

XML

(Extensible Markup Language)

Databases

Lecture By
Binu Jasim
24-Oct-2016

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<bookstore>
```

```
<book category="cooking">
```

```
<title lang="en">Everyday Sushi</title>
```

```
<author>Motto Kawasaki</author>
```

```
<year>2005</year>
```

```
<price>30.00</price>
```

```
</book>
```

```
<book category="children">
```

```
<title lang="en">Harry Potter</title>
```

```
<author>J K. Rowling</author>
```

```
<year>2005</year>
```

XML

- *W3C Standard for data representation and exchange*
- *No predefined tags (unlike HTML)*
- *XML tags describe the data. Not for formatting as in HTML*

XML Databases

- XML database allows data to be specified, and sometimes stored, in XML format
- Most often data is stored as relational data or in some other formats
- We'll learn about Querying xml databases like `/Student[Name="Alice"]/Email`

Building Blocks of XML

- Elements (Tags) are the primary components of XML documents.

Element Author with Attr id

```
<AUTHOR id = 123>  
  <FNAME> JAMES</FNAME>  
  <LNAME> RUSSEL</LNAME>  
</AUTHOR>  
  
<!-- I am comment -->
```

Element FNAME nested inside element Author.

- Attributes provide additional information about Elements.
Values of the Attributes are set inside the Elements
- Comments start with <!-- and end with -->

```
<bookstore>
```

```
<book category="cooking">  
  <title> Make Sushi </title>  
  <author> Motto </author>  
  <price> $10 </price>  
</book>
```

```
<book>  
  <title> Tale of Bikes </title>  
  <author> Yamaha </author>  
  <price> $20 </price>  
  <year> 2016 </year>  
</book>
```

```
</bookstore>
```

XML vs Relational Model

	Relational	XML
Structure	Tables	Hierarchical/Tree
Schema	Fixed Schema	No Fixed Schema
Queries	SQL	Not well established
Ordering	Not ordered	Implicitly ordered

Our Topic

1. Validating XML - DTD
2. Querying XML using XPath
3. Querying XML using XQuery

Valid XML

- Well formed XML
 - Proper nesting of tags
 - Single root element
 - Unique attribute within an element
- Additionally XML should be valid with respect to a description document - DTD or XSD

Validating XML

- DTD (Document Type Definition)
- XSD (XML Schema Definition)
- Way to specify structure/schema to XML
- Example: Every <Book> tag should have an ISBN attribute
- Why Validate? - Less effort on end users of XML

DTD

```
<?xml version="1.0" encoding="UTF-8" ?>
```

```
<!DOCTYPE bookstore [  
<!ELEMENT bookstore (book) >  
<!ELEMENT book (author)*>  
<!ATTLIST book category CDATA #REQUIRED>  

```

```
<bookstore>  
<book category="cooking"></book>  
</bookstore>
```

DTD

```
xmllint --valid --noout bookstore.xml
```

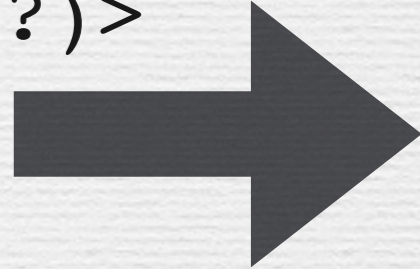
What if we add multiple `<book></book>` elements?

```
validity error : Element bookstore content does not follow the DTD,  
expecting (book), got (book book )
```

DTD

- The `<bookstore>` should have 1 or more books as sub elements
- `<book>` tag should have title, author and price as sub elements in that order (order is implied by xml)
- The element `year` is optional
- The attribute `category` is optional

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE bookstore [
<!ELEMENT bookstore (book)+ >
<!ELEMENT book (title, author, price, year?)>
<!ATTLIST book category CDATA #IMPLIED>
]>
<bookstore>
<book category="cooking">
  <title> Make Sushi </title>
  <author> Motto </author>
  <price> $10 </price>
</book>
<book>
  <title> Tale of Africa </title>
  <author> Patrick </author>
  <price> $20 </price>
  <year> 2016 </year>
</book>
</bookstore>
```



