Published in final edited form as:
Int J Med Inform. 2003 April ; 70(1): 59–77.

Temporal query of attribute-value patient data: utilizing the constraints of clinical studies

Aniruddha M. Deshpande, **Cynthia Brandt**, and **Prakash M. Nadkarni***
Center for Medical Informatics, Yale University School of Medicine, PO Box 208009, New Haven, CT 06520-8009, USA

Summary

We describe an interface and architecture for ad hoc temporal query of TrialDB, a clinical study data management system (CSDMS). A clinical study focuses primarily on the effect of therapy on a group of patients, who have individually enrolled in a study at different times. Relative times (chronological offsets from the time of enrollment) are therefore more useful than absolute times when collectively describing therapeutic or adverse events. For logistic reasons, study parameter values are typically recorded at fixed relative times ('study events'), which serve as timestamps and can be used by CSDMS temporal query algorithms to simplify temporal computations. The entity-attribute-value model of clinical data storage, used by both CSDMSs and clinical patient record systems, complicates temporal query. To apply temporal operators, data for parameters of interest must first be transiently converted into conventional relational form, with one column per parameter.

Keywords

Temporal query; Entity-attribute-value databases; Clinical study data management systems; Clinical patient record systems

1. Introduction

Recording of parameters derived from history taking, clinical examination or investigations is an integral component in patient care. Because all such parameters are time-stamped, querying of patient data is greatly facilitated by the ability to allow temporal criteria to be specified in a query. Most research to date has focused on temporal query of clinical patient record systems (CPRSs), or data abstracted from CPRSs. Clinical study data management systems (CSDMSs), however, also form an important category of patient-related data. Typically, CSDMSs are large repositories, with the ability to record data from an arbitrary number of studies with an arbitrary number of parameters for each study.

In this paper, we describe the addition of temporal query capability to TrialDB [1,2], a generic, web-accessible CSDMS that is used by multiple departments at Yale University, the Vanderbilt University Cancer Center, and the National Cancer Institute-supported Cancer Genetics Network, a national consortium of investigators.

*Corresponding author. Fax: + 1-203-764-6717. prakash.nadkarni@yale.edu (P.M. Nadkarni).

2. Background

To apply temporal operations on data, every row of data should be associated with either a single time stamp (recording when the event associated with that row occurred), or two time stamps, representing the event's start and end. Data with one and two time stamps are called instant and period (interval) data, respectively. Period data are much more challenging computationally than instant data, which is why they have been of greater research interest. For example, Shahar and his colleagues have developed a system for the temporal abstraction of both instant and period clinical data (using domain-specific ontologies acquired from domain experts), which has been applied to several clinical domains [3-5].

In clinical medicine, however, the vast majority of data tend to be instant, though a few (such as IV infusions or course of radiotherapy) are periods. Because of this factor, temporal query engines for clinical data repositories such as Nigrin and Kohane's elegant DxTractor [6], which operate on 'instant' data exclusively, are quite viable in clinical settings. DxTractor, in fact, performs an essential operation that is quite typical of real-world temporal query systems; it permits the inference of periods from raw 'instant' data.

Artificial intelligence (AI) researchers were the first to consider computational aspects of temporal reasoning and temporal data query: a good review of these efforts is presented in Refs. [7,8]. In a classic paper [9], James Allen defined a set of binary temporal Boolean operators, which compare two periods in various ways. For example, the operator BEFORE (as in X BEFORE Y) returns True if the end of period X occurs before the start of period Y. In medical informatics, the Stanford group has been particularly active in clinical temporal research, producing an impressive series of papers [4,5,10-13].

Various attempts have been made to add temporal query support to mainstream relational database management systems (RDBMSs). A standardization effort led by Snodgrass yielded a proposal, TSQL2 [14], for a core set of additions to structured query language (SQL, the data manipulation language of RDBMSs). In 1995, efforts were made to amalgamate TSQL2's features into the forthcoming version of SQL, by adding a new component called SQL/Temporal [15,16]. While these efforts were approved in 1996 by ANSI (the American National Standards Institute), the International Standards Organization (ISO) has yet to vote on them: the current version of SQL (SQL-99) therefore lacks periods and temporal operations. Historically, moreover, even when ISO approves SQL changes, most database vendors are very slow to upgrade their DBMS engines. Temporal support within RDBMSs therefore still requires approaches that are unique to a particular research group, as opposed to being standards-based.

2.1. An introduction to clinical database architecture

Much clinically related temporal research published to date makes the simplifying assumption that the clinical data on which temporal operations will be performed has individual parameters of interest residing in their own distinct columns in one or more tables. In practice, this is not usually the case. Large-scale clinical databases store much of their clinical data using a generic or entity-attribute-value (EAV) data model [17]. EAV systems used in clinical medicine include the well-known HELP system [18,19], now commercialized as the 3M clinical data repository (CDR) [20] and the Columbia-Presbyterian CDR [21,22]. In EAV design, every fact is conceptually stored in a single table with three sets of columns: the entity: (patient ID plus timestamps recording event occurrence), the ID of an attribute or parameter, e.g. 'serum potassium,' and the value of the attribute, e.g. '4.5'. One row stores a single fact. In a conventional table that has one column per attribute, by contrast, one row stores a set of facts.

EAV design is appropriate when the number of clinical parameters that potentially apply to a patient (e.g. the thousands of parameters in medicine), is vastly more than those that actually apply to a given patient. All data in a production EAV system does not have to be stored in EAV form: patient demographics, for example, are typically stored conventionally. End-users as well as analytical programs take a conventional view of the data, and therefore a usable EAV system must create the illusion of conventional data storage. This illusion is achieved through a set of tables containing metadata, whose contents comprise a kind of 'data dictionary,' and user-interface code that drives off this metadata. Examples of CSDMS metadata include the definitions of individual parameters (and whether they represent instant or period events), the case report forms that group these parameters together, the study events, information about which case report forms are used in a study (and during which study events), and so on.

The above description of EAV structure is somewhat simplified. Instead of a single EAV table, some systems, including TrialDB, segregate EAV data by the data type of the value column, so that there are separate EAV tables for strings, integers, real numbers and so forth. In any case, the existence of EAV structure means that in order to answer temporal queries, there is an additional processing step involved compared with conventional data. Specifically, the data for the attributes of interest must be extracted from the general-purpose EAV tables and transformed transiently into conventionally structured columnar data before temporal operators can be applied. This naturally increases the processing time, but is unavoidable when one is operating on a database that uses a generic design.

The temporal research literature distinguishes between two kinds of time stamps that are attached to data items. Valid-time stamps record when an event occurred in the real world. Transaction-time stamps record when a data item was entered in the system, or when it was last changed. While transaction-time is important for particular purposes (e.g. recording audit trails when sensitive data is created and altered), the focus of clinical temporal research is on valid-time.

Many participants in clinical studies are healthy volunteers, and should more accurately be labeled as subjects rather than patients. The account below, however, uses 'patient' to avoid switching back and forth when contrasting CPRSs with CSDMSs. The differences between CPRSs and CSDMSs that impact temporal query operations are now discussed. We note that many CSDMSs do not have data structures needed for temporal support, and our discussion below focuses on the few that do. For example, in many commercial CSDMSs, valid-time stamps are not associated with individual parameter values. Instead, to record valid-time information, the study designer must invent pseudo-parameters such as 'date of interview' or 'date/time of blood collection', and add them to the case report forms where other attributes are being recorded. Such approaches make robust temporal query support very difficult.

2.2. Special temporal aspects of CSDMSs

In the CPRS, clinical data gathering is open-ended, while in the CSDMS it is limited to the duration of a study in which the patient participates. In clinical studies, individual response to therapy is less important than how patients react as a group. Therefore, a study is typically divided, from the patient's perspective, into study periods by critical time points, or study events, whose timing is determined by the study's protocol. Study events are typically given meaningful labels that usually indicate chronology and/or serialization, as well as the purpose of the event, e.g. 'Baseline', '3-month follow-up', 'Chemotherapy Cycle 1', etc. At any given instant, various patients will typically be within different study periods, e.g. one patient may be halfway through the study while another has just enrolled. Whenever a patient's study event occurs, the investigator performs a predetermined set of evaluations, e.g. based on lab tests or questionnaires. All clinical parameters relevant to a particular study are not evaluated for all

events: parameters that are expensive to evaluate (or whose determination poses a significant clinical hazard) are evaluated less frequently.

