Paper 2064-2014

Integrating and Using Hierarchical Vocabularies in SAS®

Glenn T. Colby, University of Colorado Boulder

ABSTRACT

Controlled vocabularies define a common set of concepts that retain their meaning across contexts, supporting consistent use of terms to annotate, integrate, retrieve, and interpret information. Controlled vocabularies are often large hierarchical structures that cannot be represented using typical SAS® practices (e.g., SAS format statements and hash objects). This paper compares and contrasts three models for representing hierarchical structures using SAS datasets: adjacency list, path enumeration, and nested set (Celko, 2004; Mackey, 2002). Specific controlled vocabularies discussed include a large university organizational structure and several biological vocabularies (MeSH, NCBI Taxonomy, and GO). This paper presents sample data models and SAS code for populating tables and performing queries.

INTRODUCTION

Controlled vocabularies—standardized sets of terms used in a database—provide common sets of defined concepts that retain their meaning across contexts. When controlled vocabularies (CVs) are integrated into information systems, data analysts can more consistently annotate, retrieve, and interpret related information. CVs further facilitate data integration by enabling the creation of trustworthy relationships among data elements from diverse sources. CVs also provide "important advantages such as normalization of indexing concepts, reduction of noise, identification of indexing terms with a clear semantic meaning, and retrieval based on concepts rather than on words" (Baeza-Yates & Ribeiro-Neto, 1999).

Most CVs embody some form of hierarchical relationships among their concepts (Svenonius, 2003). Along with synonyms, these relationships improve recall and precision during information retrieval. However, along with this improved recall and precision comes added complexity. Integrating hierarchical CVs effectively in SAS presents technical challenges; although many CVs fit into standard hierarchical data structures, each has its own particular structure, content, and ancillary data to accommodate. In some cases, modeling the CV is the easy part—populating the data model and constructing queries to retrieve hierarchical data can present more serious challenges.

There are three general models for representing hierarchical data in relational databases in the literature: a) adjacency list, b) path enumeration, and c) nested set (Celko, 2000; Mackey, 2002). This paper describes each of these models and illustrates how one might retrieve information and populate each model in using a large, hierarchical university organizational structure as an example. Finally, several biological vocabularies (MeSH, NCBI Taxonomy, and GO) are examined to illustrate ways that the three general models might be extended to handle highly complex hierarchical CVs.

A SIMPLE TREE WITH FIXED DEPTH

In a hierarchical organization, each entity in the organization (except the highest-level entity) is a subordinate to exactly one other entity (a *parent*). In fact, the most common structure of large organizations is a hierarchy consisting of a single entity at the top with subsequent levels of entities beneath. While an entity can have only one parent in this structure, it may have many (or no) *children*, and an entity can have many *ancestors* and/or *descendants*. In graph theory, such a structure is referred to as a *rooted tree*—a directed graph in which any two vertices are connected by one and only one simple path.

An in-depth discussion of graph theory is beyond the scope of this paper; there are countless resources available (e.g., Trudeau, 1994). Instead, we now consider the simple hierarchical organization chart in Figure 1 representing a university system. In this figure, we see a single *root node* called System that has three child nodes: Campus A, Campus B, and Campus C. There are five levels in all: system, campus, division, college, and department. We see that a department node does not have any children; we call such nodes a *leaf nodes*. We see that Department C is a descendant of Campus A, and conversely Campus A is an ancestor of Department C.

Let us now pretend that a university is really organized that simply—each department falls under a college, which falls under a division, which falls under a campus, which falls under the university system. Let us suspend our knowledge that in a real university there are departments that do not fall under any college (e.g., facilities management, campus police, etc.), campuses that do not have any divisions, and many other organizational levels that will come into play (e.g., schools, research institutes, programs, etc.).

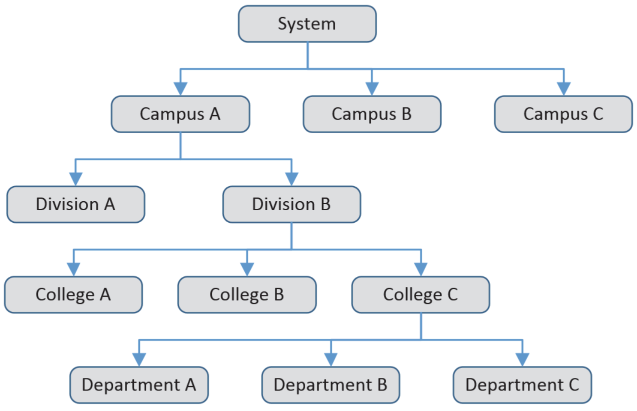


Figure 1. A simple hierarchical organizational chart representing a university system.

Forgetting what we know about real university organizational structures, we can model the hierarchical structure of the university as shown in Figure 2. With the exception of the root node (System), each level is represented as an entity, which would be physically manifested as a SAS dataset (table). Foreign keys (FKs) are defined in this data model in order to enforce referential integrity—a guarantee that each value exists as a primary key value in another table—and to improve performance during join operations. Alternate keys (AKs) are defined in this data model to enforce entity integrity—a guarantee that each value is unique and not empty—and to improve performance during search operations.

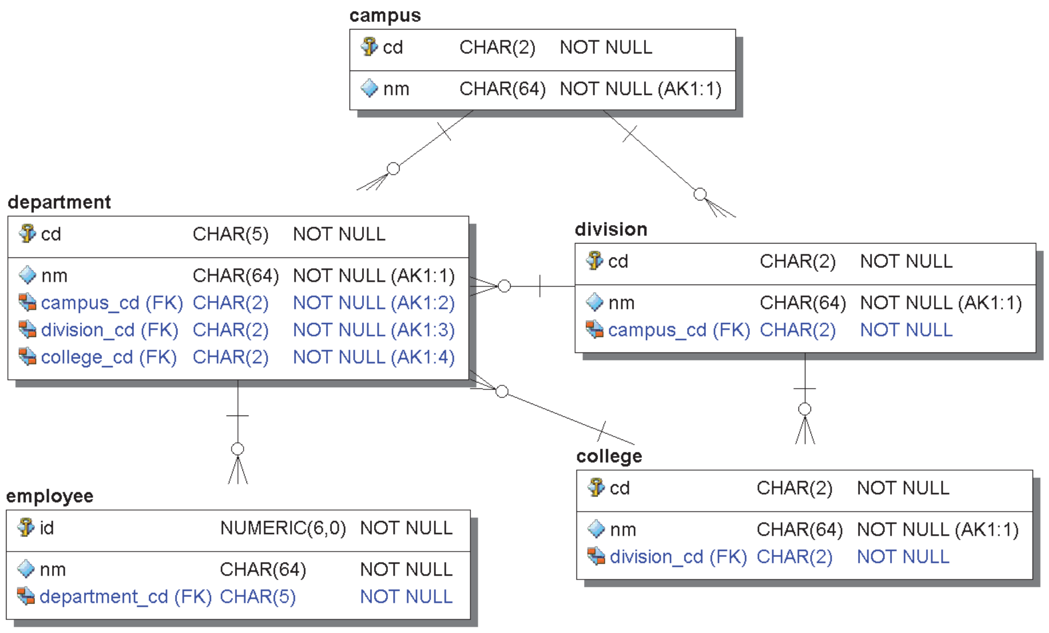

campus
cd CHAR(2) NOT NULL
nm CHAR(64) NOT NULL (AK1:1)
department
cd CHAR(5) NOT NULL
nm CHAR(64) NOT NULL (AK1:1)
campus_cd (FK) CHAR(2) NOT NULL (AK1:2)
division_cd (FK) CHAR(2) NOT NULL (AK1:3)
college_cd (FK) CHAR(2) NOT NULL (AK1:4)
division
cd CHAR(2) NOT NULL
nm CHAR(64) NOT NULL (AK1:1)
campus_cd (FK) CHAR(2) NOT NULL
college
cd CHAR(2) NOT NULL
nm CHAR(64) NOT NULL (AK1:1)
division_cd (FK) CHAR(2) NOT NULL
employee
id NUMERIC(6,0) NOT NULL
nm CHAR(64) NOT NULL
department_cd (FK) CHAR(5) NOT NULL

Figure 2. An entity–relationship (ER) diagram illustrating a possible approach to handling a fixed-depth tree representing a university system. Data definition language (DDL) code is provided in Listing 16.

