# NNQM: A novel non-navigating XML query model

LIN XUDONG[1,2] XU DE[1] WANG NING[1]
1. School of Computer and Information Technology, Beijing Jiaotong University
Beijing 100044, China
2. Department of Information Engineering, Environmental Management College of China, Qinhuangdao
Hebei 066004, China

*Abstract:* XQuery and Keyword query are the main methods to get XML data. But the former is too complicated for non-professional users and the latter doesn't have enough semantic capabilities to capture user's intents completely. Based on data models, this paper analyzes the reasons which lead to the drawbacks of existing query methods. And then, a novel auto-navigating XML query model is proposed. The query model is composed of the XML data model, the query language and the query algorithm. The data model stores XML documents in a clustering way according to the semantic relativity. The query language has the intuitive grammars and is suitable for non-professional users to use. The query algorithm has auto-navigating capability and can find the target data from XML documents without complete structure information.

*Keywords:* XML, non-navigating, query model, semantic relativity

## 1. INTRODUCTION

To get target data from XML documents, many query methods were proposed. These methods can be classified into two groups: navigating method and non-navigating method. XQuery[1] is generally acknowledged standard of the former, while keyword query methods such as XSEarch[2], Equix[3] belong to the latter.

XQuery is published by W3C. It has been supported by almost every native xml database and xml enabled database. Before using XQuery, users must learn its complicated syntax and have prior knowledge about the structure of XML documents, which prevents non-professional users from utilizing XQuery freely.

Compared with XQuery, keyword query methods [2-6] are more suitable for non-professional users. While their limited semantic capabilities couldn't capture users' intents completely, so keyword query methods often return results with too many irrelevant answers[7].

To address the existing problems in the mentioned query methods, we propose a non-navigating query model (NNQM) composed of the query language, its supporting data model and corresponding query algorithm. Our purpose is to enhance the auto-navigating capability of query algorithm and make non-professional users query from XML documents in a user-friendly way.

This paper is organized as follows: section 2 analyzes the drawbacks of existing query methods according to their data models. The details of data model is presented in section 3. Section 4 describes query language and query algorithm. Section 5 proposes some conclusions and outlines the future work.

## 2. DRAWBACKS ANALYSIS BASED ON DATA MODEL

From section 1, we know that almost every existing query method has some drawbacks. These drawbacks are mostly relevant to the limitations of their data models.

XQuery uses XML Data Model (XDM), a kind of tree data model, as its data model. Most of keyword query methods also use tree models to store XML documents [2-6]. In tree models, queries must begin from the root of the tree. If users want to query the target nodes of the tree, two methods can be chosen. The first one, which is used by XQuery, is to give the full paths from the root node to the target nodes. That is why XQuery users need to know the accurate structure of XML documents. The other one, used by keyword query methods, is to provide the names and values of target nodes. The keyword query algorithms begin from the root to

traverse all nodes of the tree to find the target nodes. In general, the overheads of keyword query algorithms are very high and how to find the meaningful lowest common ancestor (mlca) is really a challenge.

In Oracle 10g[8] and DB2 V9[9], SQL can be used to query XML data beyond relational model. Although relational model destroys the hierarchical relationships between different elements of XML documents, it also proposes flexibility: the queries can be executed by using any tables as their beginnings. The root nodes of XML documents aren't the unique entrances of queries yet. But SQL and relational model can't fully utilize this flexibility to simplify the querying process. When users want to query XML data distributed in many relational tables, they must provide every link between tables using "where" clause. In some respects, users have to do the same things as using XQuery, that is, providing the full paths of their target data.

## 3. DETAILS OF DATA MODEL

To improve the query method, we must address the issues of data models at first. The data model that supports humanized query should have the following features:

1) The model should support the query beginning from any node;
2) The model should store XML elements in a clustering way to improve traverse performance;
3) The model should have an auto-navigating capability to locate the target node by itself.

