A Survey of Indexing Techniques in Natives XML Databases

Imen Zemmar
Abdallah Benouareth
Labiba Souici-Meslati
A SURVEY OF INDEXING TECHNIQUES IN NATIVES XML DATABASES


*LABGED Laboratory, Badji Mokhtar University, BP 12, 23000, Annaba, Algeria
imen_zemmar@hotmail.com, benouareth@yahoo.com

** LRI Laboratory, Badji Mokhtar University, BP 12, 23000, Annaba, Algeria
souici_labiba@yahoo.fr

Abstract:

With the huge increase of XML documents on the Web, indexing, storing and retrieving these documents is of a great concern. Indexing and retrieving XML documents has recently become an active research area because they allow a convenient access to XML document parts. Several methods have been proposed for indexing XML documents: we can find two categories, those emanating from the database community and those arising from the information retrieval community. This article aims to present an overview of different indexing techniques in native XML databases, classifying them into categories according to their common features and comparing them to find which one is the most suitable for the new issue of semi-structured information retrieval.

Keywords: Semi-Structured Documents, Semi-Structure Information Retrieval, XML Indexing.

1. INTRODUCTION

With the intense growth and the diversity of information masses available on the web, it is more difficult for the user to reach information that meets his needs. XML is now presented as a new standard to better describe the information whose main purpose is to standardize the formatting data regardless of any proprietary format, whatever the data type. This language presents a revolution in how to handle such data. XML documents, besides the data itself, integrate meta-information and structural information. Thus, we talk of semi-structured documents.

In recent years, the functionalities of (object-)relational database management systems were extended by supporting XML data. Although XML is used mainly for data exchange, there is an emerging need for enabling persistent storage of XML data in databases. In contrast to the XML-enabled database systems, a new paradigm called native XML database management systems (XDBMS) was introduced to manage XML data exclusively. Therefore, a native XML database management system stores XML data persistently in their native form, avoiding costly transformation into relations and vice versa. Semi-structured databases, unlike traditional databases, do not have a fixed schema known in advance and stored separately from the data. Roughly speaking, semi-structured data are self-describing and can model heterogeneity more naturally than either relational or object-oriented data [22]. So when the XML data are stored in a database management system, many questions arise: How to express queries and updates? How can XML data be indexed to speed up frequent queries? And last but not least: How can indexes, that are best for a given application, be determined automatically? Tools to facilitate customized access to relevant information and that are integrated within specific information retrieval systems must react accordingly. Therefore, the problem they would deal with is the information retrieval (i.e. index management). Indexing has always been the focus of the information retrieval community. Because, it's well-known that without a good indexing, no information retrieval system could efficiently meet the user's needs.

Conventional indexing techniques consider the document as a words sequence, the problem was therefore simple, and consists in finding a way to state that such term belongs to such document. But with the semi-structured documents, the need to taking into account the documents structure during the indexing phase has quickly shown that these conventional techniques is no longer usable, at least not in their original forms. New techniques have as well emerged, but even those are not yet mature enough to be used effectively in the public information retrieval systems [8]. Essentially XML indexing is different from relational database indexing because of the tree-like model of XML. This introduces new challenges to the indexing databases methods [13]. Several studies have focused on effective indexing techniques. The goal is to build indexes that reduce the search space and thus the database portion to be analyzed to speed up the query process without consuming too much system resources.

The remainder of this paper is organized as follows. Section 2 presents the various approaches proposed in the literature to address the issue of indexing XML documents, depending on whether these approaches are databases or information retrieval oriented. An overview on the classificaitons of XML indexing techniques, according to several criteria, is given in Section 3. Section 4 discusses the different techniques for indexing XML document for native XML storage by...
classifying them into several categories according to their common characteristics. These techniques are commonly presented in heterogeneous ways and there are very few works that consider them together within a homogeneous view. Section 5 presents a comparison of indexing approaches to show which allow solving, at best, the new issues of semi-structured information retrieval. Section 6 draws a conclusion relating to the conducted study.

