The Practice of Informatics JAMIA

Application of Technology ■

Managing Attribute–Value Clinical Trials Data Using the ACT/DB Client–Server Database System

PRAKASH M. NADKARNI, MD, CYNTHIA BRANDT, MC, MPH, SANDRA FRAWLEY, PHD, FREDERICK G. SAYWARD, PHD, ROBIN EINBINDER, BS, DANIEL ZELTERMAN, PHD, LEE SCHACTER, MD, PERRY L. MILLER, MD, PHD

Abstract ACT/DB is a client–server database application for storing clinical trials and outcomes data, which is currently undergoing initial pilot use. It stores most of its data in entity–attribute–value form. Such data are segregated according to data type to allow indexing by value when possible, and binary large object data are managed in the same way as other data. ACT/DB lets an investigator design a study rapidly by defining the parameters (or attributes) that are to be gathered, as well as their logical grouping for purposes of display and data entry. ACT/DB generates customizable data entry. The data can be viewed through several standard reports as well as exported as text to external analysis programs. ACT/DB is designed to encourage reuse of parameters across multiple studies and has facilities for dictionary search and maintenance. It uses a Microsoft Access client running on Windows 95 machines, which communicates with an Oracle server running on a UNIX platform. ACT/DB is being used to manage the data for seven studies in its initial deployment.

■ **JAMIA.** 1998;5:139–151.

The Yale Cancer Center (YCC) conducts numerous studies, with different numbers of patients and different design complexities. While small studies can be managed with simple tools (e.g., flat-file managers or spreadsheets), complex trials need sophisticated database expertise not readily available to individual investigators. A single system for managing multiple trials regardless of complexity can have significant benefits besides pooling of resources for data management expertise. These include standardization and reuse of data elements and controlled vocabularies; a common user interface for entering, editing, displaying, and reporting data; and the potential for pooling data from multiple studies for meta-analyses.

After evaluating several commercial clinical trials–management packages and deciding that they were not fully suitable to YCC's requirements, we created the Adaptable Clinical Trials DataBase (ACT/DB), which is currently undergoing initial pilot use. As guidelines, we used the numerous excellent descriptions of production clinical patient record systems in

Affiliation of the authors: Center for Medical Informatics (PMD, CB, SF, FGS, RE, PLM) and Yale Cancer Center (DZ, LS), Yale University School of Medicine, New Haven, CT.

This work was supported in part by NIH grants G08 LM05583 and T15 LM07756 from the National Library of Medicine, with additional funds provided by the Yale Comprehensive Cancer Center and the Yale School of Medicine.

Correspondence and reprints: Prakash M. Nadkarni, MD, Center for Medical Informatics, Yale University School of Medicine, P.O. Box 208009, New Haven, CT 06520-8009.
e-mail: ⟨Prakash.Nadkarni@yale.edu⟩

Received for publication: 6/12/97; accepted for publication: 9/22/97.

the medical informatics literature. We also had significant help from the Biostatistics Group at the Sloan-Kettering Memorial Cancer Center in New York City. This group, which has been operating a production cancer protocols database for several years, willingly shared their ideas and design with us. ACT/DB's design was subsequently generalized to handle trials not limited to oncology (and tested in collaboration with the Yale Clinical Trials Office) and to record data for certain types of outcomes research. After initial implementation and evaluation as a standalone application, it was recently ported to client–server mode.

ACT/DB is primarily a system for the gathering, editing, display, and reporting of information. It lets the investigator define a protocol for data collection and automatically generates graphical user interface (GUI) forms for data entry. ACT/DB allows multiple iterative cycles of rapid creation and modification of protocols and their associated forms, followed by testing with real or simulated data. Almost any design decision can be changed, so that an investigator can easily explore alternative ways of structuring (and permitting entry of) the same parameters. ACT/DB is designed to maximize reuse of components of existing protocols (forms, parameters, and so on) to allow new protocols to be defined rapidly.

ACT/DB is implemented with an Internet-accessible Oracle Server running on a UNIX workstation at the back end, and Microsoft Access 97 running on personal computers at the front end. In this paper we describe the ACT/DB architecture, the way in which we have handled design issues related to the entity–attribute–value (EAV) database model, and how ACT/DB is used.

Background

Clinical trials represent an active area of informatics research.[1] Numerous aspects of trials have been studied, among them methods of meta-analysis of pooled trials,[2] managing work flow in multicenter trials,[3] creating a standardized structure for recording trial reports,[4] assuring the quality of keyed medical research data,[5] and means of data validation.[6,7] Innovative research toward building applications that feed into clinical trials management systems include voice mail integration[8] and integration with screening systems for patient eligibility.[9]

Several systems have been built by researchers for data management of large clinical trials. Many of these, like that described by Pradhan et al.,[10] have been built to address the needs of a single trial, but a few have been generalized to address the more difficult problem of managing multiple trials (e.g., interactive database management[11] and the system created at Sloan-Kettering Memorial Cancer Center, mentioned earlier.)

While several clinical trials management packages[12,13] are commercially available, they have some limitations. Their user interfaces are relatively unsophisticated: some still use terminal interfaces. Some packages have proprietary or inadequately documented architecture, making customization by anyone other than the vendor very difficult. Others generate study templates based on a single–flat-file design, which makes them inappropriate for data that are naturally relational, e.g., studies in which sets of parameters (or attributes)* are sampled redundantly at varying and multiple time points. In general, commercial packages need careful and extended evaluation before purchase by institutions with access to programming expertise, which have the option of building their own system.

Design Objectives

Our system had to be robust and easy for clinical investigators and their staff to learn and use. It had to allow the rapid setup of new studies (within a couple of hours if existing study elements were reused, or a few days if a completely new domain was used). Easy modifiability of study design was mandatory, because study designs are often in flux. The end users needed to be able to generate reports (or export accumulated data to analytic programs) with minimal effort. A true GUI was necessary for ease of use.

While providing maximal built-in functionality, the system had to be customizable to meet the needs of individual clinical departments. Because data, transaction, and concurrency loads were expected to be low, speed and efficiency, while desirable, were less important than ease of use. The design of ACT/DB, while primarily for clinical trials, had to handle the highly heterogenous attributes typical of a clinical patient record, because an oncology trial often continues for several years and many of the clinical events need to be tracked.

System Description

ACT/DB Architecture

The ACT/DB architecture is summarized in Figure 1. It is divided into server and client components. The server holds four kinds of tables:

*In this paper we will use the term "parameter" as synonymous with "attribute."

- *Dictionary tables*, which are study-independent, contain definitions of the parameters and their logical grouping and ordering for the purposes of display and reporting of data.

- *Vocabulary tables* contain externally derived controlled vocabularies. An example of a controlled vocabulary in ACT/DB is COSTART (Common Standard Thesaurus of Adverse Reaction Terms, currently used for reporting by the FDA, and shortly to be superseded by an international system).

