UDC 004.657

INDEXING OF THE SPACE DATA IN THE CSDB MICROSOFT SQL SERVER 2000

N.A. Shestakov

Tomsk Polytechnic University
E-mail: ShestakovNA@ce.cctpu.edu.ru

2 schemes of space indexing are realized in environment of CSDB Microsoft SQL Server 2000. The experimental research of the realized methods for window inquiries is carried out. Comparison of the realized methods with available in the standard means of indexing being in the given CSDB has been carried out. To find the quadrant splitting in methods Z-and XZ-indexing the heuristic algorithm which gives a smaller error of approximation in comparison with standard algorithm is proposed.

Introduction

In last years development of geoinformation systems (GIS) has resulted in an enhanced attention from the side of developers of control systems of databases (CSDB) to work with the space data. Growing requirements of GIS to volumes and reliability of a data storage have resulted in their integration with powerful universal CSDB, as a rule, of independent developers. In this case support of work with the space data (support of space types and space indexing) is required from CSDB.

CSDB MS SQL Server is one of leaders in the market of server CSDB`s. In spite of the fact that the majority of its basic competitors already have even basice means for storage of the space data, they are absent in products from Microsoft (both in MS SQL Server 2000, and in 2005 [1, 2]). In the given work the variant of realization of space indexing in CSDB MS SQL Server 2000 by methods Z-and XZ-indexing which are compared with work of standard indexes is examined. Numerical experiment gives representation as far as it is possible to increase velocity of performance of window inquiries in various conditions.

1. Space CSDB and space indexing

The basic features of space CSDB consist that they support the appropriate types of the data, inquiries in language and mechanisms of indexing.

As examples of space inquiries it is possible to name [3, 4]:

- window inquiries (*window query* or *range query*);
- search of *k* nearest neighbours (*k nearest neighbors query* или *k-NN query*);
- inquiries with space *join* (*spatial join*).

In the given work efficiency of performance of window inquiries is investigated.

In the scheme of work of the space inquiries usually two stages or two *steps of a filtration* are selected. CSDB, having weak space support, develop only the first step (a rough filtration). As a rule, at this stage the approached, approximated representation of objects is used. The most widespread type of approximation – the minimum bounding rectangle (MBR – *Minimum Bounding Rectangle*) [5]. Function of a secondary filtration is assigned to GIS or another client appendix. Working with such CSDB as MS SQL Server, it is quite justified to use representation of objects as MBR at a database level.

A correct use of indexes is decisive factor at optimization of inquiries to a DB. The special methods of indexing, for example, on the basis of *R-trees,* exist for space types of the data [6, 7].

The detailed review of space access methods (*spatial access methods*) is presented in [3]. If the CSDB has no means of space indexing it is possible to realize such indexing by the own forces. These expansions sometimes are named «superstructures», «cartridges» or «plaginas». They are realized by means of triggers and stored procedures in the language used in the CSDB, or external procedures, or by means of an application server (*application server*) if it is not enough built – in language means.

The method of Z-indexing. The method is based on use of a covering curve, or a space filling curve (*the Peano curve*, *space filling curve*). The covering curve splits space into areas (cells), each of them obtains the number (*Peano code*) and, thus, the order is set. Extended objects and inquiry area are approximated by a set of cells (and corresponding Peano codes), the example is shown on Fig. 1. Each cell are corresponded by a range of a possible Peano codes that enables to transform inquiry on area into range inquiry which is optimized by means of a standard one-dimensional index.

Each value of the Peano code can be put in conformity with the quadro-tree unit. Therefore, using a terminology of quadro-trees, we shall name the cells to be ordered as quadrants. *Quadrant* is the space area obtained by recursive division of a plane on 4 equal parts.

XZ-indexing. Shortcoming of Z-indexing at work with not point (extended) objects is that one object is correlated with some Peano numbers, i.e., some indexes. Presence of the relation « one to many « results to that indexes are kept separately from the table to be indexed. It leads to occurrence of additional connection on the index table at performance of inquiry, and the data in the basic table cannot be clustered on a Z-index.

