Querying Large Collections of Semi-structured and XML Data

Evangelos Kotsakis
Contents

• Semistructured Data
• Extensible Mark-up technology
  – XML
  – DTD
  – Style-sheets
  – XML APIs
  – XQL
• XML Schema Directory (XSD)
• References
Semistructured Data
Semistructured Data

- Schemaless, self-describing data

```
{name:"Antoni", tel:4566012, email:"Antoni.Wolski@vtt.fi" }
{name:"Jukka", tel:4565936, email:"Jukka.Kiviniemi@vtt.fi"}
{name: { first:"Antti", last:"Pesonen"}, tel:4566027, email:"Antti.Pesonen@vtt.fi"}
{name:"Jorma", tel:4566034, tel:4566034}
```
Semistructured database

db: {
  person: {name:"Antoni", tel:4566012, email:"Antoni.Wolski@vtt.fi" },
  person: {name: {first:"Antti", last:"Pesonen"}, tel:4566027, email:"Antti.Pesonen@vtt.fi"},
  person: {name:"Jorma", tel:4566034, tel:4566034}
}
Relational Data represented as Semistructured (1)

db as a set of relations. Each relation is a set of rows

db:

r1(a,b,c) r2(c,d)

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
</tr>
</thead>
<tbody>
<tr>
<td>a1</td>
<td>b1</td>
<td>c1</td>
</tr>
<tr>
<td>a2</td>
<td>b2</td>
<td>c2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>c</th>
<th>d</th>
</tr>
</thead>
<tbody>
<tr>
<td>c2</td>
<td>d2</td>
</tr>
<tr>
<td>c3</td>
<td>d3</td>
</tr>
<tr>
<td>c4</td>
<td>d4</td>
</tr>
</tbody>
</table>

10 Mar 2000  E. Kotsakis
Relational Data represented as Semistructured (2)

db as a set of rows. Each row is an instance of a relation

```
row:r1:{a:a1,b:b1,c:c1},
row:r1:{a:a2,b:b2,c:c2},
row:r2:{c:c2,d:d2},
row:r2:{c:c3,d:d3},
row:r2:{c:c4,d:d4}
```

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
<th>c</th>
</tr>
</thead>
<tbody>
<tr>
<td>a1</td>
<td>b1</td>
<td>c1</td>
<td></td>
</tr>
<tr>
<td>a2</td>
<td>b2</td>
<td>c2</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>c</th>
<th>d</th>
</tr>
</thead>
<tbody>
<tr>
<td>c2</td>
<td>d2</td>
<td></td>
</tr>
<tr>
<td>c3</td>
<td>d3</td>
<td></td>
</tr>
<tr>
<td>c4</td>
<td>d4</td>
<td></td>
</tr>
</tbody>
</table>

r1(a,b,c)  
<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
</tr>
</thead>
<tbody>
<tr>
<td>a1</td>
<td>b1</td>
<td>c1</td>
</tr>
<tr>
<td>a2</td>
<td>b2</td>
<td>c2</td>
</tr>
</tbody>
</table>

r2(c,d)  
<table>
<thead>
<tr>
<th>c</th>
<th>d</th>
</tr>
</thead>
<tbody>
<tr>
<td>c2</td>
<td>d2</td>
</tr>
<tr>
<td>c3</td>
<td>d3</td>
</tr>
<tr>
<td>c4</td>
<td>d4</td>
</tr>
</tbody>
</table>
Relational Data represented as Semistructured (3)

db as a set of relation instances

r1(a,b,c)  r2(c,d)

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
<th>c</th>
</tr>
</thead>
<tbody>
<tr>
<td>a1</td>
<td>b1</td>
<td>c1</td>
<td></td>
</tr>
<tr>
<td>a2</td>
<td>b2</td>
<td>c2</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>c</th>
<th>d</th>
</tr>
</thead>
<tbody>
<tr>
<td>c2</td>
<td>d2</td>
<td></td>
</tr>
<tr>
<td>c3</td>
<td>d3</td>
<td></td>
</tr>
<tr>
<td>c4</td>
<td>d4</td>
<td></td>
</tr>
</tbody>
</table>

db:

{  
  r1:{a:a1,b:b1,c:c1},  
  r1:{a:a2,b:b2,c:c2},  
  r2:{c:c2,d:d2},  
  r2:{c:c3,d:d3},  
  r2:{c:c4,d:d4}  
}
OO Data represented as Semistructured

db:

   person:&01{name:"Mary", age:45, child:&02, child:&03},
   person:&02{name:"John", age:17, relatives:{mother:&01, sister:&03}},
   person:&03{name:"Jane", country:"Canada", mother:&01}


Extensible Markup Technology
New Needs in processing Web data

- Applications need to manipulate web data
- Need to exchange data in an unstructured manner through the WEB
- HTML incapability. HTML was designed to describe the presentation of data
- We need to come up with a new Mark-up technology, which will
  - allow us to easily process web data.
  - be compliant with the existing technology
  - be able to express the content of data
eXtensible Markup Language (XML)

- XML was designed specifically to describe the content, rather than the presentation of the data.
- XML differs from HTML in three major aspects
  - New tags may be defined at will
  - Structure can be nested to arbitrary depth
  - A XML document can contain an optional description of its grammar.
- XML Data may be transformed to HTML though stylesheets.
XML Elements

- The basic component in XML document is the *element*.
- An element is bounded by matching tags (start/end tag) and it may contain
  - a text
  - other elements
  - a mixture of text and elements

```xml
<person>
  <name> Mary </name>
  <age> 45 </age>
  <email> mary@xyz.com </email>
</person>
```
XML Element Attributes

• An XML element may have one or more attributes
• XML attributes are viewed as element properties and they are defined only in the start-tag

```xml
<employees>
  <person language="Greek"> Vangelis </person >
  <person language="Finish"> Antoni </person >
</employees>
```

• An attribute may only occur once within a tag
• The value associated with an attribute is always a string

```xml
<employees>
  <person > Vangelis
      <language> Greek </language>
  </person >
  <person > Antoni
      <language> finish </language>
  </person >
</employees>
```
Ambiguity

• The introduction of XML attributes causes an ambiguity as to whether to represent information as attributes or elements in data exchange.

```xml
<person>
  <name> Mary </name>
  <age> 45 </age>
  <email> mary@xyz.com </email>
</person>
```

```xml
<person age="45">
  <name> Mary </name>
  <email> mary@xyz.com </email>
</person>
```

```xml
<person name="Mary" age="45" email="mary@xyz.com"/>
```
<employees>
  <person>
    <name>Vangelis</name>
    <language>Greek</language>
  </person>
  <person>
    <name>Antoni</name>
    <language>finish</language>
  </person>
</employees>
XML References

- References between XML elements can be defined by using the special element attributes ID/IDREF

```
<state ID="S1">
  <sname>Nevada</sname>
  <scode>NE</scode>
</state>

<city ID=C1>
  <cname>Carson City</cname>
  <ccode>CCN</ccode>
  <state-of IDREF="S1"/>
</city>
```
XML & Semistructured Data

- Semistructured data do not have attributes
- Semistructured data are unordered collections

```
<person>
  This is my best friend
  <name>Alan</name>
  <age>32</age>
  I am not sure about the following email
  <email>Alan@xyz.com</email>
</person>
```
Constraints for Making XML Documents

• Tags have to nest properly
• attributes have to be unique

XML documents satisfying the above constraints are well-formed
Other XML Constructs

Comments: <!-- this is a comment -->

Processing Instructions: <!xml-stylesheet href="book.ccs" type="text/css" ?>

optional starting construct: <?xml version="1.0" ?>

CDATA construct <![CDATA[ everything within is interpreted literally ]]>
A DTD defines the grammar of a XML document

```xml
<!DOCTYPE name [ mark-up declarations ]>

<!ELEMENT db(r1*,r2*)>
<!ELEMENT r1(a,b,c)>  
<!ELEMENT r2(c,d)>    
<!ELEMENT a(#PCDATA)> 
<!ELEMENT b(#PCDATA)> 
<!ELEMENT c(#PCDATA)> 
<!ELEMENT d(#PCDATA)> 

<!ELEMENT elem_name EMPTY>

<!DOCTYPE db SYSTEM "file_name.dtd"> 
```
Regular Expressions in DTD declarations

- $e^*$: zero or more $e$
- $e^+$: one or more $e$
- $e?$: zero or one $e$
- $e_1 | e_2$: alternation - $e_1$ or $e_2$
- $e_1 , e_2$: concatenation - $e_1$ and $e_2$

Recursive definition is allowed

```xml
<!ELEMENT node(leaf | (node+))>
<!ELEMENT leaf(#PCDATA)>
```

```xml
<n<br/>(node>
  <node>
    < node > L1 </ node >
  </node>
  < node > L2 </ node>
  < node > L3 </ node >
</node>
</node>
<n
```
Declaring Attributes in DTDs

```xml
<!ATTLIS elem_name att_name att_type att_qualifier>
```

att_type: CDATA, PCDATA etc.

att_qualifier: #REQUIRED, #IMPLIED etc.

