension of SQL [43];

—the query language utilizing Ben-Zvi's time relational model (TRM), discussed briefly in Section 5.1; this language, also an extension of SQL, was the first to support more than one kind of time [15];

—TSQL [65], another extension of SQL;

—the algebra of Clifford and Croker accompanying their Historical Relational Data Model (HRDM) [25].

Table I. Comparison of Query Languages Concerning Time

	TQuel	Tansel	HQuel	HTQuel	MHM	IL_s	Legol	TOSQL	TRM	TSQL	HRDM
Essential properties											
Retrieval semantics provided	✓	✓	✓	✓	☐	✓	☐	☐	✓	☐	✓
Historical queries	✓	✓	✓	☐	✓	✓	✓	☐	☐	✓	?
Rollback	✓	☐	☐	☐	☐	☐	☐	✓	✓	☐	☐
Implementability demonstrated	✓	✓	✓	✓	☐	☐	✓	☐	✓	☐	✓
Desirable Properties											
Operational semantics provided	✓	✓	✓	✓	☐	☐	☐	☐	✓	☐	✓
Snapshot completeness	✓	✓	✓	✓	✓	✓	✓	☐	✓	✓	✓
Snapshot reducibility	✓	☐	☐	✓	?	?	✓	?	✓	?	☐
Update semantics provided	✓	✓	✓	☐	☐	☐	☐	☐	✓	☐	☐
Nonprocedural query language	✓	☐	✓	✓	✓	✓	☐	✓	✓	✓	☐
Homogeneous model	✓	☐	☐	✓	☐	✓	✓	✓	✓	✓	✓
Canonical model	✓	☐	☐	☐	☐	?	?	?	?	?	?
Implementation exists	P	☐	☐	☐	☐	☐	P	☐	☐	☐	☐
Supports evolving schema	P	☐	☐	☐	☐	☐	☐	☐	☐	☐	✓
Optimization strategies	P	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐
Nondeletion	✓	☐	☐	?	?	?	?	?	✓	?	?
Aggregates formalized	✓	✓	✓	☐	☐	P	P	P	P	P	☐
Temporal indeterminacy	P	☐	☐	☐	☐	☐	☐	☐	☐	P	☐

✓	Satisfies criterion
P	Partial compliance
☐	Criterion not satisfied
?	Not specified in papers

We omit Time-By-Example [86] in this comparison due to its similarity to HQuel.

The first requirement is that a temporal query language must be well defined. More specifically, it should have a formal *retrieval semantics*. Without a formal semantics, the meaning of each construct, and the interaction between constructs, is unclear. A check in this category means that a formal retrieval semantics has been presented in the literature (as a published or working paper). TQuel is formalized in this paper using the tuple calculus. Gadia claims that HTQuel may be formalized using a special temporal relational calculus [38]. Tansel's algebra is formalized in conventional set theory [83], and HQuel is formalized in this algebra. IL_s is itself a typed, higher order lambda calculus incorporating indexical semantics. The semantics of TRM is specified by utilizing a formally defined "time view" operation to extract a snapshot slice from the TDB, on which an SQL query (which itself has a formal semantics [23]) is applied. Finally, HRDM is also formalized in conventional set theory. Here the situation is better in TDBs than with conventional relational databases, whose early query languages were often not well defined.

A temporal query language must support *historical queries*, and hence valid time. By "supporting valid time," we mean specifically that queries can be formulated that derive information (i.e., tuples) valid at a point in time from information in underlying relations valid at other points in time, much as snapshot query languages can derive information concerning entities or relationships from information in underlying relations concerning other entities or relationships. Two aspects are thereby captured: the ability to refer to the time that the information was valid and the ability to perform "join-like" operations on valid time; a check in this category means that both aspects are present in the language. TQuel accomplishes this through its valid and when clauses. HTQuel does not satisfy this criterion because tuples valid at different times cannot be used in deriving new tuples; its extension, MHM, was defined precisely to circumvent this restriction [36]. TOSQL falls short because only one relation may participate in the query, although aggregates, which are only mentioned, may provide a measure of valid-time support. Because the valid time is eliminated early in the processing of TRM queries, this language also does not fully support valid time. Legol 2.0 supports historical queries via the while, since, until, and during operators; Tansel's algebra and HQuel support them via Cartesian product and implicit Cartesian product, respectively. It is unclear whether HRDM supports historical queries, as the valid time for the result of a Cartesian product is not specified.

A temporal query language must support *rollback*, and hence transaction time. A query language supporting historical queries but not rollback is properly termed *historical*, rather than *temporal* [78]. Some papers have confused the two terms; we differentiate these terms for clarity. Only three query languages support rollback, all through as-of clauses. Transaction time could be added to the remaining query languages without difficulty [63].

