Semi Structured Data

3rd year computer science (license)

2025/2026

Chapter 4
XML Ecosystem

Chapter plan

4.1. XPath: Path Expressions

What is XPath?

- XPath (XML Path Language) is a query language for selecting nodes from an XML document.
- It treats XML as a tree of nodes and provides syntax for navigating this tree.

4.1. XPath: Path Expressions
4.1.1. Principles

Consider this XML document:

```
<?xml version="1.0"?>
<bookstore>
  <book category="fiction">
    <title lang="en">The Great Novel</title>
    <author>Jane Austen</author>
    <year>1813</year>
    <price>29.99</price>
  </book>
  <book category="computer">
    <title lang="en">XML Guide</title>
    <author>John Smith</author>
    <year>2023</year>
    <price>49.99</price>
  </book>
  <magazine category="tech">
    <title>Tech Monthly</title>
    <price>5.99</price>
  </magazine>
</bookstore>
```

4.1. XPath: Path Expressions
4.1.1. Principles

Node Types in XPath

XPath recognizes seven node types:

- **Root Node** (document root, /)
- **Element Nodes** (e.g., <book>)
- **Attribute Nodes** (e.g., category="fiction")
- **Text Nodes** (e.g., "The Great Novel")
- **Comment Nodes** (<!-- comment -->)
- **Processing Instruction Nodes** (<?xml-stylesheet ...?>)
- **Namespace Nodes**

4.1. XPath: Path Expressions
4.1.1. Principles

Pa

Expression	Description	Example	Result
nodename	Selects all child nodes with this name	book	All <book> children
/	Selects from the root node	/bookstore	Root <bookstore> element
//	Selects nodes anywhere in document	//title	All <title> elements
.	Selects the current node	.	Current context node
..	Selects the parent of current node	..	Parent element
@	Selects attributes	@category	All category attributes
*	Wildcard - selects all elements	/*/book	All <book> grandchildren of root

4.1. XPath: Path Expressions
4.1.1. Principles

Basic XPath Examples

- / # Selects the root node
- /bookstore # Selects the bookstore element
- /bookstore/book # Selects all book elements under bookstore
- //book # Selects all book elements anywhere
- book/title # Selects title elements that are children of book
- //book/title # Selects all title elements under any book
- /bookstore/*/price # Selects all price elements two levels deep
- //@category # Selects all category attributes
- //book[@category] # Selects all books that have a category attribute

4.1. XPath: Path Expr
4.1.2. Axes

Axes define the direction of navi

Complete Axes Reference

Axis Name	Description	Example
child::	Children of context node (default)	child::book = book
attribute::	Attributes of context node	attribute::category = @ca tegory
parent::	Parent of context node	parent::bookstore
ancestor::	All ancestors up to root	ancestor::bookstore
ancestor-or-self::	Ancestors including current	ancestor-or-self::*
descendant::	All children, grandchildren, etc.	descendant::title
descendant-or-self::	Descendants including current	descendant-or-self::node()
following::	All nodes after context (excluding descendants)	following::book
following-sibling::	All siblings after context	following-sibling::magazine
preceding::	All nodes before context	preceding::book
preceding-sibling::	All siblings before context	preceding-sibling::book

4.1. XPath: Path Expressions
4.1.2. Axes

Axis Examples with Our XML

- //book[1]/child::* # All children of first book
- //book[1]/attribute::* # All attributes of first book
- //title/parent::* # Parent of any title (book or magazine)
- //price/ancestor::bookstore # Bookstore ancestor of any price
- //book/following-sibling::magazine # Magazines that follow books
- //magazine/preceding-sibling::book # Books that precede magazines
- //book/descendant::text() # All text within books

4.1. XPath: Path Expressions
4.1.2. Axes

Abbreviated vs. Full Syntax

Abbreviated	Full Syntax	Description
book	child::book	Book children
@category	attribute::category	Category attribute
//title	/descendant-or-self::node()/child::title	Any title
..	parent::node()	Parent node
.	self::node()	Current node
book[1]	child::book[1]	First book child

4.1. XPath: Path Expressions
4.1.3. Filters and Predicates

Predicate Syntax

Predicates are expressions in square brackets [] that filter node sets.

