# Improve Query Performance On Hierarchical Data. Adjacency List Model Vs. Nested Set Model

Cornelia Győrödi
Department of Computer Science and Information Technology, University of Oradea
Oradea, Romania

Robert Győrödi
Department of Computer Science and Information Technology, University of Oradea
Oradea, Romania

Romulus-Radu Moldovan-Duşe
Department of Computer Science and Information Technology, University of Oradea
Oradea, Romania

George Pecherle
Department of Computer Science and Information Technology, University of Oradea
Oradea, Romania

***Abstract*—Hierarchical data are found in a variety of database applications, including content management categories, forums, business organization charts, and product categories. In this paper, we will examine two models deal with hierarchical data in relational databases namely, adjacency list model and nested set model. We analysed these models by executing various operations and queries in a web-application for the management of categories, thus highlighting the results obtained during performance comparison tests. The purpose of this paper is to present the advantages and disadvantages of using an adjacency list model compared to nested set model in a relational database integrated into an application for the management of categories, which needs to manipulate a big amount of hierarchical data.**

***Keywords—adjacency list model; nested set model; relational database; MSSQL 2014; hierarchical data***

## I. INTRODUCTION

Most of the database developers have dealt with hierarchical data in a relational database, and without a doubt, they reached the conclusion that relational database is not designed to manage data in a hierarchical way. Hierarchical data can be found in a great variety of database applications like the threads from forums or from emails, flowchart, content management categories or products categories [5].

Relational databases are widely used in most of the applications, and they have good performance when they handle a limited amount of data. To handle a huge volume of data like the internet, multimedia and social media the use of traditional relational databases is inefficient. Thus, more and more applications are beginning to use a non-relational database because they provide a more flexible structure that can shape after each user' needs; they are designed to store large amounts of data, and they have denormalized databases, which increases performance [6].

The tables from a relational database are not hierarchical. Hierarchical data have a parent-child relationship, which is not normally represented in a relational database table. A relational database does not store records in a hierarchical way. 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, as presented in [5].

These hierarchical data must also be stored in relational databases to obtain easier and more intuitive navigation through it [1] [8]. Because of these needs, there has always been an attempt to find solutions as close as possible to the application needs.

In this paper, we will examine two models dealing with hierarchical data in relational database namely adjacency list model and nested set model.

We will start with adjacency list model, and we will consider an example of the hierarchy of categories from an ads web-application as shown in Fig. 1.

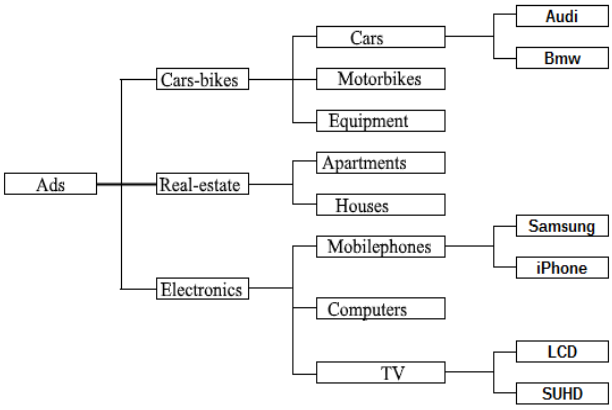


Fig. 1. Hierarchical data structure

The adjacency list model is very easy to maintain but is less efficient for queries. A hierarchical query is a method of reporting the branches of a tree in a specific order.

If we need a better performance from queries, and the hierarchical data does not have frequent changes in time, nested set model was proved by tests conducted by us to be a more efficient model.

In the next sections of this paper, we will describe the adjacency list model compared to the nested set model, query algorithm in adjacency list model compared to nested set model, study and analyse performance and the final conclusions.

## II. Description of the Nested Set Model Compared to the Adjacency List Model in Relational Databases

To be able to understand why one is better than the other models in certain situations, we must first describe the two models.

