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!

Subject: Re: Traversing subtrees
Posted by AJM on Fri, 05 Apr 2013 16:28:24 GMT
View Forum Message <> Reply to Message

Why do you think that a record with a null parent_id is not normalised? It signifies that a node does not have a parent, which either means that it is at the top of the tree, or it has been orphaned (ie: waiting to be assigned to a parent).

My tree structure is not supposed to be the only way to represent a tree, it just happens to be the one that suited my purposes many years ago when a particular requirement arose. There may be other ways to handle tree structures, so you are free to implement whatever takes your fancy.

Subject: Re: Traversing subtrees
Posted by DannyBoyPoker on Sat, 06 Apr 2013 01:22:34 GMT
View Forum Message <> Reply to Message

The table isn't normalized, although, okay, let's step through this..this is stuff on the plate, to get into, in general, about data. I guess I find it interesting, as stuff I am supposed to know..let's see if I can thread this needle..

You point out, that there may be other ways to handle tree structures, which, okay, sure, but that would be an implementation detail. On the other hand, a tree is a collection of nodes and edges, with rules dictating the connection among the nodes. Actually, there are numerous rules spelling out how these connections can occur. Consider, for example, if you want to enter an itinerary, such as gym, restaurant, work, restaurant, home. You can't. What is restaurant's parent? But, I can.

Like this:
1 gym
2 restaurant
3 work
4 home

And, in the 'edges' table:
Parent Child
1 2
2 3
3 2
2 4

So, here, a node can have multiple parents, as it can have multiple children.

You ask: 'Why do you think that a record with a null parent_id is not normalised?'
Well, one argument against nulls is that they don't have a well-defined interpretation.

To this, you offer that the data means exactly what it says and there's no duplication. Suppose that we want multiple roots, well, one may construe this as a single tree with a null root. Our multiple roots are children of a null parent. Suppose that I am demanding that all values and data types should have well-defined interpretations, therefore all nulls are bad. In this case, then, I have been answered. It's easy and it's normalized...I mean, well, it's easy anyways, it's a pretty straightforward design that's easily understood by everyone, in which case, no deep relational theory needed.

Beyond that, well, the topic of hierarchies has been beaten with an ugly stick, I suppose.

'My tree structure is not supposed to be the only way to represent a tree, it just happens to be the one that suited my purposes many years ago when a particular requirement arose.'

We're doing set operations. That, is what gives you mathematical control--control. It's not a matter of what is my purpose, my strategy. It's a matter of control. Here is my particular requirement: I want to model a collection of 'things', and their relationship among these 'things'. Sets may be structured, or not. If they are structured, then they may form a tree, or other structure. Sets can be expressed within tree structures in a number of ways. How? Well, sets are groups of resources that share a common property.That property may be a container. And tree structures are largely defined by containment relationships.

There is a formal foundation, here. Database theory. Maybe I can demonstrate its practical value, and maybe I can't, but there are the formal foundations of a set-theoretic data model. All data representations are treated
as mathematical objects. Maybe it's an important approach, and maybe it isn't.

Subject: Re: Traversing subtrees
Posted by AJM on Sat, 06 Apr 2013 10:32:23 GMT
View Forum Message <> Reply to Message

Those tables *ARE* normalised. If you think they are not then please identify which Normal Form they are violating. Missing a "well-defined interpretation" is not in any definition of any normal

form.

It is quite clear to me that you haven't understood the types of hierarchy that my tree structure is supposed to represent. My article clearly shows that it deals with an organisational hierarchy with different levels such as COMPANY, DEPARTMENT and SECTION where there can be a number of entities at each of those levels. The rule is that any entity below the top level can have only one parent, and that parent must belong to next highest level. I designed this structure decades ago for a payroll system where people at a certain level in the hierarchy could only "see" the records for those people who were their subordinates in that hierarchy. This structure made it easy to add levels, to maintain the list of nodes that existed at each level, and to link (including unlink+relink) a person with any node in that structure.

If you try to use this structure to represent anything more complex, such as your itinerary example, then you are wasting your time. You will have to do that using separate tables with specialised relationships.

