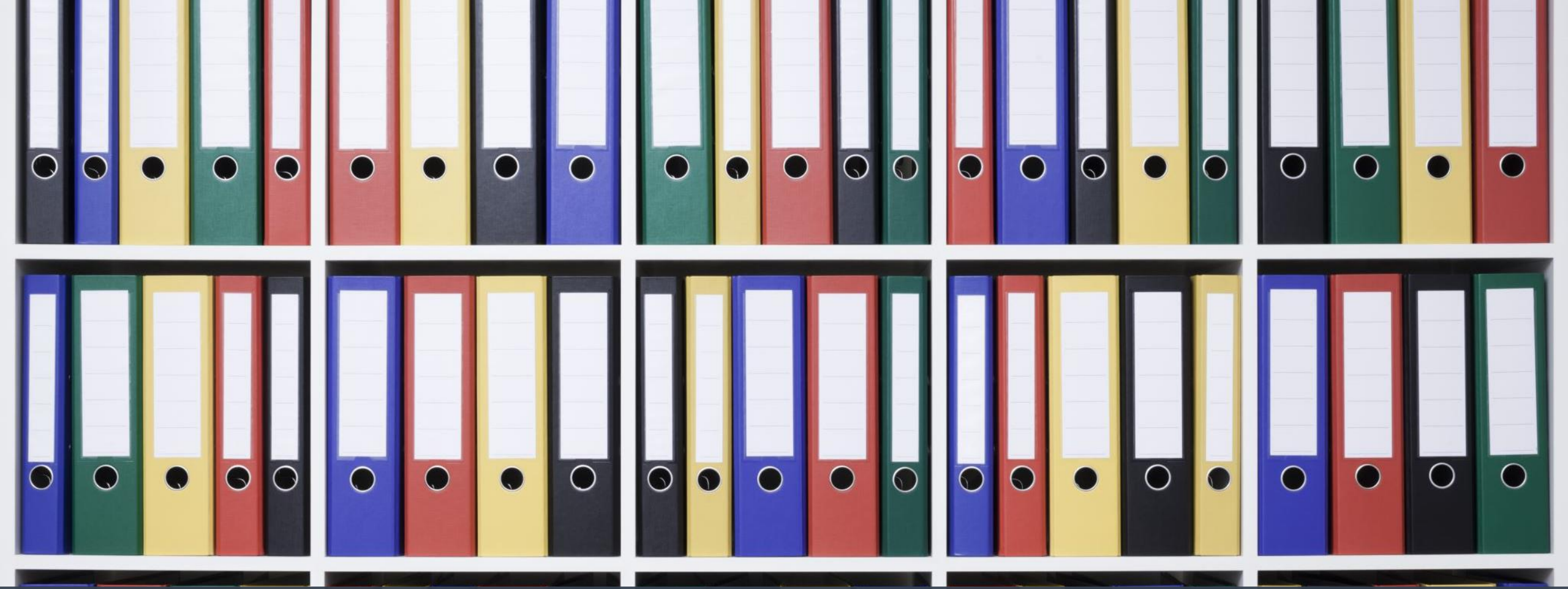


DOCUMENT DATABASES

XML

DOCUMENT DATABASE

Definition

- A place to store documents.
- Documents are self-describing structures and usually similar to each other, but they don't have to be the same.

Pros/Cons

- Documents can vary from each other and still belong to the same collection

Example

- MongoDB, Couchbase, BaseX

DOCUMENT TYPE: XML

XML

XML stands for eXtensible Markup Language.

XML was designed to store and transport data.

XML was designed to be both human- and machine-readable.