2. INDEXING TECHNIQUES FOR SEMI-STRUCTURED DOCUMENTS

The indexing process is the cornerstone of any information retrieval system. The semi-structured documents indexing seem more complex compared to the flat documents. Indeed, the flat document indexing only consists in extracting representative words of each document. The semi-structured document indexing, for its own purposes, is more complex process due to the coexistence of textual and structural information, so a semi-structured document indexing consists in finding a way representing these two information kinds. One of the main challenges in semi-structured information retrieval is to find a way to represent structural information in an XML document in order to exploit this index in the relevance calculation between a document and an XML query. Therefore, textual information should be shown as a function of structural information [1]. The vision of indexing process is then changed since it will be guided by the document structure, which generates new questions: What should we index the document structure? How connect this structure to the document content? How to weight the index terms? The objective of the indexing process in this case, is then to reconcile content and structure to be able to carry out searches of content and/or structure on XML documents. Therefore, most indexing schemes proposed in literature are redefining the storage granularity and use the XML documents structure.

According to Sauvagnat [24], taking into account the documents structure depends on how they are stored. The various approaches used for indexing semi-structured documents can be characterized along two dimensions [12]: the documents Storage Scheme, and possible Type’s Transformations between XML documents and storage structures. In our study, we were particularly interested in the storage scheme. How to store documents itself affects the indexing techniques that may be database or information retrieval oriented:

- **Database Management Systems (DBMS) Oriented-Approaches (or Mapping Middleware):** It considers a semi-structured document as table set managed by a DBMS and thus relies on indexing techniques for relational DBMS.
- **Information Retrieval Oriented-Approaches (or Native XML Storage Models):** It process the files as they occur without any changes. Native XML DBMS are developed specifically for XML. Unlike relational databases, they store complete documents or parts of documents in files and do not carry out mappings in tables. A document being a tree, they are designed to managing trees efficiently.

We can infer that the major drawback of databases oriented-approaches is the loss of the original document, and it requires a considerable time to perform joins between multiple tables to answer a simple query. Without mentioning the need for a DBMS on which the information retrieval system will be based. Current studies focus on information retrieval oriented-techniques, which do not exclude in some cases the adaptation of indexing approaches from the DBMS in information retrieval.

3. CLASSIFICATION OF INDEXING TECHNIQUES FOR SEMI STRUCTURED DOCUMENTS

Indexing approaches are usually categorized with respect to how the authors gather them along several criteria that they consider discriminatory. One criterion is the data structure used by an index. The three basic data structures used by indexes are: tables, trees and graphs. Another criterion used in [26] is the structure represented by an index. For example, an inverted file index represents a table, but it may use a B+-tree to encode the data. A third criterion involves the input and output parameters that require the XML query process.

According to the second criterion (the index structure) Felix Weigel [26] classifies the indexing techniques in three classes: elementary indices, path look-up indices and navigational indices. Whereas in [14] and according to the third criterion (the input and output parameters), M. Janek identifies three main indexes classes: elementary indexes, navigational indexes and content indexes.

Furthermore Ahn Chulho et al. in [6] present a classification of various indexing techniques suggested in terms of index structures and query evaluation algorithms. They categorized them into four different types: sequence-based indexes, structural indexes, numbering-based indexes, and keyword-based indexes.

On the other hand, Q. Zou et al. [31] and S. Mohammad et al. [20] mention that there are three major approaches for indexing XML data structures that are: path indexes (or graph indexes), node indexes and sequence-based indexes. Beda C. Hammerschmidt [13] has classified indexing approaches for XML data into three classes: structural indexes (or pure path indexes), value indexes and hybrid indexes.

4. INDEXING TECHNIQUES FOR NATIVES XML DATABASES

We classify the different indexing techniques into three classes: elementary indexes, path look-up indexes and navigational indexes (see figure 1).
to rebuild it by the query engine at the evaluation time. We present in the following some path look-up indexes.

4.2. ANNOTATED PATH LOOK-UP INDEX

The annotated path look-up index gathers under this name two similar index structures: the Element Locator Scheme [23] and the CIS index [18]. These are simple index structures for searching keyword occurrences and their paths in a corpus. They both take a keyword in input; on the other side their outputs are different. Indeed, the Element Locator Scheme returns an absolute label path for each node containing the keyword. As for the CIS index, it returns a path where each node is identified by a label and a unique ID.

