The Temporal Query Language TQuel

Richard Snodgrass

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

ABSTRACT

Recently, attention has been focussed on historical databases *(HDBs), representing an enterprise over time. We have developed a new language,* TQuel, *to query an HDB. TQuel is a superset of Quel, the query language in the Ingres relational database management system. This paper provides an overview of the language, motivating the various design decisions with the objective that it be a minimal extension, both syntactically and semantically, of Quel.*

1. INTRODUCTION

Most conventional databases, whether based on the hierarchical, network, relational, or entity-relationship model, represent the state of an enterprise at a single moment of time. Although they continue to change as new information is added, these changes are viewed as modifications to the state, with the old, out-of-date data being deleted from the database.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 at a particular moment of time.

Recently, attention has been focussed on *historical databases*, representing many states of an enterprise over an interval of time. In such databases, changes are viewed as additions to the information in the database, reflecting the progress of the enterprise over time. Historical databases (HDBs) are thus generalizations of conventional (termed *static*) databases.

We have developed a new language, *TQuel* (*T*emporal *QUE*ry *L*anguage), to query an HDB. The language was originally used in a monitoring system based on the relational model [Snodgrass 1982], but it may be used on HDBs having nothing to do with monitoring. TQuel is a superset of Quel [Held et al. 1975], the query language for the Ingres relational database management system [Stonebraker et al. 1976]. The result is a natural extension of a static relational query language into one which may query a historical database.

This paper discusses the basic design decisions in the context of previous efforts in section 2, and provides an overview of the language in the third section. Aggregates and defaults in TQuel are the topics of sections 4 and 5. The final section concludes with a brief overview of progress on developing a formal semantics for TQuel, and a description of the implementation. The appendix gives the complete syntax of the TQuel retrieve statement.

2. QUERY LANGUAGES FOR HISTORICAL DATABASES

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

The need to handle time more comprehensively surfaced in the early 1970's in the area of

medical information systems, where a patient's medical history is particularly important. The model supported by TOD (for Time Oriented Data bank) [Wiederhold et al. 1975] and several other medical DBMSs (e.g., CLINFO [Palley et al. 1976]) views the database as a set of entity-attribute-value-time quadruples, where the time portion indicates when the information represented by the tuple became valid. Hence, only events are recorded. 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, there has been increased interest in the area of HDBs. In a recent, quite extensive bibliography [Bolour et al. 1982], containing 69 articles from the period 1960 to June, 1982, over half of the referenced articles were published since 1978. This activity may be loosely classified into three emphases: the formulation of a semantics of time at the conceptual level, the development of a model for HDBs analogous to the relational model for static databases, and the design of temporal query languages. It should be noted that the problems inherent in the modeling of time are not unique to information processing; there is a significant literature on related issues in logic (c.f., [McArthur 1976, Prior 1967, Rescher & Urquhart 1971]), philosophy (c.f., [Whitrow 1980]), linguistics (c.f., [Dowty 1972, McCawley 1971, Montague 1973]), physics (c.f., [Taylor & Wheeler 1966]), and artificial intelligence (c.f., [Findler & Chen 1971, Kahn & Gorry 1975]).

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

There are at least two possible approaches to the development of a model for HDBs. One is to extend the semantics of the relational model to directly incorporate time. The other is to base HDBs on the static model, with time appearing as an additional domain type. The first has been successfully applied by Clifford and Warren [Clifford & Warren 1983], with the entity-relationship model used in the formulation of the intensional logic IL_s. This logic serves as a formalism for the temporal semantics of an HDB much as the first-order logic serves as a formalism for the static relational model. Sernadas has take the same approach in defining the temporal process specification language DMTLT, which incorporates a special modal tense logic [Sernadas 1980].