The adjacency list model is probably the most common type of hierarchical structure found in databases and is characterized by nodes and lines. The reason for the popularity of this model is that it is easy to understand and maintain. The connection between nodes is made via an attribute called parent, that is the head of the hierarchy and has the parent attribute set to NULL and the rest of the elements have values corresponding to their parent's ids. Based on the previous section example, in Fig. 2 shows the hierarchical data structure as an adjacency list model and each colour represents a level in the hierarchy:

| id | name | parentid |
|---|---|---|
| 1 | Ads | NULL |
| 2 | Car-bikes | 1 |
| 3 | Real-estate | 1 |
| 4 | Electronics | 1 |
| 5 | Cars | 2 |
| 6 | Motorbikes | 2 |
| 7 | Equipment | 2 |
| 8 | Apartments | 3 |
| 9 | Houses | 3 |
| 10 | Mobilephones | 4 |
| 11 | Computers | 4 |
| 12 | TV | 4 |
| 13 | Audi | 5 |
| 14 | BMW | 5 |
| 15 | Samsung | 10 |
| 16 | iPhone | 10 |
| 17 | LCD | 12 |
| 18 | SUHD | 12 |

Fig. 2. Adjacency List Model exemplified on a four level hierarchical structure

Unlike the adjacency list models, in nested set models we will look at hierarchy in a different way. Each node in hierarchy will contain all the children from the nodes that are subordinated directly or indirectly to it. The nodes in the hierarchy will have two attributes, which we call right and left in which numbers will be stored to help us to identify all children of a node. The numbering technique was proposed by Joe Celko in [2], by starting from the element from the top of the hierarchy, all elements will be numbered two times, saving each value in the left, and right attributes, as you can see in the example from Fig. 3.

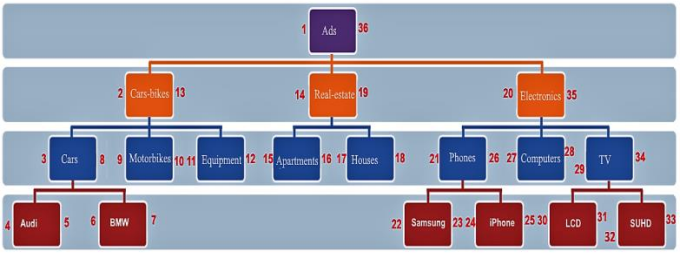


Fig. 3. Nested set model in a four level hierarchical structure

### A. Adding and deleting of nodes in a nested set model

The values of the left and right attributes of the nested set model must be recalculated when a new node is added or deleted in the hierarchy. When adding a new node in the hierarchy the left and right attributes from the parent node that will contain the new element must always be taken into consideration [3]. If we want to add a new category called *Opel* in the hierarchy in Fig. 3, which belongs to the category *Cars*, according to the numbering technique, in the new node the value 8 will be stored in the left attribute and the value 9 in the right attribute. Therefore, all left, and right attributes with values greater than or equal to the value of the right attribute of the new category *Cars* will be incremented by 2, as shown in Figure 4.

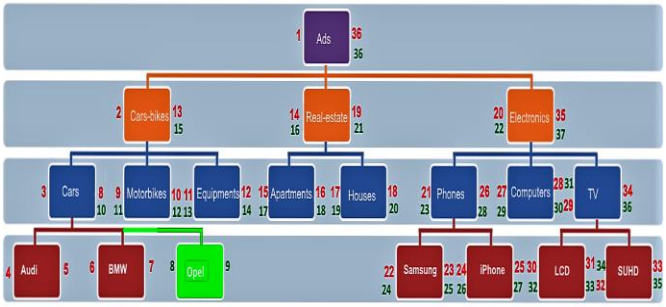


Fig. 4. Adding a subcategory in the nested set model

When we delete a node in the hierarchy, we must take into consideration the left and right attribute of the deleted node. Suppose we want to remove the *Real-estate* category in the hierarchy in Fig. 3. When deleting a category we must also take in consideration its subcategories, because they will also be deleted from the hierarchy when the category that they belong to is deleted [3]. An exact identification of all the subcategories that belong to the *Real-estate* category will be made based on the left and right attribute. We can see that all the subcategories from *Real-estate* have the left and the right attributes between 14 and 19. After we deleted the whole *Real-estate* branch, all the categories that have the value of the left and right attributes greater than 19, will be decremented with the difference between the right attribute and the left of the deleted category plus the value 1, in our case this will be 19-14+1 as you can see in the Fig. 5.