The Annotated path look-up indexes are characterized by:

- The both indexes consist of a single look-up table mapping each keyword occurrence to the simple path via which it can be reached. Paths based-indexes relate only to tree-shaped databases. They don’t apply to graphs because it is difficult to matching paths when a node can be reached by multitude paths.
- The Element Locator Scheme identifies nodes by their sibling numbers. The CIS index does require nodes to have unique ID, but with no preference for any specific ID encoding.
- The use of an identifying nodes method may preclude the use of incremental updates. The CIS index itself can be updated incrementally. If it is combined with such techniques, however, adding nodes to the database may involve a complete re-indexing under certain circumstances. Although the Element Locator Scheme doesn’t identify nodes in this way, it faces the same problem. If, for example, a new node is inserted, the number of siblings increases. It thus becomes necessary to update the nodes having the same parent as that which has just been added.

4.2. CONTEXT INDEX

The Context Index [17] is a path look-up index similar to the CIS index. It can be used to index text or label with integrated filter structure that helps to limit heuristically the retrieved keyword occurrences to those matching the structural part of the query. The index is used to search accessible occurrences through a specific labels path called a path signature that is given as input and a set of node is returned. The context index is characterized by:

- It has a simple table in which each keyword is linked to a pair of node IDs and a path signature. It is restricted to tree databases.
- An incremental update since it lacks any identifying nodes method.

4.2.3. BITCUBE

The BitCube is a three-dimensional bitmap index [30]. It adds a third dimension to the document/term matrix for the simple paths in every document. A BitCube for XML documents is defined by the quadruplet BitCube = (d, p, v, b) where d is a document, p a path, v path textual content and b is the bit value equal to 0 or 1. Being accessible by any combination of its three dimensions, the
BitCube supports six different look-up patterns. The first three can be viewed as extracting a single slice from the cube. In other words, each of these primitive operations corresponds to a different projection from a three-dimensional to a two-dimensional matrix. In addition to these primitive slice patterns, various complex operations are supported, such as searching all occurrences of a given keyword reachable by a given label path. The main features of the BitCube index are:

- As a path look-up index, the BitCube has a (three-dimensional) table as main data structure and it is best used with tree databases.
- The question of node identification does not arise.
- A single BitCube indexing the whole database can be updated incrementally. Adding a new label path, keyword, or document creates a new slice in the index, which is simply removed when the corresponding path, keyword or document disappears.

4.2.4. APEX

In general, the indexes store all the paths from the root element. The size of these structures depends on the nesting depth of XML data. This may degrade performance if the depth is relatively high. The problem of large structural summaries designed to support all queries motivates Adaptive Path Index “APEX”. APEX was introduced by Chung et al. [5] and has the following goals:

1. First frequently occurring queries should be accelerated more than general queries.
2. The path expressions starting with descendant axis (//) that lead to a full evaluation of most structural summaries should be supported effectively.

It provides a compromise between size and efficiency. Instead of indexing all the paths from the root, APEX indexes, in structure, only the paths frequently used, in another structure, it keeps the vertices of the source structure. This significantly reduces the size. APEX characterized by:

- APEX consists of two structures, a graph GAPEX with the structural summary storing references to elements, and a tree of hash tables HAPEX representing the incoming path to nodes of GAPEX. Each node of GAPEX corresponds to one extent of XML data nodes. HAPEX is a specific index optimized for frequent queries that must be selected before building the index. APEX is represented by a labeled graph. It may easily support descendant queries.
- Each node in the graph GAPEX has a unique ID.
- APEX can be updated incrementally according to a change of load queries [13].

4.2.5. VIST

The VIST (Virtual Suffix Tree) index structure is proposed by Wang et al. [25]. It has the following features:

- The data-tree and the query-tree are transformed into structure-encoded sequences. They are stored in virtual Tries constructed using B+-trees. XML data is represented by the preorder sequence of its tree structure produced by a depth-first traversal of the XML data. As a result, for deep and large XML documents, the size of the index becomes a problem as it does not scale well with an increase in data size because the top elements have to be included within the sequence of the newly inserted elements. As the paths in XML data get longer, the sequence length will increase and hence the size of the index will increase exponentially in the size of data.
- The VIST index represents the tree nodes with a pair (x, y) where x is a label node and y is his path in the tree.
- The index VIST supports dynamic data insertion, deletion, and update.

