NAVER 카페

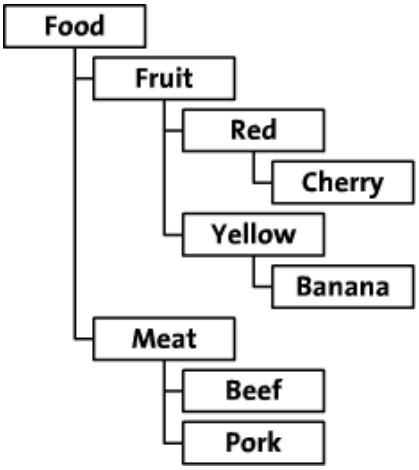
계층적 구조를 데이터베이스에 저장하기(2) | 종합 데이타베이스 출력하기 | 닫기

사모님(meetkit)

당 신이 게시판이나 메일링리스트 같은 것을 웹사이트에 게시하거나 CMS를 개발하고자 할때, 계층적 데이터 구조를 데이터베이스에 저장할 필요성이 발생 할 때가 있다. 그리고, XML기반 데이터베이스를 사용하지 않는 이상 테이블은 계층적이지 않으며 단지 평면적일 뿐이다. 당신은 계층적인 구조를 평면파일로 번역할 수 있는 방법을 찾아야만 한다.

트리(tree)를 저장하는 것은 공통된 관심사이며 여러가지 해법이 존재한다. 가장 주요한 두가지 접근법은 근접 리스트 모델(adjacency list model)과 수정된 순서화된 트리(modified preorder tree) 검색 알고리즘이다.

이 글에서는 계층적 데이터 저장의 이 두가지 방법을 설명할 것이다. 트리는 온라인 식료품점을 예로들어 설명할 것이다. 이 식료품점은 식품들을 카테고리별, 색상과 타입별로 정돈한다. 이 트리는 아래 그림과 같이 보일것이다.



이 글에서는 데이터를 저장하고 조회하는 코드 예제들을 볼 수 있다. 비록 예제코드는 PHP로 작성되었지만 당신이 사용하는 언어로도 쉽게 변경이 가능 할 것이다.

근접 리스트 모델 (The Adjacency List Model)
먼저, "근접 리스트 모델" 또는 "재귀호출 방법"이라 부르는 접근법에 대해 살펴 보겠다. 이것은 당신은 트리를 탐색하는 단순한 한가지 기능만을 필요로 하므로 가장 고상한 접근법이다.

```
+--------+--------+
| parent | title  |
+--------+--------+
|        | Food   |
| Food   | Fruit  |
| Fruit  | Green  |
| Green  | Pear   |
| Fruit  | Red    |
| Red    | Cherry |
| Fruit  | Yellow |
| Yellow | Banana |
| Food   | Meat   |
| Meat   | Beef   |
| Meat   | Pork   |
+--------+--------+
```

위 그림에서 보듯이, 근접 리스트 방법에서는 각 노드가 "parent"를 가지고 있다. 우리는 "Pear"가 "Green"의 자식(child)임을 알 수 있다. 또 "Green"은 "Fruit"의 자식임을 또한 알 수 있다. 최상위 노드인 "Food"는 더이상 부모값을 가지지 않음을 아 수 있다. 나은 "title"을 각 노드의 구분값을로 사용했다. 물론, 실제 데이터베이스 에서는 숫자형태의 ID값을 사용할 것이다.

트리(Give Me the Tree)
이제 우리의 트리를 데이터베이스에 삽입하였다. 이제 보여주는 함수를 작성할 시간이다. 이 함수는 최상위(root)노드-부모(parent)를 가지지 않는 노드-에서 출발해야만 한다. 그리고, 모든 자식노드를 출력할 것이다. 모든 자식들은 함수에 의해 그 자식노드들 까지 모두 탐색될 것이다.

