

Tomasz "Zyx" Jędrzejewski

Drzewa w PHP i MySQL

Wersja 1.0 (3.06.2006)



Szczegółowe informacje o licencji znajdują się pod artykułem.

www.zyxist.com

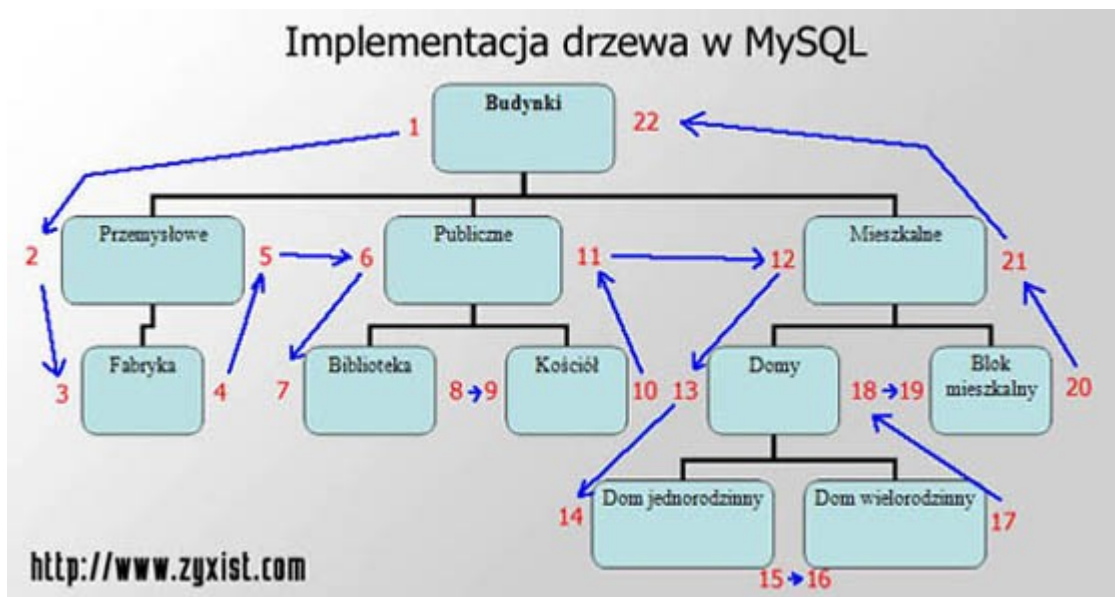
Drzewa w PHP i MySQL

Hierarchiczne, lub inaczej drzewiaste struktury znakomicie nadają się do porządkowania wielopoziomowych danych. Każdy element znajdujący się wewnątrz nich posiada tzw. "rodzica" będącego elementem nadrzędnym. Jednocześnie on sam może posiadać dowolną ilość podelementów. Podobnie działa system plików w systemach operacyjnych. Jest jeden centralny węzeł zwany korzeniem, a wewnątrz niego znajdują się pliki i katalogi. Te drugie mogą zawierać kolejne pliki i katalogi i tak w nieskończoność (w teorii).

Niestety język SQL wykorzystywany do komunikacji ze wszystkimi najpopularniejszymi systemami baz danych (Oracle, MySQL, PostgreSQL itd.) nie zawiera dosłownie żadnych mechanizmów pozwalających na szybką i efektywną implementację drzew. Normalny programista zrobiłby tak, jak podpowiada logika, tj. utworzył w tabeli pole "parent_id" przechowujące identyfikator elementu nadrzędnego. Zgoda, tyle że czas potrzebny na wyświetlenie takiego drzewa jest niezwykle długi. Język SQL nie zezwala na utworzenie zapytań rekurencyjnych, zatem dla każdego odgałęzienia i każdego poziomu musimy wywołać osobne zapytanie. Im większe drzewo, tym więcej się ich wysyła i tym wolniej wszystko działa. A tego przecież nie chcemy, prawda?

Surfując po sieci trafiłem jednak na bardzo dobre ominięcie tego problemu. Dzięki specyficznej strukturze można drzewo wyświetlić w całości jednym tylko zapytaniem bez względu na jego rozmiar oraz ilość rozgałęzień. W dodatku korzysta ono z podstawowych elementów języka SQL, co gwarantuje jego idealne działanie na wszystkich bazach danych potrafiących solidnie go obsłużyć.