Once a patient has enrolled in a study, a 'calendar of events' for that patient can be computed. This calendar is essential to the satisfactory conduct of a study, because it serves as the basis for resource and appointment scheduling. For example, to optimally schedule relatively scarce resources, such as a PET scan, a particular patient's calendar can be computed backward from the scheduled scan date. Because of the importance of study events, data-entry personnel provide two kinds of time-stamp for every clinical data item: an absolute (valid-time) stamp and the study event ID, which serves as a relative time-stamp. When studying the time course of therapeutic response or adverse effects for the set of patients collectively, as is the focus of clinical studies, the study event IDs are much more useful than absolute timestamps, which vary across patients. The event IDs are also more readily interpretable. For example, during a long-term study, the investigations that are required for a given patient during a particular event (e.g. the annual follow-up) may be numerous, and may be performed over a week or more, rather than on a single day. While the resultant data items will have slightly different absolute time stamps, they will all have the same study event ID.

The importance of study events makes it essential to have a CSDMS's query interface support search of data items in terms of their associated study events: such search behavior should be the default rather than the use of absolute timestamps. In temporal query of clinical studies, absolute date/times are only important in unusual circumstances such as problems with the batch of the drug that was administered to several patients.

Previous research in this area has demonstrated temporal query support for rule-based expert systems [23] and data abstracts of CSDMSs [3-5] utilizing absolute valid-time stamps. Systems that additionally allow temporal queries based on relative time-stamps (study events) or that directly query an EAV CSDMS have not been previously reported. We now describe such an application with a graphical interface to make it accessible to the end user.

3. System description

We first provide a brief overview of our system architecture and database model, which has been described previously [24], and then describe in detail the component that enables execution of temporal queries.

3.1. Schematic of architecture

The overall system design is based on a standard, '3-tier' web architecture, where a web server application mediates between a user operating a web browser and a database server. This is shown in Fig. 1. The web server is implemented with Active Server Pages on a Microsoft Windows 2000 server machine. The Oracle database server runs on a CPU separate from the web server.

3.2. Trial DB's data and metadata model

An overview of Trial DB's data and metadata model is depicted in Fig. 2. The metadata tables are on the right side of the diagram, and the data tables are on the left. Each arrow represents a one-to-many relationship, originating from the 'many' or child table and pointing to the 'one' or parent table. Each dotted line with a double-headed arrow represents a many-to-many relationship, which is modeled using a bridging table (not shown).

When data is gathered for a study, it is entered into a case report form (CRF) over a secure web connection. Individual questions on a CRF are segregated into question groups, based on their functional relationship and sharing of a common timestamp (or timestamps). For example,

during each clinic visit, a set of vital signs may be obtained. The temperature, pulse rate, blood pressure and respiratory rate are a group of related questions that share a common single timestamp (date/time of examination), and they would be placed together into one question group. (Since this group contains only a single timestamp, it describes instant events.)

There are, however, certain groups of questions that are unpredictable in number. For example, 'adverse events' is a group of questions that include the type of adverse reaction, the severity of the reaction, and the alleged medication/therapy that caused the reaction. But there may be zero, one or any number of adverse events reported during the same clinic visit for a given patient. Also the timestamps for each adverse event could be different. A patient could have a mild headache due to omeprazole which started on 1/1/2002 at 12:00 h and ended at 15:00 h, and severe vomiting due to chemotherapy that started on 1/2/2002 at 15:00 h and ended on 1/4/2002 at 6:00 h. Information for both of these adverse events could be collected during the weekly evaluation on 1/5/2002. Therefore, 'adverse events' is an example of a 'repeating group,' a group of questions that may have an indeterminate number of repeat instances. Note that each adverse event has two timestamps—the date/time of onset and the date/time of resolution—thus describing a period event.

The Question_Groups table contains specific temporal metadata in the form of a field called Patient_Event_Type, which associates every question group with the type of timestamp, either instant or period. The actual timestamps (data) are stored in the Event_Subheader table as Start_Of_Event only (for instant events) or both Start_Of_Event and End_Of_Event (for period events).

When a new CRF is created for a patient, a new row is created in the Event_Header table. This table records, among other things, the Patient ID and Event_ID (for study events, described later). After the form is completed and saved, a new row is created in the Event_Subheader table for each question group in the CRF. For repeating groups, a new row is created for every separate repeat instance. Every row in the Event_Subheader table gets its own calendar timestamp(s): one timestamp for instant events and two timestamps for period events.

Finally, every question with a non-null value is recorded in an EAV table with the triplet Subheader_ID, Question_ID and Value. (The figure shows a single such table, EAV_Int: we actually create a separate EAV table for each data type, e.g. integers, strings, dates, etc., which allows indexing by value.)

One aspect of the temporal model, the study event (a study-designer specified time unit), is stored in the Study_Events table. This allows individual studies to maintain information correlating every patient data point with a study event (in the Event_Header table). For example, if, during a particular study, patients are evaluated weekly, the study events would be week 1, week 2, etc. Consequently, all the information on a CRF would not only be associated with an absolute timestamp, but also a relative timestamp (relative to when the patient enrolled in the study).

3.3. The user interface

The interface is designed to perform parameter-centric query, where the user can search for patients matching criteria based on clinical or demographic parameters. Parameters are selected using either a drill-down approach based on the case report form to which they belong, or by keyword. Searching by form is more useful when multiple related parameters from the same form are to be chosen. For a selected parameter, the user can optionally specify restrictions based on comparison operators (e.g. less than, greater than, between, contains), statistical aggregate operators (e.g. average, max, min), values, and a range of study events or absolute

datetimes to limit the set of values to be searched. Individual criteria can be combined in Boolean fashion.

TrialDB's query interface is driven by metadata, so that when the user chooses a study, only that information (case report forms, attributes, etc.) that is applicable to the chosen study is presented. Metadata is also used to validate the user's actions, by preventing the selection of choices that are inappropriate to a particular parameter. For example, the user is prevented from specifying certain period operators for a parameter representing instant events.

After execution, a query's definition may be saved for later reuse and modification. A single query definition corresponds to numerous consecutively executed SQL statements (often more than fifty, when multiple Boolean and temporal criteria are specified by the user). Further, the meaning of these individual statements is not intuitively obvious to the user, especially when multiple statements correspond to a single logical operation. In order to characterize the semantics of a query concisely, we use an XML-based representation. The user's actions, in fact, result in composition of an XML stream, which is then validated and interpreted by the query engine to generate the appropriate SQL. The query process ultimately results in the creation of one or more conventionally structured tables containing the desired results.

3.4. The temporal query component

The user interface is illustrated in Fig. 3. This consists of a set of frames where the user performs individual actions related to the query: browsing of metadata and selection of attributes; composing non-temporal criteria based on selected attributes; defining temporal criteria in terms of these non-temporal criteria; and combining both temporal and non-temporal criteria in Boolean fashion. (Not all queries have a temporal component.)

It should be noted that even for clinical events of the instant type, using a non-temporal criterion that contains a statistical aggregate function effectively results in a conversion from instant data to period data. For example, even though blood hemoglobin is measured at an instant in time, if we create a criterion based on the average or minimum hemoglobin for a patient, we must specify the start and end times or events between which this average or minimum must be computed. (By default, these are the first and last events defined for the study.) The resultant aggregate value is then tagged with relative and absolute timestamps.

A sample temporal query is illustrated in Figs. 3-7. Here, we wish to identify patients from a breast cancer dataset through criteria based on prior therapy: specifically, patients who have received more than six courses (cycles) of chemotherapy and more than 3000 rads of radiation. We further wish to apply a temporal restriction, limiting the patients to those who either 1) started and completed the radiation treatment during the chemotherapy interval or 2) completed the chemotherapy cycles before and within one month of the start of the radiation therapy. The latter restrictions form the basis of a temporal query.

In Fig. 3, the user composes the non-temporal portion of the query. Here, the individual criteria are ([Number of Chemotherapy Courses] > 6) and ([Total Dose of Radiotherapy] > 3000), which are referenced as criteria 1 and 2. These are combined with a Boolean AND operator, yielding the final (compound) non-temporal criterion, '1 and 2' of the lower left of Fig. 3. Fig. 4 illustrates the next step: defining the temporal criteria in terms of the non-temporal ones. The details of an individual temporal criterion (shown in the third frame of Fig. 4) are now summarized. We first discuss traditional temporal operations; trend operations are discussed later.