The method of XZ-indexing described in [8], modifies a classical method in such a way that one object is corresponded only by one value of the Peano code. However the approximation error of in such method is higher.

Quadrant splitting of an area. At indexing of objects with the Z-indexing method, it is necessary to present the object as a set of quadrants which are located in a DB as Peano codes. The example of splitting of rectangular area on quadrants is shown on Fig. 1. The number of quadrants depends on the resolution of a base grid (the size of minimal quadrant, defined by the maximal depth of quadro-tree) and the size of the area to be splitted.

At performance of window inquiry, the area of inquiry also is broken on quadrants (for Z-and XZ-indexing methods). The range of the Peano codes corresponds to everyone quadrant. The inquiry is sampling of all objects, which Peano codes fall in these ranges.

Accuracy of representation of object that will be the better, the more is number of quadrants. It can be estimated quantitatively by the approximation error. We shall understand as a approximation error σ a ratio of the total area of quadrants ΣS_q to the area of the figure to be splitted S minus 1.

$$\sigma=\Sigma S_q/S-1=(\Sigma S_q-S)/S.$$

As a rule, there is some reasonable restriction on number of quadrants. At construction of object indexes it is some units, at construction of ranges in window inquiry – hundreds. At this the problem arises to obtain such splitting which would give the least error at the given restriction on number of quadrants. Such splitting we shall consider as optimal.

Algorithm of splitting on quadrants. The algorithm of recursive splitting with restriction of recursion depth is described In [8]. It does not give optimum splitting, but, at least, is better, than simply to split up to achievement of the maximal depth of quadro-tree. The idea of algorithm is to split recursively space up to achievement of some certain recursion depth which is calculated a priori, coming from restriction on allowable number of quadrants $N_{\max}$.

As restriction of recursion depth allows only roughly to come nearer to border of number of quadrants $N_{\max}$ at splitting the result of work of algorithm usually is not optimal.

The author proposed algorithm in which process of splitting is adjusted by some heuristics which defines which quadrantin the current splitting it is necessary to split in the following turn. It too does not find optimal splitting, but the result is enough close to optimum.

Input parameters of the algorithm are: $N_{\max}$ – is maximal allowable number of quadrants in splitting; *Rect* – is rectangular area which is necessary to split on quadrants.

Output parameter of the algorithm is *qlist* – the list of quadrants.

The step-by-step description of its work is given below.

Step 1. To establish for quadrant q the maximal size equal to the size of all the space.

Step 2. To place q in the list *qlist.*

Step 3. If the length of the list *qlist* is equal $N_{\max}$ to end algorithm.

Step 4. If there is no quadrants in the list *qlist*, which can be splitting without overflow of the list *qlist* to end algorithm.

Step 5. To choose from the list *qlist* quadrant q with the maximal value of heuristics.

Step 6. To split q on subquadrants and to place obtained subquadrants which cross the area *Rect*, in the list *qlist.*

Step 7. To pass to **the Step 3**.

The algorithm is greedy since each iteration improves the current splitting and approaches it to final result.

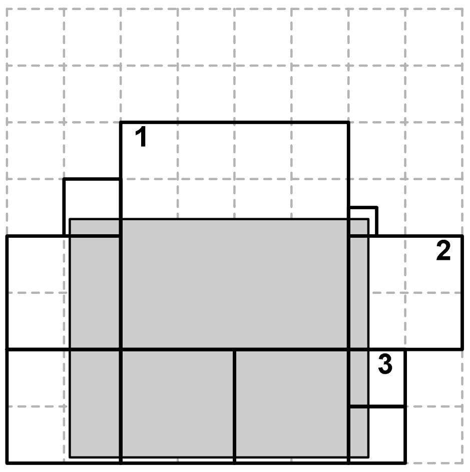

Fig. 1. *Optimal splitting, $N_{\max}$=10*

Sense of counted up heuristics is the area of the space to be reduced at splitting of quadrant, related to number of quadrants on which splitting will increase.

