Optimized Data Indexing Algorithms for OLAP Systems

Lucian BORNAZ
Faculty of Cybernetics, Statistics and Economic Informatics
Academy of Economic Studies, Bucharest
lucianbor@hotmail.com

The need to process and analyze large data volumes, as well as to convey the information contained therein to decision makers naturally led to the development of OLAP systems. Similarly to SGBDs, OLAP systems must ensure optimum access to the storage environment.
Although there are several ways to optimize database systems, implementing a correct data indexing solution is the most effective and less costly.
Thus, OLAP uses indexing algorithms for relational data and n-dimensional summarized data stored in cubes.
Today database systems implement derived indexing algorithms based on well-known Tree, Bitmap and Hash indexing algorithms. This is because no indexing algorithm provides the best performance for any particular situation (type, structure, data volume, application).
This paper presents a new n-dimensional cube indexing algorithm, derived from the well known B-Tree index, which indexes data stored in data warehouses taking in consideration their multi-dimensional nature and provides better performance in comparison to the already implemented Tree-like index types.
Keywords: *data warehouse; indexing algorithm; OLAP, n-Tree.*

1 Introduction

Data warehouses represented a natural solution towards increasing the availability of data and information, as well as their accessibility to decision makers. The warehouses store important data coming from different sources for later processing and are an integrant part of analytical processing systems (OLAP).
Unlike OLTP systems, OLAP systems must execute complex interrogations and large data volume analyses. To optimize, analytical processing systems analyze data and store aggregated information in special analytic structures, called cubes.
Similarly to OLTP systems, OLAP systems use indexing algorithms to optimize access to data stored in data warehouses, i.e. cubes.

2. General information about cubes

When stored in an OLAP system, the source data may be indexed to reduce the time necessary for their processing. To index source data, OLAP systems use indexing algorithms similar to OLTP (B-Tree, Bitmap, R-Tree etc.).
Processed data are stored in n-dimensional structures called cubes. The elements of a cube are the dimensions, members, cells, hierarchies and properties [1] (fig. 1).

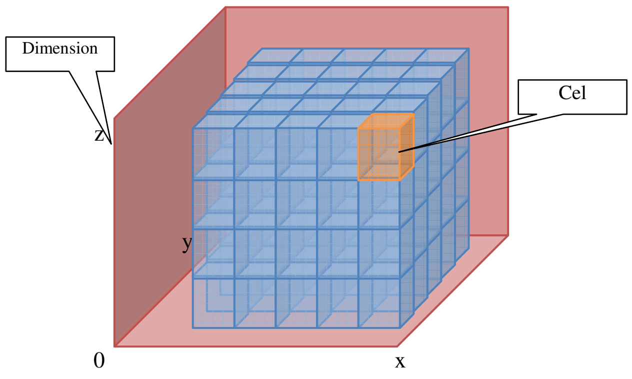


Fig. 1 - Structure of a tridimensional cube

The dimensions contain descriptive information about the data that is to be summarized. They are essential for data analysis and represent an axis of the cube [2].

Each dimension corresponds to a measure of the data source and uniquely contains each value stored in that position. During queries, the dimensions are used to reduce the search area and usually occur in the WHERE clause.

Hierarchies describe the hierarchical relationships between two or more members of the same dimension. A dimension can be part of multiple hierarchies. For example, in addition to the hierarchy of dimensions Quarter-Month-Year, Time dimension can belong to the hierarchy Day-Month-Year.

The cells of the cube contain summarized data based on dimensions values. Cells store summarized data based on the cube dimension number, dimensions values, method of analysis and is usually, the result returned by the queries.

Properties describe common features of all members of the same dimension. Properties allow selecting data based on similar characteristics. For example, the size of product volume may have an attribute which allows a certain volume products.

Analysis and data processing is based on the method chosen. The same data can be analyzed using different methods (clustering, neural nets, regression, Bayesian, Decision trees, etc.) accordingly to the user's needs. Although using different methods of analysis may result in different aggregate data and ordering different logic cells, the logical structure of the cube is the same.

