Application of Information Technology ■

Organization of Heterogeneous Scientific Data Using the EAV/CR Representation

PRAKASH M. NADKARNI, MD, LUIS MARENCO, MD, ROLAND CHEN, MD, EMMANOUIL SKOUFOS, PHD, GORDON SHEPHERD, MD, DPHIL, PERRY MILLER, MD, PHD

Abstract Entity-attribute-value (EAV) representation is a means of organizing highly heterogeneous data using a relatively simple physical database schema. EAV representation is widely used in the medical domain, most notably in the storage of data related to clinical patient records. Its potential strengths suggest its use in other biomedical areas, in particular research databases whose schemas are complex as well as constantly changing to reflect evolving knowledge in rapidly advancing scientific domains. When deployed for such purposes, the basic EAV representation needs to be augmented significantly to handle the modeling of complex objects (classes) as well as to manage interobject relationships. The authors refer to their modification of the basic EAV paradigm as EAV/CR (EAV with classes and relationships). They describe EAV/CR representation with examples from two biomedical databases that use it.

■ **JAMIA.** 1999;6:478–493.

The entity-attribute-value (EAV) approach is popular for modeling highly heterogeneous data. (In the database literature, alternative terms for *entity* and *attribute* are *object* and *parameter*, respectively.) Historically, attribute-value (A-V) pairs were first used in artificial intelligence applications in the form of LISP association lists.[1] Attribute-value pairs are the basis of Web cookies,[2] the Microsoft Windows Registry, and various tagged data interchange formats such as ASN.1.[3] They are also important components of electronic patient record systems (EPRSs), notably the pioneering HELP system[4,5] and the Columbia-Presbyterian clinical data repository.[6,7] An overview of EAV is given in the ''Background'' section.

This paper describes an enhancement to EAV representation, called EAV/CR (EAV with classes and relationships), for modeling heterogeneous data and discusses the types of scientific databases that could benefit from its use. The incentive for creating EAV/CR came from a production database for heterogeneous neuronal data (SENSELAB).[8] SENSELAB was originally implemented as four separate databases with a common Web interface. As the contents of these separate databases expanded in volume and diversity, the number of interdatabase links increased greatly. We realized that the purpose of close interdatabase coupling would be best served by merging the databases into a single physical structure. After discovering that the resulting schema would become very complex and hard to maintain, we decided to transform the data into an EAV schema, for reasons discussed shortly. (We have previously used EAV design for an EPRS-like production system for managing clinical studies data, ACT/DB, [9] which is in production use and continues to undergo enhancement.) The EAV/CR design embodies the necessary enhancements required to the basic EAV model for this purpose. We have subsequently implemented the EAV/CR framework in a prototype database for pharmacogenetics data.

Affiliation of the authors: Yale University, New Haven, Connecticut.

This work was supported by NIH grants U01-CA78266 from the National Cancer Institute, R01-DC03972 from the National Institute of Mental Health, and G01-LM05583 and T15-LM07056 from the National Library 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: 3/26/99; accepted for publication: 6/10/99.

Background

Overview of the EAV Representation of Data

Conventional design is defined, for the purposes of this paper, as one in which each parameter of interest is represented in a separate column in a table. This is the familiar design that most users of spreadsheets or flat-file databases instinctively use to store their data. The number of tables needs to grow as new kinds of data need to be managed.

An *EAV design*, in contrast, conceptually involves a table with three columns—a column for entity/object identification (ID), one for attribute/parameter (or an attribute ID pointing to an attribute descriptions table), and one for the value for the attribute. The table has one row for each A-V pair. In an EPRS, the entity is typically a patient event (a patient ID plus several time stamps recording when the event occurred, began, ended, or was recorded). In a Web cookie, the entity is the cookie itself (so the entity column can be omitted). Theoretically, most of the facts that are recorded in a database can be stored in a single EAV table.

EAV representation is primarily a means of simplifying the *physical schema* of a database, to be used when simplification would be beneficial. However, regardless of the database's physical storage, its users naturally regard the data as conventionally structured—that is, segregated into tables and columns. Furthermore, external programs used for graphical presentation or data analysis always expect to receive data as one column per attribute. The *logical schema* of a database reflects the users' perception of the data.

Because it implicitly captures a significant part of the semantics of the domain being modeled, the logical schema is domain-specific. In an EPRS, for example, one aspect of the logical schema is the grouping of individual parameters into a form, such as a battery of laboratory tests. In an EAV database, the logical schema differs greatly from the physical schema. (In a conventional database, the two are very similar.) The user interface of a good EAV system conforms to the logical schema as closely as possible, creating the illusions of conventional data organization.

An EAV system must record the logical schema through *metadata*—"dictionary" tables whose contents describe the rest of the system. Well-designed metadata are critical to the proper functioning of an EAV system. If sufficiently rich, metadata can also be used actively (i.e., during actual system operation), instead of only describing the system passively. For example, we use EAV/CR metadata to generate SQL for data manipulation as well as Web forms for the user interface. (A significant proportion of EAV/CR metadata is Web-related, because we are committed to this mode of delivery.)

Advantages of EAV Design

The advantages of EAV design are:

- *Flexibility*. There are no arbitrary limits on the number of attributes per entity. The number of parameters can grow as the database evolves, without schema redesign. This is important in the EPRS, where thousands of parameters can apply to a patient across all clinical specialties. With a conventional design, the information would have to be partitioned across an ever-growing list of tables, because of vendor-specified limits to the number of columns per table.
- *Space-efficient storage for highly sparse data.* In an EPRS, while thousands of parameters are applicable, only a few dozen parameters are actually recorded for a typical patient. In an EAV design, space does not need to be reserved for attributes whose values are null.
- *A simple physical data format with partially self-describing data.* This is important for cookies and registries. For example, Windows developers are encouraged to use the registry instead of proprietary formats to store program settings. While the A-V pairs are program-specific, a standard methodology for accessing them makes them accessible to third-party utilities.
- *"Object-at-a-time" queries against a highly complex logical schema that are significantly easier to implement with EAV than with conventional structure.* This is well known in cross-specialty clinical databases that are oriented toward retrieval of individual patients' data. The query "Tell me everything you know about patient X, in reverse chronological order" can be answered by a join of the EAV table with a table of attribute descriptions, filtering the former on the specified patient ID and sorting by time stamp. With a complex conventional schema holding dozens or even hundreds of specialty-specific data tables, on the other hand, each such table would have to be searched, since a patient may have different kinds of diseases over time. Also, as medicine progresses, new tables would be needed to handle protocols to manage new or existing disease conditions (and queries would need manual re-coding).

EAV design is potentially attractive for databases with complex and constantly changing schemas that reflect rapidly evolving knowledge in scientific domains. Here, when a conventional design is used, the tables (and the code that manipulates them, plus the user interface) need continuous redesign with each schema revision. EAV design, by simplifying the schema, may provide relative insulation against such consequences of change. Along with simplification comes the potential for domain independence. With proper design, none of the tables in the system (and only a modest proportion of the code) are domain-specific. This architecture should therefore be portable across scientific domains.

Drawbacks of EAV Design

Most production "EAV" databases also use conventional tables when it makes sense to do so. That is, their schema is heterogeneous. It is therefore important to know how to choose between a conventional and an EAV representation for a given class of data. A rational decision requires an understanding of the drawbacks of EAV design:

- As discussed later, considerable up-front programming is needed to do many tasks that a conventional architecture would do automatically. On the other hand, such programming needs to be done only once, and availability of generic EAV tools could remove this limitation.
- EAV design is less efficient than a conventional structure for bulk retrieval of numerous objects at a time. (For object-at-a-time retrieval, as through a Web browser, the volume of data is small enough that the difference is not noticeable.)
- The process of performing complex attribute-centric queries, which are based on values of attributes, and returning a set of objects is both significantly less efficient as well as technically more difficult. An example of an attribute-centric query in the EPRS context would be "Show me female patients less than 50 years old, with persistently elevated direct bilirubin levels over the last year, whose alanine and aspartate transaminase levels have been consistently normal."
- For schemas that are relatively static or simple (e.g., databases for business applications, such as inventory or accounting), the overhead of EAV design exceeds its advantages.

