EAI Endorsed Transactions
on Context-aware Systems and Applications

Research Article ICST.ORG

Query Optimization in Object Oriented Databases Based on SD-Tree and n-ary Tree

Tran Minh Bao[1,*] , Truong Cong Tuan[1]

[1] Hue University's College of Sciences, Hue University, Vietnam.

Abstract

In this paper, we suggest a new technique to create index helping to query almost identical similarities with keywords in case there is no correct match found. It is based on a SD-Tree and a n-ary Tree helping to query related information when there is no correct match. Index structure arranges signatures according to hierarchical clustering for improving assessment of query. This method is based on technique of using signature file and SD-Tree and signature files are organized according to decentralization to filter unsuitable data quickly and each signature file is saved according to SD-Tree structure for increasing speed of scanning signature. This method helps to decrease effectively search space, so therefore improving effectively complexity of query time.

Keywords: Object-oriented database system, index, signature file, SD-Tree, object-oriented query.

Received on 21 December, 2015, accepted on 11 January, 2016, published on 09 March, 2016

doi: 10.4108/eai.9-3-2016.151114

1. Introduction

Direct query on objects in object-oriented databases costs a lot of data storage during processing query and time to execute query on real data system. The problem is to describe data system in a more simple way and construct a corresponding data structure to reduce searching space during executing query while necessary objects are ensured to be searched.

To reduce space of data query, proposed indexing techniques used to evaluate query in databases [7] have been developed based on binary tree balancing mechanism which was added some special characteristics to reduce tree balance or minimize accesses to data files. These techniques have been developed to increase query speed in object-oriented databases [11, 12, 13]. The main idea is that each SD-Tree on a class in hierarchy is remained but indexes are nested by relation of subclass – target class. Besides indexes in inherited hierarchy structure, many indexing approaches used for nested attribute query have been proposed [2, 3, 4, 8, 10]. Instead of concentrating on inherited hierarchy of classes, researchers have discovered general hierarchy of classes and proposed different index structures following nested attributes [2, 3, 8, 10]… Signature file storage structures will reduce searching space and optimize data query process.

It is necessary to construct a data structure for signature file storage to improve searching. These signature file storage structures can be in form of sequential signature files, sliced signature files, signature tree structure, signature graph structure… where the cost of sliced signature file storage is double of sequential signature files and triple of sequential signature files or more [9]. The main advantage of this approach is its effect in processing new insert and query to parts of word. However, when comparing with indexing based on tree structure, using sequential signature files has 2 disadvantages: (1) they cannot be used to evaluate range query; (2) for each processed query, entire signature files need to be scanned, it makes I/O processing cost increase.

In this paper, we try to improve the second problem to a certain point. Firstly, we organize sequential

*Corresponding author. Email:tmbaovn@gmail.com

signature files in hierarchical structure to reduce searching space during query evaluating process. Next, we store signature files in form of a SD-Tree to execute scanning only one single signature file. If signature file size is large, time saved by this approach is really significant. In fact, this is a B^+-Tree constructed by signature files. Therefore, it can speed up the process of identifying signature position in a signature file. However, in a signature tree, each path is corresponding with one signature identification which can be used to determine its only corresponding signature in signature file. This way helps quickly find out a set of corresponding signatures with query signature.

The remaining of this paper is presented as follows. In Part 2, we provide background. Part 3 proposes indexing technique. Part 4 proposes an approach combining signature files and SD-Tree hierarchy. Finally, Part 5 gives the conclusion.

2. Background

2.1. Attribute Signature

In an object-oriented database, each object is presented by a set of attribute values. Signature of an attribute value is a sequence of hashed-code bits. Given an attribute value, for example the word "student", we decompose it into a string of three-letter sets as follow: "stu", "tud", "ude", "den" and "ent". Then, using hash function h, we map a triplet to an integer k which means kth bit in a string assigned value 1. For example, assuming that we have h(stu) = 2, h(tud) = 7, h(ude) = 10, h(den) = 5 and h(ent) = 11. Then we create a bit string: 010 010 100 110 which is signature of the word.

