Semi-Structured Data and XML

CSE462 Database Concepts

Demian Lessa

Department of Computer Science and Engineering
State University of New York, Buffalo

March 30 – April 08, 2011

Outline

Motivation

- The data models seen so far start with a schema.
- The schema is a rigid framework into which data is placed.
- Query engines know about the schemas.
- Data can be organized in special data structures.
- This provides a path for efficient implementations.

Motivation

- Semi-structured models provide flexibility.
- A distinguishing aspect of the model: there is no schema.
- The data is self-describing: it carries information about its schema.
- The schema may vary arbitrarily within a database and over time.
- As you may suspect, this makes query processing harder.
- But what are some advantages?
- User-friendliness: textual, self-describing data.
- Data may be augmented in-place with new attributes and relationships.
- Data exchange: messages represented as semi-structured data.
- Data integration: describe similar data with different schemas.

Data Representation

- A semi-structured database is a collection of nodes.
- Each node is either a leaf node or an interior node.
- Leaf nodes have associated data, which can be of any atomic type.
- Interior nodes have one or more outgoing arcs (directed edges).
- Arcs are labeled to indicate how the head node relates to the tail node.
- The root node is a special interior node with no incoming arcs.
- Every node must be reachable from the root.
- The database is not necessarily a tree.

To-Do

- Go over examples 11.1 and 11.2.

Outline

XML

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<bib>
  <book year="1994" copies="2">
    <title>TCP/IP Illustrated</title>
    <author><last>Stevens</last><first>W.</first></author>
    <publisher>Addison-Wesley</publisher>
    <price>65.95</price>
  </book>
  <book year="1992">
    <title>Advanced Programming in the Unix environment</title>
    <author><last>Stevens</last><first>W.</first></author>
    <publisher>Addison-Wesley</publisher>
    <price>83.45</price>
  </book>
  <book year="2000">
    <title>Data on the Web</title>
    <author><last>Abiteboul</last><first>Serge</first></author>
    <author><last>Buneman</last><first>Peter</first></author>
    <author><last>Suciu</last><first>Dan</first></author>
    <author><last>Stevens</last><first>W.</first></author>
    <publisher>Morgan Kaufmann Publishers</publisher>
    <price>39.95</price>
  </book>
  <book year="1999">
    <title>The Economics of Technology and Content for Digital TV</title>
    <editor>
      <last>Gerbarg</last><first>Darcy</first>
      <affiliation>CITI</affiliation>
    </editor>
    <publisher>Kluwer Academic Publishers</publisher>
    <price>129.95</price>
  </book>
</bib>
```

XML in a Nutshell

- eXtensible Markup Language.
- W3C recommendation.
- A markup language, like HTML.
- Textual data format.
- Represents hierarchical data.
- Designed to carry data, not display.
- Separates data from presentation.
- Simplifies data sharing and transport.
- Self-descriptive.
- Tags are not predefined.
- Well-formed vs valid.
- Validation using DTD (and others).
- Programmatic APIs: SAX, DOM.

XML Documents

```
1  <?xml version="1.0" encoding="UTF-8" standalone="yes"?>
2  <bib>
3    <book year="1994" copies="2">
4      <title>TCP/IP Illustrated</title>
5      <author><last>Stevens</last><first>W.</first></author>
6      <publisher>Addison-Wesley</publisher>
7      <price>65.95</price>
8    </book>
9    <book year="1992">
10     <title>Advanced Programming in the Unix environment</title>
11     <author><last>Stevens</last><first>W.</first></author>
12     <publisher>Addison-Wesley</publisher>
13     <price>83.45</price>
14   </book>
15   <book year="2000">
16     <title>Data on the Web</title>
17     <author><last>Abiteboul</last><first>Serge</first></author>
18     <author><last>Buneman</last><first>Peter</first></author>
19     <author><last>Suciu</last><first>Dan</first></author>
20     <author><last>Stevens</last><first>W.</first></author>
21     <publisher>Morgan Kaufmann Publishers</publisher>
22     <price>39.95</price>
23   </book>
24   <book year="1999">
25     <title>The Economics of Technology and Content for Digital TV</title>
26     <editor>
27       <last>Gerbarg</last><first>Darcy</first>
28       <affiliation>CITI</affiliation>
29     </editor>
30     <publisher>Kluwer Academic Publishers</publisher>
31     <price>129.95</price>
32   </book>
33 </bib>
```

XML Document

- XML declaration.
- One root element.
- Elements may nest arbitrarily.
- Preserves white spaces.
- Special characters: <, >, &, ', ".
- Comments: <!-- comment -->.

XML Documents

```
1  <?xml version="1.0" encoding="UTF-8" standalone="yes"?>
2  <bib>
3    <book year="1994" copies="2">
4      <title>TCP/IP Illustrated</title>
5      <author><last>Stevens</last><first>W.</first></author>
6      <publisher>Addison-Wesley</publisher>
7      <price>65.95</price>
8    </book>
9    <book year="1992">
10     <title>Advanced Programming in the Unix environment</title>
11     <author><last>Stevens</last><first>W.</first></author>
12     <publisher>Addison-Wesley</publisher>
13     <price>83.45</price>
14   </book>
15   <book year="2000">
16     <title>Data on the Web</title>
17     <author><last>Abiteboul</last><first>Serge</first></author>
18     <author><last>Buneman</last><first>Peter</first></author>
19     <author><last>Suciu</last><first>Dan</first></author>
20     <author><last>Stevens</last><first>W.</first></author>
21     <publisher>Morgan Kaufmann Publishers</publisher>
22     <price>39.95</price>
23   </book>
24   <book year="1999">
25     <title>The Economics of Technology and Content for Digital TV</title>
26     <editor>
27       <last>Gerbarg</last><first>Darcy</first>
28       <affiliation>CITI</affiliation>
29     </editor>
30     <publisher>Kluwer Academic Publishers</publisher>
31     <price>129.95</price>
32   </book>
33 </bib>
```

XML Element

- Start tag, contents, end tag.
- Start tag: <tag>.
- Start tag may contain attributes.
- Contents: data or more element(s).
- End tag: </tag>.
- End tag must match start tag.

XML Documents

