AN INDEXING METHOD FOR HANDLING QUERIES ON SET-VALUED ATTRIBUTES IN OBJECT ORIENTED DATA BASES

I. Elizabeth Shanthi
Dept of Computer Science
Avinashilingam University for Women, Coimbatore, India.
shanthianto@yahoo.com

R. Kirutthika
Dept of Mathematics and Computer Applications
PSG College of Technology, Coimbatore, India
kirutthikaraja@gmail.com

R. Nadarajan
Dept of Mathematics and Computer Applications
PSG College of Technology, Coimbatore, India
nadarajan_psg@yahoo.co.in

ABSTRACT

We propose a signature-based indexing method for object-oriented query handling in this paper. Signature file based access methods initially applied on text for their filtering capability have now been used to handle set-oriented queries in Object-Oriented Data Bases (OODBs). All the proposed techniques use either search methods that take longer retrieval time or tree based intermediate data structures that traverse multiple paths thus making comparison process tedious. In this paper a B+ tree based structure called Signature Declustering (SD) tree applied on set-valued attributes is presented. We apply and study the behavior of subset-superset queries of OODBs on SD-tree. It is observed that SD-tree which is retrieving all matching signatures in a single access handles both types of queries efficiently. Also the maintenance cost and query response time of SD-tree is substantially reduced thus making it an efficient structure for handling Object Oriented queries.

KEYWORDS
Signature file, Objected oriented query handling, Information retrieval, subset-superset queries.

1. INTRODUCTION

Object Oriented Data Base Systems (OODBSs) have been widely used by the database research community for the past two decades due to their efficient modeling of data, various navigational patterns in database etc. Many experimental and commercial systems such as Gemstone, Orion and O_2 have been built [17]. OODB offers more powerful modeling capability than Relational Data Base Management Systems. Hence OODB management becomes a prime issue to address as the volume of data increases everyday in all computer-based applications. To derive useful information from large databases indexing plays a vital role. Indexing that evolved with the concept of database management finds an extensive analysis and applications in the literature. One such method called the signature file approach is preferred for its efficient evaluation of set-oriented queries and easy handling of insert and update operations. Initially applied on text data [2] it has now been used in other applications like office filing, relational and Object-Oriented Databases [5, 17, 19] and hypertext. Signatures are hash coded abstractions of the original data. It is a binary pattern of predefined length with fixed number of 1s. The attributes' signatures are superimposed to form object's signature. To resolve a query, the query signature say Sq is generated using the same hash function and compared with signatures in the signature file for 1s sequentially and many non-qualifying objects are immediately rejected. If all the 1s of Sq match with that of the signature in the file it is called a drop. The signature file method guarantees that all qualifying objects will pass through the filtering mechanism; however some non-qualifying objects may also pass the signature test. The drop that actually matches the Sq is called an actual drop and drop that fails the test is called false drop. The next step in the query processing is the false drop resolution. To remove false drops each drop is accessed and checked individually. The number of false drops can be statistically controlled by careful design of the signature extraction method and by using long signatures [2]. Different approaches have been discussed by researchers to represent Signature file in a way conducive for evaluating queries, such as Sequential Signature File (SSF), Bit-Slice Signature file (BSSF), Multilevel Signature file, Compressed Multi Framed Signature file, S-Tree and its variants, Signature Graph and Signature tree[18]. An example of sample query evaluation using signatures is shown below.

Information	0010 0100
Retrieval	0100 0001
Block Signature	0110 0101

Sample queries

Matching query	
Keyword = Information	0010 0100
Query descriptor	0010 0100
Block signature matches	(Actual Drop)

False Match query

Keyword = Coding	0010 0001
Query descriptor	0010 0001
Block signature matches but keyword does not	(False Drop)

Non-matching query

Keyword = Information	0010 0100
Keyword = Science	0000 0110
Query descriptor	0010 0110
Block signature does not match	

In this paper we study the advantages that could arise from representing signature file in a structure called SD-tree [6] and applying it on set-valued attributes. In SD-tree all the set bits in a signature are distributed over various signature nodes and all the signatures having a common set bit are clustered together so that query processing is fast and brings all output signatures in a single access.

The rest of the paper is organized as follows. Section 2 discusses briefly discusses the background information in this track. Section 3 is devoted to SD-tree structure and tree updates. Section 4 explains the algorithms for handling superset /subset queries using SD-tree. Section 5 reports the results of the experiments conducted. Finally section 6 concludes the work with an outlook on future work.