Finally, a temporal query language must be *implementable*, if it is to be of more than theoretical value. This property may be demonstrated formally through a semantics based on a well-defined algebra or, practically, through an implementation. A box in this category means that neither has (yet) been presented in the literature. TQuel has both a prototype implementation [3] and an algebra [62]. HTQuel, HQuel, and TRM each have an algebra; Tansel's language and HRDM *are* algebras; and Legol 2.0 both is an algebra and has a prototype implementation.

The other properties are desirable; a temporal query language need not necessarily exhibit any particular property. These properties are listed approximately in order of importance.

A temporal query language should have a well-defined *operational semantics.* By this we mean that a formal temporal algebra should be defined along with a proof of equivalence of the semantics of the query language. Of course, if the query language it itself a formally specified algebraic language, this property is *a priori* satisfied. A check in this category means that a formal algebra and equivalence proof are available in the literature. An incremental algebra has been developed, formalized, and proved equivalent to the tuple calculus semantics of TQuel [62, 63]. An algebra is provided for HTQuel, HQuel, and TRM; Tansel's language and HRDM are algebras. Clifford (IL_s) and Ariav (TOSQL) do not provide an algebra; Legol 2.0 is an algebra, but does not have a formal semantics. Finally, an algebra is defined with TSQL, but the correspondence of the calculus-based TSQL and the algebra was not discussed.

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 [29]. We capture this aspect in the *snapshot completeness* property, which implies that the temporal query language, when applied to a snapshot of the database, is at least as powerful as existing conventional query languages that are complete according to Codd's definition. Note that this property involves both the retrieval and the update semantics. One language, TOSQL, is not snapshot complete, because only one relation may participate in a query.

Snapshot reducibility is a related property, requiring that the language be intuitive, based on one's understanding of snapshot query languages. More precisely, it should be possible to prove that the snapshot relation obtained by applying a temporal query to a TDB and then taking a snapshot is identical to the relation obtained by taking a snapshot of the TDB and applying the analogous snapshot query (in a conventional query language) to the resulting snapshot database. A box in this category means that such a proof is not possible; a question mark means that the proof may be possible, but has not yet been presented in the literature. Snapshot reducibility for TQuel was shown in Section 5.8. The proofs for HTQuel and TRM are simpler, because their algebras are defined in terms of snapshots. That Tansel's algebra, HQuel, and HRDM are not snapshot reducible may be seen by the following, which holds in these algebras:

$$\{(\langle \text{John}, 1, 3\rangle)\} - \{(\langle \text{John}, 1, 2\rangle)\} = \{(\langle \text{John}, 1, 3\rangle)\}$$

where $\{(\langle \text{John}, 1, 3\rangle)\}$ represents a relation containing one tuple containing a single attribute, with a value of John valid from time 1 to time 3. A snapshot at time 2 yields

$$\{(\text{John})\} - \{(\text{John})\} = \{(\text{John})\},$$

which is clearly not the semantics of the snapshot relational difference operator.

A temporal query language should have a well-defined *update semantics*. As temporal relations model reality as it evolves over time, it is especially important that the evolution of the relations is specified. A language satisfying this property will have a formal semantics for the append, delete, and modification statements. A check means that the update semantics for the language has been presented in the literature. Although most languages include update statements, the formal semantics of these statements is left unspecified.

User studies have shown that *nonprocedural* (e.g., calculus-based) *query languages* are often easier to use than procedural (e.g., algebraic) query languages [69]. Only two languages, Tansel's and Legol 2.0, are algebraic.

A temporal query language should have a *homogeneous model*. This property, first identified by Gadia, requires that the periods of validity of all the attributes in a given tuple of a temporal (or historical) relation are identical [38]. Since TQuel is based on a model with time-stamped tuples, its model is clearly homogeneous. Such a model has two substantial benefits. A snapshot of a temporal relation in a homogeneous model is well defined; in particular, it does not contain nulls. In a nonhomogeneous model, a tuple in a snapshot might only contain values for a subset of the attributes; the remaining attributes are problematic. More importantly, relations in a homogeneous model have a particularly simple intuitive semantics: Each tuple models some portion of reality during the period(s) of validity of the tuple. Note that this intuition depends on a well-defined snapshot. Relations in a nonhomogeneous model have a more complicated intuitive semantics; indeed, the intuitive semantics for these models is never explicitly stated.

The desirability of a homogeneous model is controversial. We feel the reason this criterion is not generally accepted is that it is confused with several related properties. One such property is economy of logical representation, discussed below. A second property confused with a homogeneous model is a homogeneous *representation*. In the prototype implementation of TQuel, the representation was indeed homogeneous, but an implementation based on the historical algebra [62] would not be. It is also important to note that homogeneity does not necessarily conflict with the ability to perform "join-like" operations on valid times (c.f., the historical queries criterion discussed above). TQuel, HTQuel, IL_s, Legol 2.0, TOSQL, TRM, TSQL, and HRDM, while having homogeneous models, all support historical queries. MHM and the models of HQuel and Tansel's algebra are nonhomogeneous.

