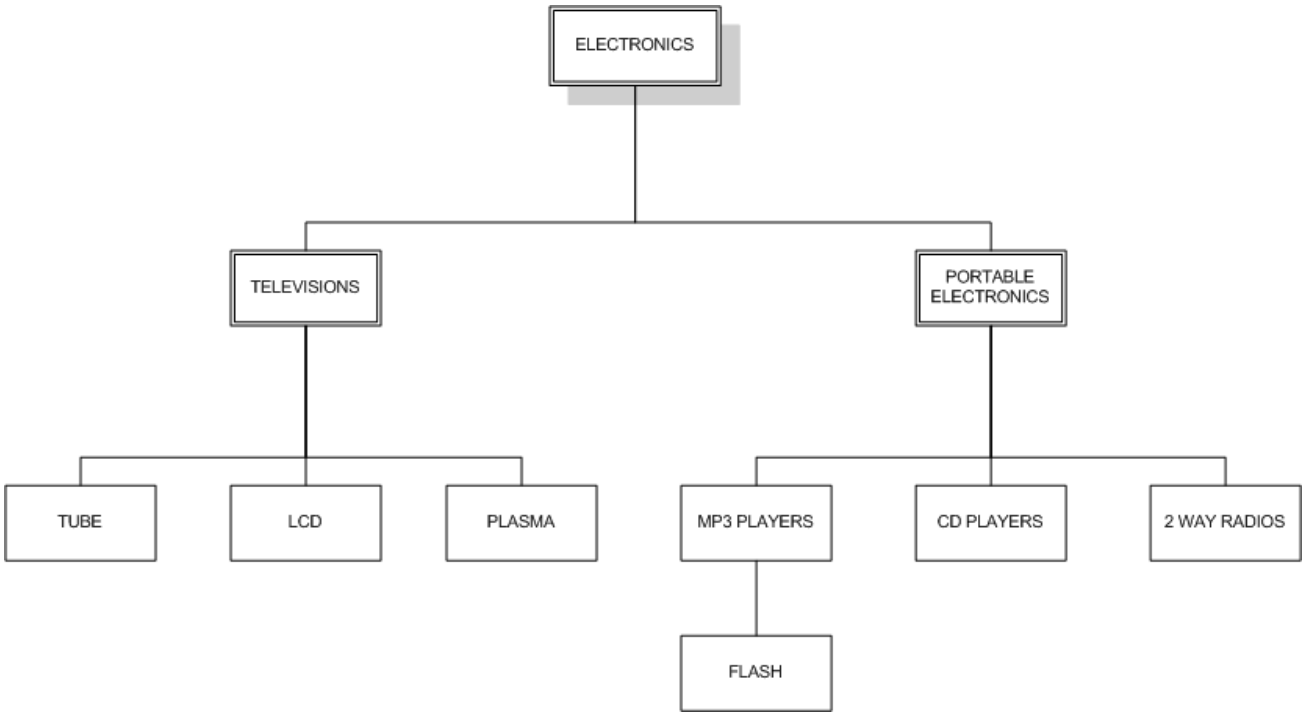
Managing Hierarchical Data in MySQL

Introduction

Most users at one time or another have dealt with hierarchical data in a SQL database and no doubt learned that the management of hierarchical data is not what a relational database is intended for. The tables of a relational database are not hierarchical (like XML), but are simply a flat list. Hierarchical data has a parent-child relationship that is not naturally represented in a relational database table.

For our purposes, hierarchical data is a collection of data where each item has a single parent and zero or more children (with the exception of the root item, which has no parent). Hierarchical data can be found in a variety of database applications, including forum and mailing list threads, business organization charts, content management categories, and product categories. For our purposes we will use the following product category hierarchy from an fictional electronics store:



These categories form a hierarchy in much the same way as the other examples cited above. In this article we will examine two models for dealing with hierarchical data in MySQL, starting with the traditional adjacency list model.

The Adjacency List Model

Typically the example categories shown above will be stored in a table like the following (I'm including full CREATE and INSERT statements so you can follow along):

```
CREATE TABLE category(
        category_id INT AUTO_INCREMENT PRIMARY KEY,
        name VARCHAR(20) NOT NULL,
        parent INT DEFAULT NULL
);

INSERT INTO category VALUES(1,'ELECTRONICS',NULL),(2,'TELEVISIONS',1),
(3,'TUBE',2),
        (4,'LCD',2),(5,'PLASMA',2),(6,'PORTABLE ELECTRONICS',1),(7,'MP3
PLAYERS',6),(8,'FLASH',7),
        (9,'CD PLAYERS',6),(10,'2 WAY RADIOS',6);

SELECT * FROM category ORDER BY category_id;
+-------------+----------------------+--------+
| category_id | name                 | parent |
+-------------+----------------------+--------+
```

```
|           1 | ELECTRONICS          |   NULL |
|           2 | TELEVISIONS          |      1 |
|           3 | TUBE                 |      2 |
|           4 | LCD                  |      2 |
|           5 | PLASMA               |      2 |
|           6 | PORTABLE ELECTRONICS |      1 |
|           7 | MP3 PLAYERS          |      6 |
|           8 | FLASH                |      7 |
|           9 | CD PLAYERS           |      6 |
|          10 | 2 WAY RADIOS         |      6 |
+-------------+----------------------+--------+
10 rows in set (0.00 sec)
```

In the adjacency list model, each item in the table contains a pointer to its parent. The topmost element, in this case *electronics*, has a NULL value for its parent. The adjacency list model has the advantage of being quite simple, it is easy to see that*FLASH* is a child of*mp3* players, which is a child of *portable electronics*, which is a child of *electronics*. While the adjacency list model can be dealt with fairly easily in client-side code, working with the model can be more problematic in pure SQL.

RETRIEVING A FULL TREE

The first common task when dealing with hierarchical data is the display of the entire tree, usually with some form of indentation. The most common way of doing this is in pure SQL is through the use of a self-join:

```
SELECT t1.name AS lev1, t2.name as lev2, t3.name as lev3, t4.name as lev4
FROM category AS t1
LEFT JOIN category AS t2 ON t2.parent = t1.category_id
LEFT JOIN category AS t3 ON t3.parent = t2.category_id
LEFT JOIN category AS t4 ON t4.parent = t3.category_id
WHERE t1.name = 'ELECTRONICS';

+-------------+----------------------+--------------+-------+
| lev1        | lev2                 | lev3         | lev4  |
+-------------+----------------------+--------------+-------+
| ELECTRONICS | TELEVISIONS          | TUBE         | NULL  |
| ELECTRONICS | TELEVISIONS          | LCD          | NULL  |
| ELECTRONICS | TELEVISIONS          | PLASMA       | NULL  |
| ELECTRONICS | PORTABLE ELECTRONICS | MP3 PLAYERS  | FLASH |
| ELECTRONICS | PORTABLE ELECTRONICS | CD PLAYERS   | NULL  |
```

```
| ELECTRONICS | PORTABLE ELECTRONICS | 2 WAY RADIOS | NULL  |
+-------------+----------------------+--------------+-------+
6 rows in set (0.00 sec)
```

FINDING ALL THE LEAF NODES

We can find all the leaf nodes in our tree (those with no children) by using a LEFT JOIN query:

```
SELECT t1.name FROM
category AS t1 LEFT JOIN category as t2
ON t1.category_id = t2.parent
WHERE t2.category_id IS NULL;

+--------------+
| name         |
+--------------+
| TUBE         |
| LCD          |
| PLASMA       |
| FLASH        |
| CD PLAYERS   |
| 2 WAY RADIOS |
+--------------+
```

RETRIEVING A SINGLE PATH

The self-join also allows us to see the full path through our hierarchies:

```
SELECT t1.name AS lev1, t2.name as lev2, t3.name as lev3, t4.name as lev4
FROM category AS t1
LEFT JOIN category AS t2 ON t2.parent = t1.category_id
LEFT JOIN category AS t3 ON t3.parent = t2.category_id
LEFT JOIN category AS t4 ON t4.parent = t3.category_id
WHERE t1.name = 'ELECTRONICS' AND t4.name = 'FLASH';

+-------------+----------------------+------------+-------+
| lev1        | lev2                 | lev3       | lev4  |
```

```
+-------------+----------------------+-------------+-------+
| ELECTRONICS | PORTABLE ELECTRONICS | MP3 PLAYERS | FLASH |
+-------------+----------------------+-------------+-------+
1 row in set (0.01 sec)
```