Complete DTD

```
<!DOCTYPE bookstore [  
<!ELEMENT bookstore (book)+ >  
<!ELEMENT book (title, author, price, year?)>  
<!ATTLIST book category CDATA #IMPLIED>  
  
<!ELEMENT title (#PCDATA)>  
<!ELEMENT author (#PCDATA)>  
<!ELEMENT year (#PCDATA)>  
<!ELEMENT price (#PCDATA)>  
  
>
```

ID & IDREF

```
<bookstore>
<book writer="Mot">
    .....
</book>
<book writer="Yam">
    .....
</book>

<authors>
  <author foo="Mot">Motto</author>
  <author foo="Yam">Patrick</author>
</authors>
</bookstore>
```



```
<!DOCTYPE bookstore [  
<!ELEMENT bookstore (book+, authors) >  
<!ELEMENT book (title, price, year?)>  
<!ATTLIST book writer IDREFS #REQUIRED>  
<!ELEMENT authors (author)+>  
<!ELEMENT author (#PCDATA)>  
<!ATTLIST author foo ID #REQUIRED>  
  
<!ELEMENT title (#PCDATA)>  
<!ELEMENT year (#PCDATA)>  
<!ELEMENT price (#PCDATA)>  
  
>
```

Limitations of DTD

- No useful type checking. e.g. year should be number can't be enforced with DTD
- Pointers (IDREFS) are untyped. e.g. `<book id="some_author_id">` will be valid under DTD
- Difficult to have sub elements in any order (which might be a good thing as well!)

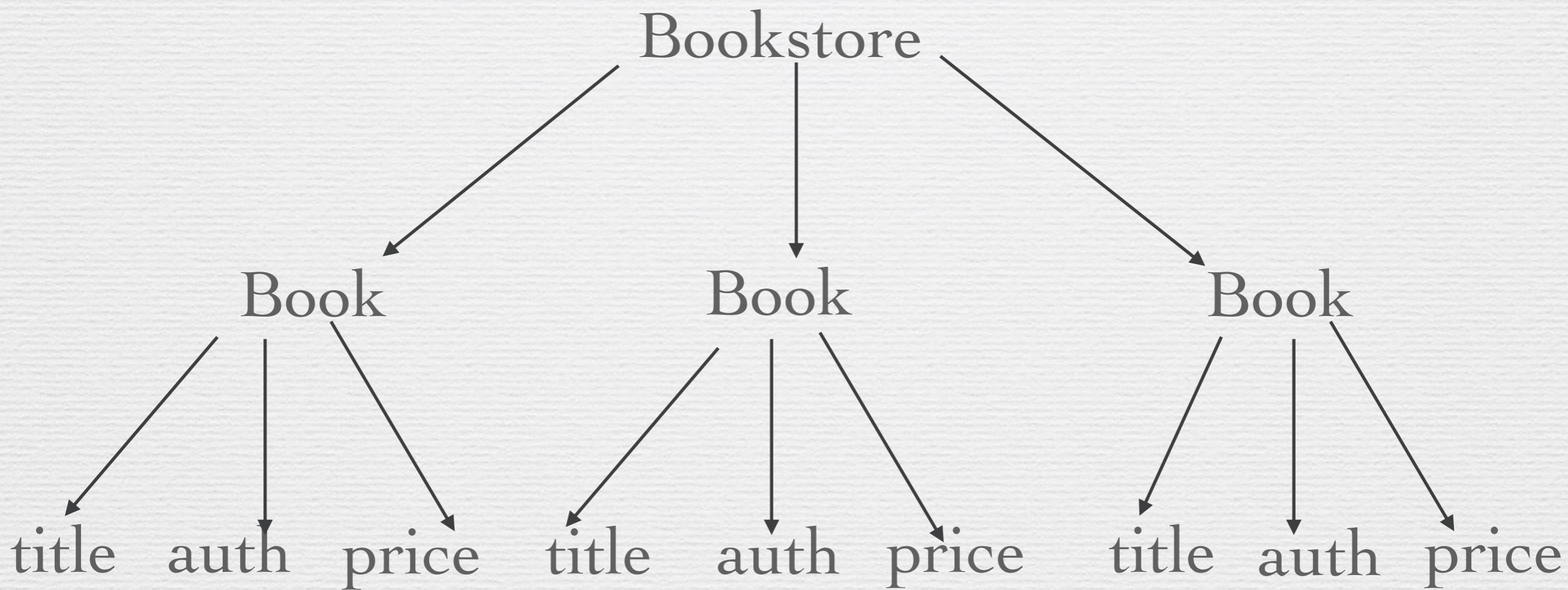
XSD (Xml Schema Definition)

```
<xs:element name="item" maxOccurs="unbounded">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="title" type="xs:string" />
      <xs:element name="note" type="xs:string"
        minOccurs="0" />
      <xs:element name="quantity" type="xs:
        positiveInteger" />
      <xs:element name="price" type="xs:decimal" />
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

More powerful & complex
We won't cover it any further...

Querying XML - XPath

- XPath - Query language for selecting nodes from an XML document.
- W3C standard
- Major component in XQuery & XSLT
- Designed to mimic URI (Uniform Resource Identifier)



XPath

- / root element or path separator
- /Bookstore/Book - returns all book nodes as xml
- @category - returns the attributes
- // any descendant including that node
- conditions [price < 10]

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<bookstore>
```

```
<book category="cooking">
```

```
<title> Make Sushi </title>
```

```
<author> Motto </author>
```

```
<price> $10 </price>
```

```
</book>
```

```
<book>
```

```
<title> Tale of Africa </title>
```

```
<author> Patrick </author>
```

```
<price> $20 </price>
```

```
<year> 2016 </year>
```