Three attributes are of particular importance:

- **ID** (any string)
- **IDREF** (element ID)
- **IDREFS** (list of IDs)
Valid XML Documents

Well-formed XML documents that have a DTD and conform to it are said to be Valid XML documents.
Limitations of DTDs as Schemas

- DTDs impose order
- The only atomic type is #PCDATA
- We cannot define constraints (value ranges etc.)
- There is no constraints in the use of IDREF/IDREFS
Stylesheets

Stylesheets are special purpose languages expressing the conversion of XML documents into HTML

There are two such languages

- Cascading Style Sheets (CSS)
- XML Stylesheet Language (XSL)
Cascading Style Sheets (CSS)

CSS associates each element type with a presentation

```xml
<STYLE TAG="address" >
  <FONT-SIZE V=10/>
  <FONT-FAMILY V="Times New Roman"/>
</ STYLE >

<?STYLESPEC "style_name" "stylefile.ssh" ?>
```
XML Stylesheet Language (XSL)

- XSL allows user to write transformations from XML to HTML
- A XSL program is a set of template rules. Each rule consists of
  - pattern
  - template

```xml
<xsl:template match= pattern >
  template
</xsl:template>
```
Applying XSL

```xml
<bib>
  <book>
    <title>T1</title>
    <author>A1</author>
    <author>A2</author>
  </book>
  <paper>
    <title>T2</title>
    <author>A3</author>
    <author>A4</author>
  </paper>
  <book>
    <title>T3</title>
    <author>A3</author>
    <author>A5</author>
  </book>
</bib>

<xsl:template match="/bib/*/title">
  <result>
    <xsl:value-of />
  </result>
</xsl:template>

<result>
  <title>T1</title>
  <title>T2</title>
  <title>T3</title>
</result>
```
XSL patterns

* any element
/e root
/e e is the top element, which is child of the root
/e1 // e2 e2 is a child of e1
/e1 // e2 e2 is a descendant of e1
//e e at any depth
@a @a matches the attribute a
/e1 | e2 alternation- e1 or e2

<!-- some comments -->
<bib> ... </bib>
<!-- some other comments -->
There are two XML Programming Interfaces

• Simple API for XML (SAX)
• Document Object Model (DOM)
Simple API for XML (SAX)

SAX API is a standard for parsing XML data
SAX defines a set of callback functions
The processing application should implement these functions.
Typical functions are
  – OnStartTag()
  – OnEndElement() etc.
Document Object Model (DOM)

- DOM represents an OO view of the XML document.
- The document is assumed an object whose behaviour is expressed via DOM functions

There are functions (DOM interfaces) for processing
- Nodes \{parentNode(), childNode() etc \}
- Elements
- Attributes
- Character Data
An XQL expression has the following general structure

```
<bib>  <book>  <title>T1</title>  
     <author>A1</author>  
     <author>A2</author>  
  </book>  
  <paper>  <title>T2</title>  
     <author>A3</author>  
     <author>A4</author>  
  </paper>  
  <book>  <title>T3</title>  
     <author>A3</author>  
     <author>A5</author>  
  </book>  
</bib>  
```

```
<bib>  <title>T1</title>  
</bib>  
<bib>  <title>T2</title>  
</bib>  
<bib>  <title>T3</title>  
</bib>  
```

```
/bib/*/title  
/[bib/*/author='A3']  
```

```
<title>T1</title>  
<title>T2</title>  
<title>T3</title>  
<title>T2</title>  
<title>T3</title>  
```
# XQL constructions

XQL patterns are the same as in XSL

<table>
<thead>
<tr>
<th>Arithmetic operations:</th>
<th>=, !=, &lt;, &lt;=, &gt;, &gt;=</th>
</tr>
</thead>
<tbody>
<tr>
<td>Logical operations:</td>
<td>$and$, $or$, $not$, $eq$, $ne$</td>
</tr>
<tr>
<td>Support functions:</td>
<td>text(), index(), end(), nodeType()</td>
</tr>
<tr>
<td></td>
<td>date(), textNode(), comment(), count()</td>
</tr>
</tbody>
</table>