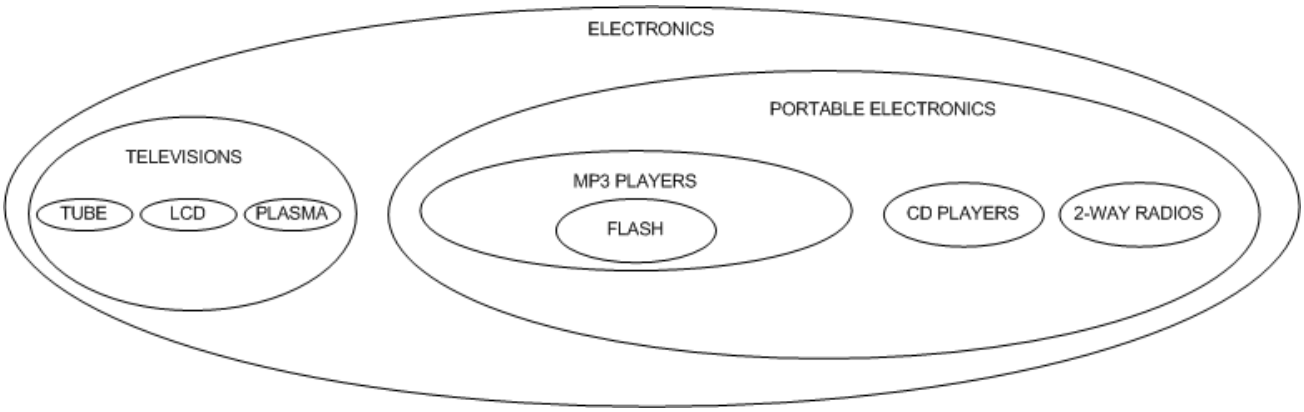
The main limitation of such an approach is that you need one self-join for every level in the hierarchy, and performance will naturally degrade with each level added as the joining grows in complexity.

LIMITATIONS OF THE ADJACENCY LIST MODEL

Working with the adjacency list model in pure SQL can be difficult at best. Before being able to see the full path of a category we have to know the level at which it resides. In addition, special care must be taken when deleting nodes because of the potential for orphaning an entire sub-tree in the process (delete the portable electronics category and all of its children are orphaned). Some of these limitations can be addressed through the use of client-side code or stored procedures. With a procedural language we can start at the bottom of the tree and iterate upwards to return the full tree or a single path. We can also use procedural programming to delete nodes without orphaning entire sub-trees by promoting one child element and re-ordering the remaining children to point to the new parent.

The Nested Set Model

What I would like to focus on in this article is a different approach, commonly referred to as the **Nested Set Model**. In the Nested Set Model, we can look at our hierarchy in a new way, not as nodes and lines, but as nested containers. Try picturing our electronics categories this way:

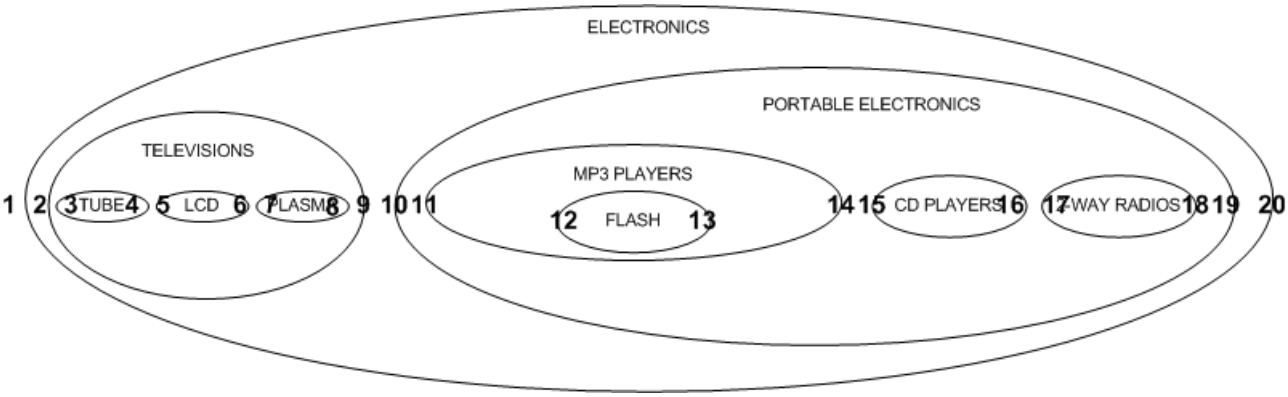


Notice how our hierarchy is still maintained, as parent categories envelop their children.We represent this form of hierarchy in a table through the use of left and right values to represent the nesting of our nodes:

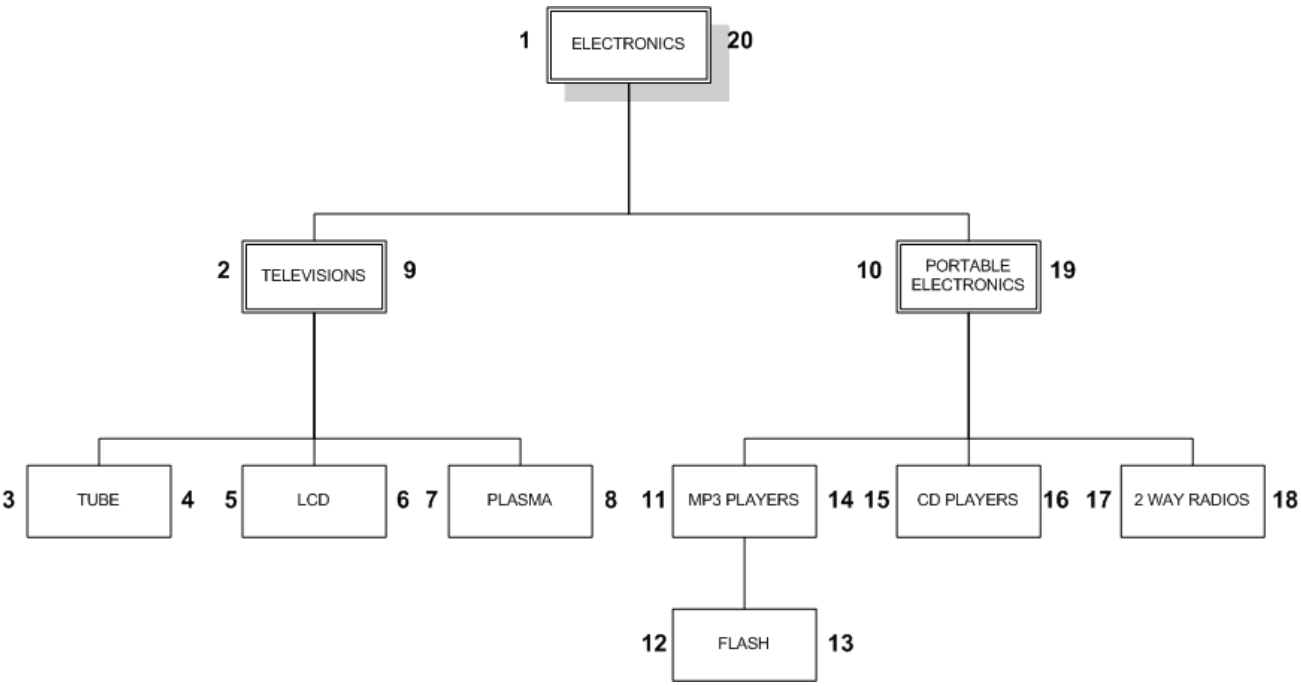
```
CREATE TABLE nested_category (
        category_id INT AUTO_INCREMENT PRIMARY KEY,
        name VARCHAR(20) NOT NULL,
        lft INT NOT NULL,
        rgt INT NOT NULL
);

INSERT INTO nested_category VALUES(1,'ELECTRONICS',1,20),
(2,'TELEVISIONS',2,9),(3,'TUBE',3,4),
 (4,'LCD',5,6),(5,'PLASMA',7,8),(6,'PORTABLE ELECTRONICS',10,19),(7,'MP3
PLAYERS',11,14),(8,'FLASH',12,13),
 (9,'CD PLAYERS',15,16),(10,'2 WAY RADIOS',17,18);

SELECT * FROM nested_category ORDER BY category_id;

+-------------+----------------------+-----+-----+
| category_id | name                 | lft | rgt |
+-------------+----------------------+-----+-----+
|           1 | ELECTRONICS          |   1 |  20 |
|           2 | TELEVISIONS          |   2 |   9 |
|           3 | TUBE                 |   3 |   4 |
|           4 | LCD                  |   5 |   6 |
|           5 | PLASMA               |   7 |   8 |
|           6 | PORTABLE ELECTRONICS |  10 |  19 |
|           7 | MP3 PLAYERS          |  11 |  14 |
|           8 | FLASH                |  12 |  13 |
|           9 | CD PLAYERS           |  15 |  16 |
|          10 | 2 WAY RADIOS         |  17 |  18 |
+-------------+----------------------+-----+-----+
```