Na czym polega sztuczka? Otóż wystarczy nasze drzewo... rozplaszczyc i utworzyć między węzłami drogę w taki sposób, jak pokazałem na rysunku. Jeżeli znasz się na algorytmice, prawdopodobnie już zauważyłeś, że opisywany tutaj sposób jest niczym innym, jak implementacją metody przechodzenia przez drzewa metodą *preorder*, tyle że zaimplementowaną w MySQL'u:



Po co te liczby? To jest klucz do sukcesu. Do każdego węzła dodajemy dwa pola: *left* oraz *right*. Na rysunku ich wartości obrazują właśnie liczby znajdujące się po lewej (*left*) oraz prawej (*right*) stronie danego elementu. Jak je dobrać? Jeżeli dany element nie zawiera żadnych podelementów, wtedy *right* jest większe o 1 od *left*. W przeciwnym wypadku różnica między nimi musi być taka, aby pomieściły się między nimi analogiczne wartości we wszystkich podwęzłach. Zobaczmy, jak to jest na rysunku: węzeł "Fabryka" nie zawiera podelementów, dlatego opisaliśmy go liczbami 3,4. Zawarty on jest w węzle "Przemysłowych" opisanym jako 2,5. Pomiędzy 2 i 5 mieści się 3,4 i stąd wiemy, że "Fabryka" leży w "Przemysłowych". Zauważmy też kolejną rzecz: jeśli dwa węzły leżą na tym samym poziomie w tej samej gałęzi, wtedy po wartościach *left* i *right* możemy określić kolejność ich położenia. Na rysunku obrazuje to gałąź "Publiczne" z "Biblioteką" i "Kościółem".

Zatem *left* i *right* wyznaczają swym zakresem pewien "pojemnik" dla wszystkich węzłów potomnych, ich węzłów potomnych itd. Takie podejście do problemu sprawia, że możemy zaimplementować nasze drzewo tak, jak chcemy. Nie korzystamy z żadnych niestandardowych rozszerzeń języka SQL, a całe drzewo da się wyświetlić jednym zapytaniem.

Przejdźmy do kodowania. Na początek struktura naszej tabeli oraz dane testowe:

```
CREATE TABLE `drzewko` (  
  `id` int(8) NOT NULL auto_increment,  
  `nazwa` varchar(32) collate utf8_polish_ci NOT NULL default '',  
  `left` int(8) NOT NULL default '0',  
  `right` int(8) NOT NULL default '0',  
  PRIMARY KEY (`id`),  
  KEY `parent` (`left`),  
  KEY `right` (`right`)  
);  
  
INSERT INTO `drzewko` VALUES (1, 'Budynki', 1, 22);  
INSERT INTO `drzewko` VALUES (2, 'Przemyslowe', 2, 5);  
INSERT INTO `drzewko` VALUES (3, 'Publiczne', 6, 11);  
INSERT INTO `drzewko` VALUES (4, 'Mieszkalne', 12, 21);  
INSERT INTO `drzewko` VALUES (5, 'Fabryka', 3, 4);  
INSERT INTO `drzewko` VALUES (6, 'Biblioteka', 7, 8);  
INSERT INTO `drzewko` VALUES (7, 'Kosciol', 9, 10);  
INSERT INTO `drzewko` VALUES (8, 'Domy', 13, 18);  
INSERT INTO `drzewko` VALUES (9, 'Blok mieszkalny', 19, 20);  
INSERT INTO `drzewko` VALUES (10, 'Dom jednorodzinny', 14, 15);  
INSERT INTO `drzewko` VALUES (11, 'Dom wielorodzinny', 16, 17);
```

Aby wyświetlić drzewko, potrzebne są nam tylko dwa zapytania:

```
# Pobierz parametry left i right dla elementu, od  
ktorego zaczac wyswietlanie  
SELECT `left`, `right` FROM drzewko WHERE id='1';  
  
# Wswietl drzewo, za @left i @right wstawiajac  
dane z poprzedniego zapytania  
SELECT `nazwa`, `left`, `right` FROM drzewko  
WHERE `left` BETWEEN @left AND @right ORDER BY `left`;
```