3.5. Traditional temporal operations

We first outline the structure of a temporal criterion, and then provide explanatory details, using Fig. 4 as reference. A temporal criterion consists of either one or two non-temporal criteria (Crit. 1 and Crit. 2). To each of these, one may optionally apply a unary temporal (qualifying) operator (Qual.1 and Qual.2). If two non-temporal criteria are used, one can apply a binary temporal (Allen) operator to them in the TimeOp column. The meaning of certain temporal operators can be optionally quantified by a triplet consisting of a relational operator (e.g. less than, greater than), a value (a real number), and a unit of time (e.g. days, weeks) that applies to the value. This triplet is recorded in the columns 'RelOp', 'Value' and 'Units'. The program serially numbers individual temporal criteria: in Fig. 4, they are designated as T1 and T2. Individual temporal criteria can be combined with Boolean operators (e.g. 'T1 OR T2').

- Allen operators: the designated names of these operators are somewhat arbitrary and unintuitive: for example, the precise meaning of 'X MEETS Y' or 'X FINISHES Y' is difficult for occasional users to remember. (Allen himself used line graphics to illustrate the operators' meanings in his paper.) Therefore, we let the user invoke a help screen and choose an operator by clicking on the graphic icon that describes its function. Fig. 5 shows the user interface for this purpose, and also illustrates the meaning of the individual Allen operators.

Although the Allen operators have been described strictly for period events, the user should not be prevented from applying them to comparisons of an instant event with a period event or even to two instant events, if the user's intended meaning is sufficiently unambiguous. Thus, for the Allen operator X BEFORE Y (which, as explained earlier, means the end of X precedes the start of Y) it does not really matter whether X, Y or both are instant events.

- Quantifiers: the quantifier triplet is optionally used to augment an Allen operator through the use of numeric thresholds. For example, in Fig. 4, criterion T2 states that non-temporal criterion 1 (Number of courses > 6) must occur BEFORE criterion 2 (total radiotherapy dose > 3000), but the gap separating the end of chemotherapy and the start of radiotherapy must be less than 1 month. The relational operator, value and time-unit are: '<', '1', and 'month' respectively. Temporal criterion T1 does not have any quantifiers: it merely states that any episode of radiotherapy > 3000 rads should have occurred during chemotherapy. (The temporal operator X DURING Y means that the start and end of X should occur within the bounds indicated by the start and end of Y.)

For certain time operators such as EQUALS or MEETS, the relational operator '<' is used with a value and time unit to indicate a significance threshold, as we now explain. X EQUALS Y is true if the two periods X and Y have the same start and end times, and X MEETS Y is true if the end of X coincides with the start of Y. In reality, because time is recorded by a computer's clock to an accuracy of a second (or fractions of a second), neither of these conditions is ever likely to be true with real clinical data. Therefore, the user can specify that two instant events should be treated as co-occurring if they are less than so many time units (e.g. 30 minutes) apart.

- Unary (qualifying) operators: for traditional temporal operations, the unary operators, which are 'Start,' 'End,' and 'Duration,' get the start time, end time and duration of an event. (For instant events, Start and End times are identical. The Duration operator applies only to periods, and is used with a quantifier triplet, so we can, for example, look for particular adverse events that lasted for more than 1 week). The Start and End unary operators allow more flexibility in specifying conditions than if we were forced to use the Allen operators exclusively. For example, suppose we only care that the start of event X comes before the start of event Y. The criterion Start(X) BEFORE Start(Y) is far more concise than the unwieldy and inefficient equivalent criterion: (X

BEFORE Y) OR (X MEETS Y) OR (X OVERLAPS Y) OR (Y FINISHES X) OR (Y DURING X).

3.5.1. Query results—The results of query execution are shown in Figs. 6 and 7. The output tables are uniquely named using a convention based on a sequentially ascending number prefixed by the user's login name. The non-temporal query results are shown in Fig. 6, while the temporal results are shown in Fig. 7. The first table, 'anid_TQ_2409_1,' is the result of the first temporal specification; the second table, 'anid_TQ_2409_2,' is the result of the second specification. Finally, the final table is the Boolean 'OR' combination of the other two.

3.5.2. Trend operators—The values of certain instant events form a time-series, within which one can look for trends, such as progressively ascending or descending values of a parameter. Trends for hematological parameters are particularly important in cancer chemotherapy. We therefore, support trend operators, which are unary Boolean operators that inspect a sequence of values for a pattern. The operators that are supported are 'Ascending', and 'Descending', whose meaning is obvious.

Sometimes, within a single study event, one may record multiple values of a parameter. An example is cancer chemotherapy, where absolute neutrophil count is recorded several times during each chemotherapy cycle. The count shows a cyclical pattern: it drops soon after chemotherapy is given, and gradually comes back toward the baseline until the next chemotherapy is administered, typically four weeks later. Here, one is less concerned with individual values than the average count or the minimum count (the 'nadir') for each cycle: we therefore allow the user to apply the statistical operators AVERAGE, MAX and MIN to group values per event before the trend operator is applied.

Another issue with trend operators is that very small differences amongst consecutive values may simply reflect the intrinsic measurement error (statistical noise). One may therefore wish to specify a 'noise' threshold [25]. Thus, a noise threshold of 2% with a DESCENDING operator would mean that, while the final value of the parameter in the time series must be less than the initial value, random ascents between consecutive values of < = 2% would be considered acceptable and still allow DESCENDING to return True.

Fig. 8 shows the use of trend operators. Here, the user selects 'AVG' in Qual. 1, and 'Descending' in TimeOp, with a noise threshold of 2%. Our implementation of trend operators is relatively recent. It is possible to conceive of numerous other operators: (e.g. based on rate of ascent or descent). A decision to support additional operators, however, will only be made after users articulate specific needs.

3.6. The temporal query engine

The user's interactions with the browser are converted by browser-based code into an XML stream (the query specification), which is then transmitted to the Web server. The XML schema for the temporal portion of the query is described in Appendix A. We use the W3C XML schema notation, rather than the older document type definition (DTD) notation, to allow more extensive data validation. The XML schema for the non-temporal part of the query has been described previously [24].

Temporal criteria are evaluated through a two-step process:

1. The first step is common to all queries. The query engine extracts rows from tables containing patient-related data based on non-temporal criteria, and performs EAV-to-conventional-structure transformations. The result is one or more temporary tables,

each with one column per attribute of interest, along with patient ID and the appropriate valid-time stamps.

2. If the user has specified temporal criteria, the temporal query engine then operates on these temporary tables to generate the desired result.

Compound Boolean expressions are evaluated using the standard 'stack machine' approach that is described in compiler-design textbooks, e.g. Ref. [26]. Briefly, one converts the expression from infix form, where the Boolean operators lie between their two operands, to postfix form, where the operators follow their two operands. For example, the infix expression (X AND Y) OR (P AND Q), where X, Y, P and Q are temporal operations, is converted to the postfix form X Y AND P Q AND OR. Infix-to-postfix conversion eliminates parentheses, and allows a very simple evaluation algorithm, which uses a stack data structure, to operate on the resultant expression.

We implement the temporal operators themselves using 'theta joins'[27], which are generalized join operations where the condition joining two tables is not necessarily equality. This requires more elaborate generation of SQL code, but query performance is better than with simpler alternative approaches that generate a Cartesian product of the tables followed by elimination of all rows not matching the condition, e.g. [28].

We illustrate this with the examples of Figs. 3-7. In the first step, we create a temporary table for each non-temporal criterion, (at least six cycles of chemotherapy, criterion 1) and (radiation dose > 3000 rads, criterion 2). These yield the temporary tables STN_507 and STN_508 of Fig. 6. Both these tables contain only the patient events that match the entire Boolean criteria of Fig. 3, along with the timestamps for each event. Consequently, both tables contain the same list of patients (although any patient may appear more than once in either table).