In order to optimize performance, OLAP systems implement cube indexing algorithms. The indexes created through this process use the data contained in a cube's dimensions to quickly access the cells containing the data required by the user.

Hence, cubes are indexed using a B-Tree type of algorithm.

3. The B-Tree Index in OLAP Systems

As the values of the cube dimensions are unique and they are stored in the index blocks and used to locate the leaf blocks which contain references to the physical location of the cube cells, implementing a B-Tree index represents an effective solution for indexing cubes.

A B-Tree index used in OLAP systems contains sub-trees corresponding to each dimension.

The sub-trees are connected in such a way that each path to go through the tree from the root node to the final level index blocks (the ones storing the references to the cube's cells) is crossed by a sub-tree corresponding to each dimension.

Thus, a three dimensional cube contains three levels. The first level represents a matrix of planes (bi-dimensional space), the second level represents a matrix of lines (one dimensional space), and the third level represents a matrix of points in space (0 dimensional space) (fig. 2).

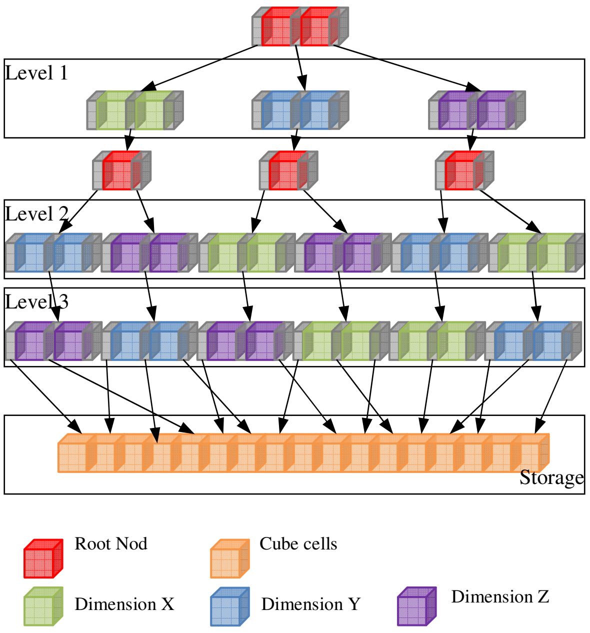


Fig. 2 - The structure of a B-Tree index for a three-dimensional cube [3]

Thus, each cube size will correspond to one sub-tree of the B-Tree index and each sub-tree will have on child sub-tree for each child dimension.

Summarized data are stored inside the cube, on pages separate from the cells. As summarized data creation and storage consumes CPU resources and storage space environment, the developer and/or system administrator can choose as they are available only at certain levels, depending on their usage, summarized data obtained from the upper levels being calculated by processing the data summarized in the lower levels.

Looking at the structure of the B-Tree index, it is easily noticeable that the cost of locating one of the cube's cells represents the sum of the costs associated with locating the last level index block of each sub-tree. The height of such an index is (f.1) [4]:

$$h \le \log_d \frac{r+1}{2} \qquad (f.1)$$

where $h \in N$, d is the number of stored values within an index block, r is the total number of values corresponding to the dimension.

The number of index blocks of a sub-tree containing d elements is:

$$\sum_{l=0}^{h} d^l = \frac{d^h - 1}{2d - 1} \qquad (f.2)$$

where $d=(m-1)$ and h is the tree height.

Considering that the number of items stored in index blocks at the top level of the sub-tree is r, we can calculate the maximum height (h) of the index portion corresponding to a specific dimension, as follows:

$$\frac{2d(d^h - 1)}{d-1} \le r \quad \Rightarrow \quad 2d^h \le r+1$$

$$\Rightarrow \quad h \le \log_d \frac{r+1}{2} \qquad (f.3)$$

where h□N.