We now examine sample dataset contents for each of the entities (SAS datasets). Looking at Table 1, we see that there are four campuses in the university system. In Table 2, we see four divisions that are located on the Boulder campus. In Table 3, we see eight colleges that fall under the Division of Academic Affairs on the Boulder campus.

cd	nm
AM	Anschutz Medical Campus
BD	Boulder Campus
CS	Colorado Springs Campus
DN	Denver Campus

Table 1. Sample 'campus' dataset contents.

cd	nm	campus_cd
AA	Division of Academic Affairs	BD
CH	Office of the Chancellor	BD
GC	General Campus	BD
PB	Senior Vice Chancellor Division	BD
…	…	…

Table 2. Sample 'division' dataset contents.

cd	nm	division_cd
AS	College of Arts and Sciences	AA
BU	Leeds School of Business	AA
EB	School of Education	AA
EN	College of Engineering & Applied Science	AA
EV	Program in Environmental Design	AA
JR	School of Journalism & Mass Communication	AA
LW	Law School	AA
MB	College of Music	AA
…	…	…

Table 3. Sample 'college' dataset contents.

Table 4 requires special consideration. Experienced data modelers will recognize that the campus_cd and college_cd columns are not actually necessary since this information is redundant with the contents of the college and division datasets. However, it is common practice to have this information duplicated in the department dataset to simplify queries[1].

cd	Nm	campus_cd	division_cd	college_cd
10032	MCDB-BIO SCIENCES INITIATIVE	BD	AA	AS
10167	CHEMISTRY	BD	AA	AS
10257	BUSINESS-FINANCE	BD	AA	BU
10283	EDUCATION-GRANTS	BD	AA	EB
10287	BUENO CENTER	BD	AA	EB
10318	AERO-AEROSPACE ENGINEERING SCI	BD	AA	EN
10344	COMPUTER SCIENCE	BD	AA	EN
…	…	…	…	…

Table 4. Sample 'department' dataset contents.

The employee dataset shown in Table 5 is provided to illustrate how other information might be related to the hierarchy. For example, we see that Erica J. Stokes is in the Chemistry department, and therefore falls under the College of Arts and Sciences in the Division of Academic Affairs on the Boulder campus.

[1] Duplicating information in multiple datasets is a dangerous path to go down. What would happen if the department dataset indicated that a department is assigned to one campus/division and the college dataset indicated something else? One might argue that this information should be stored in one and only one dataset.

id	nm	department_cd
510004	STOKES, ERICA J	10167
510012	TODD, LEIGH L	10283
510015	ROGERS, SANDRA A	10344
510021	JONES, ROBERT	10283
…	…	…

Table 5. Sample 'employee' dataset contents.

With this data model and the example dataset contents provided, we can now easily determine how many employees are in a given department as shown in Listing 1, or how many employees are in a given division as shown in Listing 2.

```
proc sql;
select count(*)
from employee
where department_cd = '10167'
;
quit;
```

Listing 1. Query that retrieves the number of employees associated with the Chemistry department.

```
proc sql;
select count(*)
from employee
where division_cd = 'AA'
;
quit;
```

Listing 2. Query that retrieves the number of employees associated with the Division of Academic Affairs.

Finally, we can join the datasets to perform more complex queries such as determining how many employees are associated with each campus by name.

```
proc sql;
select
    c.nm as campus_nm,
    count(*) as employee_cnt
from
    campus c,
    department d,
    employee e
where
    d.campus_cd = c.cd and
    e.department_cd = d.cd
group by c.nm
;
quit;
```

Listing 3. Query that retrieves the number of employees associated with each campus.

VARIABLE-DEPTH TREES

In the previous section, we assumed that the CV in question—a hierarchical university organizational structure—had a fixed depth of five levels, and that each department fell under a college, which fell under a division, which fell under a campus, which fell under the university system. If only life were that simple! In reality, a university system organizational structure likely has many more levels than the five described in the previous section, and the leaf nodes (usually departments) may fall at many different levels within the structure. In other words, the structure will likely be highly unbalanced like the example tree shown in Figure 3.

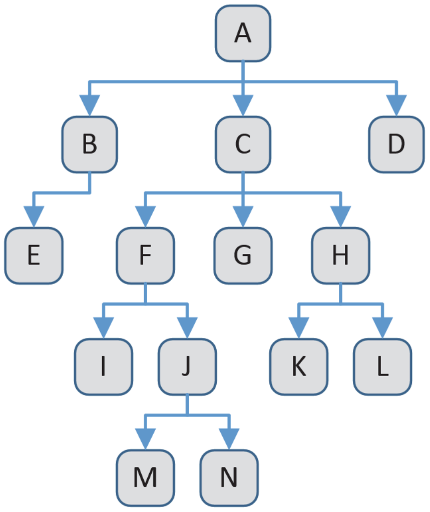


Figure 3. A highly unbalanced, variable-depth tree.

In fact, the primary organizational structure at University of Colorado currently has over 4000 nodes in a 10-level tree, with departments (leaf nodes) falling anywhere between levels 3 and 10. While many of the nodes represent colleges, divisions, and departments as in the previous section, most do not. There are subdivisions, college divisions, institutes, and so on.

A common approach to modeling such a hierarchical data structure in SAS is to use a *wide*, or *unstacked* data model in which each level is presented with a variable in a separate column.[2] Such a model is provided in Figure 4, which also includes a simple employee dataset to illustrate how other datasets might be integrated with the organization_wide dataset.

[2] Such a model can also be transposed to a *narrow*, or *stacked* data model with one column (e.g., lev) listing the level of a node and another column listing the value. The preference depends on what type(s) of analyses are to be conducted.

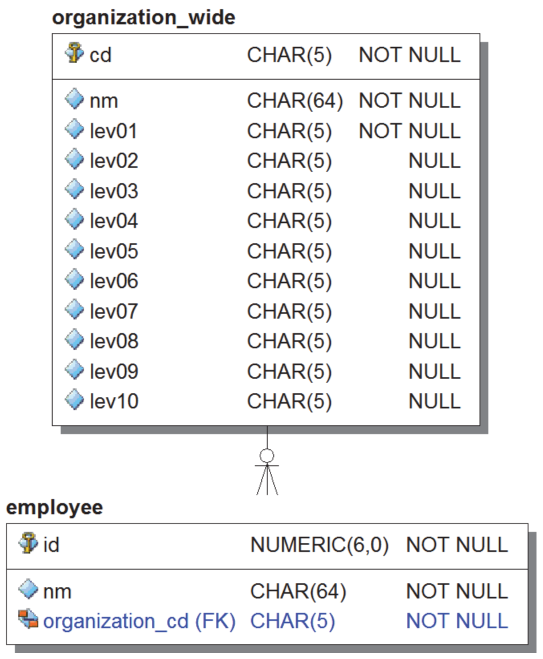


Figure 4. An ER diagram illustrating a typical (wide) approach to handling variable-depth tree data.