A temporal query language should have a *canonical model*, in which relations are identical if and only if all of their snapshots are identical. In models not satisfying this property, two relations can be radically different, yet their information content, as identified in the snapshots of the relations, can be the same. This situation is confusing to users, who must study the relations to abstract the information contained in the relations. Gadia's use of *weak equality*, which partitions relations into equivalence classes [37], violates this property; in a canonical model, each such equivalence class contains exactly one relation. If the language is based on a model that is tuple time-stamped and requires coalesced relations, then it has a canonical model. A question mark means that, although coalescing was not discussed in the presentation of the model, the model is in fact canonical if coalescing is imposed, and a box means that coalescing was explicitly disallowed. Hence TQuel definitely has a canonical model, whereas IL_s, Legol 2.0, TOSQL, TRM, and HRDM potentially satisfy this property. On the other hand, HTQuel, MHM, HQuel, and Tansel's algebra do not satisfy this criterion, in part because they employ attribute time stamping.

A temporal query language should have an *implementation*, which can provide many clues to desirable and undesirable features of the language. Although a formal semantics is much more important, temporal query languages that have not been implemented should be viewed with caution. A check means that an implementation has been completed and is discussed in the literature. Only two languages, TQuel and Legol 2.0, have been implemented, both as prototypes.

A temporal query language should support an *evolving schema*, where the schema is allowed to change over transaction time, and where past versions are accessed according to the schema in effect at the time the version was stored. TQuel supports this feature to the extent that its algebra supports it; the TQuel statements are still being designed. HRDM also supports schema evolution, but couples the lifetime of an attribute in the schema (an interval in transaction time) with the lifetime of a value of that attribute in the relation instance (an interval in valid time). Although the remaining languages include no support for schema evolution, this feature could easily be added to many of them following the approach used in TQuel's algebra [63].

The language implementation should include *optimization strategies*. A good language will aid in defining such strategies; a poor language will present impediments to potential optimizations. Unfortunately, there has been little work in optimization strategies for processing temporal queries. Ahn is actively investigating this aspect in the context of TQuel and has generated some results [1, 2].

Nondeletion was defined in Section 5.9 as the property of a model or language semantics being append only with the exception that a transition from a time of *forever* to a time of *now* be allowed. In that section we proved that the TQuel semantics satisfies this criterion. A check means that this assertion is proved for the update semantics; a box means that the update semantics violates the assertion; and a question mark means that the status of the assertion is unclear from existing explanations of the language, if, for example, no update semantics was given. Since a language must at least have a well-defined update semantics to have this property, HTQuel, MHM, IL_s, TOSQL, TSQL, and Legol 2.0 are immediately rejected. It is possible to prove that Tansel's algebra and HQuel violate the nondeletion criterion and that TRM satisfies it.

The language should include *aggregates*. Such aggregates should be an extension of snapshot aggregates, be time varying just as the relations are time varying, be well defined (i.e., possess a formal semantics), and be integrated into the operational semantics. Aggregates in TQuel are defined, formalized, and operationalized elsewhere [62, 74, 79, 90]. Aggregates are defined in several other languages, but are not given a formal semantics, except in Tansel's algebra and HQuel [85].

Finally, the language and its semantics should support *temporal indeterminacy*, that is, events for which the time of occurrence is not precisely known. This requirement increases in importance as the valid-time granularity shrinks. The granularity of the examples in this paper was 1 month. A more reasonable granularity would be 1 second or even 1 millisecond. However, it is unreasonable to expect that the time of occurrence of every event (e.g., the promotion of a faculty member) be known to that precision. TSQL's model allows a default value of NULL for the valid-to time. Nontemporal attributes can also have a value of NULL, but the handling of such values is not discussed in detail. A modification to the formal semantics to incorporate indeterminacy was proposed for TQuel [74]; more work is needed.

We should mention eight properties that have been mentioned by others, but that were *not* included. Although *simplicity* is highly desirable, it is very difficult to define. The *ability to deal explicitly with "when"* [35] is also difficult to define and may be captured by historical queries to some degree. *Expressive power* is easier to define, yet may be examined more carefully when broken into its constituent parts: historical queries, rollback, snapshot completeness, aggregates, and indeterminacy. *Efficiency* is included under the property of implementation and optimization strategies.

Economy of logical representation (*ELR*) [36] was not included because it is not necessarily even desirable. Relations in a model having this property contain fewer tuples than relations in a model not exhibiting ELR. Gadia has criticized non-ELR models as exhibiting "vertical temporal anomalies" and goes on to state that "one would get a better query language if the distinction between a logical unit of data and its physical representation is minimized" [36]. The success of the relational model is generally attributed to exactly that distinction. Codd lists physical data independence as an essential property of any relational DBMS [31]. Hence ELR is irrelevant in terms of efficiency, because it only indirectly affects the size of the *physical* representation. However, it may negatively affect such other desirable aspects as having a homogeneous model or having a canonical model.