Subject: Re: Traversing subtrees
Posted by DannyBoyPoker on Sat, 06 Apr 2013 11:02:39 GMT
View Forum Message <> Reply to Message

Which normal form, is 1NF. Which is, only atomic values, only a single value from that domain. The reasoning behind this, is that none of the domains of that relation should have elements which are themselves sets. Such as, the set of parents.

Subject: Re: Traversing subtrees
Posted by DannyBoyPoker on Sat, 06 Apr 2013 11:42:06 GMT
View Forum Message <> Reply to Message

Maybe I should add, that my perspective, is that almost every couple of months a question about how to model a tree in the database pops up at the comp.database.theory newsgroup. And, in most projects we have to deal with some kind of trees or hierarchies! Categories, organisation, threads, folders, components.

My comportment, then, is one of curiosity, the parlance is kind of overwhelming, there is the simple graph (trees are these), and, the multigraph. A 'graph', is just a structure, in theory. And a structure, consists of a set, along with a collection of finitary operations (what is finitary? well, not infinitary), and relations that are defined on it. Etc.

I'd figured, that actually I posted the structure to use, to represent anything more complex.

Subject: Re: Traversing subtrees
Posted by DannyBoyPoker on Sat, 06 Apr 2013 13:14:43 GMT
View Forum Message <> Reply to Message

I might get some more pushback, about whether that table is normalized. Well, with my change, I managed integrity constraints, to be enforced, and parents and children. One might argue that life is not all about data integrity, and win. But, the table still is not normalized. Let's go back over that, what table. I mean, the single nodes table, where each row specifies one node and its parent. Which, its parent, is NULL for the root node.

Can I give this referential integrity. Yes, I did this by suggesting that childID and parentID can be foreign keys. Like this--I suggested that two tables, one for the nodes, and the other a bridging table for their edges, would be doing things more elaborately. And now, the nodes can be personnel, or assembly parts, or
locations on a map. Also, the edges table could have additional columns for edge properties. And now, whether a graph is connected, directed, whether it is a tree, whatever, one may write a procedure which tells where you can get to from here, wherever 'here' is.

So we have a nodes table, and an edges table like this:

```
CREATE TABLE edges(
  childID CHAR(1) NOT NULL,
  parentID CHAR(1) NOT NULL,
  PRIMARY KEY(childID,parentID)
);
```

So, we select * from edges and get this:

```
| childID | parentID |
+---------+----------+
| A       | C        |
| B       | E        |
| C       | D        |
| C       | F        |
+---------+----------+
```

Maybe I ought to provide a simple approach. Here is a breadth-first search, as a MySQL stored procedure:

```
CREATE PROCEDURE ListReached( IN root CHAR(1) )
BEGIN
  DECLARE rows SMALLINT DEFAULT 0;
  DROP TABLE IF EXISTS reached;
  CREATE TABLE reached (
    nodeID CHAR(1) PRIMARY KEY
  ) ENGINE=HEAP;
  INSERT INTO reached VALUES (root );
  SET rows = ROW_COUNT();
  WHILE rows > 0 DO
    INSERT IGNORE INTO reached
      SELECT DISTINCT childID
```

```
    FROM edges AS e
    INNER JOIN reached AS p ON e.parentID = p.nodeID;
  SET rows = ROW_COUNT();
  INSERT IGNORE INTO reached
    SELECT DISTINCT parentID
    FROM edges AS e
    INNER JOIN reached AS p ON e.childID = p.nodeID;
  SET rows = rows + ROW_COUNT();
 END WHILE;
 SELECT * FROM reached;
 DROP TABLE reached;
END;
```

And, let's CALL ListReached('A'), we get NodeID = A, C, D, F. Which isn't so versatile, what about giving it input parameters which tell it whether to list child, parent, all connections, whether to recognise loops.
That's the normalization part. And, one may add, delete a node. One may also change an edge. And, this an be a connected or treelike graph, or not.

I make a big deal about it, that you have to represent the root node with a NULL, I claim, that this is working with nodes and edges denormalised to one table. Now, supposedly, denormalisation can cost, big time. Is this true, in this case? Yes, and the bigger the table, the bigger the cost. You cannot edit nodes and edges separately. Also, carrying extra node information during edge computation slows performance.