4.2.6. PRIX

VIST’s top-down transformation approach weakens the query processing because it results in large number of nodes (paths) being examined during subsequence matching for commonly occurring non-contiguous tag names. Motivated by this fact, Rao and Moon [21] propose another approach that implements bottom-up transformation instead. This approach is called PRIX (Prufer sequences for Indexing XML) is a major source of improvement over VIST schemes.

- PRIX is based on Prufer Sequences. These sequences are indexed by a Virtual Trie. The bottom-up transformation of XML data-trees in PRIX plays a crucial role in reducing the query processing time. Basically, the top-level elements of an XML tree are shared with lower-level elements by being their parent or ancestor nodes. To encode the tree with a Prufer sequence, we repeatedly delete the leaf node that has the smallest label and append the label of its parent to the sequence. PRIX is based on the B+-tree, and it is built in a way similar to VIST [21].

4.3. NAVIGATIONAL INDEXES

The indexes of this class use directed graphs as their main data structure thus representing the document paths in a compositional, i.e. stepwise, and compact way. As with path indexes, each path in a navigational index can be considered a pre-computed join of multiple look-ups in indexes from the first two classes, which is evaluated by simply following the path top-down. In the remainder of this section, we present in follows some navigational indexes.

4.3.1. BUS INDEX

The BUS (Bottom Up Scheme) index [9] is an approach that supports combined structure and content queries with result ranking. Among all nodes matching the structural part of the query, those where the query keyword occurs most often are ranked highest. A look-up in the BUS index takes a simple path and a keyword as input and returns a set of (docID, nodeID, weight) triples, where docID is a document, nodeID is a node and weight
is a ranking value. The characteristics of the BUS index are:

- The BUS index consists of three data structures namely: a schema tree which serves as structure index, a content index is presented as a table and a table of accumulators that keeps the occurrence frequency of each keyword in the nodes. The BUS index applies only to tree-shaped documents.
- As the BUS index is based on Virtual Nodes, it is not recommended for frequently changing databases. The progressive update are very limited, hence a global update is often required.

4.3.2. SIGNATURE FILE HIERARCHY

The Signature File Hierarchy [3], [4] combines signature files and Tries to index structured documents. It has a filter allowing to discard some nodes which do not contain the query keywords. This index takes as input a simple path and a keyword signature and returns a set of node identifiers. The Signature File Hierarchy is used not only as a structure index but also as a textual filter. As its name suggests, it relies on signatures which are used to represent individual keywords as well as the whole textual content of a node. Due to this signature, entire subtrees can be safely ignored during the search without any risk. The characteristics of the signature file hierarchy are:

- This approach consists of two specific index structures: the signature file hierarchy proper and signature file index as secondary index. All nodes in the document accessible via a given simple path are represented by a single node. When reaching such a node during structural query evaluation, we would like to know immediately the node that contains right keyword. The Signature File Hierarchy has been proposed for tree databases only.

- This index can be updated incrementally. When the document node content has been changed, its signature needs to be updated. This modification is propagated upwards to the database tree root.

4.3.3. DATAGUIDE

The DataGuide [10] is an index structure for both tree- and graph-shaped documents. The DataGuide takes a simple path as input and returns a set of nodes identifiers as output. It is characterized by:

- A DataGuide is a graph where every simple path is represented only once. However, when using them as structure index may cause simple paths to be associated with references which aren’t in their target set. The DataGuide may be created by depth-first traversal of graph XML data.

- The DataGuide is independent of -and therefore compatible with- any node identification scheme. Alternatively, the Virtual Nodes scheme may be used to identify parent/child pairs without a secondary index.

- The DataGuide supports progressive updates that can be costly because document nodes can be referenced in multiple points in the index structure.