- *EAV data* are stored in a "normalized" form (i.e., with minimal data redundancy). Two forms of EAV data are managed: data that have a common structure across all studies, and data whose structure varies across studies. Examples of the former are inclusion and exclusion criteria for adding a patient to a trial, and treatment with standard therapeutic agents defined as part of the study protocol. Here, the data are segregated into individual tables based on function (e.g., there is a table for on-protocol treatment history), and there are few such tables. Data whose structure varies across studies include all the evaluation parameters recorded during a study. (Many of these parameters are study-specific.) Here, information common to all the parameters in a single clinical event (patient ID, study ID, start and end event times) is stored in higher-level tables, while information about the individual parameters gathered during that event is stored in datatype-specific tables, which are described below.

- *Non-EAV data*, such as basic patient demographics (name, sex, etc.), are stored in orthodox relational form. Such information is homogeneous in structure and less of a natural fit to EAV structuring. Also, we wished to explore ways in which EAV and non-EAV information can coexist. Production systems will always have legacy non-EAV data on which existing applications depend, and such data must be usable without conversion to EAV form.

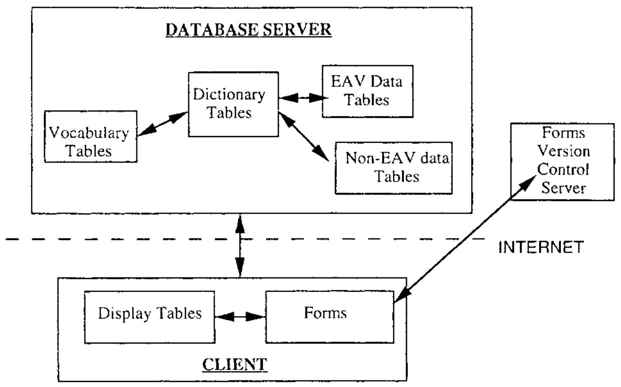


Figure 1 A diagram of the ACT/DB architecture. The database server and the forms server reside on different machines because they require different operating systems (UNIX and Windows NT, respectively). On the database server, the dictionary tables are central to the operation of the system, being consulted in virtually every database operation.

The *client component* consists of display tables to transiently capture EAV data, the forms associated with these tables and, most important, the code to manage client–server communication. Forms are managed through a network-accessible Windows NT–based Version Control server.

EAV-based Design Issues in ACT/DB

Most mainstream relational database engines have an upper limit (typically 255) on the number of columns per table. With an orthodox database design (one column per fact) numerous tables would be needed, and more would constantly need to be added as the number of recorded parameters increased. Only a relatively small number of positive and significant negative findings, however, are actually applicable (and recorded) for a given patient. As a result, the vast majority of columns would be empty (null), resulting in considerable wastage of space. Furthermore, any attempt at presenting consolidated information for a single patient (into a report, for example) would require navigating all these tables and looking for non-null fields.

This situation is similar to the computer-science problem of sparse array representation. The equivalent database design solution is *row modeling.* Here, data on a single entity (e.g., patient) are conceptually represented as multiple rows in a single table with few columns, rather than as a single row of data spread across multiple tables and columns. This design is also called the *entity–attribute–value* (EAV) design, because each row holds information on the *entity* (e.g., patient ID, visit, date), the *attribute* (the name or ID of the parameter being recorded), and the *value* of the parameter. To retrieve all facts on a patient, one simply searches the entity columns for the patient ID, ordering all rows by date if necessary.

Because of its structural simplicity, the EAV design is popular in clinical databases. Production databases using EAV components include the HELP system,[14] the Columbia-Presbyterian Medical Center's (CPMC's) clinical repository,[15] a system for tracking the care provided to a homeless population in the greater Boston area,[16] DEC-RAD,[17] a radiology package, and Oracle Clinical,[13] a commercial package for clinical trials management. Attribute–value pairs are also used in World Wide Web "cookies,"[18] which store state on the client and circumvent the stateless nature of WWW communication.

Successful operation of an EAV database requires management of the attributes. This is accomplished through a set of tables that make up an application-specific (as opposed to vendor-supplied) data dictionary. The dictionary component itself is often structured in EAV fashion. For example, the CPMC system uses the Medical Entities Dictionary (MED).[19] While EAV design is storage-efficient, it introduces complications with respect to table representation, display of data, query of data, and data input. In our system description below, we describe each problem briefly and discuss our design in the light of existing approaches.

Table Representation

While EAV information is stored *conceptually* in a single table, in reality databases are strongly typed. That is, a column can store only data of a single datatype (e.g., integer, floating-point, string, or date). Some databases (e.g., Oracle Clinical and early versions of HELP) use a single structure for all EAV information. Such a design achieves simplicity by coercing all the other simple datatypes into the least common denominator (i.e., string). This prevents efficient indexed searching based on parameter values. Value-based indexing can be important in queries aimed at rapidly identifying patients with parameter values lying within or beyond a specified range. It also becomes important when an EAV design is used as the basis of a large-scale data warehousing application and needs to support fast value-based searching. (An index on numeric values that have been coerced into strings is not very useful, because of a different sorting order: the ASCII string "123" is less than the string "23," even though it is numerically greater.)

Furthermore, this design deliberately sacrifices the ability to store long text (so called "Memo" fields) or complex Binary Large OBject (BLOB) data such as a histopathology slide or a chest radiograph.† In today's Web-oriented, multimedia-influenced world, with increasingly affordable disk storage costs (10 cents per megabyte or less) and desktop computing power, there is little justification for not supporting BLOBs when the underlying database engine is capable of handling such data.

†A BLOB or long text field cannot do double-duty as a simple field, because of the way such data is physically organized. It is typically allocated on disk in units of "pages" rather than individual bytes. Page sizes vary from 2K to 32K (in some databases, page size is configurable). Thus, with 8K pages, an object that is actually 12K in size occupies 16K. A larger page size results in more space wastage but faster retrieval time. In any case, the relative wastage of space may be modest when dealing with 100K objects, but is unacceptable when storing individual numbers or short text. Additional limitations exist on such objects. Many vendors do not allow more than one BLOB/long text field per table. Also, BLOBs and long text are "second-class" objects in that their direct manipulation often cannot be done through the native data manipulation language (SQL), but requires working through a complex programming interface.

ACT/DB segregates the EAV data into six tables, based on the datatype of the attribute: integer, floating-point, dates, short text (less than 256 characters), long text (up to 32,000 characters), and BLOB. (The inclusion of dates as EAV data is, in general, discouraged because ACT/DB has another means of tracking date–time information, as described later. EAV dates should be used only for isolated facts that will not be used for chronologic reporting, such as "date of menarche" in a gynecologic history.)

All six tables have an identical structure except for the datatype of the value field. The first four EAV tables have an index on the value column. The integer table is also used to store Boolean data and enumerated data. (We discuss enumerated and BLOB data in more detail below.) The ACT/DB data dictionary records the datatype of each parameter so that the appropriate EAV table may be accessed for querying.

Attribute Grouping. As described above, all attributes are *atomic.* However, many attributes are related to each other because they refer to the same clinical event. Consider, for example, radiotherapy. For each radiotherapy course (a separate event), the attributes of interest are the dose, the site, and the best response. One attribute by itself is relatively meaningless without the others.