2.2. Attribute Signature, Signature File

Object signature is constructed by logical OR algorithm for all signatures of attribute values of the object. Below is an example of an attribute signature:

Example 1. Consider an object which has attribute values of "student", "12345678", "professor". Suppose that signature of these attributes is:

010 010 100 110
100 010 010 100
110 100 011 000

In this case, object signature is 110 110 111 110, generated from attribute signatures by using logical OR algorithm. Object signatures of a class are stored in a file, called object signature file.

2.3. Query signature

An object query will be encoded into a query signature together with hash function applied to objects. When a query needs to be executed, object signatures will be scanned and unmatched objects will be excluded. Then query signature is compared with object signatures of signature file. There are three possibilities:

(i) The object matches with the query, i.e., for every bit in query signature s_q, corresponding bit in object signature s is the same, i.e, $s_q \wedge s = s_q$, a real object of query.

(ii) The object does not match with the query, i.e., $s_q \wedge s \neq s_q$;

(iii) Signatures are compared and matching one is found but its object does not match with searching condition of the query. To eliminate this case, objects must be checked after object signatures are matched.

Example 2. This example illustrates the query for object signature in example 1:

Query:	Query signature:	Result:
student	010 000 100 110	successful
john	011 000 100 100	unsuccessful
11223344	110 100 100 000	false drop

Comment: comparing query signature s_q to object signature s is incorrect comparison. That means, query signature s_q matches with signature s if for any 1 bit in s_q, the corresponding bit in s is also 1 bit. However, for any 0 bit in s_q, the corresponding bit in s can be 0 or 1.

2.4. Querying Object-Oriented Databases

In object-oriented database systems, an entity is represented as an object, which consists of methods and attributes. Objects having the same set of attributes and methods are grouped into the same class. Since a class C may have a complex attribute with domain C', a relationship can be established between C and C'. The relationship is called the aggregation relationship. When arrows connecting classes are used to represent the aggregation relationship, an aggregation hierarchy can be constructed to show the nested structure of the classes.

Example 3. An example of a nested object hierarchy:

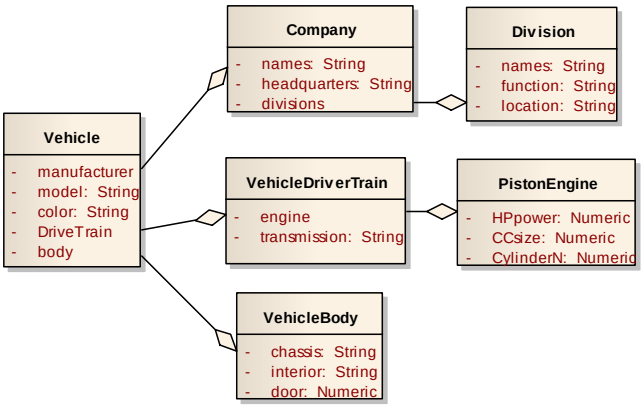

Figure 1. An example of a nested object hierarchy

If an object o is referenced as an attribute of object o', then o is said to be nested in o', and o' is referred as the parent object of o.

In object-oriented databases, the search condition in a query is expressed as a combination of attributes. The attribute may be a nested attribute of the target class.

Example 4. The query "retrieve all red vehicles manufactured by a company with a division located in Ann Arbor" can be expressed as:

select vehicle
where Vehicle.color = "red"
and Vehicle.company.Division.location = "Ann Arbor"

Without indexing structures, the above query can be evaluated in a top-down manner as follows. First, the system has to retrieve all of the objects in the class Vehicle and single out those that are red in color. Then, the system retrieves the company objects referenced by the red vehicles and checks the locations of the divisions of the manufacturers. Finally, those red vehicles made by a company that has a division located in "Ann Arbor" are returned.

2.5. Signature File Hierarchy and Query Algorithm

- *Signature File Hierarchy*

The purpose of using a signature file is to screen out most of the nonqualifying objects. A signature failing to match the query signature guarantees that the corresponding object can be ignored. Therefore, unnecessary object accesses are prevented. In terms of an aggregation hierarchy, a signature file hierarchy can be constructed as follows:

(i) The signature of an object is generated by superimposing the signatures of all its primitive and complex attributes.

(ii) The signature of a primitive attribute is obtained by hashing on the attribute values; the signature of a complex attribute is the signature of the object it references.

