COSC 304
Introduction to Database Systems

XML Querying

Dr. Ramon Lawrence
University of British Columbia Okanagan
ramon.lawrence@ubc.ca

Querying XML

We will look at two standard query languages: XPath and XQuery.

XPath allows you to specify path expressions to navigate the tree-structured XML document.

XQuery is a full query language that uses XPath for path expressions.

Example DTD

```
<!DOCTYPE Depts [
  <!ELEMENT Depts (Dept+)>
  <!ELEMENT Dept (name, Emp*, budget?)>
        <!ATTLIST Dept dno ID #REQUIRED>
        <!ATTLIST Dept mgr IDREF #IMPLIED>
  <!ELEMENT budget (#PCDATA)>
  <!ELEMENT Emp (name)>
        <!ATTLIST Emp eno ID #REQUIRED>
  <!ELEMENT name (#PCDATA)>
]>
```

Example XML Document

```
<?xml version = "1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE Depts SYSTEM "dept.dtd">
<Depts>
  <Dept dno = "D1">
        <name>Management</name>
        <Emp eno="E7"><name>R. Davis</name></Emp>
        <Emp eno="E8"><name>J. Jones</name></Emp>
  </Dept>
  <Dept dno = "D2" mgr = "E7">
        <name>Consulting</name>
        <Emp eno="E6"><name>L. Chu</name></Emp>
        <Emp eno="E3"><name>A. Lee</name></Emp>
        <budget>350000</budget>
  </Dept>
</Depts>
```

Path Descriptions in XPath

XPath provides the ability to navigate through a document using path descriptors.

Path descriptors are sequences of tags separated by slashes /.

- If the descriptor begins with `/`, then the path starts at the root.
- If the descriptor begins with `//`, the path can start anywhere.
- You may also start the path by giving the document name such as `doc(depts.xml)/`.

A path descriptor denotes a sequence of nodes. These nodes may themselves contain other nodes (elements).

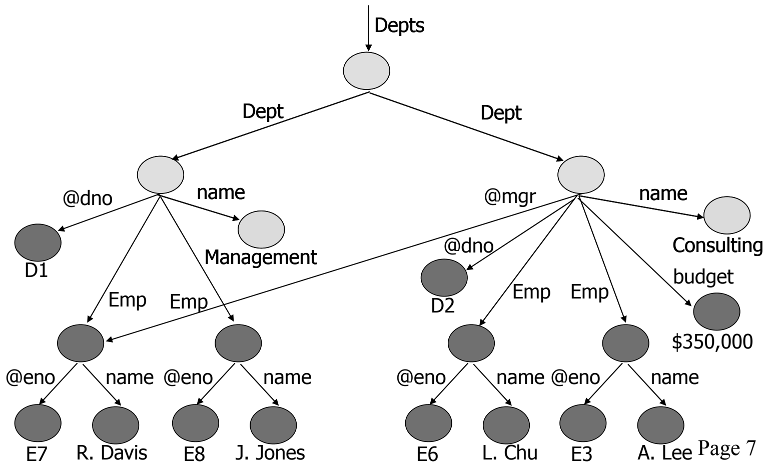
Examples:

- `/Depts/Dept/name`
- `//Dept/name`
- `doc("depts.xml")/Depts/Dept/Emp/name`

Path: /Depts/Dept/name

```
<?xml version = "1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE Depts SYSTEM "dept.dtd">
<Depts>
  <Dept dno = "D1">
        <name>Management</name>
        <Emp eno="E7"><name>R. Davis</name></Emp>
        <Emp eno="E8"><name>J. Jones</name></Emp>
  </Dept>
  <Dept dno = "D2" mgr = "E7">
        <name>Consulting</name>
        <Emp eno="E6"><name>L. Chu</name></Emp>
        <Emp eno="E3"><name>A. Lee</name></Emp>
        <budget>350000</budget>
  </Dept>
</Depts>
```

Path: /Depts/Dept/name (tree view)



Path: //Dept/name

```
<?xml version = "1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE Depts SYSTEM "dept.dtd">
<Depts>
  <Dept dno = "D1">
      <name>Management</name>
      <Emp eno="E7"><name>R. Davis</name></Emp>
      <Emp eno="E8"><name>J. Jones</name></Emp>
  </Dept>
  <Dept dno = "D2" mgr = "E7">
      <name>Consulting</name>
      <Emp eno="E6"><name>L. Chu</name></Emp>
      <Emp eno="E3"><name>A. Lee</name></Emp>
      <budget>350000</budget>
  </Dept>
</Depts>
```

Path query returns same answer as previous one.