Tak oto otrzymujemy listę podelementów ułożoną już od razu w odpowiedniej kolejności. Jednak aby przypominała ona choć trochę drzewo, potrzebny jest nam jeszcze odpowiedni skrypt, który zrobi użytek z drugiego parametru, *right*. Przy wyświetlaniu każdego elementu będziemy go odkładać na stos, który na samym początku pętli jest "przycinany" do czasu, gdy zawarte w nim wielkości będą mniejsze od wartości *right* aktualnie obrabianego węzła. W ten sposób wielkość stosu zawsze będzie wskazywała na poziom jego zagłębienia. Poniżej przedstawiam funkcję rysującą takie drzewko opisanym tu sposobem:

```
<?php  
function displayTree($root)  
{  
  // pobierz parametry glownego wezla  
  $r = mysql_query('SELECT `left`, `right` FROM  
    drzewko WHERE id=\''.$root.'\'');  
  if($row = mysql_fetch_assoc($r))  
  {  
    $right = array();  
    // wyswietl wezly  
    $r = mysql_query('SELECT `nazwa`, `left`,  
      `right` FROM drzewko WHERE `left`  
        BETWEEN \''.$row['left'].'\' AND  
        \''.$row['right'].'\' ORDER BY `left`');  
    while($row = mysql_fetch_assoc($r))  
    {  
      // czysc stos
```

```

        if(count($right) > 0)
        {
            while($right[count($right)-1] < $row['right'])
            {
                array_pop($right);
            }
        }
        // wyswietl element
        echo str_repeat('| ',count($right))."\n";
        if(count($right) - 1 > 0)
        {
            echo str_repeat('| ',
                count($right) - 1).'+- '.
                $row['nazwa']."\n";
        }
        else
        {
            echo '+- '.$row['nazwa']."\n";
        }
        // zloz jego parametr 'right' na stos
        $right[] = $row['right'];
    }
    // wszystko jest OK
    return 1;
}
// tere fere, nie ma takiego wezla
return 0;
} // end displayTree();
?>

```

Dodawanie elementów wymaga wykonania trochę większej pracy, niż w normalnie, ponieważ potrzebne są aż cztery zapytania. Gdy bowiem dodamy nowy węzeł, trzeba przesunąć w górę wartości "left" i "right" wszystkich pozostałych leżących w dalszej części naszej drogi. Oto zaczątek funkcji służącej do tego celu:

```

<?php
function createNode($id, $title)
{
    // zablokuj innym dostep do tabeli, aby sie nic nie pokopalo
    mysql_query('LOCK TABLES `drzewko` WRITE');
    $r = mysql_query('SELECT `left`, `right`
        FROM drzewko WHERE id=\''.$id.'\'');
    if($row = mysql_fetch_assoc($r))
    {
        $left = $row['left'];
        $right = $row['right'];
    }
    else
    {
        // nie znaleziono elementu
        // nadrzednego, zacznij nowe drzewo
        $left = 0;
        $right = 1;
    }

    // przesun wartosci parametrow nastepnych wezlow o 2
    mysql_query('UPDATE drzewko SET `right`=`right`+2
        WHERE `right` > '.$right);
    mysql_query('UPDATE drzewko SET `left`=`left`+2
        WHERE `left` > '.$right);

    // dodaj nowy element
    mysql_query('INSERT INTO drzewko (`nazwa`, `left`,

```

```

        `right`) VALUES (\'. $title.\', \'. $right.\',
        \'. ($right+1).\');
    // zdejmujemy blokadę, tabela ponownie nadaje
    // się do użytku
    mysql_query('UNLOCK TABLES');
} // end createNode();
?>

```

<p>Piszę "zaczątek", ponieważ obecnie, jeśli pomylimy ID rodzica, funkcja pomyśli, że zaczynamy robić nowe drzewo i wszystko pochrzani :). Dodatkowo przypominam, aby nie zapomnieć tutaj o założeniu blokady na tabelę na czas dodawania tak, jak ja to zrobiłem. Inaczej przy dużym ruchu oglądający może ujrzeć głupoty, ale to nie wszystko. W najgorszym przypadku dwie osoby mogą dodać jakiś węzeł w tym samym czasie, co ewidentnie rozwalą nam bazę.