(iii) Let C be a class, and let o_1, ...,o_l be its objects; there exists a signature file S such that each o_i(i= 1, ..., l) has an entry <osig, oid> in S.

(iv) Let S_i and S_j be two signature files associated with classes C_i and C_j, respectively. If there exists an arrow from C_i to C_j, then there is implicitly an arrow from S_i to S_j.

Example 5. Signature and signature file hierarchy:

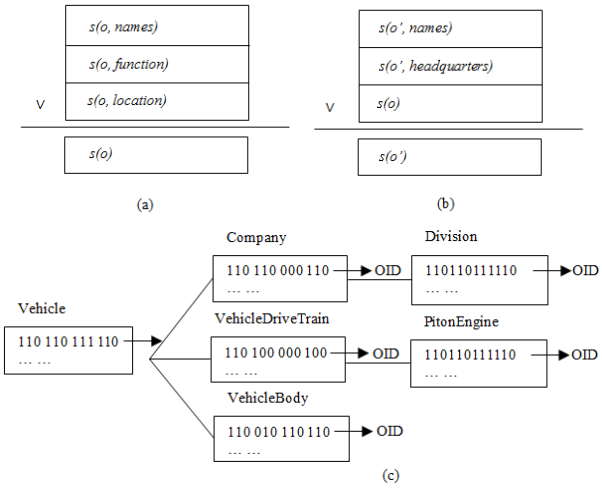

Figure 2. Signature and signature file hierarchy

Consider the class "Division" in the class hierarchy shown in figure 1, which contains no complex attributes. The signature of an object o of this class can be constructed as shown in figure 2 (a), where each s(o, x) stands for the signature produced for the attribute value x of o and s(o) for the signature of o. For a class containing complex attributes, the signature of its objects can be generated in the same way as for a class containing only primitive attributes. The only difference is that the signature of a complex attribute is the signature of the object it references. See figure 2 (b) for an illustration. In figure 2 (b), o' stands for an object of class "Company", and object o of class "Division" is the attribute value of "division" of o'. Signature file hierarchy may be constructed for a database with the schema shown in fig 1 for an illustration in fig 2(c).

- *Query Algorithm Based on Signature File*

Definition 1. (Query tree) [4] Let $p_1 \wedge p_2 \ldots \wedge p_k$ be the search condition in query Q, where each p_i is a predicate of the form: <attribute operator value>. Then, all the paths appearing in the search condition constitute a query tree, denoted as Q_t.

Example 6. Query tree:

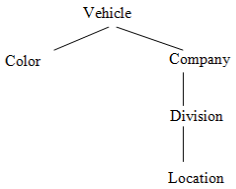

Figure 3. Query tree

Definition 2. (Query signature tree) [4] Let $p_1.p_2 \ldots .p_n$ be a path in a query tree Q_t (from the root to some leaves). Let <$p_i \ldots .p_n$ operator value> be a predicate appearing in the search condition of Q. Then p_n's signature is s_{value}. The signature of a non-leaf node in Q_t can be obtained by

superimposing the signatures of its child nodes. The query signature tree is denoted as $Q_{(s,t)}$.

Example 7. The query signature tree:

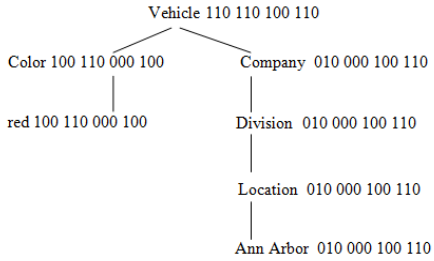

Figure 4. Query signature tree

Use the query signature tree to reduce searching space. For this purpose, two stack structures are needed to control depth-first traversal of tree structures: $stack_q$ for $Q_{(s,t)}$ and $stack_c$ for the class hierarchy. In $stack_q$, each element is a signature, while in $stack_c$, each element is a set of objects belonging to the same class reached during class hierarchy traversal.

Algorithm 1. [5] top-down-hierarchy-retrieval;

Input: an object query Q;

Output: a set of OIDs whose texts satisfy the query.

Method:

Step 1. Compute the query signature hierarchy $Q_{(s,t)}$ for the query Q.