Numeric Predicates

- //book[1] # First book element
- //book[last()] # Last book element
- //book[last()-1] # Second-to-last book element
- //book[position()<3] # First two books
- //book[position() mod 2 = 0] # Even-positioned books

4.1. XPath: Path Expressions
4.1.3. Filters and Predicates

Attribute Predicates

- //book[@category] # Books with a category attribute
- //book[@category='fiction'] # Books in fiction category
- //book[@category!='fiction'] # Books not in fiction
- //book[price/@currency='USD'] # Books with USD price attribute
- //@category[.='computer'] # Category attributes with value 'computer'

4.1. XPath: Path Expressions
4.1.3. Filters and Predicates

Element Value Predicates

- //book[price>30] # Books priced over 30
- //book[price>40 and price<60] # Books between 40 and 60
- //book[author='Jane Austen'] # Books by Jane Austen
- //book[contains(title,'XML')] # Books with 'XML' in title
- //book[starts-with(@category,'comp')] # Category starting with 'comp'
- //book[year>2000][price<50] # Books after 2000 under $50

4.1. XPath: Path Expressions
4.1.3. Filters and Predicates

Nested Predicates

- //book[author='Jane Austen']/title # Titles of books by Jane Austen
- //book[price>30]/@category # Categories of expensive books
- //book[author='Jane Austen'][1] # First book by Jane Austen
- //book[1][@category='fiction'] # First book if it's fiction

4.1. XPath: Path Expressions
4.1.3. Filters and Predicates

Complex Predicate Examples

Books where author has a middle initial (author contains two spaces)
//book[contains(author, ' ') and substring-after(author, ' ') != '']

Books where title is longer than 20 characters
//book[string-length(title) > 20]

- # Books with at least one review (if reviews existed)

//book[count(review) > 0]

- # Books where year is between 2000 and 2010

//book[year >= 2000 and year <= 2010]

4.1. XPath: Path Expressions
4.1.4 Advanced XPath Functions

String Functions

- //book[contains(title, 'XML')] # Title contains 'XML'
- //book[starts-with(title, 'The')] # Title starts with 'The'
- //book[ends-with(title, 'Guide')] # Title ends with 'Guide'
- //book[matches(title, '^[A-Z].*')] # Title starts with capital (regex)
- concat(//book[1]/title, ' by ', //book[1]/author) # String concatenation
- substring(//title, 1, 5) # First 5 characters of title
- string-length(//title) # Length of title text
- normalize-space(//title) # Remove extra whitespace

4.1. XPath: Path Expressions
4.1.4 Advanced XPath Functions

Numeric Functions

- //book[price > 30] # Simple comparison
- //book[floor(price) = 49] # Price floor = 49
- //book[ceiling(price) = 50] # Price ceiling = 50
- sum(//book/price) # Sum of all book prices
- count(//book) # Number of books
- avg(//book/price) # Average price (XPath 2.0+)
- min(//book/price) # Minimum price
- max(//book/price) # Maximum price

4.1. XPath: Path Expressions
4.1.4 Advanced XPath Functions

Boolean Functions

- //book[price > 30 and year > 2000] # AND condition
- //book[author='Smith' or author='Jones'] # OR condition
- //book[not(@category)] # Books without category
- //book[boolean(author)] # Books with author element

