The EAV/CR Model of Data Representation

The Bird's Eye View

I should warn you that the name "EAV/CR", on occasion, can be somewhat of a misnomer. EAV/CR originally stood for **"entity-attribute-value with classes and relationships"** and in some of the databases that we have implemented, this definition still holds. However, depending on the database that you are building, **some, or even most, of the classes of data are best represented as old-fashioned relational tables**. EAV/CR is really defined by its **metadata infrastructure** and the use of an **Object Dictionary** table rather than the fact that the representation of data representing individual classes of objects may or may not be in EAV form.

In production systems, using EAV/CR is something like driving a car using the stick shift. It gives you more control than using automatic transmission, but is also somewhat trickier than using old-fashioned design approaches. Therefore, you use it only when you absolutely need to.

In the account below, we use the word "class" as the equivalent of "database table" and "attribute" as the equivalent of "database column". While this usage is somewhat lax, this is what you end up doing in the vast majority of cases when you are using a relational database to store object-modeled data.

EAV/CR is an approach that seeks to facilitate the rapid development of interfaces to data (and/or their dynamic creation) through the recording of "rich" metadata that not only describes every element in the database from a "database" perspective, but also from a "presentation" and "user interaction" perspective.

The word "rapid" comes with a caveat- in earlier editions of their classic "Camel" book on Perl, Larry Wall and Randall Schwartz describe laziness as an attribute of truly great programmers - they first build a framework (the hard part, which is slow going), and then use the framework to automate chores (the easy/rapid part, which allows laziness). EAV/CR is the basis for such a framework.

By "database" perspective, we include info such as data type, constraints, relationships between classes, and so forth

Presentation/user interaction metadata encompasses details such as:

- How a conceptual column in the database is to be displayed: e.g., as a text box, a scrollable text area, listbox, combo.
- If the column represents a foreign key, how the values are to be searched. A combo/list box is an appropriate visual metaphor when the list of possible values does not exceed, say, 30. It is inappropriate for searching through 3000 values, and here, you want the system to place a "search" button that lets you search the "primary key" table equivalent either by bringing up a "search" form (where you query this table based on particular fields of interest", or in Google fashion, where you type in one or more keyphrases, and the system searches all the fields that have been designated as "Help" fields.
- When you present a single record from a particular class in a "form view", you typically also want to present related (many-to-one) records from other classes, using a "sub-form" visual metaphor. You may wish to drill down into the details of an individual related record (by opening up a new form for that class).

It would be convenient if you could build such interfaces automatically, by specifying presentation metadata in great detail and then letting the system generate an interface for you. This is what EAV/CR is about: in addition, the metadata serves as detailed and rigorous documentation for the system and its semantics. It also makes data interchange possible with programs that expect conventional (one-attribute-per-column) table structure.

The interface that is generated may be completely dynamic, or may have some static elements - the former is usually OK for browsing: the SenseLab Web site is mostly dynamic. The latter is usually required when you are doing data entry with some elaborate data validation - generating all the validation code every time the class is accessed may be too much of an overhead, and a Web page with static elements improves response. The TrialDB Web site mostly uses pages with static elements.

In brief, in a three-tier database application, the EAV/CR schema allows the middle (business logic) tier

to consult the metadata to perform many chores automatically that would otherwise have to be coded on a per-table or per-column basis.

History

- EAV itself was first used in artificial intelligence applications in the form of LISP association lists .
- EAV structure is the basis of Web cookies , the Microsoft Windows Registry, and various tagged data interchange formats such as ASN.1. XML can be regarded as a "tagged" form of EAV (with "open-attribute" and "close-attribute" tags, which supports nesting of attributes to an arbitrary degree.
- Important components of Electronic Patient Record Systems (EPRSs), notably the pioneering HELP system and the Columbia-Presbyterian Clinical Data Repository .

Definition

EAV = Entity-Attribute-Value. EAV/CR = EAV with Classes and Relationships

Conceptually, a table with three columns:

- Entity/Object ID
- Attribute/Parameter
- the Value for the attribute.

The table has one row for each Entity-Attribute-Value triple.

In reality, we prefer to segregate values based on data type, so as to support indexing and let the database perform type validation checks where possible. So there are separate EAV tables for strings, real and integer numbers, dates, long text and Binary large objects (BLOBS).

EAV is known under several alternative names: object-property-value by the O-O folks, frame-slot-value by the AI folks. When implemented in databases, an EAV table is called a "generic" table, or an "open schema" table. Resource Description Format (RDF), the basis for "Semantic Web" content, uses the same approach: RDF content consists of "Subject-Predicate-Object" triples.