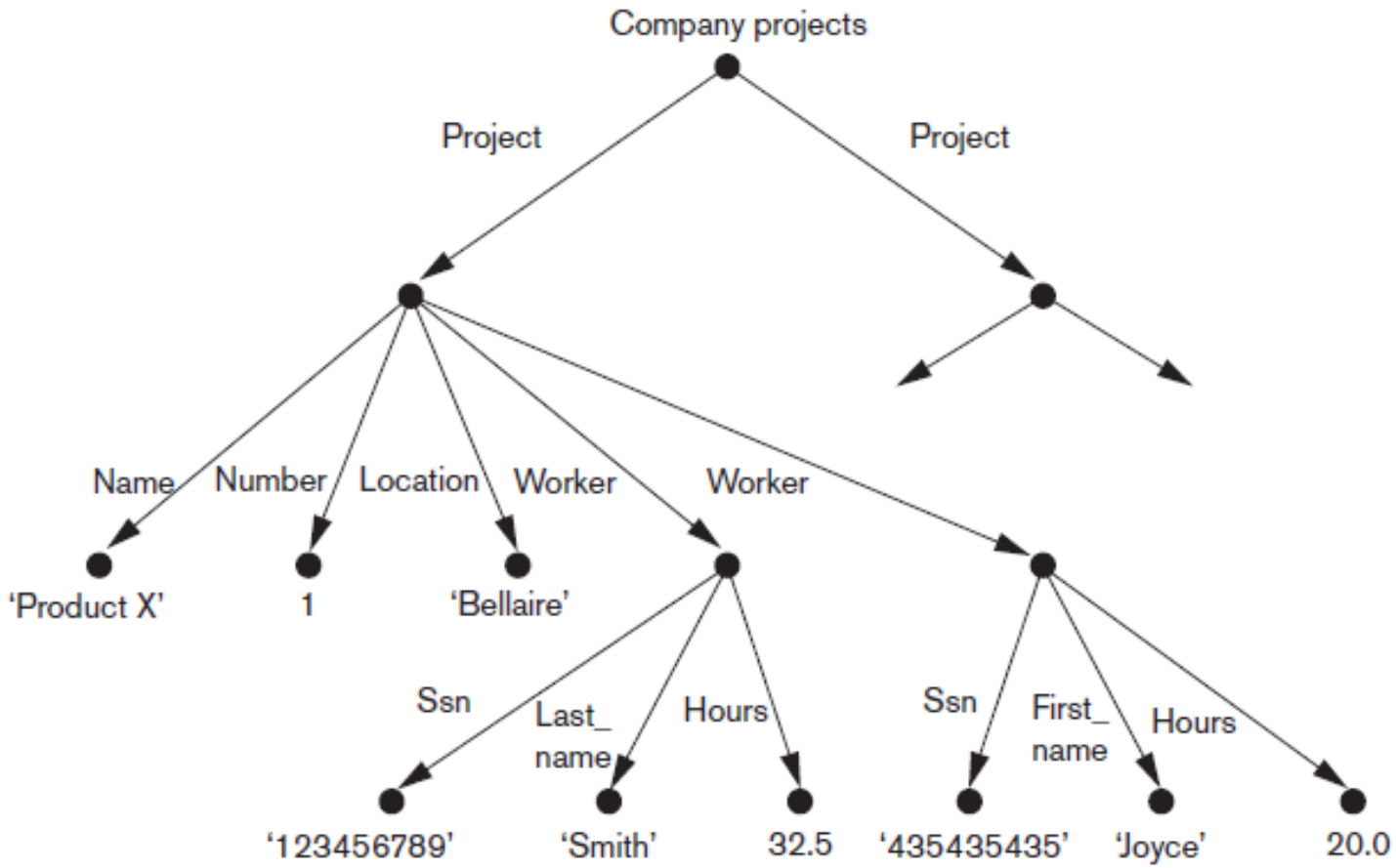
Well suited for structured or semi-structured data due to hierarchical organization.

XML (by itself) does not do anything.

Similar syntax to HTML, but:

- HTML uses many predefined tags, XML is extensible, users can define their own tags
- HTML documents do not include schema information about type of data, XML can include schema information
- HTML only meant to describe how data should be displayed, XML meant to describe how data should be interpreted

SEMISTRUCTURED DATA



XML EXAMPLE

```
<?xml version="1.0" encoding="UTF-8"?>
<note>
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>
```

XML ADVANTAGES

It simplifies data sharing

- Many computer systems contain data in incompatible formats.
- Exchanging data between incompatible systems is a time-consuming task.
- Large amounts of data must be converted, and incompatible data is often lost.