```
1  <?xml version="1.0" encoding="UTF-8" standalone="yes"?>
2  <bib>
3    <book year="1994" copies="2">
4      <title>TCP/IP Illustrated</title>
5      <author><last>Stevens</last><first>W.</first></author>
6      <publisher>Addison-Wesley</publisher>
7      <price>65.95</price>
8    </book>
9    <book year="1992">
10     <title>Advanced Programming in the Unix environment</title>
11     <author><last>Stevens</last><first>W.</first></author>
12     <publisher>Addison-Wesley</publisher>
13     <price>83.45</price>
14   </book>
15   <book year="2000">
16     <title>Data on the Web</title>
17     <author><last>Abiteboul</last><first>Serge</first></author>
18     <author><last>Buneman</last><first>Peter</first></author>
19     <author><last>Suciu</last><first>Dan</first></author>
20     <author><last>Stevens</last><first>W.</first></author>
21     <publisher>Morgan Kaufmann Publishers</publisher>
22     <price>39.95</price>
23   </book>
24   <book year="1999">
25     <title>The Economics of Technology and Content for Digital TV</title>
26     <editor>
27       <last>Gerbarg</last><first>Darcy</first>
28       <affiliation>CITI</affiliation>
29     </editor>
30     <publisher>Kluwer Academic Publishers</publisher>
31     <price>129.95</price>
32   </book>
33 </bib>
```

Well-Formed XML

- No schema.
- Syntactic rules observed.

Valid XML

- Some form of schema.
- Schema: a grammar of valid XML.
- Syntactic rules observed.
- Schema rules observed.

To-Do

- Go over example 11.4.

XML Attributes

```
1  <?xml version="1.0" encoding="UTF-8" standalone="yes"?>
2  <bib>
3    <book year="1994" copies="2">
4      <title>TCP/IP Illustrated</title>
5      <author><last>Stevens</last><first>W.</first></author>
6      <publisher>Addison-Wesley</publisher>
7      <price>65.95</price>
8    </book>
9    <book year="1992">
10     <title>Advanced Programming in the Unix environment</title>
11     <author><last>Stevens</last><first>W.</first></author>
12     <publisher>Addison-Wesley</publisher>
13     <price>83.45</price>
14   </book>
15   <book year="2000">
16     <title>Data on the Web</title>
17     <author><last>Abiteboul</last><first>Serge</first></author>
18     <author><last>Buneman</last><first>Peter</first></author>
19     <author><last>Suciu</last><first>Dan</first></author>
20     <author><last>Stevens</last><first>W.</first></author>
21     <publisher>Morgan Kaufmann Publishers</publisher>
22     <price>39.95</price>
23   </book>
24   <book year="1999">
25     <title>The Economics of Technology and Content for Digital TV</title>
26     <editor>
27       <last>Gerbarg</last><first>Darcy</first>
28       <affiliation>CITI</affiliation>
29     </editor>
30     <publisher>Kluwer Academic Publishers</publisher>
31     <price>129.95</price>
32   </book>
33 </bib>
```

XML Attribute

- Name-value pair within some tag.
- Values must be quoted.
- Values may represent any data.
- Conceptually, like a leaf node.
- Uses: keys, connections, metadata.

To-Do

- Go over example 11.5, 11.6.

XML Namespaces

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<table>
  <tr>
    <td>Apples</td>
    <td>Bananas</td>
  </tr>
</table>
```

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<table>
  <name>African Coffee Table</name>
  <width>80</width>
  <length>120</length>
</table>
```

Problem

- 1st framgment: HTML table.
- 2nd framgment: furniture information.
- Table elements in both documents.
- But different content and meaning.
- Indistinguishable to XML!
- Say we need to combine the fragments.
- How do solve the name conflict?

XML Namespaces

```
1  <?xml version="1.0" encoding="UTF−8" standalone="yes"?>
2  <tables>
3    <h:table>
4      <h:tr>
5        <h:td>Apples</h:td>
6        <h:td>Bananas</h:td>
7      </h:tr>
8    </h:table>
9    <f:table>
10     <f:name>African Coffee Table</f:name>
11     <f:width>80</f:width>
12     <f:length>120</f:length>
13   </f:table>
14 </tables>
```

Solution

- Namespaces to the rescue.
- Logical context for identifiers.
- Tags are qualified.
- “h” prefix: HTML namespace.
- “f” prefix: furniture namespace.
- Table elements are distinguishable.
- No more conflicts in the modified XML.
- However, the document is invalid.
- Name prefixes must be bound!

XML Namespaces

```
1  <?xml version="1.0" encoding="UTF-8" standalone="yes"?>
2  <tables>
3    <h:table>
4      <h:tr>
5        <h:td>Apples</h:td>
6        <h:td>Bananas</h:td>
7      </h:tr>
8    </h:table>
9    <f:table>
10     <f:name>African Coffee Table</f:name>
11     <f:width>80</f:width>
12     <f:length>120</f:length>
13   </f:table>
14 </tables>
```

Solution (cont.)

- Define a namespace for each prefix.
- Use xmlns attributes.
- Syntax: xmlns:prefix="URI"
- Form #1: at the prefixed element.
- Form #2: at the root element.
- URIs not used to lookup information!
- Just a unique logical identifier.

XML Namespaces

```
1  <?xml version="1.0" encoding="UTF−8" standalone="yes"?>
2  <tables>
3    <h:table xmlns:h="http://www.w3.org/TR/html4/">
4      <h:tr>
5        <h:td>Apples</h:td>
6        <h:td>Bananas</h:td>
7      </h:tr>
8    </h:table>
9    <f:table xmlns:f="http://www.buffalo.edu/furniture">
10     <f:name>African Coffee Table</f:name>
11     <f:width>80</f:width>
12     <f:length>120</f:length>
13   </f:table>
14 </tables>
```

Solution (cont.)

- Namespaces defined within elements.
- Child elements with same prefix are in the same namespace.
- Document is valid.

XML Namespaces

```
1  <?xml version="1.0" encoding="UTF−8" standalone="yes"?>
2  <tables xmlns:h="http://www.w3.org/TR/html4/"
3          xmlns:f="http://www.buffalo.edu/furniture">
4   <h:table>
5    <h:tr>
6     <h:td>Apples</h:td>
7     <h:td>Bananas</h:td>
8    </h:tr>
9   </h:table>
10  <f:table>
11   <f:name>African Coffee Table</f:name>
12   <f:width>80</f:width>
13   <f:length>120</f:length>
14  </f:table>
15 </tables>
```

Solution (cont.)

- Namespaces within the root.
- Elements with same prefix are in the same namespace.
- Document is also valid.

To-Do

- Go over example 11.7.
- Go over exercises 11.2.2–11.2.4.

Outline

DTD