Benefits

- 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. (Important in the EPRS)
- Space-efficient storage for highly sparse data: One need not reserve space for attributes whose values are null.
- A simple physical data format with partially self-describing data. Maps naturally to interchange formats like XML (the attribute name is replaced with start-attribute and end-attribute tags.)
- For databases holding data describing rapidly evolving scientific domains, insulation against consequences of change and potential domain independence .

Physical vs. Logical Schema

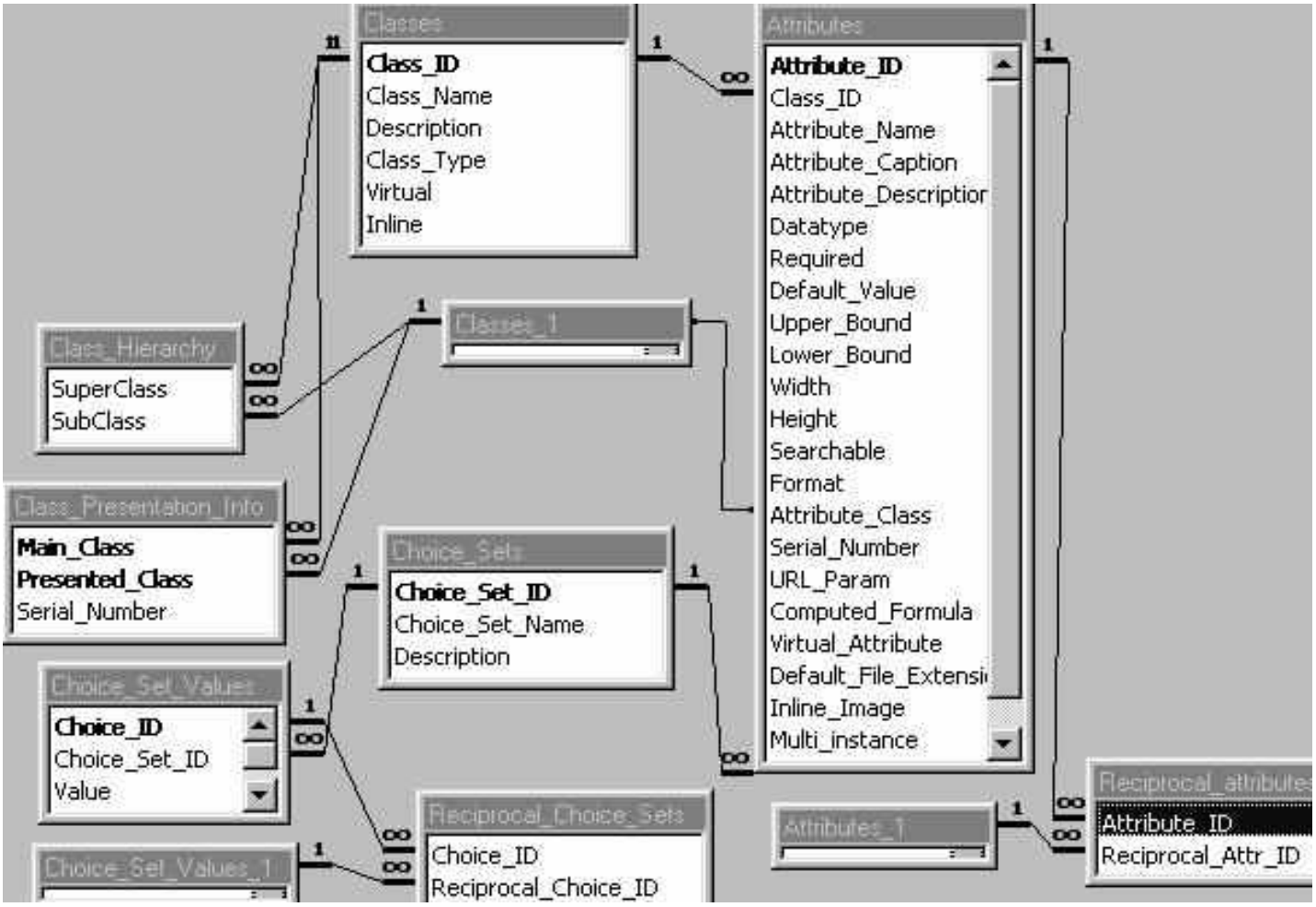
- EAV is primarily a means of simplifying the physical schema of a database.
- Users of the system (as well as analytical programs) expect the data to be conventionally structured. The logical schema of a database (which is domain-specific) reflects the users' perception of the data.
- In an EAV database, the logical schema differs greatly from the physical schema. In a conventional database, the two do not differ appreciably.
- The user interface of a good EAV system conforms to the logical schema as much as possible, creating the illusion of conventional data organization.
- An EAV system must record the logical schema through metadata.
- If sufficiently rich, metadata can also be used actively (i.e., during actual system operation), instead of only describing the system passively.