It simplifies data transport

- Plain text format – software/hardware independent

It simplifies platform changes

- Easy to expand or upgrade to new systems

It simplifies data availability

- Available to all kinds of "reading machines“

Documents that have many small data items that follow a specific structure

- May be extracted from a structured database.
- Formatted as XML documents in order to exchange over the Web.

XML HIERARCHICAL TREE MODEL

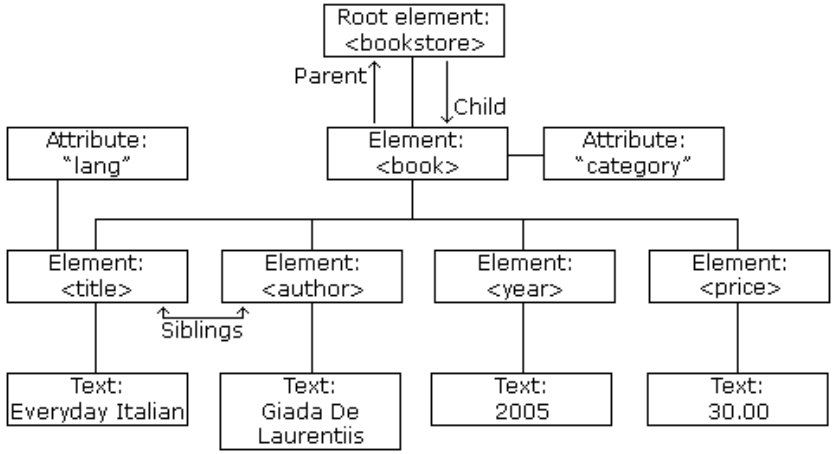
XML documents form a tree structure that starts at "the root" and branches to "the leaves"

XML documents are formed as **element trees.**

An XML tree starts at a **root element** and branches from the root to **child elements.**

All elements can have sub elements (child elements):

```
<root>
  <child>
    <subchild>.....</subchild>
  </child>
</root>
```



XML HIERARCHICAL STRUCTURE

The terms parent, child, and sibling are used to describe the relationships between elements.

Parent have children. Children have parents. Siblings are children on the same level (brothers and sisters).

All elements can have text content (Harry Potter) and attributes (category="cooking").

```
<?xml version="1.0" encoding="UTF-8"?>
<bookstore>
  <book category="cooking">
    <title lang="en">Everyday Italian</title>
    <author>Giada De Laurentiis</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>

    <price>29.99</price>
  </book>
  <book category="web">

    <title lang="en">Learning XML</title>
    <author>Erik T. Ray</author>
    <year>2003</year>
    <price>39.95</price>

  </book>
</bookstore>
```

XML SYNTAX RULES

Must have a root element

Can have a prolog:
- <?xml version="1.0" encoding="UTF-8"?>
- First thing in document

All tags should be closed
- <tag>…</tag>
- <tag />

Case sensitive

Respect nesting levels
- <a>…

Attribute values always quoted
- <tag attr="some value">

Entitiy References
- 5 predefined references

&lt;	<	less than
&gt;	>	greater than
&amp;	&	ampersand
&apos;	'	apostrophe
&quot;	"	quotation mark

- More can be added

Comments
- <!– This is a comment -->

WELL FORMED & VALID

- **Well formed**
  - Has **XML declaration**
    - Indicates version of XML being used as well as any other relevant attributes
  - Must follow the syntactic guidelines of the tree data model.
    - There should be a *single root element*
  - Every element must include matching pair of start and end tags
    - Within start and end tags of parent element
- **Valid**
  - Document must be well formed
  - Document must follow a particular schema
  - Start and end tag pairs must follow structure specified in separate XML **DTD (Document Type Definition)** file or XML schema file
- An XML document with correct syntax is called "Well Formed".
- An XML document validated against a DTD or XML Schema is both "Well Formed" and "Valid".

DTD

DTD

Allows to define the structure and legal elements and attributes of an XML document.

- Independent groups of people can agree on a standard DTD for interchanging data.