We use **lft** and **rgt** because *left* and *right* are reserved words in MySQL, see http://dev.mysql.com/doc/mysql/en/reserved-words.html for the full list of reserved words.

So how do we determine left and right values? We start numbering at the leftmost side of the outer node and continue to the right:



This design can be applied to a typical tree as well:



When working with a tree, we work from left to right, one layer at a time, descending to each node's children before assigning a right-hand number and moving on to the right. This approach is called the **modified preorder tree traversal algorithm**.

RETRIEVING A FULL TREE

We can retrieve the full tree through the use of a self-join that links parents with nodes on the basis that a node's **lft** value will always appear between its parent's **lft** and **rgt** values:

```
SELECT node.name
FROM nested_category AS node,
        nested_category AS parent
WHERE node.lft BETWEEN parent.lft AND parent.rgt
        AND parent.name = 'ELECTRONICS'
ORDER BY node.lft;

+----------------------+
| name                 |
+----------------------+
| ELECTRONICS          |
| TELEVISIONS          |
| TUBE                 |
| LCD                  |
| PLASMA               |
| PORTABLE ELECTRONICS |
| MP3 PLAYERS          |
| FLASH                |
| CD PLAYERS           |
| 2 WAY RADIOS         |
+----------------------+
```

Unlike our previous examples with the adjacency list model, this query will work regardless of the depth of the tree. We do not concern ourselves with the rgt value of the node in our BETWEEN clause because the rgt value will always fall within the same parent as the lft values.

FINDING ALL THE LEAF NODES

Finding all leaf nodes in the nested set model even simpler than the LEFT JOIN method used in the adjacency list model. If you look at the nested_category table, you may notice that the lft and rgt values for leaf nodes are consecutive numbers. To find the leaf nodes, we look for nodes where rgt = lft + 1:

```
SELECT name
FROM nested_category
WHERE rgt = lft + 1;

+--------------+
| name         |
+--------------+
| TUBE         |
| LCD          |
| PLASMA       |
| FLASH        |
| CD PLAYERS   |
| 2 WAY RADIOS |
+--------------+
```

RETRIEVING A SINGLE PATH

With the nested set model, we can retrieve a single path without having multiple self-joins:

```
SELECT parent.name
FROM nested_category AS node,
        nested_category AS parent
WHERE node.lft BETWEEN parent.lft AND parent.rgt
        AND node.name = 'FLASH'
ORDER BY parent.lft;

+----------------------+
| name                 |
+----------------------+
| ELECTRONICS          |
| PORTABLE ELECTRONICS |
| MP3 PLAYERS          |
| FLASH                |
+----------------------+
```

FINDING THE DEPTH OF THE NODES

We have already looked at how to show the entire tree, but what if we want to also show the depth of each node in the tree, to better identify how each node fits in the hierarchy? This can be done by adding a COUNT function and a GROUP BY clause to our existing query for showing the entire tree:

```
SELECT node.name, (COUNT(parent.name) - 1) AS depth
FROM nested_category AS node,
        nested_category AS parent
WHERE node.lft BETWEEN parent.lft AND parent.rgt
GROUP BY node.name
ORDER BY node.lft;

+----------------------+-------+
| name                 | depth |
+----------------------+-------+
| ELECTRONICS          |     0 |
| TELEVISIONS          |     1 |
| TUBE                 |     2 |
| LCD                  |     2 |
| PLASMA               |     2 |
| PORTABLE ELECTRONICS |     1 |
| MP3 PLAYERS          |     2 |
| FLASH                |     3 |
| CD PLAYERS           |     2 |
| 2 WAY RADIOS         |     2 |
+----------------------+-------+
```

We can use the depth value to indent our category names with the CONCAT and REPEAT string functions:

```
SELECT CONCAT( REPEAT(' ', COUNT(parent.name) - 1), node.name) AS name
FROM nested_category AS node,
        nested_category AS parent
WHERE node.lft BETWEEN parent.lft AND parent.rgt
GROUP BY node.name
ORDER BY node.lft;

+----------------------+
| name                 |
+----------------------+
| ELECTRONICS          |
|  TELEVISIONS         |
```

```
|   TUBE                |
|   LCD                 |
|   PLASMA              |
|  PORTABLE ELECTRONICS |
|   MP3 PLAYERS         |
|    FLASH              |
|   CD PLAYERS          |
|   2 WAY RADIOS        |
+-----------------------+
```

Of course, in a client-side application you will be more likely to use the depth value directly to display your hierarchy. Web developers could loop through the tree, adding and tags as the depth number increases and decreases.

DEPTH OF A SUB-TREE

When we need depth information for a sub-tree, we cannot limit either the node or parent tables in our self-join because it will corrupt our results. Instead, we add a third self-join, along with a sub-query to determine the depth that will be the new starting point for our sub-tree:

```
SELECT node.name, (COUNT(parent.name) - (sub_tree.depth + 1)) AS depth
FROM nested_category AS node,
        nested_category AS parent,
        nested_category AS sub_parent,
        (
                SELECT node.name, (COUNT(parent.name) - 1) AS depth
                FROM nested_category AS node,
                nested_category AS parent
                WHERE node.lft BETWEEN parent.lft AND parent.rgt
                AND node.name = 'PORTABLE ELECTRONICS'
                GROUP BY node.name
                ORDER BY node.lft
        )AS sub_tree
WHERE node.lft BETWEEN parent.lft AND parent.rgt
        AND node.lft BETWEEN sub_parent.lft AND sub_parent.rgt
        AND sub_parent.name = sub_tree.name
GROUP BY node.name
ORDER BY node.lft;
```

```
+----------------------+-------+
| name                 | depth |
+----------------------+-------+
| PORTABLE ELECTRONICS |     0 |
| MP3 PLAYERS          |     1 |
| FLASH                |     2 |
| CD PLAYERS           |     1 |
| 2 WAY RADIOS         |     1 |
+----------------------+-------+
```

This function can be used with any node name, including the root node. The depth values are always relative to the named node.

FIND THE IMMEDIATE SUBORDINATES OF A NODE

Imagine you are showing a category of electronics products on a retailer web site. When a user clicks on a category, you would want to show the products of that category, as well as list its immediate sub-categories, but not the entire tree of categories beneath it. For this, we need to show the node and its immediate sub-nodes, but no further down the tree. For example, when showing the PORTABLE ELECTRONICS category, we will want to show MP3 PLAYERS, CD PLAYERS, and 2 WAY RADIOS, but not FLASH.