The total cost of a search operation within a sub-tree is:

$$c_i = h + 1 \qquad (f.4)$$

Since to locate a cell of the cube is necessary to cross every sub-tree, corresponding to each dimension of the cube, we can calculate the total cost of a queries based on (f.4):

$$c_{ti} = \sum_{j=1}^{n} c_{ij} + (d-1) \qquad (f.5)$$

where n is the number of cube dimensions, c_{ij} is the cost of query sub-tree corresponding to the dimension j and d-1 is the total number of root nodes used as connecting elements of the leaf blocks of sub-trees that have several subordinated sub-tree.

Since OLAP systems incorporate very large data volumes, their performance is affected not only by the query operations cost but also by the index storage space.

Indexes tend to occupy the storage space of the cube and sometimes their size can be larger than the data stored in the cube. If an index occupies a large memory space, it means that the structure is high (number of elements, elements that store too much data, etc.), which increases the index creation time and query execution times.

Using formulas (f.2) and (f.3) it can be calculated the storage space (SS) for a sub-tree indexed using a B-Tree index:

$$S_s = \frac{d^h - 1}{2d - 1} \cdot S_p \qquad (f.6)$$

where d is the number of items stored in a block index, h is the index height and SS is the page size.

Total storage space (St) needed to store a B-Tree index used for indexing cubes is equal to storage space for all its sub-trees. For a cube with three dimensions, the number of sub-trees of a B-Tree index is:

$$N_{si} = 1 + E_a + E_b + 2E_aE_b \qquad (f.7)$$

where N_{si} is the number of sub-trees of dimension i, E_a and E_b represents the elements number of dimension index leaf block corresponding to the other two dimensions.

Based on the (f.2)-(f.7), whole B-Tree index size can be calculated as:

$$S_t = \sum_{i=1}^{j} \left(S_{si} \cdot N_{si}\right) + S_{rn} \qquad (f.8)$$

where j is the cube dimensions number, S_{si} is a storage space needed to store the sub-tree for the dimension i, the N_{si} is the number of sub-trees coresponding to the dimension i and S_{rn} is the size of all nodes connecting the sub-trees.

The query cost of the B-Tree index is the sum of the cost of all sub-trees between the root node and the index leaf block which store the physical address of the cell.

4. The n-Tree Indexing Algorithm

Given the characteristics of cubes, as well as the structure of a B-Tree index, it becomes obvious that this indexing algorithm is not optimized for n-dimensional data structures. Thus, the number of sub-trees within the index is directly proportional with the number of dimensions. As a consequence, the cube is over-indexed resulting in an overconsumption of processing time and storage space.

The proposed n-dimensional indexing algorithm pays attention to the n-dimensional structure of the data. Instead of creating sub-trees corresponding to each dimension and subsequently linking them, it creates only one tree which indexes data simultaneously on all dimensions. As a result, the n-dimensional space is gradually divided into ever smaller n-dimensional subdivisions, until the smallest sub-divisions represent the cells of the cube.

The resulting index has the following characteristics:

- no NULL values are indexed;
- the root node contains at least two subordinated index blocks if it does not coincide with the last level index block;
- each index block contains:
- values from each dimension of the cube; the combination of such values represents a reference point in the n-dimensional space;

The index maintains an ordered list containing unique values corresponding to each dimension of the cube. The values in each list represent a subgroup of the values of the respective dimension. Combining values from each list at a time, we can obtain the data needed to identify the reference points in the n-dimensional space simultaneously minimizing the space required to store them.

Any value corresponding to a dimension from the subordinated index block is smaller than the value of the respective dimension corresponding to the reference point from the upper level index block.

- references to the subordinated index blocks (r_{bs}), corresponding to the reference points (f.9):

$$r_{bs} = \prod_{i=1}^{n} a_i \qquad (f.9)$$

where $a_{1..n}$ represents the number of values from dimensions 1..n stored in the index block;

- *n* references to the index blocks that contain larger values in a dimension than the reference point (one for each dimension).