당신은 이 함수에 대한 설명에서 이것이 정규화된 패턴이 있음을 눈치챘을 것이다. 우리는 단순히 하나의 함수만을 작성할 것이며 이것은 특정 부모에 대한 자식노드를 검색할 것이다. 이 함수는 다른 인스턴스로 다시 그들의 자식들을 모두 출력하기 위해 시작될 것이다. 이것을

재귀적인 메커니즘으로 "재귀호출 방법(recurtion method)"이라 부른다.

```
<?php
// $parent is the parent of the children we want to see
// $level is increased when we go deeper into the tree,
// used to display a nice indented tree
function display_children($parent, $level) {
// retrieve all children of $parent
$result = mysql_query('SELECT title FROM tree '.
'WHERE parent="'.$parent.'";');

// display each child
while ($row = mysql_fetch_array($result)) {
// indent and display the title of this child
echo str_repeat(' ',$level).$row['title']."₩n";

// call this function again to display this
// child's children
display_children($row['title'], $level+1);
}
}
?>
```

모든 트리를 보여주기 위해 우리른 $parent에 빈공백과 $level = 0을 사용할 것이다. :

```
display_children('',0);
```

이 함수는 식료품점 트리를 아래와 같이 리턴할 것이다.

```
Food
  Fruit
   Red
     Cherry
   Yellow
     Banana

  Meat
   Beef
   Pork
```

만약, 특정 서브트리만 보고싶다면 함수에 다른 노드명을 적어주면 된다. 예를들어, "Fruit"의 서브트리를 출력하려면
display_children('Fruit',0); 과 같이 호출하면 된다.

노드의 경로
위 와 거의 비슷한 코드로 노드의 이름 또는 ID만으로 노드의 경로를 찾는것도 가능하다. 예를들어 "Cherry"의 경로는 "Food" > "Fruit" > "Red" 이다. 이 경로를 얻으려면 우리의 함수는 가장 깊은 레벨:"Cherry" 에서 시작해야만 한다. 이것은 현재노드의 부모를 찾고 경로에 추가시킨다. "Cherry"의 경우 "Red"가 된다. 우리가 "Cherry"의 부모가 "Red"라는것을 알고 있다면 우리는 "Red"를 이용해 "Cherry"의 경로를 계산할 수 있다. 그리고, 이것을 함수에 전달ㅎ해 재귀호출 함으로써 트리내의 어떤 노드든지 경로를 알아 낼 수 있다.

```
<?php
// $node is the name of the node we want the path of
function get_path($node) {
// look up the parent of this node
$result = mysql_query('SELECT parent FROM tree '.
'WHERE title="'.$node.'";');
$row = mysql_fetch_array($result);

// save the path in this array [5]
$path = array();

// only continue if this $node isn't the root node
// (that's the node with no parent)
if ($row['parent']!='') {
// the last part of the path to $node, is the name
```

```
// of the parent of $node
$path[] = $row['parent'];

// we should add the path to the parent of this node
// to the path
$path = array_merge(get_path($row['parent']), $path);
}

// return the path
return $path;
}
?>
```

이 함수는 주어진 노드의 경로를 리턴한다. 이것을 경로를 배열(array)형으로 리턴하므로 print_r(get_path('Cherry')); 명령어로 출력할 수 있다. :Cherry는 아래와 같다.

```
Array
(
[0] => Food
[1] => Fruit
[2] => Red
)
```

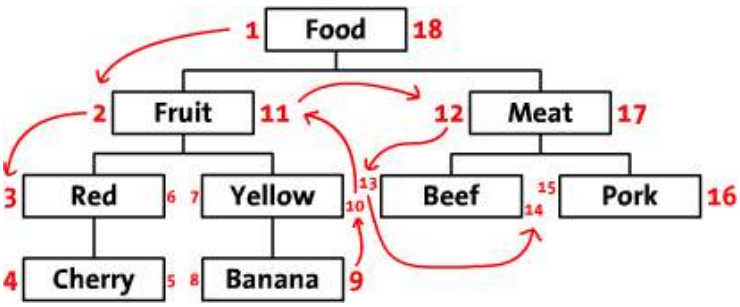
단점
이 것은 좋은 방법처럼 보인다. 이것은 이해하기 쉬고 코드도 단순하다. 근접 리스트 모델의 단점이 무엇일까? 대부분의 프로그래밍 언어에서 이런 방식은 느리며 비효율적이다. 주된 원인은 재귀호출에 있다. 우리는 트리내의 각 노드를 위한 데이터베이스쿼리가 필요하다.

두번째 이유는 당신이 사용하는 거의 모든 언어에서 이 방법은 빠르지 않다는 것이다. Lisp와 같은 언어를 제외하고 대다수는 재귀호출 함수를 고려해 설계되지 않았다. 트리가 네단계의 레벨을 갖는 경우 동시에 함수의 네개 인스턴스가 동시에 실행될 것이다. 각 함수는 메모리 조각을 사용하며 일정부분 초기화시간을 사용할 것이다. 이런 재귀호출은 트리가 클경우 매우 느려진다.

