Subject: Traversing subtrees
Posted by DannyBoyPoker on Fri, 05 Apr 2013 14:49:02 GMT
View Forum Message <> Reply to Message

Having come across the flexible tree structure in the tutorial, I say hmm..

http://www.tonymarston.net/php-mysql/tree-structure.html

Are we going for something write-heavy, or ready-heavy? What is the frequency of structure change? Because, some structures, etc. What types of informaion do you need to obtain? Because, etc. I mean, do you need to know what information will be needed from the structure, to determine the structure that will best fit your needs. Are you finding a node and all its children, a node and all its parents. Are you finding the count of child nodes meeting certain conditions.

What you have got here, is a table for nodes, with foreign key to the TREE_TYPE and TREE_LEVEL tables, which you call TREE_TYPE_ID, and foreign key to the TREE_LEVEL table. And, you identify the parent of a node. That is, you are giving each node a 'parent' field that contains the primary key of the parent node. Let's make this more concrete. YOu have a household. There is a person in the household. And, you have a vehicle owned by this person. Then, household, person, and vehicle are levels. The vehicle may be a car, may be a bike. And, it may be used often, or not. The person may be male/female, and will have an age. There are different housetypes. In different areas. A person maybe a Democract/Republican/etc. And maybe, a person may have an overseas trip. A trip may have a purpose, a number of days, countries that were visited.

Say that you have a node, and its description is 'Ireland'. It is a 'country' level node. Its parent is some particular 'trip' level node. That example may seem pretty straightforward and obvious, if I got it right.

Okay, question: wouldn't you have to represent the root node with a NULL for the parent? Which tells me, that you are denormalised.

I wonder if I am being persnickety. Hey, I'm not all that confident with this stuff, but I see this:
http://www.tonymarston.net/php-mysql/database-design-ru-novi
ce-ninja-or-nincompoop.html#2012-11-09

And, something in here, about the basic database rules that have existed for decades. I resemble, perhaps, indeed, and cringe, at the subsequent remarks about arrogant nincompoops, designers who make 'such basic mistakes'.

Let us agree, then, that a database that isn't in 1NF might actually be worse than a card catalog. And, from the perspective of relational theory, there is no choice at all. Null is not a value, and a thing that contains a null is not a relation, and a database that can contain a null is not a relational database. I know that there is a religious war, about the sort of deal concerning what to do if you want to have an option to store someone's middle initial. But, my issue isn't Nulls anyways. It's normalisation design.

Then, suppose that you do decide to normalise the attribute. Why do you have to represent the

root node with a Null? Because, you have nodes and edges denormalised to one table. And, you cannot edit nodes and edges separately. To back up a tad, what is an edge. Well, two nodes are *adjacent* if there is an edge between them. Two edges are adjacent, if they connect to a common node. The edges are, like, lines, or arcs. Between vertices. Nodes. The edges connect them. As they say, a node of indegree zero is a root node. Similarly, a node of outdegreezero is a leaf node. But, the number of edges entering a node is its indegree. What you are doing with this schema, is you have a ragged array. There are some nodes, like A, B, C, D, E, F. And, there are some adjacent nodes, like C-->A, and B-->E, and F,D,A-->C.

I guess you can drop the 'tree-type' table, as that's just a node like any other. Just have a pointer to the root node, which can be something like 'box', or whatever. Currently, you have a nodes table, where each row specifies one node and its parent (which is NULL for the root node). But, more elaborately, you could have two tables, one for the nodes, the other a bridging table for their edges. So, your personnel, assembly parts, locations on a map, whatever, are in the nodes table. The edges table has a childID and a parentID. If you 'SELECT * from edges;', now, then you get something like this:

childID parentID
A C
B E
C D
C F

Note, that you can figure out where you can get to from here, wherever 'here' is, and this isn't even necessarily a tree. Make childID and parentID foreign keys, and you give the model referential integrity. Item detail belongs in a nodes table, logic belongs in an edges table. That's a normalising rule. What if you have a bill of materials, which has subassemblies, which also have bills of materials. You might want to link component pieces to a major assembly, or break apart assemblies and subassemblies into their component parts. Then, items could occur in multiple assemblies and subassemblies. This would not be a tree. No problem. Nodes like this, laminate, planks, shelf supports, screws, wood cubes, boxes. I'm using examples of raw materials. And, edges like this: parentID="thus", childID="so".

'Edges' is a bit counterintuitive, let's call that 'assemblies'. We might also call 'nodes' something like 'items', whatever.
Thinking in terms of a parts list, a bill of materials, is how I am excogitating this.

Why is it good not to be stuck with trees. What if the gig is, there's an airline, with licenses for short flights. And, we roughly compute routes. So, we create a level that we call 'airport'. We enter a bunch of nodes that are airports. Like, LAX, JFK. And we want to have flights entered in there. So, LAX-JFK is a flight. See, we can enter all the flights, going this way and going back. Heathrow as a parent, Charles de Gaulle as a child, and vice-versa. Now, we get into maybe distance and direction, which would be attached to edges. Which is where I will leave off. Although, distance and direction would be what are called 'weights'. And we get into a whole other kind of thing. Is this starting to sound abstrurse, irrelevant. What am I, some kind of computer geek? But, what is a 'network', what does the term mean, technically? I mean, in the sense that the whole internet runs on a network, and somehow, computers deal with this. Well, you got your nodes, you got your

edges entering a node, the number of which, is its 'indegree', which, when it is zero, is a 'root node'. Etc. Make those edges weighted, and you've got a 'network'. The edges are not, however, directed. There are all these things, organisational charts, itineraries, route maps, parts exposions, language rules, massively multiplayer games, chat histories..

The matter of network and link analysis comes up, in a wide variety of fields. There are search engines, forensics, epidemiology, telecommunications. There's data mining. There's models of chemical structure hierarchies. There's biochemical processes.

There's nodes and edges. It works. One might, then, consider normalizing that schema!
