Application for "Google Summer of Code 2007":

Extending the Nested Sets Model with "Hardlinked Nested Sets"

by Enno Klasing

Abstract:

Some of the most often requested new features for Joomla! are the abilities to nest categories in an unlimited depth, and to place content into more than one category. The usage of the Nested Sets Model offers an elegant and efficient solution to this feature request.

During this year's Google Summer of Code, I would like to extend the current Node Based Scheme libraries [1] – which implement a Nested Sets Model – with a feature which I call "Hardlinked Nested Sets". Often it is needed to place the same item into different parent items at the same time. This is a simple task when dealing with just one single item each time. However, as soon as one wants to duplicate a complete branch of items (e.g. for categories with sub-items in it), this task becomes a whole lot more complicated.

Data integrity, execution time and the forming of valid routes to such sub-items are three major issues which need to be taken into account.

- Data integrity should be achieved at the library level, so that whenever a node which has hardlinked duplicates is being changed, all duplicated nodes will be changed as well. The Nested Sets Model (which is used by the NBS) already performs non-atomic actions, and therefore requires transactions. So if all extensions use the provided libraries, data integrity can be guaranteed.
- When the main functionality becomes implemented in the data-changing routines, the "Hardlinked Nested Sets" feature will have a very small impact on execution time of data reading functions, but requires a little bit more time for storing data.
- Routes to such hardlinked sets should still contain the IDs or names of the original items (except for the item where the linked branch starts), to present unique IDs or names to the outside (important e.g. for human readable URLs) and inside (important for future ACL implementations).

Detailed Description:

In the following paragraphs, I will show some use cases, outlining why I think my proposed project is of great value for Joomla!. That is being followed by a list of milestones I want to achieve, including the expected time frame for each. More detailed tasks for each milestone are listed separately. An introduction to myself can be found at the bottom of this document.

Background:

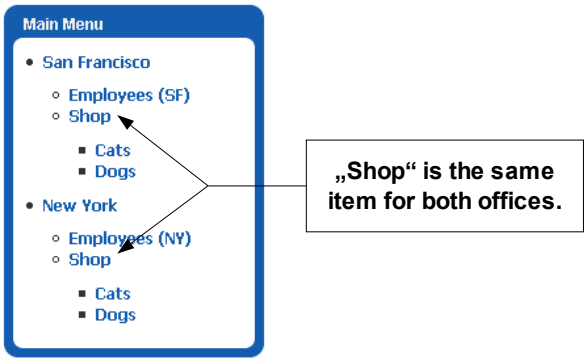
Many new features in future versions of Joomla! rely on the Nested Sets Model (NSM) [2]. Therefore, it is important that the corresponding libraries offer the needed functionality at an early stage of the development process. While the basic NSM has been implemented by Aaron Stone during Summer of Code 2006 [1], I think it is very important to extend the model with functionality for "hardlinking" whole branches of a tree. I will start by elaborating some of my ideas for the future development of Joomla!, as it will show the need for the proposed "Hardlinked Nested Sets"

project.

The NSM offers an elegant and efficient way to store tree structures of normal content items (when speaking of "content", I refer not only to articles, but also any other item, like weblinks, categories etc.). It lacks one major feature, though: the ability to copy whole branches of a tree, and to keep it in sync with the original branch afterwards. Here are a few examples where that feature is needed:

- Link whole branches of a menu tree to another "mount point". This would be needed e.g. for a company which has a menu for each locale office on their site. Let's assume each locale office wants to have a common submenu structure, e.g. a shop category structure.

  Of course this could be set up manually by copying an existing tree into the menus of all locale offices. However, when changes to the shop's category structure happen, they need to be made to all copies of the submenu structure. It would be much more convenient if the user had to make such a change to only one of the branches. With a "Hardlinked Nested Sets" framework library, this change would automatically be forwarded to the other branches of the tree.



- The same example is true for linking whole content categories into multiple parent categories.
- I hope that one day, articles will not consist of one single entry in a database table anymore. Instead of that, I propose to separate articles into pages, and pages into paragraphs. The "article=>page=>paragraph" tree could be saved as a nested set.

  This offers some nice new functionalities. For example, fine-grained ACL control up to the paragraph level can be implemented. The article could have additional pages/paragraphs which will only be shown on some output formats (e.g. an intro text for the front page or blog view, an abstract especially for feed readers, a paragraph that shows only on printouts of that article. And last but not least, it offers the possibility to show the same paragraph in multiple articles.

