A Comparison of the Temporal Expressiveness of Three Database Query Methods

Amar K. Das and Mark A. Musen
Section on Medical Informatics, Stanford University School of Medicine
Stanford, California 94305-5479
das@camis.stanford.edu

Time is a multifaceted phenomenon that developers of clinical decision-support systems can model at various levels of complexity. An unresolved issue for the design of clinical databases is whether the underlying data model should support interval semantics. In this paper, we examine whether interval-based operations are required for querying protocol-based conditions. We report on an analysis of a set of 256 eligibility criteria that the T-HELPER system uses to screen patients for enrollment in eight clinical-trial protocols for HIV disease. We consider three data-manipulation methods for temporal querying: the consensus query representation Arden Syntax, the commercial standard query language SQL, and the temporal query language TimeLineSQL (TLSQL). We compare the ability of these three query methods to express the eligibility criteria. Seventy nine percent of the 256 criteria require operations on time stamps. These temporal conditions comprise four distinct patterns, two of which use interval-based data. Our analysis indicates that the Arden Syntax can query the two non-interval patterns, which represent 54% of the temporal conditions. Timepoint comparisons formulated in SQL can instantiate the two non-interval patterns and one interval pattern, which encompass 96% of the temporal conditions. TLSQL, which supports an interval-based model of time, can express all four types of temporal patterns. Our results demonstrate that the T-HELPER system requires simple temporal operations for most protocol-based queries. Of the three approaches tested, TLSQL is the only query method that is sufficiently expressive for the temporal conditions in this system.

TEMPORAL DATA MODELS

Time is a fundamental property of the clinical information that health-care providers record and use during patient care. The answers to such questions as "when did the problem start?," "how long did it last?," or "how many times has it occurred?" place a patient's chief concern in a temporal context that is essential to reasoning about diagnosis, treatment, and prognosis. Modeling the temporal semantics of clinical data, however, is not effortless, because time is a complex phenomenon. Kahn [1] has noted that a multitude of formalisms is available for the design of temporal data models in clinical decision-support systems. System developers, consequently, must choose those features of time that are most appropriate for their applications.

Developers of clinical data-management systems, for example, have represented time traditionally as discrete, precise, and linear. These choices permit database systems to map timepoints to an integer-based representation, which is both efficient for storage and simple for manipulation. For protocol-based decision-support systems, two issues still remain unresolved for the design of clinical databases: (1) whether the data scheme should support instant stamps, interval stamps, or both, and (2) which temporal operations are necessary for a query method to verify protocol-based conditions.

Most databases, including those of the HELP system [2] and of the Regenstrief Medical Information System [3], continue to use the temporal model proposed in 1975 for the Time Oriented Database (TOD) [4]. TOD stamps each patient parameter with the time of the latter's occurrence during a clinical event (such as a visit or a laboratory test). This temporal scheme creates a *cubic view* of data, along the dimensions of (1) time, (2) patient identification, and (3) parameter type; temporal operations on this representation permit selection of data based on temporal ordering or during a window of time. The HELP and the Regenstrief systems have shown that query languages that can evaluate these types of patterns are useful for a variety of simple protocol-based alerts.

An important limitation of the TOD model is the latter's inability to support interval semantics: The model cannot represent explicitly interval-based data (such as medication dosages and patient problems), and cannot formulate queries about interval-based patterns (such as context and duration). Recently proposed temporal models, such as those of the TNET system [5] and the Chronus system [6], overcome this problem by providing interval semantics for both the storage and the manipulation of clinical data. Although these two systems have shown that their querying capabilities are required for certain types of clinically relevant temporal conditions, many current query methods do not support interval semantics. The Arden Syntax [7], for example, does not include operators on interval-based data in its ASTM-standards representation of database alerts; Arden developers argue that the number of such temporal constructs would complicate the development of a consensus query method [8]. Certain developers of decision-support systems [9] have examined, for their data-management needs, commercial standards for query languages such as the Structured Query Language (SQL) for relational databases. SQL does not support intervals

0195-4210/95/$5.00 © 1995 AMIA, Inc.

as a primitive data type, but does have general comparison operators that can manipulate timepoints.