Thus, an index block contains a total of r references (f.10):

$$r=r_{bs}+n \qquad (f.10)$$

where n equals the number of the cube's dimensions.

- the last level index blocks do not contain any references to other index blocks; instead they store the reference to the physical location of the cube's cells;
- the physical size of an index block is approximately one page.

Each index block stores an ordered list of unique values corresponding to each cube dimensions. Values from each list is a subset of these dimensions. Combining values from each list, one by one, points from the n-dimensional space can be identified, minimizing the needed storage space.

Any value from the dimensions, stored into an index block, is lower than the value belonging to the respective dimension from the reference point of the higher rank index block.

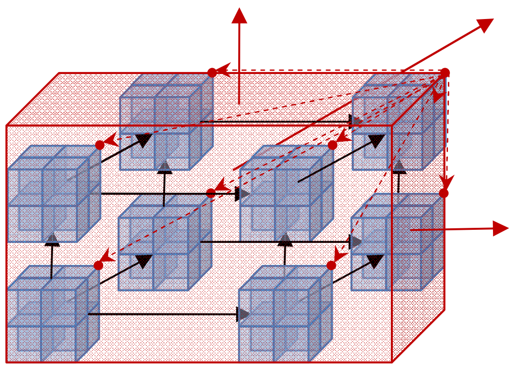


Fig. 3 - The structure of an n-dimensional index corresponding to a three dimensional cube

Because the dimensions values of the reference point are uniquely stored, the n-dimensional space is always a regular space. For a cube with three dimensions, this space is a rectangular parallelepiped, and ideally is a cube.

If some cells do not contain data units (containing null values), they are not indexed, thus reducing the size of the index. The lack of aggregated values corresponding to a cell does not affect the form of the space described by the values stored into a index block.

Every n-Tree index contains a sub-tree that indexes the summarized data which has the following characteristics:

- it contains a sub-tree for each dimension of the cube;

- each sub-tree corresponding to a dimension has a structure similar to a B-Tree index and indexes all the values pertaining to the respective dimension;
- the index blocks of the sub-trees contain references to the lower level index blocks;
- the index leaf blocks do not contain references to other indexing blocks; instead they are the sole elements containing references to the pages where the summarized data are stored;
- each element of the index leaf blocks contains references to parts of the n-dimensional space to which the respective value is assigned

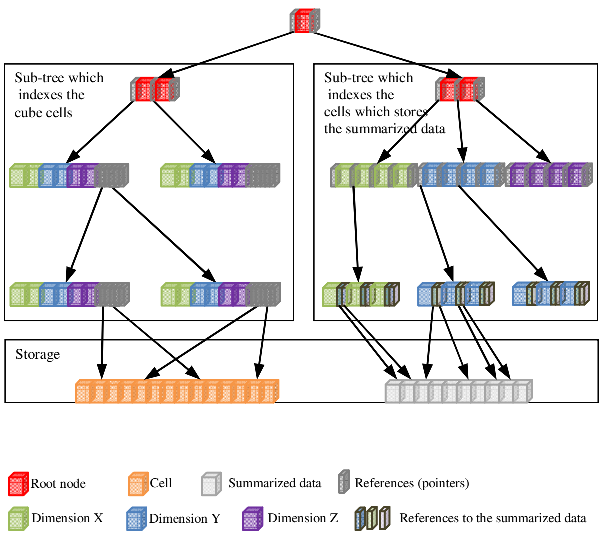

Sub-tree which indexes the cube cells
Sub-tree which indexes the cells which stores the summarized data
Storage
Root node
Cell
Summarized data
References (pointers)
Dimension X
Dimension Y
Dimension Z
References to the summarized data

Fig. 4 - The structure of an n-Tree index for to a three dimensional cube

Thus the number of referenced contained by each element of a leaf index block is:

$$rs=n-1 \qquad \textit{(f.11)}$$

where n is equal to the number of the cube's dimensions.