In Table 6, we see sample data for the organization_wide dataset corresponding to the model above. We see that the PBA-PLANNING,BUDGET & ANALYSIS department falls at level five, and we can determine the name of each ancestor for the department by looking up the code in the first column.

cd	nm	lev01	lev02	lev03	lev04	lev05	lev06	lev07	lev08	lev09	lev10
ALLCU	ALL CU ORGANIZATIONS	ALLCU									
B0001	BOULDER CAMPUS	ALLCU	B0001								
B0178	B-SENIOR VICE CHANCELLOR	ALLCU	B0001	B0178							
B0122	B-SVC-ASST VC & CONTROLLER	ALLCU	B0001	B0178	B0122						
10004	PBA-PLANNING,BUDGET & ANALYSIS	ALLCU	B0001	B0178	B0122	10004					
…	…	…	…	…	…	…	…	…	…	…	…

Table 6. Sample 'organization_wide' dataset contents.

As in the previous section, it is simple to determine how many employees are associated with a given department, as shown in Listing 4. If we did not know the code for a specific department, we can easily join the organization table in the query in order to add the organization name to the where clause of the query.

```
proc sql;
select count(*)
from employee
where organization_cd = '10004'
;
quit;
```

Listing 4. Query that retrieves the number of employees associated with the PBA-PLANNING,BUDGET & ANALYSIS department.

To determine how many employees are associated with any department falling under a higher-level node in the tree, we can must specify the level of the node by using the corresponding variable in our where clause for the query. In Listing 5, we see the lev03 variable used in the where clause since we know that the node of interest (B-SENIOR VICE CHANCELLOR) falls at level three.

```
proc sql;
select count(*)
from
       employee e,
       organization o
where
       e.organization_cd = o.cd and
       o.lev03 = 'B0178'
;
quit;
```

Listing 5. Query that retrieves the number of employees associated with the B-SENIOR VICE CHANCELLOR organization. This query assumes we know in advance that the organization is level 3.

However, we often do not know the level for a node. In individual cases, we can look up the level of a node in the organization_wide table either manually or programmatically before running such a query, or we can try to search every level of the tree as shown in Listing 6. Admittedly, the query in Listing 6 could be improved in a variety of ways, but the point remains—a simple question of how many employees fall below a particular node requires multiple queries and is therefore inefficient (and complex). In cases where performance matters (e.g., tabulating number of employees at all levels), such an approach usually becomes unfeasible.

```
proc sql;
select count(*)
from
       employee e,
       organization o
where
       e.organization_cd = o.cd and
       (
              o.lev01 = 'B0178' or
              o.lev02 = 'B0178' or
              o.lev03 = 'B0178' or
              o.lev04 = 'B0178' or
              o.lev05 = 'B0178' or
              o.lev06 = 'B0178' or
              o.lev07 = 'B0178' or
              o.lev08 = 'B0178' or
              o.lev09 = 'B0178' or
              o.lev10 = 'B0178'
              )
;
quit;
```

Listing 6. Query that retrieves the number of employees associated with the B-SENIOR VICE CHANCELLOR organization without knowing in advance which level the organization is.

In addition to the complexity of performing queries as described above, this data modeling approach often requires large amounts of storage space which—even if we have it—further degrades performance on queries. Such a data model may be acceptable for our example university organizational structure with ~4000 nodes and 10 levels, but many hierarchical CVs contain many more nodes and levels. For example, the subset of chemicals in the U.S. National Library of Medicine's (NLM) Medical Subject Headings (MeSH) CV consists of over 100,000 nodes in 11 levels. The U.S. National Center for Biotechnology Information's (NCBI) Taxonomy CV consists of over 80,000 nodes in 36 levels. Using a wide data model for such CVs may not be feasible.

In the following sections, we examine three data models for representing hierarchical data in relational databases: a) adjacency list, b) path enumeration, and c) nested set. Which model is appropriate will depend on system requirements and the nature of the CV we are using.

ADJACENCY LIST

Hierachical CVs are often diagrammed as shown in Figure 1 or Figure 3. In these figures, the rectangular boxes represent the nodes of the trees. The boxes are linked by lines in the diagrams, with arrowheads indicating the direction of the links. In graph theory, these links are called *edges*. One way to model variable-depth trees in SAS is to include the edges themselves as part of the data model. In other words, we can add a column to represent the edges of the trees.

In Figure 5, we see such a model. Rather than have a column for each potential level of the tree, we have a column (parent_organization_cd) that represents the link between a node and its parent. Using this model, we can represent a large hierarchical CV using much less space than the wide data model examined in the previous section without any information loss.

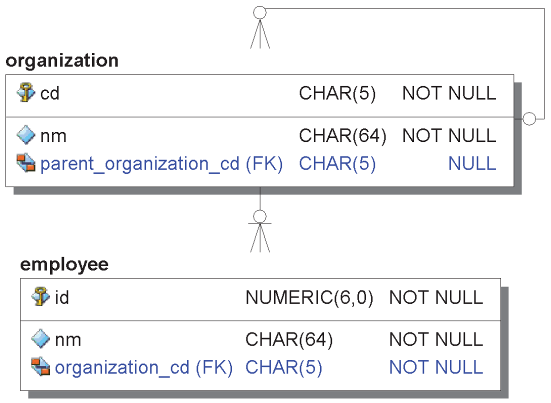


Figure 5. ER diagram illustrating a single-table adjacency list approach to handling variable-depth tree data.[3] DDL code is provided in Listing 17.

In the sample dataset shown in Table 7, we see that B-SVC-ASST VC & CONTROLLER is the parent organization of the PBA-PLANNING,BUDGET & ANALYSIS, the B-SENIOR VICE CHANCELLOR organization is the parent organization of the B-SVC-ASST VC & CONTROLLER, and so on. Any information represented in the wide data model (see **Error! Reference source not found.**) is also represented in this dataset while using about 1/10th the space.

cd	nm	parent_organization_cd
ALLCU	ALL CU ORGANIZATIONS	
B0001	BOULDER CAMPUS	ALLCU
B0178	B-SENIOR VICE CHANCELLOR	B0001
B0122	B-SVC-ASST VC & CONTROLLER	B0178
10004	PBA-PLANNING,BUDGET & ANALYSIS	B0122
…	…	…

Table 7. Sample ‘organization’ dataset contents.

Queries in the Adjacency List Data Model

As in previous examples, determining the number of employees associated with a specified department is trivial, as shown in Listing 7

[3] This ER diagram depicts a foreign key (parent_organization_cd) that references a primary key (cd) in the same dataset (organization). Such a foreign key ensures that a parent_organization_cd value cannot be assigned unless it first exists as a primary key. Unfortunately, SAS will not support a foreign key that references a primary key in the same dataset. While this *logical* data model shows such a foreign key, the *physical* data model would require an additional dataset in SAS (e.g., a separate ‘organization_pk’ dataset that itself includes a foreign key to the organization dataset).

```
proc sql;
select count(*)
from employee
where organization_cd = '10004'
;
quit;
```

Listing 7. Query that retrieves the number of employees associated with the PBA-PLANNING,BUDGET & ANALYSIS department.