Therefore, the study designer can aggregate attributes into *groups* and can also specify whether the group occurs an indefinite number of times (as in the above example) or just once in a given phase of the clinical trial. Groups are further aggregated into *clusters*, one cluster corresponding to all the groups that are to be displayed, or edited, on a single form. The designer specifies the ordering of groups within the cluster and the ordering of attributes within the group. This ordering information is used by ACT/DB's form generator, described later.

There is a many-to-many relationship between the attributes, groups, and clusters. This is because in different studies, different clusters may share the same attributes. In the interests of standardization as well as study design throughput, however, designers are encouraged to reuse existing clusters and groups rather than create new groups with existing attributes. In practice, therefore, most of the data follows a hierarchic relationship.

Data Display

It is generally desirable to present EAV data as though they were stored in orthodox format—one column per fact. Apart from being easier to look at, such a format is required for analysis by statistical and spreadsheet packages. If EAV storage is regarded as a column-to-row transformation, then display generally needs the reverse (row-to-column) transformation. If such transformation needs to be done regularly, some form of automation is desirable. One way to automate reverse transformation is through database views (predefined queries), with the EAV tables participating in multiple self-joins. Johnson et al., in their description of the CPMC system architecture,[20] mention the possibility of predefined views customized to individual classes of users. Packages like Oracle Clinical allow a database manager to generate "canned" views for sets of parameters.

The major limitation of study-specific views in an EAV database is that the number of views needed by different users may increase nonlinearly with the number of parameters in the database. (This problem is noted in Johnson et al.[20]) Furthermore, in the case of clinical trials, each time a new study is defined in the database, additional canned views need to be created. In a situation where dozens of studies are active, simply managing such views can be a major undertaking.

When numerous clinical parameters need to be extracted in a single operation, an additional problem arises. When the same EAV table is used multiple times in a view by repeatedly joining with itself, each additional use of the table is treated by the database engine as if it were a new table. Most database engines limit the number of tables participating in a multitable join. (For example, Sybase has the rather small limit of 16.) Working around such limits manually requires inelegant solutions, such as creating temporary tables in multiple steps to hold intermediate results and joining these temporary tables to each other. Automating such operations then requires writing study-specific stored procedures (server-based subroutines), which again become increasingly hard to manage as they increase in number. (Very few packages support SQL-3's package mechanism for modularization of server subroutines.)

Predefined queries run somewhat faster than dynamic SQL queries because they are "compiled" on the server into a query execution plan. Such performance enhancements are not, however, guaranteed to be dramatic. Complex joins involving a large number of tables may not run significantly faster than a number of simple queries, each involving only a few tables.[21] Especially if the total volume of the returned data is modest, it may be as efficient (or more efficient) to issue a series of simple queries and assemble the returned data on the client.[22]

In designing ACT/DB, we decided that the improved performance of study-specific queries involved too much maintenance overhead. As a result, ACT/DB uses a small number of study-independent views and lets the user select parameters, or sets of parameters, through a list-selection interface. The result is generation of dynamic SQL, which is sent to the server. The captured data are sent in EAV form to the client, where they are converted to columnar form for output.

ACT/DB displays data to the user in two ways. Parameter values gathered during an individual clinical event are shown on forms, where they can be edited. Data for an entire study are presented through reports of several kinds. Some of these reports (e.g., those relating to a single patient) present the data in attribute–value form, while others (e.g., reports on numerous patients in a single study) use columnar form. Although ACT/DB is not primarily an analytic tool, it can export columnar data into text files for direct import into spreadsheet and statistical packages.

Data Input

Input requirements for EAV data are similar to display requirements. It is desirable to create the illusion of an orthodox database design. The "forms" design components of commercial database packages are in fact geared toward such a design, with one editable form object (e.g., text box, check box) corresponding to one column in a table. Because each fact describing a patient is stored as a separate row in an EAV table (with all attributes and values in the same conceptual columns), the EAV design is not well matched for straightforward form-based data entry per se.

Two approaches have been used to address this problem. In one, protocol- or department-specific orthodox tables are the basis of forms for data entry. Such tables may also serve the purpose of operational use within a department. The information thus supplied is then translated through custom-designed (but straightforward) programs into EAV form for storage at a remote (typically, an institution-wide) repository. The second approach, in effect, allows direct data entry and editing of the EAV data.

ACT/DB's Form Generation Architecture. ACT/DB uses the second approach. The protocol designer specifies what facts are to be gathered in a form. (The designer does this by editing the data dictionary contents through forms that are accessible to users with design privileges.) ACT/DB uses tables that reside only on the Access client to transiently capture EAV data from the server for an event and present it to the user in a "flattened" (i.e., orthodox) view. These tables are viewed through the generated forms.

One such table is used for the main form, and up to five tables can be used for subforms that are embedded in the main form. Subforms are used for groups of parameters that may be repeated a definite or indefinite number of times in a single case report form. For example, in a cancer therapy protocol when we record "past therapy," we may wish to record multiple instances of past surgery, past radiotherapy, and past chemotherapy, with details specific to each. Each of these is entered in its own subform, because a given patient may have zero or more instances of a particular form of therapy. Another example is a pharmacokinetic study where blood samples are collected at a fixed number of intervals, the actual number and timing of the samples depending on the study protocol.

These client tables have fields with a simple naming convention: a single letter indicating their datatype and two numbers indicating a sequential number. (Thus, the second string field has the name S02.) For a given form, the mapping between individual parameters to be entered or viewed on the form, and the fields on the client that will transiently hold the parameter values, is generated automatically during form creation. (This mapping is stored in a server table.) When new clinical data are added or edited, the mapping allows the flattened data in the file to be translated into EAV form and sent to the server. When existing clinical data are to be viewed, EAV data are translated, again through the mapping, into flattened data that are transiently captured in the client tables.

ACT/DB allows the protocol designer to generate a form with a single button click after the parameters for that form, and the order in which they are to be presented, have been specified. These forms have standard GUI components (text boxes with vertical scrolling, check boxes, pull-down menus, subforms), are reasonably esthetic, and have a consistent appearance. Furthermore, the default fonts, character sizes, and colors for form titles, form section headers, labels, and text boxes can be altered (by interactive editing of entries in a "preferences" table) if the user so desires. The forms should require modest, if any, cosmetic alteration before they can be used in simulation or production mode. If necessary, they can be altered using Microsoft Access form design tools.

A limitation of our approach is that the number of objects on the form that correspond to parameters is limited to the maximal number of fields that an Access table can hold, which is 255. (In practice, this is significantly less: for example, the "main form" table can store up to sixty integer parameters but only three BLOB parameters and ten long-text parameters.) This is not a significant limitation, because all forms map to the same display tables, and there is no limit to the number of forms per study.

Part of a form, and the mapping between fields on the form and records in the EAV tables, are illustrated in Figure 2.