An application can use a DTD to verify that XML data is valid

If the DTD is declared in an external file, the <!DOCTYPE> definition must contain a reference to the DTD file.

```
<?xml version = "1.0" standalone = "no"?>
<!DOCTYPE Projects SYSTEM "proj.dtd">
```

XML AND DTD

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE note SYSTEM "Note.dtd">
<note>
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>
```

```
<!DOCTYPE note
[
<!ELEMENT note (to,from,heading,body)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT heading (#PCDATA)>
<!ELEMENT body (#PCDATA)>
]>
```

DTD

!DOCTYPE note defines that the root element of the document is note

!ELEMENT note defines that the note element must contain the elements: "to, from, heading, body"

!ELEMENT to defines the to element to be of type "#PCDATA"

!ELEMENT from defines the from element to be of type "#PCDATA"

!ELEMENT heading defines the heading element to be of type "#PCDATA"

!ELEMENT body defines the body element to be of type "#PCDATA"

XML BUILDING BLOCKS

Seen from a DTD point of view, all XML documents are made up by the following building blocks:

- Elements
- Attributes
- Entities
- PCDATA
- CDATA

ELEMENTS

Main building block

Can contain text, other elements or be empty

Complex elements

- Constructed from other elements hierarchically

Simple elements

- Contain data values

<element>Something something</element>

ATTRIBUTES

Attributes provide **extra information about elements**

Attributes are always placed inside the opening tag of an element

Attributes always come in name/value pairs

<element attribute="some value" />

PCDATA

PCDATA means parsed character data.

Think of character data as the text found between the start tag and the end tag of an XML element.

PCDATA is text that WILL be parsed by a parser. The text will be examined by the parser for entities and markup.

Typically used for text between tags.

CDATA

CDATA means character data.

CDATA is text that will NOT be parsed by a parser. Tags inside the text will NOT be treated as markup and entities will not be expanded.

Typically used for attribute values.

DECLARING ELEMENTS

<!ELEMENT element-name category>
or
<!ELEMENT element-name (element-content)>

Empty elements

- <!ELEMENT element-name EMPTY>

Elements with PCDATA

- <!ELEMENT element-name (#PCDATA)>

DECLARING ELEMENTS

Elements with any contents

- <!ELEMENT element-name ANY>

Elements with children (sequences)

- <!ELEMENT element-name (child1)>
  or
  <!ELEMENT element-name (child1,child2,...)>
- When children are declared in a sequence separated by commas, the children must appear in the same sequence in the document.
- In a full declaration, the children must also be declared, and the children can also have children.

XML DTD NOTATION

A * following the element name means that the element can be repeated zero or more times in the document.

- This kind of element is known as an *optional multivalued (repeating) element*.
- <!ELEMENT element-name (child-name*)>

A + following the element name means that the element can be repeated one or more times in the document.

- This kind of element is a *required multivalued (repeating) element*.
- <!ELEMENT element-name (child-name+)>

XML DTD NOTATION

A ? following the element name means that the element can be repeated zero or one times.

- This kind is an *optional single-valued (nonrepeating) element*.
- <!ELEMENT element-name (child-name?)>

An element appearing without any of the preceding three symbols must appear exactly once in the document.

- This kind is a *required single-valued (nonrepeating) element*.
- <!ELEMENT element-name (child-name)>

XML DTD NOTATION

- Parentheses can be nested when specifying elements
- A bar symbol (e_1 | e_2) specifies that either e_1 or e_2 can appear in the document.
  - <!ELEMENT note (to,from,header,(message|body))>
  - The example above declares that the "note" element must contain a "to" element, a "from" element, a "header" element, and either a "message" or a "body" element
- Mixed content can also be declared
  - <!ELEMENT note (#PCDATA|to|from|header|message)*>

DECLARING ATTRIBUTES

<!ATTLIST element-name attribute-name attribute-type attribute-value>

- DTD example:
  <!ATTLIST payment type CDATA "check">
- XML example:
  <payment type="check" />

Value	Explanation
value	The default value of the attribute
#REQUIRED	The attribute is required
#IMPLIED	The attribute is optional
#FIXED *value*	The attribute value is fixed

Type	Description
CDATA	The value is character data
(*en1*|*en2*|..)	The value must be one from an enumerated list
ID	The value is a unique id
IDREF	The value is the id of another element
IDREFS	The value is a list of other ids
NMTOKEN	The value is a valid XML name
NMTOKENS	The value is a list of valid XML names
ENTITY	The value is an entity
ENTITIES	The value is a list of entities
NOTATION	The value is a name of a notation
xml:	The value is a predefined xml value

ID AND IDREF

ID represents a unique ID name for the attribute that identifies the element within the context of the document.

IDs are like internal links in plain HTML.

ID is used primarily by programs or scripting languages that process the document.

The value for ID must be a valid XML name beginning with a letter and containing alphanumeric characters or the underscore character without any whitespace.

ID values should be unique within a document.