This can be easily accomplished by adding a HAVING clause to our previous query:

```
SELECT node.name, (COUNT(parent.name) - (sub_tree.depth + 1)) AS depth
FROM nested_category AS node,
        nested_category AS parent,
        nested_category AS sub_parent,
        (
                SELECT node.name, (COUNT(parent.name) - 1) AS depth
                FROM nested_category AS node,
                        nested_category AS parent
                WHERE node.lft BETWEEN parent.lft AND parent.rgt
                        AND node.name = 'PORTABLE ELECTRONICS'
                GROUP BY node.name
                ORDER BY node.lft
        )AS sub_tree
WHERE node.lft BETWEEN parent.lft AND parent.rgt
        AND node.lft BETWEEN sub_parent.lft AND sub_parent.rgt
```

```
        AND sub_parent.name = sub_tree.name
GROUP BY node.name
HAVING depth <= 1
ORDER BY node.lft;

+----------------------+-------+
| name                 | depth |
+----------------------+-------+
| PORTABLE ELECTRONICS |     0 |
| MP3 PLAYERS          |     1 |
| CD PLAYERS           |     1 |
| 2 WAY RADIOS         |     1 |
+----------------------+-------+
```

If you do not wish to show the parent node, change the **HAVING depth <= 1** line to **HAVING depth = 1**.

AGGREGATE FUNCTIONS IN A NESTED SET

Let's add a table of products that we can use to demonstrate aggregate functions with:

```
CREATE TABLE product
(
        product_id INT AUTO_INCREMENT PRIMARY KEY,
        name VARCHAR(40),
        category_id INT NOT NULL
);

INSERT INTO product(name, category_id) VALUES('20" TV',3),('36" TV',3),
('Super-LCD 42"',4),('Ultra-Plasma 62"',5),('Value Plasma 38"',5),
('Power-MP3 5gb',7),('Super-Player 1gb',8),('Porta CD',9),('CD To
go!',9),
('Family Talk 360',10);

SELECT * FROM product;

+------------+--------------------+-------------+
| product_id | name               | category_id |
+------------+--------------------+-------------+
|          1 | 20" TV             |           3 |
|          2 | 36" TV             |           3 |
|          3 | Super-LCD 42"      |           4 |
```

```
|          4 | Ultra-Plasma 62"  |           5 |
|          5 | Value Plasma 38"  |           5 |
|          6 | Power-MP3 128mb   |           7 |
|          7 | Super-Shuffle 1gb |           8 |
|          8 | Porta CD          |           9 |
|          9 | CD To go!         |           9 |
|         10 | Family Talk 360   |          10 |
+------------+-------------------+-------------+
```

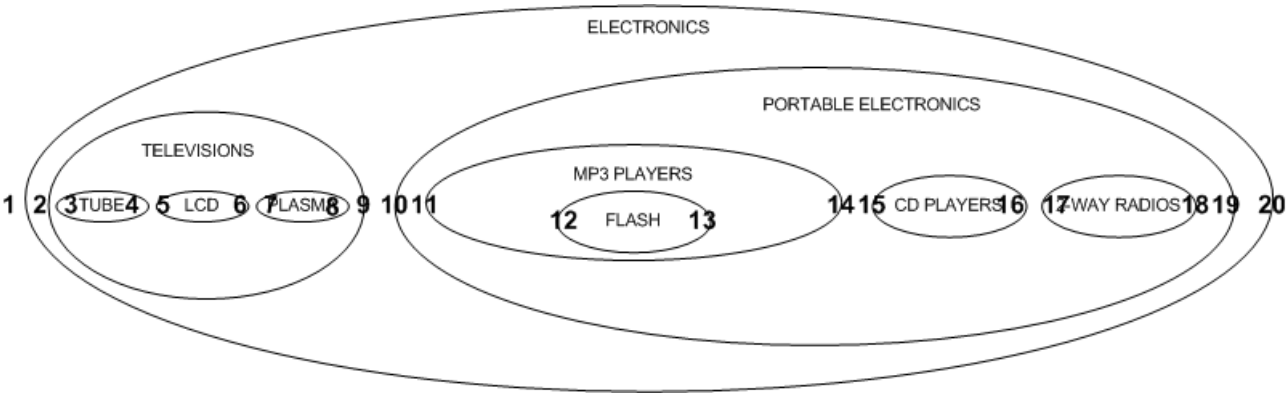
Now let's produce a query that can retrieve our category tree, along with a product count for each category:

```
SELECT parent.name, COUNT(product.name)
FROM nested_category AS node ,
        nested_category AS parent,
        product
WHERE node.lft BETWEEN parent.lft AND parent.rgt
        AND node.category_id = product.category_id
GROUP BY parent.name
ORDER BY node.lft;

+----------------------+---------------------+
| name                 | COUNT(product.name) |
+----------------------+---------------------+
| ELECTRONICS          |                  10 |
| TELEVISIONS          |                   5 |
| TUBE                 |                   2 |
| LCD                  |                   1 |
| PLASMA               |                   2 |
| PORTABLE ELECTRONICS |                   5 |
| MP3 PLAYERS          |                   2 |
| FLASH                |                   1 |
| CD PLAYERS           |                   2 |
| 2 WAY RADIOS         |                   1 |
+----------------------+---------------------+
```

This is our typical whole tree query with a COUNT and GROUP BY added, along with a reference to the product table and a join between the node and product table in the WHERE clause. As you can see, there is a count for each category and the count of subcategories is reflected in the parent categories.

ADDING NEW NODES

Now that we have learned how to query our tree, we should take a look at how to update our tree by adding a new node. Let's look at our nested set diagram again:



If we wanted to add a new node between the TELEVISIONS and PORTABLE ELECTRONICS nodes, the new node would have lft and rgt values of 10 and 11, and all nodes to its right would have their lft and rgt values increased by two. We would then add the new node with the appropriate lft and rgt values. While this can be done with a stored procedure in MySQL 5, I will assume for the moment that most readers are using 4.1, as it is the latest stable version, and I will isolate my queries with a LOCK TABLES statement instead:

```
LOCK TABLE nested_category WRITE;

SELECT @myRight := rgt FROM nested_category
WHERE name = 'TELEVISIONS';

UPDATE nested_category SET rgt = rgt + 2 WHERE rgt > @myRight;
UPDATE nested_category SET lft = lft + 2 WHERE lft > @myRight;

INSERT INTO nested_category(name, lft, rgt) VALUES('GAME CONSOLES',
@myRight + 1, @myRight + 2);

UNLOCK TABLES;
```

We can then check our nesting with our indented tree query:

```
SELECT CONCAT( REPEAT( ' ', (COUNT(parent.name) - 1) ), node.name) AS
name
```

```
FROM nested_category AS node,
        nested_category AS parent
WHERE node.lft BETWEEN parent.lft AND parent.rgt
GROUP BY node.name
ORDER BY node.lft;

+-----------------------+
| name                  |
+-----------------------+
| ELECTRONICS           |
|  TELEVISIONS          |
|   TUBE                |
|   LCD                 |
|   PLASMA              |
|  GAME CONSOLES        |
|  PORTABLE ELECTRONICS |
|   MP3 PLAYERS         |
|    FLASH              |
|   CD PLAYERS          |
|   2 WAY RADIOS        |
+-----------------------+
```

If we instead want to add a node as a child of a node that has no existing children, we need to modify our procedure slightly. Let's add a new FRS node below the 2 WAY RADIOS node:

```
LOCK TABLE nested_category WRITE;

SELECT @myLeft := lft FROM nested_category

WHERE name = '2 WAY RADIOS';

UPDATE nested_category SET rgt = rgt + 2 WHERE rgt > @myLeft;
UPDATE nested_category SET lft = lft + 2 WHERE lft > @myLeft;

INSERT INTO nested_category(name, lft, rgt) VALUES('FRS', @myLeft + 1,
@myLeft + 2);

UNLOCK TABLES;
```

In this example we expand everything to the right of the left-hand number of our proud new parent node, then place the node to the right of the left-hand value. As

you can see, our new node is now properly nested:

```
SELECT CONCAT( REPEAT( ' ', (COUNT(parent.name) - 1) ), node.name) AS
name
FROM nested_category AS node,
        nested_category AS parent
WHERE node.lft BETWEEN parent.lft AND parent.rgt
GROUP BY node.name
ORDER BY node.lft;

+-----------------------+
| name                  |
+-----------------------+
| ELECTRONICS           |
|  TELEVISIONS          |
|   TUBE                |
|   LCD                 |
|   PLASMA              |
|  GAME CONSOLES        |
|  PORTABLE ELECTRONICS |
|   MP3 PLAYERS         |
|    FLASH              |
|   CD PLAYERS          |
|   2 WAY RADIOS        |
|    FRS                |
+-----------------------+
```

DELETING NODES

The last basic task involved in working with nested sets is the removal of nodes. The course of action you take when deleting a node depends on the node's position in the hierarchy; deleting leaf nodes is easier than deleting nodes with children because we have to handle the orphaned nodes.