Ad Hoc Query

Closely coupled to the task of EAV data display is ad hoc query. No existng package seems to support ad hoc query of EAV data on arbitrary, complex Boolean criteria. This is understandable, because query of EAV data is not very efficient compared to query of tables in orthodox columnar form. Because facts on multiple parameters are stored in a single table or structurally similar tables, Boolean searches that combine multiple parameters must perform numerous self-joins on the EAV tables through relatively expensive row-based set unions, intersections, and differences for the And, Or, and Not operations respectively. These operations have not received much support, let alone efforts at optimization, among commercial database engines. Keyword support for intersection and differences was added to the SQL standard only in SQL-92.[23] Among mainstream database engines, there is very limited support for all three operators: even Oracle 7 uses different keywords for intersection and difference.[24]

We are currently working on an ad hoc query module for ACT/DB. This module is currently in prototype stage, and it is too early to tell whether our efforts will be entirely successful. Our approach is based on a GUT rather than a query language. (While potentially less powerful than a language, GUIs are simpler to implement.) The interface lets the user query the EAV data as though they were a single orthodox table with a very large number of columns. The system converts the user's complex Boolean operators into the appropriate set operators, with each clause in the query being converted into a SELECT on the EAV table corresponding to the parameter in the clause.

There are numerous complications in the ad hoc query of EAV data. For example, semantic correctness of queries must be ensured by restricting relational or aggregate operators to those appropriate to a given parameter's datatype, and a query generator usually needs to create intermediate temporary tables to process highly complex queries. These issues are beyond the scope of this paper.

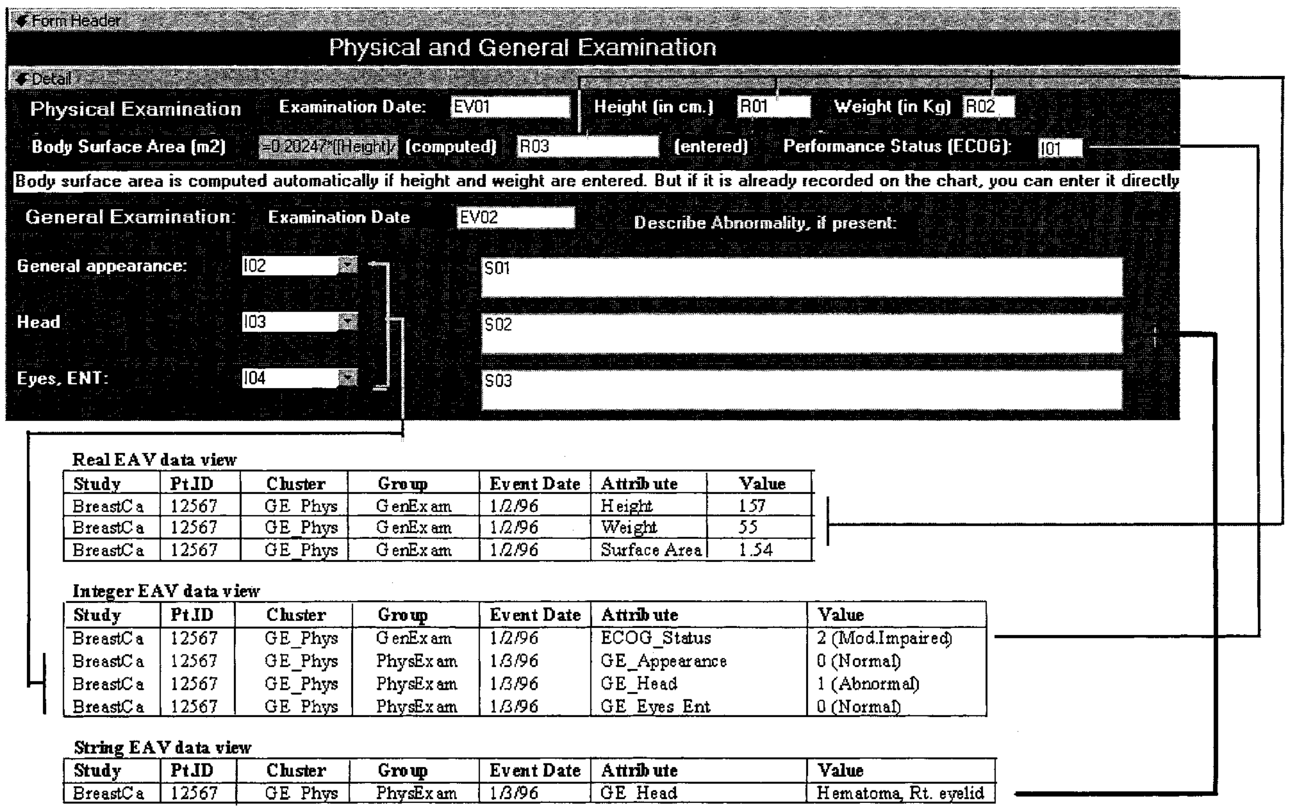
Data Management in ACT/DB

We now describe how ACT/DB manages different kinds of data, as well as controlled vocabularies and the contents of its data dictionary.

Management of Enumerated Data and Controlled Vocabularies

Many clinical parameters are coded or scored for ease of subsequent analysis. For example, a blood product transfusion may be classified as whole blood, plasma, RBC, WBC, platelet, or other. A study designer might choose to codify severity of pain as absent, mild, moderate, or severe. In each case, each identifying phrase is associated with an integer (for example, the codes on the pain scale might be 0, 1, 2, and 3, respectively). "Blood transfusion type" is a nominal parameter (individual codes can only be compared with test if they are the same or different), whereas "pain severity" is ordinal (codes can also be compared for relative magnitude). While numeric codification is useful, it is generally preferable, in the interests of transcription accuracy, to let data-entry persons view and select descriptive phrases.

ACT/DB lets the user designate "choice sets." Each member of a choice set has two components, a numeric code and a descriptive phrase. A choice set can



Form Header

Physical and General Examination

Detail

Physical Examination   Examination Date: EV01   Height (in cm.) R01   Weight (in Kg) R02

Body Surface Area (m2) =0.20247*([Height]/ (computed) R03 (entered)   Performance Status (ECOG): I01

Body surface area is computed automatically if height and weight are entered. But if it is already recorded on the chart, you can enter it directly

General Examination: Examination Date EV02   Describe Abnormality, if present:

General appearance: I02   S01

Head I03   S02

Eyes, ENT: I04   S03

Real EAV data view

Study	PtID	Cluster	Group	Event Date	Attribute	Value
BreastCa	12567	GE_Phys	GenExam	1/2/96	Height	157
BreastCa	12567	GE_Phys	GenExam	1/2/96	Weight	55
BreastCa	12567	GE_Phys	GenExam	1/2/96	Surface Area	1.54

Integer EAV data view

Study	PtID	Cluster	Group	Event Date	Attribute	Value
BreastCa	12567	GE_Phys	GenExam	1/2/96	ECOG_Status	2 (Mod.Impaired)
BreastCa	12567	GE_Phys	PhysExam	1/3/96	GE_Appearance	0 (Normal)
BreastCa	12567	GE_Phys	PhysExam	1/3/96	GE_Head	1 (Abnormal)
BreastCa	12567	GE_Phys	PhysExam	1/3/96	GE_Eyes_Ent	0 (Normal)