Summarized data relating to each value stored in the sub-tree corresponding to one dimension are equivalent to a 1 to (n-1) dimensional sub-space. The sub-spaces are distributed among the respective sub-trees, as to avoid storing multiple references to the same summarized data. The dimensions of the index are thus reduced.

For a 3-dimensional cube, the index leaf blocks will contain the following references:

- the elements of the index leaf block of the X dimension sub-tree contain:
- references to data summarized representing the space corresponding to the value of the X dimension, all values of the Y dimension and the first value of the Z dimension (one dimensional space);
- references to the data summarized representing the space corresponding to the value of the X dimension, all values of the Y dimensions and all values of the Z dimension (two dimensional space)
- the elements of the index leaf blocks of the sub-tree corresponding to the Y dimension contain:
- references to the data summarized representing the space corresponding to the value of the Y dimension, all values of the Z dimension and the first value of the X dimension (one dimensional space);
- references to the data summarized representing the space corresponding to the value of the Y dimension, all values of the X dimension and all the values of

the Z dimensions (two dimensional space);
- the elements of the index leaf blocks of the sub-tree corresponding to the Z dimension contain:
- references to the data summarized representing the space corresponding to the value of the Z dimension, all values of the X dimension and the first value of the Y dimension (one dimensional space);
- references to data summarized representing the space corresponding to the value of the Z dimension, all values of the X dimensions and all values of the Y dimension (two dimensional space).

5. Creating an n-Tree Index

To create an n-dimensional index, all data in every index is read and n-dimensional points are created. For each of these points, the following operations are carried out:
- an index block corresponding to an n-dimensional sub-space whose reference point has only values larger than that of the processed point is identified; the index block must also have enough free space to store the values of the corresponding dimensions of the processed point plus a reference;

If such an index block is identified, the values are added to the dimensions' corresponding lists and the reference to the physical location of the cube's cell is stored. Otherwise, a new index block is created by dividing one of the neighboring index blocks.
- when a new index block is created, the values of the reference point, as well as the reference to the parent index block are added, together with references to the neighboring index blocks;

$$S_e = \sum_{i=1}^{j} S_{vi} + S_{ref} \qquad (f.12)$$

where S_e is the element size, S_{vi} is the data type size for the dimension i and S_{ref} is the size of a reference.

This process could propagate itself to the root node. Generally, OLAP systems contain historical data with a low frequency of updating operations but with a large volume of updates. Updating these data also triggers an updating of the cube, and thus, of its index.

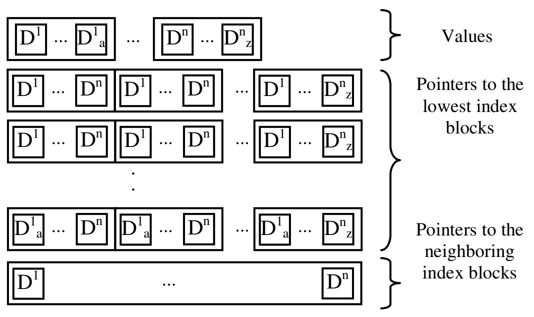
It should be noted that the space required for inserting a new element into a block index is not always the same. If some values of n-dimensional point corresponding to a specific dimension were previously inserted the necessary free space is smaller than the element size.

6. The Performance of the n-Tree Index

The performance of an index depends on its height. A larger height means more physical read operations are needed to identify the cell containing the required data.

The height of the index depends on the size of the index block, the size of the type of indexed data, the number of references to subordinated index blocks stored in each index block, and the number of cells.

By analyzing the structure of an index block (fig. 4), we can compute the number of references it can store.



Values
Pointers to the lowest index blocks
Pointers to the neighboring index blocks

Fig. 5 - Structure of an index block in an n-Tree index

The volume of the stored data in an index block may be written as (f.11):

$$S_d = \sum_{i=1}^{n} a_i \cdot S_v + \left(\prod_{i=1}^{n} a_i + n \right) \cdot S_{ref} \quad (f.13)$$

where S_d represents the volume of the stored data, S_v represents the size of the indexed value, n is the number of dimensions and S_{ref} the size of a reference.