Using these two tables, then, we narrow down to the patients that also match each temporal criterion. Thus, the specification in the second row of the third frame in Fig. 4, '1 BEFORE 2 within 1 month,' is converted to a temporary table:

```
CREATE TABLE anid_TQ_2409_1 AS
SELECT T1.Patient_ID,
    T1.Start_of_Event as SOE_1,
    T1.End_of_Event as EOE_1,
    T1.Chemotherapy_Num_Courses,
    T2.Start_of_Event as SOE_2,
    T2.End_of_Event as EOE_2,
    T2.Radiotherapy_Total_Dose
FROM STN_507 T1, STN_508 T2
WHERE T1.Patient_ID = T2.Patient_ID
AND T2.Start_of_Event-T1.Start_of_Event > 0
AND ABS (T2.Start_of_Event-T1.End_of_Event) < = 30.44
```

In the above code, the conditions on the last two lines implement the theta join. The specification in the first row of the third frame in Fig. 4, '2 DURING 1' is calculated in the same manner. Finally, a set union (to evaluate the Boolean time criteria 'T1 or T2') is performed on the above temporary temporal tables to get the final set of patients shown in Fig. 7. The

Boolean time expression, 'T1 OR T2,' is evaluated using a stack engine similar to that described in [24].

We now provide a brief explanation of the last line of the above code. Relational databases store dates in Julian form, as real numbers that are offsets from an arbitrary 'time-zero', where integer increments represent a whole day: the conventional display form of a date is computed on demand. A well-known problem with temporal data is that the concepts 'year' and 'month' do not represent fixed time intervals. Strictly speaking, therefore, one would have to determine the truth of the condition 'date X precedes date Y by less than a month' by using a complicated algorithm that successively compared various parts of X and Y: the year, month, day and fractional day. This would require several function calls that extracted each part of the date within the last part of the theta join. We use a short-cut approach by treating a year as 365.25 days and an 'average' month as 30.44 days (365.25/12). (This is explained to the user in a help screen.) Given the temporal granularity of data recorded in CSDMSs, our short-cut yields results in practice that are not significantly different from a purist's approach, while being much more efficient. In any case, since the value can be a decimal number, the user is at liberty to supply slightly different values to see if the result set is different.

Note the use of the 'ABS' function in the SQL above, which returns the absolute value of a number. Given that most RDBMS engines do not try to optimize any sub-expression that contains a function, the question is whether replacing ABS with an equivalent expression would improve performance. For example, the expression:

ABS ($T2.Start_{of}Event$-$T1.End_{of}Event$) <= 30.44 could be replaced by

$T2.Start_{of}Event$-$T1.End_{of}Event$ BETWEEN − 30.44 and 30.44

At least with the relatively modest number of rows returned in the temporary tables, our benchmarks indicate no appreciable difference in performance with either alternative.

The implementation of the trend operators is straightforward: one retrieves a chronologically sorted set of parameter values from the database, loads them into an array, and inspects consecutive pairs of values, as well as the first and last, for a trend.

At this point, it would be useful to discuss the issue of temporal granularity [25]. In a CSDMS, timestamps are usually saved either to the minute (e.g. for blood draws) or to the day (when the time of day is unimportant). The study designers determine the level of granularity appropriate for their study by creating study-specific metadata; this level of granularity is directly available to the end users during the temporal querying process. We also allow the end users to specify coarser granularities. For example, to find patients who have been concurrently given chemotherapy A and chemotherapy B, with a granularity of one week, the user would specify '1 equals 2' and ' < 1 week.' The SQL generated for this temporal specification is ABS (T2.Start_of_Event–T1.Start_of_Event) < 7. For period events, the SQL using End_of_Event would be similarly generated.

In this scenario, the relational operator, value and units can double as a granularity specification, similar to the 'noise' threshold described above for trend operators. The advantage of this specification is that the user is not limited to any predefined granularities; any arbitrary granularity, such as 72 h may be specified.

4. Present status and future work

The temporal query system has been tested and validated with a wide variety of queries on real data. Two non-programmer users within the Center for Medical Informatics have tested the system in addition to the authors. We are, however, focusing on improving response time before

making it generally available to all of TrialDB's users. Our reasons and approach are now discussed.

TrialDB's production schema is transaction-oriented. Specifically, it is highly 'normalized' [29], that is, data is stored with minimal redundancy across several tables. The conceptual 'EAV table' described in the Background section of this paper, for example, is actually divided into four physical tables for a single data type. (Without going into details, we only state here that the multiple fields comprising the 'Entity'—patient ID, study ID, etc.—are stored in a separate table. Also, multiple attribute-value pairs can be given the same valid-time stamp, because the attributes may have a common source—e.g. hematology parameters all derived from the same blood draw. This timestamp is stored in a separate table that groups the set of parameters.)

Highly normalized schemas work well for interactive updates, as well as browsing of small volumes of data (in this case, individual case report forms for a patient in a study). Every time, however, that TrialDB has to identify patients by parameter-based criteria, the RDBMS must perform a join of these three tables. Such operations can process large numbers of rows, and can be potentially lengthy. In general, queries of arbitrary complexity should not be performed on a transactional schema, because response times for users who are entering or editing data (and who must be given priority) can be affected adversely. The correct approach in such a case is to move the production data in bulk to a separate CPU on a periodic basis, e.g. nightly. The separate CPU acts as a 'query server' whose data is read-only. The read-only nature of the data then allows us to perform certain schema-redesign optimizations to allow the queries to run faster. The most straight-forward of these is 'view materialization', which is now explained.

A view is a combination (a 'join') of data from several tables that are linked together based on shared fields. View materialization is based on the philosophy that 'permanently' joining the several tables that comprise a view into a new physical table achieves superior query performance at the cost of space. Practically every column in the new table is indexed. The DBMS may save a significant amount of time because multi-table joins do not need to be performed repeatedly, though the new table takes up extra space because the patient ID, study ID, etc. are repeated redundantly for every value of every parameter. Such an approach, termed de-normalization, is used in data warehouses, and is justified when there are no interactive edits to the data, as is the case for query servers.

Materialized views can be a mixed blessing. In some cases, because the resultant rows are much larger than the original rows, the DBMS must do more work in pulling a certain number of rows off disk. Fewer rows fit into the DBMS's memory cache: therefore, for queries that search through large volumes of data, many more cycles of cache-refill must be performed. Cache effects can therefore partially offset the benefits of joins.

We have benchmarked a few queries with and without view materialization. (These benchmarks were performed on a test CPU that contained the same data set as our production CPU.) The query of Figs. 3-7 took 1.4 and 0.7 s without and with view materialization. Another looked at pretreatment serum bilirubin and albumin levels as well as chemotherapy agents for patients on concomitant therapy with any one of 12 named medications (not listed here for brevity) that are known to be transported by binding to serum albumin. The run time of this query went from 14.7 s without view materialization to 0.8 s with view materialization.

We speculate that the much greater gain in run time for the second query is due to the fact that much of the query searches a table containing string values (which can be up to 255 bytes in length), whereas all the data types specified in the first query are numbers (which have much smaller space requirements). Therefore, adverse cache effects are likely to be seen with

numbers, while their effects are less pronounced with strings. We will need to perform further experiments to determine the optimum degree of de-normalization for each data type.

We have planned a more comprehensive benchmarking of queries for the future, after which we will set up our query server and create scripts that restructure nightly imports of data that are migrated from our production CPU. Making the query server available to all of TrialDB's users will enable us to get detailed feedback.

We are also planning to allow the storage of indeterminate timestamps [25]. For example, if a patient reported that he had a 'headache two days ago sometime between noon and 15:00 h,' it should be recorded as having an indeterminate timestamp. There are several ways to approach this issue. One method involves adding a field to the Event_Subheader table (Fig. 2) that would store indeterminacy as a time value. In our example, we could assign the timestamp as 13:30 h, with an indeterminacy of 90 min. An alternative method would involve recording the minimum and maximum values for the timestamps. The method we choose will depend upon feedback from our users.

5. Discussion

5.1. Temporal operations on CSDMSs versus CPRSs

Temporal operations on CSDMS can differ significantly from temporal operations on CPRSs. Patients in a CPRS do not necessarily have any common constraining 'events' such as 'Date of Entry into Study' or 'Date of Initial Chemotherapy.' In a CSDMS, by contrast, the users of CSDMSs tag all parameter values with Study Event IDs during data entry (along with the absolute datetimes). The Study Event IDs act as relative time stamps, pre-aligning all patients' data on a common time axis, even though individual patients have entered the study at different absolute times.