In this paper, we address the question of whether interval-based query methods are necessary temporal semantics for a protocol-based decision-support system. We create a test suite of temporal conditions from a set of eligibility criteria from clinical-trial protocols that are encoded in the T-HELPER system [10]. We classify the temporal conditions into basic temporal patterns, and we use the test suite to compare the temporal expressiveness of three query methods — the Arden Syntax, SQL, and TimeLineSQL (the query language of the Chronus system [6]). Our analysis indicates that temporal conditions are common in this set of protocol-based conditions; however, complex temporal patterns that require a general interval-based query language occur infrequently. We discuss the implication of these results for the design of a general query method for protocol-based decision support.

METHODS FOR TEMPORAL QUERYING

Arden Syntax

The Arden Syntax [7, 8] is the result of a multi-institutional project to share the decision logic found in various database-surveillance programs, such as the HELP and the Regenstrief systems. The primary purpose of the Arden Syntax is not to define a database query language, but rather to create a specification language that encapsulates situation–action rules as medical logic modules (MLMs). The various systems that implement MLMs have legacy databases with heterogeneous temporal and coding representations; the creation of sharable MLMs thus entails a standardized syntax to describe queries. Our inclusion of the Arden Syntax as a candidate query method in this study allows us to evaluate the common temporal querying capabilities of well-known legacy databases.

An MLM has a *logic slot* that specifies a decision rule, and a *data slot* that uses database queries to instantiate variables used in the rest of the MLM. The data slot consists of system-specific and system-independent components. The Arden Syntax thus permits both variability of data descriptions among existing decision-support systems and standardization of common query constructs (such as temporal conditions). The general form of a data-slot query is

```
var := READ [aggregation] ({body} [WHERE IT
OCCURRED time_constraint])
```

where `var` is the variable that the query instantiates; `aggregation` is an aggregation operator (such as `LAST`); `body` is a written description of the data elements that a user must map to a system-specific query language; and `time_constraint` is one of the following constructs: `BEFORE` a certain temporal instant, `AFTER` a certain temporal instant, or `WITHIN` a certain range of time [7, 8]. (The portions of a syntax in brackets are optional.)

The Arden Syntax assumes that the underlying database associates each stored parameter with a single primary time. This temporal model is similar to that of TOD: Only discrete temporal instances are associated with clinical data in the underlying data scheme, and queries can select data based on temporal ordering or by occurrence in a window of time. Unlike the TOD model, the Arden Syntax temporal model does allow the storage and manipulation of a duration parameter. The duration parameter, however, is not anchored to any point in time; for example, the database can store the patient's age as a duration of 37 years, but does not need to represent an exact birthdate. The Arden Syntax thus does not support time intervals as primitive temporal elements.

Structured Query Language

Like the Arden Syntax, SQL is a query syntax maintained by a national standards organization (ANSI) [11]. Unlike the Arden Syntax, SQL is based on a formal model of data — the relational data model. During the past decade, SQL has emerged as the leading industry-wide standard for a database query language. One of the primary reasons for the ascendancy of SQL is that the relational model has well-defined data-definition and data-manipulation methods [12]. In the relational model, the data-definition language is cast in terms of *relations*, which are two-dimensional tables in which the columns are called *attributes* and the rows are called n-*tuples*. The data-manipulation language of the relational model comprises a set of operators on relations; these relational operators define the semantics for the constructs in the SQL syntax. A retrieval statement in SQL has the following clauses:

```
SELECT attribute_list [aggregation_list]
FROM   relation_list
[WHERE [search_condition | subquery]]
[GROUP BY attribute_list]
```

Such a statement instructs the query method (1) to take the product of relations listed in the `FROM` clause, (2) to select any tuples that satisfy the Boolean conditions in the `search_condition` of the `WHERE` clause, and (3) to return those attributes specified in the `attribute_list` of the `SELECT` clause [12]. The `GROUP BY` clause allows grouping of the results by unique values (such as patient identification); the database can derive from each group the result of any aggregation operators (such as `MIN`) in `aggregation_list`. An important feature of SQL is that every query returns data in a relational format that can be queried further; the `subquery` construct permits nesting of multiple queries in one retrieval statement.