Path: //name

```
<?xml version = "1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE Depts SYSTEM "dept.dtd">
<Depts>
  <Dept dno = "D1">
      <name>Management</name>
      <Emp eno="E7"><name>R. Davis</name></Emp>
      <Emp eno="E8"><name>J. Jones</name></Emp>
  </Dept>
  <Dept dno = "D2" mgr = "E7">
      <name>Consulting</name>
      <Emp eno="E6"><name>L. Chu</name></Emp>
      <Emp eno="E3"><name>A. Lee</name></Emp>
      <budget>350000</budget>
  </Dept>
</Depts>
```

Matches any name tag starting from anywhere in the document.

Path: /Depts/Dept

```
<?xml version = "1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE Depts SYSTEM "dept.dtd">
<Depts>
 <Dept dno = "D1">
      <name>Management</name>
      <Emp eno="E7"><name>R. Davis</name></Emp>
      <Emp eno="E8"><name>J. Jones</name></Emp>
 </Dept>
 <Dept dno = "D2" mgr = "E7">
      <name>Consulting</name>
      <Emp eno="E6"><name>L. Chu</name></Emp>
      <Emp eno="E3"><name>A. Lee</name></Emp>
      <budget>350000</budget>
 </Dept>
</Depts>
```

Wild Card Operator

The "*" wild card operator can be used to denote any ***single*** tag.

Examples:

- `/*/*/name` - Match any name that is nested 3 levels deep
- `//*` - Match anything

Path: /*/*/name

```
<?xml version = "1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE Depts SYSTEM "dept.dtd">
<Depts>
  <Dept dno = "D1">
      <name>Management</name>
      <Emp eno="E7"><name>R. Davis</name></Emp>
      <Emp eno="E8"><name>J. Jones</name></Emp>
  </Dept>
  <Dept dno = "D2" mgr = "E7">
      <name>Consulting</name>
      <Emp eno="E6"><name>L. Chu</name></Emp>
      <Emp eno="E3"><name>A. Lee</name></Emp>
      <budget>350000</budget>
  </Dept>
</Depts>
```

Same as /Depts/Dept/name

Question: What is /*/*/* ?

```
<?xml version = "1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE Depts SYSTEM "dept.dtd">
<Depts>
  <Dept dno = "D1">
       <name>Management</name>
       <Emp eno="E7"><name>R. Davis</name></Emp>
       <Emp eno="E8"><name>J. Jones</name></Emp>
  </Dept>
  <Dept dno = "D2" mgr = "E7">
       <name>Consulting</name>
       <Emp eno="E6"><name>L. Chu</name></Emp>
       <Emp eno="E3"><name>A. Lee</name></Emp>
       <budget>350000</budget>
  </Dept>
</Depts>
```

Attributes

Attributes are referenced by putting a "@" in front of their name.

Attributes of a tag may appear in paths as if they were nested within that tag.

Examples:

- `/Depts/Dept/@dno` - `dno` attribute of `Dept` element
- `//Emp/@eno` - `eno` attribute of `Emp` element

Path: /Depts/Dept/@dno

```
<?xml version = "1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE Depts SYSTEM "dept.dtd">
<Depts>
  <Dept dno = "D1">
       <name>Management</name>
       <Emp eno="E7"><name>R. Davis</name></Emp>
       <Emp eno="E8"><name>J. Jones</name></Emp>
  </Dept>
  <Dept dno = "D2" mgr = "E7">
       <name>Consulting</name>
       <Emp eno="E6"><name>L. Chu</name></Emp>
       <Emp eno="E3"><name>A. Lee</name></Emp>
       <budget>350000</budget>
  </Dept>
</Depts>
```

Question: What is /*/*/@eno ?

```
<?xml version = "1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE Depts SYSTEM "dept.dtd">
<Depts>
  <Dept dno = "D1">
       <name>Management</name>
       <Emp eno="E7"><name>R. Davis</name></Emp>
       <Emp eno="E8"><name>J. Jones</name></Emp>
  </Dept>
  <Dept dno = "D2" mgr = "E7">
       <name>Consulting</name>
       <Emp eno="E6"><name>L. Chu</name></Emp>
       <Emp eno="E3"><name>A. Lee</name></Emp>
       <budget>350000</budget>
  </Dept>
</Depts>
```

Predicate Expressions

The set of objects returned can be filtered by putting selection conditions on the path.

A ***predicate expression*** may be specified inside square brackets **[..]** following a tag. Only paths that have that tag and also satisfy the condition are included in the result of a path expression.