EAV/CR Overview

EAV/CR overlays an object-oriented framework on top of an EAV physical structure.

- A "class" and "object" in EAV/CR are similar their OOP counterparts.
- EAV/CR allows modeling of inter-class relationships.
- EAV/CR allows classes to contain other classes as members. For this purpose, EAV/CR supports class instances (Object IDs) as values.
- EAV/CR permits inheritance of properties (attributes) between classes.
- Allows representation of non-first-normal-form (NF2) data.

The EAV/CR Schema: Metadata Tables



- The metadata is intended to record a description of all the **Classes** in the database, the **Attributes/Properties** of each class, and the **Class Hierarchy**, if present.
  , if present.
- Certain attributes are based on a set of discrete values: they are presented in the user interface as pull-down lists, list boxes or radio buttons, based on the needs of the application. (Different attributes may be presented differently.)In database design parlance, this set of values forms a **Domain**.
  Rather than create separate tables for each domain, each containing just a few rows, one approach (which we use) is to create just two tables: **Choice_Sets** (defining the domain itself and the purpose it serves) and **Choice_Set_Values** (defining the values for each domain). You record the **Choice Set ID** in the Attributes table, so that the user-interface generation code knows which domain to access for a given attribute. (Some individuals claim this design is bad because you lose the ability to perform integrity checks. Sorry, I disagree. Click on the link Combining Domain Values into fewer tables: Good or Bad? for my take on this issue.)

The EAV/CR Schema: Data Tables

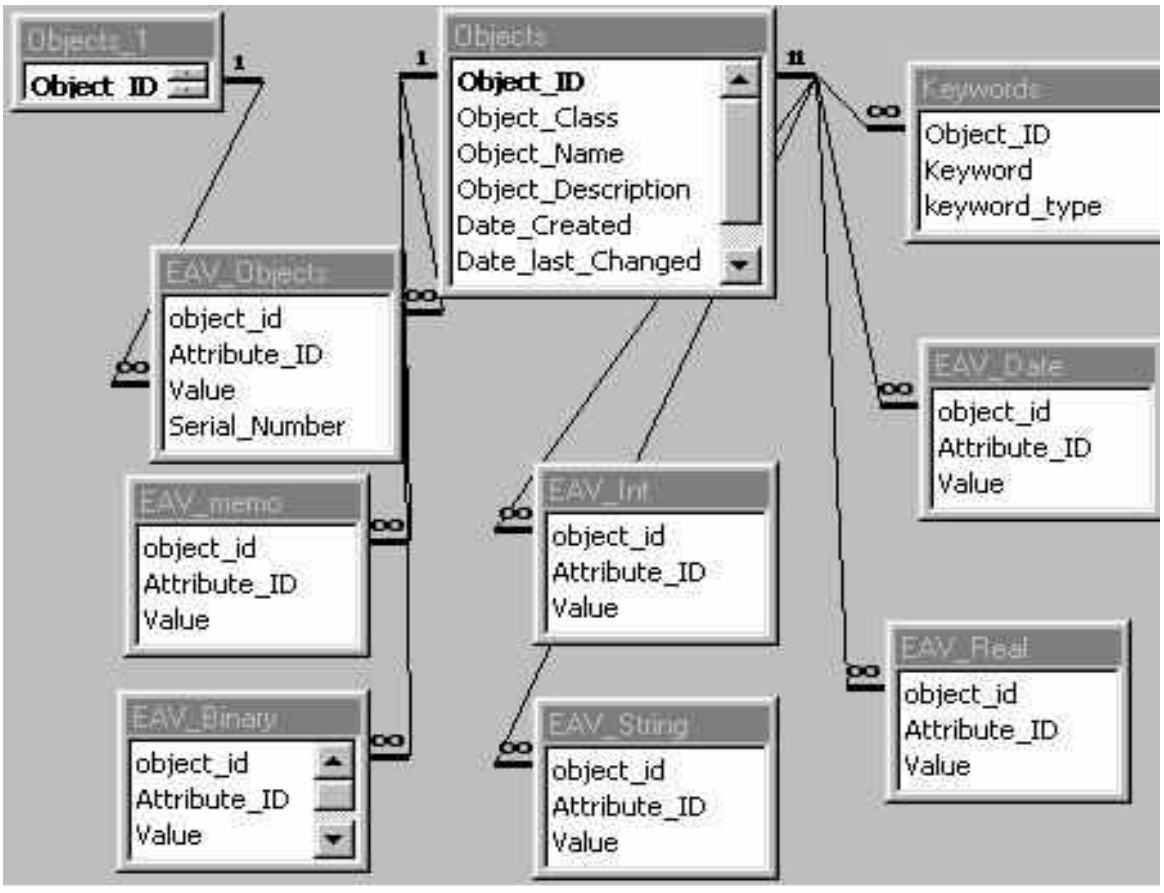
The most critical data table is the **Objects** table, also called the **Object Dictionary**. It contains summary information on every object in the database. The object Class field is a foreign key into the Classes table in