The SQL syntax illustrates that the language provides no specific clauses for temporal operations. The underlying relational representation does permit the storage of both instant and interval stamps as attributes, and the search conditions in the `WHERE` clause can make simple comparison operations (such as >, <, and =) on time stamps in the relations. SQL does not support, however, nonrelational temporal operators (such as ordinal selection or concatenation). The language also

does not give special status to the temporal attributes in the results of a query. For example, if a user does not specify, in the list of attributes in the SELECT clause, the time-stamp attribute(s) stored in a relation, the resulting set of data will not have a temporal dimension. Therefore, a major limitation to the use of SQL as a temporal query method is that a user must formulate queries in a manner that is consistent with an application's temporal model.

TimeLineSQL

Many researchers [13, 14] have attempted to overcome the lack of temporal semantics in SQL by defining novel temporal extensions to the relational model. Based on such prior research, we have designed a temporal relational model tailored to the temporal querying needs of clinical decision-support systems [15]. The result of our research is (1) a data-definition language that defines a canonical representation of temporal data as interval-stamped relations (*histories*), and (2) a data-manipulation language that consists of temporal operators on data in histories. To implement our approach, we have created the Chronus system, a temporal-query server that is built on top of a commercial relational database (Sybase) and that supports our model's interval-based operators in a query language that we call TimeLineSQL (TLSQL) [6]. A retrieval statement in TLSQL has the following syntax:

```
GRAIN   grain_size
SELECT  [ordinal] [CONCATENATED]
        attribute_list [aggregation_list]
        [INTO history]
FROM    history [, history]
[WHEN   [temporal_join |
        temporal_comparison]]
[WHERE  nontemporal_condition]
[GROUP BY  [attribute | grain_size]]
```

TLSQL is based on the simple framework of SQL. Unlike SQL, however, TLSQL permits the formal specification of temporal queries through six new constructs:

1. A GRAIN term to set the granularity level (such as day) at which the query performs all temporal comparisons on timepoints (for precedence, we can compare only timepoints of the same grain size)
2. An ordinal term to define an ordinal selection (such as FIRST, SECOND, or LAST)
3. A CONCATENATED term to undertake the temporal concatenation of intervals with equivalent nontemporal values
4. Temporal_join conditions in the WHEN clause to allow the temporal intersection or temporal difference of the time stamps of data in two histories
5. Temporal_comparison conditions in the WHEN clause to specify instant- or interval-based comparisons between time stamps
6. Grouping by grain_size (such as MONTH or YEAR) to permit temporal grouping of data

In addition to these syntactic changes to SQL, TLSQL uses the semantics of the interval operators to ensure that the result of any query on histories also will be in the history format. Interval stamps, consequently, are returned *always* from a query on a single history, and are derived *automatically* from a temporal join of two histories. A user can manipulate the results of one query in subsequent retrieval statements by saving the results into a new history (as specified in the INTO clause). The primary limitation of the use of TLSQL as a temporal query method is that the data scheme of the underlying database must support the interval-based representation. This representation requires that we store *instant-based* data as intervals with equivalent endpoints and *nontemporal* information as intervals that exist at all time periods. The Arden Syntax and SQL query methods do not require such restrictions on the representation of temporal data.

STUDY DESIGN

Selection of Test Suite

Our motivation to find a temporal query method suitable for protocol-based decision support derives from the needs of the THERAPY-HELPER (T-HELPER) clinical workstation [10] — an advice system for protocol-directed care of patients who have HIV disease. One of the goals of the T-HELPER system is to determine whether decision support can improve clinic-trial enrollment in community health-care settings. The eligibility-determination program (EDP) [16] in T-HELPER provides a method to screen patients for eligibility in HIV or AIDS clinical trials that are ongoing at two public hospitals in the San Francisco Bay area. The EDP provides advice on only eight randomly selected clinical-trial protocols at these two sites; the remaining protocols at the two sites provide baseline accrual rates for comparison.