Examples:

- `/Depts/Dept/name[.="Management"]`
- `//Depts/Dept[budget>250000]`
- `//Emp[@eno="E5"]`

//Depts/Dept/budget[.>250000]

```
<?xml version = "1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE Depts SYSTEM "dept.dtd">
<Depts>
  <Dept dno = "D1">
       <name>Management</name>
       <Emp eno="E7"><name>R. Davis</name></Emp>
       <Emp eno="E8"><name>J. Jones</name></Emp>
  </Dept>
  <Dept dno = "D2" mgr = "E7">
       <name>Consulting</name>
       <Emp eno="E6"><name>L. Chu</name></Emp>
       <Emp eno="E3"><name>A. Lee</name></Emp>
       <budget>350000</budget>
  </Dept>
</Depts>
```

Note no budget element in first Dept so does not match path.

Axes

Path expressions allow us to start at the root and execute a sequence of steps to find a set of nodes at each step. So far, we were always starting at a context node, and traversing edges to children nodes.

However, XPath defines several different axes that allow us to go from the current node to other nodes. An ***axis*** specifies the tree relationship between the nodes selected by the location step and the current node.

There are multiple different axes such as `parent::`, `child::`, `ancestor::`, and `descendant::` among others. All these define a set of nodes with the given relationship with the current node.

XPath and Axes

Thus, evaluating an XPath expression amounts to starting with current nodes (called ***context nodes***) and then moving through the node hierarchy in a particular direction called an axis.

XPath evaluation description:

- When evaluating a path expression, the nodes selected in each step become the context nodes for the following step.
- If the input to a step is several context nodes, each is evaluated in turn. Evaluating a step involves:
    - Enumerating outgoing edges with matching labels AND
    - Only keeping destination nodes if they satisfy any predicate expression
- The result is output in the order of evaluation.

Axes and Abbreviations

An axis to traverse is specified by putting the axis name before the tag name to be matched such as `child::Dept`.

Since this often results in long queries, some common axes have abbreviations:

- The default axis is `child::` which contains all children of a context node. Since it is the default, the child axis does not have to be explicitly specified.
    - Thus, `/Depts/Dept` is shorthand for `/Depts/child::Dept`
- @ is a shorthand for the `attribute::` axis.
    - `/Depts/Dept/@dno` is short for `/Depts/Dept/attribute::dno`
- `..` is short for the `parent::` axis.
- `.` is short for the `self::` axis (current node).
- `//` is short for `descendant-or-self::` axis
    - // matches any node or any of its descendants

Summary of XPath Constructs

Symbol	Usage
/	Root element or separator between path steps
*	Match any single element name
@X	Match attribute X of current element
//	Match any descendant (or self) of current element
[C]	Evaluate condition on current element
[N]	Picks the *N*th matching element (indexed from 1)
parent::	Matches parent element
descendant::	Matches descendant elements
self::	Matches the current element
ancestor::	Matches ancestor elements
child::	Matches children elements
node()	Matches any node (regardless of label)

DTD for Questions

```
<!DOCTYPE Bookstore [
  <!ELEMENT Bookstore (Book | Magazine)*>
  <!ELEMENT Book (Title, Authors, Remark?)>
  <!ATTLIST Book ISBN CDATA #REQUIRED>
  <!ATTLIST Book Price CDATA #REQUIRED>
  <!ATTLIST Book Edition CDATA #IMPLIED>
  <!ELEMENT Magazine (Title)>
  <!ATTLIST Magazine Month CDATA #REQUIRED>
  <!ATTLIST Year CDATA #REQUIRED>
  <!ELEMENT Title (#PCDATA)>
  <!ELEMENT Authors (Author+)>
  <!ELEMENT Remark (#PCDATA)>
  <!ELEMENT Author (First_Name, Last_Name)>
  <!ELEMENT First_Name (#PCDATA)>
  <!ELEMENT Last_Name (#PCDATA)>
]
```