Step 2. Push the root signature of $Q_{(s,t)}$ into $stack_q$; push the set of object OID of the target class into $stack_c$.

Step 3. If $stack_q$ is not empty, $s_q \leftarrow$pop $stack_q$; else go to (7).

Step 4. S$\leftarrow$pop $stack_c$; for each oid_i E S, if its signature $osig_i$ does not compare s_q, remove it from S; put S in S_{result}.

Step 5. Let C be the class to which the objects of S belong; let C_1, ..., C_k be the subclasses of C; then partition the OID set of the objects referenced by the objects of S into S_1, ..., S_k such that S_i belongs to C_i; push S_1, ..., S_k into $stack_c$; push the child nodes of s_q into $stack_q$.

Step 6.Go to (3).

Step 7. For each leaf object, check false drops.

In this technique, optimization is achieved by executing step (4). In this step, some objects are filtered using the corresponding signature in the query signature tree. In step (5), the referenced objects and the signatures of the child nodes of the query signature tree are put in $stack_c$ and $stack_q$, respectively. In step (7), the checking of false drops is performed.

Example 8. Assume that a part of the signature file hierarchy constructed for a database with the schema shown in Figure 1 is of the form shown in the upper part of Figure 5:

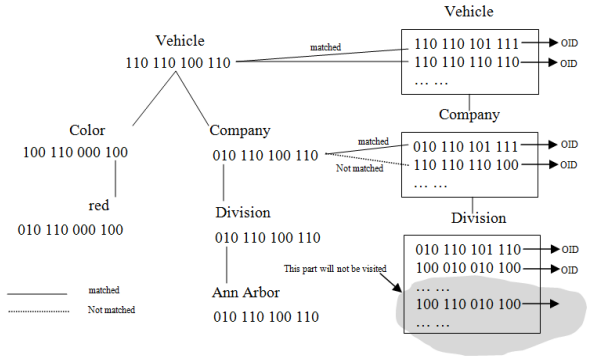

Figure 5. Illustration of query evaluation

Since both the top two signatures in the signature file for Vehicle match the corresponding signature in the query signature tree, the signatures referenced by them in the signature file for Company are further checked. Assume that the first signature in Company is referenced by the first signature in Vehicle while the second one in Company is referenced by the second one in Vehicle. We can see that the second signature in Company does not match the corresponding signature in the query signature tree. Thus, all those Division object signatures referenced by it will not be checked further (see the grey part of Fig 5 for an illustration). This is optimal compared to "top-down-retrieval" since by means of "top-down-retrieval", checking against all Division object signatures has to be performed.

2.6. SD-Tree

- *Overall Structure of SD-Tree*

Indexing technique for Object-Oriented Databases using the dynamic balancing of B^+-Tree is called SD- Tree (Signature Declustering). In this work, the positions of 1s in the signatures are distributed over a set of leaf nodes. Using this for a given query signature, all the matching signatures can be retrieved cumulatively in a single node. Query searching an optimal search path is calculated so that the entire process is speeded up.

Example 9. Overall structure of SD-Tree:

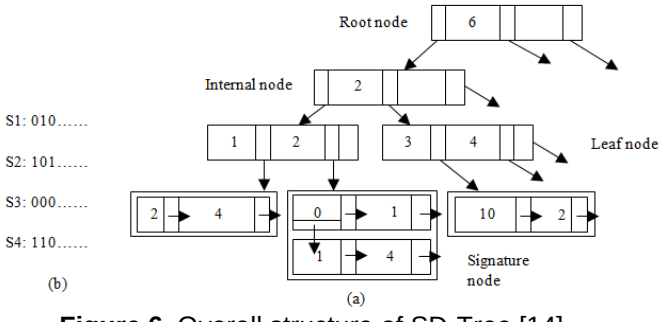

Figure 6. Overall structure of SD-Tree [14]

To process a query signature S_q, the last occurrence of 1, say at position i in S_q is found with the intermediate prefix formed (B). Then, the signature node of ith leaf node is

accessed from root and all signatures with prefix B are retrieved.

Example 10. S_q=011001000101. To find all the matching signatures for S_q the tree is traversed from root and the node values are compared with bit positions of S_q. The last occurrence of 1 in S_q is at position 12. The binary prefix generated for S_q using the position of 1s is 01100100010. A node with the key value 12 is accessed in the signature list of storing 1s in clustered form and all the signatures in the signature list is checked for the prefix value 01100100010. Hence regardless of the bit pattern of S_q, all the matching signatures are returned in a single access.

