

OrientX: an Integrated, Schema-Based Native XML Database System

□ MENG Xiaofeng, WANG Xiaofeng, XIE Min, ZHANG Xin, ZHOU Junfeng

School of Information Renmin University of China, 100872, Beijing

Abstract: The increasing number of XML repositories has stimulated the design of systems that can store and query XML data efficiently. OrientX, a native XML database system, is designed to meet this requirement. Compared with other native XML databases, OrientX has two main features: First, XML schema is fully supported in OrientX, because schema has been proved to be of great importance to XML data storage, indexing, query optimization and access control; Second, OrientX is an integrated system that meets various requirements on XML data repository, which includes a native storage subsystem, several XML query evaluators, a composite index manager, a cost-based XQuery Optimizer, an Access Control module and an extension to XQuery1.0 for XML update. The main contributions of OrientX are: a) We have implemented an integrated native XML database system, which supports native storage of XML data, and based on it we can handle XPath & XQuery efficiently; b) In our OrientX system, Schema Information is fully explored to guide the storage, optimization and query processing, which we show can boost the system performance remarkably.

Keywords: XML; Database

CLC Number: TP 391

Received date: 2006-03-25

Foundation item: supported by the grants from the Natural Science Foundation of China (60573091, 60273018)

Biography: MENG Xiaofeng(1964-), male, Professor, research direction: web integration, XML Database and mobile data management. Email: xfmeng@ruc.edu.cn

the tree structure and the data type definitions of XML data, i.e. nodes in data tree are organized together based on its schema. We argue that making good use of schema information could improve the efficiency of storing and

0 Introduction

XML is a self-describing language, and has become the new de facto standard for data representation and exchange on Internet. The increasing number of XML repositories has stimulated the design of systems that can store and query XML data efficiently. Many systems have been designed to meet the goal. All of these systems can be divided into two categories: one falls into the Relational way, which utilizes the table-based storage model of existing DBMS, and needs to map XML data into two-dimension tables(e.g.^[1]); The other is a native way, which develops a tree-based storage strategy for XML data, and doesn't need an additional mapping.

The relational strategy introduces an additional transform between the logical XML data and its physical relational storage. As hierarchy of XML data is complex, and optional or repeatable sub-elements are allowed in it, the mapping from XML data to relational data often results in large amount of tables. Due to this storage strategy, the complexity of XQuery is beyond the capacity of SQL expressiveness. None of the existing traditional DBMS could be adequately customized to support XML, despite all claims of their vendors.

In a native XML database, XML data is stored directly, which retain XML data's natural tree structure (for short, we call it data tree, and call elements of XML document in it as nodes). The query processing engine can handle query languages such as XQuery & XPath directly on the tree structure. As it reserves the tree structure of XML data, native strategy can avoid the mapping operations. Some native XML databases have appeared, for example, Timber^[2], Natix^[3], tamino^[4] and so on. Most of these systems are schema-independent, that is to say, the schema is not a necessity of the system. But OrientX believes that schema plays a crucial role in XML data management and is indispensable. XML schema describes

retrieving XML data remarkably. XML schema is of great use in the following aspects of a native XML database:

- As related(neighbor) nodes in XML data tree are

likely to be queried at the same time, it is better to store them together, and XML schema provides the information of how nodes are related.

- Path index is very important to XML query evaluation, and schema makes good support to it.
- Schema can be used in validation of query and update processing.
- Query optimizer can collect statistical information by integrating the schema.
- Schema is the precondition to access control of XML data.

OrientX stores XML data using its own storage subsystem--OrientStore, in which XML data is stored in pages on disk, the granularity of storage can be changed according to schema. When stored data are loaded into memory, tree structure of the related nodes is built dynamically, therefore OrientStore offers a DOM-like navigation interface to the upper modules. Besides OrientStore, there is a holistic Index Structure called SUPEX, in which all path index and some value index are put together. XPath queries can be handled efficiently through fast navigation and join operation with the help of SUPEX. There are two query Evaluators for XQuery in OrientX. One is navigation-based, the other is algebra-based.

1 Motivation

The goal of OrientX is to build a native database system, which can manage XML repositories conveniently and efficiently.

Among all criterions of such a system, we first focus on the two main aspects: storage and query, for a well-designed storage and an efficient query evaluator form the basis of a good database system.