2. BACKGROUND

This section discusses set-valued attributes in OODBSs and some of the previous work on signature file based approach for OODB query handling.

2.1 SIGNATURES IN OODB QUERIES

Today, most of the stored data consists of records with set-valued attributes. Since OODBSs handle efficiently multi-valued attributes they are widely used. The basic query for this kind of data is the inclusive or partial match query that retrieves all records containing specific attributes. Inclusive queries are divided into subset and superset queries [5]. Subset queries retrieve objects with set-valued attributes that contain the query set. In superset query, objects whose set-valued attribute are contained in the query set are retrieved. For example consider the object schema partially shown for a University database in Fig 1.

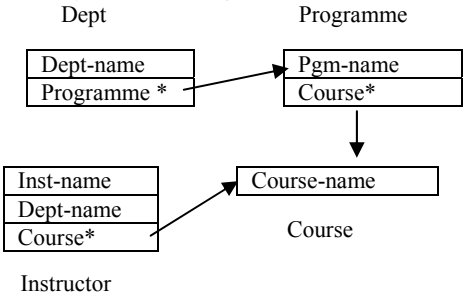


Fig 1. Sample object schema

The classes with set-valued attributes are Dept, Instructor and Programme. Every object of Dept has a list of Pgm-name offered in its attribute Programme. Similarly every Programme object has a list of Course-name and each Instructor teaches one or more Course-name denoted in the attribute Course. For any query signature O' and an object O satisfying the query, subset query searching is denoted as

$$O' \subseteq O \Rightarrow q \subseteq s \quad \text{and for superset query} \quad O' \supseteq O \Rightarrow q \supseteq s.$$

The right hand side of both equations denotes that the corresponding query (q) and object signature (s) also satisfy the same.

2.2 RELATED WORK

Research reports on the evaluation of queries with set-valued predicates are few. Some indexes particularly for object-oriented and object-relational systems like nested index [1], path index [1], multi index [11], access support relations [9] and join index hierarchies [20] have been analyzed. With the exception of signature files [2, 19] and RD Trees [8] the problem of indexing data items with set-valued attributes has been ignored by the database community [16]. Signatures are preferred to encode sets for three reasons [16]. First they are of fixed length and hence convenient for index structures. Second set comparison operators on signatures can be easily implemented by efficient bit operations. Third signatures are more space efficient compared to the conventional set representation. Some of the previous reports on OODB query handling using signatures are listed below. In order to improve selectivity in the upper level of signature-based tree structures Tousidou et. al [4] propose methods like linear split, quadratic split, cubic split etc. In [5] the variation of a new method that combines linear hashing and S-tree is proposed to handle subset-superset queries. Various schemes for evaluation of queries having nested predicates have been discussed in [7, 14, 17]. Norvag [10]

shows how logical object identifiers stored in an index structure reduces average object access cost. A trie tree superimposed on an inverted file used as an index for subset-superset query evaluation is discussed by Terrovitis et al [12]. Nikos [13] experiments various join types on set-valued attributes and infers that signature-based methods are appropriate for set equality joins rather than inverted files. Helmer et.al [15] analyzes Nested-loop and Signature-Hash algorithms for the evaluation of subset predicates joins. Paper [19] studies the subset-superset query handling in OODB using signatures in SSF, BSSF and NIX. In this paper we go in a different line for handling OODB inclusive queries using an intermediate structure called SD-tree that proves novelty over the other methods proposed recently. In the next section we describe the structure of SD-tree [6] and list the algorithms used for tree updates.

3. THE STRUCTURE OF SD-TREE

SD- tree has three types of nodes:

- Internal nodes
- Leaf nodes
- Signature nodes

The internal nodes form the upper tree and leaf nodes at the last but one level as in B+ tree. The signature nodes are at the bottom level of the SD-tree. We will now explain the structure of the nodes in detail. To make discussion simple, we assume the tree order as 3 for a signature file with 8 block signatures of length 12.

3.1 STRUCTURE OF NODES

An internal node of SD-tree is illustrated in Fig 2 in which pointers and keys alternate each other as in B+ tree. For a tree of order 3 the internal node has two keys K1 and K2 and three pointers P1, P2, P3. These pointers are tree pointers pointing to the nodes at the lower level. While searching, the left tree pointer is followed for values less than or equal to the node value, else right pointer is followed.