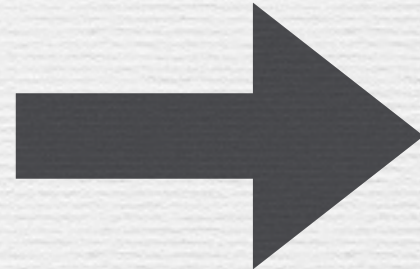
```
</book>
```

```
<magazine>
```

```
<title>National Geographic </title>
```

```
</magazine>
```

```
</bookstore>
```



Examples

- `/bookstore/book/title` - titles of all books
- `/bookstore/book/@category`
- `/bookstore/(book|magazine)/title`
- `/bookstore/*/title`
- `//title`

Examples

- use `data(@category)` to extract the value
- `/bookstore/book[@price > 10]`
Note that there is no / before []
- `/bookstore/book[@price > 10]/title`

More Examples

- `/bookstore/book[1]` - first book node
- `/bookstore/book[year]` - all books having year sub element
- `/bookstore/book[not(year)]` - all books with no year

Find all books where Ullman is an author and Widom is not an author

```
/bookstore/book  
  [ authors/author = "Ullman" and  
    authors/author != "Widom" ]
```

What is wrong with the above query?

XPath Built in functions

- xpath supports several built in functions
- `\bookstore\book[contains(remark, "great")]\title`
- `count()`, `contains()`, `name()` etc.

Find all books where Ullman is an author and Widom is
not an author

```
/bookstore/book  
  [authors/author = "Ullman" and  
   count(authors[author = "Widom"])=0]/title
```

Navigation axes

- 13 navigation axes
- parent::, following-sibling:: etc.

XQuery

- More powerful and it contains Xpath as a component
- `<Element> { ... query ... } </Element>`
- Transformations of XML is possible

FLOWR expressions

```
for $x in doc("books.xml")/bookstore/book
where $x/price>30
order by $x/title
return $x/title
```

For, Let, Order by, Where, Return

Only Return is compulsory

Q. Return the names of books that contains author's name

```
for $b in doc("books.xml")/bookstore/book
where some $a in $b/author satisfies
    contains($b/title, $a)
```

```
return
```

```
<book>
```

```
  {$b/title}
```

```
  {$b/author}
```

```
</book>
```

Existential Quantifiers

Average of Book Prices

```
let $a := avg(doc("books.xml")
              /bookstore/book/price)
return <avg>{$a}</avg>
```

Joins using Nested For

- Exercise: How to find all books written by the same author? - In XQuery and XPath?

Reference

- Introduction to Databases | Stanford Lagunita