Finally, *temporal completeness* (and its variants *minimal completeness* and *maximal completeness* [35]) was not included because it does not have an accepted definition. Gadia and Vaishnav have proposed their temporal relational algebra [35] as a benchmark [38]; however, the issue of why this particular algebra is an appropriate benchmark for completeness was never discussed. Two reasons why their algebra is perhaps inappropriate are that it is a multisorted algebra over relations and temporal domains, and that it only concerns valid time.

A perhaps more satisfying definition of temporal completeness originates from first principles. Snapshot completeness, as first proposed by Codd [29], is a rather arbitrary measure of the expressive power of the language. Temporal completeness should be an extension of (1) snapshot completeness. To prove snapshot completeness, the criteria of (2) a retrieval semantics and (3) an update semantics are necessary. The adjective *temporal* implies supporting both valid and transaction time, so (4) historical queries and (5) rollback are included. These five criteria form a minimal definition of temporal completeness, capturing a notion of expressive power that does not rely on any particular temporal algebra. We can also define *historical completeness* as temporal completeness without the rollback criterion. Working from these definitions, HQuel and Tansel's algebra are historically complete, IL_s and Legol 2.0 come close, and it may be possible to show that MHM is historically complete. Only TQuel is temporally complete.

We have compared TQuel with other temporal query languages on many criteria. An orthogonal set of criteria, less precisely defined but nonetheless important, arises from a different source: comparing the query language's expressive power with that of natural language. In their investigation of how to encode the temporal aspects of natural language, Maran et al. have proposed three kinds of natural language metaphors [57]. The first, involving event types, distinguishes between *process* versus *state*, and also between *durative* or *punctual* duration length. Durative lengths may be represented by interval relations; punctual lengths by event relations. A process may be represented by a collection of event relations; a state by a collection of interval relations. A heterogeneous collection of relations can be transformed into a process description by using the begin of/end of operators, or into a state description by using the extend operator. The duality between process and state is further explored elsewhere [74].

The second metaphor concerns event orientation, and differentiates between *onset* and *terminal* boundary references, and between *progressive* and *completive* state of time flows. The onset boundary reference can be expressed as "**when** "now" **precede end of**?"; and the terminal boundary reference can be expressed as "**when end of**? **precede** "now"." Similarly, the progressive time flow can be expressed as "**when** "now" **precede begin of**?" and the completive time flow as "**when begin of**? **precede** "now"."

Finally, the metaphor for the speaker's point of view contrasts *witness* (i.e., present), *retrospective* (i.e., past), and *modal* (i.e., future). These can be represented in TQuel, respectively, as "**as of** "now"," **as of** "beginning"," and "**as of** "now" **through** "forever"."

We have shown that the temporal data model coupled with TQuel can, to a rough approximation, express the metaphors for time in natural language. The point here is not that this is or should be the final definition of temporal completeness or even of the expressive power of natural language concerning time, but rather that comparisons between temporal query languages and natural language *can* be made and that further investigation along these lines is warranted.

7.2 Further Work

This paper has defined a temporal query language and provided a formal semantics for this language. However, much more research is necessary before a viable temporal DBMS 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:

Example 15. Who has been an associate professor for at least 5 years?

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

The semantics for relative constants is still under study.

Quel supports three attribute types, in multiple sizes: integer (1, 2, and 4 bytes long), floating point (4 and 8 bytes long), and character data (1–255 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 attributes are used correctly. Quel also supports scalar functions such as **abs**, **mod**, and **sin**. Scalar temporal functions, such as **duration**, which compute time-varying values, are needed in the language.

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 [9]? Temporal constants, as discussed in Section 4.2, provide only a partial answer. Should valid and transaction time be linear or branching? Branching time, although more complex than linear time, does have some interesting properties [11, 80]. 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 [11, 73]. Should periodic or cyclic events and intervals (e.g., fiscal year, monthly payments) or causality be incorporated [11, 14]? How well does TQuel correspond to the user's temporal perception? Further work is necessary in all of these areas.

The prototype described in Section 6 exhibits 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 [1].

Temporal DBMSs in general are at approximately the same stage as snapshot relational systems were in the early 1970s [49]: Several high-level, nonprocedural query languages have been designed and formalized, and prototype implementations exist. All the questions asked concerning snapshot relational databases, including those that have already been answered, must be asked (and answered) anew in the context of TDBs.