A Brief Overview of EAV/CR

The terminology of EAV/CR representation borrows heavily from the concepts of object-oriented programming (OOP), because EAV/CR design overlays an object-oriented framework on an EAV physical structure.

- A *class* in EAV/CR design is similar to one in OOP—namely, a complex data structure comprising various fields, some of which may be complex data structures themselves. An EAV/CR class differs from an OOP class in that OOP also enforces the concept of permissible operations on a class. EAV/CR design does not currently do so, partly because it deals with classes of domain-specific data rather than, as OOP does, classes that encapsulate reusable program code or algorithms.
- As in OOP, the term *object* refers to an instance of a class; thus, the object "dopamine" is an instance of the class "neurotransmitter."
- EAV/CR design allows modeling of *interclass relationships*. For example, there is a many-to-many relationship between neuronal cell types and neurotransmitters. In addition, some neurons release a particular neurotransmitter, while others only respond to it.
- EAV/CR design allows *classes to contain other classes as members*. For example, a particular neuron may have muscarinic M1 receptors on its surface. The "M1 receptor" object is an instance of a "receptor" class. (A receptor possesses a protein sequence, and a gene that encodes it. There are agonists and antagonists for a receptor, and it is expressed by many kinds of cells). Therefore, EAV/CR design supports *class instances (Object IDs) as values*.

In summary, database models with complex objects and relationships (represented through conventional database schemas) are widely used. Therefore, extending the EAV model to incorporate these features (as EAV/CR does) is necessary. This extension is especially important for scientific data, but it is also applicable to the EPRS (although some EPRSs, as well as ACT/DB, are focused mainly on patient events as the primary class of object).

The Challenge of Managing Highly Heterogeneous Data

As a system manages an increasing variety of data, the number of classes increases. The complexity of the system grows in two directions.

- Very often, subclasses must be derived from a parent class, because of the need to store additional information with the former. For example, "enzyme" is a subclass of "protein." While an enzyme has an amino acid sequence and three-dimensional structure like its parent, it also requires the recording of special data such as substrates, products, cofactors, and inhibitors.

- When a new class is introduced into an existing schema, possible associations with each existing class must be considered (although not all such associations may make sense). In a conventionally structured schema, each class is represented as a table, while many-to-one or many-to-many associations between classes are managed through "bridge" tables, with foreign keys referencing one or more of the class tables. Here, the number of bridge tables can become unmanageable because, with *M* classes, even if we consider binary relationships alone (i.e., those involving only two classes), the potential number of bridge tables is:

  $$\frac{M \cdot (M - 1)}{2} + M$$

  where the term "+ *M*" accounts for recursive relationships in which both objects belong to the same class, as in a parent-child hierarchy.

  To further complicate the scenario, relationships are not always binary: some relationships involve more than two classes. In pharmacogenetics, for example, one must consider the idiosyncratic effect of a *drug*, which produces a particular *adverse reaction*, in patients with a particular *genotype* (genetic composition).

To summarize, with highly heterogeneous data, we expect to eventually have numerous classes, numerous relationships between them, and a complex class hierarchy. An EAV schema, by definition, allows us to add subclasses without additional tables. Furthermore, as we shall see, an EAV/CR schema represents any number of associations of arbitrary complexity through the same physical structure.

Design Objectives

The EAV/CR framework is intended to fulfill several roles:

- It should fully support basic functions such as browsing and data editing (with some degree of data validation). In particular, it should support on-the-fly generation of Web-based user interfaces based on the metadata contents. It should make full use of the Web interface metaphor—in particular, the generation of hyperlinks where appropriate.
- It should be immune to changes in the logical schema. Only the metadata should change as scientific advances augment the domain knowledge. While the code that accesses the EAV/CR schema is harder to create, it should be generic and require little or no modification as the logical schema evolves.
- It should embody domain independence.

System Description

The EAV/CR framework consists of three components—metadata tables, data tables, and a library of code that manages the user interface and data manipulation tasks. The operation of the code library is controlled by the metadata contents. This paper focuses primarily on the tables. (While the code is not trivial, its operation can in most cases be inferred from knowledge of the table structure.) Where appropriate, we describe how an EAV/CR schema deals with the specific challenges posed by heterogeneous scientific data.

The Data Tables

The data tables within the EAV/CR schema are illustrated in Figure 1. (In all schema diagrams, a table name with the suffix "_1" indicates use of the same table more than once, as when two fields in one table are both related to another table). Before describing the tables themselves, we give a functional overview.

The Object Dictionary Approach

The EAV/CR schema uses the *object dictionary* (OD) approach, where common information on all objects across all classes is stored in a single Objects table, while information that is specific to each class is stored elsewhere. The OD approach is *orthogonal* to EAV/CR; that is, class-specific information can be stored in (numerous) conventional tables or (a few) EAV tables. (We take the latter approach.) In either case, information in these tables is linked to the Objects table through the unique identifier of each object (object ID).

The OD approach was pioneered in bioinformatics databases by Tom Slezak's team during the Lawrence Livermore chromosome 19 mapping project.[10] It has been subsequently adopted in numerous production systems, such as version 5 and subsequent versions of the Human Genome Database (HGD).[11,12] Our group has used an OD in DNA Workbench, which manages physical mapping data within a chromosomal region.[13]

The OD approach allows the use of a supporting synonyms/keyword table to provide enhanced search capability. Synonyms are very common in science (e.g., the terms 5-HT, serotonin, and 5-hydroxy-tryptamine

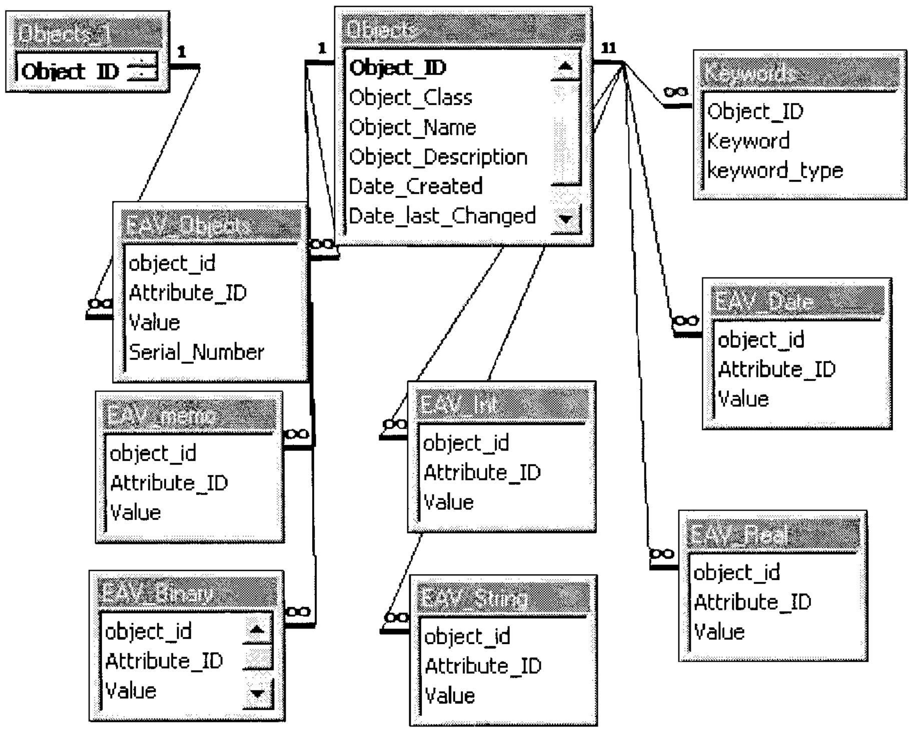

Figure 1 The EAV/CR subschema for data tables. The Objects table stores common information on all objects across all classes of data in the system. It has a supporting table, Keywords, to assist in the search for individual objects. Class-specific data are stored in the tables with an EAV_ prefix and are segregated by data type. In particular, the EAV_Objects table records objects that are components of other objects.

refer to the same neurotransmitter molecule), and any one of several synonyms should be able to locate an object. Furthermore, many objects have unique names even across classes (e.g., "muscarinic" can only refer to a receptor type, and amacrine refers only to a kind of retinal cell). When keywords for all objects across all classes are stored in one place, search tools can locate such objects directly. It is necessary to display all matches, and let the user choose one, only if the same keyword applies to different objects (possibly belonging to different classes).