수정된 순서화된 트리 검색(Modified Preorder Tree Traversal)

이 번에는, 트리를 저장하는 다른 방법에 대해 살펴 보겠다. 재귀호출은 느려질 수 있으므로, 이번에는 재귀적인 방법은 사용하지 않기로 하겠다. 우리는 데이터베이스 쿼리를 최소화할 것이다. 되도록이면 각 액티비티에 한번의 쿼리를 사용할 것이다.

우 리는 트리를 가로접근법으로 시작할 것이다. 최상위(root)노드에서 시작해 왼쪽에 1을 쓴다. 그리고, Fruit의 왼쪽에 2를 쓴다. 이 방법으로 각 노드의 좌,우측에 숫자를 써내려간다. 맨 마지막 번호는 "Food"노드의 맨 오른쪽에 쓰여질 것이다. 아래 그림에서 전체 숫자와 순서에 따른 화살표를 확인 할 수 있다.



보 시다시피, 이 숫자들은 각 노드간의 연관을 가리킨다. "Red"노드는 숫자 3과 6을 가지고 있다. 그러므로, 숫자 1과 18을 가진 "Food"노드의 자손이다. 같은 방법으로 노드의 왼쪽값이 "2"보다 크고 오른쪽 값이 "11"보다 작은 노드는 "Fruit" 노드의 자손이라고 할 수 있다. 트리구조는 이제 왼쪽값과 오른쪽값으로 저장될 수 있다. 이런 방법으로 트리를 조회하고 노드를 세는것을 "수정된 순서화 트리 검색(modified preorder tree traversal)" 알고리즘이라 부른다.

계속 진행하기 전에, 이 값들이 테이블에 어떻게 저장되는지 보자.

parent	title	lft	rgt
	Food	1	18
Food	Fruit	2	11
Fruit	Red	3	6
Red	Cherry	4	5
Fruit	Yellow	7	10
Yellow	Banana	8	9
Food	Meat	12	17
Meat	Beef	13	14
Meat	Pork	15	16

데 이터베이스에서 "left"와 "right"는 특별한 의미를 가지므로 "lft", "rgt"로 컬럼명을 구분했다. 또한, 이제 더이상 "parent" 컬럼이 필요없음도 함께 주목하자. 이제 트리 구조를 구현하기 위한 lft 와 rgt 값을 저장했다.

트리 조회하기
왼쪽값과 오른쪽값으로 트리를 검색하려면, 먼저 조회하고자 하는 상위 노드부터 알아야 한다. 예를들어 "Fruit"서브트리의 경우, 당신은 왼쪽값이 2와 11사이인 노드들만 가져와야 한다.

SELECT * from tree WHERE lft BETWEEN 2 AND 11;

이것은 다음을 리턴할 것이다.

parent	title	lft	rgt
Food	Fruit	2	11
Fruit	Red	3	6
Red	Cherry	4	5
Fruit	Yellow	7	10
Yellow	Banana	8	9

모 든 트리를 한번의 쿼리로 가져왔다. 만약, 테이블에서 행을 추가하거나 삭제했다면 테이블이 올바른 순서로 정렬되어있지 않을것이므로 이 트리를 앞서했던 재귀함수와 같이 출력하려면 ORDER BY절을 쿼리에 추가해야 한다. 그러므로, 왼쪽값을 기준으로 정렬하도록 한다.

SELECT * FROM tree WHERE lft BETWEEN 2 AND 11 ORDER BY lft ASC;

이제 남은 문제는 왼쪽에 들여쓰기를 하는것 뿐이다.

