(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) **World Intellectual Property Organization**
International Bureau

(43) **International Publication Date**
1 March 2007 (01.03.2007)

PCT

(10) **International Publication Number**
WO 2007/023136 A1

(51) **International Patent Classification:**
G06F 17/30 (2006.01)

(21) **International Application Number:** PCT/EP2006/065449

(22) **International Filing Date:** 18 August 2006 (18.08.2006)

(25) **Filing Language:** English

(26) **Publication Language:** English

(30) **Priority Data:**
11/209,997 22 August 2005 (22.08.2005) US

(71) **Applicant** *(for all designated States except US)*: **INTERNATIONAL BUSINESS MACHINES CORPORATION** [US/US]; New Orchard Road, Armonk, New York 10504 (US).

(71) **Applicant** *(for MG only)*: **IBM UNITED KINGDOM LIMITED** [GB/GB]; Po Box 41, Portsmouth Hampshire PO6 3AU (GB).

(71) **Applicant and**
(72) **Inventor: CHEN, Yao- Ching Stephen** [US/US]; 13056 Via Escuela Court, Saratoga, California 95070 (US).

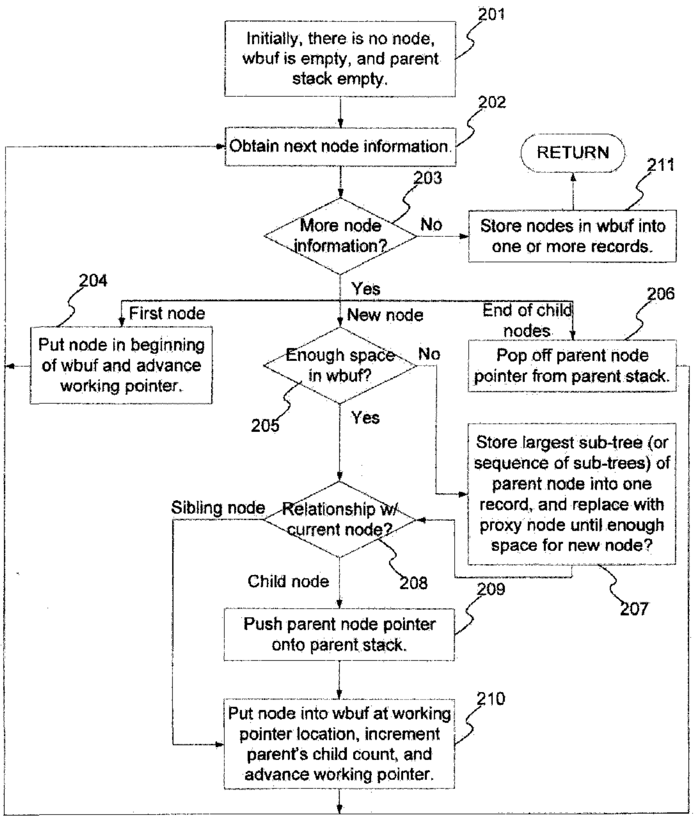
(72) **Inventors; and**
(75) **Inventors/Applicants** *(for US only)*: **HUANG, Yue** [CN/US]; 459 Casselino Drive, San Jose, California 95136 (US). **LIN, Fen-Ling** [US/US]; 1357 Mansion Court, San Jose, California 95120 (US). **TRAN, Brian Thinh-Vinh** [US/US]; 754 Carrywood Way, San Jose, California 95120 (US). **ZHANG, Guogen** [CN/US]; 6670 Tradition Court, San Jose, California 95120 (US).

(74) **Agent: ROBERTS, Scott**; IBM United Kingdom Limited, Intellectual Property Law, Hursley Park, Winchester, Hampshire SO21 2JN (GB).

(81) **Designated States** *(unless otherwise indicated, for every kind of national protection available)*: AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BW, BY, BZ, CA, CH, CN, CO, CR, CU, CZ, DE, DK, DM, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, HN, HR, HU, ID, IL, IN, IS, JP, KE, KG, KM, KN, KP, KR, KZ, LA, LC, LK, LR, LS, LT, LU, LV, LY, MA, MD, MG, MK, MN, MW, MX, MY, MZ, NA, NG, NI, NO, NZ, OM, PG, PH, PL, PT, RO, RS, RU, SC, SD, SE, SG, SK, SL, SM, SV, SY, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, ZA, ZM, ZW.

[Continued on next page]

(54) **Title:** PACKING NODES INTO RECORDS TO STORE XML XQUERY DATA MODEL AND OTHER HIERARCHICALLY STRUCTURED DATA



(57) **Abstract:** A storage of nodes of hierarchically structured data uses logical node identifiers to reference the nodes stored within and across record data structures. A node identifier index is used to map each logical node identifier to a record identifier for the record that contains the node. When a sub-tree is stored in a separate record, a proxy node is used to represent the sub-tree in the parent record. The mapping in the node identifier index reflects the storage of the sub-tree nodes in the separate record. Since the references between the records are through logical node identifiers, there is no limitation to the moving of records across pages, as long as the indices are updated or rebuilt to maintain synchronization with the resulting data pages. This approach is highly scalable and has a much smaller storage consumption than approaches that use explicit references between nodes.

WO 2007/023136 A1

(84) Designated States *(unless otherwise indicated, for every kind of regional protection available)*: ARIPO (BW, GH, GM, KE, LS, MW, MZ, NA, SD, SL, SZ, TZ, UG, ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European (AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HU, IE, IS, IT, LT, LU, LV, MC, NL, PL, PT, RO, SE, SI, SK, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

Published:

— *with international search report*

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

PACKING NODES INTO RECORDS TO STORE XML XQUERY DATA MODEL AND OTHER HIERARCHICALLY STRUCTURED DATA

FIELD OF THE INVENTION

The present invention relates to hierarchically structured data, and more particularly to the storage of hierarchically structured data in a database.

BACKGROUND OF THE INVENTION

As hierarchically structured data, such as eXtensible Mark-up Language (XML), become widely used as a data format, it also becomes a native data type for database systems. The storage of hierarchically structured data in relational databases, however, poses particular challenges.

One conventional approach is to store XML as text. This approach preserves the original documents and retrieves the entire document. However, it is inefficient in supporting queries and document updates, especially when the document is large.

Another conventional approach is to decompose and store the XML as tables in the relational database. This requires either a special relational schema for each XML schema or a generic relational representation for the XML data model. However, the result data is relatively large, and the queries are usually slow to execute.

Another conventional approach uses an object data model to store XML tree data, where many direct references or pointers are stored in the records for the parent-child relationships. However, this approach lacks scalability, has a larger data volume due to the references, and is less flexible in the re-organization of records.