String EAV data view

Study	PtID	Cluster	Group	Event Date	Attribute	Value
BreastCa	12567	GE_Phys	PhysExam	1/3/96	GE_Head	Hematoma, Rt. eyelid

Figure 2 Two-way mapping between the fields on a form and the EAV tables in the database. The top half shows part of a form (in Form Design mode, without data). The lower half shows the data that will be created when the fields are populated. Note that 1) The study and patient ID are known because they have been specified in advance from a previous form. 2) This cluster has two groups (GenExam and PhysExam) because they reflect different clinical events, which may be carried out by different persons (e.g., nurse and physician) on different days. 3) Fields in the form map to EAV tables as follows: R01..R03 map to the EAV Real table, I01..I04 to the Integer table, S01..S03 to the String table. Attribute, group, cluster, and study names are shown for simplicity, but unique integer identifiers are actually stored in these places. Also, the information is actually stored in at least three different tables: one for the cluster, one for events within the cluster, and one or more in the attribute-value tables themselves. For simplicity, the three have been combined in the diagram into a single view. 4) The fields EV01 and EV02 store the date/time stamp for the GenExam and PhysExam events, respectively. (In the data shown, these are 1/2/96 and 1/3/96.) 5) Only non-null values are stored in the database. In the example, under General Examination, general appearance and results of the eyes/ENT examination were normal, so the only abnormality to be described was for the Head. Therefore, only the S02 field is populated.

be associated by the designer with a parameter. During form generation, ACT/DB creates a pull-down menu ("combo box" in MS Access parlance) for that parameter. When the pull-down arrow is clocked, the list of descriptive phrases appears and the user can select any one. (This list is dynamic and initialized through a SQL query. If the list is subsequently modified after the form has been created, the list of values appearing when the form is used is current.) The designer can control whether codes and phrases, or only phrases, are displayed in the pull-down menus.

When ACT/DB generates reports, the user has the option of looking at enumerated values as either integer codes or descriptive phrases. The latter are easier to understand, whereas the former more suitable for statistical analysis.

Managing Large Vocabularies. Designer-defined choice sets are adequate when the list of values is reasonably small. However, a pull-down menu interface is neither ergonomic nor efficient over a network when the list of possible values is very large and is derived from an existing controlled vocabulary such as COSTART or SNOMED. To permit the user to search such vocabularies through the data-entry form, ACT/DB lets the designer associate a "search" button on the form with a generic routine.

This routine activates a standard predefined form through which the user can search any controlled vocabulary table or view based on keywords in the descriptive text associated with the code. The user can search for words beginning with or containing a phrase; searches can be combined in complex Boolean fashion using the And, Or, and Not operators. (The user's choices result in generation of SQL that is sent to the server to retrieve matching entries. The code foundation for this routine is based partly on a program developed by the first author called Concept Locator, which was originally built to search the UMLS Metathesaurus.[25])

Computed Parameters

The designer can designate parameters whose value is set through a *computation* based on other fields and define formulae for such parameters. For example, body surface area, used widely in medical oncology for chemotherapy dosing, is a function of height and weight. The designer is given assistance in formula construction (through scrolling lists of available parameters). Nonetheless, specifying such a formula requires some knowledge of Visual Basic programming syntax (even though the "program" is a single-line expression), because the formula is inserted verbatim into the computed field in the generated form. For example, exponentiation in Visual BASIC uses the caret rather than the double asterisk of FORTRAN. This is the one of the rare instance in ACT/DB where the designer may need programmer assistance.

There are times when a value must be computed through a complex subroutine involving branching and looping logic (although we have not encountered such a situation in our present protocols). Microsoft Access is fully orthogonal in that there are no restrictions on the function that may be used in a computed field. Arbitrarily complex functions can be used in a field's formula or even as expressions in queries.

The designer may also allow direct data entry for such parameters. This is a useful alternative when data are being transcribed into the electronic form from an existing, filled-out paper case report. Here, the computed value might have already been entered on a paper case report, but one or more of the parameters required for the computation are missing from the paper form. For example, body surface area is commonly computed at the patient's bedside through nomograms and entered directly on paper.

Laboratory Data and Ranges

As part of its schema, ACT/DB stores information on laboratory tests employed within Yale and the normal values (occasionally stratified by age and sex) for results of each test. These data were gathered from the Yale Clinical Pathology System for work that has been previously described by Kannry et al.,[26] and is used in two circumstances:

- During form generation, ACT/DB creates status bar text, a data entry aid, for each form object representing a parameter. (This text appears on the bottom line of the screen whenever the cursor is in that object.) For most parameters this text is specified by the designer through the data dictionary. For laboratory test parameters, however, ACT/DB generates the text automatically. This text summarizes the units and range of normal values (for male and female subjects). If the normal range varies with age and sex, the maximal and minimal values over the entire range are computed. The status bar text then displays this range and indicates that the range varies with age and sex.
- Based on the age and sex of individual patients (age being computed from date of birth), ACT/DB can generate a report that identifies values that lie within and beyond the normal range.

Our current handling of laboratory values is somewhat limited. For example, normal values may depend on physiologic conditions such as pregnancy and lactation, which are not accounted for by our system. Further, the range of normal values depends to some extent on the laboratory where a test was done (a value of "upper normal" in one lab may mean "high" in another). The Columbia MED employs an object inheritance mechanism whereby tests performed by different labs are different entities within the database, but related entities share a common parent (a "generic" prostatic acid phosphatase, for example). We have not yet had to deal with this level of complexity, but we may have to if ACT/DB ever scales up to handle collaborative interinstitutional trials.

Data Dictionary Management

The ACT/DB data dictionary, which contains the definitions of study protocols, the clusters, and the individual parameters, is constantly consulted during almost every one of ACT/DB's operations. Therefore, in order to scale up to handle a large number of studies across the institution (which implies reuse of existing parameters and forms as far as possible), it must come with tools for management. Our approach is that the data dictionary tables are another vocabulary and therefore should be managed in the same way as external vocabularies. The vocabulary searcher described earlier is used in multiple ways—to identify parameter descriptions containing one or more keywords, all form clusters containing a parameter, and so on.

Further, each cluster is associated with one or more user-defined keywords (which we call *classifiers*) that are used to categorize the cluster. There are no restrictions on what a classifier might be. For now, the classifiers table mostly contains the names of clinical fields (to identify forms that are field-specific, like "oncology"), subject names (like "chemistry") or terms such as "general." When setting up a new study, reuse of existing clusters is facilitated by letting the designer narrow down to a list of clusters described by one or more classifier keywords.

Managing Time-oriented Data

Most clinical databases attach one or more time stamps to every fact entering the database. One kind of time stamp, which is system-generated, records when the fact was entered into the database. Another kind of time stamp, which is user-supplied, records the date/time when the event described by the fact occurred. (In some databases, the fact may be tagged with an additional time stamp that records when the event ended.)

