Lecture 6
XML/Xpath/XQuery

Wednesday, November 3, 2010
Announcements

Due to popular demand, take-home final dates have moved as follows:

• Posted: Friday, 10/10, 11:59pm
• Due: Sunday, 10/12, 11:59pm

HW 4 due next week

HW 5 posted today, due in two weeks
XML Outline

• XML
  – Syntax
  – Semistructured data
  – DTDs

• Xpath

• XQuery
Additional Readings on XML

- http://www.w3.org/XML/  
  – Main source on XML, but hard to read

- http://www.w3.org/TR/xquery/  
  – Authority on Xquery

Reading: textbook chapters 27.6, 27.7, 27.8
XML

• A flexible syntax for data
• Used in:
  – Configuration files, e.g. Web.Config
  – Replacement for binary formats (MS Word)
  – Document markup: e.g. XHTML
  – Data: data exchange, semistructured data
• Roots: SGML - a very nasty language

We will study only XML as data
XML as Semistructured Data

- Relational databases have rigid schema
  - Schema evolution is costly
- XML is flexible: semistructured data
  - Store data in XML
- Warning: not normal form! Not even 1NF
From HTML to XML

HTML describes the presentation
<h1> Bibliography </h1>
<p> <i> Foundations of Databases </i>
    Abiteboul, Hull, Vianu
    <br> Addison Wesley, 1995 </p>
<p> <i> Data on the Web </i>
    Abiteoul, Buneman, Suciu
    <br> Morgan Kaufmann, 1999 </p>
XML Syntax

<bibliography>
   <book>
      <title> Foundations… </title>
      <author> Abiteboul </author>
      <author> Hull </author>
      <author> Vianu </author>
      <publisher> Addison Wesley </publisher>
      <year> 1995 </year>
   </book>
   ...
</bibliography>

XML describes the content
XML Terminology

- tags: book, title, author, ...
- elements are nested
- empty element: <red></red> abbrv. <red/>
- an XML document: single root element

well formed XML document: if it has matching tags
More XML: Attributes

```xml
<book price="55" currency="USD">
  <title>Foundations of Databases</title>
  <author>Abiteboul</author>
  ...
  <year>1995</year>
</book>
```
Attributes v.s. Elements

attributes are alternative ways to represent data

<book price = “55” currency = “USD”>
  <title> Foundations of DBs </title>
  <author> Abiteboul </author>
  …
  <year> 1995 </year>
</book>

<book>
  <title> Foundations of DBs </title>
  <author> Abiteboul </author>
  …
  <year> 1995 </year>
  <price> 55 </price>
  <currency> USD </currency>
</book>
## Comparison

<table>
<thead>
<tr>
<th>Elements</th>
<th>Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ordered</td>
<td>Unordered</td>
</tr>
<tr>
<td>May be repeated</td>
<td>Must be unique</td>
</tr>
<tr>
<td>May be nested</td>
<td>Must be atomic</td>
</tr>
</tbody>
</table>
XML v.s. HTML

• What are the differences between XML and HTML?

In class
That’s All!

• That’s all you ever need to know about XML syntax
  – Optional type information can be given in the DTD or XSchema (later)
  – We’ll discuss some additional syntax in the next few slides, but that’s not essential

• What is important for you to know: XML’s semantics
More Syntax: Oids and References

oids and references in XML are just syntax

<person id="o555">
    <name> Jane </name>
</person>

Are just keys/ foreign keys design by someone who didn’t take 444

<person id="o456">
    <name> Mary </name>
    <mother idref="o555"/>
</person>

Don’t use them: use your own foreign keys instead.
More Syntax: CDATA Section

• Syntax: `<![CDATA[ .....any text here...]]>`

• Example:

```xml
<example>
  <![CDATA[ some text here </notAtag> <>]]>
</example>
```
More Syntax: Entity References

• Syntax: &entityname;

• Example:
  `<element> this is less than &lt;</element>`

• Some entities:

| &lt;   | <    |
| &gt;  | >    |
| &amp;  | &    |
| &apos; | ‘    |
| &quot; | “    |
| &#38;  | Unicode char |
More Syntax: Comments

