Data Mining Techniques of XML Data Warehouse

Loredana MOCEAN
Babeş - Bolyai University of Cluj – Napoca
Business Information Systems Department
mloredana@econ.ubcluj.ro

In this paper we will try to answer to some questions referring to hypertext. In the same time our purpose is to explain what we mean by a query in the context of XQL, and to present a simple model, which will serve as a framework for the future research.
Keywords. *techniques, XML, XQL, query, data mining.*

1 Introduction

An important part of the advantages of hypertext against the print text are:
• Nonlinear form of the hypertext offers efficient capabilities of go through the contents;
• Electronically medium can stock an important quantity of information;
• Hypertext offers a better visualization of the content and a rapid navigation, however complex would be the documents, taking count of the specifications of the users;
• The queries, filters, different preferences and annotations of the users cane be re-use whenever need, and can be stored as part of the hypertext structure of the used documents.

2. Results and discussions

The main problems wherewith the programmers are confronted in what looks the hypertext, are:
• The conversion, automatically in general, of the plain text in hypertext format;
• The linearization of hypertext;
• The design of hypertext documents;
• The concurrent accessing of hypertext database in distributed context;
• The building of some optimal mechanisms for intelligent searching and querying of the hypertext information;
• The supporting of the multimedia extensions;
• The presenting of the hypertext documents in an easy shape for the users.

An hypertext system is compound by **nodes** (concepts) and **links** (relationships). A node represents a unique concept (an idea) and may contain any type of information (text, graphics, animation, audio, video, images or prams).
All these elements has associated a type (detail, proposition, collection, observation etc.), a semantic information. The nodes are connected with other nodes through the links. The source node is called *reference* and the destination node is called *referent* or *anchors.*
The modality of stocking the information in the nodes varies from system to system, but the most used techniques use the markup languages (SGML - Standard Generalized Markup Language and XML - *Extensible Markup Language*), the current standards built on these languages are *HyTime (Hypermedia/Time- based Structuring Language), MHEG (Multimedia* and *Hypermedia Information Coding Expert Group), HTML (HyperText Markup Language).*
The major advantage is assuring of hardware and software platform independence, the owner format lead to difficulty in navigation, search or support. Once, the nodes can contains multimedia information, the hypermedia systems must be sufficiently of flexible for supporting many graphical, audio and video formats.
The next problems we want to discuss are the query languages for XML. We are interest in that because we want to structure the data for the Web and to see the information in easy form and reports.
The solution is XML and, in achieving this desideratum, we can use two important techniques:

❖ The mark of web information : stocking, formalization and transforming;
❖ The search upon different criteria: XML based languages for making queries.
The XML is the standard meta-language for annotation the WEB and has some important properties:
❖ The requirement of offering formal definitions for all the used concepts;
❖ Possibilities of automatic processing;
❖ Inherent efforts of development of some specialized techniques and languages in index and query of the contents of web documents;
We can see the web pages as XML documents. As we know, documents can contains many characters. The most important things of which we abide are:
❖ The extracting of the data from very extensive documents, in which we process the structural part and the plain text;
❖ The transfer of XML documents between documents with different ontologies;
❖ The integration of data which are gathered from multiple sources;
❖ The transmitting of large quantities of XML data to users;
❖ Techniques of transmitting of queries to XML resources.
It is sometimes necessary to extract subsets of the data stored within an XML document. A number of languages have been created for querying XML documents. The most important methods of querying are:
1. The selection based on keywords;
2. The extrapolation of the methods of interrogate the databases, inclusively relational (with Web3QL, WebSQL, XQL, XML-QL);
3. Graphical modeling of search demands with XMLGL, WQGL, WQFL;
4. Re-iteration of previous navigation.
The requirements are shown in figure 1.