- *Query Algorithm Based on SD-Tree*

The following algorithm outlines the steps to search for signatures matching a given query signature S_q. In the procedure F ← 0 always and the algorithm lands up directly in the signature node corresponding to last 1 from root.

Algorithm 2. [15] Search(S_q)

Input: The (query) signature to search.

Output: The list of signatures matching the given signature.

Method:

Step 1. Compute the Signature weight for the query signature.

Step 2. If signature weight is greater than 50% then search the query signature in the leaf nodes for the unset bits.

Step 3. Else search the query signature in the leaf nodes for the set bits.

Step 4. Access leaf node.

Step 5. Compare the prefix of Sq.

Step 6. If Found () then read and output the list of signatures.

Step 7. Else report "no matching signatures".

3. PROPOSED INDEXING TECHNIQUE

How is it if query returns zero? In this case, to find out closer match, Hierarchical Clustering is applied. In cluster, similar objects are arranged together to create cluster. Because similar cluster objects will be suitable with any requirements (if any). This thing will increase search speed. This process included 2 step. First of all is cluster and the second is cluster searching. In hierarchical clustering, objects is linked with each other. In here, data structure of n-ary Tree is used for creating cluster.

Algorithm 3. Clustering algorithm

Input: Creating condition

Output: Qualified object is embedded in n-ary Tree

Method:

Step 1. Creating new node and inserting into graph.

Step 2. Seeking object that being content with available condition and attaching new condition.

Step 3. Inserting new node with graph that being content with new condition.

For searching almost identical similarities, locating position of condition collections is provided in hierarchy system and seeking father node of it. For locating position of node, using level order tree traversal. Algorithm to find father node such as follows:

Algorithm 4. getParent

Input: Object Condition collections.

Output: Parent node.

Method:

Step 1. Searching nodes are suitable with conditions.

Step 2. Returning qualified nodes to father node.

After gaining closest match, query is edited and information is retrieved. So therefore hierarchical tree helps for searching almost identical match.

4. Approach Combining Signature File Hierarchy and SD-Tree

4.1. Query Data Structure Model

Direct query on objects in object-oriented databases costs a large space for data storage during query process and a long time to execute query on real databases. To improve this problem, we need to represent data system more simply and construct corresponding data structure to reduce searching space during query executing process while necessary objects are still retrieved by using signature tree. From [5], to optimize the query we need to combine signature file hierarchy with signature tree. This has been shown to improve query time. From [14], query time complexity on SD-Tree is much smaller than signature tree's query time complexity. Therefore, we still use signature file hierarchy as in [5] but replace signature tree with SD-tree to improve query time. Base on theory and suggested algorithms, this paper proposes an approach which combines signature file hierarchy with SD-Tree as follows: (1) all of signature files are organized in hierarchical structure to make it easier for executing stepwise filtering technique; (2) each signature file is stored in form of SD-Tree structure to speed up signature file scanning.

In an object-oriented database, each object is presented by a set of attribute values. Signature of an attribute is a string of hash-encoded bits. Object signature is constructed by overlapping all of attribute signatures of the object. Object signatures of a class are stored in a file, called signature file. Signature files form SD-Tree.

Examle 11. Construction of SD-Tree is illustrated as below:

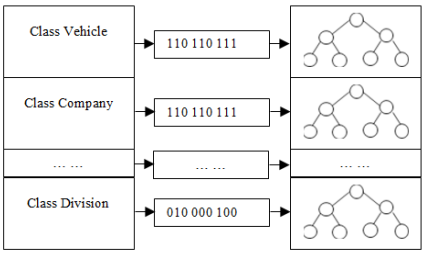

Figure 7. SD-Tree construction

On an object-oriented database, if a class C has an attribute that is composite with domain C', relation between C and C' will be created. This relation is called general relation. When connecting these classes by using arrows to present general relation, a general hierarchy is built to present nested structure of classes. Classes are encoded into signature files and signature files form signature file hierarchy. Each signature file forms a SD-Tree.