트 리와 같이 보이기 위해서, 자식은 그의 부모보다 더 들여쓰면 된다. 이것은 오른쪽값의 스택을 유지하면 가능하다. 노드의 자식에 다다를때 마다 노드의 오른쪽 값을 스택에 추가한다. 당신은 노드의 오른쪽값은 항상 부모노드의 오른쪽 값보다 작다는것을 알 수 있다. 그러므로, 현재노드의 오른쪽값과 스택의 마지막 오른쪽 노드값을 비교하면, 당신이 아직 부모의 자식노드를 출력하고 있는지 알수 있다. 노드의 출력이 끝났을때, 스택으로부터 오른쪽값을 삭제한다. 만약, 당신이 스택의 요소를 세어보면 현재노드의 레벨을 알수있을 것이다.

```
<?php
function display_tree($root) {
// retrieve the left and right value of the $root node
$result = mysql_query('SELECT lft, rgt FROM tree '.
'WHERE title="'.$root.'";');
$row = mysql_fetch_array($result);

// start with an empty $right stack
$right = array();

// now, retrieve all descendants of the $root node
$result = mysql_query('SELECT title, lft, rgt FROM tree '.
'WHERE lft BETWEEN '.$row['lft'].' AND '.
$row['rgt'].' ORDER BY lft ASC;');

// display each row
while ($row = mysql_fetch_array($result)) {
// only check stack if there is one
if (count($right)>0) {
// check if we should remove a node from the stack
while ($right[count($right)-1]<$row['rgt']) {
array_pop($right);
}
}
```

```
// display indented node title
echo str_repeat(' ',count($right)).$row['title']."₩n";

// add this node to the stack
$right[ ] = $row['rgt'];
}
}
?>
```

이 코드를 실행해보면 재귀적함수를 사용했을때와 같은 결과를 볼 수 있을 것이다. 우리의 새로운 함수는 더 빠르며, 재귀적이지 않고 단 두번의 쿼리만 사용한다.

노드의 경로
이 새로운 알고리즘으로 특정노드의 경로를 가져오는 새로운 방법도 찾을 수 있다. 이 경로를 얻기 위해서 우리는 모든 조상의 리스트를 필요로 한다.

우리의 새로운 테이블 구조는 더 많은 일을 필요로 하지 않는다. 예를들어, 4-5를 갖는 "Cherry"노드의 경우, 노드의 왼쪽값이 4보다 작고, 오른쪽값이 5보다 큰값을 찾으면 된다.

```
SELECT title FROM tree WHERE lft < 4 AND rgt > 5 ORDER BY lft ASC;
```

이 쿼리에서 정렬을 위해 ORDER BY절을 반드시 사용함을 주목하라. 이 쿼리는 다음을 리턴한다.

```
+--------+
| title |
+--------+
| Food |
| Fruit |
| Red |
+--------+
```

이제 이 값들을 "Cherry"와 연결하기만 하면 된다.

자손의 갯수
노드의 왼쪽값과 오른쪽값만 알려준다면, 몇개의 자손을 가지고 있는지 간단한 산술식으로 알아낼 수 있다.
각 자손은 노드의 오른쪽값을 2씩 증가시키므로, 자손의 숫자는 아래와 같이 계산될 수 있다:

```
자손수 = (오른쪽값 - 왼쪽값 - 1) / 2
```

이 간단한 식으로 나는 2-11의 "Fruits"노드가 4개의 자손노드를 가졌으며, 8-9인 "Banana"노드는 자식을 가지지 않은 노드임을 알 수 있다.

자동 트리 검색
지금까지 이 테이블로 할수있는 간단한 것들 몇가지를 보았다. 이제 우리가 어떻게 이 테이블을 자동을로 생성할 수 있는지 배워보겠다. 이제 우리를 위해 이 모든 카운팅과 트리조회를 수행하는 수크립트가 필요하다.

근접리스트를 수정된 순서화 트리검색 테이블로 변환하기 위한 스크립트를 작성하자.

```
<?php
function rebuild_tree($parent, $left) {
// the right value of this node is the left value + 1
$right = $left+1;

// get all children of this node
$result = mysql_query('SELECT title FROM tree '.
'WHERE parent="'.$parent.'";');
while ($row = mysql_fetch_array($result)) {
// recursive execution of this function for each
// child of this node
// $right is the current right value, which is
// incremented by the rebuild_tree function
$right = rebuild_tree($row['title'], $right);
}

// we've got the left value, and now that we've processed
// the children of this node we also know the right value
```

```
mysql_query('UPDATE tree SET lft='.$left.', rgt='.
$right.' WHERE title="'.$parent.'";');

// return the right value of this node + 1
return $right+1;
}
?>
```

이 것은 재귀함수이다. 당신은 반드시 이것을 rebuild_tree('Food',1); 과 같이 시작해야 한다. 함수는 "Food"노드의 모든 자식을 검색할 것이다. 더이상 자식이 없으면, 이것은 왼쪽과 오른쪽값을 설정한다. 왼쪽값에 값이 주어지면 오른쪽 값은 왼쪽값에 1을 더한다. 자식이 있을 경우 반복하여 마지막 오른쪽 값을 리턴한다. 이 값이 "Food"노드의 오른쪽 값이 된다.

