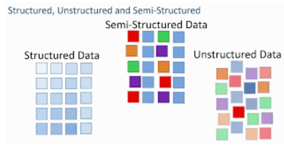
COMP9319 Web Data Compression and Search

Semistructured / Tree Data, XML, XPath

Semistructured Data

- Emails, HTML, JSON, XML, RDF, ...

Unstructured text



JSON


```
{
    "orders": [
        {
            "orderno": "748745375",
            "date": "June 30, 2088 1:54:23 AM",
            "trackingno": "T[illegible]",
            "custid": "11045",
            "customer": [
                {
                    "custid": "11045",
                    "fname": "Sue",
                    "lname": "Hatfield",
                    "address": "1409 Silver Street",
                    "city": "Ashland",
                    "state": "NE",
                    "zip": "68[illegible]3"
                }
            ]
        }
    ]
}
```


HTML

html5temp.html

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title> A Tiny HTML Document </title>
<link href ="styles.css" rel="stylesheet">
<script src="scripts.js"></script>
</head>

<body>
<p>Let's rock the browser, HTML5 style.</p>
</body>
</html>
```


RDF

```
<?xml version="1.0"?>
<rdf:RDF
     xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
     xmlns:dc="http://dublincore.org/documents/1998/09/dces/#">
  <rdf:Description>
    <dc:title>Flag of Algeria</dc:title>
    <dc:creator>Bilbo Baggins</dc:creator>
    <dc:subject>Country Flags</dc:subject>
    <dc:date>October 2001</dc:date>
    <colors>red, green, white</colors>
    <features>cresent moon, star</features>
  </rdf:Description>
</rdf:RDF>
```


XML

```
▼<root>
  ▼<Products>
    ▼<Product>
       <Code>2941</Code>
       <StockQty>65</StockQty>
       <Barcode>49020570284087</Barcode>
     </Product>
    ▼<Product>
       <Code>2778</Code>
       <StockQty>200</StockQty>
       <Barcode>72020570064306</Barcode>
     </Product>
    ▼<Product>
       <Code>2838</Code>
       <StockQty>140</StockQty>
       <Barcode>8802057003726</Barcode>
     </Product>
   </Products>
 </root>
```

Semistructured Data / JSON / XML / ...

- Semistructured =>
  - loosely structured (no restrictions on tags & nesting relationships)
  - no schema required
- XML / JSON / ...
  - under the "semistructured" umbrella
  - self-describing
  - the standard for information representation & exchange

7

7

Web Data in COMP9319

- We assume in XML form, since:
  - HTML, RDF, XHTML, ... ∈ XML
  - Other semistructured data such as JSON, Emails, ... can be easily mapped to XML

8

8

XML

XML (*eXtensible Markup Language*) is a standard developed by W3C (*World Wide Web Consortium*) and endorsed by a host of industry heavyweights such as IBM, Microsoft, SAP, Software AG, General Motors, ...

9

9

Storage format vs presentation format - The power of markup

Traditional Database or Spreadsheet

Raymond, Wong, wong, 5932, John, Smith, jsmith, 1234, ...

HTML

```
<br>
<font size=1 color="ff003a">
<ul>
 <li> <b> Raymond Wong </b> </li>
 <li> Login: wong </li>
 <li> Phone: x 5932 </li> <br>
</ul>
</font>
```

XML

```
<Staff>
  <Name>
    <FirstName> Raymond </FirstName>
    <LastName>  Wong </LastName>
  </Name>
  <Login> wong </Login>
  <Ext> 5932 </Ext>
</Staff>
```

10

10

XML Terminology

- tags: book, title, author, ...
- start tag: <book>, end tag: </book>
- elements: <book>...</book>,<author>...</author>
- elements are nested
- empty element: <red></red> abbrv. <red/>
- an XML document: single *root element*
- *well formed* XML document: if it has matching tags

11

11

Resources

- www.w3.org
- www.xml.com
- www.xml.org
- www.oasis-open.org

12

12

More XML: Attributes

```
<book price = "55" currency = "USD">
  <title> Foundations of Databases </title>
  <author> Abiteboul </author>
  ...
  <year> 1995 </year>
</book>
```

13

13

More XML: Oids and References

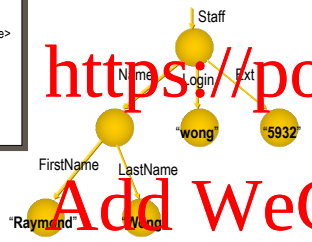
```
<person id="o555">
  <name> Jane </name>
</person>
<person id="o456">
  <name> Mary </name>
  <children idref="o123 o555"/>
</person>
<person id="o123" mother="o456">
  <name>John</name>
</person>
```

14

14

XML/JSON/semistructured data can be modeled in a tree form