In the second approach, the static relational database model [Codd 1970] serves as the underlying model of the HDB. Each relation contains an additional temporal domain specifying when that tuple was valid. The query language must provide the appropriate values for this domain in the relation being derived. Several benefits accrue from such an approach. The relational database model is simple and is based on the well-developed formalisms of set theory and predicate calculus; database models directly incorporating time are significantly more complex, and are based on newer and less developed logics such as Montague, multiple transition, and temporal logics. Extensions involving aggregates and indeterminacy are easier to formulate in the standard model. Finally, a temporal database based on the relational model can be implemented directly on conventional relational database management systems, utilizing the significant results obtained in this area in the past decade. Many of the same advantages resulted from a similar approach in the design of GEM, a query and update language for a (static) semantic data model [Zaniolo 1983].

Three query languages taking this approach have appeared in the literature. DATA (Dynamic Alerting Transaction System) extends the relational model to include time by viewing the database as time-ordered lists of transactions, each consisting of a tuple and a time when that tuple became valid [Ariav & Morgan 1981]. The database can be queried at previous points of time, or a sequence of recorded events between two times may be displayed. The query language effectively accesses a static database embedded in the HDB.

There have been two relational query languages developed that include temporal constructs. Ironically, both evolved from projects concerned more with the application of data base concepts to other areas than with the development of a new query language. The first, LEGOL 2.0, involved formalizing legislation, where the history of a case is particularly relevant [Jones, et al. 1979, Jones & Mason 1980]. The model supported by this system allows time attributes specifying the period of time each tuple is valid; events may not be stored. LEGOL 2.0 is based on the relational algebra [Codd 1972]. The language was never implemented, although an earlier version of the language was implemented [Stamper 1976] using ISBL [Todd 1976]. In addition, there has been no attempt at a formalization either of the language

or of the way the temporal constructs of the language were to be implemented. The second is TQuel, which, as has been previously mentioned, was developed in conjunction with the specification of a relational monitor. In contrast with LEGOL 2.0, TQuel is based on the relational calculus [Codd 1972], both events and time intervals may be manipulated in TQuel, and the major aspects of the language have been formalized [Snodgrass 1984] and implemented [Snodgrass 1982].

3. OVERVIEW OF TQUEL

TQuel is a superset of Quel [Held et al. 1975], the query language for Ingres [Stonebraker et al. 1976]. An important goal in the design of TQuel was that it be a minimal extension, both syntactically and semantically, of Quel. This objective had 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 1. The Faculty relation lists the faculty members and their rank (one of the values Assistant, Associate, or Full); the Submitted relation lists those papers submitted. In the discussion that follows, the reader is assumed to be familiar with Quel.

Faculty (Name, Rank):

Name	Rank
Jane	Full
Merrie	Associate
Tom	Associate

Submitted (Author, Journal):

Author	Journal
Jane	CACM
Merrie	CACM
Merrie	TODS
Tom	JACM

Figure 1: A static database

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

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

Example 1. List the Associate professors.

will result in the relation shown in Figure 2 when applied to the sample database.

Associates (Name):

Name
Merrie
Tom

Figure 2: Result of a query on a static database

To convert a static database into an HDB, a temporal domain is appended to each relation. The value of this domain specifies when that tuple was valid. For *event relations*, which consist of tuples representing instantaneous occurrences, the temporal domain contains a single time value. For *interval relations*, which consist of tuples representing a state valid during a time interval, the temporal domain contains two time values. Figure 3 illustrates the Faculty relation extended to become an interval relation, and the Submitted relation extended to become an event relation.

Faculty (Name, Rank):

Name	Rank	(Start)	(Stop)
Jane	Assistant	9-71	12-76
Jane	Associate	12-76	12-80
Jane	Full	12-80	3-84
Merrie	Assistant	9-77	12-82
Merrie	Associate	12-82	3-84
Tom	Assistant	9-75	12-80
Tom	Associate	12-80	3-84

Submitted (Author, Journal):

Author	Journal	(Start)
Jane	CACM	11-79
Merrie	CACM	9-78
Merrie	TODS	5-79
Tom	JACM	12-82

Figure 3: A historical database

Since TQuel is a strict superset of Quel, the identical query, when applied to the sample HDB, results in the relation shown in Figure 4.