The blocks number of the n-Tree index which contains d elements is:

$$\sum_{l=0}^{h} d^l = \frac{d^h - 1}{2d - 1} \qquad (f.14)$$

where h is the index height.

Ideally, d is the maximum number of the elements which can be stored inside an n-Tree index and its value is up to $\prod_{i=1}^{n} a_i + n$.

The query cost for the n-Tree index is:

$$c_i = h + 1 \qquad (f.15)$$

where c_i is the query cost, h is the index height and 1 is for the index root block.

Since the data in an OLAP system is rarely modified, the best performance is obtained when the index blocks contain a volume of data equal to their size.
Therefore, we can approximate the value of S_d to be equal to that of a page.
We assume that:
- the size of an index block is of 8kB (this is the most common size in current database systems [5]);
- the size of a reference is 6B (the most common size for a local index [6]);
- the size of the data type is 8B (this is the size of the *datetime* type of data);
- each dimension contains the same number of unique values.

Using the formulas (f.3) and (f.15), we can compare the performance of a n-Tree index to that of a B-Tree index (fig 6-11).

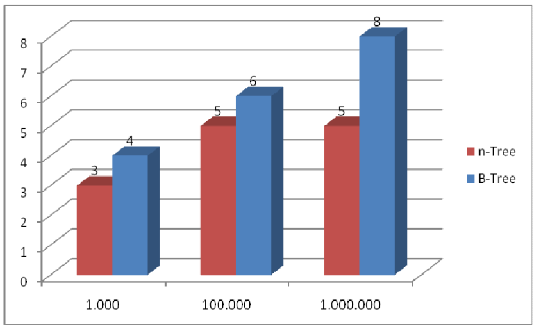


Fig. 6 - The cost of a search operation in a two-dimensional cube

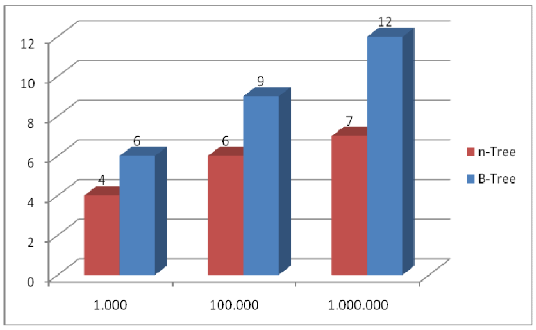


Fig. 7 - The cost of a search operation in a three dimensional cube

Analyzing fig. 6 and fig 7, it can be seen that the n-Tree index query cost is lower the the B-Tree index query cost event for 1.000 cells. The performance difference is event higher when the cells number or the dimensions number increase.

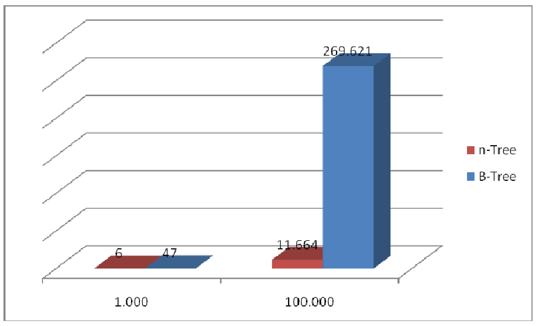


Fig. 8 - The size [in MB] of an index corresponding to a two dimensional cube

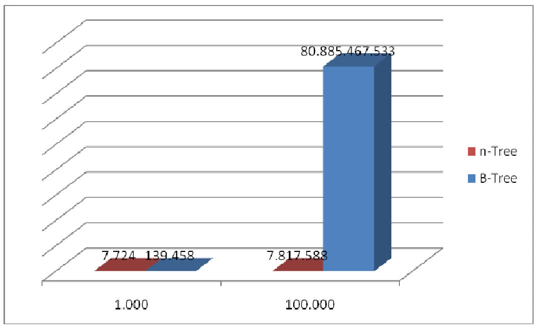


Fig. 9 - The size [in MB] of an index corresponding to a three dimensional cube

The same situation can be observed for the index size in fig. 8 and fig. 9.
The n-Tree index is much smaller than the B-Tree index. The difference comes from the lower number of the index blocks and from the flexibility of creating the summarized data.
When the data summarized data is to be query, the result depends on the location of the location of the summarized data.