The EDP encodes the eligibility criteria for the eight clinical trials as templates in an expert-system shell, and uses rule matching to instantiate the templates with patient data. Each time that the EDP must verify a patient's eligibility status, the program loads into its memory-resident fact base the patient's record from the central database (a Sybase SQL server). If the medical record is large, such data transfer is time consuming. Furthermore, the verification of the protocol criteria by the EDP requires the programmer to implement temporal querying methods for the fact base. The developers of the EDP would prefer to instantiate these templates by queries to the database. We thus examine which of three methods for temporal querying — the Arden Syntax, SQL, and TimeLineSQL — is sufficient for the task of screening patients for protocol enrollment.

Classification of Conditions

To create a test suite of queries for our comparison, we use the templates encoded in the EDP. We define an *elementary condition* as any condition that does not contain a conjunction or disjunction. We define a

temporal condition as any elementary condition that makes comparisons or computations on the time stamp(s) of a clinical parameter. We subdivide the set of temporal conditions into distinct temporal patterns, and then formulate for each temporal pattern a matching query statement in the candidate query methods.

Suitability of the Test Suite

This set of conditions is an appropriate test suite for the evaluation of a query method. We did not select the set of eligibility criteria based on our ability to instantiate the criteria by a database query method. If we had chosen conditions that a decision-support system could verify already through database queries, we would have biased the test suite to reflect the capabilities of that query method. The semantics of the conditions in the test suite are also unambiguous: The developers of clinical-trial protocols attempt to reduce variability in the interpretation of the protocols at different clinical sites by defining the criteria precisely.

RESULTS

Temporal Patterns

From the developers for the EDP, we learned that the program did not have templates for 20 criteria from the eight encoded protocols. Twelve of these criteria used a parameter that the developers could not define properly (such as "the class of surgical interventions that affect drug absorption"), whereas the other eight criteria contained a complex temporal pattern not readily modeled in a template format (such as "no combination therapy for greater than two weeks with 2 or more agents active against MAC more than 1 month before enrollment"). The latter set of unencoded criteria all qualified as temporal conditions by our definition, and the clinical parameters that they required were in the database. None of the conditions in the former set had a temporal component, and these 12 poorly defined criteria were excluded from our analysis.

By combining those elementary conditions that are encoded in the EDP with the unencoded temporal conditions, we created a test suite of 256 elementary conditions. We examined this test suite and found 202 conditions that satisfied the definition of a temporal condition. The frequency of temporal conditions was thus 79% (see Table 1). We reviewed these temporal conditions, and found four types of temporal patterns:

1. **Temporal duration.** This condition checks whether the duration of time between a reference timepoint and the instant stamp associated with a clinical datum is greater or less than a certain length of time. Figure 1 provides a visual representation of this temporal pattern. The protocols used this condition to make queries only on a patient's age (which is the duration between the birthdate value and the current time). The temporal-duration pattern represented 5% of the temporal conditions.
2. **Temporal window.** This condition finds the most recent occurrence of a patient datum within a window of time prior to a reference timepoint. The database stores each parameter used in this type of condition as an instant-based datum. This temporal pattern is shown in Figure 2. The temporal-window pattern involved 49% of the temporal conditions.
3. **Prior presence.** This condition determines whether a clinical parameter with an interval time stamp was ongoing during a window of time prior to a reference timepoint. As depicted in Figure 3, this temporal condition must verify three different interval relations. We noted this temporal pattern in 42% of the temporal conditions.
4. **Temporal concatenation.** This condition requires the concatenation of adjacent interval stamps of a parameter into larger intervals. For example, the condition may require that all adjacent interval times of a drug given at different dosages be concatenated to create periods during which the medication was given regardless of the dosage. After concatenation, the condition requires the duration of the resultant interval. Temporal concatenation is needed in 4% of the temporal conditions.