When deleting a leaf node, the process if just the opposite of adding a new node, we delete the node and its width from every node to its right:

```
LOCK TABLE nested_category WRITE;

SELECT @myLeft := lft, @myRight := rgt, @myWidth := rgt - lft + 1
```

```
FROM nested_category
WHERE name = 'GAME CONSOLES';

DELETE FROM nested_category WHERE lft BETWEEN @myLeft AND @myRight;

UPDATE nested_category SET rgt = rgt - @myWidth WHERE rgt > @myRight;
UPDATE nested_category SET lft = lft - @myWidth WHERE lft > @myRight;

UNLOCK TABLES;
```

And once again, we execute our indented tree query to confirm that our node has been deleted without corrupting the hierarchy:

```
SELECT CONCAT( REPEAT( ' ', (COUNT(parent.name) - 1) ), node.name) AS
name
FROM nested_category AS node,
        nested_category AS parent
WHERE node.lft BETWEEN parent.lft AND parent.rgt
GROUP BY node.name
ORDER BY node.lft;

+-----------------------+
| name                  |
+-----------------------+
| ELECTRONICS           |
|  TELEVISIONS          |
|   TUBE                |
|   LCD                 |
|   PLASMA              |
|  PORTABLE ELECTRONICS |
|   MP3 PLAYERS         |
|    FLASH              |
|   CD PLAYERS          |
|   2 WAY RADIOS        |
|    FRS                |
+-----------------------+
```

This approach works equally well to delete a node and all its children:

```
LOCK TABLE nested_category WRITE;

SELECT @myLeft := lft, @myRight := rgt, @myWidth := rgt - lft + 1
```

```
FROM nested_category
WHERE name = 'MP3 PLAYERS';

DELETE FROM nested_category WHERE lft BETWEEN @myLeft AND @myRight;

UPDATE nested_category SET rgt = rgt - @myWidth WHERE rgt > @myRight;
UPDATE nested_category SET lft = lft - @myWidth WHERE lft > @myRight;

UNLOCK TABLES;
```

And once again, we query to see that we have successfully deleted an entire sub-tree:

```
SELECT CONCAT( REPEAT( ' ', (COUNT(parent.name) - 1) ), node.name) AS
name
FROM nested_category AS node,
        nested_category AS parent
WHERE node.lft BETWEEN parent.lft AND parent.rgt
GROUP BY node.name
ORDER BY node.lft;

+-----------------------+
| name                  |
+-----------------------+
| ELECTRONICS           |
|  TELEVISIONS          |
|   TUBE                |
|   LCD                 |
|   PLASMA              |
|  PORTABLE ELECTRONICS |
|   CD PLAYERS          |
|   2 WAY RADIOS        |
|    FRS                |
+-----------------------+
```

The other scenario we have to deal with is the deletion of a parent node but not the children. In some cases you may wish to just change the name to a placeholder until a replacement is presented, such as when a supervisor is fired. In other cases, the child nodes should all be moved up to the level of the deleted parent:

```
LOCK TABLE nested_category WRITE;

SELECT @myLeft := lft, @myRight := rgt, @myWidth := rgt - lft + 1
FROM nested_category
WHERE name = 'PORTABLE ELECTRONICS';

DELETE FROM nested_category WHERE lft = @myLeft;

UPDATE nested_category SET rgt = rgt - 1, lft = lft - 1 WHERE lft BETWEEN
@myLeft AND @myRight;
UPDATE nested_category SET rgt = rgt - 2 WHERE rgt > @myRight;
UPDATE nested_category SET lft = lft - 2 WHERE lft > @myRight;

UNLOCK TABLES;
```

In this case we subtract two from all elements to the right of the node (since without children it would have a width of two), and one from the nodes that are its children (to close the gap created by the loss of the parent's left value). Once again, we can confirm our elements have been promoted:

```
SELECT CONCAT( REPEAT( ' ', (COUNT(parent.name) - 1) ), node.name) AS
name
FROM nested_category AS node,
        nested_category AS parent
WHERE node.lft BETWEEN parent.lft AND parent.rgt
GROUP BY node.name
ORDER BY node.lft;

+---------------+
| name          |
+---------------+
| ELECTRONICS   |
|  TELEVISIONS  |
|   TUBE        |
|   LCD         |
|   PLASMA      |
|  CD PLAYERS   |
|  2 WAY RADIOS |
|   FRS         |
+---------------+
```

Other scenarios when deleting nodes would include promoting one of the children to the parent position and moving the child nodes under a sibling of the parent node, but for the sake of space these scenarios will not be covered in this article.

Final Thoughts

While I hope the information within this article will be of use to you, the concept of nested sets in SQL has been around for over a decade, and there is a lot of additional information available in books and on the Internet. In my opinion the most comprehensive source of information on managing hierarchical information is a book called Joe Celko's Trees and Hierarchies in SQL for Smarties, written by a very respected author in the field of advanced SQL, Joe Celko. Joe Celko is often credited with the nested sets model and is by far the most prolific author on the subject. I have found Celko's book to be an invaluable resource in my own studies and highly recommend it. The book covers advanced topics which I have not covered in this article, and provides additional methods for managing hierarchical data in addition to the Adjacency List and Nested Set models.

In the References / Resources section that follows I have listed some web resources that may be of use in your research of managing hierarchal data, including a pair of PHP related resources that include pre-built PHP libraries for handling nested sets in MySQL. Those of you who currently use the adjacency list model and would like to experiment with the nested set model will find sample code for converting between the two in the Storing Hierarchical Data in a Database resource listed below.

References / Resources

- Joe Celko's Trees and Hierarchies in SQL for Smarties – ISBN 1-55860-920-2
- Storing Hierarchical Data in a Database
- A Look at SQL Trees
- SQL Lessons
- Nontraditional Databases
- Trees in SQL
- Trees in SQL (2)

- Converting an adjacency list model to a nested set model
- Nested Sets in PHP Demonstration (German)
- A Nested Set Library in PHP

109 thoughts on “Managing Hierarchical Data in MySQL”

Doug Meyer

2012/04/24 at 4:39 AM

Very nice and concise article, Mike. I bought Joe Celko’s book that you referenced and have skimmed, then read it. Your article helps me to understand the Nested Set model. I’m an old dog who has been writing adjacency list hierarchies since the early nineties. I always thought there could be ‘better way’, and this is certainly it. Thanks for taking the time to explain it all, Mike.

Chris M

2012/04/27 at 9:44 AM

Could this be made to work for hierarchies where a child can have more than one parent?

Mike Hillyer

2012/04/27 at 9:47 AM

At that point you’re dealing with a graph instead of a hierarchy. If you’re using MySQL you may want to look at the following storage engine by an associate of mine: http://openquery.com/products/graph-engine

Daniel

2012/05/20 at 3:17 PM

Hi Mike;

thanks for your well described tutorial. I noticed you do a “GROUP BY node.name” in quite a few example queries here. This is very bad practice and will return some unexpected results as soon as you have a few hundred records where node.name is not unique anymore.
You should group your rows by ID instead.

Mukesh

2012/06/01 at 11:42 PM

I am a beginner php programmer and want to develop a mlm web application in php & mysql,but I have no idea about it.
My mlm webapplication’s requirement is-
1.It work on binary tree in which every node only have left and right child.
2.In this we can add node as our wish which be add in left or right.
So how can implement it?