Another conventional approach decomposes the XML data at a high level into relational data. However, this approach is inefficient in that it places lower levels and long text into a Character Large Object (CLOB), or it stores the original textual XML redundantly along with the object model.

Accordingly, there exists a need for an improved method and system for storing hierarchically structured data in record data structures. The improved method and system should combine the advantages of relational scalability and flexibility for the re-organization of records and the object efficiency for traversal and update.

SUMMARY OF THE INVENTION

An improved method and system for storing hierarchically structured data in record data structures uses logical node identifiers to reference the nodes of a hierarchically structured data stored within and across relational data structures, such as records or pages of records. A node identifier index is used to map each logical node identifier to a record identifier for the record that contains the node. When a sub-tree is stored in a separate record, a proxy node is used to represent the sub-tree in the parent record. The mapping in the node identifier index reflects the storage of the sub-tree nodes in the separate record. This storage scheme supports document order clustering and sub-document update with the record as the unit. Since the references between the records are through logical node identifiers, there is no limitation to the moving of records across pages, as long as the indices are updated or rebuilt to maintain synchronization with the resulting data pages. The method and system in accordance with the present invention thus is significantly more scalable than conventional approaches. It has a much smaller storage consumption than conventional object approaches that uses explicit references between nodes.

BRIEF DESCRIPTION OF THE FIGURES

Embodiments of the present invention shall now be described, by way of example, with reference to the following drawings:

Figure 1 illustrates an example hierarchically structured data tree containing a plurality of nodes;

Figure 2 is a flowchart illustrating an embodiment of a method for storing hierarchically structured data in a record data structure in accordance with a preferred embodiment of the present invention;

Figure 3 illustrates an example record storing a hierarchically structured data tree in accordance with a preferred embodiment of the present invention;

Figure 4 illustrates the local and absolute node identifiers for the example tree in Figure 1;

Figure 5 is a flowchart illustrating a search for a node of the hierarchically structured data in accordance with a preferred embodiment of the present invention;

Figure 6 illustrates example records for storing a tree across multiple records in accordance with a preferred embodiment of the present invention;

Figure 7 is a flowchart illustrating a method for generating the node identifier indexes for records with proxy nodes in accordance with a preferred embodiment of the present invention;

Figure 8 illustrates example entries of the node identifier index in accordance with a preferred embodiment of the present invention;

Figures 9A and 9B illustrate in more detail the tree traversal process used by the method in accordance with a preferred embodiment of the present invention; and

Figures 10A and 10B illustrate range proxy nodes in accordance with a preferred embodiment of the present invention.

DETAILED DESCRIPTION

Embodiments of the present invention provide improved methods and systems for storing hierarchically structured data in record data structures. Various modifications to the preferred embodiment will be readily apparent to those skilled in the art and the generic principles herein may be applied to other embodiments. Thus, the present invention is not intended to be limited to the embodiment shown but is to be accorded the widest scope consistent with the principles and features described herein.

The method and system in accordance with a preferred embodiment of the present invention uses logical node identifiers to reference the nodes of a hierarchically structured data stored within and across relational data structures, such as records or pages of records. A node identifier index is then used to map each logical node identifier to a record identifier for the record that contains the node. When a sub-tree is stored in a separate record, a proxy node is used to represent the sub-tree in the parent record. The mapping in the node identifier index is then updated to reflect the storage of the sub-tree nodes in the separate record. In this manner, when re-organization of records are desired or needed, only the node identifier index needs to be updated. The logical node identifiers in the records need not be changed.

To more particularly describe the features of the preferred embodiment of the present invention, please refer to Figures 1 through 10B in conjunction with the discussion below.

Figure 1 illustrates an example hierarchically structured data tree containing a plurality of nodes. The tree 101 can represent any type of hierarchically structured data, such as XML. Although the preferred embodiment of the present invention may be described below in the context of XML, one of ordinary skill in the art will understand that the method and system can be applied to other types of hierarchically structure data. The tree 101 has a root node (Node 0) with one child node (Node 1). Node 1 has three child nodes (Nodes 2, 6, and 7). Nodes 2, 6, and 7 are thus sibling nodes. Node 6 is a leaf node (it has no child nodes). Node 2 has three child nodes (Nodes 3, 4, and 5). Node 7 has one child node (Node 8).

Figure 2 is a flowchart illustrating an embodiment of a method for storing hierarchically structured data in a record data structure in accordance with the preferred embodiment of the present invention. Assume that the hierarchically structure data comprises a plurality of nodes. Initially, there is no node, a working buffer (*wbuf*) is empty, and a parent stack is also empty, via step 201. The next node information is then obtained, via step 202. It is then determined if there is more node information, via step 203, i.e., if the traversal of the hierarchically structured data or its equivalent token information has ended. If not, and if the node is the first node, then the node is put in the beginning of the *wbuf*, via step 204, and the working pointer is advanced.

If the node is a new node, then it is determined if there is enough space in the *wbuf* for the new node, via step 205. If not, then the largest sub-tree (or a sequence of sub-trees) of the parent node is stored into one record, via step 207. The taken-out sub-tree (or a sequence of sub-trees) is replaced with a proxy node until there is enough space for the new node. If there is enough space, then the relationship between the node and a current node is determined, via step 208. If the node is the child node of the current node, then the parent node pointer is pushed onto the parent stack, via step 209. If the node is a sibling node of the current node, then step 209 is skipped. Next, the node is put into the *wbuf* at the location pointed to by the working pointer, the parent's child count is incremented, and the working pointer is advanced, via step 210. If the node is an end of a set of child nodes, then the parent node pointer is popped from the parent stack, via step 206.

Eventually, there is no more node information, via step 203. At that time, the nodes stored in the *wbuf* is stored into one or more records, via step 211.

Figure 3 illustrates an example record storing a hierarchically structured data tree in accordance with the preferred embodiment of the present invention. In this embodiment, nodes are stored within records. A plurality of records is stored within a page. A plurality of pages is stored for a document. This type of record data structure is known in the art and will not be described in detail here. Each document is assigned a document identifier (DocID). Assume that all nodes of the tree 101 is part of the same document and can be stored within one record. The record contains a record header 301 and nodes 302. The record is assigned a record identifier (RID), which references a physical address of the record. And each node is assigned a logical node identifier (node ID). A logical node ID identifies a node based upon its relationship with the other nodes in the tree. It does not identify the physical location where the node is stored. There are two types of logical node ID's, an absolute node ID and a local or relative node ID. The local node ID of a node is assigned to the node according to its sequence under that particular parent node. Child nodes of different parent nodes are assigned local node ID's independently at each level in the tree. The absolute node ID is a concatenation of the local node ID's from the root node to the node. For example, the local node ID for Node 5 is '06' to indicate that it is the third sibling node at its level, while its absolute node ID is '020206'. The absolute node ID indicates that Node 5 is the third child node of its parent node (Node 2), where its parent node is a first child node of its grandparent node (Node 1), where its grandparent is a first child node of the root node (Node 0). The root node is assigned a local node ID of '00' and is ignored.