The matter is that not each splitting results in reduction of the approximation area. For example, quadrant 1 (fig. 1) after splitting on 4 subquadrants, will not result in improvement of the current splitting as each of these subquadrants will cross still initial area. However at the further splitting empty quadrants will appear already which will be removed from the list and begin to reduce the approximation error. Quadrant 2 at splitting at once will give reduction of an error though the area of empty space in it is less, than in the quadrant 1. Quadrant 3 at the first splitting also will lead to reduction of the approximation area though this reduction is insignificant. Quadrant 1 is potentially more favourable for splitting, than the quadrant 3 though it is required more splittings. Only it is necessary to check up, is it possible to make such splitting taking into account restriction $N_{\max}$.

Therefore the number of subquadrants N_{suc} on which it is necessary to split the initial quadrant up to achievement of the first «successful» splitting (resulting in reduction of an error) is calculated for each quadrant. This reduction dS (in absolute units of the area) also is calculated. Next квадрант for splitting is chosen with the maximal value of heuristics $E=dS/(N_{suc}-1)$. But at this the necessary condition is, that N_{suc} would not exceed a difference between the current quantity of quadrants and maximal one.

If some quadrants have identical value E the choice occurs according to the maximal area of empty space of the quadrant:

$$S_{empty}=S_q-S(q\cap Rect),$$

Where S_q – is the quadrant area, $S(q\cap Rect)$ – is the area of crossing of the quadrant and the area to splitted.

It was established, that for квадрантов which are crossed by border of area vertically (as quadrants 2 and 3 on Fig. 1) or horizontally (as quadrant1), value N_{suc} depends on size of a gap between area border and external border of the quadrant (the gap corresponds to empty space of the quadrant). If to normalize the quadrant side on unit and to accept d as the gap size it is possible to obtain, that

$$N_{suc}=2\times2^{lb(d)}-2,$$

where lb (d) is the position of the first individual bit in the binary presentation of d (for example, lb (0.1101) =1; lb (0.001) =3). At this

$$dS=S_q/2^{lb}.$$

where S_q is the quadrant area.

For angular quadrants it is possible to take as d the maximal gap size. It can, in principle, lead to not absolutely exact calculation of heuristics, but has no effect for quality of work of algorithm as a whole.

Calculating complexity of heuristic function is low, as only operations of addition/subtraction and bit shifts are used at calculation.

For different values N_{max} the average approximation error s, obtaining as the result of work of algorithms was calculated. Value N_{max} was taken from two ranges: from 4 up to 8 (the characteristic values at building *of the Z-indexes* of objects) and from 400 up to 800 (the characteristic values at splitting of inquiry area to form intervals *of the Z-values*). Results are given in the table.

Table. *The approximation error s for recursive and heuristic algorithms*

N_{max}	Algorithm	
	Recursive	Heuristic
4	3,5	2,4
6	2,8	1,2
8	2,7	0,9
400	0,035	0,019
600	0,024	0,012
800	0,015	0,009

The heuristic algorithm gives the smaller (from 1,5 up to 3 times) approximation error. This difference is essential for small values of N_{max}. When restriction on number of quadrants is enough great, this difference is insignificant, as the value of error itself is small.

As to velocity of work of algorithms, here advantage is on the side of the recursive method – its operating time is asympthotic linear in relation to N_{max}. Labour consummation of heuristic algorithm work is proportional to N_{max}^2. However, the operating time of both algorithms for small values of N_{max} is negligible in comparison with time of the SQL inquiry execution.

2. Space indexing in the MS SQL Server 2000 environment

Problem of a choice of data model for test system. The scheme of the data (objective and relational) is defined by specificity of a problem. As to directly space part various questions concerning representation of the space information can be appeared before a designer. In this case at a choice of the scheme it has been solved to lean on standard OpenGIS Simple Features Specifications For SQL [9] (further in the text the standard OpenGIS) though it was not put the purposes of full conformity to the standard; for the system-prototype to be created it is not required.