In terms of these features, we propose a novel data model: Almost Black Box XML Data Model (ABBXDM), shown in Figure 2. ABBXDM is made up of four units: Gates Unit (GU), Sets Unit (SU), One-way Pointers Unit (OPU), and Two-way Pointers Unit (TPU).

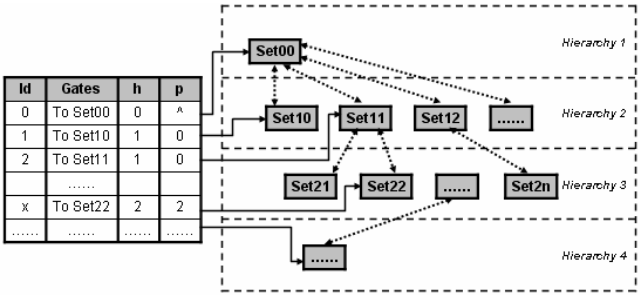


**Figure 1: ABBXDM**

GU is shown as a table in the left of Figure 1. It includes four columns, "Id", "Gates", "h" and "p", denote the unique flag, the name, the hierarchy and the parent of every gate respectively. If the gate is in the first hierarchy, the value of "p" will be "^".

The blocks named "SetX" in the right of Figure 1 are SU. These sets belong to different hierarchies.

OPU is made up of solid lines with one-way arrows which point from gates to sets in Figure 1.

The dash lines with two-way arrows which link the father sets and son sets in Figure 1 belong to TPU.

We can set up the storage structure of ABBXDM in the following steps after scanning the DTD files of XML documents [10][11]:

1) Create GU and SU;
   a) If the relationship between one element and its son element is one-multi (* or +), this son element will be put into a new set , and a corresponding tuple is added to the GU;
   b) If the relationship between one element and its son element is one-one (, or ?), this son element will be put into the same set as its father element's;
   c) The attributes of one element will be put into the same set of this element's.
2) Create the two-way pointers between the set coming from the father element and the sets coming from the son elements;
3) Create the one-way pointers between the sets and their corresponding gates.

Compared with tree models, ABBXDM congregates the semantic-relevant elements and attributes to one set and allows query begin from any set (gate). Simultaneously, ABBXDM also solves the main problem existing in relational models, that is, how to retain the hierarchical relationships between elements.

For the XML document [12] shown in Figure 2 and its DTD file shown in Figure 3, the ABBXDM with actual data is shown in Figure 4:

In the ABBXDM instance, the first column of every set is called "key column". We use "@" to denote the attribute of one element, and use " ↓ " to denote the son element. If the father element of one element is as same as "key column" element, the " ↓ " of this element can be omitted.

In fact, every set is composed of several blocks. Each block is composed of the data of all sub elements with the same name and the same father element.

```
<IndexTermsPage>
 <title id="00586000">The Sequoia 2000 Benchmark.</title>
 <authors>
 <author id="00">Michael Stonebraker</author>
 <author id="01">James Frew</author>
 <author id="02">Kenn Gardels</author>
 <author id="03">Jeff Meredith</author>
 </authors>
 <confName>ACM SIGMOD International Conference on Management of Data</confName>
 <confYear>1993</confYear>
 <volume>22</volume>
 <number>2</number>
 <initPage>2</initPage>
 <endPage>11</endPage>
 <fullText href="http://www.acm.org/pubs/articles/proceedings/mod/170035/p2-stonebraker/p2-stonebraker.pdf">
    <size>945 KB</size>
 </fullText>
 <abstract>This paper presents a benchmark that concisely captures the data base
requirements of a collection of Earth Scientists working in the SEQUOIA 2000
project on various aspects of global change research. This benchmark has the
novel characteristic that it uses real data sets and real queries that are
representative of Earth Science tasks. Because it appears that Earth Science
problems are typical of the problems of engineering end scientific DBMS users,
we claim that this benchmark represents the needs of this more general
community. Also included in the paper are benchmark results for three example
DBMSs: GRASS, IPW and POSTGRES.</abstract>
 <generalTerms>
 <term>Computer Applications, PHYSICAL SCIENCES AND ENGINEERING, Earth and atmospheric sciences.</term>
 <term>Computing Methodologies, IMAGE PROCESSING AND COMPUTER VISION, Miscellaneous, Landsat.</term>
 <term>Information Systems, DATABASE MANAGEMENT, Logical Design, Data models.</term>
 <term>Information Systems, DATABASE MANAGEMENT, Systems, Query processing.</term>
 <term>Information Systems, DATABASE MANAGEMENT, Systems, Transaction processing.</term>
 </generalTerms>
 <categoryAndSubjectDescriptors>
  <categoryAndSubjectDescriptorsTuple>
 <category>J.2</category>
 <content>Computer Applications, PHYSICAL SCIENCES AND ENGINEERING, Earth and atmospheric sciences.</content>
  </categoryAndSubjectDescriptorsTuple>
  <categoryAndSubjectDescriptorsTuple>
 <category>I.4.m</category>
```