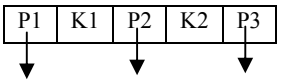


Fig 2. A typical structure of an internal node

The leaf nodes appear in the last but one level of the SD tree. Like B+ tree all the key values appear in ascending order of their values in the leaf nodes and are connected to promote sequential search. But unlike B+ tree in SD-tree each value is followed by a signature node instead of data pointer. This is shown in Fig 3. Pointers P1 and P2 point to the corresponding signature nodes for K1, K2; P3 is the forward pointer to next leaf node and P4 is the backward pointer from next leaf node. These pointers help in optimal signature insertion and selection. The structure of a signature node is shown in Fig 4. The signature node for K_i has 2^{i-1} binary combinations denoting the possible prefixes.

3.2 OVERALL STRUCTURE OF SD-TREE

Consider the partially filled SD-tree shown in Fig 5(a). The tree has been constructed for signature length F = 12. It is obvious that a tree of order 3 has height 2. Consider the signature file in Fig 5(b). To insert signature S1 with first occurrence of 1 at position 2, access the leaf node with value 2, follow its signature node and write the signature value as 1 (for S1) with the prefix 0 (for bit 1). In the same way S2 is inserted in signature node 1 with no prefix and signature node 3 with prefix 10 (for bits 1 and 2). S4 will be inserted at signature node 1 with no prefix and signature node 2 with prefix 1.

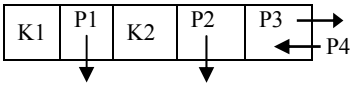
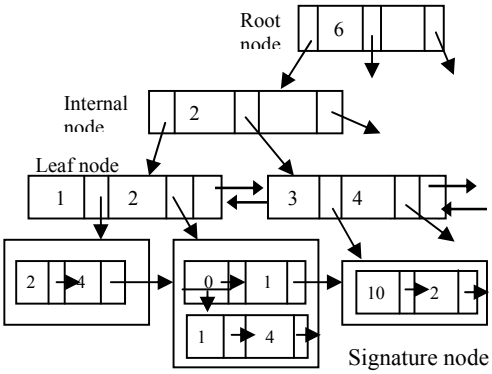


Fig 3. A typical leaf node entry

B1	List of Signatures having prefix B1
B2	List of Signatures having prefix B2
. .	. .
Bn	List of Signatures having prefix Bn

Fig 4. A typical signature node entry

While inserting a signature there are two possible options to move to next 1s position in the leaf node. One is via the sequential pointer between leaf nodes and the other is the top-down traversal of tree from root. To ensure optimal search path the threshold (Th) is fixed at h+1 (h being the tree height plus one more level for accessing signature nodes). As long as the number of sequential pointers traversed in leaf nodes is within the specified Th value the sequential pointers are followed. This is calculated by finding the difference (d) between two consecutive 1s in a signature divided by the number of entries per leaf node. This is given by the condition $d / (p-1) \leq Th$, where p is the order of the tree. The division here is integer division. When the above condition is not true, a new tree access is initiated from root. Similarly to promote optimality between queries the difference between the last set bits of two consecutive queries is compared with threshold and leaf node's forward / backward pointers are used accordingly. The procedures for tree maintenance are explained in the following section.



(a)

S1 : 010 …….
S2 : 101 …….
S3 : 000 …….
S4 : 110 …….

(b)

Fig 5. Sample SD-tree

3.3 TREE SEARCH AND UPDATES

This section lists briefly the algorithms for signature insert, delete and search operations on SD-tree [6]. A global flag F is set to 0 indicating the search path from the root of the tree by default. In the procedure after the first 1's insertion, depending on the d value F may be set to 1 indicating the search path through sequential pointers of leaf nodes.

3.3.1 Insertion

The algorithm for signature insertion is outlined in this section.
Insert (S_u)
Input : The signature to insert S_u ;

1. Move the set bit positions of S_u in a queue.
2. For each value i in queue do
   Begin
   Access leaf node i;
   Write the sig. no. with the prefix;
   End.

3.3.2 Searching

The following algorithm outlines the steps to search for signatures matching a given query signature Sq. In the procedure F ← 0 always and the algorithm lands up directly in the signature node corresponding to last 1 from root.
Search(S_q)
Input : The (query) signature to search.
Output : The list of signatures matching the given signature.

1 Let i be the last set bit of S_q.
2 Access leaf node i.
3 Compare the prefix of S_q.
4 If Found() then read and output the list of signatures.
5 Else report " no matching signatures".