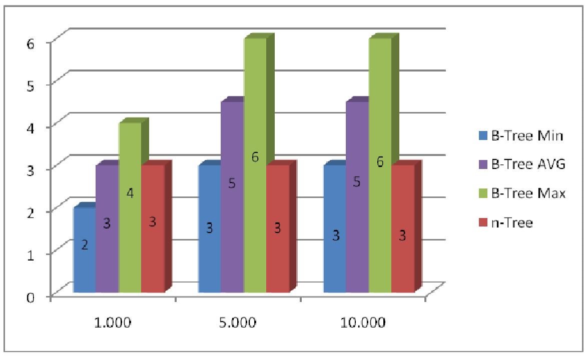


Fig. 10 - The summarized data query cost for a two-dimensional cube

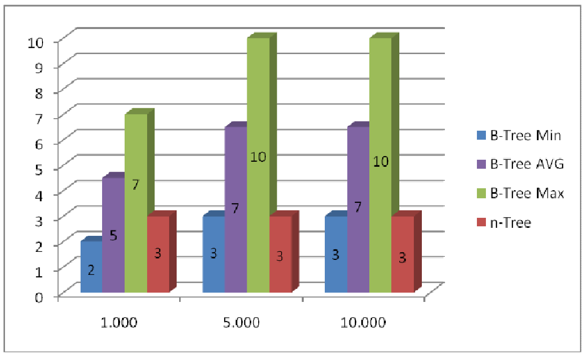


Fig. 11 - The summarized data query cost for a three-dimensional cube

Anyway, in fig. 10 and fig. 11 it can be observed that the n-Tree index query cost is lower than the B-Tree index query cost in any situation, excepting when the cells number is very low.

7. Conclusions

Implementing a new indexing algorithm, with much wider scope and increased flexibility, could be the database systems optimization solution, especially when other indexing algorithms do not provide the desired results.
The n-Tree index could be considered a more generalized B-tree index. If B-Tree index can index only uni-dimesional data, the n-Tree index is optimized for any n-dimensional data. Moreover, the n-Tree index will be more suitable for indexing spatial data.
As shown in figures 5-9, the n-Tree index outperforms the B-Tree index in locating the cells of the cube. Moreover, the difference in performance increases as the number of the cube's cells rises. In addition, the space occupied by the n-Tree index in much smaller than that needed for a B-Tree index. Again the superiority of the n-Tree index is all the more evident when the number of the cube's cells increases.

References

[1] Revista Informatica Economica, nr. 4 (24), 2002;
[2] Revista Informatica Economica, nr. 1 (17), 2001;
[3] Computing Partial Data Cubes for Parallel Data Warehousing Applications, Frank Dehne, Andrew Rau-chaplin, Computational Science - ICCS 2001;
[4] „Ubiquitous B-Tree", Douglas Corner, ACM, 1979;
[5] MCTS 70-431: Implementing and Maintaining Microsoft SQL Server 2005, Que, 2006;
[6] „Index Internals", Julian Dyke, 2005.

Lucian Bornaz is PhD candidate at Academy of Economic Studies since 2006. His research domain is database systems with focus on data indexing algorithms. He graduated Airforce Academy in 1998 and master course at Academy of Economic Studies in 2006.
Lucian Bornaz is certified by Microsoft as MCP, MCSA, MCSE, MCDBA, MCAD and MCTS.