Two kinds of information need to be stored to preserve the structure of the document tree: node and edge. Node denotes element's data type and content. Edge denotes the direct relationship (Parent-Child in data tree) between nodes.

Another issue in XML data management is query, on which much work has been done too. A group of automata-based approaches are proposed for XPath and twig query processing, but they become incapable when facing complex XQuery statement. However, XQuery has become more and more popular because of its expressiveness and flexibility. There are two main

methods to process XQuery, one is navigating through the data tree, and the other is using an algebra.

Besides storage and query, many problems remain in XML data management, such as indexing, query optimization, updating and accessing control on data repository. As XML is more and more popular today, both the industry and research communities show close interest to these problems.

2 An overview of OrientX

The overall architecture of OrientX is shown in Fig.1, it can be divided into three layers: data storage, access interface and execution engine.

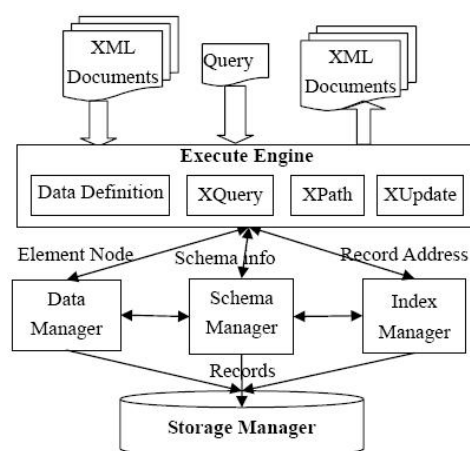


Fig. 1 OrientX Architecture

● Data storage

In OrientX, XML documents are organized in datasets according to their schema, which is a main feature of data repository. Interfaces for user to create and/or delete dataset are supplied. Documents should first be imported into certain datasets before any other jobs being done. Validation of document is done during importing process, if data does not conform to schema, a relevant error message is reported and importing process will be interrupted. OrientX also supports document export operation. The detail of the storage strategy will be discussed in section IV.

● Access interface

The access interface consists of data manager, schema manager and index manager, which is uniform to upper level applications and data storage is hidden by it.

Data manager offers a uniform interface to access the storage subsystem. The only entity type the query evaluator can obtain is a logical element, which is the same as node. And some navigation methods are bound to logical elements to support navigation on data tree.

OrientX system is based on XML schema, and schema manager is a key module in the system. The idea and implementation of schema manager are discussed later in section V.

- Execution engine

The execution engine composes of several modules. The data definition defines a document in the dataset. XPath and XQuery evaluators are the two main parts of execution engine. The basic strategy of XPath processing is navigation, however, in co-operating, we also develop the index-based strategy that acts effectively, the detail has been described in our previous work^[5].

Our previous work^[5,6] has discussed storage module, schema management, index module and optimization module in detail, so here we only briefly describe the above four modules, we will focus on the new modules of our OrientX system, which include the query processing engine, update handling strategy and the access control module.

3 OrientStore: A multi-granularity storage strategy

Several native storage strategies have been developed^[2,3,4,6]. According to the granularity of the records, these storage methods can be classified into Element-Based(EB),Subtree-Based(SB)and document-Based (DB). We observe that schema plays a key role in designing effective storage strategies for XML management systems. OrientX exploits schema information in the design and implementation of two storage strategies^[7]: Clustering Element-Based (CEB), and Clustering Subtree-Based strategies. OrientX also implements the above schema-independent storage strategies DEB and DSB. Detailed description can be found in our previous work^[6,8].

4 Schema Management

OrientX is schema-based. XML Schema strictly constrains the type and structure of data. So, data storing, retrieving and updating are all under the schema's guidance. Schema information can be used in data layout, in choice of index, in type checking, in user access control, and in query optimization. Schema in OrientX is consistent with the XML Schema standard. Detailed description can be found in our previous work^[9].

5 SUPEX: A Schema-Guided Index

Structure

SUPEX^[5] consists of two structures: structural graph (SG), and element map (EM). SG is constructed according to the schema, and represents the structure summary of XML data; each node in it represents a list of elements in XML document with the same tag name as the schema node. And EM provides fast entries to the nodes in SG. Further information can be found in our previous work^[5].

6 Query Processing

We have implemented several kinds of Query Processing Engines in our OrientX system based on the original work^[6]. Currently two XML query engines have been used extensively in OrientX, one for XPath, the other for XQuery. Here we only introduce the techniques utilized in our new navigation-based XQuery evaluator^[9].