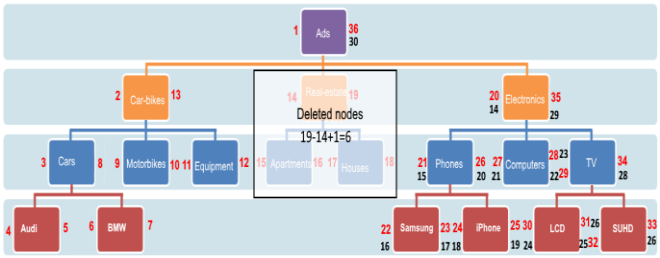


Fig. 5. Deleting a subcategory in the nested set model

## III. Queries in the Adjacency List Model Vs. the Nested Set Model

### A. Getting a subtree in the hierarchical data structure

We consider hierarchical structure presented above to exemplify queries. One of the most common queries, in this case, is to get all the subcategories that belong to a specific category.

If we want to get the subcategories that belong to the *Electronics* category, and we are considering the adjacency list model we have two options. First we could use self-join, in this case we must know the number of levels in the structure of categories and make one self-join for each level to get the categories from the lower level. For the second option we could use common table expression to build a recursive query which will return all the sublevels of category without needing to know the number of the levels existing.

The self-join query allows us to see the full path through our hierarchical data structure as shown below and in [5]:

```
SELECT t1.name AS nivel1, t2.name as nivel2,
t3.name as nivel3, t4.name as nivel4
FROM category AS t1
LEFT JOIN category AS t2 ON t2.ParentId = t1.Id
LEFT JOIN category AS t3 ON t3.ParentId = t2.Id
LEFT JOIN category AS t4 ON t4.ParentId = t3.Id
WHERE t1.name = 'Electronics';
```

```
SQL Server Execution Times:
   CPU time = 16 ms, elapsed time = 28 ms.
```

```
Table 'Workfile'. Scan count 0, logical reads 0,
physical reads 0, read-ahead reads 0, lob logical reads 0,
lob physical reads 0, lob read-ahead reads 0.
Table 'Worktable'. Scan count 0, logical reads 0,
physical reads 0, read-ahead reads 0, lob logical reads 0,
lob physical reads 0, lob read-ahead reads 0.
Table 'Category'. Scan count 3, logical reads 59,
physical reads 0, read-ahead reads 0, lob logical reads 0,
lob physical reads 0, lob read-ahead reads 0.
```

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, as presented in [5].

We use a recursive query to obtain all the sublevels of the category without needing to know the number of the existing levels.

Recursive query:

```
with CategoryBranch as (
     select Id, Name,ParentId
     from Category
     where name = 'Electronics'
union all
     select c.Id, c.Name, c.ParentId
     from Category c
     join CategoryBranch p on c.ParentId = p.Id
)select Name from CategoryBranch order by
ParentId;
```

```
SQL Server Execution Times:
  CPU time = 94 ms,  elapsed time = 89 ms.
```

```
Table 'Category'. Scan count 1, logical reads
2151, physical reads 0, read-ahead reads 0, lob
logical reads 0, lob physical reads 0, lob read-
ahead reads 0.
Table 'Worktable'. Scan count 2, logical reads
679, physical reads 0, read-ahead reads 0, lob
logical reads 0, lob physical reads 0, lob read-
ahead reads 0.
```

In the nested set model, the hierarchical data is maintained, as parent categories contain all the children from the nodes that are subordinated directly or indirectly to it. We can represent this form of hierarchical data in a table through the use of *TreeLeft* and *TreeRight* values.

```
CREATE TABLE NestedCategory (
     category_id INT AUTO_INCREMENT PRIMARY KEY,
     name VARCHAR(20) NOT NULL,
     TreeLeft INT NOT NULL,
     TreeReft rgt INT NOT NULL
);
```