Associates (Name):

Name	(Start)	(Stop)
Jane	12-76	12-80
Merrie	12-82	3-84
Tom	12-80	3-84

Figure 4: The same query on a historical database

Providing a temporal domain is not sufficient for defining the semantics of an HDB, for users must be constrained in the manner in which they employ this capability. The query language must be designed so that temporal domains are used correctly. The approach taken here is to make the temporal domain implicit in the query language, and to provide facilities in the language for manipulating this implicit domain. TQuel augments the retrieve statement with two components, analogous to the components of the Quel retrieve statement, one specifying how the implicit time domain is computed, and one specifying the temporal relationship of the tuples participating in the derivation.

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, which represent the implicit time domain of the associated relations. The syntax is similar to *path expressions*, which are regular expressions augmented with parallel operators [Andler 1979, Habermann 1975, Shaw 1980].

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

```
range of a is Associates
retrieve into FirstDayAssociates (Name = f.Name)
when a overlap "Sept. 1, 1983"
```

Example 2. List the associate professors on the first day of class.

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

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

Example 3. What papers were written by associate professors?.

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

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

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

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

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

```
retrieve Disgruntled (Name = a.Name)
when (start of a) precede "Jan. 1, 1979"
      and "Jan. 1, 1984" precede (end of a)
```

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

This example also illustrates the **and** operator; the **or** operator is also allowed. The **not** operator is conspicuously absent; there were so many difficulties encountered in defining its semantics that it was disallowed.

Given the **precede** operator, the **extend** operator may be introduced. This operator is similar to the **overlap** operator; in fact, when used alone they are identical:

```
retrieve Full (Name = f1.Name)
where a.Name = "Tom" and f1.Rank = "Full"
when f extend start of a
```

Example 6. Version 2 of: Who were the full professors when Tom was promoted to associate?.

The **overlap** operator may be thought of as a temporal *and* operator, in that it is true when *both* arguments are true: the predicate

(a overlap b) precede c

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

(a extend b) precede c

is true when the period extending to the end of a and the end of b precedes the start of c. **overlap** and **extend** are commutative; precede is not.

The *valid clause* serves the same purpose as the target list: specifying the value of a domain in the derived relation. In this case, the domain in question is the implicit time domain. 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 domain.

```
retrieve AssociatePromotions (Name = a.Name)
valid at start of a
```

Example 7. When were the associate professors promoted to this rank?.

In this query, the underlying relation, Associates, is

an interval relation. One time value, the start time, was selected as the time value in the derived (event) relation. The valid clause contains an *e-expression*, also syntactically similar to path expressions. E-expressions include the operators **start of**, **end of**, **overlap**, **extend**, and **precede**. The boolean binary operators **and** and **or** are *not* allowed, since they introduce ambiguity as to which time value is desired.

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

```
range of f1 is faculty
range of f2 is faculty
range of f3 is faculty
retrieve Stars (Name = f1.Name)
valid from start of f1
      to start of f3
where f1.Name = f2.Name and f2.Name = f3.Name
when (f1 overlap a) and (f2 overlap a)
      and (f3 overlap a)
```

Example 8. Who got promoted from assistant to full professor while other faculty remained at the associate rank?.

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

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

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

Example 9. start of *in concert with* overlap.

specifies that the time value returned should be the first instant when both tuples are valid.

The primary difference between path expressions, temporal predicates, and e-expressions is

- path expressions specify *constraints* on the allowed ordering of events;
- temporal predicates denote a *boolean* value, indicating whether the events were ordered as specified; and
- e-expressions denote one of the *time values* involved in the expression, depending on the actual order of occurrence of the events.

Path expressions were designed for use in concurrent programs such as operating systems; temporal predicates and e-expressions were defined solely for use in TQuel.