Table 1. Frequency of temporal patterns in a set of eligibility criteria.

Pattern Type	Frequency (%)
Nontemporal patterns	21
Temporal patterns	79
Temporal duration	4
Temporal window	39
Prior presence	33
Temporal concatenation	3

Query Method Expressiveness

Each of the temporal conditions in the test suite satisfies one and only one of the four temporal patterns. As a result, we can choose for each pattern a representative condition, and can illustrate with a single example the ability of a query method to support that type of pattern. We do not include the nontemporal conditions in our analysis, since all three query languages can check these criteria readily.

In the following examples, the queries return for each condition the parameter value and the time stamp(s). The EDP uses a reference time to evaluate certain types of temporal conditions; the reference time is equal to the time stamp of the patient's most recent datum. We use the variable `r_time` to refer to this value in the example queries. We also use patient identification (`PID`) 2207 to represent an example patient.

Temporal Duration. An example pattern of this type is in ACTG protocol 177, and states that a patient should be "greater than or equal to 13 years old." Given a reference time, we can express this condition in the Arden Syntax as

```
data: birth_date := READ ({'birth_date'})
logic: IF  r_time - birth_date >= 13 THEN
CONCLUDE TRUE;
```

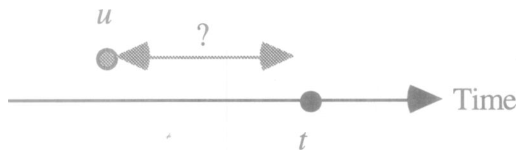

Figure 1. The temporal-duration pattern finds the length of time between a reference time t and the time u of a clinical datum.

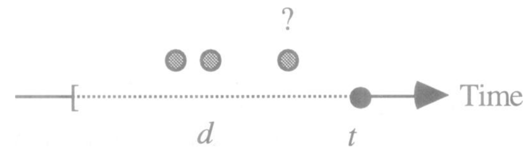

Figure 2. The temporal-window pattern selects the last occurrence of a instant-stamped patient datum within a span of time d prior to a reference time t.

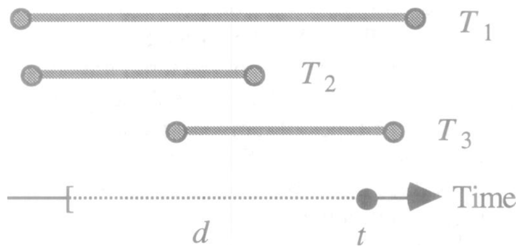

Figure 3. The prior-presence pattern determines whether the interval stamp of a datum overlaps a span of time d prior to a reference time t. Three separate interval comparisons (as represented by T_1, T_2, and T_3) can instantiate this pattern.

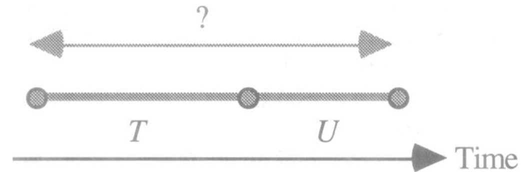

Figure 4. The temporal-concatenation pattern requires the union of adjoining or overlapping intervals, T and U. This pattern returns the duration of the resultant interval.

The Arden Syntax cannot express this condition in the data slot alone, since it must specify how to calculate a duration in the logic slot. In the SQL syntax, we would express the condition as

```
SELECT birth_date
FROM   demographics
WHERE  pid = 2207 AND DATEDIFF(year,
       birth_date, r_time) >= 13
```

For the SQL query, we assume that the parameter is in the `demographics` relation. The `DATEDIFF` function in the `WHERE` clause returns the length of time between the final two operands at a grain size that is defined by the first operand. In TLSQL, we can encode this pattern in a manner similar to SQL:

```
GRAIN  year
SELECT birth_date
FROM   demographics
WHEN   DURATION(birth_date, r_time) >= 13
WHERE  pid = 2207
```

The `DURATION` function in TLSQL is equivalent to the `DATEDIFF` function in SQL; the value of grain size in the `GRAIN` clause determines the granularity at which the resulting value is returned.