In CPRSs, on the other hand, when we compare the data for different patients with similar clinical conditions, this alignment must be computed by temporarily subtracting the first time stamp for each patient's data (for a given clinical parameter) from all subsequent time points for that patient. This operation involves manipulation of fairly significant data volumes, and therefore adds a database-intensive computation step when processing temporal queries

The use of study event IDs greatly simplifies many queries. For example, suppose we wish to get a list of patients with 'late bone marrow toxicity,' or bone marrow toxicity nine cycles after starting chemotherapy (but not earlier). With the use of study events, we can generate the query without specifying using a temporal engine. We simply specify patients with bone marrow toxicity between two periods (Fig. 9), where toxicity is defined by the criterion [WBC count in thousands] < 2.5, and execute the query in one step. (Not shown in the figure, but accessible by the user, is a 'key' that shows the chronological sequence of individual events: for cancer chemotherapy, for example, the events are 4 weeks apart, so that event 10 would occur at 40 weeks after start of therapy.)

Without study events, we would need to take the following steps: (1) Find the date of entry into a study for each patient. (2) Find all patients who were treated with any chemotherapy agent and who developed bone marrow toxicity, along with the date of toxicity. (3) Find the date difference between the date of entry into the study and the date of first appearance of toxicity. (4) Narrow down to the list of patients whose date difference was between 40 and 48 weeks, and save this list. (5) Repeat steps 2–4 to eliminate from this list patients where the toxicity appeared earlier.

5.2. Putting our approach in perspective

The approaches used by temporal query engines described in the literature fall into two categories. Engines at one end of the spectrum, typified by Nigrin and Kohane's DxTractor, aim at users with very modest database skills: few assumptions are made about the user's expertise. The user interface lets the user compose complex operations one step at a time, in much the same manner as interfaces to bibliographic databases such as Ovid™ and Silver Platter™. The engine returns intermediate results every step of the way, until the user obtains the final results of interest. Such systems are by far the easiest to learn, and much of their sophistication lies in their user interface-code.

At the other end are engines such as CHRONUS (Das and Musen) [10] and CHRONUS II (O'Connor, Tu and Musen) [28]. These implement a complete temporal query language, typically a SQL extension, and assume a power-user fluent in SQL. Their strength lies in parsing/interpreter technology: they are less concerned with the user interface, beyond returning the data requested. These engines are also the most powerful and expressive in terms of what the user can do in a single step, and performance is also good.

The pros of one technology constitute the cons of the other:

- The slow performance of easy-to-learn engines is partly a function of their user interface metaphor. Intermediate results must be specified by the user in single steps (and stored by the system), while in a more advanced system that allows the user to write arbitrarily complex constructs, these intermediate results can often be optimized away either by the DBMS or the implementation.
- Temporal languages are, understandably, hardest to learn. Another practical problem is seen with temporal languages that are used in production systems. (This is due to the inertia of international committees and database vendors rather than the fault of the researchers concerned.) The language specification needs to be reworked, often significantly, as and when standards change. For example, some of CHRONUS II's software engineering involves changing the earlier syntax of CHRONUS to be TSQL2-compatible. Further, production code (e.g. stored queries) may then need to be rewritten; it is hard to maintain backward-compatibility.

Our approach falls between these two.

- Like DxTractor, we focus significantly on the user interface. We postpone SQL generation, however, until the user has specified several operations. The resultant SQL code then performs multiple DxTractor-equivalent steps in one statement. This allows the back-end DBMS to perform some optimizations, and fewer intermediate results need to be stored.

For example, the binary temporal (Allen) operators are really a form of shorthand, whose equivalent can be achieved, as DxTractor does, by combining the Start and End operators with relational and Boolean operators. Thus, the operator DURING (as in X DURING Y) can be expressed as (START (X) > START (Y)) AND (END (X) < END (Y)). Supporting the Allen operators directly, as we do, allows generating compact SQL code, and reduces (albeit very modestly) the user effort required to compose such queries.

- We diverge from the temporal-language approach in that we use an XML-based exchange format rather than a true computational language. Like temporal-SQL query, an XML-based query specification can be saved for later reuse. In our case, however, the choice of XML was virtually forced on us for several reasons:
  - Our user interface hides the difference between attributes represented in EAV form and attributes (such as demographics) that are represented

conventionally. This is because the representation of an attribute in a particular fashion is a pragmatic decision that is made by the database developer, and may even change during the lifetime of a system. (Such changes have occurred in production EAV systems such as the 3M CDR.) The query engine should be able to determine the representation of an attribute dynamically by consulting the attribute's metadata, without having to force the user to write program code that uses two different data-access metaphors (which has to be rewritten if the underlying representation of an attribute is changed). For languages tied closely to SQL, such differences in data representation cannot be hidden, short of inventing a completely new language for clinical database query.

- The XML-based specification defines the query in terms of higher-level semantics that are converted to multiple SQL operations. Such a specification is possibly less vulnerable to changes in the underlying RDBMS engine than a computational language. For example, it is easier to take advantage of true temporal support if and when it becomes available in RDBMSs without requiring a rewrite of the specification existing code that uses the XML.
- From the software-engineering perspective, implementing an XML-based specification is simply much less work. XML is readily parsed and validated using lightweight software components that are often available at no cost (e.g. the Microsoft XML parser). A temporal language, by contrast, requires the building of a custom interpreter using lexer-and parser-generator technology. Similarly, the turnaround time for revision of an XML-based specification (and the 'interpreter') is much shorter than for a computational language. (Our specification has undergone several revisions in the course of its development.)
- Trend operators are not specified in TSQL2, and we wished to avoid devising our own language/syntax for these operators, especially because more operators may be added by us in future.

- Temporal expressivity: our engine is able to support three of the four temporal patterns specified by Das [12]. The three temporal patterns, 'Temporal Duration,' 'Temporal Window' and 'Prior Presence' are directly supported by our user interface.

The fourth temporal pattern, 'Temporal Concatenation,' is not supported. The example shown in Das' paper consists of finding patients 'with a documented history of a positive PPD skin test without subsequent treatment with more than one month of any antimycobacterial medication.' The authors define an operator that 'concatenates adjacent or overlapping intervals' (italics added)—in this case combining all the adjacent/overlapping intervals during which any antimycobacterial drug was administered. The authors' example, however, is somewhat artificial. To be more practical, their concatenation operator should allow the user to specify an acceptable temporal gap. Time intervals separated by less than this gap could be considered adjacent and therefore could be concatenated. For example, if a patient took Isoniazid from days 1–29, skipped day 30 and resumed it on days 31–59, the gap of one day might not be clinically significant enough to preclude concatenation of these intervals.

Although our query engine can generate temporary tables (which the user can download) containing patients with a history of positive PPD and their timestamped antimycobacterial treatment regimens, one would have to write procedural code to concatenate adjacent/ overlapping intervals and generate the desired result.

The cons of our approach also follow from its positioning: it is possibly harder to use than a DxTractor-like approach, and it reliance on XML makes it distinctly less expressive than one based on a true computational language. The above account is by no means intended to be critical of other approaches: they are eminently suited to the problems for which they have been devised. Our own ‘compromise’ approach appears to be a reasonable fit for the needs of production CSDMSs. The contribution of this paper is in highlighting how CSDMSs differ temporally from CPRSs, and how to utilize these differences in simplifying the code generation that supports their temporal query.

Availability of software: TrialDB is copyrighted software, but is freely available from Dr. Nadkarni as open-source to investigators in academia.