```
<?xml version = "1.0" encoding="UTF-8" standalone = "yes"?>
<!DOCTYPE CONTACTS [
        <!ELEMENT CONTACTS ANY>
        <!ELEMENT CONTACT (NAME, EMAIL)>
        <!ELEMENT NAME (#PCDATA)>
        <!ELEMENT EMAIL (#PCDATA)>
        <!ATTLIST CONTACT CONTACT_NUM ID #REQUIRED>
]>
<CONTACTS>
        <CONTACT CONTACT_NUM = "1">
                <NAME>Lok Siu</NAME>
                <EMAIL>siu@lok.com</EMAIL>
        </CONTACT>
        <CONTACT CONTACT_NUM = "2">
                <NAME>Joseph Misuraca</NAME>
                <EMAIL>joe@misuraca.com</EMAIL>
        </CONTACT>
</CONTACTS>
```

ID AND IDREF

The IDREF type allows the value of one attribute to be an element elsewhere in the document provided that the value is the ID of the referenced element.

The IDREFS type is like IDREF but allows a list of space-separated IDs.

```
<!DOCTYPE CONTACTS [
        <!ELEMENT CONTACTS ANY>
        <!ELEMENT CONTACT (NAME, EMAIL)>
        <!ELEMENT NAME (#PCDATA)>
        <!ELEMENT EMAIL (#PCDATA)>
        <!ATTLIST CONTACT CONTACT_NUM ID #REQUIRED>
        <!ATTLIST CONTACT MOTHER IDREF #IMPLIED>
]>
<CONTACTS>
        <CONTACT CONTACT_NUM = "2”>
                <NAME>Teri Mancuso</NAME>
                <EMAIL>teri@teri.com</EMAIL>
        </CONTACT>
        <CONTACT CONTACT_NUM = "1" MOTHER = "2">
                <NAME>Kristin Mancuso</NAME>
                <EMAIL>kristin@kristin.com</EMAIL>
        </CONTACT>
</CONTACTS>
```

DTD EXAMPLES

```
<!DOCTYPE TVSCHEDULE [

<!ELEMENT TVSCHEDULE (CHANNEL+)>
<!ELEMENT CHANNEL (BANNER,DAY+)>
<!ELEMENT BANNER (#PCDATA)>
<!ELEMENT DAY (DATE,(HOLIDAY|PROGRAMSLOT+)+)>
<!ELEMENT HOLIDAY (#PCDATA)>
<!ELEMENT DATE (#PCDATA)>
<!ELEMENT PROGRAMSLOT (TIME,TITLE,DESCRIPTION?)>
<!ELEMENT TIME (#PCDATA)>
<!ELEMENT TITLE (#PCDATA)>
<!ELEMENT DESCRIPTION (#PCDATA)>

<!ATTLIST TVSCHEDULE NAME CDATA #REQUIRED>
<!ATTLIST CHANNEL CHAN CDATA #REQUIRED>
<!ATTLIST PROGRAMSLOT VTR CDATA #IMPLIED>
<!ATTLIST TITLE RATING CDATA #IMPLIED>
<!ATTLIST TITLE LANGUAGE CDATA #IMPLIED>
]>
```

DTD EXAMPLES

```
<!DOCTYPE NEWSPAPER [

<!ELEMENT NEWSPAPER (ARTICLE+)>
<!ELEMENT ARTICLE (HEADLINE,BYLINE,LEAD,BODY,NOTES)>
<!ELEMENT HEADLINE (#PCDATA)>
<!ELEMENT BYLINE (#PCDATA)>
<!ELEMENT LEAD (#PCDATA)>
<!ELEMENT BODY (#PCDATA)>
<!ELEMENT NOTES (#PCDATA)>