Virtual Classes

The object dictionary approach would seem to imply that every class must have instances (objects) and that every object in the database must be recorded in the Objects table and given a unique object ID. However, this is not always necessary. In OOP terminology, a class without any actual instances is called *virtual*. We illustrate virtual classes with an example below.

Bibliographic citations are ubiquitous in a scientific database, being tagged to the descriptions of numerous other classes of objects. Because many bibliographic databases, such as MEDLINE, are accessible through the Web, one need not record all the detailed information (title, journal, author list, etc.) on most citations. Only a single value (e.g., MEDLINE UID) is necessary to provide a hyperlink to the desired citation, e.g., through the National Center for Biotechnology Information's PubMed.

From the EAV/CR viewpoint, a MEDLINE citation is a class that is a member of several other classes and has a single attribute (MEDLINE UID). "MEDLINE citation" should be a distinct class, because the attribute information records a URL (unique resource locator) template (as discussed later) that lets us compose the correct hyperlink. If the template needed to be changed, it would have to be changed in only a single place. However, it is not worth creating an object in the database for every MEDLINE citation in the system, because the object ID does not do much more for us than the MEDLINE UID itself. Instead, the MEDLINE UID can be stored directly with the object that it describes. Therefore, we implement "MEDLINE citation" as a virtual class—it has values that are not objects.

Storage of Class-specific Data

EAV/CR representation allows EAV data to coexist with conventional data. In SENSELAB, the only conventional class is a table of references, which is used only to store references without an external bibliographic (e.g., MEDLINE) ID, because the references are very recent or represent personal communications. Most of the records in this table are transient. In the prototype pharmacogenetics database, the genetics-related tables—pedigrees, populations, typing data on individuals, allele information—are stored conventionally, in part because these are components that can be used across databases, and in part because a large body of code depends on them. (For example, we have reused code from other databases previously created by our group, such as PhenoDB,[33] that performs computations such as Hardy-Weinberg frequencies and tests of significance.)

For a heterogeneous (i.e., EAV and conventional) system, it is critical that the metadata record how a class is actually stored, so that the appropriate user interface or data manipulation code can be generated. Generation of such code understandably becomes more complex with mixed data, but in the real world this is unavoidable. (In our system, the code generation for conventional data is partly derived from the earlier SQLGEN library, built by Nadkarni and Cheung.[14]) In this paper, we describe only the EAV tables, which have a generic structure.

The Structure of the Objects and Keywords Tables

The Objects table records the following data for each object.

- **Object ID** (a machine-generated unique ID), **(preferred) object name, object description, date/time of creation**, and **date/time of last change**.
- **Class ID**, the class to which it belongs. This references a Classes metadata table, described later.
- In databases used in large-scale scientific collaborations, such as HGD, the **creator/owner of the object** would also be recorded.

The Keywords table is linked to the Objects table. For each keyword, we record the *keyword type*—synonym or nonsynonym. For simplicity in searching, the preferred name of each object is redundantly recorded in the Keywords table.

EAV Tables for Class-specific Data

EAV/CR representation uses *strong data typing*; that is, when an attribute is defined, its data type is defined as well, and there is a separate EAV table for each data type. (Many EPRSs, in contrast, store all data, even numbers and dates, as short strings.) Strong data typing is used for several reasons.

- Binary large object (BLOB) data, such as nucleotide sequences, chemical formulas, or three-dimensional structural coordinates, cannot be coerced into short strings and therefore need their own EAV table.
- Strong data typing greatly simplifies code generation for form-based (i.e., browser-side) data validation. Browser-side validation, where possible, provides instant feedback. When all errors are detected only at the database server, in contrast, error messages are returned after some delay, the duration of which may be unacceptable over a relatively slow communication link over a wide area network.
- Strong data typing is necessary for support of class instances (object IDs).
- We eventually want to explore the possibility of attribute-centric queries. Ensuring the correctness of attribute-centric queries is greatly facilitated by strong typing. We have previously described attribute-centric query in the context of ACT/DB.[15] In some respects, the attribute-centric query problem for scientific data is more complex, because of the existence of numerous classes as opposed to a single "patient" class. In other respects, it is simpler, because we do not have a stream of time-stamped values for the same object, unlike a series of laboratory values for the same patient over time.

Seven data-type-specific tables record, respectively, integers, reals, short string,* long string, date/time, binary (to handle BLOBS), and object IDs. The last table EAV_Objects, which is doubly related to the Objects table) is essential for allowing objects to become members of other objects, or to let objects associate with each other.

Each EAV table has at least three columns—an **Object (Entity) ID**, an **attribute ID**, (which is linked to an Attributes metadata table that is described later), and **Value**. The EAV_Binary and EAV_Objects tables have an extra field each, whose functions are described later.

Enumerated Values and Booleans

The EAV_Int table in Figure 1 records Boolean values as well as *enumerated values (choice sets)*. The latter are codes that are associated with descriptive phrases,

*The maximum length of a "short string" depends on the database engine—2,000 characters in Oracle, 8,000 in Microsoft SQL Server 7.0, and 255 in Microsoft SQL Server 6.5 and Sybase. The differences between short and long strings are that the former can be indexed, whereas the latter (which have arbitrary length) cannot be. Also, characters in a short string are stored contiguously on disk, whereas characters in a long string are stored as separate "blocks," possibly on different disk sectors, that are chained together.

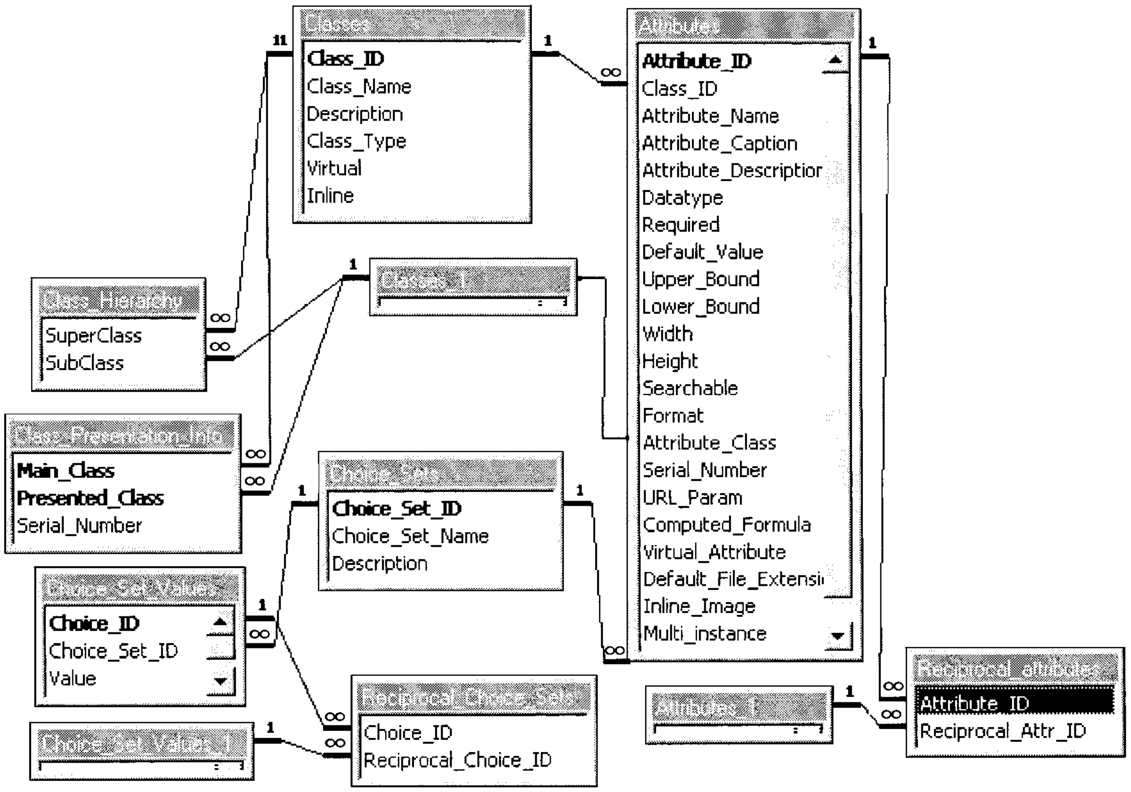

Figure 2 The EAV/CR subschema for metadata tables. The Classes table stores basic descriptions of each class of data, while the Attributes table records details on the attributes of each class. Classes may have subclasses, as recorded in the Class_Hierarchy table. The table Class_Presentation_Info indicates which class data are presented when a particular object is displayed in a browsing interface. The values of some attributes may be derived from controlled-vocabulary items, whose contents are recorded in the tables Choice_Sets and Choice_Set_Values. The tables Reciprocal_Attributes and Reciprocal_Choice_Sets record semantic inverses.