• Syntax <!-- .... Comment text... -->

• Yes, they are part of the data model !!!
XML Namespaces

- name ::= [prefix:]localpart

```xml
</book>
```
<data>
  <person id="o555">
    <name>Mary</name>
    <address>
      <street>Maple</street>
      <no>345</no>
      <city>Seattle</city>
    </address>
  </person>
  <person>
    <name>John</name>
    <address>Thailand</address>
    <phone>23456</phone>
  </person>
</data>

Order matters !!!
XML as *Data*

- XML is self-describing
- Schema elements become part of the data
  - Relational schema: persons(name,phone)
  - In XML `<persons>, <name>, <phone>` are part of the data, and are repeated many times
- Consequence: XML is much more flexible
- XML = *semistructured* data
Mapping Relational Data to XML

The canonical mapping:

<table>
<thead>
<tr>
<th>Name</th>
<th>Phone</th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
<td>3634</td>
</tr>
<tr>
<td>Sue</td>
<td>6343</td>
</tr>
<tr>
<td>Dick</td>
<td>6363</td>
</tr>
</tbody>
</table>

XML:

```xml
<persons>
  <row>
    <name>John</name>
    <phone>3634</phone>
  </row>
  <row>
    <name>Sue</name>
    <phone>6343</phone>
  </row>
  <row>
    <name>Dick</name>
    <phone>6363</phone>
  </row>
</persons>
```
# Mapping Relational Data to XML

## Natural mapping

### Persons

<table>
<thead>
<tr>
<th>Name</th>
<th>Phone</th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
<td>3634</td>
</tr>
<tr>
<td>Sue</td>
<td>6343</td>
</tr>
</tbody>
</table>

### Orders

<table>
<thead>
<tr>
<th>PersonName</th>
<th>Date</th>
<th>Product</th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
<td>2002</td>
<td>Gizmo</td>
</tr>
<tr>
<td>John</td>
<td>2004</td>
<td>Gadget</td>
</tr>
<tr>
<td>Sue</td>
<td>2002</td>
<td>Gadget</td>
</tr>
</tbody>
</table>

## XML

```xml
<persons>
  <person>
    <name>John</name>
    <phone>3634</phone>
    <order>
      <date>2002</date>
      <product>Gizmo</product>
    </order>
  </person>
  <person>
    <name>Sue</name>
    <phone>6343</phone>
    <order>
      <date>2004</date>
      <product>Gadget</product>
    </order>
  </person>
</persons>
```
XML is Semi-structured Data

• Missing attributes:

```
<person>  <name> John</name>  
  <phone> 1234 </phone>
</person>

<person>  <name> Joe </name>
</person>
```

• Could represent in a table with nulls

<table>
<thead>
<tr>
<th>name</th>
<th>phone</th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
<td>1234</td>
</tr>
<tr>
<td>Joe</td>
<td>-</td>
</tr>
</tbody>
</table>
XML is Semi-structured Data

• Repeated attributes

```xml
<person>
  <name>Mary</name>
  <phone>2345</phone>
  <phone>3456</phone>
</person>
```

• Impossible in tables:

<table>
<thead>
<tr>
<th>name</th>
<th>phone</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mary</td>
<td>2345</td>
</tr>
<tr>
<td></td>
<td>3456</td>
</tr>
</tbody>
</table>

Two phones!
XML is Semi-structured Data

- Attributes with different types in different objects

```
<person>  <name>  <first> John </first>  
          <last> Smith </last> 
       </name>  
       <phone> 1234 </phone> 
</person>
```

- Nested collections (no 1NF)
- Heterogeneous collections:
  - `<db>` contains both `<book>`s and `<publisher>`s
Document Type Definitions

**DTD**

- part of the original XML specification
- an XML document may have a DTD
- XML document:
  - **Well-formed** = if tags are correctly closed
  - **Valid** = if it has a DTD and conforms to it
- validation is useful in data exchange
DTD

Goals:
• Define what tags and attributes are allowed
• Define how they are nested
• Define how they are ordered

Superseded by XML Schema
• Very complex: DTDs still used widely
Very Simple DTD

```xml
<!DOCTYPE company [ 
 <!ELEMENT company ((person|product)*)>
 <!ELEMENT person (ssn, name, office, phone?)>
 <!ELEMENT ssn (#PCDATA)>
 <!ELEMENT name (#PCDATA)>
 <!ELEMENT office (#PCDATA)>
 <!ELEMENT phone (#PCDATA)>
 <!ELEMENT product (pid, name, description?)>
 <!ELEMENT pid (#PCDATA)>
 <!ELEMENT description (#PCDATA)>
 ]>
```
Very Simple DTD