```
<!DOCTYPE bib [
  <!ELEMENT bib (book*)>
  <!ELEMENT book (title, (author+|editor+), publisher, price?)>
  <!ATTLIST book year CDATA #REQUIRED
                 copies CDATA #IMPLIED>
  <!ELEMENT title (#PCDATA)>
  <!ELEMENT author ((last, first)|(first,last))>
  <!ELEMENT publisher (#PCDATA)>
  <!ELEMENT price (#PCDATA)>
  <!ELEMENT editor ((first, last, affiliation?)|(last, first, affiliation?))>
  <!ELEMENT first (#PCDATA)>
  <!ELEMENT last (#PCDATA)>
  <!ELEMENT affiliation (#PCDATA)>
]>
...
```

DTD in a Nutshell

- Document Type Definition.
- Defines the structure of documents.
- Allows validation of documents.
- Internal: within the document
- External: separate text file
- Declarations: element and attribute-list.

DTD

```
<!DOCTYPE bib [
  <!ELEMENT bib (book*)>
  <!ELEMENT book (title, (author+|editor+), publisher, price?)>
  <!ATTLIST book year CDATA #REQUIRED
                 copies CDATA #IMPLIED>
  <!ELEMENT title (#PCDATA)>
  <!ELEMENT author ((last, first)|(first,last))>
  <!ELEMENT publisher (#PCDATA)>
  <!ELEMENT price (#PCDATA)>
  <!ELEMENT editor ((first, last, affiliation?)|(last, first, affiliation?))>
  <!ELEMENT first (#PCDATA)>
  <!ELEMENT last (#PCDATA)>
  <!ELEMENT affiliation (#PCDATA)>
]>
...
```

Element Type

- empty:
  <!ELEMENT name EMPTY>
- parsed data:
  <!ELEMENT name (#PCDATA)>
- children:
  <!ELEMENT name (child1, child2, ...)>
- one child:
  <!ELEMENT name (child)>
- one or more children:
  <!ELEMENT name (child+)>
- zero or more children:
  <!ELEMENT name (child*)>
- zero or one child:
  <!ELEMENT name (child?)>
- either/or:
  <!ELEMENT name (child1 | child2)>
- mixed:
  <!ELEMENT name (#PCDATA | child)*>
- any:
  <!ELEMENT name ANY>

DTD

```
1  <!DOCTYPE bib [
2    <!ELEMENT bib (book*)>
3    <!ELEMENT book (title, (author+|editor+), publisher, price?)>
4    <!ATTLIST book year CDATA #REQUIRED
5                        copies CDATA #IMPLIED>
6    <!ELEMENT title (#PCDATA)>
7    <!ELEMENT author ((last, first)|(first,last))>
8    <!ELEMENT publisher (#PCDATA)>
9    <!ELEMENT price (#PCDATA)>
10   <!ELEMENT editor ((first, last, affiliation?)|(last, first, affiliation?))>
11   <!ELEMENT first (#PCDATA)>
12   <!ELEMENT last (#PCDATA)>
13   <!ELEMENT affiliation (#PCDATA)>
14 ]>
15 ...
```

Attribute Default Value

- CDATA:
  character data
- $(val1|val2|\ldots)$:
  one value from the list
- ID:
  unique identifier (within the XML!)
- IDREF:
  ID of another element
- IDREFS:
  list IDs of other elements

```
1  <!DOCTYPE bib [
2    <!ELEMENT bib (book*)>
3    <!ELEMENT book (title, (author+|editor+), publisher, price?)>
4    <!ATTLIST book year CDATA #REQUIRED
5                   copies CDATA #IMPLIED>
6    <!ELEMENT title (#PCDATA)>
7    <!ELEMENT author ((last, first)|(first,last))>
8    <!ELEMENT publisher (#PCDATA)>
9    <!ELEMENT price (#PCDATA)>
10   <!ELEMENT editor ((first, last, affiliation?)|(last, first, affiliation?))>
11   <!ELEMENT first (#PCDATA)>
12   <!ELEMENT last (#PCDATA)>
13   <!ELEMENT affiliation (#PCDATA)>
14 ]>
15 ...
```

Attribute Type

- Syntax:

  <!ATTLIST element-name
  attr1 type default
  attr2 type default
  ...
  attrN type default>

- Examples:

  <!ATTLIST person
  ssn ID #REQUIRED
  blood CDATA #FIXED "O-"
  fax CDATA #IMPLIED>
  <!ATTLIST payment
  type (check | cash) "cash">

```
1  <!DOCTYPE family [
2    <!ELEMENT family (person*)>
3    <!ELEMENT person (first, last)>
4    <!ATTLIST person id ID #REQUIRED
5                     parents IDREFS #IMPLIED>
6  ]>
7  <family>
8    <person id="e10001" parents="e10002 e10003">
9      <first>Bart</first>
10     <last>Simpson</last>
11   </person>
12   <person id="e10002">
13     <first>Homer</first>
14     <last>Simpson</last>
15   </person>
16   <person id="e10003">
17     <first>Marge</first>
18     <last>Simpson</last>
19   </person>
20 </family>
```

To-Do

- Go over examples 11.8–11.11.
- Go over exercises 11.3.1, 11.3.5.

Outline

XSchema

```
1 <?xml version="1.0" encoding="UTF−8"?>
2 <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
3 ...
4 </xs:schema>
```

```
1 <?xml version="1.0" encoding="UTF−8"?>
2 <bib xmlns="http://www.cse.buffalo.edu"
3       xmlns:xsi="http://www.w3.org/2001/XMLSchema−instance"
4       xsi:schemaLocation="http://www.cse.buffalo.edu bib.xsd">
5 ...
6 </bib>
```

XSchema in a Nutshell

- XML Schema.
- An XML document itself.
- More expressive than DTDs.
- Specify types: integer, string, etc.
- Define cardinality constraints.
- Declare keys and foreign keys.
- Namespace: xs.
- Prefix xs: causes tags to be interpreted based on XSchema rules.

XSchema

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
3    <xs:complexType name="authorType">
4      <xs:sequence>
5        <xs:element name="last" type="xs:string" />
6        <xs:element name="first" type="xs:string" />
7      </xs:sequence>
8    </xs:complexType>
9    <xs:complexType name="editorType">
10     <xs:sequence>
11       <xs:element name="last" type="xs:string" />
12       <xs:element name="first" type="xs:string" />
13       <xs:element name="affiliation" type="xs:string"
14                    minOccurs="0" />
15     </xs:sequence>
16   </xs:complexType>
17   ...
18 </xs:schema>
```

XSchema Elements

- Similar to DTD's ELEMENT.
- Defines structure and/or constraints.
- name: element's tag name.
- type: simple or complex type.
- Modifiers: several!
- Simple: xs:string, xs:integer, etc.
- Complex: xs:sequence, xs:choice, etc.