3.3.3 Deletion

The algorithm to delete a signature from SD-tree is described below.
Delete (S_u)
Input: S_u, the signature to delete.

1. Let $i_1, i_2, \ldots. i_n$ be the positions of 1 in S_u.
2. For each i_k ($1 \le k \le n$) form prefix B as in Insert (S_u).
3. Access the leaf node and follow the signature node;
4. Access prefix B and search for u.
5. If present, delete it.
6. Repeat steps (2) through (5) for all i_ks.

On the whole SD-tree maintenance is very simple because insertions and deletions are reflected only in signature nodes. Further by varying the order of the tree p, the tree height can be made minimal. This results in shorter tree suitable for fast query processing regardless of the size of the signature file.

4. QUERY ALGORITHMS

The following section explains the algorithms for processing subset-superset queries using SD-tree.

4.1 ALGORITHM SUBSET

Subset query processing search for all signatures whose set bit positions match with that of query signature. In SD-tree based retrieval the output is the common elements of all signature nodes at the set bit positions of query signature. The steps are as follows.

1. Let Sq be the given query to search for.
2. Let i_1 , i_2, … i_n be the positions of set bits in Sq.
3. Move i_1 , i_2, … i_n into a queue. Let j = 1.
4. While queue not empty do
   Begin
   Read i_x from queue.
   Access leaf node i_x ->sig.node;
   Let Sj = Set of all sig. no.s in the sig. node;
   j = j+1;
   End.
5. Result = S1 ∩ S2 ….∩ Sn.
6. {Result} has all signatures matching Sq.

4.2 ALGORITHM SUPERSET

Superset query contains the maximum number of set bits to search. Hence, the algorithm finds all signatures whose set bits form subsets of the query. The steps in the algorithm are given below.

1. Let Sq be the query signature.
2. Let n be the number of attributes superimposed to form Sq .
3. Move all possible subsets (2^n -1) (superimposed signatures of attribute combinations except null set) into the queue. Let i = 1;
4. While queue not empty do
   Begin
   Read signature x from queue;
   Access sig. node of last set bit and let Sn be the nth prefix in signature node compared with Sq.
   While x->sig.node not empty do
   Begin
   If Sq.prefix ∩ Sn.prefix = = Sq.prefix then
   Si = Set of sig. no.s pointed by Sn.
   i = i+1;
   End.
5. Result = S1 ∩ S2 ….∩ Sn.
6. {Result} has all signatures matching Sq.

4.3 EXAMPLE:

For the schema depicted in Fig 1 sample signatures and query evaluation is given below.

Query 1 : List of programmes offering atleast Comp-applns and Applied Maths in Courses

Query element	Signature
Comp.applns	0010 1000
Applied Maths	1000 0001
Query signature	1010 1001

Class : Programme			
Pgm-name	*Courses*	*Object signature*	*Output*
1. M.E	Comp.applns, Applied Maths	1011 1101	Actual drop
2.M.Sc(S.E)	Software engg, Comp. engg	1100 1101	Does not match
3.M.Sc(Phy)	Comp.applns,Nuclear physics	0111 1000	Does not match

Consider now a query on class programme with set-value 1. For example "course = Comp.applns" (0010 1000) the output is objects 1 and 3 are actual drops and object 2 does not match.

Query 2 *:* All instructors handling *none other than* Applied Maths and Calculus

Query element	Signature
Applied Maths	1000 0001
Calculus	0010 0100
Query signature	1010 0101

Class : Instructor				
Inst-name	*Dept-name*	*Courses*	*Object signature*	*Output*
1. John	Comp.sc	Software engg, Comp. applns	1110 1011	Does not match
2.Adams	Mathematics	Applied Maths, calculus	1010 1101	Actual drop
3. James	Comp.sc	Software engg	0100 1001	Does not match
4. Janes	Physics	Nuclear physics	0111 0010	Does not tmatch

For superset single valued set query like "none other than Software engg" (0100 0001) object 1 is a false drop, objects 2 and 4 do not match and object 3 is actual drop. Due to the abstraction incurred in forming signatures the false drop occurs for lower set cardinality. For higher cardinal values false drop ratio drops considerably.

5. PERFORMANCE ANALYSIS

The performance measures used in signature-based methods are different- some methods use the number of disk accesses while others use signature reduction ratio to speed up the retrieval time [3]. In this section we show the benefits obtained by using SD-tree over Signature tree in 5.1. Then in 5.2 we discuss the time and space complexity of SD-tree-based-retrieval for subset-superset queries.