**Figure 2: XML fragment**

```
<!ENTITY % carSet SYSTEM 'CarSet.cfg'>
%carSet;
 <!ELEMENT IndexTermsPage (title,authors,confName,confYear,volume,number,initPage,
 endPage,fullText,abstract,generalTerms,categoryAndSubjectDescriptors)>
 <!ELEMENT title (#PCDATA)>
 <!ATTLIST title id CDATA #IMPLIED>
 <!ELEMENT authors (author)*>
 <!ELEMENT author (#PCDATA)>
 <!ATTLIST author id CDATA #IMPLIED>
 <!ELEMENT confName (#PCDATA)>
 <!ELEMENT confYear (#PCDATA)>
 <!ELEMENT volume (#PCDATA)>
 <!ELEMENT number (#PCDATA)>
 <!ELEMENT initPage (#PCDATA)>
 <!ELEMENT endPage (#PCDATA)>
 <!ELEMENT fullText (size)?>
 <!ELEMENT size (#PCDATA)>
 <!ENTITY % Xlink
    " xml:link CDATA #FIXED 'simple'
     href CDATA #IMPLIED
     inline (true | false) #FIXED 'true'
    "
 >
 <!ATTLIST fullText %Xlink;>
 <!ELEMENT abstract (#PCDATA)>
 <!ELEMENT generalTerms (term)*>
 <!ELEMENT term (#PCDATA)>
 <!ELEMENT categoryAndSubjectDescriptors (categoryAndSubjectDescriptorsTuple)*>
 <!ELEMENT categoryAndSubjectDescriptorsTuple (category,content)>
 <!ELEMENT category (#PCDATA)>
 <!ELEMENT content (#PCDATA)>
```

**Figure 3: DTD**

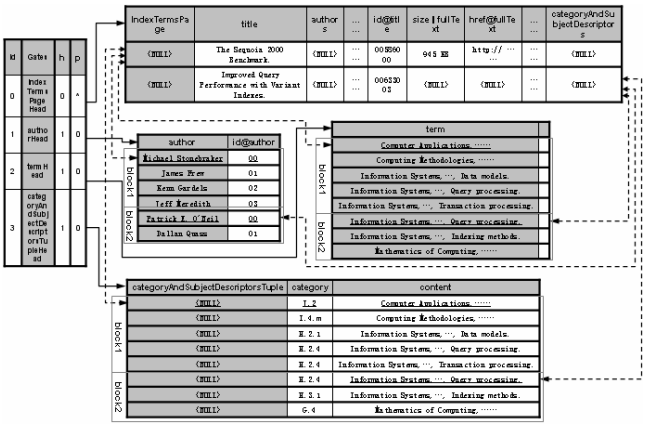


**Figure 4: ABBX instance**

## 4. QUERY LANGUAGE AND QUERY ALGORITHM

In ABBXDM, users can choose some gates to begin their queries, and the novel querying algorithm, Query By Itself (QBI) can find the target data automatically. Before describe the details of QBI, let's introduce the query language, Get-Enter-Where (GEW) at first.