XSchema

```
1  <?xml version="1.0" encoding="UTF−8"?>
2  <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
3    ...
4    <xs:complexType name="bookType">
5      <xs:attribute name="year" type="xs:integer" use="required"
6                    minInclusive="1900" />
7      <xs:attribute name="copies" type="xs:integer" default="0" />
8      <xs:sequence>
9        <xs:element name="title" type="xs:string" />
10       <xs:choice>
11         <xs:sequence>
12           <xs:element name="author" type="authorType"
13                       maxOccurs="unbounded" />
14         </xs:sequence>
15         <xs:sequence>
16           <xs:element name="editor" type="editorType"
17                       maxOccurs="unbounded" />
18         </xs:sequence>
19       <xs:choice>
20       <xs:element name="publisher" type="xs:string" />
21       <xs:element name="price" type="xs:decimal" minOccurs="0" />
22     </xs:sequence>
23   </xs:complexType>
24 </xs:schema>
```

XSchema Attributes

- Defines information for complex types.
- name: attribute's name.
- type: primitive type.
- use: required or optional.
- default: value used if none is provided.
- Modifiers: several!

```
1  <?xml version="1.0" encoding="UTF−8"?>
2  <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
3    <xs:simpleType name="bookGenre" >
4      <xs:restriction base="xs:string" >
5        <xs:enumeration value="Database Theory" />
6        <xs:enumeration value="Programming Languages" />
7        <xs:enumeration value="Software Engineering" />
8        ...
9      </xs:restriction>
10   </xs:simpleType>
11   <xs:simpleType name="bookYear" >
12     <xs:restriction base="xs:integer" >
13       <xs:minInclusive value="1900" />
14       ...
15     <xs:restriction >
16   </xs:simpleType>
17   ...
18 </xs:schema>
```

XSchema Restricted Types

- Restricted simple types.
- Same underlying domain.
- Restricted set of values.
- Range restriction (numerical).
- Enumerations.

To-Do

- Go over examples 11.12–11.17.
- Go over exercises 11.4.1, 11.4.2.

XSchema

```
1  <?xml version="1.0" encoding="UTF−8"?>
2  <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
3    ...
4    <xs:element name="bib">
5      <xs:complexType>
6        <xs:sequence>
7          <xs:element name="book" type="bookType"
8                      minOccurs="0" maxOccurs="unbounded" />
9        </xs:sequence>
10     </xs:complexType>
11     <xs:key name="bookKey" >
12       <xs:selector xpath="book" />
13       <xs:field xpath="title" />
14       <xs:field xpath="@year" />
15     </xs:key>
16   </xs:element>
17   ...
18 </xs:schema>
```

XSchema Keys

- Given a class of elements, field values within the class are unique.
- Class: sequence of elements.
- Class is defined by a selector.
- Field: subelement/attribute of the last element on the selector path.
- Use xs:key if fields must exist.
- Use xs:unique if some field may not exist.

```
1  <?xml version="1.0" encoding="UTF−8"?>
2  <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
3    ...
4    <xs:complexType name="orderType">
5      <xs:attribute name="oid" type="xs:string" use="required" />
6      <xs:attribute name="qty" type="xs:integer" use="required" />
7      <xs:sequence>
8        <xs:element name="bookTitle" type="xs:string" />
9        <xs:element name="bookYear" type="xs:integer" />
10     </xs:sequence>
11   </xs:complexType>
12   <xs:element name="bib">
13     <xs:complexType>
14       <xs:sequence>
15         <xs:element name="book" type="bookType"
16                     minOccurs="0" maxOccurs="unbounded" />
17         <xs:element name="order" type="orderType"
18                     minOccurs="0" maxOccurs="unbounded" />
19       </xs:sequence>
20     </xs:complexType>
21     <xs:key name="bookKey" >
22       <xs:selector xpath="book" />
23       <xs:field xpath="title" />
24       <xs:field xpath="@year" />
25     </xs:key>
26     <xs:keyref name="orderBookRef" refers="bookKey">
27       <xs:selector xpath="order"/>
28       <xs:field xpath="bookTitle"/>
29       <xs:field xpath="bookYear"/>
30     </xs:keyref>
31   </xs:element>
32 </xs:schema>
```

XSchema Foreign Keys

- Given a class of elements, field values within the class must match field values within the referenced key or unique.
- Class: sequence of elements.
- Class is defined by a selector.
- Field: subelement/attribute of the last element on the selector path.

To-Do

- Go over examples 11.18–11.20.
- Go over exercises 11.4.3.

Online Examples

XML

http://www.w3schools.com/XML/xml_examples.asp

DTD

http://www.w3schools.com/dtd/dtd_examples.asp

http://www.w3schools.com/dtd/dtd_el_vs_attr.asp

http://www.xmlvalidation.com/

XSchema

http://www.w3schools.com/Schema

http://www.datypic.com/books/defxmlschema/

Outline

XPath Data Model

- The XPath data model is a sequence of items.
- Items in a sequence need not be of the same type.
- An item is either an atomic value or a node.
- Atomic values: string, boolean, decimal, date, time, etc.
- Node values: document, element, attribute, text, etc.
  - Document, element, and attribute nodes encapsulate respective XML entities.
  - Text nodes encapsulate XML character content.
- Every XPath query refers to a document.
- Construction of a document node from an XML document:
  `doc`(*doc_uri*)
  where the document URI is provided in double quotes.
- This node represents the XML document itself, not the root node.

Path Expressions

- The simplest XPath query expressions have the form:
  $/\mathtt{T}_1/\mathtt{T}_2/\cdots/\mathtt{T}_n$
  where each $\mathtt{T}_i$ is a tag.
- Evaluation starts with a sequence of one node: the document node.
- Each $\mathtt{T}_i$ is evaluated in turn, starting with $\mathtt{T}_1$.
- To evaluate $\mathtt{T}_i$, consider the sequence $\mathcal{S}$ of items obtained from processing all previous tags.
- Examine the items of $\mathcal{S}$ in order and, for each one, find all subelements whose tag is $\mathtt{T}_i$ and append them to the output sequence, in document order.