4.1. XPath: Path Expressions
4.1.4 Advanced XPath Functions

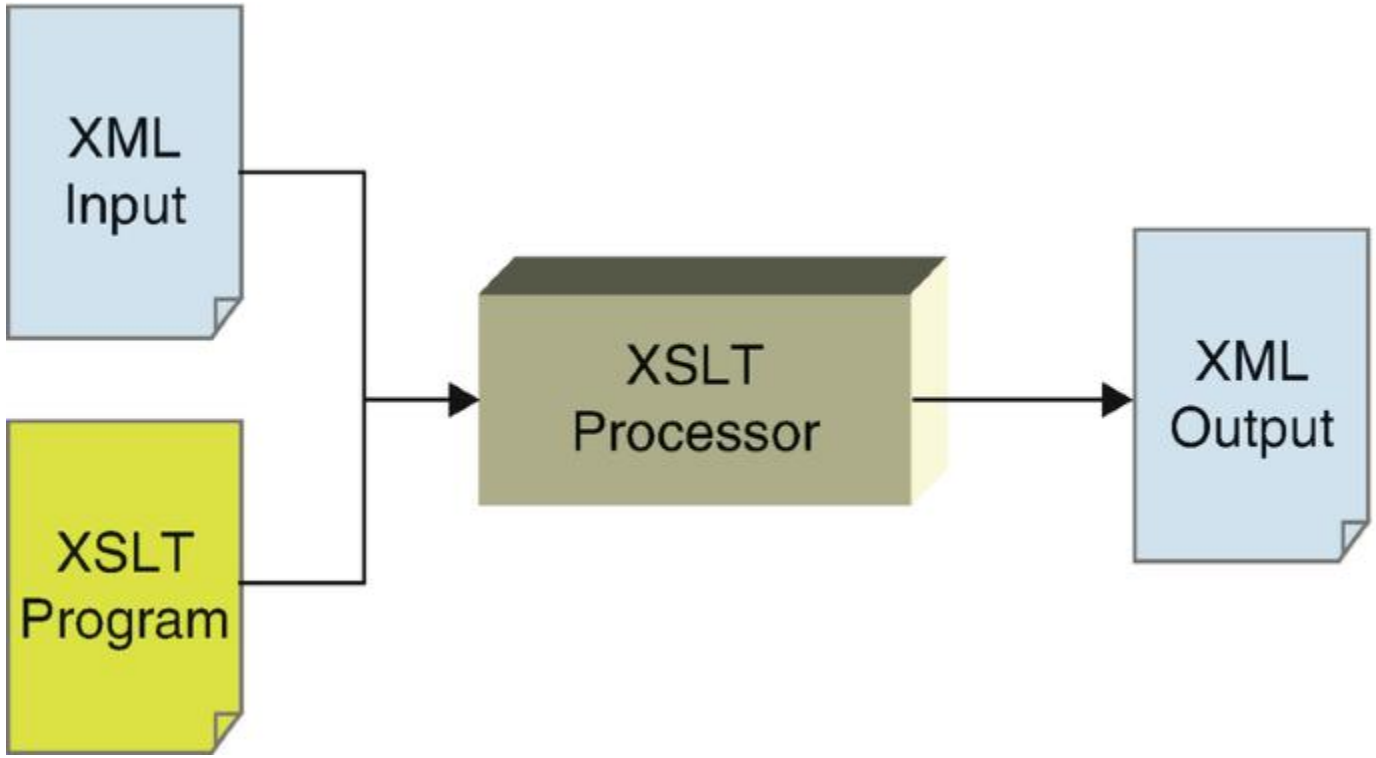
Node Functions

- name(//book[1]) # Element name: 'book'
- local-name(//book[1]) # Local name (ignores namespace)
- namespace-uri(//book[1]) # Namespace URI
- lang('en') # Tests if language is English

4.2. XSL and XSLT Transformations
4.2.1 Introduction to XSLT

XSLT (eXtensible Stylesheet Language Transformations) is a language for transforming XML documents into other formats (HTML, text, XML).

XSLT Processing Model



4.2. XSL and XSLT Transformations

Basic XSLT Structure

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

    <xsl:output method="html" indent="yes"/>

    <!-- Templates go here -->

</xsl:stylesheet>
```

4.2. XSL and XSLT Transformations

4.2.2 XSLT Elements Reference

Core Elements

Element	Purpose	Example
<xsl:template>	Defines a template rule	<xsl:template match="/">
<xsl:value-of>	Extracts node value	<xsl:value-of select="title"/>
<xsl:for-each>	Loops through node set	<xsl:for-each select="book">
<xsl:if>	Conditional processing	<xsl:if test="price>30">
<xsl:choose>	Multiple conditions	<xsl:choose>...<xsl:when>...
<xsl:apply-templates>	Applies templates to children	<xsl:apply-templates select="author"/>
<xsl:sort>	Sorts output	<xsl:sort select="price"/>
<xsl:variable>	Declares variable	<xsl:variable name="tax" select="0.1"/>

4.2. XSL and XSLT Transformations
4.2.2 XSLT Elements Reference

<xsl:param>	Declares parameter	<xsl:param name="discount"/>
<xsl:copy>	Copies current node	<xsl:copy/>
<xsl:copy-of>	Copies node set	<xsl:copy-of select="*"/>
<xsl:attribute>	Creates attribute	<xsl:attribute name="class">
<xsl:element>	Creates element	<xsl:element name="div">
<xsl:text>	Writes text	<xsl:text>Hello</xsl:text>
<xsl:comment>	Creates comment	<xsl:comment>Generated</xsl:comment>

4.2. XSL and XSLT Transformations
4.2.3 Complete XSLT Examples

Example 1: Transforming XML to HTML Table

Source XML (books.xml):

