A TreeView of Life: Implementing the Linnean Hierarchy

Kevin Wilkinson

Massey University
Wellington, New Zealand

K.J.Wilkinson@massey.ac.nz

Proceedings of the 15th Annual NACCQ, Hamilton New Zealand July, 2002 www.naccq.ac.nz

ABSTRACT

Although the Entity-Relationship approach may seem like an obvious one for implementing the Linnean hierarchy in a museum collections management system, in fact a more successful approach involves holding the data in a single table with a supporting table defining the taxonomic levels. Using a standard viewing component such as the Microsoft Windows TreeView then becomes attractively feasible. A general form of this solution is derived that can be applied to other kinds of hierarchies.

1. THE TASK

1.1 A DEMANDING CASE STUDY

A third-year case study requires students to prototype a museum collections management system. Each student's solution must include one of a number of quite challenging features.

Elsewhere, I discuss the feature of allowing new sub-catalogues to be defined without any reprogramming. Here, the feature dealt with is implementing the Linnean taxonomy of living organisms as part of the collections management system and allowing natural history items to be tagged with references to an element anywhere within that hierarchy.

A note on spelling: although "Linnaean" is closer to the surname of the Swedish naturalist Carolus Linnaeus (1707-78) who did much of the pioneering work on the classification of organisms, "Linnean" is more commonly found in the scientific literature and is preferred here.

1.2 HORIZONTAL STRUCTURE - TAXONOMIC LEVELS

The Linnean hierarchy is typically represented as a two-dimensional structure showing taxonomic level in the horizontal dimension and order of emergence in the vertical (fig. 1).

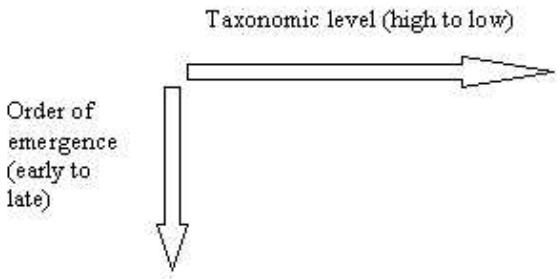

Figure 1: The dimensions of the Linnean hierarchy

One of the first things a student should recognise is how different two implementations of the hierarchy can be. Even which categories should appear in the horizontal dimension are not fixed. A simple scheme might require only the seven main ones, which are

Kingdom
Phylum
Class
Order
Family
Genus
Species

For most purposes, however, this structure is too limited and intermediate ranks have to be introduced using prefixes such as super, sub, and infra. Thus, for example, one might find a hierarchy with the phylum and class levels expanded to

...
Phylum
Subphylum
Superclass
Class
Subclass
Infraclass
...

Clearly, some design decisions are called for to allow this sort of flexibility and to manage the extent to which categories can be changed once implemented.

1.3 VERTICAL STRUCTURE - EMERGENCE

Even if one went to the trouble of sorting all the sub-phyla of vertebrates alphabetically, the result would not be acceptable. What is needed is an ordering that reflects evolutionary history, such as Agnatha, Sharks and rays, Bony fish, Amphibians, Reptiles, Birds, and Mammals.

Order of emergence is a matter of academic opinion and, because this can change, the user must be able to determine the sequence flexibly. This includes being able to promote or demote an item without having to delete and reenter it.

The challenge facing the student is that nothing in the data the user wishes to enter determines the correct vertical order, yet the software must have some way of returning the list to the sequence in which it was left after the previous maintenance session.

1.4 MAINTAINING THE HIERARCHY

As implied by the above discussion, the user must have the ability to insert new items and to change or delete existing ones . The horizontal (taxonomic) level should be variable. It should also be possible to initially position, and later promote or demote, an item in the vertical order of emergence.

While this is easy enough to say, there are plenty of interesting issues involved in carrying out the requirement, such as how to handle jumps in the sequence and how to treat the first and last items in the hierarchy.

The number of taxonomic levels will not be large, so their selection can be managed conveniently by a pick list of the available options. With the order of emergence, however, there is no set of predefined options that can be presented, so the system has to provide a way of recording where an item is when maintenance of the hierarchy is finished.

1.5 USING THE HIERARCHY

Interesting though it be, the Linnean hierarchy is not maintained for its own sake. It is in the system to allow biological field specimens to be classified by being linked to a position in the hierarchy.

Once this is done, the higher taxonomic levels must be displayed. To make life easier for the user, and also to guarantee consistency, this is best done automatically using the hierarchy itself rather than manually.