As with other languages, there are several ways to write most queries. The **and** operator can considerably simplify matters:

```
retrieve Stars (Name = f1.Name)
valid from start of f1
      to start of f3
where f1.Name = f2.Name and f2.Name = f3.Name
when (f1 and f2 and f3) overlap a
```

Example 10. Same as the previous example.

In keeping with the path expression origins of temporal predicates and e-expressions, the keyword "**overlap**" may be abbreviated with a comma, "**precede**" may be abbreviated with a semicolon, and "**or**" may be abbreviated with a vertical bar. Since non-temporal domains are designated by "<tuple-variable> . <domain>", the prefix unary operators "**start of**" and "**end of**" may be replaced by the postfix operators "**.start**" and "**.stop**".

```
retrieve Stars (Name = f1.Name)
valid from f1.start to f3.start
where f1.Name = f2.Name and f2.Name = f3.Name
when (f1 and f2 and f3) , d
```

Example 11. Same as the previous example.

The operator precedence order, from highest to lowest, is the unary operators ("**start of**", "**end of**"), followed by the temporal binary operators ("**extend**", "**overlap**", "**precede**"), followed by the logical binary operators ("**and**", "**or**"). Operators of equal precedence are left associative. The appendix includes the complete BNF of the TQuel retrieve statement, except for the abbreviations mentioned previously.

4. AGGREGATE FUNCTIONS

Quel uses the aggregate operators **count**, **sum**, **avg**, **min**, **max**, and **any** (the value is 1 if any tuples satisfy the qualification) to aggregate a computed expression over a set of tuples. The argument of such an operator can be either a single tuple variable or any expression involving constants, arithmetic operators, or domains of a single relation. The argument of the aggregate operator may be qualified by an internal where clause:

```
retrieve TODSpapers (Number =
      Count(s where s.Journal = "TODS"))
```

Example 12. How many papers were submitted to TODS?.

This query contains a *simple aggregate*, which evaluates to a single scalar value. *Aggregate functions*, on the other hand, partition the set of qualifying tuples into groups, each of which is assigned a value for the expression.

```
retrieve AssocPapers (Name = a.Name,
      PaperCount = Count(s by s.Name)
where a.Name = s.Name
```

Example 13. How many papers were written by each associate professor?.

Operationally, **count** partitions the tuples into groups by name, then associates with each tuple in

the group the cardinality of the group. Each tuple receives the same value.

Aggregate operators are more complicated in TQuel, due to the time-varying behavior of relations. Aggregate operators on event relations are *cumulative,* in that they take all previously valid tuples into account in their computation. For instance, the **count** operator in the last example would count the number of (submission) events which had occurred. The AssocPapers relation has a value of 1 from 11-79 to 12-82, and a value of 2 from 12-82 to 3-84.

There are two versions of aggregates on interval relations, the cumulative and *instantaneous* versions. The **countC** operator is used to indicate the cumulative version, which works exactly as it does on event relations. The result of the (instantaneous) **count** operator

```
retrieve CurrentAssociates (Number = count(a.Name))
```

Example 14. How many associate professors were there at any point in the past?.

may be fairly oscillatory, but

```
retrieve CurrentAssociates
    (Number = countC(a.Name))
```

Example 15. How many faculty members have been promoted to associate professor?.

must increase monotonically over time.

The **avgC** operator is slightly more complicated, since it takes the length of time the tuple was valid into account when computing the average. The value of the argument of the **avgC** operator is weighted by the duration of the tuple, and intervening intervals (when no tuple is valid) are treated as tuples with a value of 0 for the argument.

```
retrieve TenuredRatio
    (Value = avgC(f.Name
                where f.Rank = "Associate"
                   or f.Rank = "Full"))
```

Example 16. How many tenured faculty were there, on average?.

TenuredRatio is also a temporal relation, with values ranging from 0, for the period 9-71 through 12-76, when there were no tenured faculty, to .58 on 3-84. The average will reach 1.0 in one more year, when the two tenured faculty will counteract the four year period when there were no tenured faculty.