Example of valid XML document:

```xml
<company>
  <person>  
    <ssn> 123456789 </ssn>
    <name> John </name>
    <office> B432 </office>
    <phone> 1234 </phone>
  </person>
  <person>
    <ssn> 987654321 </ssn>
    <name> Jim </name>
    <office> B123 </office>
  </person>
  <product> ... </product>
  ...
</company>
```
DTD: The Content Model

- Content model:
  - Complex = a regular expression over other elements
  - Text-only = #PCDATA
  - Empty = EMPTY
  - Any = ANY
  - Mixed content = (#PCDATA | A | B | C)*
DTD: Regular Expressions

sequence
<!ELEMENT name (firstName, lastName)>  
optional
<!ELEMENT name (firstName?, lastName)>  
Kleene star
<!ELEMENT person (name, phone*)>  
alternation
<!ELEMENT person (name, (phone|email))>>
DTD: Regular Expressions

sequence

<!ELEMENT name (firstName, lastName)>  

optional

<!ELEMENT name (firstName?, lastName)>  

Kleene star

<!ELEMENT person (name, phone*)>  

alternation

<!ELEMENT person (name, (phone|email))>
**DTD: Regular Expressions**

**DTD**

- Sequence
  - `<!ELEMENT name (firstName, lastName)>`

- Optional
  - `<!ELEMENT name (firstName?, lastName)>`

- Kleene star
  - `<!ELEMENT person (name, phone*)>`

- Alternation
  - `<!ELEMENT person (name, (phone|email))>`

**XML**

- `<name>
  - `<firstName> . . . . . </firstName>`
  - `<lastName> . . . . . </lastName>`
  - `</name>`

- `<person>
  - `<name> . . . . . </name>`
  - `<phone> . . . . . </phone>`
  - `<phone> . . . . . </phone>`
  - `<phone> . . . . . </phone>`
  - `</person>`
Begin Optional Material
XSchema

• Generalizes DTDs

• Uses XML syntax

• Two parts: structure and datatypes

• Very complex
  – criticized
  – alternative proposals: Relax NG
DTD v.s. XML Schemas

DTD:

```xml
<!ELEMENT paper (title,author*,year, (journal|conference))>
```

XML Schema:

```xml
<xs:element name="paper" type="paperType"/>
<xs:complexType name="paperType">
  <xs:sequence>
    <xs:element name="title" type="xs:string"/>
    <xs:element name="author" minOccurs="0"/>
    <xs:element name="year"/>
    <xs:choice>
      <xs:element name="journal"/>
      <xs:element name="conference"/>
    </xs:choice>
  </xs:sequence>
</xs:element>
```
Example

A valid XML Document:

```xml
<paper>
  <title>The Essence of XML</title>
  <author>Simeon</author>
  <author>Wadler</author>
  <year>2003</year>
  <conference>POPL</conference>
</paper>
```
Elements v.s. Types

Both say the same thing; in DTD:

```xml
<!ELEMENT person (name, address)>```
• Types:
  – Simple types (integers, strings, ...)
  – Complex types (regular expressions, like in DTDs)

• Element-type Alternation:
  – An element has a type
  – A type is a regular expression of elements
Local v.s. Global Types

• Local type:
  
  `<xs:element name="person">`  
  [define locally the person’s type]
  `</xs:element>`

• Global type:
  
  `<xs:element name="person" type="ttt"/>`

  `<xs:complexType name="ttt">`  
  [define here the type ttt]
  `</xs:complexType>`

Global types: can be reused in other elements
Local v.s. Global Elements

- **Local element:**
  
  ```xml
  <xs:complexType name="ttt">
    <xs:sequence>
      <xs:element name="address" type="..."/>
    </xs:sequence>
  </xs:complexType>
  ```

- **Global element:**
  
  ```xml
  <xs:element name="address" type="..."/>
  <xs:complexType name="ttt">
    <xs:sequence>
      <xs:element ref="address"/>
    </xs:sequence>
  </xs:complexType>
  ```

**Global elements: like in DTDs**
Regular Expressions

Recall the element-type-element alternation:

```xml
<xs:complexType name="....">
  [regular expression on elements]
</xs:complexType>
```

Regular expressions:

- `<xs:sequence> A B C </...>`
  
  = A B C

- `<xs:choice> A B C </...>`

  = A | B | C

- `<xs:group> A B C </...>`

  = (A B C)

- `<xs:... minOccurs="0" maxOccurs="unbounded"> ..</...>`

  = (....)*

- `<xs:... minOccurs="0" maxOccurs="1"> ..</...>`

  = (....)?
Local Names

name has different meanings in person and in product
Subtle Use of Local Names

Arbitrary deep binary tree with A elements, and a single B element

Note: this example is not legal in XML Schema (why ?)
Hence they cannot express all regular tree languages
Attributes in XML Schema

<xs:element name="paper" type="papertype">
  <xs:complexType name="papertype">
    <xs:sequence>
      <xs:element name="title" type="xs:string"/>
      . . . . .
    </xs:sequence>
    <xs:attribute name="language" type="xs:NMTOKEN" fixed="English"/>
  </xs:complexType>
</xs:element>

Attributes are associated to the type, not to the element
Only to complex types; more trouble if we want to add attributes to simple types.
“Mixed” Content, “Any” Type

Better than in DTDs: can still enforce the type, but now may have text between any elements

Means anything is permitted there
“All” Group

A restricted form of & in SGML

Restrictions:
- Only at top level
- Has only elements
- Each element occurs at most once

E.g. “comment” occurs 0 or 1 times

```xml
<xs:complexType name="PurchaseOrderType">
  <xs:all>
    <xs:element name="shipTo" type="USAddress"/>
    <xs:element name="billTo" type="USAddress"/>
    <xs:element ref="comment" minOccurs="0"/>
    <xs:element name="items" type="Items"/>
  </xs:all>
  <xs:attribute name="orderDate" type="xs:date"/>
</xs:complexType>
```
Derived Types by Extensions

<complexType name="Address">
  <sequence>
    <element name="street" type="string"/>
    <element name="city" type="string"/>
  </sequence>
</complexType>

<complexType name="USAddress">
  <complexContent>
    <extension base="ipo:Address">
      <sequence>
        <element name="state" type="ipo:USState"/>
        <element name="zip" type="positiveInteger"/>
      </sequence>
    </extension>
  </complexContent>
</complexType>

Corresponds to inheritance
Derived Types by Restrictions

<complexType>
  <restriction base="ipo:Items">
    ... [rewrite the entire content, with restrictions]...
  </restriction>
</complexType>

- (*) : may restrict cardinalities, e.g. (0,infty) to (1,1); may restrict choices; other restrictions...

Corresponds to set inclusion
Simple Types

- String
- Token
- Byte
- unsignedByte
- Integer
- positiveInteger
- Int (larger than integer)
- unsignedInt
- Long
- Short
- ...

- Time
- dateTime
- Duration
- Date
- ID
- IDREF
- IDREFS
Facets of Simple Types

Facets = additional properties restricting a simple type

15 facets defined by XML Schema

Examples

- length
- minLength
- maxLength
- pattern
- enumeration
- whiteSpace

- maxInclusive
- maxExclusive
- minInclusive
- minExclusive
- totalDigits
- fractionDigits
Facets of Simple Types

• Can further restrict a simple type by changing some facets
• Restriction = subset
Not so Simple Types

• List types:

```xml
<xsl:complexType name="listOfMyIntType">
  <xs:list itemType="myInteger"/>
</xs:complexType>