4.3.4. STRONG DATAGUIDE

Goldman and Widom [10] presented one of the early structure summaries called a Strong DataGuide. In this scheme, the nodes in the source data are partitioned based on their root path, that is, the path from the root to the indexed node. The structure summary of an XML data-graph is a Strong DataGuide if it fulfills two conditions:

1. Every distinct root path in the source data appears only once in the graph index.
2. All the paths in the graph index have at least one matching root path in the original source data. In other words, there are no invalid paths in the graph index.

Updating Strong DataGuides could be as simple as inserting a new leaf into tree structured data, which requires only one target set to be recomputed and one new object to be added to the Strong DataGuide. In the worst case, a tree updating with a subgraph of structured data that has loops and sharing may incur recomputation to a large portion of the Strong DataGuide. Both types of updates, namely, edge and subgraph additions are supported by the Strong Data Guide scheme. An edge insertion update requires touching a number of nodes and edges.

4.3.5. APPROXIMATE DATAGUIDE

Experiments have shown, in general, that the Strong DataGuide size is much smaller than the original database. There are cases, however, where the size of the Strong DataGuide is unreasonably large (e.g., for cyclic data). To overcome this disadvantage, an Approximate DataGuide (ADG) is proposed by Goldman and Widom [11]. ADG ignores the second requirement of the Strong DataGuide, but maintains the first one. Therefore, it ensures that every distinct root path in the data source appears exactly once in the ADG, but it does not ensure that all ADG paths exist in the original data. Experiments demonstrate that there is a trade-off between the size of ADG and its accuracy. In general, Strong DataGuide features are applicable for ADG, except that the size of the ADG is often smaller [20].

4.3.6. CONTENT-AWARE DATAGUIDE

The problem of the expensive intersection operation is faced in the Content-Aware DataGuide (CADG) by Weigel et al. [27], [28] where a content/structure join is pre-computed and leads up to a 400 times faster execution compared to the conventional DataGuide. Weigel et al. introduce two approaches: The naive content-centric approach where a separate DataGuide is established for each value. For each element value of the sample data the content-centric CADG contains one conventional DataGuide that refers to all nodes that contain this value. This approach wastes a lot of space it is only suitable for single key queries. Path expressions with more than one value comparison or a range of values are supported less efficiently and require join-operations.

The authors that are aware of these severe disadvantages propose a second approach that is structure-centric. It takes a conventional DataGuide and enriched extents with content information. This information is
taken to prune irrelevant paths when processing a path expression.

A major issue of the CADG is its limited capability to deal with updates. When adding/deleting a node or when changing the value of a node the corresponding signatures/IDs must be identified and recalculated respectively deleted. In general, an update implies that all extents of the CADG must be touched.

4.3.7. CTree INDEX

The CTree [15], [31] is a two-level tree containing a level group that provides an overview of hierarchical structures (paths summaries) and contains edges from parent groups to their corresponding child groups, and a level element that provides, meanwhile, detailed relationships between elements. The CTree index is constructed in two stages: the first step is to build the tree summaries of paths. As the latter do not preserve the hierarchical relationships between individual nodes, a second step is necessary to order nodes. The CTree takes a simple path and an attribute as input and returns a nodes IDs list. It is characterized by:

- The CTree as presented in [15] has two index structures: a CTree tree as index structure and a value index which consists of five distinct structures, each one represents a table (invert, list, Numbers, DTime and ID). A CTree index is used for tree databases and probably acyclic directed graphs.
- The node identification is not required, but the use of secondary index as the value index requires an identification. Why, updating may not be incremental. It is progressive.

4.3.8. T-INDEX

The template index, known as T-index [19], was designed to provide an effective general index structure for semi structured data with competitive results. A concept of templates is used by this index to take advantage of indexing most frequent queries. With its utilization of regular expressions into input parameters, indexing almost any path relationship type within document is granted. Special cases of the T-index include the 1-index, for indexing all absolute paths, and the 2-index, which covers relative paths. The T-index is tailored to answer specific queries whereas the 1-index and the 2-index can be used to process any path expressions. These indexes are characterized by:

- The T-index is build to evaluate regular path expression queries with path templates. The T-index may be defined as a data structure designated to compute queries that match templates $t = \exists x_1 \exists x_2 \exists x_n$ where $\exists$ is a placeholder that will be instantiated by a regular path expression. This is the most general case of a path template. However, two specific cases of path templates exist.