APPENDIX A. 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[1] 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

[1]UNIX is a trademark of AT&T Bell Laboratories.

following nonterminals 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, floating point, or temporal
⟨attribute⟩	the name of an attribute
⟨relation⟩	a relation name
⟨string⟩	a string constant
⟨tuple variable⟩	the name of a tuple variable
⟨attribute specs⟩	a list of the names and types for the user-specified attributes

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⟩ ⟨attribute 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⟩	::= ⟨attribute⟩ ⟨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⟩
                          | begin of ⟨either-expression⟩
                          | end of ⟨either-expression⟩
                          | ( ⟨e-expression⟩ )
⟨i-expression⟩          ::= ⟨interval element⟩
                          | ⟨either-expression⟩ overlap ⟨either-expression⟩
                          | ⟨either-expression⟩ extend ⟨either-expression⟩
                          | ( ⟨i-expression⟩ )
⟨either-expression⟩     ::= ⟨e-expression⟩ | ⟨i-expression⟩
⟨event element⟩         ::= ⟨tuple variable⟩
⟨interval element⟩      ::= ⟨tuple variable⟩ | ⟨temporal constant⟩
⟨temporal constant⟩     ::= ⟨string⟩
⟨temporal pred⟩         ::= ⟨interval element⟩
                          | ⟨event element⟩
                          | ⟨either-expression⟩ precede ⟨either-expression⟩
                          | ⟨either-expression⟩ overlap ⟨either-expression⟩
                          | ⟨either-expression⟩ equal ⟨either-expression⟩
                          | ⟨temporal pred⟩ and ⟨temporal pred⟩
                          | ⟨temporal pred⟩ or ⟨temporal pred⟩
                          | ( ⟨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 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 (**begin of**, **end of**, **not**) have the highest precedence, followed in order by the binary temporal constructors (**extend**, **overlap**), the temporal predicate operators (**precede**, **overlap**, **equal**), and finally the binary logical operators (**and**, **or**). Binary operators of equal precedence are left associative; unary operators of equal precedence are right associative. The binary temporal constructors, temporal predicate operators, and logical operators are all commutative, except for **precede**.

Note that the distinction between ⟨interval element⟩ and ⟨event element⟩ makes the grammar context sensitive. In practice, this distinction is ignored in the LALR(1) parser, and the resulting parse tree is type-checked in the semantic analysis phase.

ACKNOWLEDGMENTS

The author is grateful to Ilsoo Ahn, Santiago Gomez, Bharat Jayaraman, Ed McKenzie, and Juan Valiente 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.

REFERENCES

1. AHN, I. Towards an implementation of database management systems with temporal support. In *Proceedings of the International Conference on Data Engineering* (Los Angeles, Calif., Feb.). IEEE Press, New York, 1986, pp. 374–381.