Similarly, determining how many employees are associated with the child organizations for a specified organizational unit is also trivial, as shown in Listing 8.

```

proc sql;
select count(*)
from
       employee e,
       organization o
where
       e.organization_cd = o.cd and
       o.parent_organization_cd = 'B0122'
;
quit;
```

Listing 8. Query that retrieves the number of employees associated with the B-SVC-ASST VC & CONTROLLER organization.

However, determining how many employees are associated with any descendant of a specified organizational unit is not an easy task in this single-table adjacency list data model, even if we know in advance the level of the unit. Performing such a query would require iteratively identifying the child organizations, the children's child organizations, and so on until all potential levels have been checked. If we need to perform such queries often, then a single-table adjacency list data model is probably not appropriate. If we do not have such a requirement, then this data model may meet our needs.

Populating the Adjacency List Data Model

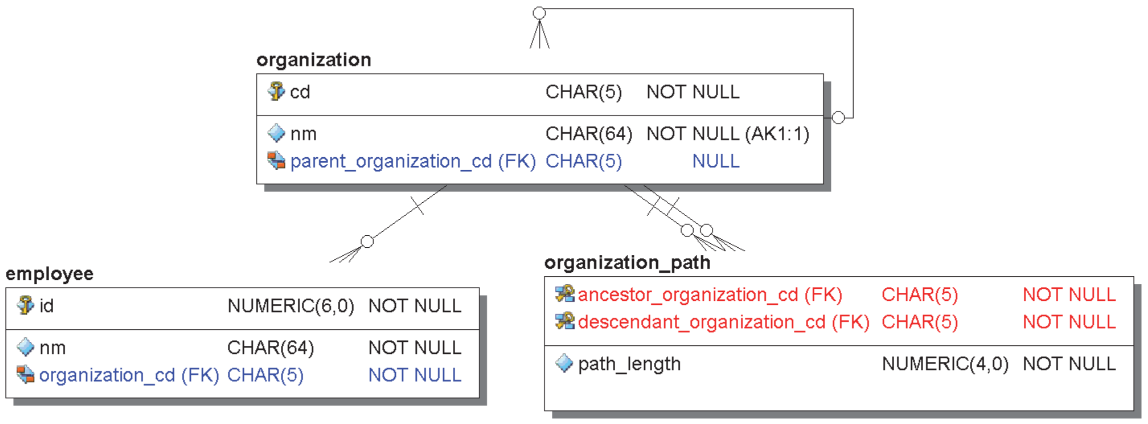
Populating the organization dataset can also present technical challenges. If a data source provides a CV using a wide data model such as the organization_wide dataset in **Error! Reference source not found.**, we need to perform some sort of iteration to load the data into the dataset using the adjacency list data model. Listing 9 provides an example of SAS code used to populate such a dataset.

```sas
%LET NUM_LEVELS = 10;

%MACRO load_organization_level(level);
    %if &level. eq 1 %then
        %let parent_level = 1;
    %else
        %let parent_level = %eval(&level.  1);
    %let z_level = %sysfunc(putn(&level., Z2.));
    %let z_parent_level = %sysfunc(putn(&parent_level., Z2.));

    proc sql;
    insert into organization
    select distinct
        ow.lev&z_level.,
        ow.nm,
        case
            when &level. = 1 then
                ''
            else
                ow.lev&z_parent_level.
        end
    from organization_wide ow
    where ow.cd = lev&z_level.
    ;
    quit;
%MEND load_organization_level;

%MACRO load_organization;
    %do i = 1 %to &NUM_LEVELS.;
        %load_organization_level(&i)
    %end;
%MEND load_organization;

%load_organization;
```

Listing 9. Example SAS code to populate the 'organization' dataset.

PATH ENUMERATION

To address the shortcomings of the adjacency list data model, we can augment the adjacency list data model described in the previous section with an additional dataset that stores all of the paths between all of the nodes in the tree as depicted in Figure 6. Such a dataset is called a *path enumeration*, or *transitive closure*, of the tree. The rows in the dataset represent the start and end nodes of each path in the tree. Including a variable representing each path length is convenient for sorting query results or limiting queries to ranges of levels.



organization
cd CHAR(5) NOT NULL
nm CHAR(64) NOT NULL (AK1:1)
parent_organization_cd (FK) CHAR(5) NULL
employee
id NUMERIC(6,0) NOT NULL
nm CHAR(64) NOT NULL
organization_cd (FK) CHAR(5) NOT NULL
organization_path
ancestor_organization_cd (FK) CHAR(5) NOT NULL
descendant_organization_cd (FK) CHAR(5) NOT NULL
path_length NUMERIC(4,0) NOT NULL

Figure 6. ER diagram illustrating the path enumeration approach to handling variable-depth tree data. DDL code is provided in Listing 18.

To illustrate how this works, the sample dataset in Table 8 shows that there path from the root node, ALLCU, to all of the descendants of ALLCU. Similarly, but not shown, the dataset would include a path from each node to each of its descendants.

Obviously such a dataset might grow extremely large when applied to a large and deep CVs such as NLM's MeSH CV or NCBI's Taxonomy CV. In fact, the path enumeration dataset for University of Colorado's organizational structure (~4000 nodes and 10 levels) has over 27,000 rows—an almost 700% increase in rows. However, since the dataset consists primarily of foreign keys, the size of the path enumeration dataset is less than twice that of the adjacency list dataset itself. In fact, if the primary keys were numeric, it is possible that a path enumeration dataset would require *less* space than the associated adjacency list dataset.

ancestor_organization_cd	descendant_organization_cd	path_length
ALLCU	ALLCU	0
ALLCU	B0001	1
ALLCU	C0001	1
ALLCU	S0001	1
ALLCU	U0001	1
ALLCU	40000	2
ALLCU	50000	2
ALLCU	60000	2
ALLCU	B0002	2
ALLCU	B0010	2
…	…	…

Table 8. Sample 'organization_path' dataset contents.

Queries in the Path Enumeration Data Model

Once we have augmented our adjacency list dataset with a path enumeration dataset, constructing a query to determine how many employees are associated with a specified organizational unit—even when we do not know the level in advance—is both simple and efficient. In Listing 10, we determine the number of employees associated with the B-SENIOR VICE CHANCELLOR organization or any of its descendants. A similar query could be constructed to retrieve information related to the ancestors of a specified organizational unit.

```
proc sql;
select count(*)
from
       employee e,
       organization_path op,
       organization o
where
       e.organization_cd = o.cd and
       o.cd = op.descendant_organization_cd and
       op.ancestor_organization_cd = 'B0178'
;
quit;
```

Listing 10. Query that retrieves the number of employees associated with the B-SENIOR VICE CHANCELLOR organization.

Finally, we note that the query in Listing 10 combines records from all three tables in the path enumeration data model shown in Figure 6 even though we use no information in the organization dataset itself. Therefore, it is possible to perform the query without involving the organization dataset at all, as shown in Listing 11.

```
proc sql;
select count(*)
from
       employee e,
       organization_path op
where
       e.organization_cd = op.descendant_organization_cd and
       op.ancestor_organization_cd = 'B0178'
;
quit;
```

Listing 11. Simpler, more efficient version of query that retrieves the number of employees associated with the B-SENIOR VICE CHANCELLOR organization. This query works despite excluding the 'organization' dataset from join operations.