<listOfMyInt>20003 15037 95977 95945</listOfMyInt>
```

• Union types

• Restriction types
End Optional Material
Typical XML Applications

- Data exchange
  - Take the data, don’t worry about schema
- Property lists
  - Many attributes, most are NULL
- Evolving schema
  - Add quickly a new attribute
Approaches to XML Processing

• Via API
  – Called DOM
  – Navigate, update the XML arbitrarily
  – BUT: memory bound

• Via some query language:
  – Xpath or Xquery
  – Stand-alone processor OR embedded in SQL
Querying XML Data

Will discuss next:

• XPath = simple navigation on the tree

• XQuery = “the SQL of XML”
Sample Data for Queries

<bib>
  <book>
    <publisher> Addison-Wesley </publisher>
    <author> Serge Abiteboul </author>
    <author> <first-name> Rick </first-name> <last-name> Hull </last-name>
    <author> Victor Vianu </author>
    <title> Foundations of Databases </title>
    <year> 1995 </year>
  </book>
  <book price="55">
    <publisher> Freeman </publisher>
    <author> Jeffrey D. Ullman </author>
    <title> Principles of Database and Knowledge Base Systems </title>
    <year> 1998 </year>
  </book>
</bib>
Data Model for XPath

The root element

The root

bib

book

book

publisher

canon

author

Addison-Wesley

Serge Abiteboul
XPath: Simple Expressions

/bib/book/year

Result:  
<year> 1995 </year>  
<year> 1998 </year>

/bib/paper/year

Result: empty  (there were no papers)

What’s the difference?
XPath: Restricted Kleene Closure

//author

Result: <author> Serge Abiteboul </author>

<author> <first-name> Rick </first-name>
  <last-name> Hull </last-name>
</author>

<author> Victor Vianu </author>
<author> Jeffrey D. Ullman </author>

Result:  <first-name> Rick </first-name>

/bib//first-name
Xpath: Attribute Nodes

/bib/book/@price

Result: should be “55”, but XPath can’t print (WHY ?)

@price means that price is an attribute
Xpath: Wildcard

//author/*

Result: <first-name> Rick </first-name>
       <last-name> Hull </last-name>

* Matches any element
@* Matches any attribute
Xpath: Text Nodes

Result:  Serge Abiteboul
        Victor Vianu
        Jeffrey D. Ullman

Rick Hull doesn’t appear because he has \texttt{firstname, lastname}

Functions in XPath:
- \texttt{text()} = matches the text value
- \texttt{node()} = matches any node (= * or @* or \texttt{text()})
- \texttt{name()} = returns the name of the current tag
Xpath: Predicates

\[ /\text{bib/book/author[first-name]} \]

Result: <author> <first-name> Rick </first-name> <last-name> Hull </last-name> </author>
Xpath: More Predicates

/bib/book/author[first-name][address[.//zip][city]]/last-name

Explain how this is evaluated!
Xpath: More Predicates

/bib/book/author[first-name][address[.//zip][city]]/last-name

Result: <lastname> … </lastname>
         <lastname> … </lastname>

How do we read this?
First remove all qualifiers (predicates):
/bib/book/author/last-name

Then add them one by one:
/bib/book/author[first-name][address]/last-name
Xpath: More Predicates

/bib/book[@price < 60]

/bib/book[author/@age < 25]

/bib/book[author/text()]
Xpath: More Axes

. means current node

/bib/book[./review]

/bib/book[./review]  Same as  /bib/book[review]

/bib/book/. /author  Same as  /bib/book/author
Xpath: More Axes

.. means parent node

/bib/book/author/../../author  

Same as  
/bib/book/author

/bib/book/author../../first-name/../../last-name

Same as  
/bib/book/author[first-name][last-name]
Xpath: Brief Summary

bib matches a bib element
*
matches any element
/
matches the root element
/bib
matches a bib element under root
bib/paper
matches a paper in bib
bib//paper
matches a paper in bib, at any depth
//paper
matches a paper at any depth
paper|book
matches a paper or a book
@price
matches a price attribute
bib/book/@price
matches price attribute in book, in bib
bib/book[@price<“55”]/author/lastname matches…
XQuery

• Based on Quilt, which is based on XML-QL

• Uses XPath to express more complex queries
FLWR ("Flower") Expressions

FOR ... 
LET...
WHERE...
RETURN...
FOR-WHERE-RETURN

Find all book titles published after 1995:

```xml
for $x$ in document("bib.xml")/bib/book
where $x$/year/text() > 1995
return $x$/title
```

Result:

```xml
<title> abc </title>  
<title> def </title>  
<title> ghi </title>
```
FOR-WHERE-RETURN

Equivalently (perhaps more geekish)

```
return $x$
```

And even shorter:

```
```
FOR-WHERE-RETURN

• Find all book titles and the year when they were published:

```xml
for $x in document("bib.xml")/bib/book
return <answer>
    <title> { $x/title/text() } </title>
    <year>{ $x/year/text() } </year>
</answer>
```

Result:

```xml
<answer> <title> abc </title> <year> 1995 </year> </answer>
<answer> <title> def </title> <year> 2002 </year> </answer>
<answer> <title> ghk </title> <year> 1980 </year> </answer>
```
FOR-WHERE-RETURN

• Notice the use of “{“ and “}”
• What is the result without them?