5.1 SIGNATURE TREE VERSUS SD-TREE

In this section the parameters which are generally considered in the analysis of indexing structures like time and space complexities are reported. We compare the results of Signature tree [18] with that of the SD-tree. The observed results are listed in Table 1.

Parameter	Signature tree	SD-tree	Improvement
Time complexity	O(nF)	O(nm)	m < F Shorter tree
Tree height	$O(\log_2 n)$	**$O(\log_p (F/(p-1))$**	p>2 ; Shorter tree
Search cost	$O(\lambda.\log_2 n)$	$O(\log_p F+a)$	F < n; Cost< Sig. tree

Table 1: Signature tree Vs SD-tree

In Table 1,
n - Number of signatures in signature file
F – Length of signature
m - Number of set bits
p – Order of SD-tree
λ – Number of path traversed in query searching
a – Average no. of signatures / signature node

5.2 PERFORMANCE OF SD-TREE-BASED-RETRIEVAL

To run queries we implement SD-tree in Java and for every test run the tree is constructed statically before signature insertion. The parameters considered in the experiments' data sets are Signature length (F), and the no. of set-value attributes considered in the query (s). The experiments were carried out in a standalone system with Intel Pentium IV processor. The main memory size is 512 MB and the hard disk capacity is 80 GB. We consider abstract data set having 60000 objects fixed for various signature lengths.

5.2.1 Time Complexity

Like in other signature applications we use the response time as the performance measure. The time complexity of handling subset queries depend on the average no. of set bits (m) in the query signature. The search time within the signature node of set bits depend on the average number of entries (a). Hence, the complexity of subset queries is bounded by O(ma). The number of values in the set is varied from 2 to 4 for signature lengths 8, 12 and 16 and the values are plotted in Fig 6. In order to improve selectivity of signatures for varying number of attributes in the query, signature length was varied to keep false drop probability at minimum. It is obvious that the number of values of the set-attribute considered in the query definitely increase the set bits in the output signature increasing the searching time.

Otherwise the time variations are only approximate as signatures are mere abstraction of original attribute values and superimposing attribute values further vary the number of set bits in the output object signature.
For superset queries the algorithm considers the subsets of the values of the set-attribute (except null set) that vary from 2 to (s). Obviously the time taken for query searching increases with (s). This is shown in Fig 7. Searching within the signature node proceeds as in SD-tree but for $(2^s - 1)$ subset values. It is apparent that the time taken for query searching increases with the constituent signatures of (s). If w = {subsets of s}, then the time complexity is bounded by O(wa). This value is substantially less compared to the signature tree search cost that is dependent on the signature file size N.

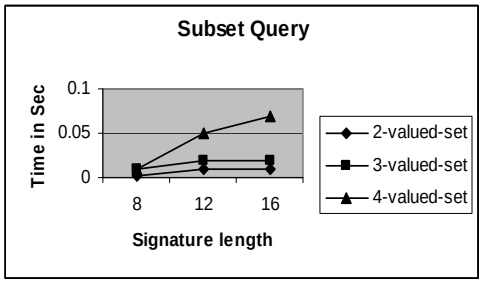


Fig 6. Subset Query

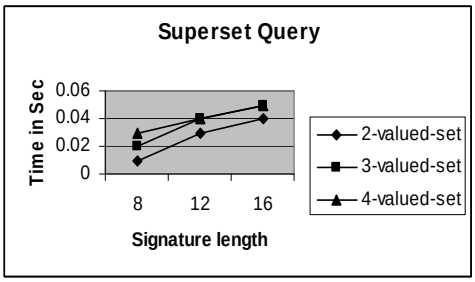


Fig 7. Superset Query

Tree maintenance and space overhead
Tree maintenance is same as that of SD-tree [6] which means that the process of handling subset-superset queries is incorporated in the algorithm steps which do not affect the tree structure. Fig 8 shows the space overhead of SD-tree measured for various signature weight distributions.