Standard OpenGIS defines various data schemes for two language SQL environments: SQL92 and SQL92 with Geometry Types. As the last assumes presence of geometrical types in the language for MS SQL Server it does not pass. For SQL92 two possible variants of realization: with storage of geometrical figure elements and with use so-called «the large binary objects» (BLOB) for storage of geometry are determined. The author chose the second way as it does not limit the geometrical description of figures by primitives, defined by the standard and, on some supervision, is used more often.

Features of realization in the MS SQL Server 2000 environment. Geometrical objects are presented as rectangulars (MBR) which are described in the table by a set of four attributes: x_0, x_1, y_0, y_1. To store additional information on geometry of objects the attribute *data*, having type *image* which is BLOB type in realization of MS SQL Server is foreseen.

For the XZ-indexing method the basic table is expanded with additional attribute – value of a XZ-index (type *int* – 4 bites). In the Z-indexing method indexes are stored in the separate table connected with the basic on an external key.

No attributive information on objects it was stored in experimental DB. It is supposed, that the space and attributive information are divide on different tables, and here we are interested only with a space component.

Method of independent indexes. The given method does not use space indexing. The sense of its use consists in estimation as far as standard means can be worse or better than those which are thought out in exchange.

In the basic table non-cluster indexes (independently on every columns) are built on coordinates MBR. The inquiry in this case represents simple sample on crossing of four ranges. Simplicity of this way is obvious. First, it is not required to build own indexes, second, the data are chosen by the simplest inquiry.

Method of Z-indexing. The input information for algorithm are coordinates of the inquiry window. On output the set of the objects corresponding to the initial window is formed. A sequence of actions is:

Step 1. To transform the inquiry window to the set of Z-value intervals:

1.1. To split the window on quadrants.

1.2. To transform every quadrant to the corresponding interval.

1.3. To execute merge of close intervals from the obtained set.

Step 2. To form and to make SQL-inquiry on the set of intervals.

Step 3. In addition to filter the obtained set of objects, using the information on MBR of objects.

Sampling identifiers of objects from the table keeping Z-indexes (ordered by cluster index) is carried out on the **step 2**, then connection of this list on the basic table with objects is made.

Method of XZ-indexing. The general algorithm coincides with that is used in a method of Z-indexing, only the algorithm of building of Z-values and intervals is modified. Also on **the step 2** connection in inquiry is not carried out as XZ-indexes are kept in the basic table, and on a field of the XZ-index is built the cluster index.

3. Numerical experiment

Conditions of realization of experiment. To carry out experiments with the development environment Borland Delphi 7 it was written the test client appendix. The server part worked under control of SCDB MS SQL Server 2000 (MSDE), started on a uniprocessor computer (Athlon XP 2400) with 512 Mb of memory and IDE hard disk with volume 80 Gb and cash 8 Mb. The following characteristics were used as the basic criteria of an overall performance of window inquiries:

- time of execution of inquiry (*execution time*);
- quantity of accesses to a disk (*disk accesses*).

Measurements of time were made on the client side. To define quantity of disk operations values of global variables @@ TOTAL_READ and @@ TOTAL_WRITE were measured.

Experiment was carried out on 9 sets of the artificially generated data distributed evenly on bidimensional square area. Quantity of objects in the sets was varied: 200 thousand, 600 thousand, 1 million; size of objects was varied, too: point, normal ($5,8\cdot10^{-7}$ of researched space areas) and big ($5,8\cdot10^{-6}$ of space area). Possible combinations of quantity of objects and the size give 9 variants of test data sets.

Measurement of time and quantity of disk operations for window inquiry with use of each of three realized methods was carried out on each data set. The window of inquiry accepted values 0,01, 0,04, 0,2, 1,0 and 5,0 % from the area of all the space (all the space is defined by a possible range of coordinates, in this case, from – 32768 up to 32767 for used type *shortint*, that is a grid 65536×65536). Objects are distributed evenly in space. Thus, the size of a window actually characterizes final selectivity of inquiry.