2. AHN, I. Performance modeling and access methods for temporal database management systems. Ph.D. dissertation, Computer Science Dept., Univ. of North Carolina, Chapel Hill, July 1986.
3. AHN, I., AND SNODGRASS, R. Performance evaluation of a temporal database management system. In *Proceedings of ACM SIGMOD International Conference on Management of Data* (Washington, D.C., May). ACM, New York, 1986, pp. 96–107.
4. ALLEN, J. F. An interval-based representation of temporal knowledge. In *Proceedings of the International Joint Conference on Artificial Intelligence* (Vancouver, B.C.). 1981, pp. 221–226.
5. ALLEN, J. F. Maintaining knowledge about temporal intervals. *Commun. ACM 26*, 11 (Nov. 1983), 832–843.
6. ALLEN, J. F. Towards a general theory of action and time. *Artif. Intell. 23*, 2 (July 1984), 123–154.
7. ANANTHARAMAN, T. S., CLARKE, E. M., FOSTER, M. J., AND MISHRA, B. Compiling path expressions into VLSI circuits. In *Proceedings of the ACM Symposium on Principles of Programming Languages* (New Orleans, La., Jan.). ACM, New York, 1985, pp. 191–204.
8. ANDERSON, T. L. The database semantics of time. Ph.D. dissertation, Univ. of Washington, Jan. 1981.
9. ANDERSON, T. L. Modeling time at the conceptual level. In *Improving Database Usability and Responsiveness*, P. Scheuermann, Ed. Academic Press, New York, 1982, pp. 273–297.
10. ANDLER, S. A. Predicate path expressions: A high-level synchronization mechanism. Ph.D. dissertation, Computer Science Dept., Carnegie-Mellon Univ., Aug. 1979.
11. ARIAV, G. Preserving the time dimension in information systems. Ph.D. dissertation, Dept. of Decision Sciences, The Wharton School, Univ. of Pennsylvania, Apr. 1984.
12. ARIAV, G. A temporally oriented data model. *ACM Trans. Database Syst. 11*, 4 (Dec. 1986), 499–527.
13. ARIAV, G., AND MORGAN, H. L. MDM: Handling the time dimension in generalized DBMS. Work. Pap., Dept. of Decision Sciences, The Wharton School, Univ. of Pennsylvania, May 1981.
14. ARIAV, G., AND MORGAN, H. L. MDM: Embedding the time dimension in information systems. TR 82-03-01, Dept. of Decision Sciences, The Wharton School, Univ. of Pennsylvania, 1982.
15. BEN-ZVI, J. The time relational model. Ph.D. dissertation, Univ. of California, Los Angeles, 1982.
16. BERZINS, V., AND KAPUR, D. Denotational and axiomatic definitions for path expressions. Comput. Struct. Group Memo 153-1, Laboratory for Computer Science, Massachusetts Institute of Technology, Cambridge, Mass., Nov. 1977.
17. BJORK, L. A., JR. Generalized audit trail requirements and concepts for data base applications. *IBM Syst. J. 14*, 3 (1975), 229–245.
18. BONTEMPO, C. J. Feature analysis of query-by-example. In *Relational Database Systems*. Springer-Verlag, New York, 1983, pp. 409–433.
19. BREUTMANN, B., FALKENBERG, E. F., AND MAUER, R. CSL: A language of defining conceptual schemas. In *Data Base Architecture*. North-Holland, Amsterdam, 1979.
20. BUBENKO, J. A., JR. The temporal dimension in information modeling. Tech. Rep. RC 6187 26479, IBM Thomas J. Watson Research Center, Nov. 1976.
21. BUBENKO, J. A., JR. The temporal dimension in information modeling. In *Architecture and Models in Data Base Management Systems*. North-Holland, Amsterdam, 1977.
22. BUBENKO, J. A., JR. Information modeling in the context of system development. In *Proceedings of IFIP Congress 80* (Oct. 6–17). 1980, pp. 395–411.
23. CERI, S., AND GOTTLOB, G. Translating SQL into relational algebra: Optimization, semantics, and equivalence of SQL queries. *IEEE Trans. Softw. Eng. SE-11*, 4 (Apr. 1985), 324–345.
24. CHEESEMAN, P. A representation of time for planning. Tech. Note 278, Artificial Intelligence Center, Feb. 1983.
25. CLIFFORD, J., AND CROKER, A. The historical data model (HRDM) and algebra based on lifespans. In *Proceedings of the International Conference on Data Engineering* (Los Angeles, Calif., Feb.) IEEE Press, New York. To be published.
26. CLIFFORD, J., AND TANSEL, A. U. On an algebra for historical relational databases: Two views. In *Proceedings of ACM SIGMOD International Conference on Management of Data* (Austin, Tex., May). ACM, New York, 1985, pp. 247–265.
27. CLIFFORD, J., AND WARREN, D. S. Formal semantics for time in databases. *ACM Trans. Database Syst. 8*, 2 (June 1983), 214–254.

28. CODD, E. F. A relational model of data for large shared data banks. *Commun. ACM 13*, 6 (June 1970), 377–387.
29. CODD, E. F. Relational completeness of data base sublanguages. *Data Base Systems*. Vol. 6, Courant Computer Symposia Series. Prentice-Hall, Englewood Cliffs, 1972, pp. 65–98.
30. CODD, E. F. Extending the database relational model to capture more meaning. *ACM Trans. Database Syst. 4*, 4 (Dec. 1979), 397–434.
31. CODD, E. F. An evaluation scheme for database management systems that are claimed to be relational. In *Proceedings of the International Conference on Data Engineering* (Los Angeles, Calif., Feb.). IEEE Press, New York, 1986, pp. 719–729.
32. COPELAND, G., AND MAIER, D. Making Smalltalk a database system. In *Proceedings of ACM SIGMOD International Conference on Management of Data* (Boston, Mass., June). ACM, New York, 1984, pp. 316–325.
33. DOWTY, D. R. Studies in the logic of verb aspect and time reference in English. Tech. Rep., Dept. of Linguistics, Univ. of Texas, Austin, 1972.
34. FAGAN, L. M. VM: Representing time-dependent relations in a medical setting. Ph.D. dissertation, Stanford Univ., June 1980.
35. GADIA, S. K. Toward completeness of temporal databases. Unpublished manuscript.
36. GADIA, S. K. Toward a multihomogeneous model for a temporal database. In *Proceedings of the International Conference on Data Engineering* (Los Angeles, Calif., Feb.). IEEE Press, New York, 1986, pp. 390–397.
37. GADIA, S. K. Weak temporal relations. In *Proceedings of the ACM Symposium on Principles of Database Systems* (Los Angeles, Calif.). ACM, New York, 1986.
38. GADIA, S. K., AND VAISHNAV, J. H. A query language for a homogeneous temporal database. In *Proceedings of the ACM Symposium on Principles of Database Systems* (Apr.). ACM, New York, 1985.
39. HABERMANN, A. N. Path expressions. Tech. Rep. Computer Science Dept., Carnegie-Mellon Univ., June 1975.
40. HAMMER, M., AND MCLEOD, D. Database description with SDM: A semantic database model. *ACM Trans. Database Syst. 6*, 3 (Sept. 1981), 351–386.
41. HELD, G. D., STONEBRAKER, M. R., AND WONG, E. INGRES—A relational data base system. In *Proceedings of the 1975 National Computer Conference*, vol. 44 (Anaheim, Calif., May 19–22). AFIPS Press, Reston, Va., 1975, pp. 409–416.
42. HIRSCHMAN, C., AND STORY, G. Representing implicit and explicit time relations in narrative. In *Proceedings of the International Joint Conference on Artificial Intelligence* (Vancouver, B.C., Aug.). 1981, pp. 289–295.
43. IBM. SQL/Data-System, concepts and facilities. Tech. Rep. GH24-5013-0, IBM, Jan. 1981.
44. JAYARAMAN, B. Constructing a parallel implementation from high-level specifications: A case study using resource expressions. In *Proceedings of the 1983 International Conference on Parallel Processing* (Aug. 23-26). IEEE Press, New York, 1983, pp. 416–420.
45. JAYARAMAN, B., AND KELLER, R. M. Resource expressions for applicative languages. In *Proceedings of the 1982 International Conference on Parallel Processing* (Aug.). IEEE Press, New York, 1982, pp. 160–167.
46. JONES, S., AND MASON, P. J. Handling the time dimension in a data base. In *Proceedings of the International Conference on Data Bases* (Heyden, July). British Computer Society, 1980, pp. 65–83.
47. JONES, S., MASON, P., AND STAMPER, R. LEGOL 2.0: A relational specification language for complex rules. *Inf. Syst. 4*, 4 (Nov. 1979), 293–305.
48. KAHN, K., AND GORRY, G. A. Mechanizing temporal knowledge. *Artif. Intell. 9* (Sept. 1975), 87–108.
49. KIM, W. Relational database systems. *ACM Comput. Surv. 11*, 3 (Sept. 1979), 185–211.
50. KIM, W. On optimizing an SQL-like nested query. *ACM Trans. Database Syst. 7*, 3 (Sept. 1982), 443–469.
51. KIMBALL, K. A.. The DATA System. Master's thesis, Univ. of Pennsylvania, Philadelphia, 1978.
52. KLOPPROGGE, M. R. TERM: An approach to include the time dimension in the entity-relationship model. In *Proceedings of the 2nd International Conference on the Entity Relationship Approach* (Washington, D.C., Oct. 12–14), 1981, pp. 477–512.