```xml
for $x$ in document("bib.xml")/bib/book
  return <answer>
    <title> $x/title/text() </title>
    <year> $x/year/text() </year>
  </answer>
```
FOR-WHERE-RETURN

• Notice the use of “{" and “}”
• What is the result without them?

```xml
for $x in document("bib.xml")/bib/book
return <answer>
    <title> $x/title/text() </title>
    <year> $x/year/text() </year>
</answer>
```

<answer> <title> $x/title/text() </title> <year> $x/year/text() </year> </answer>
<answer> <title> $x/title/text() </title> <year> $x/year/text() </year> </answer>
<answer> <title> $x/title/text() </title> <year> $x/year/text() </year> </answer>
<answer> <title> $x/title/text() </title> <year> $x/year/text() </year> </answer>
<answer> <title> $x/title/text() </title> <year> $x/year/text() </year> </answer>
Nesting

For each author of a book published in 1995, list all books she published:

```xml
for $b in document(“bib.xml”)/bib,
    $a in $b/book[year/text()=1995]/author
return <result>
    { $a,
      for $t in $b/book[author/text()=$a/text()]/title
        return $t
    }
</result>
```

In the **RETURN** clause comma concatenates XML fragments
Result

<result>
  <author>Jones</author>
  <title>abc</title>
  <title>def</title>
</result>
<result>
  <author>Smith</author>
  <title>ghi</title>
</result>
Aggregates

Find all books with more than 3 authors:

```xml
for $x$ in document("bib.xml")/bib/book
where count($x/author)>3
return $x
```

- `count` = a function that counts
- `avg` = computes the average
- `sum` = computes the sum
- `distinct-values` = eliminates duplicates
Aggregates

Same thing:

```latex
for $x$ in document("bib.xml")/bib/book[count(author)>3] return $x$
```
Aggregates

Print all authors who published more than 3 books

```xml
for $b$ in document("bib.xml")/bib,
   $a$ in distinct-values($b/book/author/text())
where count($b/book[author/text()=$a])>3
return <author> { $a } </author>
```
Flattening

• “Flatten” the authors, i.e. return a list of (author, title) pairs

```xml
for $b$ in document("bib.xml")/bib/book,
    $x$ in $b/title/text()$, 
    $y$ in $b/author/text()$
return <answer>
    <title> { $x$ } </title>
    <author> { $y$ } </author>
</answer>
```

Result:
```
<answer>
    <title> abc </title>
    <author> efg </author>
</answer>
<answer>
    <title> abc </title>
    <author> hkj </author>
</answer>
```


Re-grouping

• For each author, return all titles of her/his books

```xml
for $b$ in document("bib.xml")/bib
let $a$:distinct-values($b/book/author/text())
for $x$ in $a$
return

<answer>
  <author> { $x$ } </author>
  { for $y$ in $b/book[author/text()=$x]/title
    return $y } 
</answer>
```

Result:

```xml
<answer>
  <author> efg </author>
  <title> abc </title>
  <title> klm </title>
  ....
</answer>
```
Re-grouping

• Same thing:

```xml
for $b$ in document("bib.xml")/bib,
    $x$ in distinct-values($b/book/author/text())
return
    <answer>
        <author> { $x$ } </author>
        { for $y$ in $b/book[author/text()=$x]/title
            return $y$ }
    </answer>
```
SQL and XQuery Side-by-side

**Product(pid, name, maker, price)**  
Find all product names, prices, sort by price

```
SELECT x.name,  
    x.price  
FROM Product x  
ORDER BY x.price
```

SQL
SQL and XQuery Side-by-side

Product(pid, name, maker, price)  
Find all product names, prices, sort by price

**SQL**