We can see the full path of our hierarchical data using of a self-join that connects parents with nodes based on the fact that a node's *TreeLeft* value will always appear between its parent's left and right values as also shown in [5]:

```
SELECT node.name FROM NestedCategory AS node,
NestedCategory AS parent
WHERE node.TreeLeft BETWEEN parent.TreeLeft AND
parent.TreeRight
AND parent.name = 'Electronics'
ORDER BY node.TreeLeft;
```

```
SQL Server Execution Times:
  CPU time = 0 ms,  elapsed time = 2 ms.
```

```
Table 'NestedCategory'. Scan count 1, logical reads
233, physical reads 0, read-ahead reads 0, lob logical
reads 0, lob physical reads 0, lob read-ahead reads 0.
```

We can see from the performed tests and displayed parameters that query execution time for this type of query is the best when using the nested set model.

### B. Finding all the leaf nodes

This type of query refers to obtaining all the categories from the tree that do not contain other subcategories. Thus we need all the parents who do not have children.

In the case of the adjacency list models, we need a self-join to get parent categories that have no children [5].

```
SELECT parent.name FROM category AS parent
LEFT JOIN category as child ON parent.Id =
child.ParentId
WHERE child.Id IS NULL;
```

```
SQL Server Execution Times:
CPU time = 16 ms,  elapsed time = 387 ms.
```

```
Table 'Category'. Scan count 2, logical reads 38,
physical reads 0, read-ahead reads 0, lob logical
```

```
reads 0, lob physical reads 0, lob read-ahead reads 0.
```

For nested set model, this type of query is simpler, because it is based on a rule from this model which says that leaf node has the left and right attributes with consecutive values. Thus we have to look at the categories where left + 1 = right [5].

```
SELECT name
FROM NestedCategory
WHERE TreeRight = TreeLeft + 1;

SQL Server Execution Times:
   CPU time = 16 ms, elapsed time = 371 ms.
```

```
Table 'NestedCategory'. Scan count 1, logical reads 22, physical reads 0, read-ahead reads 0, lob logical reads 0, lob physical reads 0, lob read-ahead reads 0.
```

The execution time is insignificant distinct between those two models but the execution plan is clearly better for the nested set model, as shown in Fig. 6.

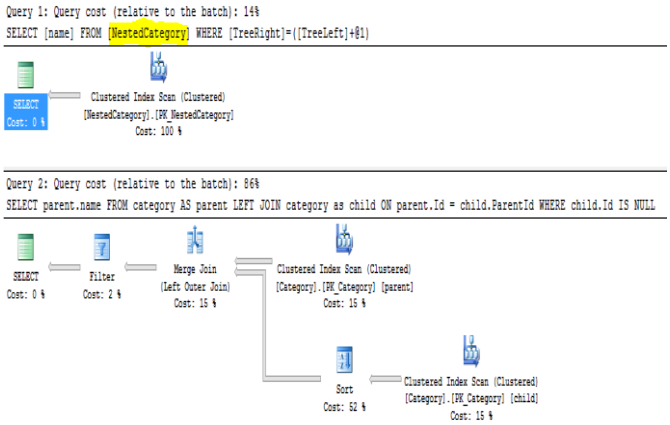


Fig. 6. Execution plan for the nested set model vs. adjacency list model

There are situations where working with adjacency list model directly in SQL can be difficult, such as the cases where we necessarily need self-joins and the exact number of existing levels in the tree or the level of the node to which we refer. In such a situation, it would be necessary to calculate the level on that is each node in the tree.

In the case of nested set model, to calculate the level of each category from the tree, we will be doing COUNT function on the parent nodes of each category based on the same rule that says any subcategory will have the left attribute value between the left and right values of the parent category.

```
SELECT node.name, COUNT(parent.name) AS CategoryLevel
FROM NestedCategory AS node,
NestedCategory AS parent
WHERE node.TreeLeft BETWEEN parent.TreeLeft AND parent.TreeRight
GROUP BY node.name
ORDER BY CategoryLevel;
```

A special case occurs when on the hierarchical data structure, we have assigned data from another table, and we want to know how they are distributed on each node in the structure. Supposing that we have a relational table with ads, and each ad belongs to a "leaf" category, if we want to display a list of all the categories and the number of ads each category, for the parent node categories we will consider the number of ads from each child subcategory that belongs to it.