Temporal Window. ACTG protocol 268 provides a typical temporal-window pattern: "The most recent platelet count within the past 60 days should have value greater or equal to 50,000." In the Arden Syntax, we create a new variable `start_time` (defined outside the data-slot syntax) to be equal to 60 days prior to the reference time. The query in the Arden Syntax is as follows:

```
data: platelet_count := READ LAST ({'platelet
count'} WHERE IT OCCURRED WITHIN start_time
TO r_time)
logic: IF platelet_count > 50000 THEN
CONCLUDE TRUE
```

As is true with the temporal-duration pattern, we cannot specify in Arden the complete temporal condition in the data slot alone. SQL can express the condition in a retrieval statement with a subquery (which returns the all timepoints within the temporal widow):

```
SELECT time_stamp, result
FROM   laboratory_tests
WHERE  pid = 2207 AND result > 50000
       AND type = 'platelet count'
       AND time_stamp =
       (SELECT  MAX(time_stamp)
       FROM     laboratory_tests
       WHERE    time_stamp >
                DATEADD(day, -60, r_time)
                AND time_stamp < r_time)
                AND pid = 2207 AND
                result > 50000 AND
                type = 'platelet count'
```

In SQL, we can use the `DATEADD` function to create a timepoint value equal to 60 days prior to the reference time. In TLSQL, the `ADDGRANULE` function has the same capabilities as the `DATEADD` function in SQL. TLSQL does not have the subquery capability of SQL, but can express the condition in two query statements — the results of the first query are saved as an intermediary result that is used by the second query. The syntax of the TLSQL statement is as follows:

```
GRAIN  day
SELECT result INTO tmp
FROM   laboratory_tests
WHEN   start_time DURING
       [ADDGRANULE(-60, r_time), r_time]
```

```
SELECT LAST result
FROM   tmp
WHERE  pid = 2207 AND type = 'platelet count'
       AND result > 50000
```

Prior Presence. An example prior-presence pattern is found in CCTG protocol 553, and requires the patient to have "no treatment with the fluconazole medication at a dose greater than 200 mg/d within the past 30 days." The T-HELPER database stores interval stamps for medication data. The Arden Syntax does not have the ability to express the start and stop times of interval-based data, so it cannot express this pattern. In SQL, we specify the condition as

```
SELECT start_time, end_time, drug_name
FROM   medication
WHERE  pid = 2207 AND drug_dose > 200 AND
       drug_name = 'fluconazole' AND
       ((start_time >
       DATEADD(day, -60, r_time) AND
       start_time < r_time) OR (end_time >
       DATEADD(day, -60, r_time) AND
       end_time < r_time) OR (start_time <
       DATEADD(day, -60, r_time) AND
       end_time > r_time))
```

The query syntax in TLSQL is similar, but can express all temporal comparisons as one interval comparison:

```
GRAIN  day
SELECT drug_name
FROM   medication
WHEN   [start_time, end_time] OVERLAPS
       [ADDGRANULE(-60, r_time), r_time]
WHERE  pid = 2207 AND drug_dose > 200
       AND drug_name = 'fluconazole'
```

Temporal Concatenation. The temporal-concatenation pattern comprises the eight complex temporal criteria that the EDP does not support. For these conditions, the temporal union of adjacent or overlapping intervals is the first step of verification. Neither the Arden Syntax nor SQL supports concatenation or other interval computations. TLSQL can concatenate intervals readily, and can use the resulting intervals for further temporal querying [6].