Populating the Path Enumeration Data Model

As was the case with the adjacency list data model in the previous section, populating the organization_path dataset can present technical challenges[4]. If a data source provides a CV using a wide data model such as the organization_wide dataset in **Error! Reference source not found.**, we need to perform some sort of iteration to load the data into the path enumeration dataset. Listing 12 provides an example of SAS code used to populate such a dataset.

[4] If the hierarchical CV is fixed depth—that is, we know that it will always have the same number of levels—then populating the path enumeration dataset can be straightforward.

```
%LET NUM_LEVELS = 10;

%MACRO load_paths(ancestor_level, descendant_level);
    %let path_length = %eval(&descendant_level.  &ancestor_level.);
    %let z_ancestor_level = %sysfunc(putn(&ancestor_level., Z2.));
    %let z_descendant_level = %sysfunc(putn(&descendant_level., Z2.));

    proc sql;
    insert into organization_path
    select distinct
        lev&z_ancestor_level.,
        lev&z_descendant_level.,
        &path_length.
    from organization_wide
    where cd = lev&z_descendant_level.
    ;
    quit;
%MEND load_paths;

%MACRO load_organization_path;
    %do ancestor_level = 1 %to &NUM_LEVELS.;
        %do descendant_level = &ancestor_level. %to &NUM_LEVELS.;
            %load_paths(&ancestor_level., &descendant_level.)
        %end;
    %end;
%MEND load_organization_path;

%load_organization_path;
```

Listing 12. Example SAS code to populate the 'organization_path' dataset.

NESTED SET MODEL

The nested set data model (Celko, 2000) is a lesser-known approach to modeling hierarchical data, but it has a major advantage in that it takes advantage of the set-oriented nature of the SQL language. In this approach, rather than conceptualize a hierarchy as a tree with nodes and edges we conceptualize a hierarchy as nested sets. Figure 7 shows the university system explored earlier (see Figure 1) as nested sets.

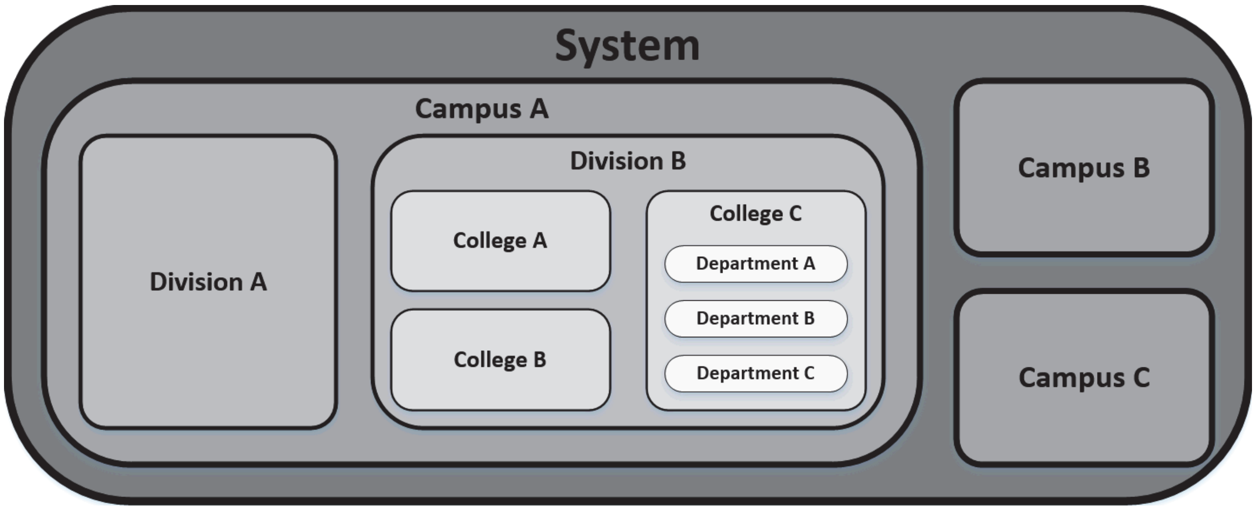


Figure 7. A university system represented as nested sets.

Whereas the adjacency list data model represented nodes of a tree and the path enumeration data model represented the edges of a tree, the nested set model simply models sets and all the subsets contained within those sets. Each set is represented by a pair of numbers representing the boundaries of the subset—the ‘left’ number is the lower bound and the ‘right’ number is the upper bound. This is visualized in Figure 8.

In Figure 8, we can tell whether a set is a subset of another set by comparing the subset numbers. For example, we can tell that College C is a subset of Campus A because College C’s subset numbers (10 and 17) lie within the range of Campus A’s subset numbers (2 and 19).

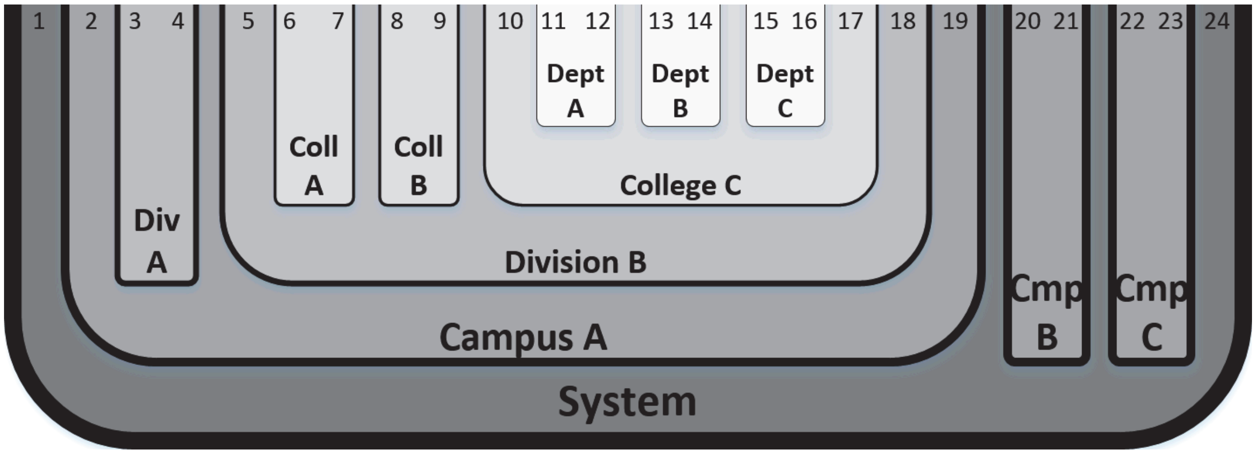


Figure 8. Another version of a university system represented as nested sets, flattened onto a number line.

Figure 9 provides an ER diagram of such a data model. Note that this model is identical to the single-table adjacency list data model discussed earlier (see Figure 5), and unlike the path enumeration data model (see Figure 6) it requires no additional datasets to perform hierarchical queries.