which are displayed in the user interface as pull-down or scrolling lists. Enumerated values are of two types —nominal and ordinal. Ordinals can be compared for relative magnitude, whereas nominals can be compared only for equality. In pharmacogenetics, an example of an ordinal parameter is the clinical importance of a drug effect in patients with a genetic syndrome—theoretically possible, minor, major, life-threatening (with the codes 1 to 4, respectively). An example of a nominal parameter is the broad category of site of drug-syndrome interaction—absorption, distribution, target site, metabolism, excretion, or unknown. A Boolean is a special case of an enumerated parameter that takes just two values, True and False.

The definitions of the sets of enumerated values (and the values themselves, with their associated descriptions) are kept in two metadata tables, Choice_Sets and Choice_Set_Values respectively.

The Metadata Tables: Classes and Attributes

In a database's logical schema, classes are analogous to tables and attributes to fields in those tables. In an EAV/CR schema, the Classes and Attributes tables are the most important part of the metadata and contain not only the logical schema description but also information essential to a Web-based user interface. Their structure is illustrated in Figure 2 and is described below.

The Structure of the Classes Table

The Classes table has the following fields.

- **Class name, Description**, and a unique **Class ID**.
- **EAV_Flag** (Boolean), indicating whether the class is in stored in EAV or conventional form.
- **Class Type**, Entity or Association. (As discussed later, an association between class instances is itself treated as a class.)
- **Virtual** (Boolean), as discussed earlier.
- **Inline** (Boolean). If a class is contained in another class, then when an object belonging to the "container" class is shown in a Web form, the default method for displaying the "contained" object is by

generating a hyperlink. (The hyperlink, which triggers code that retrieves the details of the component object, is appropriate when the component object itself has numerous attributes.) However, if a class has very few fields, the user can be saved a mouse click if the details of the component object are displayed inline, i.e., along with the parent object. Virtual classes are always inline.

The Class_Hierarchy Table

The Class_Hierarchy table has two fields, **Parent Class** and **Child Class**, both linked to the Classes table. It records parent-child relationships between class definitions. It is consulted when the user specifies a query based on a superclass that might encompass subclasses as well. (In pharmacology, for example, the class Drug Family is a parent of the class Drug.) We discuss object and class hierarchies later.

The Attributes Table

The Attributes table has the most detail of all the metadata tables. It has the following fields:

- **Attribute ID** (a unique ID), **Class ID** (points to the class to which the attribute belongs), **Internal Name**, a **Caption** (seen by the user), *Description* (for documentation), and **Serial Number** (order of presentation in a generic Web form).
- **Datatype**, which indicates which of the seven EAV tables is used to store the data.
- **Attribute Class**, applicable only where the data type is Object. Indicates the class of the attribute itself.
- **Defaulted Value** (applicable only for integer, real, string, and date/time data types). If specified, and if the actual value of this attribute does not happen to be stored for a particular instance, then this value is presented to the user. **Upper Bound** and **Lower Bound**, if applicable, are used for data entry validation along with data type.
- **Required** (Boolean). If true, this value must be supplied for a new record.
- **Width and Height**. These numbers, applicable to strings and images, indicate how the attribute is to be displayed in a Web form. (A short string may be displayed either as an INPUT field, if the height is zero, or a TEXTAREA field, where the height indicates the number of columns. A long string will always be displayed as a TEXTAREA.) For numbers and dates, the width computation is based on defaults or on the format (see below), if specified.
- **Format** (picture), a data-type-specific string indicating how a value is to be formatted when displayed—e.g., dates may be shown with date and time or date alone, and real numbers may be displayed with a certain number of decimal places.
- **Searchable** (Boolean). If true, it indicates that a field for this attribute should be included in the search form generated by the system to let the user search for objects within a class on complex Boolean criteria.
- **Computed Formula**. Certain attributes may be computed on the basis of the value of other attributes (if they are non-null). This field holds a Javascript template—an expression with placeholders that are replaced by the values of the appropriate attributes during run time.
- **Virtual Attribute** (Boolean). This is applicable only to attributes with computed formulas. If true, this implies that the computed value is shown on screen but not permanently stored in the database. Notice that most attributes, even those based on formulas, are not virtual, because they need permanent storage. Very often, it happens that data are transcribed from another source, where the value of the computed attribute is available but the values of one or more of the attributes used in the computation are not. In such a case, the computed formula on screen would be null, so we must allow overriding during data entry.
- **URL Template**, a string used to generate a hyperlink to an external data source. The template contains one or more placeholders (vertical bars), which are replaced at run time with segments of the attribute's value. (This value, if it consists of more than one part, is itself a string segmented by vertical bars.)

  For the "MEDLINE ID" example cited earlier, the template is www.ncbi.nlm.nih.gov/htbinpost/Entrez/query?uid=|&form=6&db=m&Dopt=b." The stored MEDLINE ID replaces the vertical bar after the designation "uid=."
- **Default File Extension** (for binary data types only). Binary data are treated by most database engines merely as a stream of bytes. When a binary object is to be served via the Web, the bytes must first be downloaded from the database to the Web server, and a temporary file created on a Web server folder. This file, which is indicated in a Web page via a hyperlink, must have an extension that the client browser recognizes, so that it can call the appropriate program to handle it when downloaded. (For example, files with the extension .WAV are handled by the CD player software on Windows).

The EAV_Binary table differs from the other EAV tables in that the actual file extension is stored for each object if it differs from the default extension. (This allows, for example, more than one image format for a particular attribute.)

- **Inline Image** (Boolean). This is currently applicable to binary image objects only. If true, it causes the image to be placed inline (along with the text), using the IMG SRC= tag. If false (the default), then only a hyperlink is created. (In the latter case, clicking the link would open the image in a separate window.)
- **Multi-instance** (Boolean). This is applicable only when the data type equals Object. This allows an *array of objects*, if needed, instead of a single object alone. (Each object in an array is distinguished by a *serial number* in the EAV_Objects table. Serial numbers are used for ordering the elements during presentation. If ordering is not needed, serial numbers may be left null.) Multi-instance attributes allow us to simulate *non-first-normal form* (NF-NF or NF2), a feature of object-oriented databases,[16] so called because it violates E. F. Codd's first rule of relational database normalization (''repeating groups should be eliminated'').[17] An example of an array of objects in both our databases is a list of bibliographic citations associated with a single object. The use of simulated NF2 arrays is discussed when we consider the EAV/CR representation of associations.

  For multi-instance attributes, we also record **Minimum Repeat Count** and **Maximum Repeat Count**, to specify the minimum and maximum number of times this particular attribute must occur. (If not specified, the number of occurrences is indefinite).

Tables to Manage Reciprocal Semantics

Two metadata tables, **Reciprocal_Attributes** and **Reciprocal_Codes**, are used to store reciprocal semantic information when semantic tags are assigned to objects in a relationship. Their function is described later.

Controlling Web Display of Associated Information on an Object

When an object is displayed in a Web page, all the attributes specific to that object's class must be shown. In addition, however, a class can also participate in one-to-many or many-to-many relationships with other classes. Some of these relationships (which are classes in their own right, as discussed later) are important enough that it is useful to display them whenever an object from a given class is displayed. For example, when basic information on a *genetic locus* (a site on a chromosome responsible for a particular genetic trait) is displayed, one almost invariably wants to look at the variants (*alleles*) for that locus.