Path Expressions

- An XPath query expressions that retrieves attributes:
  $/\mathtt{T}_1/\mathtt{T}_2/\cdots/\mathtt{T}_n/@\mathtt{A}$
  where each $\mathtt{T}_i$ is a tag and $\mathtt{A}$ is an attribute of $\mathtt{T}_n$.
- To evaluate the expression, first compute the expression $/\mathtt{T}_1/\mathtt{T}_2/\cdots/\mathtt{T}_n/$.
- For each element in the resulting sequence, if attribute $\mathtt{A}$ exists in its opening tag, its value is appended to the output sequence.

To-Do

- Discuss examples 12.1–12.3.

XPath Axes

- XPath provides a rich set of axes, or modes of navigation.
- We have seen two axes: child (default) and attribute ("@").
- We can prefix a tag or attribute with an axis name as follows:
- $/\texttt{child}::\texttt{T}_1/\texttt{child}::\texttt{T}_2/\cdots/\texttt{child}::\texttt{T}_n$ is equivalent to $/\texttt{T}_1/\texttt{T}_2/\cdots/\texttt{T}_n$.
- $/\texttt{child}::\texttt{T}_1/\texttt{child}::\texttt{T}_2/\cdots/\texttt{child}::\texttt{T}_n/\texttt{attribute}::\texttt{A}$ is equivalent to $/\texttt{T}_1/\texttt{T}_2/\cdots/\texttt{T}_n/@\texttt{A}$ is the same as
- Other axes: parent (".."), ancestor (proper), descendant (proper), next-sibling (to the right), previous-sibling (to the left), self ("."), descendant-or-self ("//").
- Semantics assumes that elements in the result of an XPath expression are references to elements in the actual document.

XPath Axes

Axis	Result
ancestor	Selects all ancestors (parent, grandparent, etc.) of the current node
ancestor-or-self	Selects all ancestors (parent, grandparent, etc.) of the current node and the current node itself
attribute	Selects all attributes of the current node
child	Selects all children of the current node
descendant	Selects all descendants (children, grandchildren, etc.) of the current node
descendant-or-self	Selects all descendants (children, grandchildren, etc.) of the current node and the current node itself
following	Selects everything in the document after the closing tag of the current node
following-sibling	Selects all siblings after the current node
namespace	Selects all namespace nodes of the current node
parent	Selects the parent of the current node
preceding	Selects everything in the document that is before the start tag of the current node
preceding-sibling	Selects all siblings before the current node
self	Selects the current node

Other Expressions

- Wildcards: “*” for any tag or attribute.
- Conditional navigation: boolean expressions within square brackets.
  - Operands may be path expressions.
  - Path expression operands have existential semantics.
  - Integer [`i`]: true only for the i-th child of the parent.
  - Tag [`T`]: true only for elements having one or more subelements with tag `T`.
  - Tag [`A`]: true only for elements having a value for attribute `A`.

To-Do

- Go over examples 12.4–12.8.
- Go over exercises 12.1.1–12.1.2.

Outline

XPath Queries

Select the root element AAA.

```
doc('abc.xml')/AAA
```

or

```
doc('abc.xml')/child::AAA
```

```
<AAA>
  <BBB></BBB>
  <CCC>Joe</CCC>
  <BBB></BBB>
  <BBB></BBB>
  <DDD>
    <BBB></BBB>
  </DDD>
  <CCC>Jane</CCC>
</AAA>
```

XPath Queries

Select CCC elements that are children of the root element AAA.

```
doc('abc.xml')/AAA/CCC
```

or

```
doc('abc.xml')/child::AAA/child::CCC
```

```
<AAA>
  <BBB></BBB>
  <CCC>Joe</CCC>
  <BBB></BBB>
  <BBB></BBB>
  <DDD>
    <BBB></BBB>
  </DDD>
  <CCC>Jane</CCC>
</AAA>
```

XPath Queries

Select BBB elements that are children of DDD elements that are children of the root element AAA.

```
doc('abc.xml')/AAA/DDD/BBB
```

or

```
doc('abc.xml')/AAA/child::DDD/BBB
```

```
<AAA>
  <BBB></BBB>
  <CCC>Joe</CCC>
  <BBB></BBB>
  <BBB></BBB>
  <DDD>
    <BBB></BBB>
  </DDD>
  <CCC>Jane</CCC>
</AAA>
```

XPath Queries

Select the text content of CCC elements.

```
doc('abc.xml')//CCC/text()
```

```
<AAA>
  <BBB></BBB>
  <CCC>Joe</CCC>
  <BBB></BBB>
  <BBB></BBB>
  <DDD>
    <BBB></BBB>
  </DDD>
  <CCC>Jane</CCC>
</AAA>
```

XPath Queries

Select the CCC elements having some text content.

```
doc('abc.xml')//CCC[text()]
```

```
<AAA>
  <BBB></BBB>
  <CCC>Joe</CCC>
  <BBB></BBB>
  <BBB></BBB>
  <DDD>
    <BBB></BBB>
  </DDD>
  <CCC>Jane</CCC>
</AAA>
```

XPath Queries

Select BBB elements anywhere.

```
doc('abc.xml')//BBB
```

or

```
doc('abc.xml')/descendant::BBB
```

```
<AAA>
 <BBB></BBB>
 <CCC></CCC>
 <BBB></BBB>
 <DDD>
  <BBB></BBB>
 </DDD>
 <CCC>
  <DDD>
   <FFF></FFF>
   <BBB></BBB>
   <FFF></FFF>
   <BBB></BBB>
   <FFF></FFF>
   <BBB></BBB>
  </DDD>
 </CCC>
</AAA>
```

XPath Queries

Select BBB elements that are children of some DDD element.

```
doc('abc.xml')//DDD/BBB
```

or

```
doc('abc.xml')/descendant-or-self::DDD/child::BBB
```

```
<AAA>
 <BBB></BBB>
 <CCC></CCC>
 <BBB></BBB>
 <DDD>
  <BBB></BBB>
 </DDD>
 <CCC>
  <DDD>
   <FFF></FFF>
   <BBB></BBB>
   <FFF></FFF>
   <BBB></BBB>
   <FFF></FFF>
   <BBB></BBB>
  </DDD>
 </CCC>
</AAA>
```

XPath Queries

Select all elements that enclosed by the path AAA/CCC/DDD.

```
doc('abc.xml')/AAA/CCC/DDD/*
```

```
<AAA>
  <BBB></BBB>
  <CCC></CCC>
  <BBB></BBB>
  <DDD>
    <BBB></BBB>
  </DDD>
  <CCC>
    <DDD>
      <FFF></FFF>
      <BBB></BBB>
      <FFF></FFF>
      <BBB></BBB>
      <FFF></FFF>
      <BBB></BBB>
    </DDD>
  </CCC>
</AAA>
```