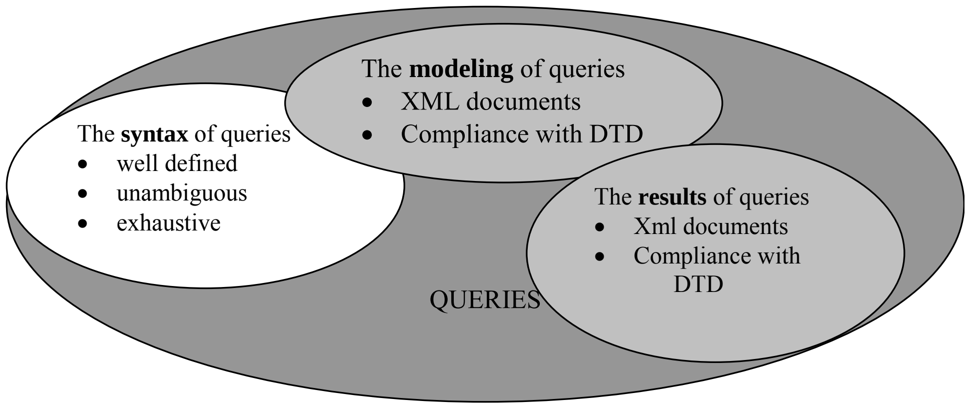

Fig.1. The requirements of correct queries

XQL was designed as a general-purpose query language for XML. During the design of the language, four big types of queries were designed to determine the requirements, as we see in table 1.

Name	Type	Characteristics	Describe
Q1	Query within a single document	- used in scripting languages - may define different access right - provide powerful non-procedural access to document	- the name of elements - the name of attributes - the type of a node - the content of the node - the relationships between nodes
Q2	Query within collections of documents	- used in document assembly - used in queries performed on a single website or across websites - used in scripting languages to provide powerful non-procedural access to document data	- a set of documents or a set of nodes within multiple documents - need to be able to address the individual documents

Q3	Addressing within or across documents	- used for referencing known locations in documents using hyperlinks - allow anchors to be placed within documents - address any node in the document - specifies relative and absolute path in the document	- the name of elements - the name of attributes - the type of a node - the content and value of the node - the relationships between nodes by the above criteria, including hierarchy, sequence and position
Q4	XSL Patterns	- used to specify tree-to-tree transformations on documents	- the name of elements - the name of attributes - the type of a node - the content of the node - the relationships between nodes

Table 1. Types of queries in XQL

Query language should be useable in a variety of environments: in programming language and scripting language strings, in URLs, and as attributes in documents or XSL templates. XQL queries can easily be typed as strings on a command line, generated by graphical query interfaces, or embedded as strings in programs.
As an example, we propose to write a query that returns *Nrfactura* that are children of *Facturi* elements, in different ways, to be examples for implementing:

Query method	**Query description**
XQL query	`Facturi/Nrfactura`
Java String	`String qstring = Facturi/Nrfactura;`
As a part of an URL	`http://www.bazadedate.com/docs#Facturi/Nrfactura`
Embedded in attributes of HTML or XML documents	`<a href>=”http://www.bazadedate.com/docs#Facturi/Nrfactura”>`

Table 2. Examples of query methods

In XQL we operate with the following concepts:
❖ The *database* is a set of one or more XML documents;
❖ *Queries* are done in XQL, a query language that uses the structure of XML as a basic model;
❖ A query is given *a set of input nodes* from one or more documents, and examines those nodes and their descendants;
❖ *The result of a query* is a set of XML document nodes, which can be wrapped in a root node to create a well-formed XML document.
We will try to illustrate some of these concepts. The input to the query is `<Facturi>` element, which is the root of the main document. The search context is:

```
<Facturi>
  <Campuri>
      <NrFactura>1000</NrFactura>
      <DataFactura>12.01.2007</DataFactura>
      <IdPartener>100</IdPartener>
      <TipFactura>I</TipFactura>
  </Campuri>
  <Indecsi>
      <NrFactura>1000</NrFactura>
      <IdPartener>100</IdPartener>
  </Indecsi>
</Facturi>
```

The result of the query `<Facturi>`, is the set of all `<Facturi>` elements in the search context. In our example, the result set is equivalent to the search context, the previous document, the same one. The search context and the result set contain one node each.
The result set contains only one node. But a query returns more than one node, though, a text representation of the result set is not a well-formed XML document, because an XML document can have only one root node. We will do a more complicated query, using a wildcard, regardless of element name, and the parent/child operator ('/'). The following query searches for all children of `<Indecsi>` elements which are children of `<Facturi>` elements:
`Facturi/Indecsi/*`
The result set is:

```
<NrFactura>1000</NrFactura>
<IdPartener>100</IdPartener>
```

This result set contains two nodes and it is not a valid XML document. We must wrap the nodes of the set in a common root element. We will have a valid XML document. So, the result document of an XQL query always wraps the nodes of the result set in an **`<xql:result>`** element:

```
<xql:result>
<NrFactura>1000</NrFactura>
<IdPartener>100</IdPartener>
</xql:result>
```