Returning to Figure 3, the record header 301 contains an absolute node ID of the rooted node. Each node 302 within the record contains a node kind, node length, number of children, and the nodes for the children. It also stores its local node ID. Figure 4 illustrates the local and absolute node IDs for the example tree 101 in Figure 1. Logical node ID's are further described in co-pending U.S. patent application serial no. 10/709,415 published with publication number 20060004858 titled "Self-Adaptive Prefix Encoding for Stable Node identifiers", filed on May 4, 2004, and assigned to the assignee of the present application. Applicant hereby incorporates this patent application by reference. The logical node ID provides stable node encodings that allow for arbitrary insertion, deletion or replacement of nodes. Existing node ID's need not be modified when a node is inserted, deleted, or replaced to keep node ID's in document order. This holds true because a logical node ID is not modeled as a fixed string of decimal numbers, but rather as a variable-length binary string.

In this embodiment, the storage of the tree 101 into records is based on a preorder traversal process, known in the art. However, other types of traversal processes can be used. With the preorder traversal processing, as the nodes are constructed, a grouping logic keeps track of the sub-tree being constructed for the length of the sub-tree rooted at the current node. For example, assume that the maximum record size, R, is known. A working buffer of 2xR or more in size is used in the construction. If the entire tree is smaller than R, then the entire tree is stored into one record. Otherwise, the tree is split into multiple records. The storage of a tree in multiple records is described further below.

For example, referring to both Figures 1 and 3, the root node (Node 0) is first stored with an indication that it has one child node. Its child node (Node 1) is then stored with an indication that it has three child nodes. Next, the first child node (Node 2) is stored with an indication that it has three child nodes. These child nodes (Nodes 3, 4, and 5) are then stored. The traversal process returns to Node 2 and continues with the next sibling node (Node 6). Nodes 6 and 7 are then stored, with an indication that Node 7 has one child. Node 8 is then stored after Node 7. Thus, with the preferred embodiment of the present invention, the relationships among the nodes of the tree 101 are captured by the nesting structure. No explicit links are used.

Referring now to Figure 5, to obtain a node with a given logical node ID, such as in response to a query, the node identifier index is searched, via step 501, to obtain the RID corresponding to a logical node ID. The record corresponding to the RID is then traversed to obtain the

node corresponding to the logical node ID, via step 502. To locate the node inside the record, the same traversal process used when storing the tree is used to locate the node with the local node ID at each level.
[032] A hierarchically structured data tree is stored within a single record whenever possible. Occasionally, multiple records are required to store the hierarchically structured data tree. When more than one record is required, the method in accordance with the preferred embodiment of the present invention stores sub-trees in a separate record, and represents this sub-tree in the parent record with a "proxy node", which itself does not contain a logical node ID. Assume for example, that the tree 101 in Figure 1 cannot be stored within one record. The sub-tree of Node 2, containing Nodes 2, 3, 4, and 5, is then stored in a separate record. Here, each record stores one sub-tree.

Assuming again that the maximum record size, R, is known, as the nodes are constructed node by node in the preorder traversal process, if the entire tree is larger than R, then the tree is split into multiple records. The largest sub-tree is searched and copied into a separate record. The copied sub-tree is replaced with a proxy node, and the length of the nodes in the separate record is excluded from the calculation of the sub-tree length. Only the length of the proxy node is included. All the length information is updated accordingly. Figure 6 illustrates example records for storing a tree across multiple records in accordance with the preferred embodiment of the present invention. Here, the parent record 601 contains a proxy node 603 that represents the sub-tree rooted in Node 2.

In order to find the sub-tree nodes represented by a proxy node, a node identifier (node ID) index is created. This index is to map a node ID to the RID of a record that contain the node with the given node ID. All the node IDs in document order can be viewed as points in a line. The records break this line into a plurality of intervals. The node ID index contains the upper end point of each interval. Figure 7 is a flowchart illustrating a method for generating the node identifier indexes for records with proxy nodes in accordance with the preferred embodiment of the present invention. First, the record is traversed to find the proxy node, via step 701. An entry is then created for the largest logical node ID before the proxy node, via step 702, with a mapping to the record's RID. Another entry is created for the largest node ID in the record, via step 703, with a mapping to the record's RID. These entries represent the range of logical node ID's that encompass the tree. For a logical node ID that falls within any two of entries, the greater RID is used to locate the node.

For example, referring to Figure 8, assume that node identifier index entries are being created for the records 601 and 602 (Figure 6). First, the record 601 is traversed to find the proxy node 603, via step 701. Node 1 has the largest logical node ID before the proxy node 603, so an entry 801 is created for Node 1 with a mapping to the RID for the record 601 (rid2), via step 702. Node 8 has the maximum logical node ID in the record 601, so an entry 803 is also created for Node 8 with a mapping to the RID (rid2) of the record, 601, via step 703. For record 602, there are no proxy nodes, so steps 701 and 702 are skipped. Node 5 has the largest logical node ID for the record 602, so entry 802 is created for Node 5 with a mapping to the RID (rid1) for the record 602, via step 703.

Thus, to locate Node 4 with logical node ID '020204', for example, a search of the node identifier index finds the three entries 801-803. The identifier '020204' is greater than '02' of entry 801, but less than '020206' of entry 802. Node 4 is thus mapped to the RID (rid1) for the sub-tree record 602. If Node 8 with logical node ID '020602' is to be located, '020602' is greater than '020206' of entry 802 and equal to '020602' of entry 803. Node 8 is thus mapped to the RID (rid2) for the parent record 601.

By using proxy nodes to reference sub-tree nodes stored outside of a parent record, the storage of hierarchically structured data is significantly more scalable than conventional methods. This is especially true since the nodes of the tree are stored as a few records, and the nodes of sub-trees can be moved together more efficiently. When nodes are updated, the records may require reorganization once it is discovered that not all nodes of the tree can be stored in one record. Upon this discovery, a sub-tree that can be stored in a separate record is identified. The nodes of the sub-tree are then replaced with the proxy node. If the records are less clustered, reorganization can be performed to make records in document order again. Because references between records are accomplished through logical node ID's rather than explicit references, this reorganization is significantly more easily accomplished, allowing greater scalability.