Appendix A: The XML-based query specification

The query specification is given below using the W3C XML schema syntax. The schema shown is only the part used for temporal operations. A description of the elements in the specification follows.

```
- <xs:schema
    targetNamespace="http://amd47.med.yale.edu/TrialDB_AdHocQuery"
    xmlns="http://amd47.med.yale.edu/TrialDB_AdHocQuery"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    elementFormDefault="qualified" attributeFormDefault="unqualified">
  - <xs:element name="DATA">
    - <xs:complexType>
      - <xs:all>
        - <xs:element name="TIME_CRITERIA">
          - <xs:complexType>
            - <xs:sequence>
              - <xs:element name="TIME_ROW" maxOccurs="unbounded">
                - <xs:complexType>
                  - <xs:all>
                      <xs:element name="TIME_SERIAL_NO" type="xs:integer"
                        />
                      <xs:element name="CRITERION_1" type="xs:integer" />
                    - <xs:element name="QUALIFIER_1">
                      - <xs:simpleType>
                        - <xs:restriction base="xs:string">
                            <xs:enumeration value="" />
                            <xs:enumeration value="Start" />
```

```
            <xs:enumeration value="End" />
            <xs:enumeration value="Duration" />
            <xs:enumeration value="AVG" />
            <xs:enumeration value="MIN" />
            <xs:enumeration value="MAX" />
          </xs:restriction>
        </xs:simpleType>
      </xs:element>
    - <xs:element name="TIMEOP">
      - <xs:simpleType>
        - <xs:restriction base="xs:string">
            <xs:enumeration value="" />
            <xs:enumeration value="Before" />
            <xs:enumeration value="Meets" />
            <xs:enumeration value="Equals" />
            <xs:enumeration value="During" />
            <xs:enumeration value="Starts" />
            <xs:enumeration value="Finishes" />
            <xs:enumeration value="Overlaps" />
            <xs:enumeration value="Ascending" />
            <xs:enumeration value="Descending" />
          </xs:restriction>
        </xs:simpleType>
      </xs:element>
      <xs:element name="CRITERION_2" type="xs:integer" />
    - <xs:element name="QUALIFIER_2">
      - <xs:simpleType>
        - <xs:restriction base="xs:string">
            <xs:enumeration value="" />
            <xs:enumeration value="Start" />
            <xs:enumeration value="End" />
            <xs:enumeration value="Duration" />
          </xs:restriction>
        </xs:simpleType>
      </xs:element>
      <xs:element name="TIME_RELOP" />
      - <xs:simpleType>
        - <xs:restriction base="xs:string">
            <xs:enumeration value="" />
            <xs:enumeration value="less_than" />
            <xs:enumeration value="greater_than" />
            <xs:enumeration value="less_than_or_equals" />
            <xs:enumeration value="greater_than_or_equals" />
            <xs:enumeration value="equals" />
            <xs:enumeration value="within" />
          </xs:restriction>
        </xs:simpleType>

      <xs:element name="TIME_VALUE" type="xs:decimal" />
    - <xs:element name="TIME_UNITS">
      - <xs:simpleType>
        - <xs:restriction base="xs:string">
            <xs:enumeration value="" />
            <xs:enumeration value="mn" />
            <xs:enumeration value="hr" />
            <xs:enumeration value="dd" />
            <xs:enumeration value="wk" />
            <xs:enumeration value="mm" />
            <xs:enumeration value="yy" />
            <xs:enumeration value="pp" />
          </xs:restriction>
        </xs:simpleType>
      </xs:element>
    </xs:all>
  </xs:complexType>
</xs:element>
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:all>
</xs:complexType>
</xs:element>
</xs:schema>
```

A.1. Description of the temporal query specification

The specification consists of *TIME_CRITERIA* that describe a set of one or more *TIME_ROWs*. Description of Individual Row Elements (TIME_ROW):

TIME_SERIAL_NO

Each row depicts one temporal specification and has an associated row number, numbered serially with a 'T' prefix (for 'Temporal'). The row numbers are used to compose complex Boolean criteria, such as (T1 and T2) or (T3 and T4).

CRITERION_1, CRITERION_2

An integer, which refers to the row number in the non-temporal query specification.

QUALIFIER_1

May be used to qualify the period data for Criterion_1. Possible values include 'Start,' 'End,' and 'Duration,'. For trend operators, we allow the statistical aggregates 'Average', 'Min' and 'Max'.

QUALIFIER_2

In some scenarios, the period data for Criterion_2 may also be qualified. The possible values are 'Start', 'End' and 'Duration.' The trend operators are unary, and therefore the statistical aggregates do not apply here

TIMEOP

These are the Allen temporal operators for period data, such as 'Before,' 'Meets,' 'Equals,' 'During,' 'Starts,' 'Finishes,' and 'Overlaps.' The meaning of these operators is illustrated in Fig. 5. The permissible trend operators are 'Ascending,' and 'Descending.'

TIME_RELOP (Relational Operator

This is used to express equality or inequality between the CRITERIA and the TIME_VALUE/TIME_UNITS, and consist of the five operators: <, < =, =, > =, >. For trend operators, 'WITHIN' is used to specify a 'random noise' threshold, as discussed in the text

TIME_VALUE

A real number, referred to by the TIME_RELOP in conjunction with the TIME_UNITS.

TIME_UNITS

Possible values include minutes (mn), hours (hr), days (dd), weeks (wk), months (mm), years (yy). For trend operators, we also permit percent (pp), which works in conjunction with WITHIN.

Acknowledgments

Grant support: NIH grants R01 LM06843-02 from the National Library of Medicine and from the U01 CA78266-04 from the National Cancer Institute (Nadkarni). Dr. Deshpande is supported through the National Library of Medicine Medical Informatics Fellows Program.

References

1. Nadkarni PM, Brandt C, Frawley S, et al. Managing attribute-value clinical trials data using the ACT/DB client – server database system. J Am Med Inform Assoc 1998;5(2):139–151. [PubMed: 9524347]
2. Brandt C, Nadkarni P, Marenco L, et al. Reengineering a database for clinical trials management: lessons for system architects, Control. Clin Trials 2000;21(5):440–461.
3. Shahar Y, Chen H, Stites DP, et al. Semi-automated entry of clinical temporal-abstraction knowledge. J Am Med Inform Assoc 1999;6(6):494–511. [PubMed: 10579607]

4. Shahar Y, Musen M. Knowledge-based temporal abstraction in clinical domains. Artif Intell Med 1996;8(3):267–298. [PubMed: 8830925]
5. Shahar Y, Cheng C. Intelligent visualization and exploration of time-oriented clinical data, Top. Health Inf Manage 1999;20(2):15–31.
6. Nigrin D, Kohane I. Temporal expressiveness in querying a time-stamp-based clinical database. J Am Med Inform Assoc 2000;7:152–163. [PubMed: 10730599]
7. Shoham Y, McDermott D. Problems in formal temporal reasoning. Artif Intell 1988;36(1):49–61.
8. Shoham, Y. Reasoning About Change: Time and Causation from the Standpoint of Artificial Intelligence. MIT Press; Cambridge, MA: 1988.
9. Allen JF. Maintaining knowledge of temporal intervals. Commun ACM 1983;26:832–843.
10. Das AK, Musen MA. A temporal query system for protocol-directed decision support. Methods Inform Med 1994;33(4):358–370.
11. Das, AK.; Shahar, Y.; Tu, SW.; Musen, MA. Proceedings of the 18th Symposium on Computer Applications in Medical Care. Baltimore, MD: 1994. A temporal abstraction mediator for protocol-based decision support.
12. Das, AK.; Musen, MA. Annual Symposium on Computer Applications in Medical Care. IEEE Press; Los Alamitos, CA: 1995. A comparison of the temporal expressiveness of three database query methods.
13. Shahar Y. A framework for knowledge-based temporal abstraction. Artif Intell 1997;90:79–133.
14. Snodgrass RT, Ahn I, Ariav G, et al. TSQL2 language specification. ACM SIGMOD Rec 1994;23 (1):65–86.
15. Snodgrass, RT.; Boehlen, MH.; Jensen, CS., et al. ANSI Experts' Contribution. International Organization for Standardization; Geneva, Switzerland: 1996. Adding Valid Time to SQL Temporal.
16. Snodgrass, RT.; Boehlen, MH.; Jensen, CS., et al. ANSI Experts' Contribution. International Organization for Standardization; Geneva, Switzerland: 1996. Adding Transaction time to SQL Temporal.
17. Niedner, CD. Proceedings of the 10th Conference on Computer Applications in Radiology. Anaheim, CA, Symposia Foundation; Carlsbad, CA: 1990. The entity-attribute-value data model in radiology informatics.
18. Huff, SM.; Berthelsen, CL.; Pryor, TA., et al. Proceedings of the 15th Symposium on Computer Applications in Medical Care. Washington, DC, IEEE Computer Press; Los Alamitos, CA: 1991. Evaluation of a SQL model of the help patient database.
19. Huff, SM.; Haug, DJ.; Stevens, LE., et al. Proceedings of the 18th Symposium on Computer Applications in Medical Care. Washington, DC, IEEE Computer Press; Los Alamitos, CA: 1994. HELP the next generation: a new client-server architecture.
20. 3M Health Information Systems. 3M Clinical Data Repository, 3M Corporation; Murray, UT: 2002.
21. Friedman, C.; Hripcsak, G.; Johnson, S., et al. Proceedings of the 14th Symposium on Computer Applications in Medical Care. Washington, DC, IEEE Computer Press; Los Alamitos, CA: 1990. A generalized relational schema for an integrated clinical patient database.
22. Johnson, S.; Cimino, J.; Friedman, C., et al. Proceedings of the 14th Symposium on Computer Applications in Medical Care. Washington, DC, IEEE Computer Press; Los Alamitos, CA: 1990. Using metadata to integrate medical knowledge in a clinical information system.
23. Kahn MG, Tu S, Fagan LM. TQuery: a context-sensitive temporal query language. Comput Biomed Res 1991;24(5):401–419. [PubMed: 1743002]
24. Deshpande A, Brandt C, Nadkarni P. Metadata-driven ad hoc query of patient data: meeting the needs of clinical studies. J Am Med Inform Assoc 2002;9(4):369–382. [PubMed: 12087118]
25. Combi C, Cucchi G, Pinciroli F. Applying object-oriented technologies in modeling and querying temporally oriented clinical databases dealing with temporal granularity and indeterminacy. IEEE Trans Inf Technol Biomed 1997;1(2):100–127. [PubMed: 11020814]
26. Aho, AV.; Sethi, R.; Ullman, JD. Compilers: Principles, Techniques, Tools. Addison-Wesley; Reading, MA: 1988. Section 2.3: syntax-directed translation; p. 33-40.
27. Garcia-Molina, H.; Ullman, J.; Widom, J. Database Systems Implementation. Prentice Hall; Upper Saddle River, NJ: 2000.