```
SELECT x.name, x.price
FROM Product x
ORDER BY x.price
```

**XQuery**

```
for $x in document("db.xml")/db/product/row
order by $x/price/text()
return <answer>
    { $x/name, $x/price }
</answer>
```
Xquery’s Answer

<answer>
  <name> abc </name>
  <price> 7 </price>
</answer>
<answer>
  <name> def </name>
  <price> 23 </price>
</answer>

....
SQL and XQuery Side-by-side

Find all products made in Seattle

Product(pid, name, maker, price)
Company(cid, name, city, revenues)

SELECT x.name
FROM Product x, Company y
WHERE x.maker=y.cid
    and y.city="Seattle"
SQL and XQuery Side-by-side

Product(pid, name, maker, price)
Company(cid, name, city, revenues)

Find all products made in Seattle

SELECT x.name
FROM Product x, Company y
WHERE x.maker=y.cid
and y.city="Seattle"

for $r in document("db.xml")/db,
   $x in $r/product/row,
   $y in $r/company/row
where
   $x/maker/text()=$y/cid/text()
and $y/city/text() = “seattle”
return { $x/name }
SQL and XQuery Side-by-side

Product(pid, name, maker, price)
Company(cid, name, city, revenues)

Find all products made in Seattle

SQL

```
SELECT x.name
FROM Product x, Company y
WHERE x.maker=y.cid
  and y.city="Seattle"
```

XQuery

```
for $r in document("db.xml")/db,
  $x in $r/product/row,
  $y in $r/company/row
where
  $x/maker/text()=$y/cid/text()
  and $y/city/text() = "seattle"
return { $x/name }
```
<product>
  <row>
    <pid>123</pid>
    <name>abc</name>
    <maker>efg</maker>
  </row>
  <row>....</row>
  ...
</product>

...
For each company with revenues < 1M, count how many products with price > $100 they make

```
SELECT y.name, count(*)
FROM Product x, Company y
WHERE x.price > 100 and x.maker=y.cid and y.revenue < 1000000
GROUP BY y.cid, y.name
```
Find companies with at least 30 products, and their avg price

**SQL**

```
SELECT y.name, avg(x.price)
FROM Product x, Company y
WHERE x.maker=y.cid
GROUP BY y.cid, y.name
HAVING count(*) > 30
```

**XQuery**

```
for $r in document("db.xml")/db,
  $y in $r/company/row
let $p := $r/product/row[maker/text()=$y/cid/text()]
where count($p) > 30
return
  <thecompany>
    <companynname> { $y/name/text() } </companynname>
    <avgprice> avg($p/price/text()) </avgprice>
  </thecompany>
```
FOR v.s. LET

FOR
• Binds node variables \(\rightarrow\) iteration

LET
• Binds collection variables \(\rightarrow\) one value
FOR v.s. LET

```
for $x$ in /bib/book
return <result> { $x$ } </result>
```

Returns:
```
<result> <book>... </book> </result>
<result> <book>... </book> </result>
<result> <book>... </book> </result>
...
```

```
let $x$ := /bib/book
return <result> { $x$ } </result>
```

Returns:
```
<result> <book>... </book> </result>
<book>... </book>
<book>... </book>
...
```
```
</result>
```
Summary of XQuery

• FOR-LET-WHERE-RETURN = FLWR

Diagram:

FOR/LET Clauses
  ↓
  List of tuples

WHERE Clause
  ↓
  List of tuples

RETURN Clause
  ↓
Instance of Xquery/XML data model
The Rest is Optional Material
XML in SQL Server

• Create tables with attributes of type XML

• Use Xquery in SQL queries

• Rest of the slides are from:
  Shankar Pal et al., *Indexing XML data stored in a relational database*, VLDB’2004
CREATE TABLE DOCS (  
  ID int primary key,  
  XDOC xml)  

SELECT ID, XDOC.query('for $s in /BOOK[@ISBN= "1-55860-438-3"]//SECTION  
  return <topic>{data($s/TITLE)} </topic>')  
FROM DOCS
XML Methods in SQL

- `Query()` = returns XML data type
- `Value()` = extracts scalar values
- `Exist()` = checks conditions on XML nodes
- `Nodes()` = returns a rowset of XML nodes that the Xquery expression evaluates to
Examples

• From here:
XML Type

CREATE TABLE docs ( pk INT PRIMARY KEY, xCol XML not null )
Inserting an XML Value

```
INSERT INTO docs VALUES (2,
'<doc id="123"> 
  <sections>
  <section num="1"><title>XML Schema</title></section>
  <section num="3"><title>Benefits</title></section>
  <section num="4"><title>Features</title></section>
  </sections>
</doc>
')
```
Query()

```
SELECT pk, xCol.query('//*[@id = 123]/section')
FROM   docs
```
Exists( )