First, the 1-index that computes queries for template $t_1 = x_1 \exists$. Such template represents absolute paths. Then the 2-index which computes queries for template $t_2 = \exists x_1 \exists x_2$. Such template selects both absolute and relative paths from a document.

- All three variations of the T-index are generally non-deterministic graphs with node sharing and cycles. Only for pure tree databases they become deterministic trees.
- Node identification schemes encoding hierarchical relationships between individual document nodes are useful with any T-index, including the 2-index, when the template's structure is too coarse to identify all desired nodes. However, Virtual Nodes as well as interval encoding are restricted to tree databases.
- There is no sophisticated algorithm for incremental update of the T-index. However, an incremental approach presented in [2] is applicable to T-index if the model does not contain placeholders, and only for an insertion or deletion, but not both.

4.3.9. INDEXFABRIC

The IndexFabric [7] indexes both paths and content of tree databases in a balanced hierarchy of PATRICIA Tries. Look-ups in the IndexFabric may follow the typical pattern for navigational indexes: a simple path is taken as input and a set of document node IDs is returned. The IndexFabric has been designed to index both the structure and the content of large databases. The main features of the IndexFabric are:

- In its simplest form, the IndexFabric is a Patricia Trie indexing label combination and strings. To this end, a symbol, called designator is assigned to each label's documents collection. The designators are stored in a designator dictionary. Before inserting a path in the IndexFabric all its labels are replaced by their respective designators. If the last path node has text content, the matching string is attached to the path labels coded by designators. The IndexFabric is used for tree databases and probably acyclic directed graphs.
- The node identification using numbering schemes such as interval encoding or virtual nodes may be valuable.
- This index may support incremental schemes such as as interval encoding or virtual nodes may be valuable.

5. COMPARISON OF INDEXING TECHNIQUES FOR SEMI STRUCTURED DOCUMENTS

After providing a description of the different indexing techniques of semi-structured documents, we present in this section, a comparison of these techniques to show how to solve, at best, the new issue of semi-structured information retrieval. Unfortunately, there are not enough studies of this type in the literature and those that were made, are not satisfactory because, on one hand, they don’t consider all approaches but represent only few of them, and on the other hand, the comparison is mainly based on theoretical criteria since there are no results for all approaches on the same document collection. Our main goal is not to demonstrate that one approach is better than another, but it is mainly to indicate to what degree an approach is more desirable than another and this according to several points of view [8].
So we will first present an overview of three major categories of indexing approaches. Then, we will present a comparison of indexing approaches of the last two categories namely the path look-up indexes and navigational indexes. The elementary indexes are excluded from this comparison since they all have the same structure, the same performance and represent the foundation of any indexing approach. In addition they are never used individually (see next section), so the introduction of these indexes in the comparison is not required and does not providing any additional information to our study. Finally, we will present the main criteria used to establish our comparison.

5.1. OVERVIEW OF THE INDEXING APPROACHES CATEGORIES

We presented in Section 4, the general principle of the three indexing techniques categories, we addresses here their strengths and weaknesses. We will notice, throughout this discussion, that is the disadvantages of one category that led to the following one. But this doesn’t mean that a class has replaced another. The three categories solve each in its own way, a part of the structured information retrieval problem.

The "elementary indexes" don’t hold much space, thus allowing the complete documents collection representation in main memory, which significantly reduces the research latency. These indexes, being the first to emerge, are directly deducted from the semi-structured documents hierarchical structure. They index different document parts independently and this is their main weakness. The "path look-up indexes" solve the problem of relationship between nodes and have a finer granularity. But even with this significant improvement over their predecessors, they remain unable to answer more complex structures queries such as trees. Where they can, there is a significant time loss in joins and processing queries are found. The final approach category overcomes this problem by storing the complete structural context in the index and with different trees variants more compact than the database. They remain quite complex to implement.