I suggested that 'parent' is a set, you might reply that there is only one parent. Even though, that means that our model cannot model a family, though we speak of parents and children. Your reply, is that when you use a different tree model, you'll have to mess with the data being modelled.

But, now that's fixed.

One will now need some functions, though, like what if I want to return a node description for a parent or child ID in the 'edges' table.
Or, call it the 'familyTree' table (I suggested 'assemblies' before, as well, maybe there's no improving on 'edges').

Here:

```
CREATE FUNCTION NodeDescription( personID SMALLINT )
RETURNS CHAR(20)
BEGIN
 DECLARE NodeDescription CHAR(20) DEFAULT '';
 SELECT description INTO NodeDescription FROM edges WHERE ID=nodeID;
 RETURN NodeDescription;
END;
```

So, what can this do.

Something like this:
SELECT NodeDescription(childID) AS 'reports directly to Bill Gates'
FROM edges
WHERE parentID = (SELECT ID FROM edges WHERE description = 'Bill Gates');

And you get, like, three people returned, maybe.

At the same time, the function can be used like this:

SELECT nodeDescription(childID) AS subordinate, nodeDescription(parentID) AS superior
FROM edges;

And you get a list of each subordinate, and their superior.

I suppose that I stick in a foreign key on delete cascade on update cascade, for the foreign key(parentID), in my 'edges' table.

Then, I can delete from 'edges' a particular node. Suppose, that leaves your company hierarchy, for example.
Then it's just 'delete from edges where id = 2;'

So, we have simple queries to retrieve basic facts about the tree, perhaps? Can I collect parent nodes, with their children?
Easy:

SELECT parentID AS Parent, GROUP_CONCAT(childID) AS Children
FROM edges JOIN nodes ON edges.parentID=f.ID
GROUP BY parentID;

A list of parent, and the children of each parent. Can we retrieve subtree statistics. Do a left join. Ask for the root node, with some info, or ask for the leaf nodes, with some info. Maybe it's a list of dead people, and you have their age, or their birth and death date, at least. You can pull the average age of leaf nodes (childless) with a left join.

Inserting a new node requires no revision of existing rows. Add a new node row, then a new edges row. Deletion is a two-step, delete the edges row, delete the node row.

What about traversing subtrees.

It can be done, in a generic version, general purpose. And there are no serious scaling issues. Insert a subtree, point its top edge at an existing node as parent,and..INSERT. This idea, is flexible. There is depth-first subtree traversal, then. What about path enumeration. Doable. Want the procedure? I'm capable of being less breezy. Seems to me, that this is framework stuff. I'm pulling this actual MySQL (there's more) from a handout that I got in a database class in graduate school. Where, we had to all implement our own database server. Not that I'm a good student--although, prof was mightily amused that I did so providing a command-line interface, as

everybody else put a Java Swing interface on it. 'Old school, cool!'

Subject: Re: Traversing subtrees
Posted by AJM on Sat, 06 Apr 2013 13:39:23 GMT
View Forum Message <> Reply to Message

Your definition of 1NF is wrong! The *REAL* definition is this:
Quote:A table is in first normal form if all the key attributes have been defined and it contains no repeating groups

There are no repeating groups in those tables, so they do *NOT* violate 1NF.

Subject: Re: Traversing subtrees
Posted by AJM on Sat, 06 Apr 2013 13:44:02 GMT
View Forum Message <> Reply to Message

I too have seen many articles regarding hierarchical structures and how to implement them. There is no single solution. The one described in my article covers a simple organisational hierarchy with simple rules. If you have a different type of hierarchy with different rules then my solution will not be relevant.

Subject: Re: Traversing subtrees
Posted by AJM on Sat, 06 Apr 2013 13:50:29 GMT
View Forum Message <> Reply to Message

Foreign key constraints are irrelevant as far as my article is concerned.

You appear to be telling me how *YOU* would implement your own tree structure, so my answer is "go ahead, I'm not stopping you". But what has that got to do with the RADICORE framework?