ACTG protocol 177 provides an example temporal-concatenation pattern that requires a patient to have a "documented history of a positive PPD skin test, without subsequent treatment with more than 1 month of any antimycobacterial medication." Such a criteria requires two queries: The first query concatenates the adjacent or overlapping intervals of any antimycobacterial medication that the patient was taking, and the second query verifies the duration of the concatenated intervals and the relation of the intervals to an episode of tuberculosis in the `problems` relation:

```
GRAIN  day
SELECT CONCATENATED drug_class INTO tmp
FROM   medication
WHERE  pid = 2207 AND drug_class =
       'antimycobacterial medication'
```

```
SELECT tmp.drug_name
FROM   tmp, problems
WHEN   [problems.start_time,
       problems.end_time] BEFORE
       [tmp.start_time, tmp.end_time] AND
       DURATION(tmp.start_time,
       tmp.end_time) < 30
WHERE  problems.name = 'PPD_TB'
       AND problems.value = 'yes'
       AND tmp.pid = problems.pid
```

Our results indicate that the Arden Syntax can support two temporal patterns (54% of the temporal conditions). SQL is more expressive than is the Arden Syntax: SQL can encode three types of temporal patterns (96% of the temporal conditions). TLSQL can express the same patterns as does SQL, but also has the ability to query the remaining infrequent complex temporal patterns that involve concatenation.

DISCUSSION

An undetermined issue for temporal data management is whether or not a general query language should support operations on intervals. Our previous research [6, 15] has indicated that both instant-based models (such as that of Arden) and the relational model that underlies SQL are limited in their ability to express complex interval-based queries. We had not determined, however, whether such limitations hinder the use of these data models for the design of a general data-management technique. In this paper, we have provided an analysis of the set of temporal conditions needed by a protocol-based decision-support system. Our results indicate that interval-based queries are indeed common, but that complex temporal queries that require the capabilities of a general interval-based query method (such as TLSQL) occur infrequently.

We must consider our comparison of temporal query methods in the context of the goal to find a generalized query method for decision support. The Arden Syntax is currently the only specification language for sharing both the situation–action rules for protocol-based conditions and the database queries necessary to instantiate these rules. Yet, because Arden's implicit model of time supports only the temporal formalisms of legacy database systems, the data slot in the Arden Syntax cannot express the significant number of interval-based conditions found in protocol documents. Users of the Arden Syntax can overcome this limitation by writing in the *logic slot* rules for interval operations. Such specifications, however, create undesired variability in the temporal models across different sites. In addition, if users choose SQL to instantiate data for Arden rules, they will limit unnecessarily SQL's ability to formulate certain interval-based queries.

Because relational-database technology is increasingly widespread, SQL appears to be a viable option for a standard query method. Our results show that, on average, SQL could express all queries but one for each of the eight clinical-trial protocols. Unlike the Arden

Syntax, the SQL syntax does not provide clauses for temporal patterns — a deficiency that makes queries with temporal conditions awkward to express and difficult to understand. The lack of an underlying temporal model in SQL also requires the user to specify queries in a manner consistent with the temporal model of the application. Thus, the expression of temporal queries in SQL can create a significant data-engineering obstacle for the developer of a decision-support system.

TLSQL can express interval-based comparisons more succinctly than does SQL (for example, consider the queries for the prior-presence pattern). As we have demonstrated previously [6], these temporal constructs in TLSQL incur only a small additional cost to the search times in a relational database. The interval-based operators that we provide in TLSQL can serve also to extend the Arden Syntax, so that the latter method can support more than two temporal patterns.

We have built and validated a system (Chronus) that implements the TLSQL query method on top of relational databases. A current limitation of embedding Chronus in legacy systems is that the method requires the database schema to support our uniform representation of time. To overcome this problem, we are extending our temporal query method, so that it can map automatically data with heterogeneous temporal representations into the canonical interval-based format. Our extended system (which we call Synchronus) will have the ability, consequently, to query legacy SQL databases that support various time-stamping methods.

Although we continue to refine our temporal query methods, we do not consider the representation and manipulation of interval-based data an unresolved research issue that hinders the development of a general query method. The analyses reported in this paper and in prior studies [6, 15] demonstrate that decision-support systems require well-established interval semantics for querying temporal conditions. Our interval-based model of time provides TLSQL the flexibility to instantiate a wide variety of temporal patterns, including the complex interval-based conditions that the Arden Syntax and SQL cannot express.