<!ATTLIST ARTICLE AUTHOR CDATA #REQUIRED>
<!ATTLIST ARTICLE EDITOR CDATA #IMPLIED>
<!ATTLIST ARTICLE DATE CDATA #IMPLIED>
<!ATTLIST ARTICLE EDITION CDATA #IMPLIED>

<!ENTITY NEWSPAPER "Vervet Logic Times">
<!ENTITY PUBLISHER "Vervet Logic Press">
<!ENTITY COPYRIGHT "Copyright 1998 Vervet Logic Press">

]>
```

XML DTD

XML DTD

- Data types in DTD are not very general
- Special syntax
  - Requires specialized processors
- All DTD elements always forced to follow the specified ordering of the document
  - Unordered elements not permitted

- DTD alternatives:
  - XSD: XML Schema Definition
  - Rexlax NG
  - Schematron

Xpath & XQuery

QUERYING XML DOCUMENTS

XML LANGUAGES

Two query language standards

- **XPath**
  - Specify path expressions to identify certain nodes (elements) or attributes within an XML document that match specific patterns
- **XQuery**
  - Uses XPath expressions but has additional constructs

XPATH

- XPath is a syntax for defining parts of an XML document
- XPath uses path expressions to navigate in XML documents
- XPath contains a library of standard functions
- XPath is a major element in XSLT and in XQuery
- XPath is a W3C recommendation

TERMINOLOGY

Nodes

- There are seven kinds of nodes: element, attribute, text, namespace, processing-instruction, comment, and document nodes.
- XML documents are treated as trees of nodes. The topmost element of the tree is called the root element

Atomic Values

- Nodes with no children or parent

Items

- Items are atomic values or nodes

EXAMPLE

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

<bookstore>
 <book>
  <title lang="en">Harry Potter</title>
  <author>J K. Rowling</author>
  <year>2005</year>
  <price>29.99</price>
 </book>
</bookstore>
```

<bookstore> (root element node)

<author>J K. Rowling</author> (element node)

lang="en" (attribute node)

J K. Rowling (atomic)

"en" (atomic)

XPATH: SPECIFYING PATH EXPRESSIONS IN XML

XPath expression

- Returns a sequence of items that satisfy a pattern as specified by the expression
  - Either values (from leaf nodes) or elements or attributes
- Similar to path expressions in traditional computer file systems

Qualifier conditions

- Further restrict nodes that satisfy pattern

Separators used when specifying a path:

- Single slash (/) and double slash (//)

RELATIONSHIP OF NODES

Parent

- Each element and attribute has one parent.

Children

- Element nodes may have zero, one or more children.

Siblings

- Nodes that have the same parent.

Ancestors

- A node's parent, parent's parent, etc.

Descendants

- A node's children, children's children, etc.

SYNTAX

XPath uses path expressions to select nodes or node-sets in an XML document by following a path or steps.

Expression	Description
nodename	Selects all nodes with the name "*nodename*"
/	Selects from the root node
//	Selects nodes in the document from the current node that match the selection no matter where they are
.	Selects the current node
..	Selects the parent of the current node
@	Selects attributes

BASIC SYNTAX EXAMPLE

```
<bookstore>

<book>
  <title lang="en">Harry Potter</title>
  <price>29.99</price>
</book>

<book>
  <title lang="en">Learning XML</title>
  <price>39.95</price>
</book>

</bookstore>
```

Path Expression	Result
bookstore	Selects all nodes with the name "bookstore"
/bookstore	Selects the root element bookstore **Note:** If the path starts with a slash (/) it always represents an absolute path to an element!
bookstore/book	Selects all book elements that are children of bookstore
//book	Selects all book elements no matter where they are in the document
bookstore//book	Selects all book elements that are descendant of the bookstore element, no matter where they are under the bookstore element
//@lang	Selects all attributes that are named lang

PREDICATES

Predicates are used to find a specific node or a node that contains a specific value.

Predicates are always embedded in square brackets.