```
SELECT xCol.query('/doc[@id = 123]/section')
FROM docs
WHERE xCol.exist ('/doc[@id = 123]') = 1
```
Value( )

SELECT xCol.value('data(/doc//section[@num = 3]/title)[1]', 'nvarchar(max)')
FROM docs
Nodes( )

SELECT nref.value('first-name[1]', 'nvarchar(50)')
    AS FirstName,
    nref.value('last-name[1]', 'nvarchar(50)')
    AS LastName
FROM   @xVar.nodes('//author') AS R(nref)
WHERE  nref.exist('.[first-name != "David"]') = 1
SELECT nref.value('(@genre, 'varchar(max)')') LastName
FROM docs CROSS APPLY xCol.nodes('//book') AS R(nref)
Internal Storage

• XML is “shredded” as a table
• A few important ideas:
  – Dewey decimal numbering of nodes; store in clustered B-tree indices
  – Use only odd numbers to allow insertions
  – Reverse PATH-ID encoding, for efficient processing of postfix expressions like //a/b/c
  – Add more indexes, e.g. on data values
<BOOK ISBN="1-55860-438-3">
  <SECTION>
    <TITLE>Bad Bugs</TITLE>
    Nobody loves bad bugs.
    <FIGURE CAPTION="Sample bug"/>
  </SECTION>
  <SECTION>
    <TITLE>Tree Frogs</TITLE>
    All right-thinking people <BOLD>love</BOLD> tree frogs.
  </SECTION>
</BOOK>
<table>
<thead>
<tr>
<th>ORDPATH</th>
<th>TAG</th>
<th>NODE_TYPE</th>
<th>VALUE</th>
<th>PATH_ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1 (BOOK)</td>
<td>1 (Element)</td>
<td>Null</td>
<td>#1</td>
</tr>
<tr>
<td>1.1</td>
<td>2 (ISBN)</td>
<td>2 (Attribute)</td>
<td>'1-55860-438-3'</td>
<td>#2#1</td>
</tr>
<tr>
<td>1.3</td>
<td>3 (SECTION)</td>
<td>1 (Element)</td>
<td>Null</td>
<td>#3#1</td>
</tr>
<tr>
<td>1.3.1</td>
<td>4 (TITLE)</td>
<td>1 (Element)</td>
<td>'Bad Bugs'</td>
<td>#4#3#1</td>
</tr>
<tr>
<td>1.3.3</td>
<td>10 (TEXT)</td>
<td>4 (Value)</td>
<td>'Nobody loves Bad bugs.'</td>
<td>#10#3#1</td>
</tr>
<tr>
<td>1.3.5</td>
<td>5 (FIGURE)</td>
<td>1 (Element)</td>
<td>Null</td>
<td>#5#3#1</td>
</tr>
<tr>
<td>1.3.5.1</td>
<td>6 (CAPTION)</td>
<td>2 (Attribute)</td>
<td>'Sample bug'</td>
<td>#6#3#1</td>
</tr>
<tr>
<td>1.5</td>
<td>3 (SECTION)</td>
<td>1 (Element)</td>
<td>Null</td>
<td>#3#1</td>
</tr>
<tr>
<td>1.5.1</td>
<td>4 (TITLE)</td>
<td>1 (Element)</td>
<td>'Tree frogs'</td>
<td>#4#3#1</td>
</tr>
<tr>
<td>1.5.3</td>
<td>10 (TEXT)</td>
<td>4 (Value)</td>
<td>'All right-thinking people'</td>
<td>#10#3#1</td>
</tr>
<tr>
<td>1.5.5</td>
<td>7 (BOLD)</td>
<td>1 (Element)</td>
<td>'love '</td>
<td>#7#3#1</td>
</tr>
<tr>
<td>1.5.7</td>
<td>10 (TEXT)</td>
<td>4 (Value)</td>
<td>'tree frogs'</td>
<td>#10#3#1</td>
</tr>
</tbody>
</table>

Infoset Table
SELECT SerializeXML (N2.ID, N2.ORDPATH)
FROM infosettab N1  JOIN infosettab N2 ON (N1.ID = N2.ID)
WHERE N1.PATH_ID = PATH_ID(/BOOK/@ISBN)
    AND N1.VALUE = '1-55860-438-3'
    AND N2.PATH_ID = PATH_ID(BOOK/SECTION)
    AND Parent (N1.ORDPATH) = Parent (N2.ORDPATH)