53. KLUG, A. Equivalence of relational algebra and relational calculus query languages having aggregate functions. *J. ACM 29*, 3 (July 1982), 699–717.
54. LAUER, P. E., AND CAMPBELL, R. H. Formal semantics of a class of high-level primitives for coordinating concurrent processes. *Acta Inf. 5*, 4 (1975), 297–332.
55. LONG, W. J., AND RUSS, T. A. 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.
56. LUM, V., DADAM, P., ERBE, R., GUENAUER, J., PISTOR, P., WALCH, G., WERNER, H., AND WOODFILL, J. Designing DBMS support for the temporal dimension. In *Proceedings of ACM SIGMOD International Conference on Management of Data* (Boston, Mass., June). ACM, New York, 1984, pp. 115–130.
57. MARAN, L. R., SPOOR, D. T., AND WALTZ, D. L. Encoding the natural language meaning of time toward a conceptual model for temporal meaning. Work. Pap. 37, Univ. of Illinois, Urbana-Champaign, May 1983.
58. MCARTHUR, R. P. *Tense Logic.* Reidel, Hingham, Mass., 1976.
59. MCCAWLEY, J. *Tense and Time Reference in English.* Holt, Reinhardt and Winston, New York, 1971.
60. MCDERMOTT, D. A temporal logic for reasoning about processes and plans. *Cognitive Sci. 6* (Dec. 1982), 101–155.
61. MCKENZIE, E. Bibliography: Temporal databases. *ACM SIGMOD Rec. 15*, 4 (Dec. 1986), 40–52.
62. MCKENZIE, E., AND SNODGRASS, R. Supporting valid time: An historical algebra and evaluation. Tech. Rep. TR87-008, Computer Science Dept., Univ. of North Carolina, Chapel Hill, 1987.
63. MCKENZIE, E., AND SNODGRASS, R. Supporting transaction time in the relational algebra. In *Proceedings of the ACM SIGMOD International Conference on Management of Data* (San Francisco, May 1987). ACM, New York, 1987.
64. MONTAGUE, R. The proper treatment of quantification in ordinary English. In *Approaches to Natural Language.* Reidel, Hingham, Mass., 1973.
65. NAVATHE, S. B., AND AHMED, R. A temporal relational model and a query language. Tech. Rep., Computer and Information Sciences Dept., Univ. of Florida, Apr. 1986.
66. OVERMYER, R., AND STONEBRAKER, M. Implementation of a time expert in a database system. *ACM SIGMOD Rec. 12*, 3 (Apr. 1982), 51–59.
67. PALLEY, N. A., ET AL. CLINFO user's guide: Release one. Tech. Rep. R-1543-1-NIH, Rand Corp., 1976.
68. PRIOR, A. *Past, Present, Future.* Oxford University Press, New York, 1967.
69. REISNER, P. Human factors studies of database query languages: A survey and assessment. *ACM Comput. Surv. 13*, 1 (Mar. 1981), 13–31.
70. Relational Technology. *MicroINGRES Reference Manual.* Relational Technology, 1984.
71. RESCHER, N. C., AND URQUHART, A. *Temporal Logic.* Springer-Verlag, New York, 1971.
72. SERNADAS, A. Temporal aspects of logical procedure definition. *Inf. Syst. 5*, 3 (1980), 167–187.
73. SHANNON, K. P. The display of temporal information. Master's thesis, Computer Science Dept., Univ. of North Carolina, Chapel Hill, July 1986.
74. SNODGRASS, R. Monitoring distributed systems: A relational approach. Ph.D. dissertation, Computer Science Dept., Carnegie-Mellon Univ., Dec. 1982.
75. SNODGRASS, R. The temporal query language TQuel. In *Proceedings of the ACM Symposium on Principles of Database Systems* (Waterloo, Ontario, Apr.). ACM, New York, 1984, pp. 204–212.
76. SNODGRASS, R. ED. Research concerning time in databases: Project summaries. *ACM SIGMOD Rec. 15*, 4 (Dec. 1986), 19–39.
77. SNODGRASS, R., AND AHN, I. A taxonomy of time in databases. In *Proceedings of ACM SIGMOD International Conference on Management of Data* (Austin, Tex., May). ACM, New York, 1985, pp. 236–246.
78. SNODGRASS, R., AND AHN, I. Temporal databases: *Computer 19*, 9 (Sept. 1986), 35–42.
79. SNODGRASS, R., AND GOMEZ, S. Aggregates in the temporal query language TQuel. Tech. Rep. TR86-009, Computer Science Dept., Univ. of North Carolina, Chapel Hill, Mar. 1986.