XPath Queries

Select all elements.

```
doc('abc.xml')//*
```

```
<AAA>
  <BBB></BBB>
  <CCC></CCC>
  <BBB></BBB>
  <DDD>
    <BBB></BBB>
  </DDD>
  <CCC>
    <DDD>
      <FFF></FFF>
      <BBB></BBB>
      <FFF></FFF>
      <BBB></BBB>
      <FFF></FFF>
      <BBB></BBB>
    </DDD>
  </CCC>
</AAA>
```

XPath Queries

Select the first BBB child of the every DDD element.

```
doc('abc.xml')//DDD/BBB[1]
```

```
<AAA>
  <BBB></BBB>
  <CCC></CCC>
  <BBB></BBB>
  <DDD>
    <BBB></BBB>
  </DDD>
  <CCC>
    <DDD>
      <FFF></FFF>
      <BBB></BBB>
      <FFF></FFF>
      <BBB></BBB>
      <FFF></FFF>
      <BBB></BBB>
    </DDD>
  </CCC>
</AAA>
```

XPath Queries

Select the last BBB child of the every DDD element.

```
doc('abc.xml')//DDD/BBB[last()]
```

```
<AAA>
  <BBB></BBB>
  <CCC></CCC>
  <BBB></BBB>
  <DDD>
    <BBB></BBB>
  </DDD>
  <CCC>
    <DDD>
      <FFF></FFF>
      <BBB></BBB>
      <FFF></FFF>
      <BBB></BBB>
      <FFF></FFF>
      <BBB></BBB>
    </DDD>
  </CCC>
</AAA>
```

XPath Queries

Select all id attributes.

```
doc('abc.xml')//@id
```

```
<AAA>
  <BBB id="b1"></BBB>
  <CCC></CCC>
  <BBB id="b2"></BBB>
  <DDD>
    <BBB name="jane"></BBB>
  </DDD>
  <CCC>
    <DDD>
      <FFF id="f1"></FFF>
      <BBB id="b3"></BBB>
      <FFF id="f2"></FFF>
      <BBB name="joe"></BBB>
      <FFF></FFF>
      <BBB></BBB>
    </DDD>
  </CCC>
</AAA>
```

XPath Queries

Select BBB elements having an id attribute.

```
doc('abc.xml')//BBB[@id]
```

```
<AAA>
  <BBB id="b1"></BBB>
  <CCC></CCC>
  <BBB id="b2"></BBB>
  <DDD>
    <BBB name="jane"></BBB>
  </DDD>
  <CCC>
    <DDD>
      <FFF id="f1"></FFF>
      <BBB id="b3"></BBB>
      <FFF id="f2"></FFF>
      <BBB name="joe"></BBB>
      <FFF></FFF>
      <BBB></BBB>
    </DDD>
  </CCC>
</AAA>
```

XPath Queries

Select BBB elements having some attribute.

```
doc('abc.xml')//BBB[@*]
```

```
<AAA>
  <BBB id="b1"></BBB>
  <CCC></CCC>
  <BBB id="b2"></BBB>
  <DDD>
    <BBB name="jane"></BBB>
  </DDD>
  <CCC>
    <DDD>
      <FFF id="f1"></FFF>
      <BBB id="b3"></BBB>
      <FFF id="f2"></FFF>
      <BBB name="joe"></BBB>
      <FFF></FFF>
      <BBB></BBB>
    </DDD>
  </CCC>
</AAA>
```

XPath Queries

Select BBB elements without attributes.

```
doc('abc.xml')//BBB[not(@*)]
```

```
<AAA>
  <BBB id="b1"></BBB>
  <CCC></CCC>
  <BBB id="b2"></BBB>
  <DDD>
    <BBB name="jane"></BBB>
  </DDD>
  <CCC>
    <DDD>
      <FFF id="f1"></FFF>
      <BBB id="b3"></BBB>
      <FFF id="f2"></FFF>
      <BBB name="joe"></BBB>
      <FFF></FFF>
      <BBB></BBB>
    </DDD>
  </CCC>
</AAA>
```

XPath Queries

Select BBB elements a name attribute with value 'joe'.

```
doc('abc.xml')//BBB[@name='jane']
```

```
<AAA>
  <BBB id="b1"></BBB>
  <CCC></CCC>
  <BBB id="b2"></BBB>
  <DDD>
    <BBB name="jane"></BBB>
  </DDD>
  <CCC>
    <DDD>
      <FFF id="f1"></FFF>
      <BBB id="b3"></BBB>
      <FFF id="f2"></FFF>
      <BBB name="joe"></BBB>
      <FFF></FFF>
      <BBB></BBB>
    </DDD>
  </CCC>
</AAA>
```

XPath Queries

Select elements having three BBB children.

```
doc('abc.xml')//*[count(BBB) = 3]
```

```
<AAA>
  <BBB id="b1"></BBB>
  <CCC></CCC>
  <BBB id="b2"></BBB>
  <DDD>
    <BBB name="jane"></BBB>
  </DDD>
  <CCC>
    <DDD>
      <FFF id="f1"></FFF>
      <BBB id="b3"></BBB>
      <FFF id="f2"></FFF>
      <BBB name="joe"></BBB>
      <FFF></FFF>
      <BBB></BBB>
    </DDD>
  </CCC>
</AAA>
```

XPath Queries

Select elements having more than four children.

```
doc('abc.xml')//*[count(*) > 4]
```

```
<AAA>
  <BBB id="b1"></BBB>
  <CCC></CCC>
  <BBB id="b2"></BBB>
  <DDD>
    <BBB name="jane"></BBB>
  </DDD>
  <CCC>
    <DDD>
      <FFF id="f1"></FFF>
      <BBB id="b3"></BBB>
      <FFF id="f2"></FFF>
      <BBB name="joe"></BBB>
      <FFF></FFF>
      <BBB></BBB>
    </DDD>
  </CCC>
</AAA>
```

XPath Queries

Select BBB and FFF elements.

```
doc('abc.xml')//BBB | doc('abc.xml')//FFF
```

```
<AAA>
  <BBB id="b1"></BBB>
  <CCC></CCC>
  <BBB id="b2"></BBB>
  <DDD>
    <BBB name="jane"></BBB>
  </DDD>
  <CCC>
    <DDD>
      <FFF id="f1"></FFF>
      <BBB id="b3"></BBB>
      <FFF id="f2"></FFF>
      <BBB name="joe"></BBB>
      <FFF></FFF>
      <BBB></BBB>
    </DDD>
  </CCC>
</AAA>
```