- **Combine continuous steps in one XPath into a single path.** An XPath fragment has been separated into a series of step expressions, the navigation processing on it should be nested loop. But the only element we want is end step of the long XPath, all traversal for its ancestors is redundant. With the help of OrientStore and the SUPEX index, we can access any elements directly.
- **Reform syntax tree into reduced execution plan.** Structure of execution plan of navigation processing is similar to the structure of the syntax tree of XQuery statement. Especially, the key FLW(O)R structure in syntax tree is also the key operation in processing. Therefore, we use a "reduced"¹ syntax tree to denote the execution plan. Note some tricks in the reducing and reforming process:
 - 1) multi-processing units may be put together into one syntax node, for example, the FOR-Var binding, LET-Varbinding, WHERE-Predicate and the RETURN-EleConstruct form the FLWR node.
 - 2) multi-syntax nodes may be put into one processing unit; this is one kind of reducing operation

¹ Reduce here means omitting some hierarchies and nodes in the syntax tree.

We explain the reforming from syntax tree to execution plan with an XQuery example Q_1 :

```

<results>{
  let $doc := document("bib.xml")
  for $t in distinct-values($doc//book/title)
  let $p := $doc//book[title = $t]/price
  return
    <minprice title="{ $t }">
      <price>{ min($p) }</price>
    </minprice>
}</results>

```

Q_1 's execution plan is shown in Fig 2. We can see that it is pretty small and is similar to Q_1 in the structure. Most transforming actions are straightforward.

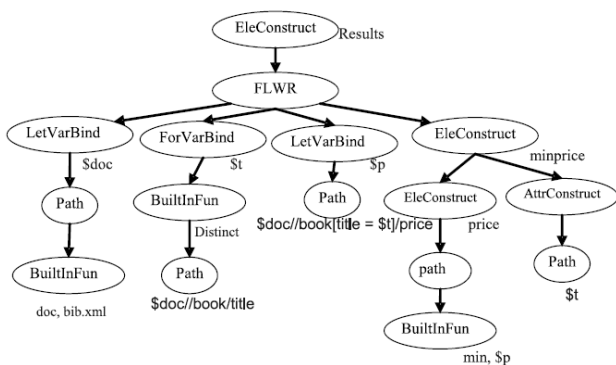


Fig. 2 Execution plan for Q_1

In execution, a pointer pointing to a node in the execution plan is hold during the whole query process. It points to the root in the beginning, and it travels through the plan tree in the top-down and left-right manner, if it encounters a source data node, for example, a ForVarBind node, then proper data is located by navigating on source document tree, on its way going, operations on subsequent "action" node are done on current located data. When the end of a FLWR is encountered, the pointer is redirected to beginning to this FLWR node except that there is no proper source data available. Query execution stops at the same time when the pointer stops.

The whole query process in OrientX is shown in Fig. 3, the white rectangle denotes processing modules and the rectangle in grey is auxiliary data or data structure. The chain composed of thin solid line is the data flow of XQuery statement, the chain composed of thick solid

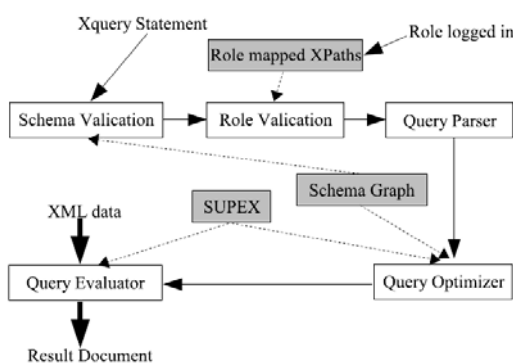


Fig. 3 Query process

Line is data flow of source XML data. The broken line denotes the fact that the auxiliary data is used in the target module.

7 Update

With the extensive use of XML in application over the Web, how to update XML data is becoming an important issue. So we currently implement a new update engine to the users. We extend XQuery with a FOR ...LET...WHERE...UPDATE structure for updates.

For XML documents with schema, we should check whether the update request violates the schema constraint. This is called update validation.

As elements are all encoded with region code in OrientX, we have to encode the newly inserted data right after inserting process, some efficient strategy is used in code encoding for updating during update/inserting^[10]. Currently, update for index is not implemented, it seems that it is similar to the update for common relational index.