Ramakant

2012/06/05 at 5:26 AM

Very useful link to see example of self joins
Here you can select user and parent from same table
http://www.devshed.com/c/a/MySQL/MySQL-Table-Joins/4/

what i use is here:

select distinct e.employeeid,**e.givenname**,e.contact_no,**b.givenname as parent_name**,e.status from employee as e ,employee as b, agent_details as a,user_group as u
where a.agent_id = e.employeeid AND a.parent_id=b.employeeid AND e.employeeid=u.employeeid AND u.groupid=10 GROUP BY e.employeeid ORDER BY e.givenname

Bartek

2012/06/07 at 12:03 AM

I use this data model for years. I first found it on dev.mysql.com few years back. I have structures many levels deep and data with hundreds of rows – It never fails, it's really fast. The ability to cut part of the structure with one question is simply awesome! It is part of my framework now with full credit to Mike of course 🙂
Thank you so much for sharing.

ashwini

2013/02/06 at 2:41 AM

> How do you manage inserts? I beleive you would have a programmatic way of doing inserts. Most online resources on this subject start from "pen-paper", as in, number nodes/elements by hand and then fire individual inserts. Is there a resource that takes in a tree/xml and generates the numbering?

Leo

2012/06/14 at 7:46 PM

I like your code for "Retrieving a Single Path" in a "Nested Set Model".

However, how would you retrieve a single path if there were multiple nodes with the same name? For example, what if Flash was listed both under MP3 Players and under Movie Characters, and you wanted to retrieve the path that led to the movie character only?

Clay Stuckey

2012/06/14 at 9:15 PM

Your adjacency model is super close to what I need. Is it possible to perform the "retrieve a full tree" query and have the number of "children" be dynamic. Your

example will work great if you have =1 levels.

 **Martin Andreev**

2012/07/04 at 1:33 PM

Awesome tutorial. I am really impressed. Its something that i didn't know and i'll gladly use it 🙂 Thank your for taking you time to write it.

 **Alejandro Palacios**

2012/07/10 at 10:38 PM

Very nice explanation of the adjacent list and nested set models,

thanks a lot !

 **anand**

2012/07/25 at 8:01 AM

really very help full for developing my project
super like

Thanks

 **Chris**

2012/07/30 at 12:54 AM

In your demonstration of inserting a row between 'Televisions' and 'Portable Electronics', what if the table was extremely large, perhaps hundreds of thousands of rows, and the left insert point were very close to zero? Unlike an adjency list, such an insert into a nested set will trigger massive updates.

It seems to me that a good extension to the model might be to reserve space, perhaps depth dependent, so insertions trigger less expansive updates.

Rasmus Schultz

2012/08/02 at 11:26 AM

Very useful article – I had to do some digging, as this article recently vanished from mysql.com for some reason. The formatting of this article on your blog is somewhat broken – it would be nice if you could clean this up so it's legible again. Thanks!

Pingback: MySQL structure of website with parent and child topics

---

Andy Taylor

2012/08/05 at 3:52 PM

Firstly, huge thanks for a great & clear explanation.
I implemented some of your code examples against a database I'd generated of a hard disk directory structure and got some strange results.
In the examples you've given, you're using 'GROUP BY node.name'. However, if I understand things correctly, this will only work if 'node.name' is unique for all entries.
If, as I discovered, you have multiple entries with the same 'node.name' then doing 'GROUP BY node.category_id' seems to work.

Pingback: How to create threaded comment system using PHP, MySQL, jQuery and Ajax

---

Marcus Ahle

2012/08/10 at 9:39 AM

How big of strain does this place on the Db when the amount of categories gets large? For example, say you have 3 main categories, Electrnics, Clothing, and groceries. Say each of those has 200 total nodes underneath each. When adding a

new node to the very beginning seems like it could place a big strain in terms of having to update the lft and rgt of hundreds of nodes. At what point does size hinder performance?

Pingback: Trees and Hierarchical Data in mySQL using the nested sets model « Alin's DevBlog

---

Zeeshan

2012/09/15 at 4:54 AM

+1 to you Mike, I had the same table schema but implemented the retrieval stuff in PHP code which was a bit slow (I used it for categories on a retail store portal). I was looking for an idea about hierarchical retrieval via SQL queries, and you gave me a great food for thought.

Ercan Mutlu

2012/09/28 at 12:01 AM

Hi,Nice article BUT ON Adjacency List Model Getting CAt->SUbs YOU HAVE WRONG QUERY OR NOT PROPER QUERY, I CANT LIST
LIKE THIS :
Missing cats-subs
a-> (I need this id)
a->b (Also this id)

Your Cat-subs
a->b->c (NO directly THIS ID)
a->b->c->d
...
YOUR QUERY PROBLEM IS :
CORRECT QUERY MUST BE
————————
ELECTRONICS | NULL |NULL | NULL |

ELECTRONICS | TELEVISIONS |NULL | NULL |
ELECTRONICS | TELEVISIONS |TUBE | NULL |

YOURQUERY
———————————————
ELECTRONICS	TELEVISIONS	TUBE	NULL
ELECTRONICS	TELEVISIONS	LCD	NULL
ELECTRONICS	TELEVISIONS	PLASMA	NULL

AND the Question IS HOR TO GET LIKE THIS ?

Tim

2012/11/27 at 5:30 PM

Great article. Really helpful. Quick question though. Let's say I want to get the entire tree but the depth be an offset from the current queried node. For example (querying for Portable Electronics):

Electronics = -1
Portable Electronics = 0
MP# Players = 1
CD Player = 1
2 Way Radio = 1
Flash = 2

Marco Demaio

2012/12/16 at 10:25 AM

Great article, I looked for it on dev.mysql but it's not ther anymore.

One question on Adjacency List Model:

Let's say you have in table also `position` INT field, that let's you sort nodes that are **on the same level**, in your example let's say I want "LCD" to come before "TUBE", or I want "PORTABLE ELECTRONICS" to come before "TELEVISION".

Is it still possible to retrieve the full tree with one single query as in your example, but with the nodes already ordered also by position?

Cause I think adding just `ORDER BY position` at the end of the query would not be a solution.

ivanr

2012/12/20 at 4:10 PM

Great article Mike! I have a menu structured in the adjacency list model and I needed to retrieve a single path where the last node could be at any level. I ended up writing a stored procedure that would return one string the different nodes separated by semi-colon and have PHP parse that string.

The code is something like this, bear in mind I'm not the best coder and I'm sure it can be optimized:

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `GetMenuPath`(IN `myNode` VARCHA
    NO SQL
BEGIN

DECLARE myParent INT;
DECLARE myNewParent INT;
DECLARE myNewPath VARCHAR(200);

SET menuPath = myNode;
SELECT menu_parent INTO myParent FROM menu WHERE menu_item = myNode;

while myParent &gt; 0 do  /* Not zero or null */
        SELECT concat(menuPath, ';', menu_item) INTO myNewPath FROM menu WHE
        SET menuPath = myNewPath;
        SELECT menu_parent INTO myNewParent FROM menu WHERE menu_id = myPare
        SET myParent = myNewParent;
END while;

END
```

Stefan Stavrev

2012/12/26 at 3:27 AM

If I may point one possible improvement over your model. When you do your SELECT and GROUP BY statements, to use the ID instead of the name. Imagine you have child nodes with the same name, but different parents.

Choerun Asnawi

2012/12/26 at 9:29 PM

Good article! In addition, use this query to simulate the “old-way” adjacency list model table structure:

```
SELECT child.category_id AS category_id,
  child.name AS name,
  parent.category_id AS parent
FROM nested_category AS child
  LEFT JOIN nested_category AS parent
    ON parent.lft = (
      SELECT MAX(anchestor.lft)
      FROM nested_category AS node,
        nested_category AS anchestor
      WHERE anchestor.lft  node.rgt
        AND node.category_id = child.category_id
    )
  ORDER BY child.lft;
```

victor wallin

2013/01/07 at 4:43 AM

1) It’s okay to have credits hole in the series?
name	lft	rgt
ELECTRONICS	1	8
TELEVISION	2	3
PORTABLE ELECTRONICS	6	7

2) is this a good idea?
Name	lft	rgt	parent_lft	parentrgt
ELECTRONICS	1	8	NULL	NULL
TELEVISION	2	3	1	8
PORTABLE ELECTRONICS	6	7	1	8

3) what is the easiest way to convert a tree to a Nested Set?

shahzad

2014/01/18 at 3:47 AM

I have a category structure up to 5 level
Music -> Pop -> Michael Jackson -> Thriller -> Thriller – Song
Suppose i want to search the category with like query '%s%'.

Now s is in 3 level (Level1 = Music),(Level3 = Michael Jackson),(Level5 = Thriller – Song).

Now its give me 3 results instead of 1. How i will get 1 results?
If i found the result at first level(levle1), it will not search at second level (level2).
if i found the result at Second level(levle2), it will not search at Third level (level3) and so on.

Have you idea?

Oguzcan

2014/01/05 at 4:46 PM

Thank you in advance for this article. Could you show how to add new root nodes ?

Gerry

2016/01/27 at 5:31 PM

Heck yeah this is excltay what I needed.

Quick Correction

2014/01/29 at 11:43 AM

In the ordering for the nested set, I believe it should be
'ORDER BY parent.lft' as opposed to 'node.lft'.

If we look at the Retrieving a Single Path section, it does not make sense to order by node.lft, since node.lft is all the same tuple (we've queried for a node.lft='FLASH').

If we had a "SELECT parent.name, node.name ..." it would like like the following:

name | name
————————————————————
MP3 PLAYERS | FLASH
FLASH | FLASH
ELECTRONICS | FLASH
PORTABLE ELECTRONICS | FLASH

Pingback: mySQL | leaf nodes | SQL snippets

Heinz

2014/04/02 at 7:23 AM

Hello
I have set up my page with nested tree. But the biggest problem is still when I add a new category/subcategory. Afterwards you have to renumber all articles in your database because most of them got a new category number. If you have 500'000 acticle this takes long and all your references change too.
I wonder how ebay is doing this? You can't stop trading while updating.

Mike Hillyer

2014/04/02 at 7:31 AM

Hello Heinz, look at Trees and Hierarchies in SQL for Smarties to get a lot of advanced tips. One I can share here is to look at using index numbers that increment by 1,000 instead of by one, when you need to add a node you place it

between two existing index numbers, then occasionally during slow periods you re-index everything.

Mohsen Nekoulal Tak

2014/04/05 at 9:27 AM

very nice article. But now i have a question. how can i receive directly a specified depth of a optional parent? For example, i choose PORTABLE ELECTRONICS as the parent and i want to get the depth:2 which will be resulted the FLASH.
Can u help me?

Misha Dar

2014/11/04 at 12:30 PM

First compliments on article – it's really straight & forward explaining approach concepts of building hierarchical model in RDBS.
But, with all respect you can not take full advantage of using it in real world in MySQL DB engine without applied knowledge in structuring and using Spatial Data Type extension... cause using default (BTREE) index to perform nested set join like

FROM nested_table AS node, nested_category AS parent

will chock a query on performance in mega volume sets.

So in MySQL, LFT RGT columns define as geometrical data type is highly recommended and so self cartesian product join will look something like:

FROM nested_table AS node
JOIN nested_table AS AS parent ON (MBRWithin(Point(0, node.lft), parent.lft_rgt_range))

or close to it 🙂
Cheers

Peter Salzmann

2016/10/15 at 11:20 PM

Misha is correct.

We've built an Enterprise marketplace product and converted from Adj. to Nested. It's incredible how fast result sets are accomplished with the proper query. While the queries in this article work they are definitely not large-category + huge product volume set ready.. everything works great with < 1,000 products and/or < 10,000 categories.

What's missing from this article IMO is the spatial aspect on the table joins for the categories. For example:

products table contains 1 million products
categories table contains 28,301 categories.

How we accomplish the finishing touches to make it production ready was the following:

```
SELECT ...
FROM category AS node
JOIN category AS parent ON (MBRWithin(Point(0, node.lft), parent.sets))
.....
```

Without it a category with 11,000 items results in 18 seconds delay. With the ON clause to the parent table join reduced it to < 0.1 second. It's blazing fast. We've also added necessary single and combined indexes which also help I'm sure.

You'll need (if your category table is named category):