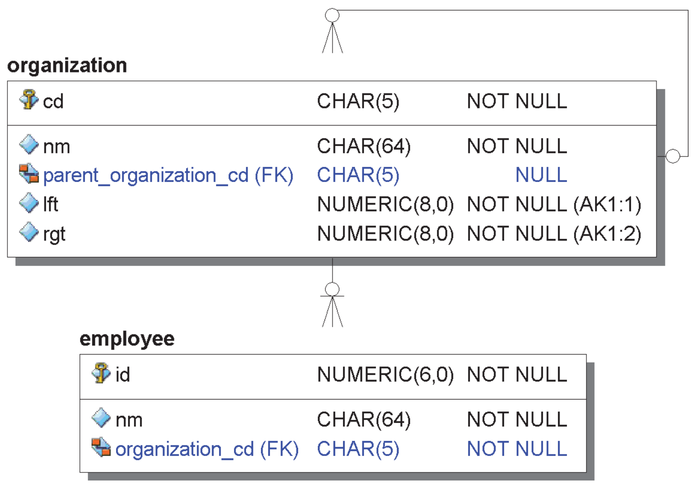

organization
cd CHAR(5) NOT NULL
nm CHAR(64) NOT NULL
parent_organization_cd (FK) CHAR(5) NULL
lft NUMERIC(8,0) NOT NULL (AK1:1)
rgt NUMERIC(8,0) NOT NULL (AK1:2)
employee
id NUMERIC(6,0) NOT NULL
nm CHAR(64) NOT NULL
organization_cd (FK) CHAR(5) NOT NULL

Figure 9. ER diagram illustrating the nested set approach to handling variable-depth tree data. DDL code is provided in Listing 19.

To further illustrate how the nested set model works, we now consider the sample dataset in Table 9. In this dataset, we again see that College C is a subset of Campus A because College C's subset numbers (10 and 17) lie within the range of Campus A's subset numbers (2 and 19).

nm	parent_organization_nm	lft	rgt
System		1	24
Campus A	System	2	19
Division A	Campus A	3	4
Division B	Campus A	5	18
College A	Division B	6	7
College B	Division B	8	9
College C	Division B	10	17
Department A	College C	11	12
Department B	College C	13	14
Department C	College C	15	16
Campus B	System	20	21
Campus C	System	22	23

Table 9. Sample 'organization' dataset contents for a nested set model.

Queries in the Nested Set Data Model

Here lies the strength of the nested set data model. Having the left and right subset numbers available allows us to use the BETWEEN operator in our queries, which is usually more efficient than performing join operations on multiple datasets. For example, to determine how many employees are associated with departments under College A in Table 9, we simply add a where clause to look for subset numbers between 2 and 19 (see Listing 13).

```
proc sql;
select count(*)
from
    employee e,
    organization o
where
    e.organization_cd = o.cd and
    o.lft between 2 and 19
;
quit;
```

Listing 13. Query that retrieves the number of employees associated with the B-SENIOR VICE CHANCELLOR organization using a nested set model.

Of course, in practice we generally would not know what left and right subset numbers are assigned to a specified organizational unit without looking them up first. In this case, we can simply retrieve the subset numbers as part of our query, as shown in Listing 14.

```
proc sql;
select count(*)
from
    employee e,
    organization o
where
    e.organization_cd = o.cd and
    o.lft between
        (
            select lft
            from organization
            where cd = 'B0178'
            ) and
        (
            select rgt
            from organization
            where cd = 'B0178'
            )
;
quit;
```

Listing 14. Query that retrieves the number of employees associated with the B-SENIOR VICE CHANCELLOR organization using a nested set model.

Populating the Nested Set Data Model

The nested set data model is not commonly used to represent hierarchical CVs in databases, although there are often clear advantages to doing so. The reasons are both historical and technical—the adjacency list data model was the first model developed for representing hierarchical data in a relational database (Celco, 2000), and there are technical challenges for managing the data in the context of SAS and other relational databases that represent data to users as “tables (and nothing but tables)” (Date, 1990).

Populating a nested set data model in SAS presents a serious technical challenge. The data model is populated using a recursive, depth-first search (DFS) algorithm. While the adjacency list and path enumeration data models can be populated with fairly straightforward SAS macros (see Listing 9, Listing 12), the SAS Macro Language programming does not support recursion—invoking procedures that in turn call themselves.[5]

Rather than provide a lengthy, complex example of how we might populate a nested set data model in SAS, Listing 15 provides pseudocode for populating such a model. Given the challenges of implementing such an algorithm in SAS, I would argue that it would be more appropriated to implement the algorithm in another language that supports recursion (e.g., Python) and them import the data into a SAS dataset.

[5] Numerous authors have proposed or provided solutions for implementing recursive algorithms (e.g., DFS) in the SAS Macro Language. While often impressive, these approaches are usually highly complex, and examining such a problem would be beyond the scope of this paper.

```
POPULATE-SUBSET_NUMBERS(T)
    r ← root[T]
    DFS-VISIT(r)

DFS-VISIT(u, n ← 1)
    left_subset_no ← n
    for each v ∈ children[u] do
        n ← DFS-VISIT(v, n + 1)
    right_subset_no ← n + 1
    SET-SUBSET-NUMBERS(u, left_subset_no, right_subset_no)
    return right_subset_no
```

Listing 15. Pseudocode for a recursive, depth-first search (DFS) algorithm used to populate a nested set data model.

BEYOND TREES

The examples we have examined are relatively straightforward compared to many of the complex hierarchical CVs we may need to integrate into our information systems. Some large CVs, such as NCBI's Taxonomy CV, consist of straightforward mappings between terms and nodes in a tree (or sets and subsets) that can be represented using any of the data models discussed so far in this paper. Other CVs, such as NCBI's MeSH CV and the Gene Ontology Consortium's Gene Ontology (GO) CV, cannot be modeled effectively with any of these data models *per se* while still providing powerful and efficient hierarchical query capabilities. In fact, many CVs are actually not trees, as we shall see in this section.

NATIONAL LIBRARY OF MEDICINE'S (NLM) MEDICAL SUBJECT HEADINGS (MESH)

Anyone who has recently conducted research in biology is familiar with MeSH, a comprehensive CV used to index and search articles and books in the life sciences. MeSH is used extensively in the MEDLINE and PubMed databases, and has been in use since 1963 (http://www.ncbi.nlm.nih.gov/mesh). MeSH consists of over 25,000 *subject headings*, or *descriptors*, that are arranged in a hierarchical tree. Integrating MeSH into an information system can provide powerful mechanisms to annotate, integrate, retrieve, and interpret information.

Since MeSH is organized as a tree, we might consider using one of the three data models discussed earlier in this paper. However, there is a catch. In MeSH, a subject heading (term) can be assigned to multiple nodes in the tree. For example, in Figure 10 we see the subject heading Arsenicals appears at two nodes: D01.075 and D02.691.088. Similarly, the term arsenic acid[6] appears at two nodes. By allowing a term to be assigned to multiple nodes, users can perform powerful queries across multiple sections of the MeSH tree.

Of course, along with this added power comes added complexity. Since MeSH treats terms and nodes separately, we must therefore model the concepts separately, as shown in Figure 11. By combining elements of all three data models discussed in this paper, we can store the entire subset of chemicals in MeSH in a manner that allows powerful and efficient hierarchical queries.

[6] Technically, arsenic acid is listed in MeSH as a Supplementary Concept rather than as a subject heading, but since arsenic acid is assigned to the subject heading Arsenates, it is shown as falling under both nodes for Arsenates.