All values of criterial parameters are obtained by averaging on 25 measurements. The standard deviation from average value in a series was 10...15 % for values of time and less than 5 % for number of disk operations.

Results of experiment and their discussion. Results of experiment for objects of the normal size are given on fig. 2 diagrams for time of performance of inquiries are presented At the left (Fig. 2, *a, b, c*), for number of disk operations – on the right (Fig. 2, *g, d, e*).

By results of experiment it is possible to make the following notes:

1. It is obvious, that with increase of database volume the operating time of all inquiries is increased. At this time in the independent index method and in the Z-indexing method grows faster, than in the XZ-indexing method.

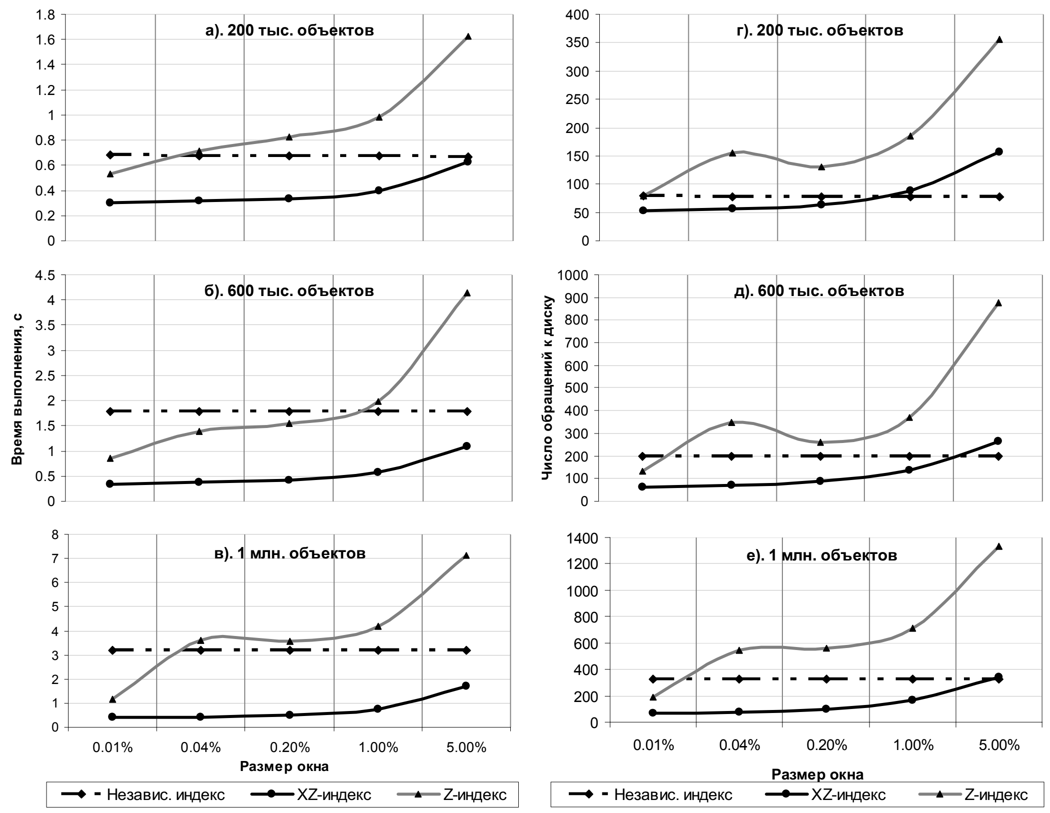

Fig. 2. *Results of experiment (at the left – measurements of time, on the right – measurements of number of disk operations)*