```
ALTER TABLE category ADD `sets` LINESTRING NOT NULL
UPDATE category SET `sets` = LineString(Point(-1, lft), Point(1, rgt))
CREATE SPATIAL INDEX sx_categories_sets on category (sets)
```

If you do your queries a different way, we do it 2 different ways; you can try this one as well:

SELECT ...
FROM category AS parent
JOIN category AS midpoint
JOIN category AS node
....
AND (MBRWithin(Point(0, node.lft), parent.sets))

Let me know if it helps!
Cheers!

Pingback: Hierarcy Query Mysql - DL-UAT

Pingback: PHP 4 Switch

Pingback: Stored Procedure to update an adjacency Model to Nested Sets Model | DL-UAT

Pingback: Solution: How to determine the first and last iteration in a foreach loop? #solution #programming #it | IT Info

---

Pingback: How to merge two tables for a query? | FYTRO SPORTS

Adam

2015/01/29 at 11:45 PM

Hello friends. I've just switched from Adjacency List Model to nested set model.

I'think i'm starting to understand what is going on, but i have big problem – i cannot draw my tree as html menu. i found some sources/examples (http://stackoverflow.com/questions/1310649/getting-a-modified-preorder-tree-traversal-model-nested-set-into-a-ul) on the net, but it's a little hard to

me to port them to work with Your examples. Can anybody help me with that? Thank you.

spratters

2015/02/04 at 4:28 AM

Hey there,
Just wanted to thank you for the article. I was working out ways to best get results from a nested set and my mind was turning to mush. Luckily I had a read through this article and all became clear.
Cheers

Pingback: Getting the depth of a nested set model query | 我爱源码网

Pingback: Count of items by depth with a nested set model - Technology

---

Pingback: Select products where the category belongs to any category in the hierarchy | XL-UAT

Pingback: What are the Options for Storing Hierarchical Data in a Relational Database? - Technology

---

Pingback: MySql Hierarchical Database in Java Web App Jquery Tree | 我爱源码网

Adam

2015/04/18 at 9:35 PM

Hi, I've implemented nsm in my project, thanks to this article.
Now I'm trying to get a simple method of showing all (lets say) products) of given

parent category (which can contain many subcategories). Is there any chance to somebody give any tip on how to acheive this?

best regards
Adam

Martin F

2015/04/22 at 4:07 PM

Under Retrieving a Single Path, you can/should leave out (excuse the pun) the
“t1.name = ‘ELECTRONICS’ AND”
part of the WHERE clause. The query will find the path to the root.

Pingback: MySQL 5.6.4 mit neuen Performance-Schema-Eigenschaften - entwickler.de

---

Bruno Lebtag

2015/05/02 at 1:00 PM

I tried Retriving a single path and your query didn’t worked. I changed ‘ORDER BY node.lft’ to ‘ORDER BY parent.lft’ and worked. My mysql is 5.6.19-1~exp1ubuntu2. Excellent work! thanks a lot.

Pingback: Nested set model: retrieve sub-tree containing nodes that match condition | DL-UAT

tyan4g

2015/06/16 at 3:33 AM

Finding the Depth of the Nodes:
SELECT node.name, (COUNT(parent.name) – 1) AS depth
FROM nested_category AS node,

nested_category AS parent
WHERE node.lft BETWEEN parent.lft AND parent.rgt
GROUP BY node.name
ORDER BY node.lft;
I find this is problematic, cause if I add two item with the same name will cause the depth got a wrong result..plz help me..

 **Andreas Beder**

2015/06/23 at 8:06 AM

Thanks !!!

Pingback: Listing of all elements recursively through postgresql - dex page

---

Pingback: Adjacency List Model vs Nested Set Model for MySQL hierarchical data? - HTML CODE

Pingback: Laravel Query builder returns empty array despite it being not null - HTML CODE

Pingback: Multi level hierarchy relation in flat table design (MySql)

 **David Kaštánek**

2016/01/17 at 6:20 AM

Awesome idea! I knew this method existed and after reading this I also know how to implement it. Thank you very much 🙂

 **e2xist**

2016/01/28 at 2:46 PM

thanks a lot !

Pingback: Options for Storing Hierarchical Data in a Relational Database | 柠檬树博客

ummmmm

2016/02/09 at 4:09 PM

Finding the Depth of the Nodes,

select
node.id as id,
node.lft as lft,node.rgt as rgt,
parent.id as top_parent_id,
(count(parent.id)-1) as depth
FROM
nested_category as parent_ctl,
nested_category as parent,
nested_category as node
where
parent.lft between parent_ctl.lft and parent_ctl.rgt
and node.lft between parent.lft and parent.rgt
and parent_ctl.id = ?
group by node.id
order by node.lft

* title -> id

Pingback: Designing a threaded commenting system [ANSWERED] - Tech ABC to XYZ

Wojciech Glapa

2016/02/23 at 4:41 AM

Very usefull, thanks.

Thomas

2016/03/01 at 7:26 AM

Man, you save our lives. Thanks so much! Nice job and was the best tutorial about this issue. Best regards

---

Sagar Khandelwal

2016/03/04 at 3:24 AM

Insert a new node
———————————-
LOCK TABLE nested_category WRITE;

SELECT @myRight := rgt FROM nested_category
WHERE name = 'TELEVISIONS';

UPDATE nested_category SET rgt = rgt + 2 WHERE rgt > @myRight;
UPDATE nested_category SET lft = lft + 2 WHERE lft > @myRight;

INSERT INTO nested_category(name, lft, rgt) VALUES('GAME CONSOLES',
@myRight + 1, @myRight + 2);

UNLOCK TABLES;
———————————-
This gives Game Console left=10 and right = 11, while right of Television remains 9. Resulting a corrupted tree.

Instead, you can select @myRight := rgt – 1, and rest of the query would work fine.

Regards

Zhong chaojun

2016/03/14 at 4:35 AM

In section RETRIEVING A SINGLE PATH, order by parent.lft not node.lft

---

Carl Sanders

2016/03/23 at 8:02 AM

Fantastic article Mike!

There is a further improvement to this model where you store the depth with each node. That makes querying for sub-trees and adjacent entities far simpler. In symphony there is a nested set library available for doctrine (which I have used) and there's also a php lib called Boabab (which I haven't).

Programmer

2016/04/11 at 7:06 AM

Great article.

How to move a node?

Vahid

2016/05/08 at 5:30 AM

Thanks for this awesome article. It shows the deep of concepts which author obtained. Bravo
There's a question i'm thinking about it:

How we can build a lft,rgt type (The Nested Set) tree with an Adjacency List tree which already we have?

imagine there's a table : id,parent_id,title with 500000 or more records.
How can convert this table?

Regards.

Vahid

2016/05/09 at 12:41 AM

I found this great soloution:
http://dba.stackexchange.com/questions/89051/stored-procedure-to-update-an-adjacency-model-to-nested-sets-model

---

Jeff

2016/05/16 at 9:21 AM

First thank you for the post, it's really helpful. I have a question, maybe a silly one, but in case I have two parents, for instance, following your reasoning, suppose we have Electronics and Home appliance as the topmost parents, so this model would work the same? Just adding another topmost category as NULL? Tks in advance.

Mike Hillyer

2016/05/16 at 9:49 AM

Yes, you will have an un-displayed parent, and the first two children are Electronics and Home.

Pingback: AngularJS: Rekursive Templates — Web und die Welt

Amit kumar

2016/06/09 at 3:31 AM

I want to make a single query that works multiple child level
and i want to this with single sql query

Thankyou.

Pingback: Managing Hierarchical data ... – cyberkrul

bhavin

2016/07/02 at 5:21 AM

Thank you so..much for this great blog. Awesome explanation. You are clear my concept of parent , child structure in 2 hours.

Pingback: MySQL: Tree | Codeba

Amit

2016/07/20 at 1:29 AM

Is it possible to use the nested set model with intersecting sets? For example, if there is a Portable Television, so it should be a shared child between Television and Portable Electronics. Can the nested set model be applied to such a case?

Mike Hillyer

2016/08/08 at 11:46 AM

Keep the data on the television in a separate table, and put the primary key of the television in the nested set under television and under portable electronics.

Pingback: Storing Hierarchical Data in a Relational Database | find24

Pingback: Managing Hierarchical Data in MySQL | Mike Hillyer's Personal Webspace – MUBASHSHIR ZAKIR

Reinaldo

2016/08/22 at 11:34 AM

I think it would be easy to make a hierarchical table if the DBMS implement a numeric data type as a software version, eg.: 1.3.7.25, so we could use this type to do a column "Hierarchy", where the first node would be the 1, her first child would be 1.1, his second child 1.2 , the first child of the first child would be 1.1.1, the second child of the first child would be 1.1.2 and so on. To insert correctly in the hierarchy just think in trigger to do it. To delete we could use the "Hierarchy" column to remove a parent and all its children.

PS.: It is just a initial idea.

Reinaldo

2016/08/23 at 7:06 AM

In addition; To get the resultset hierarchically just order by "hierarchy". Clean, easy and graceful.

paola

2016/09/01 at 4:15 PM

very good, realy helpful

thank you

Sergey

2016/09/09 at 8:24 AM

Really useful article, thank you so much!

Aman Singh

2016/11/03 at 5:06 AM

You can get calculated columns parent & depth with following query:
SELECT node.category_id, node.name,
node.lft, node.rgt,
(select parent.category_id from nested_category AS parent
WHERE node.lft BETWEEN parent.lft AND parent.rgt
ORDER BY parent.rgt offset 1 limit 1) AS parent,
(select (COUNT(parent.name) – 1) from nested_category AS parent
WHERE node.lft BETWEEN parent.lft AND parent.rgt) as depth
FROM nested_category AS node;

Siah

2016/12/27 at 12:46 PM

This is great article. I'm going develop an ORM that will handle nested sets and hide the complexities. Unfortunately most ORMs do no support nested sets and typing all the necessary SQL for managing nested sets in every project that needs this feature is a nightmare.

Pingback: The Nested Set Model – nate_the_dba

---

egoing

2017/01/26 at 10:07 PM

Thank you. Great article.

I think this is more easy to read. 🙂

```
SET @SUB_TREE_NAME = 'TELEVISIONS';
SET @LFT = (SELECT lft
FROM nested_category
WHERE name = @SUB_TREE_NAME);
SET @RGT = (SELECT RGT
FROM nested_category
WHERE name = @SUB_TREE_NAME);
SET @SUB_TREE_DEPTH = (
SELECT COUNT(node.category_id) – 1
FROM
nested_category AS node
INNER JOIN
nested_category AS parent
ON
node.lft BETWEEN parent.lft AND parent.rgt
WHERE node.name = @SUB_TREE_NAME
GROUP BY node.category_id
);
SELECT
*,
(COUNT(node.category_id) – @SUB_TREE_DEPTH – 1) AS depth
FROM nested_category AS node LEFT JOIN nested_category AS parent ON
node.lft BETWEEN parent.lft AND parent.rgt
WHERE node.lft BETWEEN @LFT AND @RGT
GROUP BY node.category_id
ORDER BY node.lft;
```

Pingback: Converting comma separated fields to MySQL JSON – a case study – Cloud Data Architect

---

Pingback: 域名更换通知 - 莹莹之色

Pingback: Tree structure in relational db's: theory and use in Symfony – Ekreative

---

Pingback: 如何创建一个MySQL分层递归查询 – CodingBlog

Pingback: 最佳树结构 – CodingBlog

Pingback: MySQL: Tree-Hierarchical query - 功夫 熊猫 - – 引力一族

Pingback: Деревья в БД — КОРПОРАТИВНЫЙ БЛОГ САТЭК

Anjum Rizwi

2017/09/20 at 1:05 AM

Very useful, as I heard first time about “lft” & “rgt”.
Very good detail articles especially diagram that cleared my doubt

---

Pingback: Options for Storing Hierarchical Data in a Relational Database - 程序旅途

Pingback: Is it possible to query a tree structure table in MySQL in a single query, to any depth? - ExceptionsHub

---

Pingback: Is it a good idea to use MySQL and Neo4j together? - ExceptionsHub

---

Pingback: php / Mysql best tree structure - QuestionFocus

Pingback: Build a tree from a flat array in PHP - QuestionFocus

---

Pingback: Managing Hierarchical Data in MySQL – Lets Code

Pingback: Reverse query for Hierarchical Data MS SQL Server

Pingback: Recursive SQL query on nested-set tree with stop condition – Mysql Questions

Comments are closed.