80. STONEBRAKER, M., AND KELLER, K. Embedding experts and hypothetical data bases in a relational data base system. In *Proceedings of ACM SIGMOD International Conference on Management of Data* (Santa Monica, Calif., May). ACM, New York, 1980.
81. STONEBRAKER, M., WONG, E., KREPS, P., AND HELD, G. The design and implementation of INGRES. *ACM Trans. Database Syst. 1*, 3 (Sept. 1976), 189–222.
82. TANDEM COMPUTERS. *ENFORM Reference Manual.* Tandem Computers, Cupertino, Calif., 1983.
83. TANSEL, A. U. Adding time dimension to relational model and extending relational algebra. *Inf. Syst.* To be published.
84. TANSEL, A. U., AND ARKUN, M. E. HQUEL, a query language for historical relational databases. In *Proceedings of the 3rd International Workshop on Statistical and Scientific Databases* (July) 1986.
85. TANSEL, A. U., AND ARKUN, M. E. Aggregation operations in historical relational databases. In *Proceedings of the 3rd International Workshop on Statistical and Scientific Databases* (July) 1986.
86. TANSEL, A. U., ARKUN, M. E., AND OZSOYOGLU, G. Time-By-Example query language for historical databases. Work. Pap., Dept. of Statistics and Computer Information Systems, Baruch College, City University of New York, 1985.
87. TAYLOR, E. F., AND WHEELER, J. A. *Space-Time Physics.* Freeman, San Francisco, Calif., 1966.
88. TSOTSOS, J. K. Temporal event recognition: An application to left ventricular performance. In *Proceedings of the International Joint Conference on Artificial Intelligence* (Vancouver, B.C., Aug.). 1981. pp. 900–907.
89. ULLMAN, J. D. *Principles of Database Systems.* 2nd ed. Computer Science Press, Rockville, Md., 1982.
90. VALIENTE, J. Implementing TQuel aggregates. Master's thesis, Computer Science Dept., Univ. of North Carolina, Chapel Hill. In progress.
91. VILAIN, M. B. A system for reasoning about time. In *Proceedings of the American Association for Artificial Intelligence* (Pittsburgh, Pa., Aug.) 1982, pp. 221–226.
92. WHITROW, G. J. *The Natural Philosophy of Time.* Oxford University Press, New York, 1980.
93. WIEDERHOLD, G., FRIES, J. F., AND WEYL, S. Structured organization of clinical data bases. In *Proceedings of the National Computer Conference*, vol. 44 (Anaheim, Calif., May 19–22). AFIPS Press, Reston, Va., 1975, pp. 479–485.
94. ZANIOLO, C. The database language GEM. In *Proceedings of ACM SIGMOD International Conference on Management of Data* (San Jose, Calif., May). ACM, New York, 1983, pp. 207–218.

Received August 1984; revised August 1985; accepted July 1986