- `//author[text()='A1']` find authors whose text is 'A1'
- `//author[index() = 0]` find the first author within author's parent node
- `//author[end()]` find the last author within author's parent node

`nodeType()` could be `element(1)`, `attribute(2)`, `text(3)`, `PI(7)`, `comment(8)`, `document(9)`

- `//paper[author!nodeType()=1]` find papers with an author as a sub-element
**XQL set functions**

set functions: $\text{union}$, $\text{intersect}$

$\text{union}$ is also abbreviated as $|$

$\text{//(paper | book)/author}$ find the authors of papers or books

$\text{//bib/(paper | book)}$ find the papers or books of bib

$\text{//bib/(paper/author $\text{intersect}$ book/author)}$ find the authors who have written a paper and a book
XML Schema Directory (XSD)
XML Schema Directory (XSD)

- XSD is a data structure for indexing XML schemata
- XSD structure allows the execution of XML queries over a corpus of XML documents

Needs for such a structure
- The proposed query languages assume that the input to query is a set of known documents
- To execute XML queries, the query engine should be supplied with
  - the queries string and
  - the URL data sources on which the queries will perform.
XSD Core Idea

• A XML corpus consists of many XML documents of different types.
• A document type represents a piece (knowledge component) of the domain knowledge of the corpus.
• If the input documents to a query are not known, we need to go through exhaustive searching in order to answer the query.

The Question is ..... 

Is there any way to classify these knowledge component to accelerate query processing
XSD structure

- XSD is used to find relevant XML documents
- XSD is based on clustering XML Schemata
- The output of the clustering process is a hierarchical (tree-like) structure, in which, each non-leaf node is a merger XML schema and each leaf node is a basic XML schema
- Query evaluation is performed by matching the query to a merger schema and then to the children of it recursively until we reach the leafs. Those leafs that match the query are considered relevant and thus any XML document derived from them.
- Then the Query may performed separately on the relevant documents found by using XQL.
XSD tree

XML documents
XML Query Structure

/Portfolio/Stock/Name[/Portfolio/Stock//DMin > 10000]

<Name> DJ INDU AVERAGE </Name>
<Name> NIKKEI 225 INDEX </Name>
<Portfolio>
    <Stock Market="SAO" Ticker="^BVSP">
        <Name>BRSP BOVESPA IND</Name>
        <Last>18008.73</Last>
        <Volume>N/A</Volume>
        <Change>-0.57%</Change>
        <DMin>18008.73</DMin>
        <DMax>18297.38</DMax>
        <YMin>15349.78</YMin>
        <YMax>18885.84</YMax>
    </Stock>
</Portfolio>
Similarity between XML Schemata

• Similarity between XML schemata is based on tree distances

• The distance between two XML schemata is based on the edit operations that should be performed to one of them in order to obtain the other.

• An edit operation on a XML schema may be
  – insertion,
  – deletion
  – substitution of nodes
Advantages of the XSD structure

- **No need to parse non-relevant documents.** This results in executing XML queries faster since the search space is limited only to the relevant XML documents.

- **The accuracy in answering XML queries is high.** The result set is exactly the same as the one that would be obtained through exhaustive searching if the proposed XSD structure were not used.

- **Maintaining the XSD structure is not difficult** since new XML schemata may be added or old one deleted without performing time-consuming operations that requires reorganization of the XSD structure.
References
WWW LINKS

WWW Consortium : http://www.w3.org/
XML home page: http://www.w3.org/TR/1998/REC-xml-19980210
XQL home page : http://www.w3.org/TandS/QL/QL98/pp/xql.html
Stylesheets: http://www.w3.org/Style/
DOM : http://www.w3.org/DOM/

BOOKS

Serge Abiteboul, Peter Buneman, Dan Suciu, *Data on the Web : From Relations to Semistructured Data and Xml*. Morgan Kaufmann, Published October 1999, 258 pages, ISBN 155860622X


Papers


[BAS99] Peter Buneman, Serge Abiteboul and Dan Suciu, Data on the Web:From Relations to Semistructured Data & XML, 10/1999, Morgan Kaufmann

[BC00] Angela Bonifati, Stefano Ceri, Comparative Analysis of Five XML Query Languages, ACM Sigmod Record, March 2000