Figures 9A and 9B illustrate in more detail the tree traversal process used by the method in accordance with the preferred embodiment of the present invention. Figure 9A illustrates the traversal process with one record, while Figure 9B illustrates the traversal process with two records. A stack is used to track each level of nodes. In Figures 9A and 9B, the node ID's 901-902 are absolute node ID's in a variable-length binary string (2-byte length, followed by the node ID encoding). The length of each local node ID is kept in a separate array. Both node ID and length of node IDs are used as a stack when the tree nodes are traversed. The level is used as the stack top pointer. This way, the (absolute) node ID can be maintained easily, and is always available as a variable-length binary string format.In-scope namespaces in the XQuery data model can be similarly maintained for each node.

Here, a sub-tree starts at a current node and ends at the current node start position plus the sub-tree length. A tree can be traversed using two primitives: getFirstChild and getNextSibling. The primitive 'getFirstChild' starts from the current node, and if the number of children is '0', then 'not found' is returned. Otherwise, the next node is the first child. The primitive 'getNextSibling' starts from the current node, and if it is the root node, then 'not found' is returned. Otherwise, the total sub-tree length rooted at the current node is added to the start position of the node to get the next node position. If it is beyond the sub-tree rooted at the parent node, then 'not found' is returned. Otherwise, that next node is the next sibling.

If a proxy node is encountered, the search key for the node ID index is set to '(DocID, node ID)'. The index will return the RID of the record that contains the node. This record is then fetched and the traversal continues. To find a node with a given node ID, a node with the local node ID at each level is found using the above two primitives.

To further improve efficiency, a proxy node, called a range proxy node, can represent a sequence of sub-trees contained in a record, and multiple proxy nodes next to each other within a record can be collapsed into a single "range proxy node". For example, as illustrated in Figure 10A, a range proxy node 1001 can represent two proxy nodes that are collapsed, each of which represents a sequence of sibling nodes (or sub-trees) 1002-1003 stored in a separate record. For another example, as illustrated in Figure 10B, a range proxy node 1004 can represent multiple proxy nodes 1005-1007, each corresponding to a record that may contain a sub-tree or multiple sub-trees.

An improved method and system for storing hierarchically structured data in relational data structures are disclosed. The method and system uses logical node identifiers to reference the nodes of a hierarchically structured data stored within and across relational data structures, such as records or pages of records. A node identifier index is used to map each logical node ID to a RID for the record that contains the node. When a sub-tree is stored in a separate record, a proxy node is used to represent the sub-tree in the parent record. The mapping in the node identifier index is then updated to reflect the storage of the sub-tree nodes in the separate record. This storage scheme supports the following:

Clustering. To support document order clustering, the DocID and node ID for the sub-tree root are used. To improve the efficiency of the clustering, the DocID and the minimum node ID of the nodes, which is also the absolute node ID of the sub-tree root, can be put into separate fields within the record of nodes.

Update. A sub-document update can be performed with the record as the unit. Insert, delete, or replace of a sub-tree can be performed easily.

Re-organization of records. Since the references between the records are through logical node ID's, then there is no limitation to the moving of records across pages, as long as the indices are updated or rebuilt to maintain synchronization with the resulting data pages.

Partitioning. Even a document can be partitioned based on node ID ranges.

The method and system in accordance with the preferred embodiment of the present invention thus is significantly more scalable than conventional approaches. It has a much smaller storage consumption than conventional object approaches that uses explicit references between nodes. They can also leverage existing indexing approaches and reuse some of its utilities.

Although the present invention has been described in accordance with the embodiments shown, one of ordinary skill in the art will readily recognize that there could be variations to the embodiments. Accordingly, many modifications may be made by one of ordinary skill in the art without departing from the spirit and scope of the appended claims.

CLAIMS

1. A method for storing hierarchically structured data in record data structures, the hierarchically structured data containing a plurality of nodes, comprising:

storing each node of the hierarchically structured data in at least one record of a relational data structure; and

referencing each node of the hierarchically structured data using a logical node identifier.

2. The method of claim 1, wherein the logical node identifier identifies its corresponding node based upon the corresponding node's relationship with others of the plurality of nodes.

3. The method of claim 1, wherein the storing comprises:

storing with each node of the hierarchically structured data a corresponding local node identifier, wherein the local node identifier is assigned according to its sequence under its parent node.

4. The method of claim 3, wherein the hierarchically structured data is traversed using getFirstChild and getNextSibling primitives.

5. The method of claim 1, wherein the storing comprises:

storing an absolute node identifier of a context node in a header of the record, wherein the absolute node identifier comprises a concatenation of local node identifiers from a root node to a current node, wherein the context node comprises a parent node of a root node of a sub-tree or root nodes of sub-trees that share a same parent.

6. The method of claim 1, wherein a construction for storing nodes is performed based on a preorder traversal sequence.

7. The method of claim 1, further comprising:

constructing a node identifier index ;

searching the node identifier index for a record identifier corresponding to a given logical node identifier; and

traversing a record corresponding to the record identifier to obtain a node corresponding to the given logical node identifier.

8. The method of claim 1, wherein the storing comprises:

determining that not all of the plurality of nodes can be stored within one record;

finding a sub-tree of the plurality of nodes that can be stored within a separate record;

storing nodes of the sub-tree in the separate record; and

replacing the nodes of the sub-tree in a parent record with a proxy node.

9. The method of claim 8, further comprising:

changing a node identifier index to update a record identifier that corresponds to the nodes of the sub-tree, wherein the logical node identifiers for the nodes of the sub-tree need not be changed.

10. The method of claim 8, further comprising:

traversing the record and finding the proxy node;

creating an entry in a node identifier index for a largest logical node identifier before the proxy node with a mapping to a record identifier for the record; and

creating another entry in the node identifier index for a maximum logical node identifier for the plurality of nodes with a mapping to the record identifier for the record.

11. The method of claim 8, further comprising:

collapsing a plurality of proxy nodes in the parent record into a single range proxy node.

12. The method of claim 11, wherein the range proxy node represents a sequence of sibling proxy nodes stored in the separate record.

13. The method of claim 11, wherein the range proxy node represents the plurality of proxy nodes, wherein each proxy node corresponds to a record that contains a sub-tree or multiple sub-trees.

14. A computer readable medium with program instructions for storing hierarchically structured data in record data structures, the hierarchically structured data containing a plurality of nodes, comprising instructions for:

storing each node of the hierarchically structured data in at least one record of a relational data structure; and

referencing each node of the hierarchically structured data using a logical node identifier.