```
<Staff>
  <Name>
    <FirstName> Raymond </FirstName>
    <LastName> Wong </LastName>
  </Name>
  <Login> wong </Login>
  <Ext> 5932 </Ext>
</Staff>
```



15

15

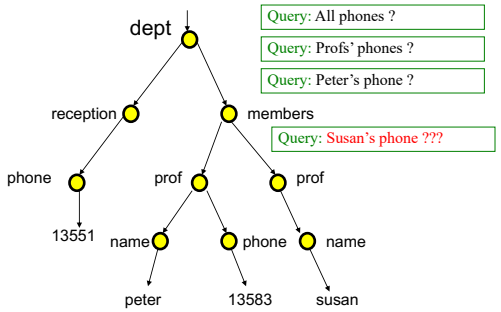
Why need to query tree data

- To extract data from a large tree
- To exchange data (data- or query-shipping)
- To exchange data between different user communities or ontologies or schemas
- To integrate data from multiple data sources

16

16

Answering queries requiring navigation of the data tree



17

17

XPath 1.0

- http://www.w3.org/TR/xpath (11/99)
- Building block for other W3C standards:
  - XSL Transformations (XSLT)
  - XML Link (XLink)
  - XML Pointer (XPointer)
  - XPath 2.0
  - XQuery
- Was originally part of XSL

18

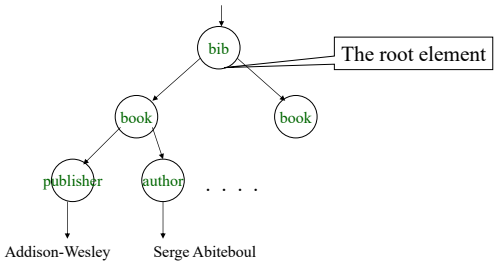
18

Example for XPath Queries

```
<bib>
   <book>  <publisher> Addison-Wesley </publisher>
           <author> Serge Abiteboul </author>
           <author> <first-name> Rick </first-name>
                    <last-name> Hull </last-name>
           </author>
           <author> Victor Vianu </author>
           <title> Foundations of Databases </title>
           <year> 1995 </year>
   </book>
   <book price="55">
           <publisher> Freeman </publisher>
           <author> Jeffrey D. Ullman </author>
           <title> Principles of Database and Knowledge Base Systems </title>
           <year> 1998 </year>
   </book>
</bib>
```

19

Data Model for XPath



20

20

XPath: Simple Expressions

/bib/book/year

Result: <year> 1995 </year>
<year> 1998 </year>

/bib/paper/year

Result: empty

21

21

XPath: Restricted Kleene Closure

//author

Result: <author> Serge Abiteboul </author>
<author> <first-name> Rick </first-name>
<last-name> Hull </last-name>
</author>
<author> Victor Vianu </author>
<author> Jeffrey D. Ullman </author>

//author/first-name

Result: <first-name> Rick </first-name>

22

22

XPath: Text Nodes

/bib/book/author/text()

Result: Serge Abiteboul
Victor Vianu
Jeffrey D. Ullman

Rick Hull doesn't appear because he has firstname, lastname

Functions in XPath:

- text() = matches the text value
- node() = matches any node (= * or @* or text())
- name() = returns the name of the current tag

23

23

XPath: Wildcard

//author/*

Result: <first-name> Rick </first-name>
<last-name> Hull </last-name>

* Matches any element

24

24

XPath: Attribute Nodes

/bib/book/@price

Result: "55"

@price means that price is has to be an attribute

25

25

XPath: Qualifiers

/bib/book/author[firstname]

Result: <author> <first-name> Rick </first-name>
<last-name> Hull </last-name>
</author>

26

26

XPath: More Qualifiers

/bib/book/author[firstname][address[//zip][city]]/lastname

Result: <lastname> … </lastname>
<lastname> … </lastname>

27

27

XPath: More Qualifiers

/bib/book[@price < "60"]

/bib/book[author/@age < "25"]

/bib/book[author/text()]

28

28

XPath: More Details

- We can navigate along 13 axes:
  - ancestor
  - ancestor-or-self
  - attribute
  - child
  - descendant
  - descendant-or-self
  - following
  - following-sibling
  - namespace
  - parent
  - preceding
  - preceding-sibling
  - self

29

29

Differences from traditional DB

- What sets semistructured/XML data servers apart from RDBMS or OODB is the lack of typing.
  - This affects mostly the way the data is stored and indexed.
- Also, Web data are inherently distributed

30

30

Implementing XML Repository

- Repository backend
  - plain text file
  - relational database
  - object database
  - tailor-made, specialized XML database
- Type information
  - even partial typing information can be used to improve the storage

31

31

Text files

- *it's the simplest way to store*
- *easy to handle*
- *widely available*
- have to check out an entire doc in order to retrieve a datum
- simultaneously access/update
- access/modify an item from a large catalog collection

32

32

Relational databases

- existing, proven technology to provide full database management
- it's not easy and efficient to manage XML data in traditional RDBMS

33

33

An Example (using RDBMS)

- assume no typing information
- data can be an arbitrary graph
- let's use two tables for the XML instances:
  - one to store all edge information
  - one to store values

34

34

The two tables

Ref(src, label, dst)

Val(oid, value)

Suppose a simple query like:

family/person/hobby

in XPath

35

35

The same query in SQL

```
select v.value
from Ref r1, Ref r2, Ref r3, Val v
where r1.src = "root" AND r1.label = "family"
AND r1.dst = r2.src AND r2.label = "person"
AND r2.dst = r3.src AND r3.label = "hobby"
AND r3.dst = v.oid
```

This is a 4-way join!!!

It's very inefficient though index on label can help a lot.

36

36

Efficiency problem

- even simple query will have a large no of joins
- RDBMS organizes data based on the structure of tables and type info => clustering, indexing, query optimization are not working properly for XML data
- Also #ways to traverse path expressions are much more than that on tables

37

37