In nested set model, as usual, we will use all of the attributes of the left and right to get the parent-child relationship between the categories and a join operation between the child table and table with articles to refer to articles assigned to a category and add a COUNT function in each category.

```
SELECT parent.name, COUNT(Articles.Id)
FROM NestedCategory AS child ,
        NestedCategory AS parent,
        Articles
WHERE child.TreeLeft BETWEEN parent.TreeLeft AND parent.TreeRight
        AND child.Id = Articles.NestedCategoryId
GROUP BY parent.name
ORDER BY parent.name;
```

## IV. Implementation and Performance Analysis

The application we developed using Microsoft SQL Server 2014 Management Studio [7] has 4 sections. The first section we exemplified the navigation through the categories and the listing of ads from a selected category. In the second and the third sections we implemented the management of categories using the two models described above and in a fourth section we displayed response times to queries on ads comparing the performance of the two hierarchical data models.

Because the test results depend on the computer on which these tests are carried out, it is important to note that all the results presented below were obtained from studies conducted on a computer with the following characteristics: Windows 10 Home Edition 64-bit, processor Intel Core i5 (2.4 GHz), 4 GB RAM memory. The database contains 2902 categories and 310 818 ads distributed by category.

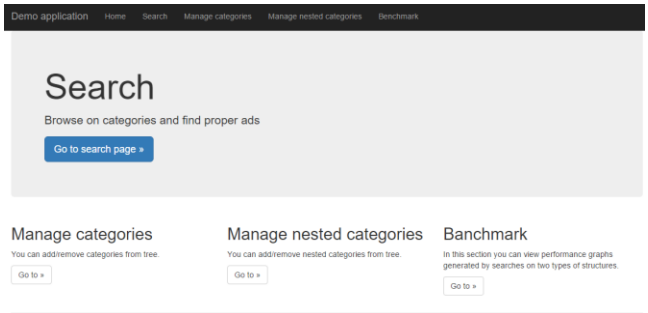


Fig. 7. Main page of the application

### A. Navigation section

For this section of the application, we used two queries that highlight the usefulness of nested set models. For both queries, we used stored procedures, *sp_GetNestedCategoryIncludingCountArticles* and *sp_GetArticlesByNestedCategoryId*.

The first procedure, named *sp_GetNestedCategoryIncludingCountArticles* is used in the navigation menu on the left, where besides displaying the hierarchical data structure of categories is also shows the number of ads in each category. Thus,

sp_*GetNestedCategoryIncludingCountArticles* procedure returns a structure of categories with the number of ads in each category, as shown in Fig 8.

```
   CREATE PROCEDURE
[dbo].[sp_GetNestedCategoryIncludingCountArticles]
        @ParentId int
   AS
   BEGIN
        SET NOCOUNT ON;

        SELECT parent.Id, parent.Name,
parent.ParentId, parent.TreeLeft, parent.TreeRight,
COUNT(Articles.Id) as CountArticles
   FROM NestedCategory AS node,
           NestedCategory AS parent,
           Articles
   WHERE node.TreeLeft BETWEEN parent.TreeLeft AND
parent.TreeRight AND node.Id =
Articles.NestedCategoryId AND parent.ParentId =
@ParentId
   GROUP BY parent.Id, parent.Name, parent.ParentId,
parent.TreeLeft, parent.TreeRight
   ORDER BY parent.Id;
   END
```