Attribute name prefixed by the @ symbol

PREDICATE EXAMPLES

```
<bookstore>

<book>
  <title lang="en">Harry Potter</title>
  <price>29.99</price>
</book>

<book>
  <title lang="en">Learning XML</title>
  <price>39.95</price>
</book>

</bookstore>
```

Path Expression	Result
/bookstore/book[1]	Selects the first book element that is the child of the bookstore element. **Note:** In IE 5,6,7,8,9 first node is[0], but according to W3C, it is [1]. To solve this problem in IE, set the SelectionLanguage to XPath: *In JavaScript: xml*.setProperty("SelectionLanguage","XPath");
/bookstore/book[last()]	Selects the last book element that is the child of the bookstore element
/bookstore/book[last()-1]	Selects the last but one book element that is the child of the bookstore element
/bookstore/book[position()<3]	Selects the first two book elements that are children of the bookstore element
//title[@lang]	Selects all the title elements that have an attribute named lang
//title[@lang='en']	Selects all the title elements that have a "lang" attribute with a value of "en"
/bookstore/book[price>35.00]	Selects all the book elements of the bookstore element that have a price element with a value greater than 35.00
/bookstore/book[price>35.00]/title	Selects all the title elements of the book elements of the bookstore element that have a price element with a value greater than 35.00

WILDCARDS

Wildcard symbol *

- Stands for any element
- Example: `/book/*`

Wildcard	Description
*	Matches any element node
@*	Matches any attribute node
node()	Matches any node of any kind

Path Expression	Result
/bookstore/*	Selects all the child element nodes of the bookstore element
//*	Selects all elements in the document
//title[@*]	Selects all title elements which have at least one attribute of any kind

SELECTING SEVERAL PATHS

By using the | operator in an XPath expression you can select several paths.

Path Expression	Result
//book/title | //book/price	Selects all the title AND price elements of all book elements
//title | //price	Selects all the title AND price elements in the document
/bookstore/book/title | //price	Selects all the title elements of the book element of the bookstore element AND all the price elements in the document

OPERATORS

Operator	Description	Example
|	Computes two node-sets	//book | //cd
+	Addition	6 + 4
-	Subtraction	6 - 4
*	Multiplication	6 * 4
div	Division	8 div 4
=	Equal	price=9.80
!=	Not equal	price!=9.80
<	Less than	price<9.80
<=	Less than or equal to	price<=9.80
>	Greater than	price>9.80
>=	Greater than or equal to	price>=9.80
or	or	price=9.80 or price=9.70
and	and	price>9.00 and price<9.90
mod	Modulus (division remainder)	5 mod 2

LIMITS OF XPATH

- Path that specifies the pattern also specifies the items to be retrieved
- Difficult to specify certain conditions on the pattern while separately specifying which result items should be retrieved

XQUERY

XQuery is to XML what SQL is to databases.

XQuery is designed to query XML data.

XQuery is built on XPath expressions

XQuery is a W3C Recommendation

XQuery 1.0 and XPath 2.0 share the same data model and support the same functions and operators.

XQUERY: SPECIFYING QUERIES IN XML

XQuery FLWR expression

- Four main clauses of XQuery
- **For** - selects a sequence of nodes
- **Let** - binds a sequence to a variable
- **Where** - filters the nodes
- **Order by** - sorts the nodes
- **Return** - what to return (gets evaluated once for every node)

XQUERY: SPECIFYING QUERIES IN XML

- Form:

```
FOR <variable bindings to individual nodes (elements)>
LET <variable bindings to collections of nodes (elements)>
WHERE <qualifier conditions>
RETURN <query result specification>
```

- Zero or more instances of `FOR` and `LET` clauses

XQUERY EXAMPLE

```
/bookstore/book[price>30]/title
```

```
for $x in /bookstore/book
where $x/price>30
return $x/title
```

```
<title lang="en">XQuery Kick Start</title>
<title lang="en">Learning XML</title>
```

XQUERY

```
for $x in /bookstore/book
where $x/price>30
order by $x/title
return $x/title
```

The **for** clause selects all book elements under the bookstore element into a variable called $x.

The **where** clause selects only book elements with a price element with a value greater than 30.

The **order by** clause defines the sort-order. Will be sort by the title element.

The **return** clause specifies what should be returned. Here it returns the title elements.