Currently ACT/DB does not meet the definition of a time-oriented database, but it has the basic data structures on which such features may be built. As mentioned above, attributes are aggregated into groups when they apply to the same clinical event. These groups are tagged with start and end date/times. In the data dictionary entry for the group, the protocol designer specifies whether the event recorded is a *period* event (both start and end times) or an *instant* event (start time only; end time is null). For example, a radiotherapy course is a period event. Through a view linking the group with individual parameters in the group, every clinically related fact in an EAV table is, in effect, time stamped. This allows straightforward generation of a chronologic report for a single patient by sorting all events on start and end date/times.

The protocol designer also specifies the prompts for the starting (and, optionally, the ending) date/times for the group and the format of the events—date and time, or date only. This information is used by ACT/DB when generating forms. Either one or two text boxes (depending on the type of event) are created with these labels for date/time entry.

For questionnaire-based outcomes research, as opposed to long-term clinical trials, time stamps as defined here are often not needed on many data items (e.g., "smoking history"), and prompts for their entry on a generated form do not serve any purpose. ACT/DB therefore lets a designer indicate, for individual groups, that *no* date/time information should be associated with them. No prompts for time stamps will be generated for such groups during form creation. (In the database, nulls will be stored against both starting and ending event date/times.) For clinical studies, the use of groups without time stamp information should, in general, be kept to a minimum.

ACT/DB currently lacks support for time-oriented operators and joins, like those described for Das and Musen's CHRONUS system.[27,28] Certainly, adding support for time-based ad hoc queries on EAV data (as opposed to the non-EAV data in CHRONUS) will pose an additional challenge.

Managing BLOB Data

BLOB data, such as histopathology images, are important in patient records and clinical trials: it is convenient to have data online for reference or publication. ACT/DB manages BLOB data display and editing through the object-linking mechanism.[29] For a given BLOB attribute, the protocol designer specifies the application that will handle the BLOB by specifying the file extension to be appended to BLOBs downloaded from the server. (A single BLOB attribute is therefore limited to a single kind of BLOB data. This limitation is not serious.)

BLOBs are uploaded to the server by pasting them into "object frames," which are areas in a form for displaying and editing the BLOB. However, it is inadvisable to send the bytes in the frame itself to the server. Instead, dramatic savings in storage requirements are achieved by locating the BLOB's original file and sending the bytes in this file to the server. This is because disk files storing multimedia BLOB data usually use highly efficient compression techniques (e.g., Joint Photographic Experts Group, or JPEG, images often use compression ratios of 30:1 or more).

BLOB data can also be viewed in the columnar report. This requires viewing the file through Microsoft Excel 97 (which is also commonly used for simple statistical analysis and graphing). When the user includes a BLOB parameter among the list of parameters to be viewed, ACT/DB downloads all BLOB data into a series of sequentially numbered files into a temporary directory. Within the text file itself, each cell representing a BLOB data item contains a formula that uses the hyperlink function to link to the pathname representing the corresponding file. (This function can also link to Web URLs in a similar fashion.) An illustration of BLOB data in a form is shown in Figure 3.

Forms Library Management

In an institution-wide clinical trials database, the forms (data-entry screens) can number several hundred, making them unmanageable without automated assistance. Forms must be stored within the Microsoft Access client application in order to be used in an application. Because forms are developed for use within individual departments, an individual client machine needs to store only those forms actively being used on that machine. Still, there must be a central repository for all forms, especially if forms have been modified with developer assistance. Individual forms may exist in multiple versions, and a version-control system is needed to track these versions.

ACT/DB uses Visual SourceSafe, Microsoft's source code control system, to maintain a forms library on a network-accessible Windows machine that acts as a forms server. (Access 97 is the first version of Access to have hooks to a version-control package.) When a developer needs to change a form, the form is checked out from the server, altered, and checked back in. When an individual client is configured, it can incorporate the current versions of the forms that are needed. (Currently, this process is manual and cannot be done through program code. Fortunately, it needs to be done very infrequently.) Visual SourceSafe also manages version control for program modules, facilitating coordination between multiple developers working on ACT/DB.

Streamlining Client-Server Communication

Certain tables in ACT/DB exist only on the client, and certain tables exist only on the server, while certain server tables of modest size that change very rarely are replicated on the client. The mechanism to coordinate client and server interaction uses a subroutine and template library called SQLGEN,[30] which was originally created to facilitate Sybase (server) and Macintosh 4th Dimension client development.

Two components of SQLGEN are particularly important for ACT/DB. One is an upsizing tool that automatically converts an existing Access schema to an operational Oracle schema. (Microsoft distributes an Access-to-SQL server upsizing tool, but one for Oracle is not, as far as we know, commercially available.) Our tool goes somewhat beyond Microsoft's in functionality. For example, ACT/DB makes heavy use of sequentially generated long-integer unique identifiers (UIs) as primary keys for most tables. (Microsoft Access has a field data type called "autonumber," which increments automatically for each new record.) To port such tables, ACT/DB creates an insert trigger on each table. This trigger fetches and uses the next available sequence number whenever a new record is added. ACT/DB also automatically generates referential integrity constraints.

SQLGEN also facilitates searching of data through a query-by-form (QBF) interface, where the user fills in one or more criteria matching the desired data. The appropriate SQL is then generated and sent to the server. (Microsoft Access's built-in QBF is powerful but not oriented to client–server or networked operation. This is because for every field in the form, a set of unique values is composed by a search of the data to be made available through pull-down menus. This can consume a considerable amount of server capacity as well as network bandwidth.) The details of the SQLGEN QBF facility are described by Nadkarni et al.[30]

Use of Enabling Technology

Many of the interesting features of ACT/DB are possible because of enabling technology that has become available only recently. To cite only a partial list:

- ACT/DB generates forms with GUI components whose visibility and appearance can be programmatically controlled, and associates form component events (e.g., double-clicking) with developer-defined routines. This was not possible with early versions of Microsoft Access.
- For generating standard reports, transformation of EAV server data (irrespective of data type) into cli-

ent arrays that represent an orthodox columnar view of data is simplified by the availability of the "variant" datatype,[17] which can store any kind of data except BLOB. Variants have become available only recently in several languages on the Microsoft Windows platform. Their availability provides the "typeless" convenience that LISP and Smalltalk programmers have long taken for granted.