28. O'Connor MJ, Tu S, Musen M. Applying temporal joins to clinical databases. Proceedings of the AMIA Fall Symposium. 1999

29. Date, CJ. An Introduction to Database Systems. 9. Addison-Wesley; Reading, MA: 2001.

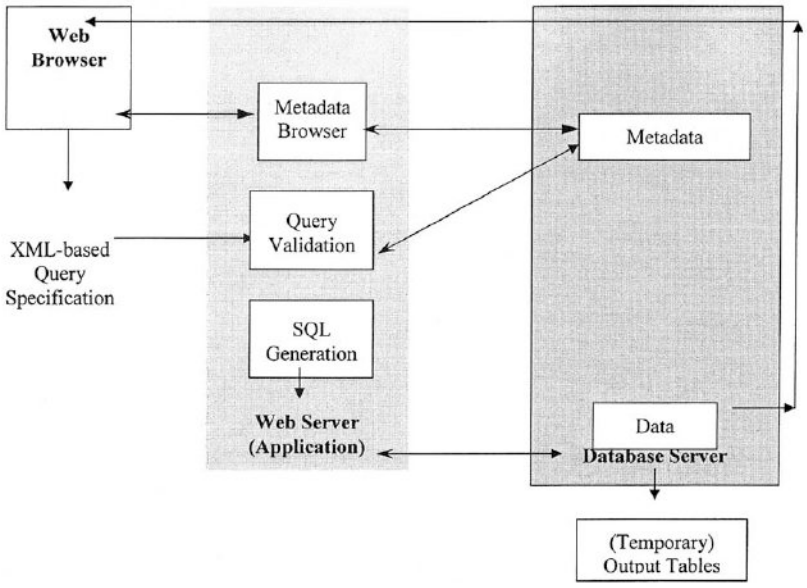


Fig. 1.

Schematic of the TrialDB query system architecture. A web server mediates communication between the web browser and the database. The user browses the study metadata, identifies parameters of interest and composes nontemporal and temporal criteria based on these parameters. The user's actions are converted at the browser into an XML-based query specification. When the user decides to execute a query, the browser sends this specification to the Web server, which validates it semantically, and then uses it to generate SQL that extracts data and creates one or more output tables that the user can then browse. Queries can be saved for later reuse.

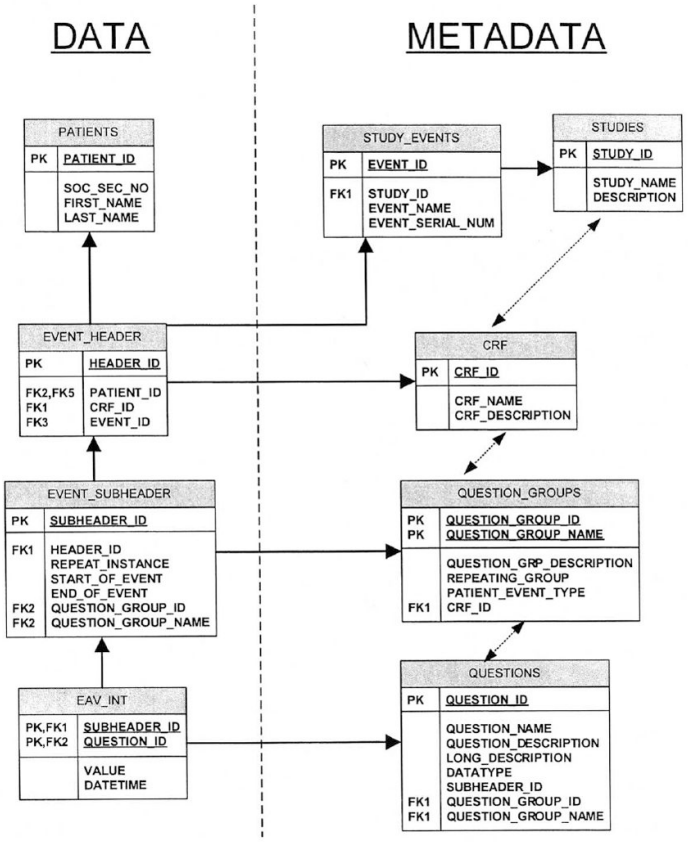


Fig. 2.

The Trial DB data and metadata model. A simplified overview of the Trial DB model is shown. A dashed line separates the data tables and metadata tables. Every arrow represents a one-to-many relationship and originates from the ‘many’ table and points to the ‘one’ table. Dotted lines with double-headed arrows represent many-to-many relationships and use bridging tables, which are not shown. Metadata tables associate case report forms (CRFs) with studies, question groups with CRFs and questions with question groups. Question groups record the temporal nature of individual attributes (in the Patient_Event_Type field). Data tables: the Event_Header table records the case report form ID as well as the patient ID and study event ID. The Event_Subheader table records the question group ID as well as the timestamps for the group of questions. The EAV tables (only one of the six types is shown) contain the subheader ID, question ID and the value for each question in the question group that has an actual recorded value (we do not record null values in the EAV tables).

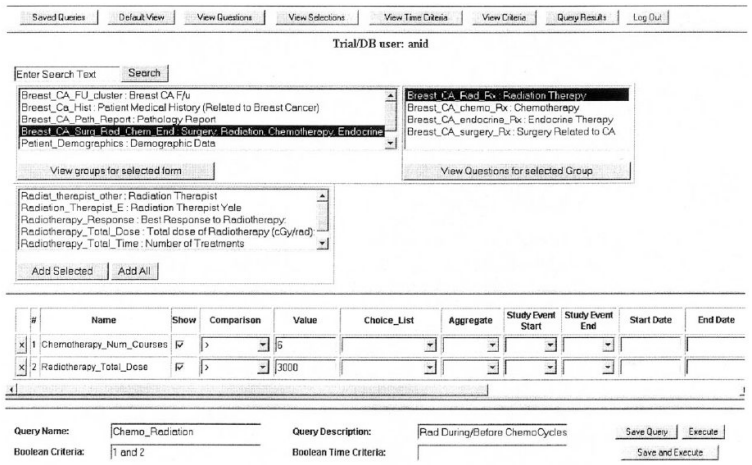


Fig. 3.

The Ad Hoc query user interface: specifying non-temporal criteria. As described in the text, temporal query is integrated with general-purpose ad hoc query. The parameters of interest are identified using search (by keyword) or by drill-down from the list of case report forms for the present study, as shown in the second frame from top. Selected parameters are appended to the table in frame 3. The user then applies restrictions based on these parameters, by combining with comparison operators and values. Here, we are restricting the set of patients of interest to those who have had more than six courses (cycles) of chemotherapy and a total dose of more than 3000 rads of radiation. Individual criteria (denoted by row number) can be combined in Boolean fashion in the 'Boolean Criteria' textbox in the bottom frame. (Here, we are using the Boolean operator 'AND.') When the user clicks on 'View Time Criteria' in the top frame, the temporal criteria frame becomes visible (which is described in Fig. 4).