XPath Queries

Select the parents of some BBB element.

```
doc('abc.xml')//BBB/..
```

or

```
doc('abc.xml')//BBB/parent::*
```

```
<AAA>
 <BBB id="b1"></BBB>
 <CCC></CCC>
 <BBB id="b2"></BBB>
 <DDD>
  <BBB name="jane"></BBB>
 </DDD>
 <CCC>
  <DDD>
   <FFF id="f1"></FFF>
   <BBB id="b3"></BBB>
   <FFF id="f2"></FFF>
   <BBB name="joe"></BBB>
   <FFF></FFF>
   <BBB></BBB>
  </DDD>
 </CCC>
</AAA>
```

XPath Queries

Select the ancestors of some BBB element.

```
doc('abc.xml')//BBB/ancestor::*
```

```
<AAA>
  <BBB id="b1"></BBB>
  <CCC></CCC>
  <BBB id="b2"></BBB>
  <DDD>
    <BBB name="jane"></BBB>
  </DDD>
  <CCC>
    <DDD>
      <FFF id="f1"></FFF>
      <BBB id="b3"></BBB>
      <FFF id="f2"></FFF>
      <BBB name="joe"></BBB>
      <FFF></FFF>
      <BBB></BBB>
    </DDD>
  </CCC>
</AAA>
```

XPath Queries

Select the siblings following BBB elements.

```
doc('abc.xml')//BBB/following-sibling::*
```

```
<AAA>
  <BBB id="b1"></BBB>
  <CCC></CCC>
  <BBB id="b2"></BBB>
  <DDD>
    <BBB name="jane"></BBB>
  </DDD>
  <CCC>
    <DDD>
      <FFF id="f1"></FFF>
      <BBB id="b3"></BBB>
      <FFF id="f2"></FFF>
      <BBB name="joe"></BBB>
      <FFF></FFF>
      <BBB></BBB>
    </DDD>
  </CCC>
</AAA>
```

XPath Queries

Select the siblings preceeding BBB elements.

```
doc('abc.xml')//BBB/preceeding-sibling::*
```

```
<AAA>
  <BBB id="b1"></BBB>
  <CCC></CCC>
  <BBB id="b2"></BBB>
  <DDD>
    <BBB name="jane"></BBB>
  </DDD>
  <CCC>
    <DDD>
      <FFF id="f1"></FFF>
      <BBB id="b3"></BBB>
      <FFF id="f2"></FFF>
      <BBB name="joe"></BBB>
      <FFF></FFF>
      <BBB></BBB>
    </DDD>
  </CCC>
</AAA>
```

XPath Queries

Select the elements following the BBB element named 'jane'.

```
doc('abc.xml')//BBB[@name='jane']/following::*
```

```
<AAA>
  <BBB id="b1"></BBB>
  <CCC></CCC>
  <BBB id="b2"></BBB>
  <DDD>
    <BBB name="jane"></BBB>
  </DDD>
  <CCC>
    <DDD>
      <FFF id="f1"></FFF>
      <BBB id="b3"></BBB>
      <FFF id="f2"></FFF>
      <BBB name="joe"></BBB>
      <FFF></FFF>
      <BBB></BBB>
    </DDD>
  </CCC>
</AAA>
```

XPath Queries

Select the elements preceeding the FFF element with id ’f2’.

```
doc(’abc.xml’)//FFF[@id=’f2’]/preceeding::*
```

```
<AAA>
  <BBB id="b1"></BBB>
  <CCC></CCC>
  <BBB id="b2"></BBB>
  <DDD>
    <BBB name="jane"></BBB>
  </DDD>
  <CCC>
    <DDD>
      <FFF id="f1"></FFF>
      <BBB id="b3"></BBB>
      <FFF id="f2"></FFF>
      <BBB name="joe"></BBB>
      <FFF></FFF>
      <BBB></BBB>
    </DDD>
  </CCC>
</AAA>
```

Outline

Overview

- Extension of XPath.
- Standard for higher-level XML applications.
- A functional language.
- The main query expression is the FLWOR expression.
- FLWOR is XQuery's analogous of SQL's SPJ expressions.

FLWOR Expressions

FLWORExpr ::= (ForClause | LetClause)+
WhereClause?
OrderByClause?
ReturnClause

- Starts with one or more FOR and/or LET clauses.
- An optional WHERE clause follows.
- An optional ORDER BY clause follows.
- Ends with one RETURN clause.

LET Clause

LetClause ::= **"let"** "$"VarName TypeDeclaration? ":=" ExprSingle
(", " "$"VarName TypeDeclaration? ":=" ExprSingle)*

- All LET variables start with a dolar sign.
- Multiple simultaneous assignments are supported.
- The result of ExprSingle is an ordered sequence of items.
- The entire sequence is bound to the LET variable.

FOR Clause

ForClause ::= **"for"** "$"VarName TypeDeclaration? PositionalVar? **"in"** ExprSingle
(", " "$"VarName TypeDeclaration? PositionalVar? **"in"** ExprSingle)*

- All FOR variables start with a dolar sign.
- Multiple simultaneous assignments are supported.
- The result of ExprSingle is an ordered sequence of items.
- Each items is bound to the FOR variable, in turn.

WHERE Clause

WhereClause ::= **"where"** ExprSingle

- Filters the item sequence, retaining some items and discarding others.

ORDER BY Clause

OrderByClause ::= (("**order" "by**") | ("**stable" "order" "by**")) OrderSpecList

- Used to reorder the item sequence.
- Keywords ascending and descending supported.

RETURN Clause

ReturnClause ::= **"return"** ExprSingle

- Evaluated once for every item in the sequence, after filtering by the WHERE clause.
- The final result is an ordered sequence containing the results of these evaluations.
- Heads Up! The RETURN clause is typically executed multiple times inside for loops.

Joins

- Define two or more variables.
- Enforce the join condition using the WHERE clause.
- Alternatively, use conditional path expressions.
- Heads Up! Comparing elements is analogous to comparing objects in a language such as Java– it is a reference comparison. You normally want to compare primitive values, for which you can use the `fn:data()` function (atomization).