We cannot state which of these three categories is the most widely used in real information retrieval systems, but from the materials used for this study, we can confirm that the elementary indexes are widely used in commercial systems, and in most cases, as secondary indexes alongside to more complex structures. Navigational indexes are being improved and mach current researches.

5.2. COMPARISON CRITERIA

The comparison criteria are derived based mainly on the characteristics that we presented for describing various indexing techniques and the work of Laurent Yeh et al. [29] that presents some indexes evaluating criteria. These criteria are: the research granularity, the consulting number, the node identification and the updates. So, before comparing the different techniques presented in this paper (see sections 4), we give below a summary table of main features (Table 1).

<table>
<thead>
<tr>
<th>Indexing Technique</th>
<th>Research Granularity</th>
<th>Consulting Number</th>
<th>Node Identification</th>
<th>Updates</th>
</tr>
</thead>
<tbody>
<tr>
<td>Element Locator Scheme</td>
<td>Node</td>
<td>1</td>
<td>Sibling Number</td>
<td>Global</td>
</tr>
<tr>
<td>CIS index</td>
<td>Node</td>
<td>1</td>
<td>Virtual Nodes</td>
<td>Global</td>
</tr>
<tr>
<td>BitCube</td>
<td>Document</td>
<td>2</td>
<td>Any</td>
<td>Progressive</td>
</tr>
<tr>
<td>VIST</td>
<td>Node</td>
<td>1</td>
<td>Virtual Nodes</td>
<td>Incremental</td>
</tr>
<tr>
<td>PRIX</td>
<td>Node</td>
<td>1</td>
<td>Post-order</td>
<td>Progressive</td>
</tr>
<tr>
<td>BUS</td>
<td>Node</td>
<td>1</td>
<td>Virtual Nodes</td>
<td>Global</td>
</tr>
<tr>
<td>Signature file hierarchy</td>
<td>Node</td>
<td>1</td>
<td>None</td>
<td>Incremental</td>
</tr>
<tr>
<td>DataGuide</td>
<td>Node</td>
<td>1</td>
<td>Any</td>
<td>Progressive</td>
</tr>
<tr>
<td>Strong DataGuide</td>
<td>Node</td>
<td>1</td>
<td>Any</td>
<td>Progressive</td>
</tr>
<tr>
<td>Approximate DataGuide</td>
<td>Node</td>
<td>1</td>
<td>Any</td>
<td>Progressive</td>
</tr>
<tr>
<td>content-aware DataGuide</td>
<td>Node</td>
<td>1</td>
<td>Signatures/ID</td>
<td>Global</td>
</tr>
<tr>
<td>CTree</td>
<td>Node</td>
<td>1</td>
<td>None</td>
<td>Progressive</td>
</tr>
<tr>
<td>T-Index</td>
<td>Node</td>
<td>1</td>
<td>Interval Encoding/ Virtual Nodes</td>
<td>Incremental</td>
</tr>
<tr>
<td>IndexFabric</td>
<td>Node</td>
<td>1</td>
<td>Interval Encoding/ Virtual Nodes</td>
<td>Incremental</td>
</tr>
</tbody>
</table>

Table 1. Comparative Table of Indexing Techniques

5.2.1. RESEARCH GRANULARITY

The research granularity is the smallest structural unit returned by the information retrieval system. In the case of classical information retrieval, the granularity is still at the document level. But with the structured information retrieval, we may go down to the node. Finer is the search granularity, better is the method. We will be said that indexing approach has a granularity at element level if this approach will return a set of nodes IDs in output and if its index structure allows us to trace the reporting relationship which exists between nodes. According to Table 1, we note that almost all approaches have a fine granularity (Node Level), besides the BitCube and the context index. Both approaches don’t provide any way to expressing hierarchical link between nodes. Using signatures, the context index serves as more than structural filter for paths-based queries. It may thus match a label path without taking into account the nodes order but cannot return nodes and paths that leading to it, precisely because of this relationship lack between nodes. The BitCube expresses a ternary relationship between documents, nodes and terms and doesn’t provide any way to express the hierarchical relationship between nodes. Navigational approaches have all a fine granularity as they record the collection structural context. The path look-up approaches are little more difficult to meet this fine granularity, and, when possible, it be summarized to making a nodes set with paths leading to them and original structure rebuilding remains the query engine purview.