The metadata table **Class_Presentation_Info** records, for each class, the associated data from other classes that are to be displayed and the order in which these data are to be presented.

EAV/CR and the Management of Relationships (Associations)

In EAV/CR, associations between class instances (objects/entities) are themselves treated as classes. The management of associations is discussed below.

With neuronal data, associations between objects are often *N*-ary, where *N* is greater than 2. For example, consider the following information on the neurons of the nigrostriatal pathway (whose function is impaired in parkinsonism).

Neurons: nigrostriatal
Anatomic origin: pars compacta of substantia nigra
Projecting to: corpus striatum
Neurochemical/s released: dopamine
Receptors/s involved: D2
Electrophysiological function: inhibitory
Neurons projected to (efferents): striatopallidal neurons, striatostriatal neurons
Neurons providing input (afferents): pars reticularis of substantia nigra, striatonigral fibers

This information is *multi-axial*; that is, each referenced class of object—receptor, transmitter, neuron, anatomic structure, etc.—represents one axis of the data. One method of representing multi-axial associations, widely employed in business data-warehouse design, is the ''star'' schema.[18,19] Here a central Facts table, which stores one or more quantitative columns and one or more coded columns, is related many-to-one to multiple Dimension (class) tables. Each class table stores information on entities in a single axis. This design has proved valuable for such tasks as analyzing pharmacy prescription data by generic drug name, class of drug, physician, specialty, and so on.[20] However, in a scientific database, the implementation of star-schema principles needs considerable modification, for several reasons.

- The *nature* of axes varies with the nature of the fact, necessitating multiple fact tables to hold different kinds of facts when a conventional schema is used. The actual *number* of axes applicable to a particular fact also varies. Many attributes may be null, and others may be irrelevant for certain instances of data. In other words, the data are often sparse.
- Certain axes may have *multiple object instances*. In

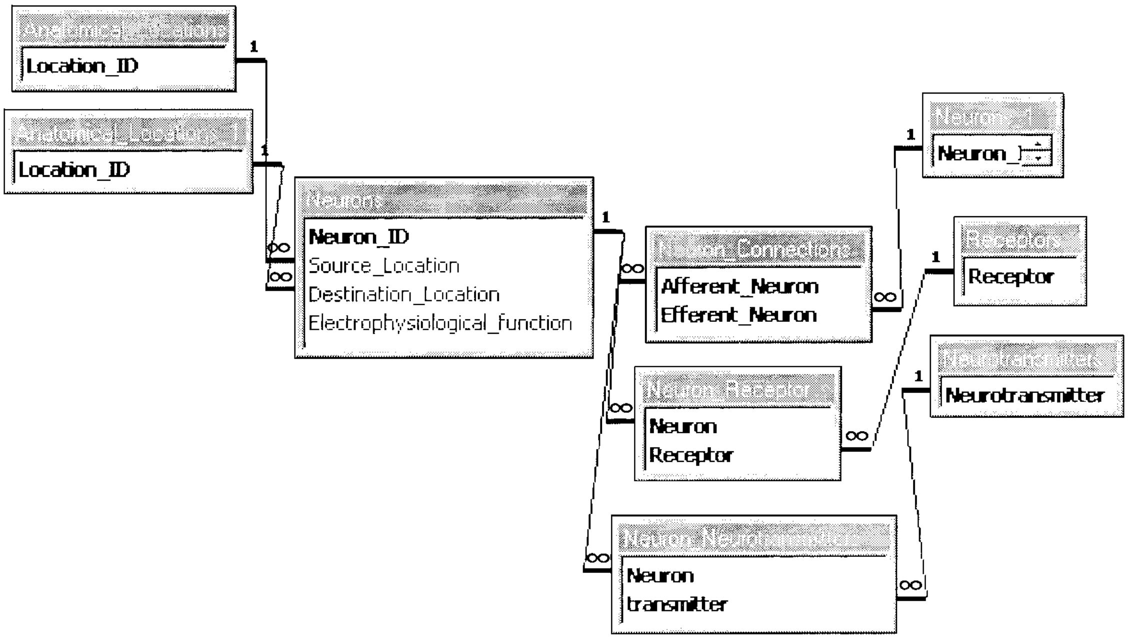

Figure 3 A conventional schema to manage data described in the nigrostriatal neuron example. Even such a small example requires the creation of numerous tables for satisfactory representation.

the example, nigrostriatal neurons both receive inputs and generate outputs for multiple neurons. For nervous system data, in fact, most axes are likely to be multivalued. (For example, a single neuron has multiple kinds of receptors on its surface, and many kinds of neurons are known to release more than one neurochemical simultaneously.) In a normalized relational database design, columns of a table must be atomic and not multivalued, so multivalued data must be factored out into separate tables.

- Within a single axis, entities may be interrelated, through *recursive relationships* of the parent-child type. This complicates the process of query because of the need to pre-explode an object instance specified in a query and retrieve all its children prior to scanning of the association data. In the nigrostriatal neuron example, the pars compacta is part of the substantia nigra, which is part of the midbrain. To process a query that asks for anatomic locations of various receptors in the midbrain, all "child" anatomic sites in the midbrain would first have to be retrieved and then the association data searched against this set of child sites. This exemplifies a recurring problem in scientific databases; that is, *a query specified at a coarser level of granularity must also retrieve facts stored at a finer granularity level*. The standard algorithms for this purpose (which involve determining transitive closure) have been well researched for the "bill of materials problem."[21] Limited transitive closure support is part of the forthcoming SQL-3 standard.[22]

Hierarchic data occur whether the data are in conventional or EAV form. However, we mention them here because their existence necessarily slows down query speed in comparison with the simple, nonhierarchic data typical of a business-warehouse star schema. (On the other hand, the much smaller size of many scientific databases, compared with business data warehouses, means that speed degradation may not be great enough to affect user ergonomics significantly.)

Handling the Nigrostriate Example with Conventional and EAV/CR Schemas

The conventional relational schema for handling the example data is shown in Figure 3. Seven tables are involved (two of them being used more than once). Three of these, indicated by a "Neuron_" prefix, are bridge tables. These bridge tables are needed because, for a given neuronal type, there can be more than one afferent neuron, efferent neuron, neurotransmitter, and receptor. Furthermore, it turns out that two Class tables (Neurotransmitters and Receptors) are physically small (i.e., they have few records) and are likely to remain so, because the numbers of neurotransmitter and receptor molecules are modest. As we start storing different kinds of data, it is not too hard to visualize how the small class tables and bridge tables will proliferate.

One attempt to reduce the number of bridge tables is by using NF2 in a database engine (such as Oracle 8) that permits it. Specifically, each bridge table can be

Table 1 ■

Metadata for the Neuronal_Info Association Class Attributes

Attribute Name	Datatype
Primary_Neuron	Class, Neuron
Soma_location	Class, Anatomical_Location
Axon_Terminus_Location	Class, Anatomical_Location
Neurotransmitter_released	Class, Neurotransmitter, multi-instance
Receptor_Type	Class, Receptor, multi-instance
Electrophysiological_Effect	Integer (member of a Choice Set)
Receptor_Type	Class, Anatomical_Location
Efferent_Neuron	Class, Neuron, multi-instance
Afferent_Neuron	Class, Neuron, multi-instance

NOTE: This table and Table 2 represent the nigrostriatal data example in an EAV/CR framework. This table describes relevant metadata, and Table 2 shows the data. For simplicity, the names of objects and classes are shown instead of their class or object identification.