```
<?xml version="1.0"?>
<library>
   <book isbn="1234">
      <title>XML Basics</title>
      <author>John Smith</author>
      <price currency="USD">39.99</price>
      <year>2023</year>
   </book>
   <book isbn="5678">
      <title>XSLT Guide</title>
      <author>Jane Doe</author>
      <price currency="USD">49.99</price>
      <year>2024</year>
   </book>
</library>
```

4.2. XSL and XSLT Transformations

XSLT Stylesheet (books-to-html.xsl):

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:output method="html" indent="yes" doctype-public="-//W3C//DTD HTML 4.01//EN"/>
    <!-- Root template -->
    <xsl:template match="/">
        <html>
            <head>
                <title>Book Library</title>
            </head>
            <body>
                <h1>Library Catalog</h1>
                <table>
                    <thead>
                        <tr>
                            <th>ISBN</th>
                            <th>Title</th>
                            <th>Author</th>
                            <th>Year</th>
                            <th>Price</th>
                        </tr>
                    </thead>
                    <tbody>
                        <xsl:apply-templates select="//book"/>
                    </tbody>
                </table>
            </body>
        </html>
```

4.2. XSL and XSLT Transformations

```
</xsl:template>
   <!-- Template for each book -->
   <xsl:template match="book">
      <tr>
         <td><xsl:value-of select="@isbn"/></td>
         <td><xsl:value-of select="title"/></td>
         <td><xsl:value-of select="author"/></td>
         <td><xsl:value-of select="year"/></td>
         <td>
            <xsl:value-of select="price"/>
            <xsl:text> </xsl:text>
            <xsl:value-of select="price/@currency"/>
         </td>
      </tr>
   </xsl:template>
</xsl:stylesheet>
```

4.2. XSL and XSLT Transformations

Example 2: Conditional Formatting with xsl:if

```
<xsl:template match="book">
    <tr>
        <!-- Highlight expensive books -->
        <xsl:if test="price > 45">
            <xsl:attribute name="style">background-color:
yellow;</xsl:attribute>
        </xsl:if>
```

4.2. XSL and XSLT Transformations

```
<td><xsl:value-of select="title"/></td>
      <td><xsl:value-of select="author"/></td>
      <td>
         <xsl:choose>
            <xsl:when test="price > 50">
               <span style="color: red;">
                  <xsl:value-of select="price"/>
               </span>
```

4.2. XSL and XSLT Transformations

```
</xsl:when>
        <xsl:when test="price > 30">
            <span style="color: orange;">
                <xsl:value-of select="price"/>
            </span>
        </xsl:when>
        <xsl:otherwise>
            <span style="color: green;">
                <xsl:value-of select="price"/>
            </span>
```

4.2. XSL and XSLT Transformations

```
 </xsl:otherwise>
          </xsl:choose>
       </td>
    </tr>
</xsl:template>
```

4.2. XSL and XSLT Transformations

Example 3: Sorting and Grouping

```
<xsl:template match="/">
    <html>
        <body>
            <h1>Books by Author</h1>
            <!-- Group books by author using Muenchian method -->
            <xsl:for-each select="//author">
                <xsl:if test="generate-id(.) = generate-id(key('author', .)[1])">
                    <h2><xsl:value-of select="."/></h2>
```

4.2. XSL and XSLT Transformations

```
<ul>
        <xsl:for-each select="key('author', .)">
            <xsl:sort select="price" order="descending"/>
            <li>
                <xsl:value-of select="title"/>
                (<xsl:value-of select="price"/>)
            </li>
        </xsl:for-each>
    </ul>
```

4.2. XSL and XSLT Transformations

```
</xsl:if>
            </xsl:for-each>
        </body>
    </html>
</xsl:template>
<!-- Key definition for grouping -->
<xsl:key name="author" match="book" use="author"/>
```