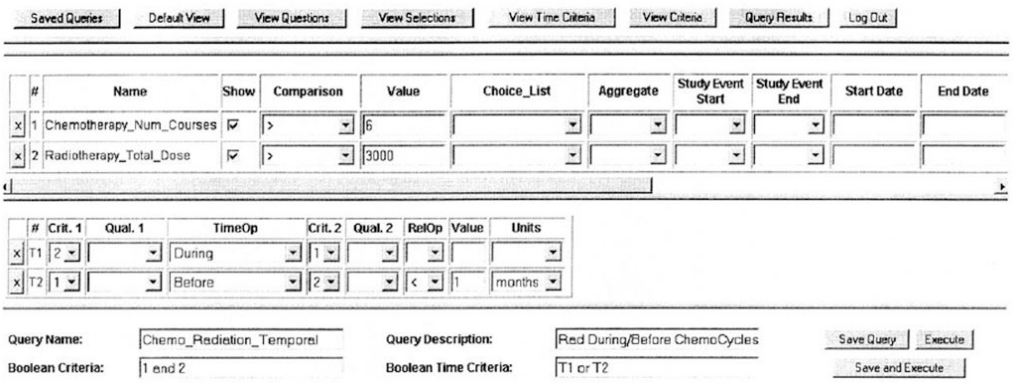


Fig. 4.

The Ad Hoc query user interface: specifying temporal criteria. The user has specified patients who have received more than six courses (cycles) of chemotherapy and more than 3000 rads of radiation (the criteria of Fig. 3, reproduced in the second frame). The temporal criteria (in the third frame from top) specify that either (a) the radiation dose occurs ‘during’ the chemotherapy cycles or (b) the chemotherapy cycles occur less than 1 month ‘before’ the radiation dose. The Boolean Criteria and the Boolean Time Criteria is specified in the bottom frame.

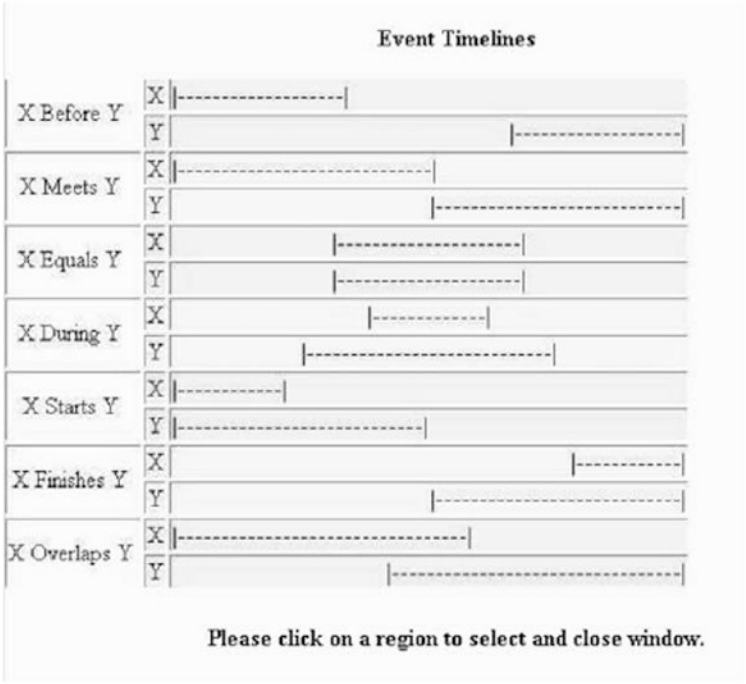


Fig. 5.
The GUI for the Allen operators for period data. The user may select a temporal operator by clicking in the corresponding region.

Trial/DB: Results for Query Chemo_Radiation_Temporal (# 2409)

Table: anid_STN_507

PATIENT ID	START OF EVENT	END OF EVENT	CHEMOTHERAPY NUM COURSES
1689766	3/1/1996	8/1/1996	8
1694263	2/16/1996	8/30/1996	8
1704173	2/5/1998	7/10/1998	8
1741997	1/11/1995	6/6/1995	8

Table: anid_STN_508

PATIENT ID	START OF EVENT	END OF EVENT	RADIOTHERAPY TOTAL DOSE
1689766	4/3/1996	5/7/1996	4600
1689766	5/8/1996	5/20/1996	6400
1694263	5/5/1997	6/9/1997	5000
1704173	8/6/1998	9/14/1998	5000
1741997	7/3/1995	8/4/1995	4600
1741997	7/13/1998	7/31/1998	3450

Fig. 6.

Results for non-temporal query. The ‘clinical events’ of interest for patients who have received more than 6 courses of chemotherapy and more than 3000 rads of radiation. To maintain the meaning of the start of event and end of event, every clinical event is listed (rather than just the patient id).

Temporal Query Results:

(SOE: Start of Event; EOE: End of Event)

Table anid_TQ_2409_1

PATIENT ID	SOE 1	EOE 1	RADIOTHERAPY TOTAL DOSE	SOE 2	EOE 2	CHEMOTHERAPY NUM COURSES
1689766	4/3/1996	5/7/1996	4600	3/1/1996	8/1/1996	8
1689766	5/8/1996	5/20/1996	6400	3/1/1996	8/1/1996	8

Table anid_TQ_2409_2

PATIENT ID	SOE 1	EOE 1	CHEMOTHERAPY NUM COURSES	SOE 2	EOE 2	RADIOTHERAPY TOTAL DOSE
1704173	2/5/1998	7/10/1998	8	8/6/1998	9/14/1998	5000
1741997	1/11/1995	6/6/1995	8	7/3/1995	8/4/1995	4600

Final Temporal Query Table of Patient ID's

Table anid_TQ_2409

PATIENT ID
1689766
1704173
1741997

Fig. 7.

Temporal query results. The top table consists of the patient events for the first temporal specification (radiation dose occurs ‘during’ the chemotherapy cycles). The middle table consists of the patient events for the second temporal specification (chemotherapy cycles occur less than 1 month ‘before’ the radiation), and the bottom table shows a list of patients matching the Boolean combination (‘or’) of the other two tables.

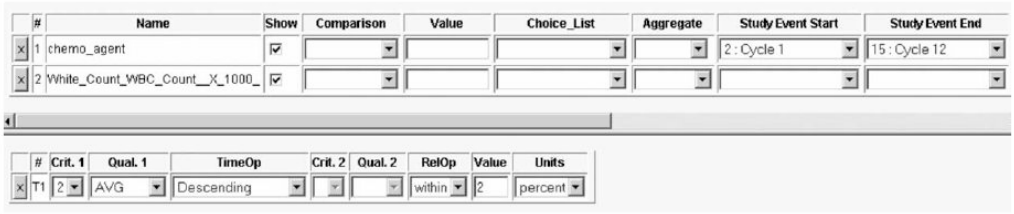
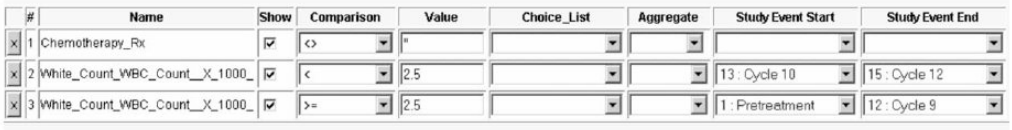


Fig. 8.

Use of trend operators: criterion 2 in the top frame chooses the parameter ‘total WBC count’ (the ‘X 1000’ in the name indicates that the parameter is expressed in thousands as a decimal number, versus being expressed as an absolute count). Criterion 2 is the basis of a temporal query in the lower frame that identifies patients where the average white cell count (per study event/cycle) shows a descending trend (where random increases between consecutive values of less than or equal to 2% are ignored).



	#	Name	Show	Comparison	Value	Choice_List	Aggregate	Study Event Start	Study Event End
x	1	Chemotherapy_Rx	☑	<>	''				
x	2	White_Count_WBC_Count__X_1000_	☑	<	2.5			13 : Cycle 10	15 : Cycle 12
x	3	White_Count_WBC_Count__X_1000_	☑	>=	2.5			1 : Pretreatment	12 : Cycle 9

Fig. 9.

Use of study events to define temporal criteria. By specifying values in the study event start and study event end columns, the user can limit the results to patients who have received chemotherapy and then developed leukopenia in cycles 10– 12 (but not in earlier cycles). This does not require the use of the temporal query engine. (criterion 1, chemotherapy_Rx < > ‘’, is a means of specifying ‘any chemotherapy’.) Not shown in the figure, but accessible by the user, is a ‘key’ that shows the chronological sequence of individual events: for cancer chemotherapy, for example, the events are four weeks apart, so that event 10 would occur at 40 weeks after start of therapy.