For example, any item classified as Aardvark should be analysed as shown in fig. 2.

This assumes, of course, that the user wants consistency. If the system has to cater for inconsistent

Figure 2: An analysis of higher taxonomic levels

analyses of the same type of item by different classifiers, each collection record will have to carry its own hierarchy of higher levels.

2. THE ENTITY-RELATIONSHIP APPROACH

The natural way to approach any data analysis task involving a hierarchy is to think in terms of entity-relationships. After all, chains of ONE:MANY relationships are precisely what that approach is designed to handle. However, the E-R route leads to some formidable obstacles, any one of which might force a change of tactic and the combination of which is a gruesome prospect indeed.

2.1 ENTITY COUNT

The first problem is that the entity count may vary from 7 to about 17 depending on the user's requirement for taxonomic levels.

Even if the number can be fixed during the design phase, the idea of a chain of 17 (or even 7) tables linked by foreign keys is not appealing.

Allowing the flexibility of adding new levels dynamically would require the creation of new tables and the breaking and remaking of relationship links from within the software. This would in turn require the automatic respecification of all the queries (sub-schemas) involving those tables and relationships.

2.2 TAXONOMIC GAPS

Even if the entity count problem were overcome, allowing for gaps in the hierarchy would be another major challenge.

Living organisms belong to an irregular hierarchy. Not every item belonging to a particular level has the same pattern of higher abstraction. For example, the class Agnatha would need to link upwards to a record on the sub-phylum table and from there to the phylum level thus:

...

Phylum Cordates

Sub-phylum Vertebrates

Class Agnatha (primitive jawless fish)

...

whereas the class Ciliophora would not need to be linked to any sub-phylum record. Instead, it would link straight up to the phylum level thus:

...

Phylum Protozoans

Class Ciliophora

...

Implementing gaps like this would be awkward using the E-R approach. To preserve referential integrity, one could plug the gaps using dummy entries which would then have to be ignored as the hierarchy was navigated. Alternatively, one could make each record indicate the table of its next higher entry as well as the key of that entry, but such linkages would be costly to maintain and use.

2.3 FLEXIBLE VIEWING

Even if an E-R solution to the implementation of the Linnean hierarchy can be found, using it to display and manipulate the data in the structure would reveal another dimension of the problem.

As we saw in 1.4 and 1.5 above, the structure needs to be expanded, contracted, and scrolled. Records need to be inserted, changed, deleted, promoted, and demoted. Add to this the fact that there would be gaps in the chain of relationships and it seems likely that any solution would be clumsy and involve a lot of resource-hungry navigation of table structures.

As we saw in 2.1 and 2.2, table relationships may have to be changed dynamically to cater for new taxonomic levels and particular records may have a chain of related records through the hierarchy that leaves gaps at some table locations. It is tricky enough under normal circumstances to get inner and outer joins to deliver the subset of the database that satisfies a query without these added complexities.

3. THE TREEVIEW APPROACH

Using Visual Basic as the programming language, an alternative to the E-R approach is available courtesy of a standard component among the Microsoft Windows Common Controls called TreeView. The control is typically used to display lists of folders and their contents nested to any depth. (Bradley and Millspaugh 2001 pp. 49-54 gives a good account of its use.)

3.1 A HIERARCHY OF NODES

To implement the Linnean hierarchy using a treeview, a Nodes collection is built and displayed as part of a TreeView control. Nodes may be expanded and contracted individually, giving the user great flexibility in how the data is presented.

Building a Nodes collection is a matter of adding each item to a node's list or defining it as the child of an existing item and thus creating a new node.

A child is indented to the right of its parent when displayed, thus emphasising its taxonomic level. The vertical ordering of each list gives the order in which items emerged.

The easiest way to build the node hierarchy is to read through a table of items that is already in the correct sequence and to add them one by one using the following logic:

IF	A top level (Kingdom) item THEN
	Add THIS to the list of the root node
ELSEIF THEN	A lower level than the previous item
	Add THIS as a child of the previous item
ELSEIF THEN	A higher level than the previous item
	Find the most recent item of the next higher level that has been encountered so far and
	Add THIS as a child of that item
ELSE	Add THIS to the list of its level, whatever that may be.