Table 2 ■

EAV Data for the Nigrostriatal Neuron

Entity ID	Attribute	Value
100	⟨Primary_Neuron⟩	⟨Nigrostriate cell⟩
100	⟨Soma_location⟩	⟨ParsCompacta, S.Nigra⟩
100	⟨Axon_Terminus_Location⟩	⟨Corpus Striatum⟩
100	⟨Neurotransmitter_Released⟩	⟨Dopamine⟩
100	⟨Receptor_Type⟩	⟨D2⟩
100	⟨Efferent_Neuron⟩	⟨Striato-pallidal⟩
100	⟨Efferent_Neuron⟩	⟨Striato-striatal⟩
100	⟨Afferent_Neuron⟩	⟨Striato-nigral⟩
100	⟨Afferent_Neuron⟩	⟨Pars Reticulata, S.Nigra⟩
100	⟨Electrophys_effect⟩	⟨2 = Inhibition⟩

NOTE: This table and Table 1 represent the nigrostriatal data example in an EAV/CR framework. Table 1 describes relevant metadata, and this table shows the data. For simplicity, the names of objects and classes are shown instead of their class or object identification.

replaced with a corresponding *array* field in the Neurons table, resulting in four arrays—afferents, efferents, transmitters, and receptors. In NF2 database implementations, an array holds an implicitly ordered, fixed number of values. A serial number does not need to be stored with each element (as in conventional systems), resulting in space savings that are especially significant for signal data, such as electrocardiographic and electroencephalographic data.

However, true NF2 arrays have the drawback that array elements can neither be individually indexed nor searched as autonomous entities across rows. Therefore, their use is inappropriate when such capability is required. In the present example, the query focus might very well be shifted, for example, to "List all afferents to the striatopallidal neurons." An NF2-array-based table cannot answer such a question efficiently, because the entire table needs to be scanned linearly. (If an index could be used, it would return results in logarithmic instead of linear time.)

The EAV/CR representation (using the schema shown in Figures 1 and 2) is shown in Tables 1 and 2. Table 1 shows the metadata for this association class, "Neuronal_Info." (For brevity, it is assumed that classes such as Neurons, Receptors, and Anatomic Locations, have been previously defined.) The attributes Neurotransmitter_Released, Receptor_Type, Afferent_Neuron and Efferent_Neuron are permitted to be multi-instance. The attribute Electrophysiological_Effect is defined as an integer, whose values are derived from a choice set (e.g., 1 = Excitatory, 2 = Inhibitory, 3 = Modulates Excitation, and 4 = Modulates Inhibition).

The actual data for "Nigrostriate_Neuron_Info" is shown in Table 2. For clarity, we have used the attribute name in the attribute column, rather than the ID that is actually stored. One row (electrophysiologic effect) is stored in the EAV_Int table, while the rest of the rows are stored in the EAV_Objects table. Similarly, we use the names of each object instead of its object ID. (We assume that the data for the D2 receptor, the dopamine molecule, and such have already been created.) While it definitely takes more space than NF2, the EAV/CR representation has the advantage of allowing indexing on the value column.

Names for Entities versus Names for Associations

As stated earlier, in the description of the Objects table, every object has a (preferred) name and description. For objects that are entities, such as "acetylcholine," the name is meaningful. For objects that are associations, however, names are often not very useful. While "Nigrostriate_Neuron_Info" may be meaningful, one is hard-pressed to devise a name for an association with the semantics "benzaldehyde (an odor molecule) increases cyclic AMP (a second messenger molecule) in catfish (a species) melanophores (a tissue)." For such associations, such names as are assigned are necessarily artificial (they may even be machine-generated). It is the description field that is all-important, because the semantics of the association can always be described in narrative form.

The Need to Manage Inverse Semantics

In a database, an association between *N* objects is similar to a miniature semantic network.[23] A semantic network is defined as a directed graph data structure where a set of **nodes** (vertices)—in this case, the objects—are connected by **edges** that are assigned semantic labels. "Directed" means that an edge from node A to node B is not equivalent to an edge from node B to node A. In the case of a semantic net, the former edge is the *semantic inverse* (or reciprocal) of the latter. For example, the statement "Iron *absorption is increased by* vitamin C" can also be stated as "Vitamin C *increases the absorption of* iron." The semantic inverses (indicated by italics) result in the subject and object of the statement being reversed. Some concepts are their own semantic inverse, as when two drugs mutually reduce the absorption of each other.

Controlled-vocabulary semantic tags are extensively used in scientific databases (e.g., the Human Genome Database). Their use is orthogonal to EAV/CR per se. When the only kinds of interobject associations are binary, the same fact can be stored twice (with subject and object reversed). Redundant storage results in faster search speed (this is a well-known example of a "space-for-time" tradeoff), and therefore several production databases use it. For example, if a fact were stored only once, both "subject" and "object" columns would need to be scanned to locate all information on iron. This is because it would not be known, a priori, whether "iron" was the subject or the object of any given fact. If however, we stored the fact twice, we would need to inspect only one of the two columns.

For *N*-ary associations, however, the number of ways to represent the same fact can increase nonlinearly with *N*. For example, consider the different ways the fact "Chemical X augments the transformation of substrate Y by enzyme Z" could be expressed. In such circumstances, it becomes necessary to represent semantic inverses explicitly. Therefore, the EAV/CR metadata have a table (Reciprocal_Choice_Sets) for this purpose. (Several well-known systems in medical informatics, such as the MED—the Columbia Presbyterian Medical Center Medical Entities Dictionary[24]—also manage semantic inverses.)

In addition to inverse semantic tags, one can also conceive of *inverse attributes*, especially when objects in an association come from the same class. (This is an artifact of the way databases are designed, with some facts being represented explicitly as semantic tag, and others implicitly and space-efficiently through column/attribute definitions.) In the nigrostriatal neuron example, stating that the nigrostriatal neurons send their output to the (*efferent*) striatopallidal neurons is the same as stating that the striatopallidal neurons receive their input from the (*afferent*) nigrostriatal neurons. By explicitly managing definitions of inverse attributes, a query such as "List the afferents to the striatopallidal neurons" would correctly translate facts from the nigrostriatal neuron data even though neither striatopallidal nor nigrostriatal neurons are described under the tag "afferent."

To summarize, inverse semantic concepts allow associations to be stored just once and permit automatic broadening of queries that are partially based on semantic tags as query criteria.

N-ary Associations as Inverted-file Indexes

While semantic labels impose a rigor on the definition of a given association, the number of possible labels becomes very large in a large and complex domain (such as the nervous system). In many cases, moreover, many semantic edges that connect objects, while necessary for completeness, unnecessarily recapitulate facts of basic biology already known to the database's users. In the previously described benzaldehyde example, the semantic links "catfish *has-a-tissue* melanophores" and "melanophores *contain-a-messenger* cAMP" are uninteresting to a biologist; the only link of interest is that between odor molecule and messenger ("increases"). In such cases, the narrative prose in the association description is more concise than any graphic representation of a semantic net, because uninteresting links can be left unrepresented.

For most database purposes, therefore, the full rigor of semantic nets is generally not required, because we are not performing inferences on the association data. However, retrieval of associations containing one or more objects of interest must be rapid. It is here that the EAV_Objects table is useful. To borrow an analogy from text information retrieval (IR), the EAV_Objects table can be regarded as an *inverted-file index*[25] to the narrative prose. The difference between IR indexes and the object IDs that are the values in EAV_Objects are that the latter represent a somewhat more rigorous controlled vocabulary (because each object belongs to a specific class).

Implementation Overview

This section gives a brief overview of how the EAV/CR implementation operates. (We will shortly document our code library in detail and put in on the SENSELAB Web site, whose URL is given later.)

The generated code (including Web forms) operates at two levels—the Web server and the Web browser. The Web server component is Active Server Pages (ASP) scripts (generated in the Visual Basic scripting language), which are specific to Microsoft Internet Information Server 4.0 and Windows NT. While CGI (common gateway interface) scripts are the older, "universal" technology for server-side programming, we have found ourselves considerably more productive with ASP. Also, because ASP files are simply standard HTML files with embedded scripting code, the generated forms can be partly edited and visually enhanced through Web-page design programs, if desired.