SYNTAX

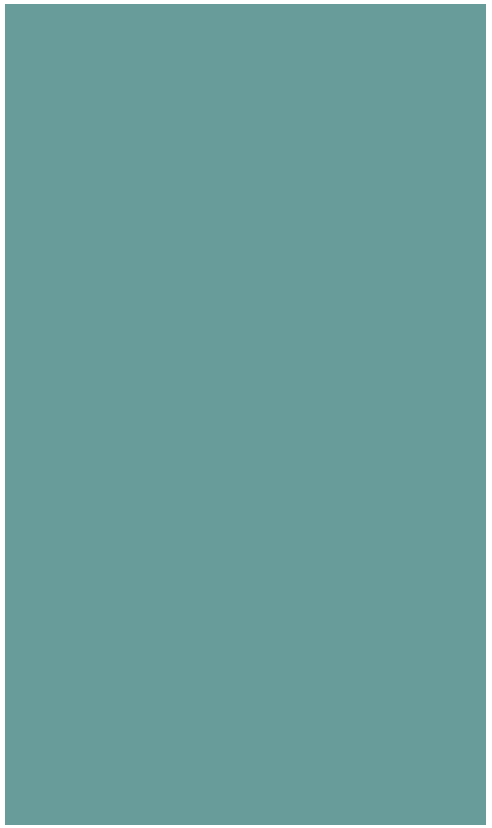
XQuery is case-sensitive

XQuery elements, attributes, and variables must be valid XML names

XQuery string value can be in single or double quotes

XQuery variable is defined with a $ followed by a name, e.g., $bookstore

XQuery comments are delimited by (: and :), e.g. (: XQuery Comment :)



SYNTAX

"If-Then-Else" expressions are allowed in XQuery.

```
for $x in /bookstore/book
return if ($x/@category=“children")
then <child>{data($x/title)}</child>
else <adult>{data($x/title)}</adult>
```

SYNTAX - FOR

The for clause binds a variable to each item returned by the in expression.

The for clause results in iteration.

There can be multiple for clauses in the same FLWOR expression.

To loop a specific number of times in a for clause, use the **to** keyword

```
for $x in (1 to 5)
return <test>{$x}</test>
```

```
<test>1</test>
<test>2</test>
<test>3</test>
<test>4</test>
<test>5</test>
```

SYNTAX - FOR

The **at** keyword can be used to count the iteration

- `for $x at $i in /bookstore/book/title`
  `return <book>{$i}. {data($x)}</book>`
  - <book>1. Everyday Italian</book>
    <book>2. Harry Potter</book>
    <book>3. XQuery Kick Start</book>
    <book>4. Learning XML</book>

More than one in expression is allowed in a for clause. Use commas to separate.

- `for $x in (10,20), $y in (100,200)`
  `return <test>x={$x} and y={$y}</test>`
  - <test>x=10 and y=100</test>
    <test>x=10 and y=200</test>
    <test>x=20 and y=100</test>
    <test>x=20 and y=200</test>

SYNTAX - LET

The let clause allows variable assignments and it avoids repeating the same expression many times.

The let clause does not result in iteration

```
let $x := (1 to 5)
return <test>{$x}</test>
```

```
<test>1 2 3 4 5</test>
```

SYNTAX – ORDER BY

The order by clause is used to specify the sort order of the result

```
for $x in /bookstore/book
order by $x/@category, $x/title
return $x/title
```

```
<title lang="en">Harry Potter</title>
<title lang="en">Everyday Italian</title>
<title lang="en">Learning XML</title>
<title lang="en">XQuery Kick Start</title>
```

SYNTAX - RETURN

The return clause specifies what is to be returned.

```
for $x in /bookstore/book
return $x/title
```

```
<title lang="en">Everyday Italian</title>
<title lang="en">Harry Potter</title>
<title lang="en">XQuery Kick Start</title>
<title lang="en">Learning XML</title>
```

XQUERY AND XPATH FUNCTIONS

XPath 2.0 and XQuery 1.0, share the same functions library

Over 200 available functions

Full list: https://www.w3schools.com/xml/xsl_functions.asp

Example:

id(expression): Finds nodes matching the given ids and returns a node-set containing the identified nodes.

XQUERY UPDATE

Provides CUD operations on XML

- Insert
- Delete
- Replace

Reference: https://www.w3.org/TR/xquery-update-10/