  While it brings huge advantages to change the article structure in such a way, it makes putting one article into multiple categories harder (or even nearly impossible) without a "Hardlinked Nested Sets" library.

I think this outlines the need for "Hardlinked Nested Sets".

Technical details:

The usual table for nested sets only needs four fields:

- the primary key
- two values for ordering the items (usually called "lft" and "rgt")
- actual data: either fields to store the data within the tree entry itself, or the key of a linked item (e.g. the real weblink, or an article). The latter form has the advantage of producing smaller tree tables and offering to link the same item more than once to put it into different

nodes.

The idea to "Hardlinked Nested Sets" is to extend these fields by one new field: the ID of the linked item. I will call this field "ref_id" in this document.

For usual items, this would only mean that the primary key will be duplicated into the "ref_id" field. Linked items will be inserted just like any other node into the tree. The only difference is that the "ref_id" will be set to the id of the original item.

The following picture shows a tree where "GROUP1" has been inserted into "GROUP2". The "level" field is not part of the table, but generated at execution time for demonstration purposes.

id	lft	rgt	ref_id	comment	level
1	1	28	1	USERS	0
2	2	11	2	GROUP1	1
4	3	4	4	alice	2
5	5	6	5	bob	2
9	7	8	9	charlie	2
10	9	10	10	dora	2
3	12	27	3	GROUP2	1
6	13	14	6	enno	2
7	15	16	7	friesengeist	2
11	17	26	2	HARDLINKED: GROUP1	2
12	18	19	4	alice	3
13	20	21	5	bob	3
14	22	23	9	charlie	3
15	24	25	10	dora	3

Usual data retrieving functions should use the "ref_id" field when reading the tree, instead of "id".

Data modifying functions need to apply the current action on all hardlinked items. While there is not much to say about functions which simply modify or reorder the tree, special attention has to be paid when deleting items. When deleting one item out of a hardlinked tree (e.g. "bob", ref_id=5), this should be done to all copies as well. However, when deleting the "mount point" of a hardlinked tree (id=11 or id=2), only the specified branch should be deleted, leaving all other copies as they are. When deleting the main tree (in this case id=2), which is hardlinked somewhere else (id=11), we must take care to delete one hardlinked clone (here id=11), and move the main tree there afterwards, to ensure that we do not delete items which are referenced from another part of the tree.

A word about normalization and transactions: I do realize that this proposal does add one more constraint to all data updating functions. However, since the NSM already requires transactions to ensure the correctness of all NBS tables, data integrity can be achieved for "Hardlinked Nested Sets" as well. It is important that only the library functions will be used to alter any tree structures.

Goal for this Summer of Code Project:

The goal for this Summer of Code Project shall be as follows. The more detailed tasks for each milestone can be found in the next section.

- Milestone 1:
  Extend and test the current NBS implementation.
  ~ 3 Weeks
- Milestone 2:
  Create, test and document Joomla! framework functionality which allows for creating, editing, moving and deleting of "Hardlinked Nested Sets".
  ~ 5.5 Weeks
- Milestone 3:

Create, test and document Joomla! framework functionality which allows for building and retrieving routes to tree items.
~ 3.5 Weeks

- Milestone 4 (optional):
  Create, test and document Joomla! framework functionality which allows for loose syncing between different NBS tables.

From past experience I know that I tend to underestimate the time needed to accomplish certain tasks (especially testing). Therefore, I have allocated plenty of time to the first three milestones. I hope to get them done in less time than noted above, so that I can also work on milestone #4.

Tasks:

- Read the relevant parts of "Joe Celko's Trees and Hierarchies in SQL for Smarties" before the programming time of SoC starts.
- Test and enhance the existing implementation of the NBS:

  The current implementation lacks some features like the ability to have additional fields in the NBS table, and some functions like "insertNodeBefore", "moveNodeAfter", "moveNodeBefore" as well as "modifyNode".

  Currently, it is assumed to have a separate database table just for the tree structure (NSM) for each table which holds objects that can be put into a tree structure. This is a very good approach for most circumstances. However, it is sometimes required to enhance the object with attributes that are better stored with the tree than with the actual object. To give an example, the human readable names of menu items will often be searched for, and therefore it makes sense to store them together with the tree to avoid unnecessary table joins. The NBS library should be able to handle both, the plain NBS tree, as well as optional additional data. For small and simple components, one could even think about storing the actual data in the same table than the tree structure. (This leads to denormalization when using hardlinked tree entries, but as previously said, the NBS framework library can deal with that, and ensure the correctness of all linked items.)