Observations

- Boolean interpretation of XQuery expressions.
- `xs:false`: empty sequence, empty string, 0, NaN.
- `xs:true`: non-empty sequences, non-empty strings, all other numbers.
- Any text is permissible between tags or as attributes in XQuery expressions. To include an expression, surround it with curly braces.

Example

```
let $e := (<one />, <two />, <three />)
return <out>{$s}</out>
```

```
<out>
  <one />
  <two />
  <three />
</out>
```

Example

```
let $e := (<one />, <two />, <three />)
return <out>{$s}</out>
```

```
<out>
  <one />
</out>
<out>
  <two />
</out>
<out>
  <three />
</out>
```

Example

Variables and their scopes.

```
1 for $x in $w, $a in f($x)
2 let $y := g($a)
3 for $z in p($x, $y)
4 return q($x, $y, $z)
```

Example

Positional variables.

```
1 for $car at $i in ("Ford", "Chevy"),
2     $pet at $j in ("Cat", "Dog")
```

Binds variables as follows:

```
1 ($i = 1, $car = "Ford", $j = 1, $pet = "Cat")
2 ($i = 1, $car = "Ford", $j = 2, $pet = "Dog")
3 ($i = 2, $car = "Chevy", $j = 1, $pet = "Cat")
4 ($i = 2, $car = "Chevy", $j = 2, $pet = "Dog")
```

Example

Positional variables for sampling.

```
1 ...
2 let $avg := fn:avg(for $x at $i in $input
3                    where $i mod 100 = 0
4                    return $x)
5 ...
```

Example

```
1  for $d in fn:doc('depts.xml')/depts/deptno
2  let $e := fn:doc('emps.xml')/emps/emp[deptno = $d]
3  where fn:count($e) >= 10
4  order by fn:avg($e/salary) descending
5  return
6    <big−dept>
7      {
8      $d,
9      <headcount>{fn:count($e)}</headcount>,
10     <avgsal>{fn:avg($e/salary)}</avgsal>
11     }
12   </big−dept>
```

Example

```
1  for $d in fn:doc('depts.xml')/depts/deptno
2  let $e := fn:doc('emps.xml')/emps/emp[deptno = $d]
3  where fn:count($e) >= 10
4  order by fn:avg($e/salary) descending
5  return
6    <big−dept>
7      {
8      $d,
9      <headcount>{fn:count($e)}</headcount>,
10     <avgsal>{fn:avg($e/salary)}</avgsal>
11     }
12   </big−dept>
```

- From depts.xml, find all deptno elements.
- From emps.xml, find all emp elements in department $d.
- Discard sequences of emp elements with fewer than 10 elements.
- Return the department, number of employees, and their average salary.

To-Do

- Go over examples 12.9-13.

Conditional Expressions

IfExpr ::= **"if"** "(" Expr ")" **"then"** ExprSingle **"else"** ExprSingle

- The conditional expression above returns the first expression if the condition in parenthesis is true or the second expression otherwise.
- Heads Up! This is not a statement– XQuery is a functional language, so every expression must return a sequence of items. This means that the **else** part is mandatory. If you do not wish to return anything, return the empty sequence: ().

Examples

Using a comparison expression:

```
1 if ($widget1/unit−cost < $widget2/unit−cost)
2   then $widget1
3   else $widget2
```

Testing for existence:

```
1 if ($part/@discounted)
2   then $part/wholesale
3   else $part/retail
```

Quantification

QuantifiedExpr ::=

("**some**" | "**every**") "$"VarName TypeDeclaration?
"**in**" ExprSingle ("," "$"VarName TypeDeclaration? "**in**" ExprSingle)*
"**satisfies**" ExprSingle

- General form: `quantifier expression-list test-expression`.
- Quantifiers: **some**, **every**.
- Expression list: binds variables.
- Test expression (**some**): evaluates to true if at least one evaluation of the test expression returns true (false for zero bindings).
- Test expression (**every**): evaluates to true if every evaluation of the test expression returns true (true for zero bindings).

Examples

Test if every element has a particular attribute (regardless of values):

```
1 every $part in /parts/part satisfies $part/@discounted
```

Test if some element satisfies a condition:

```
1 some $emp in /emps/employee satisfies ($emp/bonus > 0.25 * $emp/salary)
```

Quantified tests over nine pairs of variable bindings:

```
1 some $x in (1, 2, 3), $y in (2, 3, 4) satisfies $x + $y = 4
2 every $x in (1, 2, 3), $y in (2, 3, 4) satisfies $x + $y = 4
```

Aggregation

- Special case of functions that take as input a sequence and output a scalar value.
- E.g., `fn:avg()`, `fn:count()`, `fn:max()`, `fn:min()`, and `fn:sum()`.
- No construct for grouping!
- If you need, you must group explicitly.
- fn:distinct-values comes to the rescue.
- It returns a sequence of distinct atomic values, not elements!.

Example

```
<bib>
  <book>
    <title>TCP/IP Illustrated</title>
    <author>Stevens</author>
    <publisher>Addison−Wesley</publisher>
  </book>
  <book>
    <title>Advanced Programming in the Unix Environment</title>
    <author>Stevens</author>
    <publisher>Addison−Wesley</publisher>
  </book>
  <book>
    <title>Data on the Web</title>
    <author>Abiteboul</author>
    <author>Buneman</author>
    <author>Suciu</author>
  </book>
</bib>
```

Example

```
1  <authlist>
2  {
3    for $a in fn:distinct−values($bib/book/author)
4    order by $a
5    return
6      <author>
7        <name> {$a} </name>
8        <books>
9        {
10         for $b in $bib/book[author = $a]
11         order by $b/title
12         return $b/title
13       }
14       </books>
15     </author>
16 }
17 </authlist>
```

Example

- fn:distinct-values eliminates duplicate values from a list of author nodes.
- For each author value, the query returns a name and a list of book titles.
- The name is the current author's name.
- The book titles list includes titles of all books written by the current author.
- The author list, and the lists of titles published by each author, are returned in alphabetic order.

Example

```
<authlist>
  <author>
    <name>Abiteboul</name>
    <books><title>Data on the Web</title></books>
  </author>
  <author>
    <name>Buneman</name>
    <books><title>Data on the Web</title></books>
  </author>
  <author>
    <name>Stevens</name>
    <books>
      <title>Advanced Programming in the Unix Environment</title>
      <title>TCP/IP Illustrated</title>
    </books>
  </author>
  <author>
    <name>Suciu</name>
    <books><title>Data on the Web</title></books>
  </author>
</authlist>
```

To-Do

- Go over examples 12.16-19.
- Go over section 12.2.