2. Efficiency of inquiry in the independent index method does not depend on the size of a inquiry window (in the range to be researched). Methods Z- and XZ-indexing are much more sensitive to the window size.
3. It is possible to notice not absolutely clear at first sight behaviour of the Z-indexing method, when number of accesses to the disk for the certain value of the window size is too high (as on fig. 2, *d* and *e*) for value of the window size of 0,04 %). The reason of such jump is that at connection of tables optimizator chooses the unsuccessful inquiring plan. If directly to specify to optimizator, what type of the plan to use, it results in reduction of connection efficiency for the small sizes of trhe window.
4. The size of kept objects does not render appreciable influence for the period of performance of inquiries (therefore results only for objects of the normal size are given). Exception is the Z-indexing method which works little bit faster for point objects. It speaks that point objects are not splitted at building of an index, and the table with Z-indexes becomes of the smaller size.
5. At identical values of the disk operations, the operating time of the independent index method is more, than at other methods. Probably, it speaks that this method uses more resources of the processor, than others.

Conclusions on experiment. It is possible to tell, that at certain conditions (enough good selectivity and the sufficiently plenty of objects) use of special way for indexing of the spatce data can give enough appreciable gain of productivity if to use the XZ-indexing method. The Z-indexing method of has proved to be unsatisfactorily even in comparison with use of independent indexes (excepting the case, when selectivity of inquiry is very small – 0,01 %). As a whole, to increase velocity of performance of window inquiries, it is possible to recommend to use XZ-indexing for tables which contain more than 600 thousand records at sampling on a window with the size less than 1 %.

Conclusion

For the test database containing space objects and working under management of SCDB MS SQL Server 2000, methods of space indexing (Z-indexing and XZ-indexing) were realized. Testing productivity of indexing was carried out in comparison with standard means being available in SCDB (the method of independent indexes). Results have shown, that the XZ-indexing method can accelerate essentially work of window inquiries at certain conditions (for example, for inquiry selectivity less than 1 % and quantity of objects more than 600 thousand – more than in 3 times). However at enough large size of the inquiry window or on small volumes of the data the XZ-indexing method can work even more slowly, than standard.

The proposed heuristic algorithm of quadrant splittings provides the smaller approximation error, than standard algorithm at the same given restriction on number of quadrants in splitting.

Literature

1. Francica J. A Spatial Database Technology Update with Dr. Ignacio Guerrero, Intergraph (interview) // Directions Magazine. – 2003. – № 1(Jan), http://www.directionsmag.com/
2. Microsoft SQL Server: Future Plans for Supporting Spatial Data // Directions Magazine. – 2003. – № 10. http://www.directionsmag.com/
3. Gaede V., Gunter O. Multidimensional Access Methods // ACM Comput. Surv. – 1998. – V. 30. – № 2 (June). – P. 170–231.
4. Guting R.H. An Introduction to Spatial Database Systems // VLDB Journal. – 1994. – V. 3. – № 4. – P. 297-308.
5. Papadias D., Theodoridis T. Spatial Relations, Minimum Bounding Rectangles and Spatial Data Structures // Technical Report KDB-SLAB-TR-94-04, http://www.cs.ust.hk/faculty/dimitris/PAPERS/ijgis97.pdf
6. Guttman A. R-trees: A dynamic index structure for spatial searching // Proc. of the ACM SIGMOD Intern. Conf. on Management of Data, 1984. – P. 47–54.
7. Faloutsos C., Kamel I. Hilbert R-Tree: An Improved R-Tree Using Fractals // Department of CS, University of Maryland, Technical Research Report TR-93-19, https://drum.umd.edu/dspace/handle/1903/581
8. Bohm C., Klump G., Kriegel H.-P. XZ-Ordering: A Space – Filling Curve for Objects with Spatial Extension // University of Munich, Computer Science Institute, http://www.dbs.informatik.uni-muenchen.de/Publikationen/Papers/SSD-XZ-Order.final.pdf
9. OpenGIS Simple Features Specification For SQL Rev. 1.1 // Open GIS Consortium, 1999. http://portal.opengeospatial.org/files/?artifact_id=829