  This task belongs to milestone #1.
- Implement, test and document functions to allow for creating hardlinked nested sets:

  Creating hardlinked copies mainly means copying an existing tree structure and adjusting IDs to linked items. Special care has to be taken for nested sets which contain hardlinked nested sets within itself.

  This task belongs to milestone #2.
- Two tasks: Adjust all functions which (a) modify or (b) delete nodes:

  Actually, the interface of these functions should stay nearly the same, since linked nested sets appear as usual nested sets to higher level program parts.

  This task belongs to milestone #2.
- Follow routes that are given from the outside.

  Complete routes (where a key for each level is given, e.g. a URL like "/regions/germany/bavaria") need to resolve to a set of nodes. On the other hand, the system has to be able to return such a set of nodes (the path) also if just a single ID of a node is know (e.g. the ID for "bavaria"). If multiple nodes for the same item exist, due to the existence if a hardlinked nested set for such a item, we should return the path to the first occurrence to this item, or upon request all paths to this item. It is important though, to have

a defined mode of operation, to avoid the problems that the Itemid brought in Joomla! 1.0.

This task belongs to milestone #3.

- Create routes to items.

  This task is about creating complete routes to given items, by using a helper class. The idea is to be able to specify any database field that should be used to create such a route, e.g. the IDs of the nodes, or the names of the nodes.

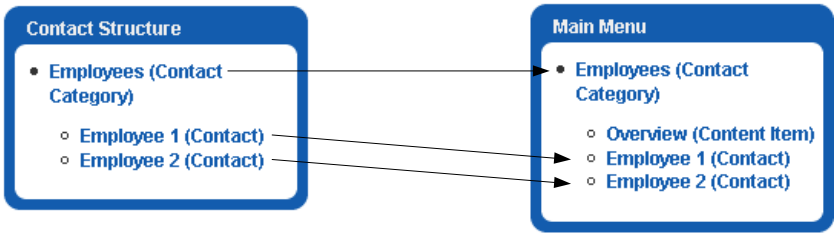
  When dealing with hardlinked nodes, one should optionally be able to specify branches which should be searched first or only. This allows for returning nodes from within a specific branch of the tree. In all cases, the first found route will be returned, be it on a prioritized branch, or the first found route on the whole tree if no nodes to search were given. This makes the algorithm as deterministic as it can be.

  This task belongs to milestone #3.

- Loose syncing between different NBS tables.

  The difference of content structure vs. menu structure can be pretty hard to grasp, especially for newcomers. Therefore, I want to propose a mechanism which I call "loose syncing", to forward changes that are done on one NBS table into an other. Syncing a content category to a corresponding menu entry would be an example for this. If e.g. a new contact is created, and if a menu item to the contact category exists and has the "syncing option" enabled, this would mean that a new menu entry to this contact is automatically created as well.

  I call this feature "loose" syncing for two reasons: unlike with hardlinked nested sets, it is not important to have atomic or transactional operations. If the change to one table is not being forwarded to the other table, this is not critical in any way for the correctness of the tables. Secondly, as it's a one-way syncing, the corresponding (in this example) menu node can hold items which are not present in the contact category. E.g. the menu could have an entry linking to an additional content item. This picture shows both trees, the contact and the menu tree:



  This task belongs to milestone #4.

- Unit testing of all created functions.

  Ever since I wrote my first unit test (not so long ago) I believe in the power of unit tests. I want to write a unit test for each function I write, ideally even before writing the function itself.

- Documentation of the whole created code.

  Besides using DocBook tags in my code, I plan on writing a small abstract about each function and it's usage. I think it is important to document things by the time of writing the code, as it is much more unlikely to be written at any later time.

Self-Introduction and Motivation:

My Name is Enno Klasing, I'm a German guy in the age of 23. Currently, I'm in my seventh semester, studying "Electrical Engineering and Information Technology" at the "Universität

Karlsruhe (TH)".

[Personal details have been deleted from this document, they are available for mentors on the original application.]

Links:

[1] http://code.google.com/soc/2006/joomla/appinfo.html?csaid=23A1F7A6CF74EC01
[2] http://www.intelligententerprise.com/001020/celko.jhtml