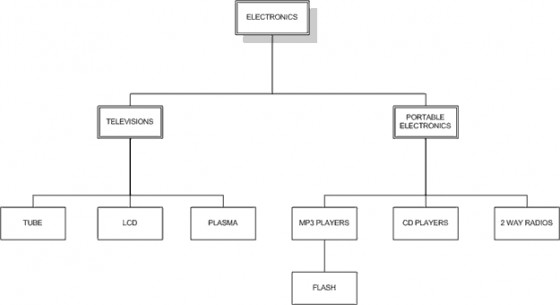
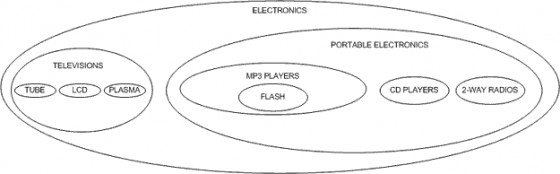
**预排序遍历树算法(非递归无限极算法)**

下面是一个典型的多层数据结构示意图：

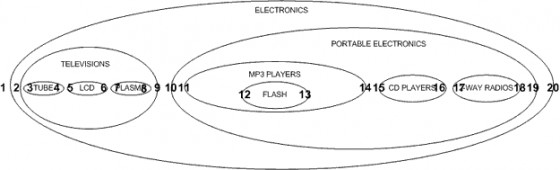


相关创建数据语句：  
CREATE TABLE category(  
category\_id INT AUTO\_INCREMENT PRIMARY KEY,  
name VARCHAR(20) NOT NULL,  
parent INT DEFAULT NULL);  
  
  
INSERT INTO category  
VALUES(1,'ELECTRONICS',NULL),(2,'TELEVISIONS',1),(3,'TUBE',2),  
(4,'LCD',2),(5,'PLASMA',2),(6,'PORTABLE ELECTRONICS',1),  
(7,'MP3 PLAYERS',6),(8,'FLASH',7),  
(9,'CD PLAYERS',6),(10,'2 WAY RADIOS',6);  
  
SELECT \* FROM category ORDER BY category\_id;  
  
在这种数据结构中，各层之间通过字段 parent 来形成邻接表，我们查询某些层级的关系的时候一般都是通过递归的方式，遍历某个层级关系的SQL的查询次数会顺着层级的增加，想想在层级有20的时候，根据某个底层节点取它到顶层节点的查询次数吧。  
  
为了解决这个问题，人们想出了嵌套集模型（The Nested Set Model），请看下图：

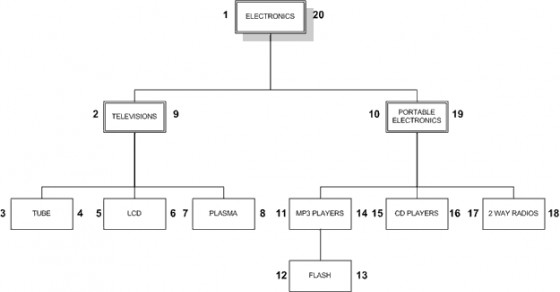


上图依然是表现的与图一相同的层级关系，但是却更换了一种表现形式 下面是新的关系表和数据（关系和数据与之前相同，但是表结构不一样）：

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;  
  
这里将 left,right 修改为 lft,rgt因为这两个词在MYSQL中属于关键字 下面我们将插入的数据标识在图上：



同样，我们将数据标识在原来的结构上：



怎么样，是不是很明确了  
  
下面使我自己标定一种形式，方便理解  
  
[1  
      [2  
           [3 4]   
           [5 6]   
           [7 8]  
      9]   
      [10  
           [11  
                 [12 13]  
           14]  
           [15 16]  
           [17 18]  
      19]  
20]  
  
**遍历整个树，查询子集 条件：左边 > 父级L， 右边 < 父级R**  
  
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         |  
+----------------------+  
  
**- 查询所有无分支的节点 条件：右边 = 左边L + 1**  
  
SELECT name  
FROM nested\_category  
WHERE rgt = lft + 1;  
  
**- 查询某个字节点到根节点的路径**  
  
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;  
  
  
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;  
  
**- 查询子节点的深度**  
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;

**- 插入新节点**  
算法详解：   
1.所有分类 左边和右边的值 > 插入节点的左边节点记录的右值 的全部 + 2  
2.插入节点 左值 = 插入位置左边节点记录的右值 + 1， 右值 = 插入位置左边节点记录的右值 + 2  
例子:  
在 R = 9（L8, R9）与 L = 10（L10，R11） 节点之间插入一个新节点  
那么所有 左值 和 右值 > 9 的节点的左值和右值需要 + 2  
例如新节点右边的节点（L10，R11）左值右值都需要 + 2 那么插入后的新值为 L12 R13  
新节点的左值为 9 + 1 = 10 右值为 9 + 2 = 11  
SQL语句实现  
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;  
  
**- 删除新节点**  
删除节点的算法与添加一个节点的算法相反  
  
删除一个没有子节点的节点  
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;  
  
删除一个分支节点和它所有的子节点  
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;

删除一个节点后移动其字节点到  
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;

**总结：**

预排序遍历树算法的核心就是牺牲了写的性能来换取读取的性能

在你的开发的应用遇到此类问题的时（读压力 > 写压力），尝试下使用预排序遍历树算法来提高你的程序的性能吧。