Note that the presence of an aggregate operator in a retrieve statement automatically implies that the resulting relation will be an interval relation. The **valid at** clause may be used to specify that an event relation is to be derived. The conversion from single event relations to interval relations is handled by the **extendC** aggregate operator (not to be confused with the **extend** operator found in temporal predicates and e-expressions), which extends an event to an interval stretching to the next event. It is cumulative since the derived interval depends on the preceding event.

5. DEFAULTS

The defaults assumed in the language are an important aspect of the definition. The defaults for the additional clauses in TQuel should be natural to the user. If only one tuple variable (say, A) is used, and it is associated with an interval relation, then the defaults are

```
valid from start of A to end of A
when true
```

Example 17. Defaults for one interval tuple variable.

These defaults say that the result tuple is to start when the underlying tuple started and stop when the underlying tuple stopped. When an event relation is associated with the one tuple variable, the default is

```
valid at A
when true
```

Example 18. Defaults for one event tuple variable.

specifying simply that the result tuple was valid at the same instant the underlying tuple was valid. The first TQuel query given,

```
retrieve into Associates (Name = f.Name)
where f.Rank = "Associate"
```

Example 19. List the associate professors.

has the following default clauses,

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

Example 20. The previous query, with defaults.

When two or more tuple variables are used, the situation is more complex. Let us assume initially that all the tuple variables are associated with interval relations. The retrieve statement with defaulted temporal constructs looks identical to a standard Quel retrieve statement; thus it should have an identical semantics. An Ingres database is not temporal; instead, it advances in discrete jumps. Whenever a relation is updated, the "clock" advances, and the database is assumed consistent at the new time. Hence, the tuples participating in a retrieval are all valid at the time the query is executed. Extending this semantics to a temporal database is now straightforward: the result tuple is valid at all the points in time when *all* the underlying tuples were valid. Thus, if the tuple variables t_1, t_2, ... , t_k are involved in the query, then the default temporal clauses are

valid from start of (t_1 overlap ... overlap t_k)
to end of (t_1 overlap ... overlap t_k)
when (t_1 overlap ... overlap t_k)

Example 21. Defaults for several interval tuple variables.

The **valid from** clause specifies that the result tuple is to start the instant all the underlying tuples are valid; the **valid to** clause specifies that the result tuple is to end as soon as any underlying tuple is no longer valid. The **when** clause states that all the tuples should overlap each other to some extent. If a particular tuple variable t_i is associated with an event relation, simply replace 't_i **overlap**' in the above clauses with 't_i **extend**'.

When aggregate operators are used in interval relations, the decision needs to be made whether to consider the instantaneous or cumulative version to be the default. An argument similar to the one above concerning multiple tuple variables concludes that the instantaneous version more closely preserves the semantics of standard Quel. Hence the **count** operator will be the instantaneous version; **countC** must be used if the cumulative version is desired.

6. STATUS

Significant progress has already been made on both the theoretical and practical issues involved in introducing time into an existing, static, calculus based relational query language. The semantics of the entire TQuel retrieve statement, including aggregates and indeterminacy, has been informally specified. A formal semantics based on the tuple calculus [Ullman 1982] has been developed for the language with indeterminacy but without aggregates [Snodgrass 1984]. The semantics is relatively simple, enabling the extensions necessary to formalize the remainder of the language. Given the defaults discussed in the previous section, it is possible to show that the semantics reduces to the standard Quel semantics when applied to a static database slice (all tuples valid at a particular point in time) of the HDB. Work on the formalization and implementation of aggregates is proceeding [Gomez 1984]. Extending the other Quel statements to operate on an HDB is also an important area for future research.