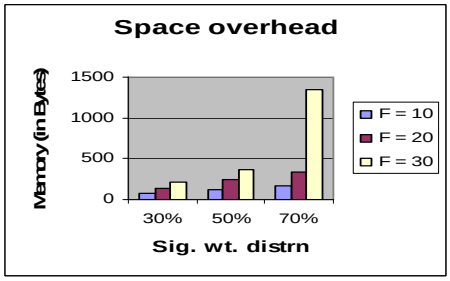


Fig 8. Space overhead of SD-tree

6. Conclusion and future research directions

In this paper we presented a novel way to represent signature file called SD-tree and adapted it to support subset-superset queries and analyzed the performance for query response time. The contributions were that the proposed system is a simple and flexible indexing structure to represent signature files. It supports efficient handling of Object-Oriented inclusive queries on set-valued attributes. The limitation is that there is no way currently to access OIDs from the structure for handling complex queries. The future work includes implementing the system to run on real and large OODB and to tailor the structure to handle point and range queries in OODB.

REFERENCES:

1. Bertino E, Kim W, Indexing techniques for queries on nested objects, IEEE TKDE, Vol.1, No.2, 1989,196–214.
2. Christos Faloutsos, Access Methods for Text, ACM Computing surveys, Vol. 17, No. 1, Mar 1985, 49 – 74.
3. Dik Lun Lee, Young Man Kim, Gaurav Patel, Efficient signature file methods for text retrieval, IEEE TKDE , Jun 1995, Vol.7, No.3, 423-435.
4. Eleni Tousidou, Alex Nanopoulos, Yannis Manolopoulos, Improved Methods for Signature-Tree Construction, The Computer Journal, 2000, Vol. 43, No. 4, 301 – 314.
5. Eleni Tousidou, Panayiotis Bozanis, Yannis Manolopoulos, Signature-based structures for objects with set-valued attributes, Information Systems, Vol. 27, No. 2, 2002, 93 – 121.
6. I. Elizabeth Shanthi, Y. Izzaz, R. Nadarajan, On the SD-tree construction for optimal signature operations, Proc. of ACM COMPUTE , ACM Digital Library (Jan 2008).
7. Hwan-Seung Yong, Sukho Lee, Hyoung-Joo Kim, Applying Signatures for Forward Traversal Query Processing in Object-Oriented Databases, Proc. 10th Intl. conf. Data Engg, Feb 1994, 518 – 525.
8. Joseph M. Hellerstein, Avi Pfeffer, The RD-tree: an index structure for sets, Technical Report 1252, University of Wisconsin at Madison, 1994.
9. Kemper A, Moerkotte G ,Access support relations: an indexing method for object bases, Inform Sys Vol. 17, No.2, 1992, 117–146.
10. Kjetil Norvag 1999. Efficient Use of Signatures in Object-Oriented Database Systems, Proc.of Advances in Database and Information Systems ADBIS'99 (Sep, 1999).

11. Maier D, Stein J, Indexing in an object-oriented database, Proc. of the IEEE workshop on object-oriented DBMSs, Asilomar, CA, September 1986.
12. Manolis Terrovitis, Spyros Passas, Panos Vassiliadis, Timos Sellis, A Combination of Trietrees and Inverted files for the Indexing of Setvalued Attributes, Proc. of CIKM 2006, 728-737.
13. Nikos Mamoulis, Efiicient Processing of Joins on Set-valued Attributes Proc. of SIGMOD (Jun 2003) 157-168.
14. Shin, Hakgene, Chang, Jaewoo 1996. A new Signature Scheme for Query Processing in Object-Oriented Database, Proc. of COMPSAC '96 (Aug, 1996) 400-405.
15. Sven Helmer, Guido Moerkotte, Evaluation of main memory join algorithms for joins with subset join predicates, Proc. of VLDB 1997,386-395.
16. Sven Helmer, Guido Moerkotte, A performance study of four index structures for set-valued attributes of low-cardinality, VLDB journal 12, 2003, 244-261.
17. Wang-chien Lee, Dik L. Lee, Signature File Methods for Indexing Object-Oriented Database Systems, Proc. 2nd Intl. Comp. Sc. Conf, Dec 1992, 616 – 622.
18. Yangjun Chen, Yibin Chen, On the Signature Tree Construction and Analysis, IEEE TKDE , Sep 2006, Vol.18, No. 9, 1207 – 1224.
19. Yoshiharu Ishiwaka, Hiroyuki Kitagawa, Nobuo Ohbo, Evaluation of Signature Files as Set Access Facilities in OODBs, Proc. of ACM SIGMOD 1993, 247 – 256.
20. Xie Z, Han J, Join index hierarchies for supporting efficient navigation in object-oriented databases, VLDB 1994, 522–533.