The update process is as follows: Firstly we locate the referred element in document. Secondly we validate the update request according to its schema. The validation includes the check of the new data's inner constraint and whether the new data as a whole can appear at the specified location. The request being valid means that all constraints are fulfilled, if not, it is rejected. Finally, valid request is evaluated.

8 Node-Mapping Role Based Access Control

Access control module is an essential part for an integrated database system. Because of the different data model between relation data and XML data, the access control mechanism in relational database is not capable of managing XML data any more. Some important aspects need to be reconsidered, such as the granularity of access control, the semantic of authority, the relation among the rights on relative node in XML structure. At the same time, large data capacity and alteration of the data also should be taken care of. In this section, we will discuss a new Node-Mapping Role Based Access Control module for XML data that is utilized in the OrientX system.

The module is excited from the fact that the part of

dataset one can access in an XML document being best described by one node or several nodes in schema graph. It is true that if role *A* is the superior to role *B*, then the part of dataset that role *A* can access should be the superset of the part of dataset that role *B* can access. Therefore, we can map the two roles to two nodes in XML schema so that ancestor is the superior and descendant is the junior. In this way we can achieve both the excellence of Role and the convenience of XML data access controlling.

We give the definition of role here: A role is a set of triples $R = \{Node, Context, Action\}$, where *Node* denotes the tag name of the root of the subtree in schema graph; *Context* is an XPath locating the unique position of the node in the schema, and the *Action* represents a collection of allowed operation on the node, including reading, inserting, deleting and updating.

Access rules can be defined on roles(nodes), for example, we have used the Dynamic Separation of Duty Relation(DSD) characteristic to solve the problem of illegal association information accessing. Roles can be assigned to the user in two ways: positive and negative. The positive roles assign the actions user can do, and the negative roles assign the actions user can not do. In OrientX, general roles and DSD roles are all supported for compatibility. A user can choose many general roles during one session, and only one DSD role during one session.

9 Conclusion and Expectation

In this paper, we described the system structure and design of OrientX, an integrated, schema-based native XML database proposed by Renmin University of China. We have explored many issues on XML data management and proposed some new ideas. We also proved that schema plays a crucial role in XML data management system. Right now, the storage, query and index parts mentioned in this paper have already been implemented, and the query optimization, access control parts are being integrated and will be completed soon.

References

[1] Florescu D, Kossman D. Storing and Querying Xml Data Using an Rdbms [J]. *IEEE Data Eng Bull*, 1999,22(3):27-34
 [2] Jagadish H V, Divesh S, Wu Yuqing ,et al. Timber: A Native Xml Database[J]. *The VLDB Journal*, 2002, 11(4):274-291.

[3] Kanne C and Moerkotte G. *Efficient Storage of Xml Data*[M]. California: IEEE Computer Society,2000.
 [4] McHugh J, Abiteboul S, Goldman R, Quass D, et al. Lore: A Database Management System for Semi-structured Data [J]. *SIGMOD Record*, 1997,26(3):54-66 .
 [5] Wang Jing, Meng Xiaofeng, and Wang Shan. Supex: A Schema-guided Path Index for Xml Data[C] // *Proceedings of 28th International Conference on Very Large Data Bases(VLDB)*, Hong Kong ,2002.8.
 [6] Meng Xiaofeng and Wang Yu. OrientX: A Native XML Database System[C] // *Proceedings of 20th NDBC*, Chang Sha: 2003.10(Ch)
 [7] Li Quanzhong and Moon B. Indexing and Querying Xml Data for Regular Path Expressions[C] // *Proceedings of 27th International Conference on Very Large Data Bases(VLDB)*, Roma, 2001.9
 [8] Meng Xiaofeng, Luo Daofeng, An Jing ,et al. OrientStore: A Schema Based Native XML Storage System[C] // *Proceedings of 29th International Conference on Very Large Data Bases(VLDB)*, Berlin, 2003.9
 [9] Lu Shichao, Meng Xiaofeng, Lin Can , et al. Navigation Implementation for XQuery in OrientX [C] // *Proceedings of 20th NDBC* ,XiaMen,2004.10(Ch).
 [10] Jiang Yu, Luo Daofeng, Meng Xiaofeng, et al. Dynamically Updating Xml Data: Numbering Scheme Revisited[J]. *World Wide Web*, 2005, 8(1):.5-26