15. The medium of claim 14, wherein the logical node identifier identifies its corresponding node based upon the corresponding node's relationship with others of the plurality of nodes.

16. The medium of claim 14, wherein the storing comprises:
storing with each node of the hierarchically structured data a corresponding local node identifier, wherein the local node identifier is assigned according to its sequence under its parent node.

17. The medium of claim 16, wherein the hierarchically structured data is traversed using getFirstChild and getNextSibling primitives.

18. The medium of claim 14, wherein the storing comprises:
storing an absolute node identifier of a context node in a header of the record, wherein the absolute node identifier comprises a concatenation of local node identifiers from a root node to a current node, wherein the context node comprises a parent node of a root node of a sub-tree or root nodes of sub-trees that share a same parent.

19. The medium of claim 14, wherein a construction for the storing nodes is performed based on a preorder traversal process.

20. The medium of claim 14, further comprising:
constructing a node identifier index;
searching the node identifier index for a record identifier corresponding to a given logical node identifier; and
traversing a record corresponding to the record identifier to obtain a node corresponding to the given logical node identifier.

21. The medium of claim 14, wherein the storing comprises:
determining that not all of the plurality of nodes can be stored within one record;
finding a sub-tree of the plurality of nodes that can be stored within a separate record;
storing nodes of the sub-tree in the separate record; and
replacing the nodes of the sub-tree in a parent node with a proxy node.

22. The medium of claim 21, further comprising:
changing a node identifier index to update a record identifier that corresponds to the nodes of the sub-tree, wherein the logical node identifiers for the nodes of the sub-tree need not be changed.

23. The medium of claim 21, further comprising:
traversing the record and finding the proxy node;
creating an entry in a node identifier index for a largest logical node identifier before the proxy node with a mapping to a record identifier for the record; and
creating another entry in the node identifier index for a maximum logical node identifier for the plurality of nodes with a mapping to the record identifier for the record.

24. The medium of claim 21, further comprising:
collapsing a plurality of proxy nodes in the parent node into a single range proxy node.

25. The medium of claim 24, wherein the range proxy node represents a sequence of sibling proxy nodes stored in the separate record.

26. The medium of claim 24, wherein the range proxy node represents the plurality of proxy nodes, wherein each proxy node corresponds to a record that contains a sub-tree or multiple sub-trees.

27. A system, comprising:
a relational data structure comprising at least one record, wherein a plurality of nodes of a hierarchically structured data is stored within the at least one record, wherein each node of the hierarchically structured data is referenced using a logical node identifier; and
a node identifier index for mapping each logical node identifier for each node to a record identifier corresponding to a record containing the node.

28. The system of claim 27, wherein the at least one record comprises:
the plurality of nodes, where each node comprises its corresponding local node identifier, wherein the local node identifier is assigned according to its sequence under its parent node; and
a header comprising an absolute node identifier comprising a concatenation of the local node identifiers from a root node to a current node.

29. The system of claim 27, wherein the at least one record comprises a proxy node, wherein the proxy node represents node of a sub-tree of the plurality of nodes that cannot be stored within one record, wherein the nodes of the sub-tree are stored in a separate record.

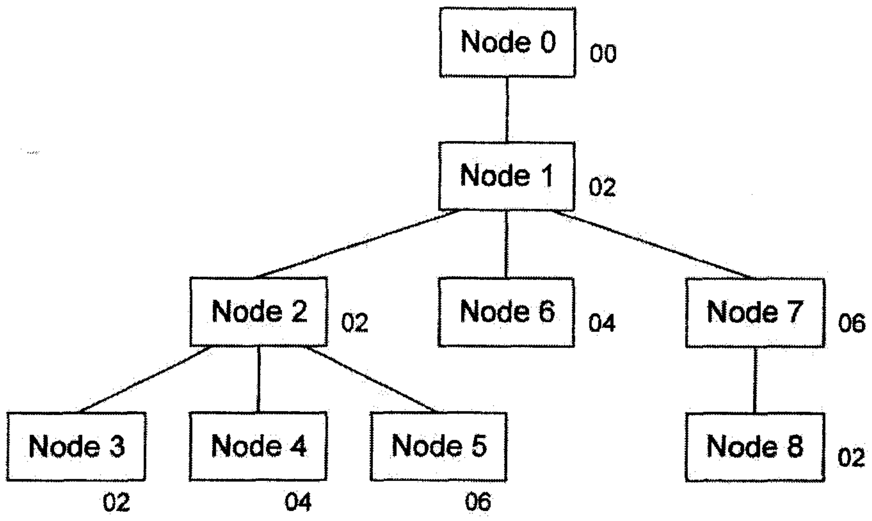
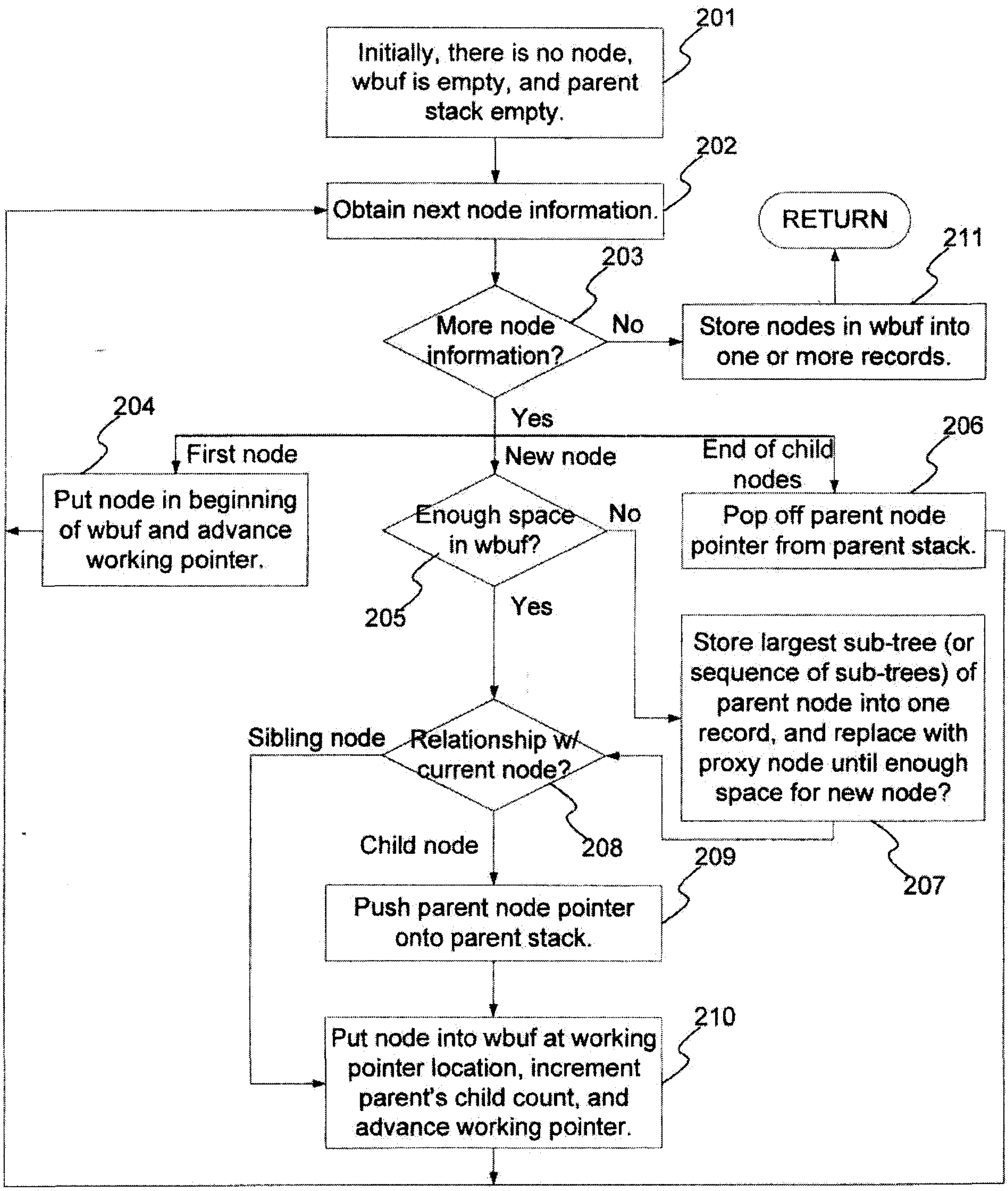