5.2.2. CONSULTING NUMBER

The consulting number allows the indexing method flexibility estimation. Higher is this number, more flexible is the index structure which gives more search
choices. This helps unloading the query engine and reducing the research choices. From the results expressed in Table 1, we see that the unique approach to allow broad flexibility is BitCube, his three-dimensional vision of the indexing problem can make six different searches on the index structure without joint efforts by the query engine. Approaches with a simple table as basic structural unit all have a single consultation type which is expressed by the failure to perform direct search with a simple path and a keyword. The context index is the unique approach in this group which presents an exception in having two consultations. The backup node IDs with a signature path instead of full path allow searching keywords and node labels simultaneously. Tree-based approaches provide only one consultation type and this because the tree representing the structural links between nodes is typically used to match a query path and not seeking for paths set that meet certain criteria.

5.2.3. NODE IDENTIFICATION

Indexing approaches use different methods for identifying nodes and documents. The approaches which do not use identification method gain in the index update time but lose enormously in the research speed since that the system should scan retrieved documents to identify nodes at the research time. Since we compare indexing approaches, those that don’t identify nodes as the BitCube, the Signature File Hierarchy and the C-Tree are better than others because, even if it penalizes the query engine, they accelerate updates.

5.2.4. INDEX UPDATES

The indexes update is more or less fast depending on the information amount to be inserted and on the processing to be performed on the existing index. The update speed depends on the adopted method for updating. The three used update methods for describing indexing techniques in this article can be ordered from slower to faster as follows: Global update → Progressive update → Incremental update.

Besides the BitCube, all paths look-up indexing approaches behave very bad in updating. The paths backup as strings, removes the index structure node concept. However, mostly, the updates only concern a specific nodes set in the collection. As the main and basic unit in such approach is a string, changing a single node involves the alteration of all strings where this node appears. The fact that the BitCube more interested in the relationship between nodes and the terms in a given document, rather than paths leading to these terms, has meant that this approach was an exception to the rule. Therefore the BitCube allows the incremental update because the insertion is done directly in the index structure without taking into account existing data and change only a few well-defined bits structure.

Navigational approaches allowing a better update over previous and this is due to using a more sophisticated structure that is the tree. The nodes are well represented with hierarchical relationships while maintaining a separate existence. Each node of a tree can be achieved and updated individually and the hierarchical relationships represented by edges in the tree may also be redirected without altering existing data. The updating of such approaches is generally incremental, except when using a secondary index as in DataGuide and C-Tree. The BUS index has a global update because of its two secondary indexes (the content index and table of accumulators).

6. CONCLUSION

As we state previously the indexing process is crucial in information retrieval. The accurate function of an information retrieval system depends on the indexing quality for a document, because a wrong indexed document may not be accessible to users. The main problem of indexing semi-structured documents is that, in addition to text content, it must index the structure and relate the two.

We have presented in previous sections, a set of indexing approaches for semi-structured documents. These approaches have been classified into three broad categories: elementary indexes, path look-up indexes and navigational indexes. These categories differ in the way that is used to index the documents structural information. While the first, which are derived directly from the elementary characteristics of semi-structured documents, index the parts of them individually, the second are based on paths to represent the structure. Navigational approaches are the most sophisticated by allowing a high fidelity representation of original structure.

By studying more closely the index results, we can draw two important conclusions: First, we note that the approaches that use the signatures principle for representing structural information as in the context index, or those who use the content, as in the signature file hierarchy, can gain considerable space and processing time because the bits manipulation is encoded in the processor and the binary strings comparison is usually a very fast operation. Second, the use of the Tries principle for representing the collection structure as in the signature file hierarchy, the IndexFabric, the DataGuide and the C-Tree can summarize the structural relationships of a large heterogeneous documents collection in a very small structure.

This article is the result of our preliminary comparative study that is an essential step in our work to the development and implementation of a new indexing XML data technique and the adaptation of the keywords search model by querying native XML databases.

REFERENCES:


321