Currently, the user interface relies on the latest versions (version 4 and greater) of both Netscape and Microsoft browsers. It makes extensive use of dynamic HTML (through client-side Javascript) to perform tasks such as simple data validation or to allow or deny edits to some fields on the basis of the contents of others. However, in the future, as we add more powerful features, we may be forced to decide how much cross-platform support we are prepared to do. Netscape's implementation of dynamic HTML lags greatly behind Microsoft's, and the two use vastly different (largely incompatible) underlying models, so writing common-denominator client-side code is very difficult, and not always possible. In the ongoing ACT/DB project, from which some of the code was borrowed, we were servicing a restricted set of users and so had mandated a Microsoft browser platform. This was because we needed certain features of Internet Explorer that Netscape Navigator lacked, such as the ability to add or delete rows from HTML tables (which simulate subforms) without making a round trip to the server.

The database engines used are Microsoft Access 97 (for prototyping) and Oracle 7.3 (for SENSELAB production) and Microsoft SQL Server (for the prototype pharmacogenetics system). We have, however, tried to use ANSI-standard SQL rather than vendor-specific syntax to make the code as portable as possible. Where this has not been possible (e.g., Oracle handles dates idiosyncratically), we use ODBC ''escape sequences'' within the SQL. (ODBC, for open database connectivity, is a Microsoft standard for vendor-independent database access.) The ODBC driver that mediates communication between the application and the database automatically translates the escaped strings into vendor-specific code.

Status Report

The SENSELAB unified database is publicly accessible through the Web via the URL http://fondue.med.yale.edu/senselab/. It is focused on various aspects of the olfactory system (a major research focus of Dr. Shepherd's laboratory). The four categories of SENSELAB data correspond to the databases that the system originally comprised.

- *NeuronDB*[26] stores data on various neuronal cells—receptors, neuronal currents and neurotransmitters—and interneuron connectivity.
- *ModelDB*[27] stores computational models of neuronal function.
- *ORDB* (olfactory receptor database)[28] holds amino acids and nucleotide sequences, researcher and laboratory information, and hyperlinks to other Web-related data.
- *OdorDB* (odors database) stores chemical, biological, and experimental data on odor molecules, neurotransmitters, second messenger molecules, electrophysiologic behavior, and cell types.

In addition, we expect to incorporate neural circuitry data and three-dimensional functional magnetic resonance images of the olfactory cortex. The three-dimensional data in particular will provide a serious test of whether the EAV/CR framework is truly general (or if not, what work needs to be done).

Porting of the contents of these databases was straightforward, although somewhat tedious. (These databases resided within various engines, such as Sybase, Illustra, and Microsoft Access.) We describe the process below.

- We first exported the table and column definitions in the individual database schemas into tabular text, which was then imported into the Classes and Attributes tables of Microsoft Access database that contained the EAV/CR schema. Some fields, notably foreign-key fields that referenced other tables, were then manually edited. Duplicate definitions of classes (e.g., the neuron class was defined in more than one database) were manually removed.
- Once the metadata were defined, the contents of individual tables in the original databases were then imported into corresponding tables in the Microsoft Access database. At the end of this step, we had a conventional unified schema coexisting with an EAV/CR schema whose EAV data tables were empty.
- We then wrote a series of short conversion scripts (incorporating SQL) that reformatted the data into EAV form table by table. Data export was phased. ''Primitive'' classes (which did not contain other classes) were exported first. Unique IDs were assigned to the objects created after loading into the Objects table. These IDs were then referenced, where appropriate, in the next phase, which consisted of importing objects that referred to objects from other classes.
- The database was stripped of conventional tables and eventually upsized to Oracle.

The prototype pharmacogenetics database contains a very modest set of test data. One reason for exploring pharmacogenetics concurrently with neurobiology was to identify potential problem areas in EAV/CR that implementation for a single domain might not have revealed. (Also, by using an alternative database engine, we were able to consider ways of making database-access code generation vendor-independent.)

Discussion

From a historical perspective, EAV/CR is the inverse of Jeffrey Ullman's universal database (UDB) concept.[29] UDB aims to create the illusion of a single table in a database, thus sparing the user from having to specify intertable joins during ad hoc query. In other words, *UDB simulates a simple logical schema in the presence of a complex physical schema.* (UDB is somewhat limited, because certain queries, such as those using self-joins, are ambiguous unless explicit joins and aliases are specified.[30]) *EAV/CR representation, on the other hand, simulates a complex logical schema by using a simple physical schema.*

The EAV/CR schema overlays a formal object-oriented framework on the basic EAV model, through the definition of classes and the permissible attributes that each class may contain. Scientific databases differ widely in purpose and scope, however, and EAV/CR design is not presented as a panacea. It is necessary to define the kinds of scientific databases for which it appears suited.

- EAV/CR design is not currently intended to support temporal logic. While time-based signal data are very common in experimental science, their processing is independent of the EAV/CR schema. Relational database management systems (RDBMSs) are not intended to store each point in a signal tracing as an individual record in any case. Most RDBMSs currently treat a signal data file as a binary large object. In other words, the file can be stored and retrieved as a whole, but analysis of its contents is the responsibility of special routines. (Electronic patient record systems, in contrast, must support temporal logic directly or indirectly. Every parameter recorded for a patient is time-stamped. Medical logic modules[31,32] often require data on the temporal course of clinical parameters.)
- EAV/CR design is not intended to directly support the specialized experiments of a particular laboratory that use a specific methodology. While nothing prohibits its use for this purpose, the EAV model offers no special advantages for a logical schema that may not be particularly complex, especially if the application can be built using conventional user-interface technologies with little or no programming. If, however, a laboratory needs to integrate diverse data into a single database, EAV/CR will be useful.
- EAV/CR design is especially appropriate for recording summaries of experiments, and is advantageous when experiments involving a variety of different methodologies (in a research consortium, for example) must be summarized. EAV/CR design can similarly be used to record archival or historical data covering various aspects of a particular scientific domain. In all these situations, the "goodness of fit" of an EAV/CR solution increases with the number of classes and the interclass or intraclass relationships. With a smaller number of classes (e.g., ten or less) in a schema that is not likely to evolve greatly, EAV/CR design would be overkill, and a conventional design should be used.

The scientific contribution of this paper is the demonstration that, with sufficiently rich metadata, an EAV system is a viable alternative in the planning of certain types of scientific databases. The importance of metadata in various branches of informatics has been progressively growing over the years. RDBMSs record a database's structure (in addition to stored procedure code) in "system" metadata tables. These tables perform tasks such as type checking, referential integrity across tables, and field level validation constraints. They are consulted whenever an SQL command is checked for semantic correctness or ambiguity. However, RBDMS system metadata have several limitations.

- The structure varies greatly across vendors, because standardization efforts have lagged. (This is one area where vendors compete). Several DBMSs do not even permit adding descriptive comments to the definition of a table or field.
- Client-server RDBMSs have traditionally been concerned only with data management and not with the user interface (a task relegated to client software). Consequently, they do not store enough metadata for a complete working application. Furthermore, their system tables are not extensible, and so client-development packages (e.g., PowerBuilder) have had to define their own tables for this purpose. (PowerBuilder pioneered the concept of "extended attributes," such as the visual formatting desired for individual fields.)

Microcomputer DBMSs such as Microsoft Access do store much richer metadata, but thus far the metadata are oriented toward generation of vendor-specific client interfaces rather than Web interfaces. (While Access can generate Web forms, these forms currently make extensive use of client-side ActiveX controls, which can run only on Wintel platforms. Furthermore, the security model of ActiveX is very limited. While client-side ActiveX controls make sense in an intranet setting, many users and system administrators are wary of downloading such controls from a less trustworthy or unknown site across the Internet.)

In any case, for an EAV architecture, system metadata are completely uninformative, since the semantics of the system lie in the *contents* of the data rather than the *structure* of the tables and fields. Therefore, any EAV system needs to provide its own metadata. To some extent, creating an EAV framework to do many tasks that a conventional architecture would do automatically—such as type checking and referential integrity—represents wheel reinvention. In circumstances where direct data entry and editing must be supported, however, this is unavoidable and represents part of the price one pays for EAV. (Many EPRSs are able to bypass this issue completely, because their data is bulk-imported from several conventionally structured systems and then changed to EAV form. The imported data have presumably passed the checks that were built into their parent systems.) One of the motivations behind the creation of EAV/CR was that, if generic code could be written for such tasks, one would be saved the trouble of writing tedious and ad hoc code on a case-by-case basis.