The second procedure, entitled *sp_GetArticlesByNestedCategoryId* is used to return from the database a list of ads for the selected category including subordinate categories. Thus, *sp_GetArticlesByNestedCategoryId* procedure returns ads that belong to certain categories respectively subcategories within the given category.

```
   CREATE PROCEDURE
[dbo].[sp_GetArticlesByNestedCategoryId]
        @NestedCategoryId int,
        @FromArticleId int = null
   AS
   BEGIN
        SET NOCOUNT ON;

           IF(@FromArticleId is null) SET
        @FromArticleId = 0
        DECLARE @TreeLeft int
        DECLARE @TreeRight int

   select @TreeLeft=TreeLeft, @TreeRight=TreeRight
   from NestedCategory where id = @NestedCategoryId

        select top 10
   Id, CategoryId, NestedCategoryId, Title, Body
        from Articles
           where Id > @FromArticleId and
        NestedCategoryId in    (select id from
        NestedCategory where TreeLeft >= @TreeLeft
        and TreeRight <= @TreeRight)
        ORDER BY Id
   END
```

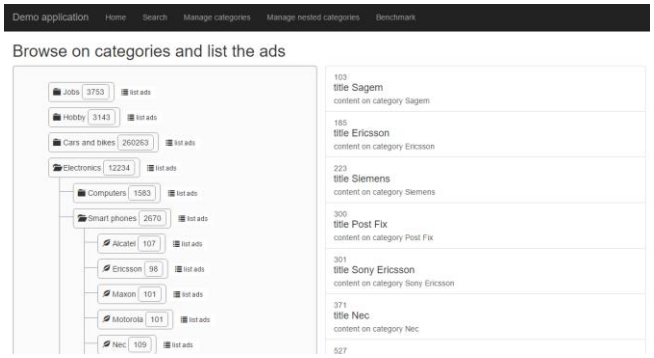


Fig. 8. Browse on categories and list the ads

## B. *Categories management section*

In this section of the application shown in Fig. 9, we can add or delete categories from the two hierarchical data structures.

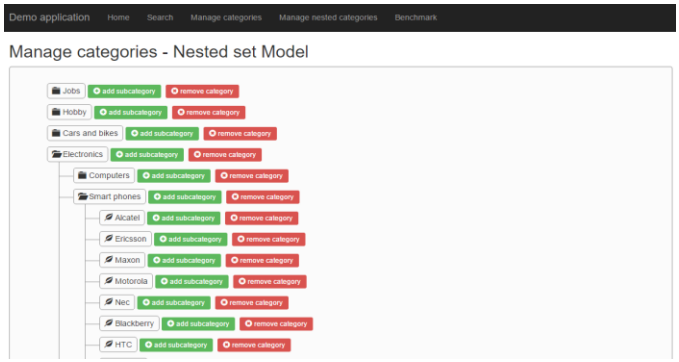


Fig. 9. Categories management page

The management interface is identical between the two hierarchical models, but at level database specific stored procedures are called for each model. In the adjacency list model the procedure that makes adding a new category has as input parameters the parent category name and the name of the new category, identify *ParentId* in the category table and then insert the new category. For the deleting a category in the adjacency list model, we used a stored procedure that has as input parameters the category name to identify the first *Id* of category and then delete the category and its children (all the categories which have *ParentId* identical with *Id* of category).

The procedure, entitled *sp_InsertNestedCategory* is used in the nested set model to add a category in the hierarchical data structure. The procedure has as input parameters the parent category name and the name of the new category, identify *ParentId* in the table, then left and right nodes updated with new values and then insert the new category.

```
CREATE PROCEDURE [dbo].[sp_InsertNestedCategory]
    @ParentName nvarchar(50),
    @Name nvarchar(50)
AS
BEGIN
    SET NOCOUNT ON;

    DECLARE @ParentId int
       SET @ParentId = (select Id from
    NestedCategory where Name = @ParentName)

       if (@ParentId is null or @Name is null or
    exists(select * from NestedCategory where
    Name = @Name))
     return;

    DECLARE @parentTreeLeft INT
    DECLARE @parentTreeRight INT
    DECLARE @countChilds int

       SET @parentTreeLeft = (SELECT TreeLeft
    FROM NestedCategory WHERE id = @ParentId)
       SET @parentTreeRight = (SELECT TreeRight
    FROM NestedCategory WHERE id = @ParentId)
       SET @countChilds = (select 2*count(*)
    from NestedCategory where TreeLeft >
    @parentTreeLeft and TreeRight <
    @parentTreeRight)

    BEGIN TRAN
    UPDATE NestedCategory
        SET TreeLeft = CASE WHEN TreeLeft >
    @parentTreeRight
                  THEN TreeLeft + 2
                  ELSE TreeLeft END,
           TreeRight = CASE WHEN TreeRight >=
        @parentTreeRight
                  THEN TreeRight + 2
                  ELSE TreeRight END
     WHERE TreeRight >= @parentTreeRight
       INSERT INTO NestedCategory(TreeLeft,
    TreeRight, ParentId, Name)
    VALUES(@parentTreeLeft + @countChilds + 1,
    @parentTreeLeft + @countChilds + 2,
    @ParentId, @Name);
    IF @@ERROR != 0
           ROLLBACK TRAN
    ELSE
           COMMIT TRAN
END
```