재귀는 이것을 이해하기 매우 복잡한 함수로 구현한다. 하지만, 이 함수는 이 섹션의 맨 처음에 구현했던것과 같은 결과를 달성한다. 이것은 트리를 조회하면서 노드를 하나씩 추가한다. 이 함수를 실행한 후에, 당신은 왼쪽과 오른쪽값이 그대로임을 볼 수 있을 것이다 (빠른 확인법: 최상위노드의 오른쪽값은 전체노드수의 2배이다).

노드 추가
트 리에 노드를 어떻게 추가할 것인가? 거기에는 두가지 방법이 있다: 테이블에 부모컬럼을 유지하고 단지 rebuild_tree()함수를 재실행하는 것 -- 간단하지만 고상하지 않은 함수이다. 또는, 새로운 노드의 오른편에서 부터 모든 노드의 왼쪽과 오른쪽값을 갱신하는 것이다.

첫번째 방법은 단순하다. 당신은 근접 리스트 방법은 갱신에만 사용하고, 수정된 순서화 트리 검색 알고리즘을 조회를 위해 사용한다. 만약, 당신이 새로운 노드를 추가하려면 단지 테이블에 추가하고 부모컬럼을 설정하기만 하면 된다. 그리고 단순히 rebuild_tree() 함수를 재호출하는 것이다. 이것은 쉽다. 하지만, 큰 트리구조에서는 매우 비효율적이다.

두번째 방법은 새로운 노드의 오른쪽에 있는 노드의 왼쪽값과 오른쪽값을 변경하여 추가, 삭제하는것이다. 예를들어 보자. 우리는 새로운 과일종인 "Strawberry"를 "Red"의 마지막 자손에 추가하고 싶다. 먼저, 우리는 약간의 공간을 확보해야 한다. "Red"의 오른쪽 값은 6에서 8로 변경되어야 하며, 7-10인 "Yellow"노드는 9-12로 변경되어야 한다. "Red"노드를 갱신한다는 것은 왼쪽과 오른쪽값이 5보다 큰값들에 2를 더해야 함을 의미한다.

우리는 다음 쿼리를 사용할 것이다.

```
UPDATE tree SET rgt=rgt+2 WHERE rgt>5;
UPDATE tree SET lft=lft+2 WHERE lft>5;
```

이제 우리는"Strawberry"노드를 새로은 공간에 추가할수 있다. 이 노드는 왼쪽에 6, 오른쪽에 7을 가진다.

```
INSERT INTO tree SET lft=6, rgt=7, title='Strawberry';
```

display_tree()함수를 실행하면, 새로운 "Strawberry"노드가 추가된 것을 볼 수 있을 것이다.

```
Food
Fruit
Red
Cherry
Strawberry
Yellow
Banana
Meat
Beef
Pork
```

단점
수 정된 순서화된 트리조회 알고리즘(modified preorder tree traversal algorithm)은 이해하기 약간 어려워 보인다. 이것은 분명히 근접리스트 방법 보다 덜 간단하다. 하지만, 한번 당신이 왼쪽과 오른쪽 속성을 사용하게 되면, 당신은 근접리스트방법으로 했던 모든것을 이 테크닉으로 할수 있음이 명확해 질것이다. 그리고, 수정된 순서화 트리조회 알고리즘은 훨씬 빠른다. 트리를 갱신하는데는 쿼리를 더해 느려지지만 노드를 조회하는것은 단지 하나우 쿼리로 할 수 있다.

결론
이제 당신은 데이터베이스테 트리를 저장하는 두가지 방법과 친숙해 졌다. 나는 수정된 순서화 트리 조회를 좀 더 선호하지만 당신은 때에 따라 근접리스트 방법이 더 나을수도 있다. 그 결정은 당신의 몫으로 남겨두겠다.

마 지막 조언: 이미 언급한바와 같이 나는 노드의 제목(title)으로 노드를 참조하는것을 추천하지 않는다. 당신은 반드시 데이터베이스 일반화의 기본적인 규칙을 따라야 한다. 나는 예제가 읽기 편하게 하기 위해 숫자형태의 구분값을 사용하지 않았을 뿐이다.

원문주소 : http://www.sitepoint.com/print/hierarchical-data-database
번역 : 이원찬