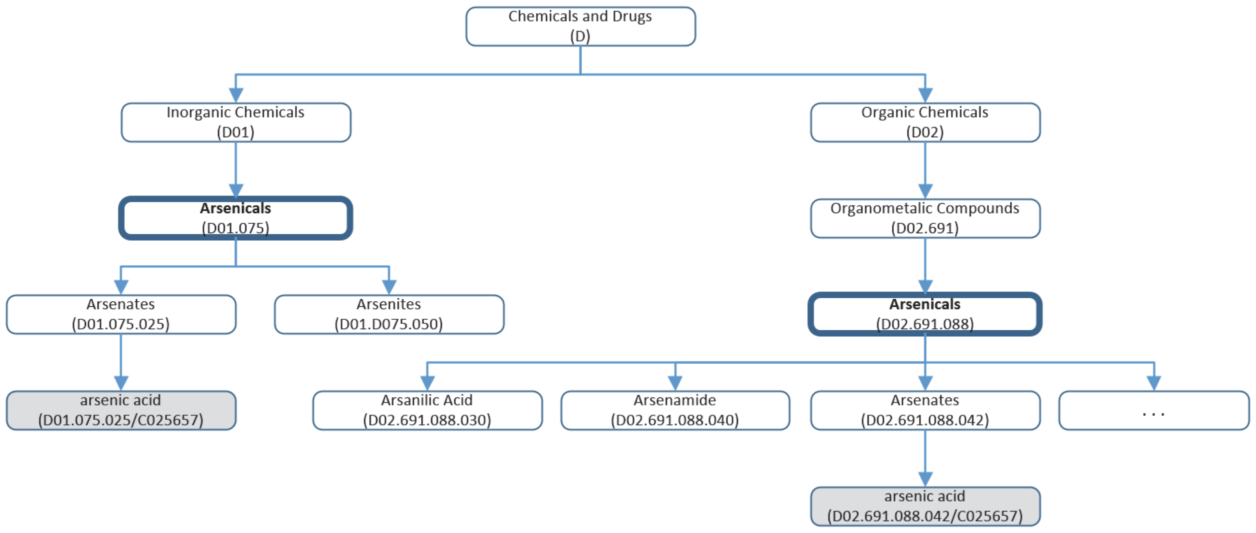


Figure 10. A simplified example of the tree structure for chemicals in MeSH. The chemical catergory 'Arsenicals' appears at two different nodes in the tree. A MeSH tree number is shown in parentheses for each node.

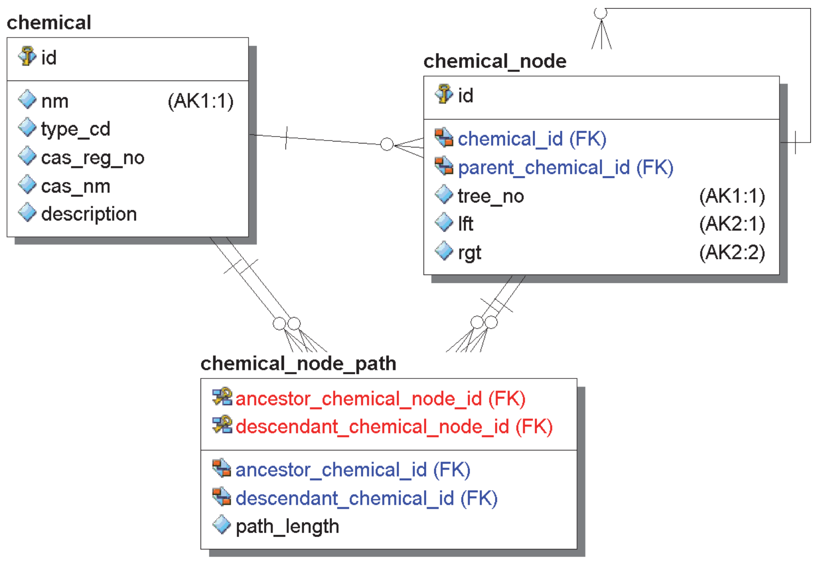


Figure 11. A high-level ER diagram of the MeSH chemical vocabulary.

GENE ONTOLOGY (GO)

As a final example, we now examine the Gene Ontology Consortium's *Gene Ontology* (GO) CV. Like other CVs we have examined, GO is organized hierarchically with many levels and thousands of terms. GO describes gene products in terms of their associated biological processes, cellular components, and molecular functions (Gene Ontology Consortium, 2000), and is used by biologists around the world to annotate, integrate, retrieve, and interpret information.

Examining the example in Figure 12, two important features need to be highlighted. First, we see that the edges of the tree (the lines between the rectangular boxes) include text indicating the nature of the relationship between each parent and child node. For example, we see that biological process is 'part of' the Gene Ontology. We also see that response to stimulus 'is a' physiological process.

Second, a more significant feature of the GO CV is that there can be multiple paths to a descendant node. While immune response 'is a' defense response (and therefore a response to stimulus), we see that it is also defined as an organismal physiological process. In terms of graph theory, we would say that this structure cannot be referred to as a rooted tree, as the previous examples were, since a rooted tree is commonly defined a directed graph in which any two vertices are connected by one and only one simple path. Instead, GO is a more general structure—a *directed acyclic graph* (DAG).

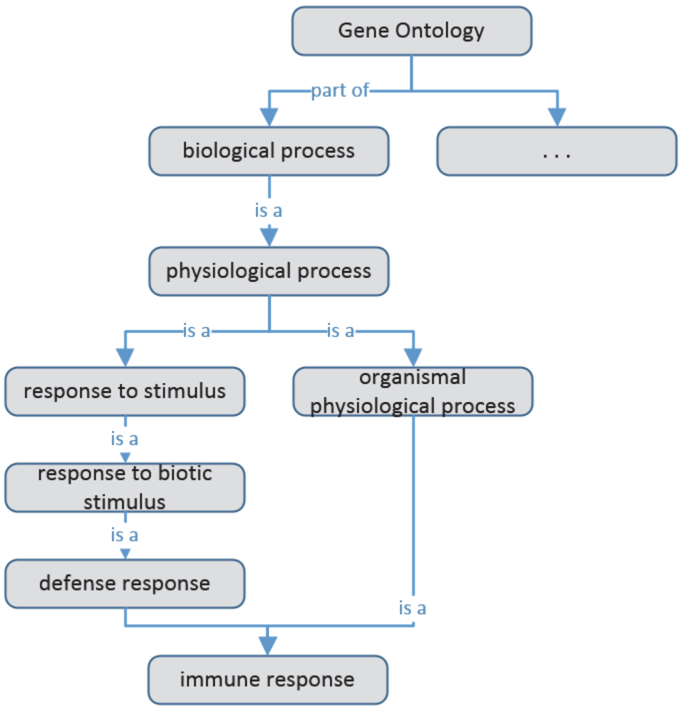


Figure 12. A simplified example of the GO directed acyclic graph. Note that there are two paths to the *immune response* node.

Like the MeSH CV discussed in the previous section, modeling a DAG such as the GO CV in SAS requires a hybrid data model. Without getting too bogged down in the details of the GO CV, suffice it to say that modeling the GO CV would likely require explicit representation of not only the nodes, edges, and paths of the CV, but also edge types (e.g., 'is a'), labels (to handle synonyms and so forth), and other various and sundry features of the CV. Fundamentally, though, the underlying structure can be represented with the three data models presented in this paper. For example, researchers at the Mount Desert Island Biological Laboratory have used a path enumeration data model in their implementation of the Comparative Toxicogenomics Database ([CTD] Davis *et al.*, 2013) to incorporate the GO CV into their database[7].