For the deleting a category in the nested set model we used a stored procedure entitled *sp_DeleteNestedCategory*, that has as input parameters category name to identify first *Id* of category, then the category will be deleted along with related subcategories, and left and right nodes updated by difference between *TreeRight* and *TreeLeft* + 1 of removed category.

```
CREATE PROCEDURE [dbo].[sp_DeleteNestedCategory]
    @Name nvarchar(50)
AS
BEGIN
    SET NOCOUNT ON;

    DECLARE @Id int
       SET @Id = (select id from NestedCategory
    where Name = @Name)
    if (@Id is null) return;

    DECLARE @treeLeft INT
    DECLARE @treeRight INT
    DECLARE @parentId INT
    DECLARE @treeWidth INT

       SET @treeLeft = (SELECT TreeLeft FROM
    NestedCategory WHERE id = @Id)
       SET @treeRight = (SELECT TreeRight FROM
    NestedCategory WHERE id = @Id)
    SET @treeWidth = @treeRight - @treeLeft + 1
       SET @parentId = (SELECT ParentId FROM
    NestedCategory WHERE id = @Id)

    BEGIN TRAN
       UPDATE Articles SET NestedCategoryId =
    @parentId WHERE CategoryId in (select id
    from NestedCategory where TreeLeft >=
    @treeLeft and TreeRight <= @treeRight)
       DELETE FROM NestedCategory where TreeLeft
    between @treeLeft and @treeRight
       UPDATE NestedCategory SET TreeLeft =
    TreeLeft - @treeWidth WHERE TreeLeft >
    @treeRight
       UPDATE NestedCategory SET TreeRight =
    TreeRight - @treeWidth WHERE TreeRight >
    @treeRight

    IF @@ERROR != 0
           ROLLBACK TRAN
    ELSE
           COMMIT TRAN
END
```

### C. Performance analysis of the queries

For this analysis, we considered the stored procedures used for the ads listing from a specified category for the two models studied. The two stored procedures *sp_GetArticlesByCategoryId* and *sp_GetArticlesByNestedCategoryId*, are described below. Each of the two procedures returns ten ads that are from the specified category.

```
   CREATE PROCEDURE
[dbo].[sp_GetArticlesByCategoryId]

   @CategoryId int,
   @FromArticleId int = null
   AS
   BEGIN
        SET NOCOUNT ON;
        IF(@FromArticleId is null) SET
@FromArticleId = 0;

        with CategoryBranch as (
        select Id
        from Category
        where Id = @CategoryId
   union all
        select c.Id
        from Category c
        join CategoryBranch p on c.ParentId = p.Id
   )
   select top 10
      Id, CategoryId, NestedCategoryId, Title, Body
   from Articles
   where Id > @FromArticleId and CategoryId in
         (select Id from CategoryBranch)
   ORDER BY Id;
   END

   CREATE PROCEDURE
[dbo].[sp_GetArticlesByNestedCategoryId]

        @NestedCategoryId int,
        @FromArticleId int = null
   AS
   BEGIN
   SET NOCOUNT ON;
```

```
    IF(@FromArticleId is null) SET @FromArticleId = 0
    DECLARE @TreeLeft int
    DECLARE @TreeRight int

        select                    @TreeLeft=TreeLeft,
      @TreeRight=TreeRight   from       NestedCategory
      where id = @NestedCategoryId

        select top 10
        Id,   CategoryId,   NestedCategoryId,   Title,
Body
        from Articles
        where     Id     >     @FromArticleId     and
NestedCategoryId  in(select  id  from  NestedCategory
where  TreeLeft  >=  @TreeLeft  and  TreeRight  <=
@TreeRight)
        ORDER BY Id
    END
```