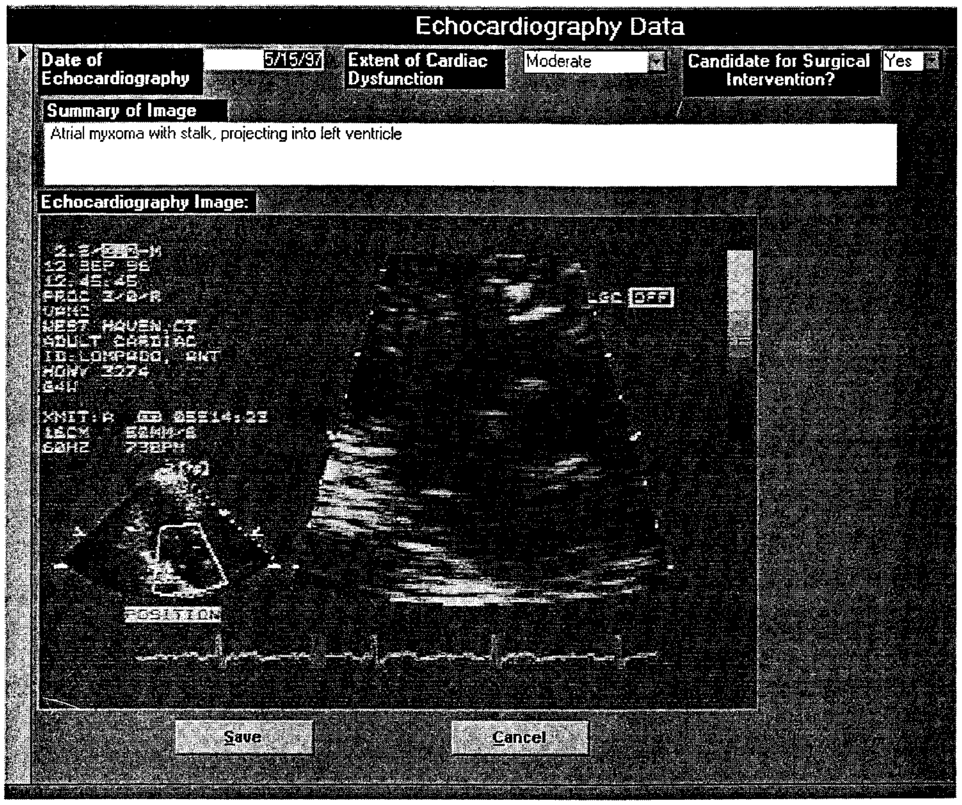
- Object linking makes it feasible to store, display, and edit efficiently any kind of BLOB data without knowledge of their format or the hardware/software codecs (compression–decompression modules) required to manipulate them. It is possible to operate programmatically on the contents of such objects if the object's parent application has been designed for programmatic manipulation through an OLE client. We do not intend to do this within ACT/DB, but freedom to do so might prove important in specialized applications for handling particular kinds of BLOB data.

Status Report

Currently ACT/DB has data for seven studies spanning the domains of oncology, cardiothoracic surgery, and cardiology: there are about 750 attributes across all studies, with the number of attributes per study ranging from 50 to 150. Protocols have been defined and tested for several additional studies that are due to begin soon. We plan shortly to incorporate protocols for multicenter studies conducted by the YCC and affiliated medical institutions in southern Connecticut. This should provide a good test of ACT/DB's capabilities in a wide-area–network scenario. We must emphasize that ACT/DB is in the early stages of deployment, and our data and concurrency loads are modest. There are undoubtedly some inefficiencies in our code, and stressing the system over a wide-area–network will force us to optimize operation.

For the users of ACT/DB, however, efficiency is less important than ease of use, and it is important for us to consider how we can continually enhance ACT/DB's usability. ACT/DB is operated at several user levels. At the simplest (data entry and retrieval) level, clinical data can be added, displayed, and edited through forms or reports. At the intermediate (protocol designer) level, new protocols can be defined, and forms to support these protocols can be generated and customized. The highest (developer/administrator) level permits anything, including code development and schema changes.

Learning to operate ACT/DB at the data-entry level takes only a few hours of training. Learning to operate it at the designer level takes considerably longer to learn. The reasons for this, and the related issues that arise, are as follows:


Echocardiography Data

Date of Echocardiography: 5/15/97
Extent of Cardiac Dysfunction: Moderate
Candidate for Surgical Intervention? Yes

Summary of Image: Atrial myxoma with stalk, projecting into left ventricle

Echocardiography Image:

Figure 3 A sample data-entry form showing a BLOB data field (an echocardiographic still image) along with fields for related descriptive parameters. (Echocardiogram obtained from Dr. Ira Cohen of the West Haven, CT, Veterans Administration Medical Center.)

- The designer must learn how to translate paper forms into electronic forms in the most expeditious way. Well-designed electronic forms can be an improvement on, rather than a mere replica of, paper forms because of such facilities as choice lists and default values. As with any package, there is more than one way to do things, and one way may be better (or worse) than another, depending on the circumstances.
- The designer must learn how to explore the data dictionary to maximize reuse of existing data elements. This skill involves understanding how to use complex Boolean logic, and acquiring it is similar to learning how to use a facility such as MEDLINE. It also involves acquiring something of a compulsive mindset. Before creating a new parameter, one must make a serious effort to search the dictionary to see whether an existing parameter can, in fact, be reused.

The potential advantages of letting designers create their own parameters are high design throughput and instant feedback through automatic generation of forms that are ready for data entry or iterative refinement. Our currently limited manpower resources have forced us to take this route. In the future, however, we may choose to designate a data librarian to work with individual designers, because "pollution" of a dictionary through redundant data items is significantly easier to prevent than undo. A lackadaisical attitude can result in a situation similar to the well-known data-warehousing catastrophe, in which a national pharmacy chain had 20 different spellings or codes for the preparation category "lozenge" in their sales database, making it impossible to consolidate data for analysis without extensive manual cleanup.[31]

- Since ACT/DB's data organization is based on data types, the designer must learn enough about them to be able to designate the appropriate data type for a parameter. The pitfalls of using inappropriate data types (e.g., designating phone numbers as numeric instead of alphanumeric) are well known.
- Finally, the designer must learn about operating Microsoft Access. To avoid reinventing the wheel, we have used existing Access facilities, which are provided through menus, toolbars, and right mouse-button clicks, instead of programming our own alternative user interfaces. For example, Access provides a "binocular" icon to search fields for patterns as well as icons for sorting columns in ascending or descending order, but the user must know how to use such icons productively.

While the ACT/DB designer's manual is being continually revised, our experience is typical of most software developers; that is, most people do not read manuals. We have found it necessary to actually walk individual designers through the process of translating protocols and entering real or test data. This helps them become familiar with the processes of protocol analysis and iterative design refinement. Because we are, in effect, letting the designer create a custom database (with reusable elements), organizing the protocol is similar to (though less difficult than) normalizing a database design. Such a skill is not difficult to acquire, but one must learn to ask the right questions of the data in terms of many-to-one relationships or dependencies.

Discussion and Future Directions

ACT/DB's innovation is in demonstrating the feasibility of partitioning EAV data by data type and in managing BLOB data in the same regular fashion as other EAV data. The ability to index values should speed up value-based queries: Certain recent advances in indexing techniques,[32,33] though not yet commercially available, should benefit such an architecture. Our innovation should be considered evolutionary rather than revolutionary.

There are several design issues that we have not addressed, but which may become important in large-scale production use. Issues related to software engineering rather than research include bulk import of data from laboratory systems (possibly through HL-7) and machine-readable forms, and partial access through the Web rather than through Access clients.