Acknowledgments

We thank S. Tu for his comments on this research and L. Dupré for her editorial assistance in the preparation of the final paper. This work has been supported in part by grant HS06330 from the Agency for Health Policy and Research, and by grants LM05208, LM05708, and LM07033 from the National Library of Medicine. Dr. Musen is a recipient of an NSF Young Investigator Award.

References

1. Kahn, M.G. Modeling time in medical decision support programs. *Medical Decision Making*, 1991. 11:249–64.
2. Kuperman, G.J., Gardner, R.M., and Pryor, T.A. *HELP: A Dynamic Hospital Information System.* 1991, New York: Springer-Verlag.
3. McDonald, C.J., Tierney, W.M., Martin, D.K., and Overhage, J.M. The Regenstrief Medical Record System: 20 years' experience in hospital outpatient clinics and neighborhood health centers. *MD Computing*, 1992. 9:206–217.
4. Wiederhold, G., Fries, J.F., and Weyl, S. Structured organization of clinical data bases. *AFIPS NCC.* AFIPS. 1975, pp. 479–85.
5. Kahn, M.G., Fagan, L.M., and Tu, S. Extensions to the Time-Oriented Database model to support temporal reasoning in medical expert systems. *Methods of Information in Medicine*, 1991. 30:4–14.
6. Das, A.K., and Musen, M.A. A temporal query system for protocol-directed decision support. *Methods of Information in Medicine*, 1994. 33: 358–370.
7. Hripcsak, G., Clayton, P.D., Pryor, T.A., Haug, P., Wigertz, O.B., and van der Lei, J. The Arden syntax for medical logic modules. *Fourteenth Annual Symposium on Computer Applications in Medical Care.* Washington, DC. R.A. Miller (ed), IEEE Computer Society Press. 1990, pp. 200–204.
8. Hripcsak, G., Ludemann, P., Pryor, T.A., Wigertz, O.B., Clayton, P.D. Rationale for the Arden Syntax. *Computers and Biomedical Research*, 1994. 27:291–324.
9. Huff, S.H., Berthelson, C.L. and Pryor, T.A. Evaluation of an SQL model of the HELP patient database. *Fifteenth Annual Symposium on Computer Applications in Medical Care.* Washington, DC. P.D. Clayton (ed), McGraw-Hill. 1991, pp. 386–390.
10. Musen, M.A., Carlson, C.W., Fagan, L.M., Deresinski, S.C., and Shortliffe, E.H. T-HELPER: Automated support for community-based clinical research. *Sixteenth Annual Symposium on Computer Applications in Medical Care.* Baltimore, MD. M.E. Frisse (ed), McGraw-Hill. 1992, pp. 719–23.
11. Date, C.J. *A Guide to the SQL Standard.* 1989, Reading, MA: Addison-Wesley.
12. Ullman, J.D. *Principles of Database and Knowledge-Base Systems.* Vol. 1. 1988, Rockville, MD: Computer Science Press.
13. McKenzie, L.E., and Snodgrass, R.T. Evaluation of relational algebra incorporating the time dimension in databases. *ACM Computing Survey*, 1991. 23:501–543.
14. Tansel, A.U., Clifford, J., Gadia, S., et al. *Temporal Databases: Theory, Design, and Implementation.* 1993, Redwood City, CA: Benjamin/Cummings.
15. Das, A.K., Tu, S.W., Purcell, G.P., and Musen, M.A. An extended SQL for temporal data management in clinical decision-support systems. *Sixteenth Annual Symposium on Computer Applications in Medical Care.* Baltimore, M.D. M.E. Frisse (ed), McGraw-Hill. 1992, pp. 128–132.
16. Tu, S.W., Kemper, C.A., Lane, N.M., Carlson, R.W., and Musen, M.A. A methodology for determining patients' eligibility for clinical trials. *Methods of Information in Medicine*, 1993. 32: 317–325.