Node 0
00
Node 1
02
Node 2
02
Node 6
04
Node 7
06
Node 3
02
Node 4
04
Node 5
06
Node 8
02

FIG. 1



Initially, there is no node, wbuf is empty, and parent stack empty.
201
Obtain next node information.
202
More node information?
203
No
Store nodes in wbuf into one or more records.
211
RETURN
Yes
First node
New node
End of child nodes
Put node in beginning of wbuf and advance working pointer.
204
Enough space in wbuf?
205
No
Pop off parent node pointer from parent stack.
206
Yes
Store largest sub-tree (or sequence of sub-trees) of parent node into one record, and replace with proxy node until enough space for new node?
207
Sibling node
Relationship w/ current node?
208
Child node
Push parent node pointer onto parent stack.
209
Put node into wbuf at working pointer location, increment parent's child count, and advance working pointer.
210

FIG. 2

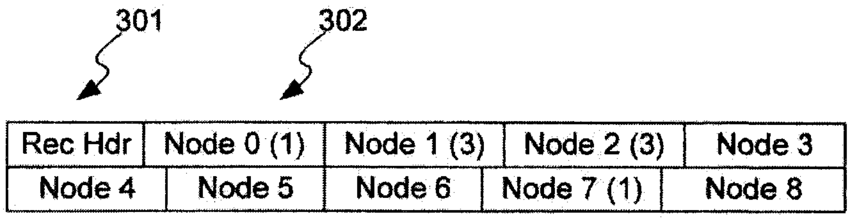


FIG. 3

Node No.	Local Node ID	Absolute Node ID
0	00	00
1	02	02
2	02	0202
3	02	020202
4	04	020204
5	06	020206
6	04	0204
7	06	0206
8	02	020602

FIG. 4

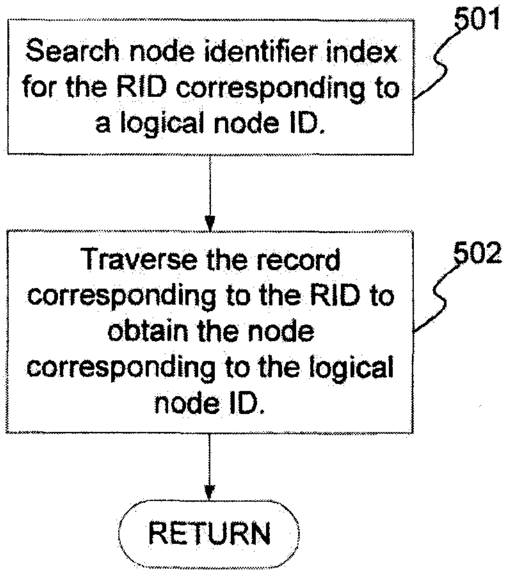

Search node identifier index for the RID corresponding to a logical node ID.
501
Traverse the record corresponding to the RID to obtain the node corresponding to the logical node ID.
502
RETURN

FIG. 5

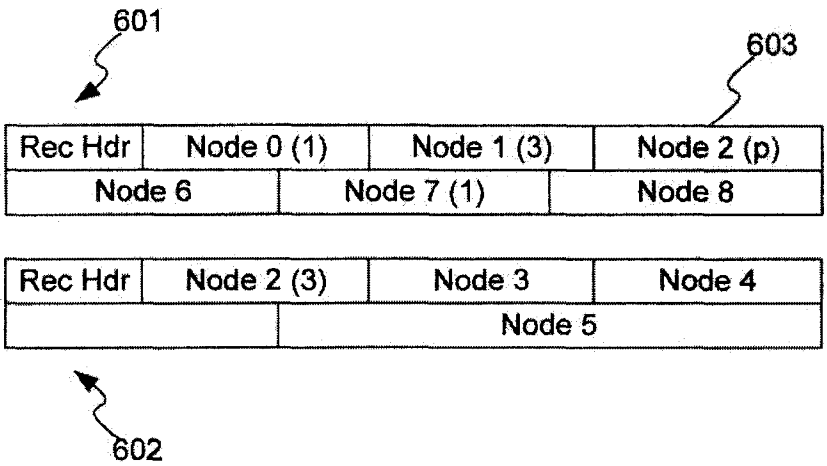

601
603
Rec Hdr
Node 0 (1)
Node 1 (3)
Node 2 (p)
Node 6
Node 7 (1)
Node 8
Rec Hdr
Node 2 (3)
Node 3
Node 4
Node 5
602