CONCLUSION

This paper presented three known data models that can be used to represent large, complex hierarchical CVs in SAS. Which model(s) might be best for a given CV depend on many factors, not the least of which is the cost and effort of implementing such the data model. In practice, developers may focus efforts on one particular model by developing a robust library of code to handle data for that model, and users may appreciate having a consistent data modeling approach among various CVs.

The data models discussed in this paper provide a foundation for modeling hierarchical CVs, but this is only the tip of the iceberg for many systems. For example, a major assumption throughout this paper is that we are integrating *existing* CVs that are maintained by some other data sources. If we had to create and/or maintain CVs ourselves, the challenges might become enormous. Consider what it would take to insert a new node into a CV with a nested set data model. Or consider what it would take to delete a node and its descendants from a CV with a path enumeration data model.

Analysts can better annotate, integrate, retrieve, and interpret information by integrating hierarchical CVs into information systems. However, integrating hierarchical CVs in SAS is challenging work. The three data models discussed in this paper—adjacency list, path enumeration, and nested set—provide a foundation for SAS developers and analysts to face this challenge.

[7] CTD provides excellent technical documentation, including data models for their internal and public databases (see http://ctdbase.org/about/documentation/).

REFERENCES

Baeza-Yates, R., & Ribeiro-Neto, B. (1999). *Modern Information Retrieval*. New York: ACM Press.

Celko, J. (2000). *Joe Celko's SQL for Smarties: Advanced SQL Programming*, 2nd Ed. San Francisco: Morgan Kaufmann.

Celko, J. (2004). *Joe Celko's Trees and Hierarchies in SQL for Smarties*. San Francisco: Morgan Kaufmann.

Davis AP, Murphy CG, Johnson R, Lay JM, Lennon-Hopkins K, Saraceni-Richards C, Sciaky D, King BL, Rosenstein MC, Wiegers TC, Mattingly CJ. The Comparative Toxicogenomics Database: update 2013. *Nucleic Acids Res*. 2013 Jan 1;41(D1):D1104-14.

Date, C. J. (1986). *An Introduction to Database Management Systems*, Vol. 1, 4th Ed.. Reading: Addison-Wesley.

Gene Ontology Consortium (2000). Gene Ontology: Tool for the unification of biology. *Nature Genet*., **25**, 25-29.

Mackey, A. (2000). *Relational Modeling of Biological Data: Trees and Graphs*. Available online at http://www.oreillynet.com/pub/a/network/2002/11/27/bioconf.html

Svenonius, E. (2003). *Design of Controlled Vocabularies.* Encyclopedia of Library and Information Science, New York, Marcel Dekker, p82-109. DOI:10.1081/E-ELIS 120009098

Trudeau, R. J. (1994). *Introduction to Graph Theory*. Mineola, NY: Dover Publications.

ACKNOWLEDGMENTS

Much of the content in this paper was derived from previous work with Michael C. Rosenstein at Mount Desert Island Biological Laboratory. Michael taught me everything I know about data modeling (but not everything *he* knows). ☺

RECOMMENDED READING

- *Joe Celko's SQL for Smarties: Advanced SQL Programming*
- *Joe Celko's Trees and Hierarchies in SQL for Smarties*

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Name: Glenn T. Colby
Organization: University of Colorado Boulder
Address: 15 UCB; Regent Administrative Center; 2055 Regent Dr., 1B29
City, State ZIP: Boulder, CO 80309-0015
Work Phone: 303-492-3769
Fax: 303-492-0996
Email: glenn.colby@colorado.edu
Web: http://www.colorado.edu/pba/

APPENDIX: DDL CODE FOR SELECTED DATA MODELS

```
proc sql;
CREATE TABLE campus (
    cd    CHAR(2)     NOT NULL,
    nm    CHAR(64)    NOT NULL,
    CONSTRAINT pk PRIMARY KEY (cd)
     );
CREATE UNIQUE INDEX nm ON campus(nm);

CREATE TABLE division (
    cd           CHAR(2)     NOT NULL,
    nm           CHAR(64)    NOT NULL,
    campus_cd    CHAR(2)     NOT NULL,
    CONSTRAINT pk PRIMARY KEY (cd),
    CONSTRAINT campus_fk FOREIGN KEY (campus_cd)
         REFERENCES campus
     );
CREATE UNIQUE INDEX nm ON division(nm);

CREATE TABLE college (
    cd             CHAR(2)     NOT NULL,
    nm             CHAR(64)    NOT NULL,
    division_cd    CHAR(2)     NOT NULL,
    CONSTRAINT pk PRIMARY KEY (cd),
    CONSTRAINT division_fk FOREIGN KEY (division_cd)
         REFERENCES division
     );
CREATE UNIQUE INDEX nm ON college(nm);

CREATE TABLE department (
    cd             CHAR(5)     NOT NULL,
    nm             CHAR(64)    NOT NULL,
    campus_cd      CHAR(2)     NOT NULL,
    division_cd    CHAR(2)     NOT NULL,
    college_cd     CHAR(2)     NOT NULL,
    CONSTRAINT pk PRIMARY KEY (cd),
    CONSTRAINT campus_fk FOREIGN KEY (campus_cd)
         REFERENCES campus,
    CONSTRAINT division_fk FOREIGN KEY (division_cd)
         REFERENCES division,
    CONSTRAINT college_fk FOREIGN KEY (college_cd)
         REFERENCES college
     );

CREATE TABLE employee (
    id               NUM         NOT NULL,
    nm               CHAR(64)    NOT NULL,
    department_cd    CHAR(5)     NOT NULL,
    CONSTRAINT pk PRIMARY KEY (id),
    CONSTRAINT department_fk FOREIGN KEY (department_cd)
         REFERENCES department
    );
quit;
```

Listing 16. DDL code for Figure 2.

```
proc sql;
CREATE TABLE organization (
    cd                          CHAR(5)      NOT NULL,
    nm                          CHAR(64)     NOT NULL,
    parent_organization_cd      CHAR(5),
    CONSTRAINT pk PRIMARY KEY (cd)
    );

CREATE TABLE employee (
    id                  NUM         NOT NULL     format=z6.,
    nm                  CHAR(64)    NOT NULL,
    organization_cd     CHAR(5)     NOT NULL,
    CONSTRAINT pk PRIMARY KEY (id),
    CONSTRAINT organization_fk FOREIGN KEY (organization_cd)
        REFERENCES organization
      );
quit;
```

Listing 17. DDL code for Figure 5.

```
proc sql;
CREATE TABLE organization_path (
    ancestor_organization_cd      CHAR(5)     NOT NULL,
    descendant_organization_cd    CHAR(5)     NOT NULL,
    path_length                   NUM         NOT NULL     format=4.,
    CONSTRAINT pk PRIMARY KEY (
        ancestor_organization_cd,
        descendant_organization_cd
        )
    );
quit;
```

Listing 18. DDL code for Figure 6.

```
proc sql;
CREATE TABLE organization (
    cd                          CHAR(5)      NOT NULL,
      nm                          CHAR(64)     NOT NULL,
    parent_organization_cd      CHAR(5),
      lft                         NUM          NOT NULL,
      rgt                         NUM          NOT NULL,
    CONSTRAINT pk PRIMARY KEY (cd)
      );
CREATE UNIQUE INDEX organization_ak ON organization(lft, rgt);
quit;
```

Listing 19. DDL code for Figure 9.