Example 12. Combination of signature file hierarchy and SD-Tree is illustrated as follow:

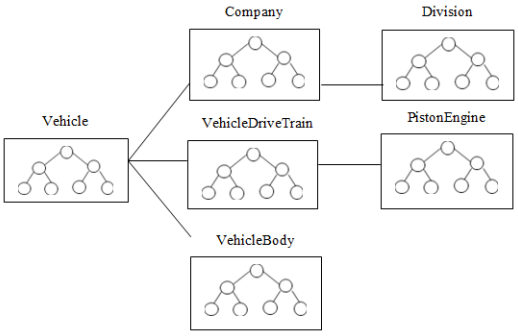

Figure 8. Signature files hierarchy and SD-tree

Data structure is stored entirely in the main memory. In this case, inserting and deleting a signature on SD-tree is executed easily. However, files in databases are usually very big. Therefore, data structure cannot be stored in the main memory but external memory. For object-oriented databases, they will be stored and executed in external memory. An object-oriented database has many classes, each class has many objects. A SD-tree structure will be constructed corresponding with each class, in the same time, each object will form an object signature. The entire object-oriented database will be organized in form of hash table structure including object signatures to execute queries.

4.2. Time Complexity

- *Comparison Of Searching Between Young's Method and Signature File Hierarchy*

In [5], to estimate number of accessed objects in a query, we use two different approaches: (1) Yong's method is proposed in [16]; (2) Top down hierarchy retrieval.

(i) Yong's method

Yong's method, the signature of a referenced object is stored in the referring one. Then, predicate checking can be performed against their signatures before they are accessed. In this way, a lot of I/O operations can be saved.

(ii) Top down hierarchy retrieval

This method has a stronger filtering ability than Yong's method. This is because in each check against a node in a query signature hierarchy, not only is the predicate related to the current node involved, but also some other predicates whose impacts are propagated up several paths to that node. Using the query signature hierarchy, a lot of objects of the target class can also be removed by checking the corresponding signature file, leading to a drastic reduction in the total number of accessed objects.

In [5], we can achieve high performance by means of top down hierarchy retrieval. From an abstract point of view, the query signature hierarchy is a "*global*" filter, while the replication technique developed in Yong's method can be thought of as a "*local*" one. Both reduce the number of objects accessed.

- *Comparison Of Time Complexity Between Signature Tree and SD-Tree*

(i) Signature tree method

In [14], time complexity for inserting in a signature tree is $O(nF)$, where n is number of file's signatures and F is length of signature including 0 bit and 1 bit. With signature tree, tree's height is limited by $O(\log 2n)$, n is number of leaf nodes. Average cost of searching signature tree is $O(\lambda.\log 2n)$, where λ is number of visited paths.

(ii) SD-Tree method

In [14], SD-tree is used as an index structure for set of large data, small F value reduces time of constructing SD-tree. Inserting time complexity is limited by $O(n.m)$, where n is number of signatures in the file and m is the number of 1 bits in a given signature. Another useful characteristic of SD-tree is that with higher F value, tree's height can be small by changing p, h value, to speed up searching which is limited by $O(\log p(F/p-1))$. Searching time for a query with a set of bits at the ith position which is total of time for access to leaf node (T_{li}) and time for searching signature node (T_{si}) is calculated as follows:

$T_s=T_{li}+T_{si}$.

T_{li} does not change for any leaf node in an active balance structure like SD-tree and T_{si} increases when value of i increases. Therefore, searching time is limited by $O(T_{li}+2^{i-1})$.

Comparing time complexity of signature tree $O(\lambda.\log 2n)$ and of SD-tree $O(T_{li}+2^{i-1})$, it is clear that value T_{li} is much smaller than value λ, it is also an advantage of SD-tree.