Example XML Document for Questions

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE Bookstore SYSTEM "bookstore.dtd">
<Bookstore>
   <Book ISBN="ISBN-0-201-70857-4" Price="65" Edition="3rd">
   <Title>Database Systems</Title>
   <Authors>
      <Author> <First_Name>Thomas</First_Name> <Last_Name>Connolly</Last_Name> </Author>
      <Author> <First_Name>Carolyn</First_Name> <Last_Name>Begg</Last_Name></Author>
   </Authors>
   </Book>
   <Book ISBN="ISBN-0-13-031995-3" Price="75">
   <Title>Database Systems: The Complete Book</Title>
   <Authors>
      <Author> <First_Name>Hector</First_Name><Last_Name>Garcia-Molina</Last_Name></Author>
      <Author> <First_Name>Jeffrey</First_Name> <Last_Name>Ullman</Last_Name> </Author>
      <Author> <First_Name>Jennifer</First_Name> <Last_Name>Widom</Last_Name> </Author>
   </Authors>
   <Remark> Amazon.com says: Buy these books together for a great deal!</Remark>
   </Book>
</Bookstore>
```

XPath Questions

What are the elements selected by these XPath queries:

- `/Bookstore/*/Title`
- `//First_Name[.="Thomas"]`
- `//Last_Name[.="Ullman"]/parent::node()/parent::node()/parent::node()[@Price < 60]`

Write XPath queries to retrieve:

- all book titles
- all books < $70
- all last names anywhere
- all books containing a remark
- all book titles where the book < $80 and Ullman is an author
- retrieve the second book

XQuery

XQuery allows querying XML documents, using path expressions from XPath to describe important sets.

FLWOR expressions (“for-let-where-order by-return") are similar to SQL and consist of:

- One or more `for` and/or `let` clauses (bind variables)
  - ⇨FOR clause iterates through bound variables, while LET does not.
- An optional `where` clause (filters bound tuples)
  - ⇨Evaluated for each set of bound variables (tuple)
- An optional `order by` clause
- A `return` clause (generates output)
  - ⇨Executed once for each set of bound variables (tuple)

Variables begin with a dollar sign "$".

XQuery `for` Clause Example

for loop variable

XPath expression to retrieve sequence of nodes to iterate over

```
for $en in /Depts/Dept/Emp/name
return <EmpName>{$en}</EmpName>
```

Brackets used to denote not regular text

```
Result:
<EmpName><name>R. Davis</name></EmpName>
<EmpName><name>J. Jones</name></EmpName>
<EmpName><name>L. Chu</name></EmpName>
<EmpName><name>A. Lee</name></EmpName>
```

XQuery `let` Clause Example

```
let $en := /Depts/Dept/Emp/name
return <EmpName>{$en}</EmpName>
```

```
Result:
<EmpName>
  <name>R. Davis</name>
  <name>J. Jones</name>
  <name>L. Chu</name>
  <name>A. Lee</name>
</EmpName>
```

XQuery WHERE Clause Example

```
for $e in /Depts/Dept/Emp
where $e/name > "I"
return <EmpName>{data($e/name)}</EmpName>
```

Returns data between tags instead of open/close tags and data together.

```
Result:
<EmpName>R. Davis</EmpName>
<EmpName>J. Jones</EmpName>
<EmpName>L. Chu</EmpName>
```

XQuery FOR/LET Clause Example

Return all departments with at least 2 employees.

```
<result>
{
for $d in /Depts/Dept
where count($d/Emp) >= 2
return <DeptName>{data($d/name)}</DeptName>
}
</result>

Result:
<result>
<DeptName>Management</DeptName>
<DeptName>Consulting</DeptName>
</result>
```

XQuery and IDREFs

In XQuery, it is possible to perform joins by using multiple XPath expressions.

A common case is to join an IDREF attribute with an ID attribute.

XQuery IDREF Example

Print the manager name for each department.

```
for $d in /Depts/Dept,
      $e in /Depts/Dept/Emp[@eno = $d/@mgr]
return
<DeptMgr>
  <DeptName>{data($d/name)}</DeptName>
  <MgrName>{data($e/name)}</MgrName>
</DeptMgr>

Result:
<DeptMgr>
  <DeptName>Consulting</DeptName>
  <MgrName>R. Davis</MgrName>
</DeptMgr>
```

XQuery Questions

Write XQuery queries to retrieve:

- 1) Return the book ISBN and price for each book.
- 2) Return only those books that have more than 2 authors.
- 3) Return average price of all books.
- 4) All titles of books costing < 80 where "Ullman" is an author.

Conclusion

XPath is a language for specifying paths through XML documents. An XPath expression enumerates a sequence.

XQuery is a full query language based on XPath. The basis of XQueries is the FLWR expression.

- XPath is used to enumerate sequences of nodes.
- FOR is used to iterate through nodes, LET to store the entire sequence.
- WHERE is used to filter bindings and the RESULT clause specifies the output result.

XPath and XQuery are standards defined by the W3C.

Objectives

- Given an XML document and query description, write an XPath query to retrieve the appropriate node sequence to answer the query.
- Given an XML document and an XPath expression, list the result of evaluating the expression.
- Be able to write simple XQuery queries given English text descriptions.