In the course of the work described in [Snodgrass 1982], a compiler and runtime system for a subset of TQuel were implemented. The compiler produces an *update network* for each TQuel retrieve statement. An update network is essentially an executable parse tree of the equivalent relational algebraic expression for the query. However, the algorithms of the relational operators, while performing standard functions such as join and select, are nevertheless quite different from their static counterparts, since they have been tuned for the dynamic incremental updating of temporal relations. The system runs on a Vax under Berkeley Unix [Ritchie & Thompson 1974]. The parser was derived from the Ingres front end, and thus benefits from the functions provided by the Ingres terminal handler, particularly the extensive macro facilities. The system consists of a compiler that generates an update network, and an update network interpreter. Both components were written in FranzLisp [Foderaro 1980]; further developments, including a more robust compiler as well as an update network compiler, will be written in C [Ritchie et al. 1978].

7. BIBLIOGRAPHY

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

[Foderaro 1980] Foderaro, J.K. *Franz Lisp Manual.* Opus 33b ed. UC Berkeley, 1980.

[Gomez 1984] Gomez, S. L. *A Study of Aggregates in Temporal Relational Databases.* Computer Science Department, University of North Carolina at Chapel Hill 1984. In progress.

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

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

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

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

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

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

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

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

[Montague 1973] Montague, R. *The proper*

treatment of quantification in ordinary English, in Approaches to Natural Language. Dordrecht, Holland: D. Reidel Publishing Co., 1973.

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

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

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

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

[Ritchie & Thompson 1974] Ritchie, D.M. and K. Thompson. *The Unix Time-Sharing System. Communications of the Association of Computing Machinery,* 17, No. 7, July 1974, pp. 365-375.

[Ritchie et al. 1978] Ritchie, D.M., S.C. Johnson, M.E. Lesk and B.W. Kernighan. *Unix Time-Sharing System: The C Programming Language. Bell System Technical Journal,* 57, No. 6 (1978) pp. 1991-2019.

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

[Shaw 1980] Shaw, A.C. *Software Specification Languages Based on Regular Expressions.* Springer-Verlag, 1980.

[Snodgrass 1982] Snodgrass, R. *Monitoring Distributed Systems: A Relational Approach.* PhD. Diss. Computer Science Department, Carnegie-Mellon University, Dec. 1982.

[Snodgrass 1984] Snodgrass, R. *Formal Semantics of a Temporal Query Language.* 1984. (Submitted for publication.)

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

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

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

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

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

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

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

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

APPENDIX: BNF OF THE TQUEL RETRIEVE STATEMENT

This appendix lists the syntax for the TQuel retrieve statement. Since TQuel is a strict superset of Quel, all legal Quel retrieve statements are also legal TQuel retrieve statements. The following non-terminals are not included in the syntax description because they are identical to their Quel counterparts.

<bool expression>	returns a value of type boolean
<expression>	returns a value of type integer, string, floating point, or temporal
<integer>	an integer constant
<domain>	the name of a domain
<relation>	a relation name
<string>	a string constant
<tuple variable>	the name of a tuple variable

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

<TQuel retrieve> ::= <retrieve head> <retrieve tail>

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

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

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

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

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

<is> ::= **is** | = | **by**

<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> ::= ϵ | **at** <e-expression>

<e-expression> ::= <element>
| <e-expression>
| **start of** <e-expression>
| **end of** <e-expression>
| <e-expression> **precede** <e-expression>
| <e-expression> **overlap** <e-expression>
| <e-expression> **extend** <e-expression>
| (<e-expression>)

<element> ::= <tuple variable> | <string> | <integer>

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

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

<when clause> := ϵ | **when** <temporal predicate>

<temporal predicate> ::= <element>
| **start of** <temporal predicate>
| **end of** <temporal predicate>
| <temporal predicate> **precede** <temporal predicate>
| <temporal predicate> **overlap** <temporal predicate>
| <temporal predicate> **extend** <temporal predicate>
| <temporal predicate> **and** <temporal predicate>
| <temporal predicate> **or** <temporal predicate>
| (<temporal predicate>)