4.3. Object-Oriented Query Processing

To execute a query of an object in an object-oriented database, firstly we have to change an object-oriented database into data structure as above. We do:

Step 1. Attribute of the object is hashed into binary signatures and attributes which form object signatures.
Step 2. Object signatures in a same layer will form SD-Tree.
Step 3. Create signature file hierarchy where each file is a SD-Tree.
After having data structure for query, we execute object query process on object-oriented databases as follow:
Step 1. Encode key words which need to be retrieved into binary signature.
Step 2. Execute key word signature query to determine classes which need to be searched.
Step 3. Execute key word signature query on SD-Tree corresponding with determined classes.
Step 4. In case exact match is unavailable we change to step 5. Opposite, turning to step 6.
Step 5. Finding closest match.
Step 5.1. Using Clustering Algorithms on n-ary Tree to create new requirement.
Step 5.2. Finding out information is suitable with new requirement.
Step 5.3. Seeking match of son node and returning corresponding father node.
Step 6. Updating information in database.

5. Conclusion

In this paper, we suggested technique for creating new index used for Object-oriented database system to find out suitable match when exact match is unavailable. This approaches is hierarchical combination between signature and SD-Tree. Plus, in case exact match is unavailable, n-ary Tree is used for locating corresponding coincident position. To optimize decentralized scanning object, we are based on decentralization of signature files to decrease tree branch. However, because signature file only works as an incorrect filter, it's impossible to be arranged or implementing binary search so we can't use it to increase speed of signature file scanning process. So therefore, we suggest to create a SD-Tree on signature file with role such as a node in hierarchical signature file. This technique can avoid to search orderly helping for decreasing needed time to search on signature file.

References

[1] S. Sung and J. Fu, (1996), Access Methods on Aggregation of Object-Oriented Database. *IEEE International Conference*, Vol (2), pp.977-982.
[2] Bertino, (1990), Optimization of queries using nested indices, *in Proceedings of International Conference on Extending Database Technology*, pp. 44-59.
[3] Bertino and C. Guglielmani, (1992), Optimization of object-oriented queries using path indices, *in 2nd International Workshop on Research Issues on Data Engineering: Transaction and Query Processing*, pp. 140-149.
[4] S. Choenni, E. Bertino, H. M. Blanken, and T. Chang, (1994), *On the selection of optimal index configuration in OO databases, in Proceedings of 10th International Conference on Data Engineering*, pp. 526-537.
[5] Yangjun Chen, (2004), Building Signature Trees into OODBs, *Journal of Information Science and Engineering*, 20(2), pp. 275-304.
[6] Dervos, Y. Manolopoulos, and P. Linardis, (1998), Comparison of signature file models with superimposed coding, *Journal of Information Processing Letters*, Vol. 65, pp. 101-106.
[7] R. Elmasri and S. B. Navathe (1989), Fundamentals of Database Systems, *Benjamin Cumming*, California.
[8] Fotouhi, T. G. Lee, and W. I. Grosky, (1991), The generalized index model for object-oriented database systems, *in 10th Annual International Phonix Conference on Computers and Communication*, pp. 302-308.
[9] Y. Ishikawa, H. Kitagawa, and N. Ohbo, Evaluation of signature files as set access facilities in OODBs, in Procreedings of ACM SIGMOD International Conference on Management of Data, 1993, pp. 247-256.
[10] W. Kim, K. C. Kim, and A. Dale, (1989), Indexing Techniques for Object Oriented Databases, *Addison Wesley*, pp. 371-394.
[11] Kemper and G. Moerkotte, (1992), Access support relations: an indexing method for object bases, *Information Systems*, Vol. 17, pp. 117-145.
[12] C. C. Low, B. C. Ooi, and H. Lu, (1992), H-trees: a dynamic associative search index for OODB, *in Proceedings of 1992 ACM SIGMOD Conference on the Management of Data*, pp. 134-143.
[13] Sreenath and S. Seshadri, (1994), The hcC-tree: an efficient index structure for object oriented database, *in Proceedings of International Conference on Very Large Database*, pp. 203-213.
[14] I.E. Shanthi, R. Nadarajan, (2009), Applying SD-Tree for Object-Oriented Query Processing, *Informatica (Slovenia)*, 33(2), 169-179.
[15] Ms. Ankita Thakur, Ms. Meena Chauhan, (2012), Optimizing Search for Fast Query Retrieval in Object Oriented Databases Using Signature Declustering, *International Journal of Engineering Research and Development*, pp. 46-50
[16] S. Yong, S. Lee, and H. J. Kim, (1994), Applying signatures for forward traversal query processing in object-oriented databases, *in Proceedings of 10th International Conference on Data Engineering*, pp. 518-525.