When a query has operators, evaluation becomes a more complex operation. When an operator is evaluated for a given search context, it selects the appropriate search context for each of its operands. The search context for which an operand is evaluated need not be identical to the search context of the query.
Most XQL expressions will evaluate either to a set of nodes or a Boolean value. All XQL query expressions may be said to evaluate to true or false.
In XQL we can find two kinds of return operators. The *shallow return operator* ("?") returns just the node to which it is applied. The *deep return operator* ("??") returns the element and all its children. Return operators can simplify queries for complex document structures. Here is an example for our discussion of return operators:

```
<?xml version="1.0">
<DocumenteFacturate>
<Facturi>
<IdPartener>100</IdPartener>
  <Campuri = 2>
  <Intrare cantitate=1>
      <NrFactura>1000</NrFactura>
      <DataFactura="12.01.2007"/>
      <TipFactura>I</TipFactura>
  </Intrare>
  <Intrare cantitate=2>
      <NrFactura>1001</NrFactura>
      <DataFactura="12.01.2007"/>
      <TipFactura>I</TipFactura>
  </Intrare>
</Facturi>
<Facturi>
<IdPartener>101</IdPartener>
  <Campuri = 2>
  <Intrare cantitate=1>
      <NrFactura>1002</NrFactura>
      <DataFactura="13.01.2007"/>
      <TipFactura>I</TipFactura>
  </Intrare>
  <Intrare cantitate=2>
      <NrFactura>1003</NrFactura>
      <DataFactura="13.01.2007"/>
      <TipFactura>I</TipFactura>
  </Intrare>
  </Campuri>
</Facturi>
<Indecsi>
      <NrFactura>1000</NrFactura>
      <IdPartener>100</IdPartener>
</Indecsi>
</DocumenteFacturate>
```

We want to see all the entries (the date of the document) which is recorded on an element called `Facturi`. We can successfully use the following query:
`Facturi//DataFactura`
The results for the above query are shown bellow:

```
<xql:result>
      <DataFactura="12.01.2007"/>
      <DataFactura="12.01.2007"/>
      <DataFactura="13.01.2007"/>
      <DataFactura="13.01.2007"/>
</xql:result>
```

A shallow return ("?") on the `<Factura>` element returns the `<Factura>` element,

providing an element within which the products can be listed:
`Facturi??//DataFactura`
These are the results:

```
<xql:result>
<Facturi>
      <DataFactura="12.01.2007"/>
      <DataFactura="12.01.2007"/>
</Facturi>
<Facturi>
      <DataFactura="13.01.2007"/>
      <DataFactura="13.01.2007"/>
</Facturi>
</xql:result>
```

If we want to see the customer for each document `<Factura>`, the date and the Id number, this can be done by specifying both `<DataFactura>` and `<IdPartener>` using the deep return operator:
`Facturi?[IdPartener??]//DataFactura??`
The results are:

```
<xql:result>
<Facturi>
<IdPartener>101</IdPartener>
      <DataFactura="12.01.2007"/>
      <DataFactura="12.01.2007"/>
</Facturi>
<Facturi>
<IdPartener>101</IdPartener>
      <DataFactura="13.01.2007"/>
      <DataFactura="13.01.2007"/>
</Facturi>
</xql:result>
```

3. Conclusions

As we saw in this paper, XQL is a query language designed specifically for XML documents. In the same sense that SQL is a query language for relational database, XQL is a query language for XML documents. The basic constructs of XQL correspond directly to the basic structures of XML. Since queries, transformation patterns, and links are all based on patterns in structures found in possible XML documents, a common model for the query language used in these paper is both possible and desirable, and a common syntax to express the patterns expressed by that model simplifies the task of the user who must be able to use a variety of XML query technologies.
We have described a few things about XQL, keeping in mind its continuity with relational database standards such as SQL.

References.

[Brut07] Brut M., Buraga S., *Limbaje de interogare XML,* whitepaper, 2007
[Buraga06] Buraga S., *Tehnologii XML,* Ed. Polirom, 2006
[Buraga07] Buraga S., *Incursiune în teoria hypetextului,* whitepaper, 2007
[Mocean06] Mocean L. (coord), ş.a., *Baze de date şi programare – Culegere de probleme,* Ed. Risoprint, Cluj – Napoca, 2006
[Phillips01] Phillips L.A., *XML*, Ed. Teora, 2001
http://www.texcel.ro