W analogiczny sposób realizuje się usuwanie; trzeba tylko odwrócić czynności.

Opisana tu struktura ma jeszcze dwie ciekawe właściwości. Na początek odpowiemy na pytanie, ile elementów zawiera dany węzeł. Możemy to wyliczyć bez uciekania się do funkcji COUNT:

```
(right-left-1)/2
```

Następnie zajmijmy się stworzeniem paska nawigacyjnego. Wykorzystują go często i gęsto rozmaite serwisy WWW, aby oglądający mógł zorientować się, jak głęboko w nich ugrząsł: *Zyxlaski.pl* » *Importowane* » *Mahoń* » *Prod. szwedzka* » *Superkijek 2000 Pro*. Tu również wystarczą nam dwa zapytania:

```

# Pobierz parametry left i right elementu, w którym jesteś
SELECT `left`, `right` FROM drzewko WHERE id='1'

# Wyświetl ścieżkę do niego
SELECT `nazwa` FROM drzewko WHERE `left` <= @left AND
`right` >= @right ORDER BY `left`

```

Jeśli chcemy, aby na pasku NIE pojawiła się także nazwa właśnie oglądanego elementu, zamieniamy operatory `<= i >=` na `< i >`.

Na koniec taka uwaga. Osobiście stwierdziłem, że mimo wszystko najlepiej jest połączyć ten sposób z metodą tradycyjną, tj. zostawić także pole "parent_id". Dzięki temu będziemy mogli też wyświetlać poziomy płasko, bez zgłębiania się w ich podwężły itd. Na jego podstawie jesteśmy też w stanie odtworzyć budowę drzewa, gdyby ktoś przez przypadek namieszał nam z parametrami *left* i *right*.

W powyższym kodzie brakuje wielu istotnych metod, m.in. zamieniania dwóch równorzędnych węzłów miejscami. Pozostawiam to czytelnikowi jako ćwiczenie. Opracowanie wszystkich funkcji zarządzania drzewem zajęłoby tak dużo miejsca, że przekracza możliwości tego tekstu, a napisanie ich samodzielnie nie stanowi zbyt dużego problemu, jeśli pojmie się zasadę, na jakiej wszystko funkcjonuje.

Na tym kończę ten artykuł. Gwoli ścisłości dodam, że pomysł na wyświetlanie drzew w taki sposób dał mi tekst [Storing Hierarchical Data in a Database \(http://www.sitepoint.com/article/hierarchical-data-database\)](http://www.sitepoint.com/article/hierarchical-data-database).

Tomasz "Zyx" Jędrzejewski

Aktualna wersja artykułu zawsze na **www.zyxist.com**

Licencja

Artykuł rozpowszechniany jest na licencji **Creative Commons Uznanie autorstwa-Użycie niekomercyjne-Bez utworów zależnych 2.5 Polska**.

Wolno:

- kopiować, rozpowszechniać, odtwarzać i wykonywać utwór

Na następujących warunkach:

1. *Uznanie autorstwa*. Utwór należy oznaczyć w sposób określony przez Twórcę lub Licencjodawcę*.
2. *Użycie niekomercyjne*. Nie wolno używać tego utworu do celów komercyjnych.
3. *Bez utworów zależnych*. Nie wolno zmieniać, przekształcać ani tworzyć nowych dzieł na podstawie tego utworu.

W celu ponownego użycia utworu lub rozpowszechniania utworu należy wyjaśnić innym warunki licencji, na której udostępnia się utwór.

Każdy z tych warunków może zostać uchylony, jeśli uzyska się zezwolenie właściciela praw autorskich.

Powyższe postanowienia w żaden sposób nie naruszają uprawnień wynikających z dozwolonego użytku ani żadnych innych praw.

Internetowa skrócona wersja licencji:

<http://creativecommons.org/licenses/by-nc-nd/2.5/pl/>

Pełen tekst licencji:

<http://creativecommons.org/licenses/by-nc-nd/2.5/pl/legalcode>

Podczas publikacji należy podać następujące informacje:

1. Link do internetowej skróconej wersji licencji.
2. Informację o wersji publikowanego tekstu.
3. Informacje o autorze w następujący sposób:

Tomasz "Zyx" Jędrzejewski

Aktualna wersja artykułu zawsze na **www.zyxist.com**