FIG. 6

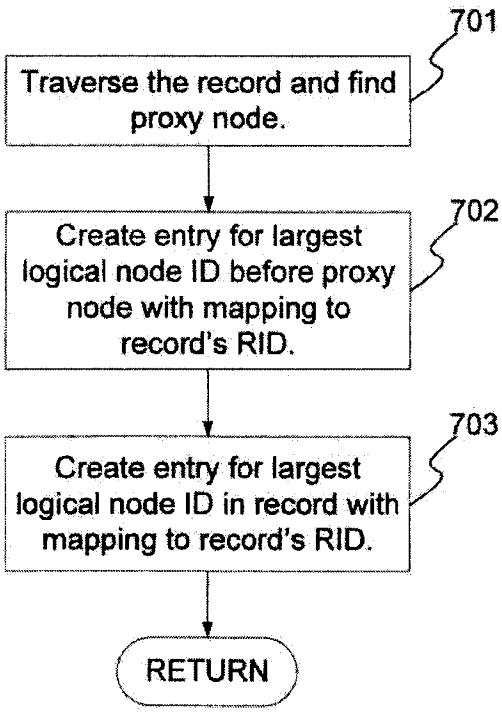
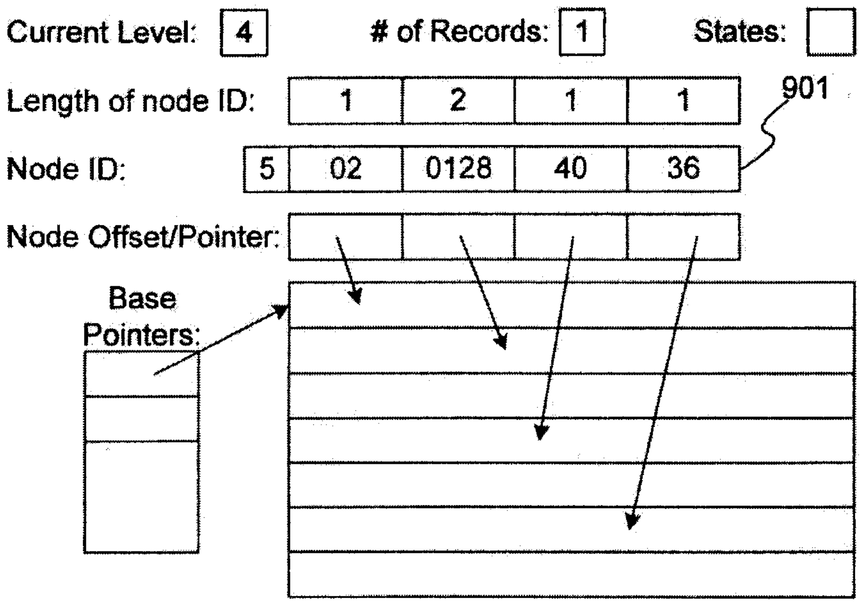


FIG. 7

Doc ID	Node ID	RID
docid	02	rid2
docid	020206	rid1
docid	020602	rid2

801 802 803

FIG. 8



Current Level:
4
# of Records:
1
States:
Length of node ID:
1
2
1
1
901
Node ID:
5
02
0128
40
36
Node Offset/Pointer:
Base
Pointers:

FIG. 9A

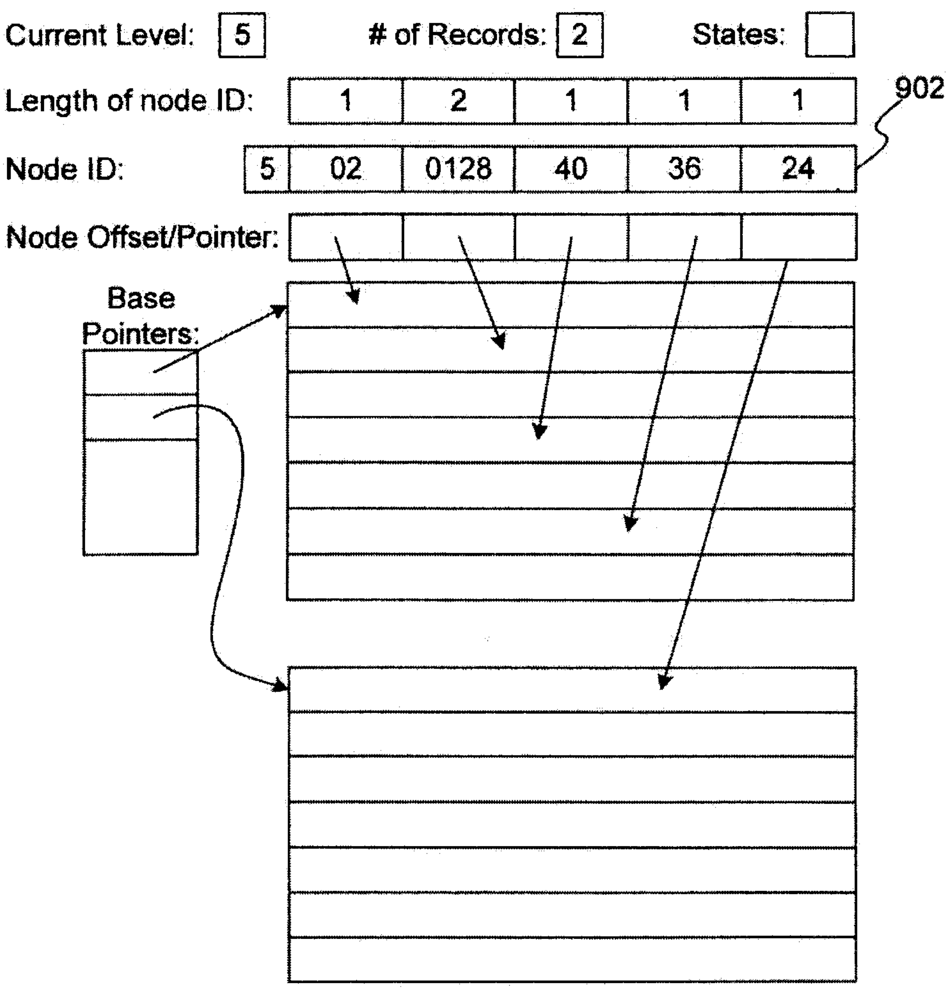

Current Level: 5
# of Records: 2
States:
Length of node ID: 1 2 1 1 1
902
Node ID: 5 02 0128 40 36 24
Node Offset/Pointer:
Base Pointers:

FIG. 9B

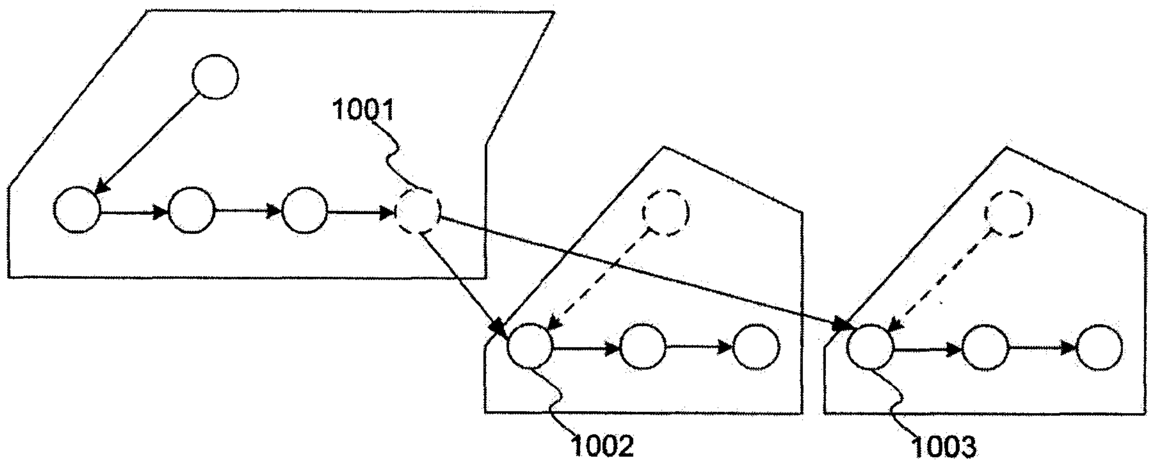


FIG. 10A

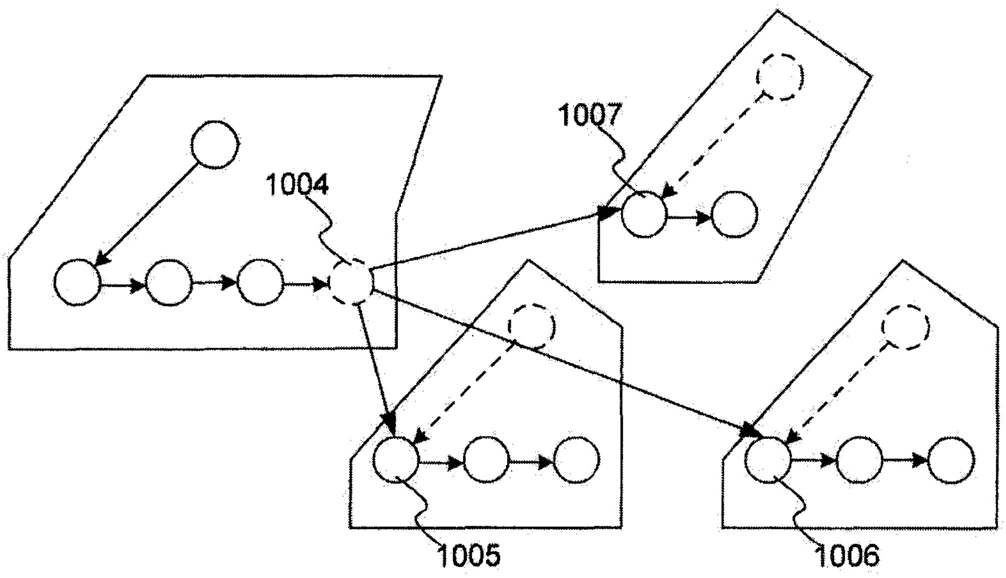


FIG. 10B

INTERNATIONAL SEARCH REPORT

International application No
PCT/EP2006/065449

A. CLASSIFICATION OF SUBJECT MATTER
INV. G06F17/30

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
G06F

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practical, search terms used)

EPO-Internal, INSPEC

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	THORSTEN FIEBIG, SVEN HELMER, CARL-CHRISTIAN KANNE, JULIA MILDENBERGER, GUIDO MOERKOTTE, ROBERT SCHIELE, TILL WESTMANN: "Anatomy of a Native XML Base Management System" TECHNICAL REPORT 01, UNIVERSITY OF MANNHEIM, [Online] 2002, pages 1-52, XP002404139 Retrieved from the Internet: URL:http://citeseer.ist.psu.edu/fiebig02anatomy.html> [retrieved on 2006-11-23]	1-4, 6-17, 19-27,29
Y	page 10, line 6 - page 12, last line figures 9,11 page 20, line 14 - page 21, line 15	5,18,28

	-/--	

[X] Further documents are listed in the continuation of Box C.

[X] See patent family annex.

* Special categories of cited documents :

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier document but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.

"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
23 October 2006	09/11/2006

Name and mailing address of the ISA/	Authorized officer
European Patent Office, P.B. 5818 Patentlaan 2 NL – 2280 HV Rijswijk Tel. (+31-70) 340-2040, Tx. 31 651 epo nl, Fax: (+31-70) 340-3016	Bykowski, Artur

INTERNATIONAL SEARCH REPORT

International application No
PCT/EP2006/065449

C(Continuation). DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	KANNE C-C ET AL: "Efficient Storage of XML Data" TECHNICAL REPORT UNIVERSITY OF MANNHEIM, 1999, pages 1-20, XP010378712 page 2, line 1 - page 6, line 10 page 8, lines 27,28 figures 8,15	1-4,6-9, 11-17, 19-22, 24-27,29
X	FIEBIG T ET AL: "Anatomy of a native XML base management system" VLDB JOURNAL, SPRINGER VERLAG, BERLIN, DE, vol. 11, 2002, pages 292-314, XP002325045 ISSN: 1066-8888 page 295, right-hand column, lines 5-7 page 296, right-hand column, lines 23,24 page 301, left-hand column, lines 40-47	1-4,6-9, 11-17, 19-22, 24-27,29
X	AMER-YAHIA S ET AL: "Logical and physical support for heterogeneous data" PROCEEDINGS OF THE ELEVENTH INTERNATIONAL CONFERENCE ON INFORMATION AND KNOWLEDGE MANAGEMENT. CIKM 2002 ACM NEW YORK, NY, USA, 4 November 2002 (2002-11-04), - 9 November 2002 (2002-11-09) pages 270-281, XP002404140 ISBN: 1-58113-492-4 page 277, left-hand column, line 22 - last line figure 8	1-6, 14-19, 27,28
Y	WO 01/42881 A2 (B BOP ASSOCIATES INC [US]) 14 June 2001 (2001-06-14) page 10, line 3 - page 14, line 6	5,18,28

INTERNATIONAL SEARCH REPORT

Information on patent family members

International application No
PCT/EP2006/065449

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
WO 0142881 A2	14-06-2001	AU 4521101 A	18-06-2001