the Metadata schema diagram below. The **Keywords** table allows objects to be searched by key-phrase: its contents are typically composed by combining the text of the Name and Description fields and indexing them. (In database engines such as Microsoft SQL Server, the Keywords table is implemented through a full-text index on these fields, which allows Google-style search.

The **EAV_xxx** tables below implement a strongly typed EAV approach - a value of a given data type goes into a specific table, allowing the possibility of indexing by value and more compact storage without having to coerce everything into the string data type.

Note that storing data in EAV tables makes sense only if either of two conditions are met:
1. The data for a particular class are sparse: that is, most attributes are NULL.
2. You have only a handful (e.g., a few dozen at most) objects belonging to a particular class, and so you do not gain significant efficiency by create a separate table for this class.

If you have hundreds or thousands of non-sparse objects for a given class, you should create a distinct table for this class, preferably with the same name as the Class itself. (Naturally, such tables are not illustrated in the schema below, since they are specific to your application.) The summary information for each object, however, still lies in the Objects table. This way, the Keywords table lets users locate Objects of interest irrespective of what Class they belong to. When you use a Mixed Design approach, where some data is stored in regular tables, and other data in EAV form, the metadata in the Classes table above should have a flag that stated whether Object data for a given class is recorded in conventional columnar form or in EAV form. This way, your application code knows what to do when the user wishes to inspect the details of a particular object.



Managing Relationships

Example: Data on Nigrostriate Neurons

Neurons: Nigrostriatal
Anatomical Origin:Pars Compacta of Substantia Nigra.
Projecting To: Corpus Striatum
Neurochemical/s Released: Dopamine

Receptor/s involved: D2
Electro-physiological Function: Inhibitory
Neurons Projected To (Efferents): Striato-Pallidal neurons, Striato-Striatal neurons
Neurons Providing Input (Afferents): Pars Reticulata of Substantia Nigra, Striato-Nigral fibers.

- This information is multi-axial. Each referenced class of object –receptor, transmitter, neuron, anatomical structure, etc.–represents one axis of the data.
- The nature of axes varies with the nature of the fact, so multiple fact tables are needed with a conventional schema.
- 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.
- Certain axes may have multiple object instances. (e.g., afferents, efferents)For nervous system data, in fact, most axes are likely to be multi-valued. In a normalized relational database design, columns of a table must be atomic and not multi-valued, and so multi-valued data must be factored out into separate tables.
- Within a single axis, entities may be inter-related, through recursive relationships of the parent-child type. (e.g., CNS anatomical structures) A query specified at a coarser level of granularity must also retrieve facts stored at a finer granularity level.

Representing the Nigrostriate Example in EAV/CR

A. 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

B. EAV Data for the Nigrostriate 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>

Relationship Details as Inverted-File Indexes

- Knowledge of the object involved in a relationship does not describe the relationship itself, because the objects can interact in various ways.
- Therefore the description of a fact in a relationship is sometimes best served by narrative text.
- The objects allow the fact to be indexed. (This way, facts related to a particular object can be rapidly retrieved.) The set of objects that index particular facts serve the same role as the Inverted Files used in Text Information Retrieval, with the difference that the objects are strongly typed because they belong to specific classes.

Drawbacks of EAV/CR

- Considerable up-front programming (wheel reinvention) is needed to do the tasks that a conventional architecture would do automatically. However, 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 retrieving data in bulk on numerous objects at a time. (For object-at-a-time retrieval, such as through a Web-based browsing interface, the volume of data is small enough that the difference is not noticeable.)
- Performing complex attribute-centric queries is both significantly less efficient as well as technically more difficult. This needs a query generator. However, most queries on scientific databases are relatively straightforward, and directed toward specific objects of interest.
- For schemas that are relatively static and/or simple (e.g., databases for business applications, such as inventory or accounting), the overhead of EAV design exceeds its advantages. So don't blindly represent all data in EAV form: use conventional tables where the objects are numerous and non-sparse.

But by all means use the Classes, Attributes and Choice Sets/choice-set-values tables : they serve to document your schema and your domains.