Future Directions and Conclusions

The EAV/CR schema is a work in progress. While EAV/CR design incorporates the basic functionality necessary to the creation of a Web-based interface, much remains to be done. Future research will take several directions.

- Database developers are increasingly considering standardization of approaches to modeling schemas and metadata. Support appears to be converging on Rational Software's Unified Modeling Language (UML).[34] The specifications for UML are freely available, and it subsumes older methodologies such as entity-relationship modeling.[35] The developer expresses the metadata through diagrams that have a textual equivalent. Computer-assisted design packages such as Visio work with UML diagrams, and several vendors (including Rational Software) sell programs that will convert a UML specification into a database schema. UML is also supported by Microsoft and is used in the Microsoft Repository.

  Given that UML may well become a de facto metadata standard, we are actively exploring the possibility of using a UML specification to populate most of the metadata in an EAV/CR database. This would greatly facilitate the automated conversion of existing conventional schemas into EAV/CR equivalents.

- We will eventually need to create a query generator for attribute-centric queries. In our experience, most queries to scientific databases are relatively straightforward and are directed toward specific objects of interest (i.e., they are "object-centric"). The SENSELAB Web site currently lacks attribute-centric query but still provides access to its information quite intuitively.† However, as the scope and volume of data grow, attribute-centric query generation and processing will become important.

- As the system grows, it will eventually become necessary to provide tools to search the metadata robustly. One means for doing this is to implement a keywords and synonyms table that allows alternative descriptions or names for classes and attributes. It may also be necessary to provide more than one caption for an attribute (e.g., a brief caption and a long caption), so that one or the other could be used appropriately.

- We will need to stress-test the ability of EAV/CR design to deal with larger volumes of data. Our work, as described here, should provide useful guidelines to others who are designing scientific databases, and we invite scientific collaborations from such researchers.

The authors thank Carol Friedman, PhD, of Columbia Presbyterian Medical Center, for a discussion highlighting the differences between EAV/CR and the universal database concept. Daniel Masys, MD, of the University of California–San Diego, identified the importance of the disparity between the physical and logical schemas in an EAV database.

References ■

1. Winston PH. Artificial Intelligence. 2nd ed. Reading, Mass: Addison-Wesley, 1984.
2. Dwight J, Erwin M (eds). Special Edition: Using CGI. Indianapolis, Ind: Que Corporation, 1996.
3. Huff SM, Rocha RA, Solbrig HR, Barnes MW, Schrank SP,

†The history of public genome-related databases also supports our experience. Both NCBI's Entrez and the Human Genome Database provide object-at-a-time access through their Web front ends, and their obvious success seems to imply that most users don't care about complex query. (On the other hand, for the small minority of scientists doing hard research on large subsets of the data, complex query is essential.)

Smith M. Linking a medical vocabulary to a clinical data model using Abstract Syntax Notation 1. 1998;37:440–52.
4. Huff SM, Berthelsen CL, Pryor TA, Dudley AS. Evaluation of a SQL model of the HELP patient database. Proc 15th Annu Symp Comput Appl Med Care. 1991:386–90.
5. Huff SM, Haug DJ, Stevens LE, Dupont CC, Pryor TA. HELP the next generation: a new client–server architecture. Proc 18th Annu Symp Comput Appl Med Care. 1994:271–5.
6. Johnson S, Cimino J, Friedman C, Hripcsak G, Clayton P. Using metadata to integrate medical knowledge in a clinical information system. Proc 14th Annu Symp Comput Appl Med Care. 1990:340–4.
7. Friedman C, Hripcsak G, Johnson S, Cimino J, Clayton P. A generalized relational schema for an integrated clinical patient database. Proc 14th Annu Symp Comput Appl Med Care. 1990:335–9.
8. Shepherd GM, Healy MD, Singer MS, et al. SENSELAB: a project in multidisciplinary, multilevel sensory integration. In: Koslow SH, Huerta MF (eds). Neuroinformatics: An Overview of the Human Brain Project. Mahwah, NJ: Lawrence Erlbaum, 1997:21–56.
9. 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):139–51.
10. Slezak T, Wagner M, Yeh M, et al. A database system for constructing, integrating, and displaying physical maps of chromosome 19. In: Hunter L, Shriver BD (eds). Proceedings of the 28th Hawaii International Conference on System Sciences. Los Alamitos, Calif: IEEE Computer Society Press. 1995:14–23.
11. Fasman KH, Letovsky SI, Cottingham RW, Kingsbury DT. Improvements to the GDB human genome data base. Nucl Acids Res. 1996;24(1):57–63.
12. Letovsky SI, Cottingham RW, Porter CJ, Li PWD. GDB: the human genome database. Nucl Acids Res. 1998;26(1):94–9.
13. Nadkarni PM, Cheung K-H, Castiglione C, Miller PL, Kidd KK. DNA workbench: a database package to manage regional mapping. J Comput Biol. 1996;3(2):319–29.
14. Nadkarni PM, Cheung KH. SQLGEN: an environment for rapid client-server database application development. Comput Biomed Res. 1995;28(12);479–99.
15. Nadkarni P, Brandt C. Data extraction and ad hoc query of an entity-attribute-value database. J Am Med Inform Assoc. 1998;5(6):511–27.
16. Bancilhon F, Delobel C, Kanellakis P. Building an object-oriented database system: the story of O2. San Mateo, Calif: Morgan Kaufmann, 1992.
17. Date CJ. An Introduction to Database Systems. 7th ed. Reading, Mass: Addison-Wesley, 1991.
18. Kimball R. The Data Warehousing Toolkit. New York: John Wiley, 1997.
19. Inmon WH. Building the Data Warehouse. New York: John Wiley, 1996.
20. Kimball R. Help for Dimensional Modeling. DBMS Mag. 1998;11(9):14–7.
21. Goodman N. Bill of Materials in Relational Database. InfoDB. 1990;5(1):2–13.
22. Melton J (ed). ISO-ANSI Working Draft Database Language SQL (SQL3). ISO / IEC SQL Revision. New York: American National Standards Institute, 1992.
23. Russell S, Norvig P. Artificial Intelligence: A Modern Approach. Englewood Cliffs, NJ: Prentice-Hall, 1995.
24. Cimino JJ, Hripcsak G, Johnson SB, Clayton PD. Designing an introspective, multipurpose, controlled medical vocabulary. Proc 13th Annu Symp Comput Appl Med Care. 1989: 513–8.
25. Salton G. Automatic Text Processing: The Transformation, Analysis, and Retrieval of Information by Computer. Reading, Mass: Addison-Wesley, 1989.
26. Mirsky JS, Nadkarni PM, Healy MD, Miller PL, Shepherd GM. Database tools for integrating neuronal data to facilitate construction of neuronal models. J Neurosci Methods. 1998;82(1):105–21.
27. Peterson B, Healy M, Nadkarni P, Miller P, Shepherd G. ModelDB: an environment for running and storing computational models and their results applied to neuroscience. J Am Med Inform Assoc. 1996;3(6):389–98.
28. Healy MD, Smith JE, Singer MS, et al. Olfactory receptor database (ORDB): a resource for sharing and analyzing published and unpublished data. Chem Senses. 1997;22:321–6.
29. Ullman JD. Principles of database and knowledge-base systems. Rockville, MD: Computer Science Press, 1989.
30. Date CJ. Selected Database Readings, 1985–1989. 7th ed. Reading, Mass: Addison-Wesley, 1990.
31. Hripcsak G. Writing Arden syntax medical logic modules. Comput Biol Med. 1994;24(5):331–63.
32. Hripcsak G, Ludemann P, Pryor TA, Wigertz OB, Clayton PD. Rationale for the Arden syntax. Comput Biomed Res. 1994;27(4):291–324.
33. Cheung K-H, Nadkarni P, Silverstein S, et al. PhenoDB: an integrated client / server database for linkage and population genetics. Comput Biomed Res. 1996;29:327–37.
34. Rumbaugh J, Jacobson I, Booth G. The unified modeling language reference manual. Reading, Mass: Addison-Wesley, 1999.
35. Gogolla M. An extended entity-relationship model: fundamentals and pragmatics. New York: Springer-Verlag, 1993.