4.2. XSL and XSLT Transformations

Example 4: Creating New XML Structure

```
<xsl:template match="/">
   <catalog>
      <summary>
         <totalBooks>
            <xsl:value-of select="count(//book)"/>
         </totalBooks>
         <averagePrice>
            <xsl:value-of select="sum(//price) div count(//book)"/>
         </averagePrice>
```

4.2. XSL and XSLT Transformations

```
</summary>
    <booksByYear>
      <xsl:for-each select="//book">
        <xsl:sort select="year"/>
        <xsl:if test="generate-id(.) = generate-id(key('year', year)[1])">
          <year value="{year}">
            <xsl:attribute name="count">
              <xsl:value-of select="count(key('year', year))"/>
            </xsl:attribute>
          </year>
        </xsl:if>
      </xsl:for-each>
    </booksByYear>
```

4.2. XSL and XSLT Transformations

```
 <xsl:apply-templates select="//book"/>
   </catalog>
</xsl:template>
<xsl:key name="year" match="book" use="year"/>

<xsl:template match="book">
   <bookEntry isbn="{@isbn}">
      <xsl:copy-of select="title|author|year"/>
      <priceWithTax>
         <xsl:value-of select="price * 1.1"/>
      </priceWithTax>
   </bookEntry>
</xsl:template>
```

4.2. XSL and XSLT Transformations
4.2.4 XSLT Modes and Priorities

Using Modes for Different Views

```
<xsl:template match="book" mode="summary">
   <div class="summary">
      <xsl:value-of select="title"/> by <xsl:value-of select="author"/>
   </div>
</xsl:template>
<xsl:template match="book" mode="detail">
   <div class="detail">
      <h3><xsl:value-of select="title"/></h3>
      <p>Author: <xsl:value-of select="author"/></p>
```

4.2. XSL and XSLT Transformations
4.2.4 XSLT Modes and Priorities

```
<p>Year: <xsl:value-of select="year"/></p>
      <p>Price: <xsl:value-of select="price"/></p>
   </div>
</xsl:template>
<xsl:template match="/">
   <div>
      <h2>Summary View</h2>
      <xsl:apply-templates select="//book" mode="summary"/>
      <h2>Detail View</h2>
      <xsl:apply-templates select="//book" mode="detail"/>
   </div>
</xsl:template>
```

4.2. XSL and XSLT Transformations

Template Priorities

```
<!-- Generic template (low priority) -->
<xsl:template match="book" priority="0">
    <div class="generic-book">
        <xsl:value-of select="title"/>
    </div>
</xsl:template>
<!-- Specific template (high priority) -->
<xsl:template match="book[@category='fiction']" priority="2">
```

4.2. XSL and XSLT Transformations

```
 <div class="fiction-book">
        <xsl:value-of select="title"/>
    </div>
</xsl:template>
<!-- Most specific template (highest priority) -->
<xsl:template match="book[author='Jane Austen']" priority="3">
    <div class="classic">
          <xsl:value-of select="title"/> - A Classic!
    </div>
</xsl:template>
```

4.2. XSL and XSLT Transformations
4.2.5 XSLT Variables and Parameters

Variables

```
<xsl:variable name="taxRate" select="0.1"/>
<xsl:variable name="siteName">My Book Library</xsl:variable>
<xsl:template match="book">
   <div>
      <xsl:value-of select="concat($siteName, ': ', title)"/>
      <br/>
      Price with tax:
      <xsl:value-of select="price * (1 + $taxRate)"/>
   </div>
</xsl:template>
```

4.2. XSL and XSLT Transformations
4.2.5 XSLT Variables and Parameters

Parameters

```
<xsl:param name="discount" select="0"/>
<xsl:param name="currency" select="'USD'"/>
<xsl:template match="book">
   <div>
      <xsl:value-of select="title"/>
      <br/>
      Discounted Price:
      <xsl:value-of select="price * (1 - $discount)"/>
      <xsl:text> </xsl:text>
      <xsl:value-of select="$currency"/>
   </div>
</xsl:template>
```

4.2. XSL and XSLT Transformations
4.2.5 XSLT Variables and Parameters

Calling with parameters:

```
<xsl:apply-templates select="book">
   <xsl:with-param name="discount" select="0.2"/>
   <xsl:with-param name="currency" select="'EUR'"/>
</xsl:apply-templates>
```