An important research issue is the creation of a robust audit-trail mechanism. We have postponed doing this partly because the vendor tools for audit trails will get progressively more powerful, making our task significantly simpler. For example, the specifications of the language TSQL2 (Temporal SQL)[34] include the designation of transaction tables. Records in a transaction table are automatically time-stamped when created. Furthermore, instead of being altered or deleted by SQL's UPDATE and DELETE statements, respectively, they are merely tagged as "obsolete" and time-stamped when the change or deletion occurs. In this way, a complete history of all the changes to a row is available.

We have already mentioned the problem of ad hoc query of EAV data in arbitrarily complex Boolean fashion. Complicating the query problem are support of temporal primitives and the imposition of a hierarchy or network mechanism (as in the Columbia

MED) to make the query mechanism more powerful by interrelating data elements. Our current representation of data is simple and pragmatic but "shallow," because *semantic* interrelationships between parameters are not stored. At this stage it is too early to predict whether a deeper and richer representation, such as that described by Campbell et al.,[35] will strengthen the query process.

Dr. Vincent de Vita, Director of the Yale Comprehensive Cancer Center, took the initiative in defining the need for an institutional database for clinical trials. The authors thank Dr. Colin Begg, Danny Wu, and Collette Houston of the Department of Biostatistics at the Sloan-Kettering Memorial Hospital.

The ACT/DB schema (as an SQL definition) and Developer's Guide (MS-Word format) are available through ⟨ftp://paella.med.yale.edu/pub/chr12/ACTDB.zip⟩

References ■

1. Rector A. Art and science: problems and solutions (editorial). Methods Inf Med. 1996;35(3):181–4.
2. Cappelleri J, Ioannidis J, Schmid C, et al. Large trials vs meta-analysis of smaller trials: how do their results compare? (review). JAMA. 1996;276(16):1332–8.
3. Pampallona S. A model to control data flow in multicenter clinical trials. Methods Inf Med. 1995;34(3):283–8.
4. Sim IGR. A trial bank model for the publication of clinical trials. Proc 19th Annu Symp Comput App Med Care. Philadelphia, PA: Hanley & Belfus, 1995:863–7.
5. Blumenstein B. Verifying keyed medical research data. Stat Med. 1993;12(17):1535–42.
6. The Inter-company Clinical Quality Assurance Working Group. Computer validation: methods at investigator sites. Appl Clin Trials. 1997;July:36–40.
7. Stokes T. Computer systems validation, part 6: a survive-and-thrive approach to audits and inspections. Appl Clin Trials. 1997;Aug:40–4.
8. Marshall B, Hoffman S, Babadzhov V, Babadzhov M, McCallum R. The Automatic Patient Symptom Monitor (APSM): a voice mail system for clinical research. Proc 17th Annu Symp Comput App Med Care. New York: McGraw-Hill, 1993:32–6.
9. Carlson R, Tu S, Lane N, et al. Computer-based screening of patients with HIV/AIDS for clinical-trial eligibility. Online J Curr Clin Trials. 1995;Mar 28:Doc 179.
10. Pradhan E, Katz J, LeClerq S, West KJ. Data management for large community trials in Nepal. Control Clin Trials. 1994;15(3).
11. Othman R. Interactive database management (IDM). Comput Methods Programs Biomed. 1995;47(3):221–7.
12. ClinTrials Inc. ClinTrials. A brief product description is available from the Web site ⟨http://www.clintrialsresearch.com⟩, 1997.
13. Oracle Clinical Version 3.0: User's Guide. Redwood Shores, CA: Oracle Corporation, 1996.
14. Huff SM, Haug DJ, Stevens LE, Dupont CC. A PT.HELP the next generation: a new client–server architecture. Proc 18th Symp Comput App Med Care. Philadelphia, PA: Hanley & Belfus, 1994:271–5.
15. Friedman C, Hripcsak G, Johnson S, Cimino J, Clayton P. A generalized relational schema for an integrated clinical patient database. Proc 14th Symp Comput App Med Care. Washington, DC: IEEE Computer Society Press, 1990:335–9.
16. Chueh HC, Barnett GO. Client–server, distributed database strategies in a healthcare record system for a homeless population. J Am Med Inform Assoc. 1994;1(2):186–98.
17. Niedner C. Use of SQL with an entity–attribute–value database. MUG Quarterly. 1991;21(3):40–5.
18. Dwight J, Erwin M (eds). Using CGI (special edition). Indianapolis, IN: Que Corporation, 1996.
19. Cimino JJ, Clayton PD, Hripcsak G, Johnson SB. Knowledge-based approaches to the maintenance of a large controlled medical terminology. J Am Med Inform Assoc. 1994; 1:35–50.
20. Johnson S, Cimino J, Friedman C, Hripcsak G, Clayton P. Using metadata to integrate medical knowledge in a clinical information system. Proc 14th Symp Comput App Med Care. Washington, DC: IEEE Computer Society Press, 1990: 340–4.
21. Celko J. Everything you know is wrong. DBMS Magazine. 1996;9(9):18–20.
22. Kimball R. The data Warehousing Toolkit. New York: John Wiley & Sons, 1997.
23. Melton J, Simon AR. Understanding the new SQL: a complete guide. San Mateo, CA: Morgan Kaufman, 1993.
24. Oracle Corporation. Oracle Version 7: PL-SQL Programmer's Guide. Redwood Shores, CA: Oracle Corporation, 1995.
25. Nadkarni PM. Concept locator: a client–server application for retrieval of UMLS Metathesaurus concepts through complex Boolean query. Comput Biomed Res. 1997; in press.
26. Kannry JL, Wright L, Shifman M, Silverstein S, Miller PL. Portability issues for a structured clinical vocabulary: mapping from Yale to the Columbia Medical Entities Dictionary. J Am Med Inform Assoc. 1996;3:66–78.
27. Das AK, Musen MA. A temporal query system for protocol-directed decision support. Meth Inform Med. 1994;33(4): 358–70.
28. Das AK, Musen MA. A comparison of the temporal expressiveness of three database query methods. Proc 19th Annu Symp Comput App Med Care. Philadelphia, PA: Hanley & Belfus, 1995:331–7.
29. Microsoft Access 97 User's Guide. Redmond, WA: Microsoft Corporation, 1997.
30. Nadkarni PM, Cheung KH. SQLGEN: an environment for rapid client–server database application development. Comput Biomed Res. 1995;28(12):479–99.
31. Kimball R. Dealing with dirty data: the science of maintaining clean data in your warehouse, and why nobody talks about it. DBMS Magazine. 1996;9(10):55–6.
32. O'Neil P, Graefe G. Multi-table joins through bitmapped join indices. SIGMOD Record. 1995;24(3).
33. Bontempo CJ, Saracco CM. Accelerated indexing techniques. Database Programming and Design. 1996;9(July): 36–43.
34. Snodgrass RT, Ahn I, Ariav G, et al. TSQL2 language specification. ACM SIGMOD Record 1994;23(1):65–86.
35. Campbell KE, Das AK, Musen MA. A logical foundation for representation of clinical data. J Am Med Inform Assoc. 1994;1(3):218–32.