The general GEW grammars have three formats:

1) The first format:
   - **Get** set1.*, set2.* …,
   - **Enter** gate1, gate2 …,
   - **Where** condition of gate1 **and/or/not** condition of gate2 & ⋯;
2) The second format (The syntax of "Enter" is different, adding "In" keyword.):
   - **Enter** (gate1 **In** ancestorGate1), (gate2 **In** ancestorGate2)
3) The third format (The syntax of "Enter" is different, adding "Of" keyword.):
   - **Enter** (gate1 **Of** fatherGate1), (gate2 **Of** fatherGate2) …,

Keywords "Get", "Enter" and "Where" are called main keywords of GEW. "Get" is used to tell QBI the target data, "Enter" tells QBI the beginnings of the queries, and "Where" is the restrictions of queries in QBI..

Keywords "In", "Of" are called additional keywords of GEW. When users wants to provide the ancestor gate of one gate, "In" can be used. It can help QBI to find the accurate gate when there are some gates with the same name. "Of" can tell QBI the father gate of one gate. "Of" can be used nestedly to help QBI find the accurate gate when nested relationships between elements of XML exist.

The query Q1 is described in the natural language:

**Q1**: find the titles and publishing years of the books whose authors' first names are "Joey" and titles of certain chapter are "Introduction".

Q1 can be denoted using GEW as below:

**Get** book.title, book.publishing_year,

**Enter** author, chapter,

**Where** author. first_name = 'Joey' **And** chapter. title = 'Introduction';

The query described in GEW will be submitted to QBI to execute. QBI has the capability to find the target data automatically. For QBI, SLCA (Smallest Lowest Common Ancestor) is a very important conception. Let's explain it at first:

For a tree, SLCA can be defined as the root node of the tree which fulfills the following two demands [13][14]:

1) The sub tree must contain all query keywords in their leaf

nodes;

2) The smaller sub trees which also contain all query keywords don't exist.

QBI will execute the query in the following steps:

1) Find the gates from "**Enter**" keyword;
2) Enter the gates and reach the corresponding sets;
3) Find the corresponding tuples of the sets using the conditions following "**Where**" keyword;
4) Begin from every corresponding tuples found in 3), find the corresponding father tuples respectively (the father tuples are themselves if the tuples belong to the root set), at the same time, compare the found father tuples:
   a) If these father tuples are in the same set:
      i. If an intersection can be found from these father tuples, go to 5);
      ii. If the intersection doesn't exist, **the result of this query is null, quit the algorithm;**
   b) Else, if these father tuples are in the different sets, continue to find the higher father tuples, and then go to 4.a);
5) Record the highest father set which includes intersection tuples as "BEGIN_SLCA";
6) Reorder the sets appearing in "**Get**" keyword in the order of hierarchies (from the highest one to the lowest one), but **to return the results with the right order**, record the original order of every set at the same time;
7) Find the top one from the reordered sets appearing in "**Get**" keyword, record it as "REAULT_SLCA";
8) Compare "BEGIN_SLCA" with "RESULT_SLCA":
   a) If "BEGIN_SLCA" is higher than "RESULT_SLCA": find the SLCA of "BEGIN_SLCA" and "RESULT_SLCA" which can be recorded as "REAL_SLCA";
      i. If "REAL_SLCA" equals "BEGIN_SLCA": begin from the corresponding tuples of "REAL_SLCA", continue to find the corresponding lower tuples until the tuples of "RESULT_SLCA", and then go to 9);
      ii. Else, if "REAL_SLCA" is higher than "BEGIN_SLCA": begin from the corresponding tuples of "BEGIN_SLCA", continue to find the corresponding higher tuples until the tuples of "REAL_SLCA", and then, begin from the corresponding tuples of "REAL_SLCA", continue to find the corresponding lower tuples until the tuples of "RESULT_SLCA", and then go to 9);
   b) Else, if "BEGIN_SLCA" is in the same hierarchy as "RESULT_SLCA":
      i. If "BEGIN_SLCA" equals "RESULT_SLCA": go to 9);
      ii. If "BEGIN_SLCA" is different from "RESULT_SLCA": find the "REAL_SLCA", begin from the corresponding tuples of "BEGIN_SLCA", continue to find the corresponding higher tuples until the tuples of "REAL_SLCA", and then, begin from the corresponding tuples of "REAL_SLCA", continue to find the corresponding lower tuples until the tuples of "RESULT_SLCA", and then go to 9);
   c) Else, if "BEGIN_SLCA" lower than "RESULT_SLCA": start from the corresponding tuples of "BEGIN_SLCA", continue to find the corresponding higher tuples until the tuples of "RESULT_SLCA", and then go to 9);
9) Begin from corresponding tuples of "RESULT_SLCA", get the target data according to the "**Get**" keyword in the order of hierarchy;
10) Reorder the results with the original order recorded in 6).