In the graphs from Fig 10 and Fig. 11, we can see the response time for the two stored procedures from runs with one iteration to runs with ten iterations. We noticed that the response time is affected by the total number of the ads from the sub-branch on which the search is made, but always the best execution time is obtained by the nested set model as shown in Fig 11.

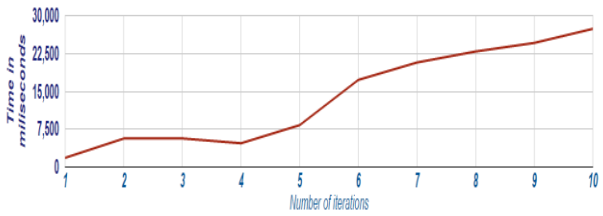


Fig. 10. Query performance for the adjacency list model

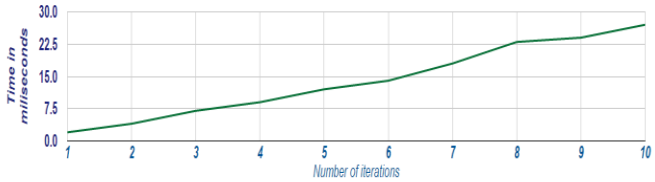


Fig. 11. Query performance for the nested set model

We can see that if we grow the number of iterations the difference of time is higher between the two models and the nested model has the best performance between the two.

## V. Conclusions

When we work with hierarchical data structures with more than 2 levels, and the number of levels varies from one branch to another of the hierarchy, then it is better to store the hierarchical data as a nested set model in the database. In the nested set model is more difficult to do the adding, moving and deleting of nodes because we need to update every time the value of the left and right attributes to keep the integrity of the hierarchy. However, the advantage is pretty big because the number of the queries on the relational table is the same no matter the number of hierarchy levels from the nested set model, on the other hand, the number of queries for the adjacency list model is equal to the number of levels of hierarchy.

In case the hierarchical structure is large, it is suggested to break it into smaller hierarchical structures that are to be stored in separate tables, thus allowing a better administration of each hierarchical structure.

As an extension of our study, we would like to compare the performance of the nested set model with the hierarchical data model implemented in Microsoft SQL Server 2012.

## References

[1] Joe Celko “Trees and Hierarchies in SQL for Smarties”, 2nd Edition Morgan-Kaufmann, 2012, ISBN 978-0-12-387733-8

[2] Joe Celko, “Trees in SQL”, Available: http://www.ibase.ru/devinfo/DBMSTrees/sqltrees.html (dec. 2015)

[3] T. Stryja, “Nested set model practical examples, part I”, Available: http://we-rc.com/blog/2015/07/19/nested-set-model-practical-examples-part-i (nov. 2015)

[4] T. Stryja, “Nested set model practical examples, part II”, Available: http://we-rc.com/blog/2015/07/19/nested-set-model-practical-examples-part-ii (nov. 2015)

[5] Mike Hillyer, “Managing Hierarchical Data in MySQL”, Available: http://mikehillyer.com/articles/managing-hierarchical-data-in-mysql/ (dec. 2015)

[6] Cornelia Győrödi, Robert Győrödi, George Pecherle, Andrada Olah, “A comparative study: MongoDB vs. MySQL”, IEEE - 13th International Conference on Engineering of Modern Electric Systems (EMES), 2015, Oradea, Romania, 11-12 June 2015, ISBN 978-1-4799-7649-2, pag. 1-6.

[7] Microsoft SQL Server 2014 Management Studio, Available: https://www.microsoft.com/en-us/download/details.aspx?id=42299 (dec 2015)

[8] Joe Celko's “SQL for Smarties”, 5th Edition, Morgan-Kaufmann, 2014, ISBN 9780128008300.