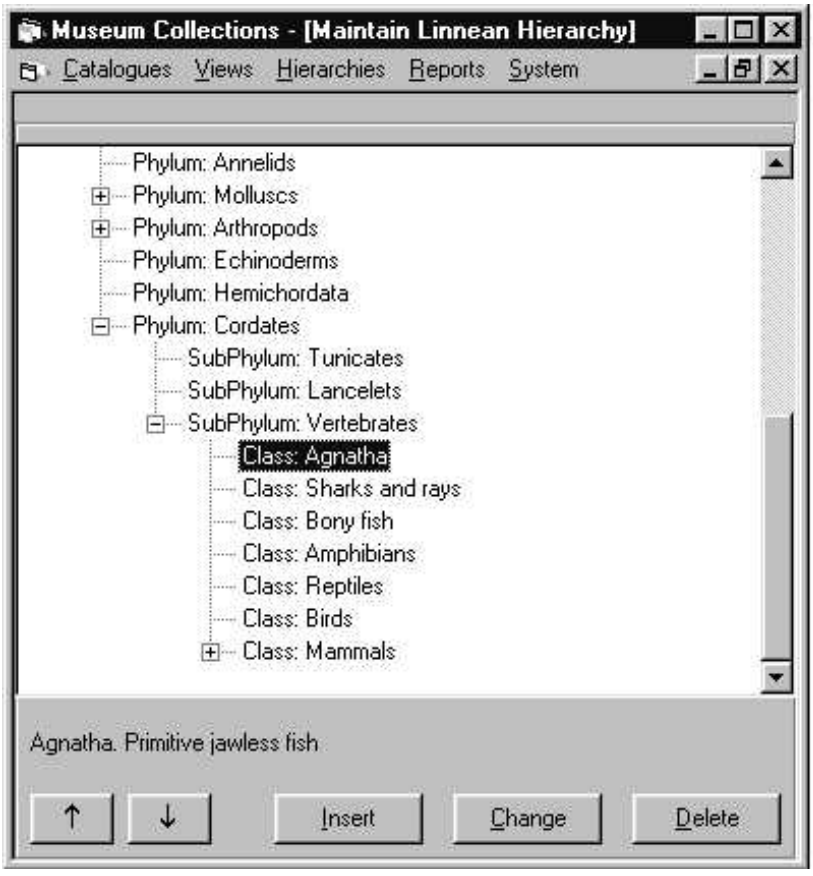

Figure 3: The Linnean hierarchy in a TreeView format

This approach requires each item to know its own taxonomic level, but vertical positions are controlled by the software, which is why the table must be presented in the correct sequence initially. The hierarchy can be displayed, maintained, and used in a format such as that illustrated in fig. 3.

3.2 ADVANTAGES OF THE TREEVIEW APPROACH

The decisive advantage of a format such as that shown in fig. 3 is the elegant, intuitive, and convenient presentation and navigation it affords the hierarchy. Once the Nodes object is populated with data, features such as the ability to scroll, expand, collapse, select from, and skip through the structure are immediately available as part of the functionality of the component.

Because of the simplicity of the data schema, no messy table navigation or query definition is required.

Gaps in the linkages, a major problem for the E-R approach, are catered for. As the logic in 3.1 makes clear, mapping towards a leaf is a matter of keeping track of the identity of the previous item and using the "child of" relation. Mapping towards the root requires knowing the identity of the most recently encountered item of any higher level in case the next higher has not been used so far and an even higher one has to be used as the parent.

Taxonomic levels can be added to the supporting table after implementation, but deletions should be allowed only for levels that are not in use.

The program Linnean.exe is available to demonstrate a model solution to implementing the Linnean hierarchy using the TreeView approach.

4. WIDER IMPLICATIONS

The present solution can be adapted to other hierarchies providing its essential features are preserved, which are:

- Two tables, one for the items in the hierarchy and one for the level definitions.
- Levels that are manageably small in number, such as three for Ocean -- Sea -- Bay. The Linnean taxonomy might have up to twenty levels, but a hierarchy with a hundred would probably be too much trouble to navigate by means of a treeview.
- Items that are controlled by just three numeric fields: a primary key to uniquely identify each item; a foreign key to identify the level of nesting; a computer-controlled number that records the user's preference for the vertical order of the lists within the hierarchy.

Implementing the Linnean hierarchy provides a good opportunity for students to think beyond the obvious and most familiar techniques and to exploit the features of standard components when deciding how best to store and process hierarchical data.

REFERENCES

Bradley, Julia Chase and Millspaugh, Anita C. (2001). Advanced Programming in Visual Basic 6.0, McGraw-Hill.