## 5. CONCLIUSION

In this paper, we propose a novel query model to make non-professional users can query XML documents in a user-friendly way. We believe NNQM is not only valuable for querying XML documents, but also can play an important role in updating XML documents.

We will continue to research and expose the details of NNQM in the future work..

## References

[1] W3C, XQuery1.0: An XML Query Language. http://www.w3.org/TR/2007/REC-xquery-20070123/, 2007

[2] Sara Cohen, Jonathan Mamou, Yaron Kanza, Yehoshua Sagiv, XSEarch: A Semantic Search Engine for XML, In Proceedings of the 29th International Conference on Very Large Databases (VLDB '03), 2003, pp.45-56

[3] Sara Cohen, Yaron Kanza, Yakov Kogan, Werner Nutt,

Yehoshua Sagiv, Alexander Serebrenik, EquiX: A search and query language for XML, Journal of the American Society for Information Science and Technology, vol.53, no.6, 2002, pp.454–466

[4] Chavdar Botev, Jayavel Shanmugasundaram, Context-Sensitive Keyword Search and Ranking for XML, 8th International Workshop on the Web and Databases (WebDB 2005), 2005

[5] Lin Guo, Feng Shao, Chavdar Botev, Jayavel Shanmugasundaram, XRANK: Ranked Keyword Search over XML Documents, SIGMOD, 2003

[6] Li Xiaoguang, Yu Ge, Gong Jian, Wang Daling, Bao Yubin. Towards Effective and Efficient NFS Querying on XML Document, Chinese Journal of computers, vol.30, no.1, 2007, pp.57–67 (in Chinese with English abstract)

[7] Yunyao Li, Cong Yu, H.V.Jagadish, Schema-Free XQuery, In Proceedings of the 30th International Conference on Very Large Databases (VLDB '04), 2004

[8] Oracle White Paper, Mastering XML DB Queries in Oracle Database 10g Release 2, 2005

[9] DB2 Redbooks, DB2 9 pureXML Guide, 2007

[10] Mustafa Atay, Artem Chebotko, Dapeng Liu, Shiyong Lu, Farshad Fotouhi, Efficient schema-based XML-to-Relational data mapping. Information Systems, vol.32, no.3, 2007, pp.458 – 476

[11] J. Wang, X. Meng, S. Wang, SUPEX: A Schema-Guided Path Index for XML Data, Doctoral Poster, In Proceedings of the 28th International Conference on Very Large Databases (VLDB '02), 2002

[12] DBLP dataset, http://dblp.uni-trier.de/xml/

[13] Lin Guo, Feng Shao, Chavdar Botev, Jayavel Shanmugasundaram, The Index-Based XXL Search Engine for Querying XML Data with Relevance Ranking, EDBT, 2002, pp.477–495

[14] Yu Xu, Yannis Papakonstantinou, Efficient Keyword Search for Smallest LCAs in XML Databases, SIGMOD, 2005