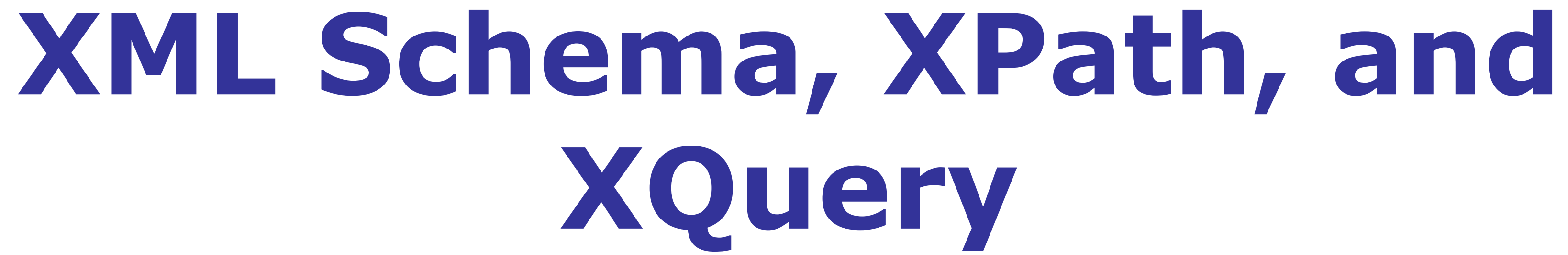


Juliana Freire

- some slides by David Koop, 2007
- some material taken from http://www.w3.org/TR/xmlschema-0/
- some slides from Zachary G. Ives, 2005, http://www.seas.upenn.edu/~zives/cis550/

XML Example

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<dblp>
 <mastersthesis mdate="2002-01-03" key="ms/Brown92">
  <author>Kurt P. Brown</author>
  <title>PRPL: A Database Workload Specification Language</title>
  <year>1992</year>
  <school>Univ. of Wisconsin-Madison</school>
 </mastersthesis>
 <article mdate="2002-01-03" key="tr/dec/SRC1997-018">
  <editor>Paul R. McJones</editor>
  <title>The 1995 SQL Reunion</title>
  <journal>Digital System Research Center Report</journal>
  <volume>SRC1997-018</volume>
  <year>1997</year>
  <ee>db/labs/dec/SRC1997-018.html</ee>
  <ee>http://www.mcjones.org/System_R/SQL_Reunion_95/</ee>
 </article>
```

Why XML?

- XML is the confluence of several factors:
  - The Web needed a more declarative format for data
  - Documents needed a mechanism for extended tags
  - Database people needed a more flexible interchange format
  - "Lingua franca" of data
  - It's a text file; edit with any text editor!
  - It's parsable even if we don't know what it means!
- Original expectation:
  - The whole web would go to XML instead of HTML
- Today's reality:
  - Not so… But XML is used all over "under the covers"

Why DB People Like XML

- Can get data from all sorts of sources
  - Allows us to touch data we don't own!
  - This was actually a huge change in the DB community
- Interesting relationships with DB techniques
  - Useful to do relational-style operations
  - Leverages ideas from object-oriented, semistructured data
- Blends schema and data into one format
  - Unlike relational model, where we need schema first
  - … But too little schema can be a drawback, too!

XML Anatomy

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<dblp>
  <mastersthesis mdate="2002-01-03" key="ms/Brown92">
    <author>Kurt P. Brown</author>
    <title>PRPL: A Database Workload Specification Language</title>
    <year>1992</year>
    <school>Univ. of Wisconsin-Madison</school>
  </mastersthesis>
  <article mdate="2002-01-03" key="tr/dec/SRC1997-018">
    <editor>Paul R. McJones</editor>
    <title>The 1995 SQL Reunion</title>
    <journal>Digital System Research Center Report</journal>
    <volume>SRC1997-018</volume>
    <year>1997</year>
    <ee>db/labs/dec/SRC1997-018.html</ee>
    <ee>http://www.mcjones.org/System_R/SQL_Reunion_95/</ee>
  </article>
```

Processing Instr.

Open-tag

Element

Attribute

Close-tag

Well-Formed XML

- A legal XML document – fully parsable by an XML parser
  - All open-tags have matching close-tags (unlike so many HTML documents!), or a special:

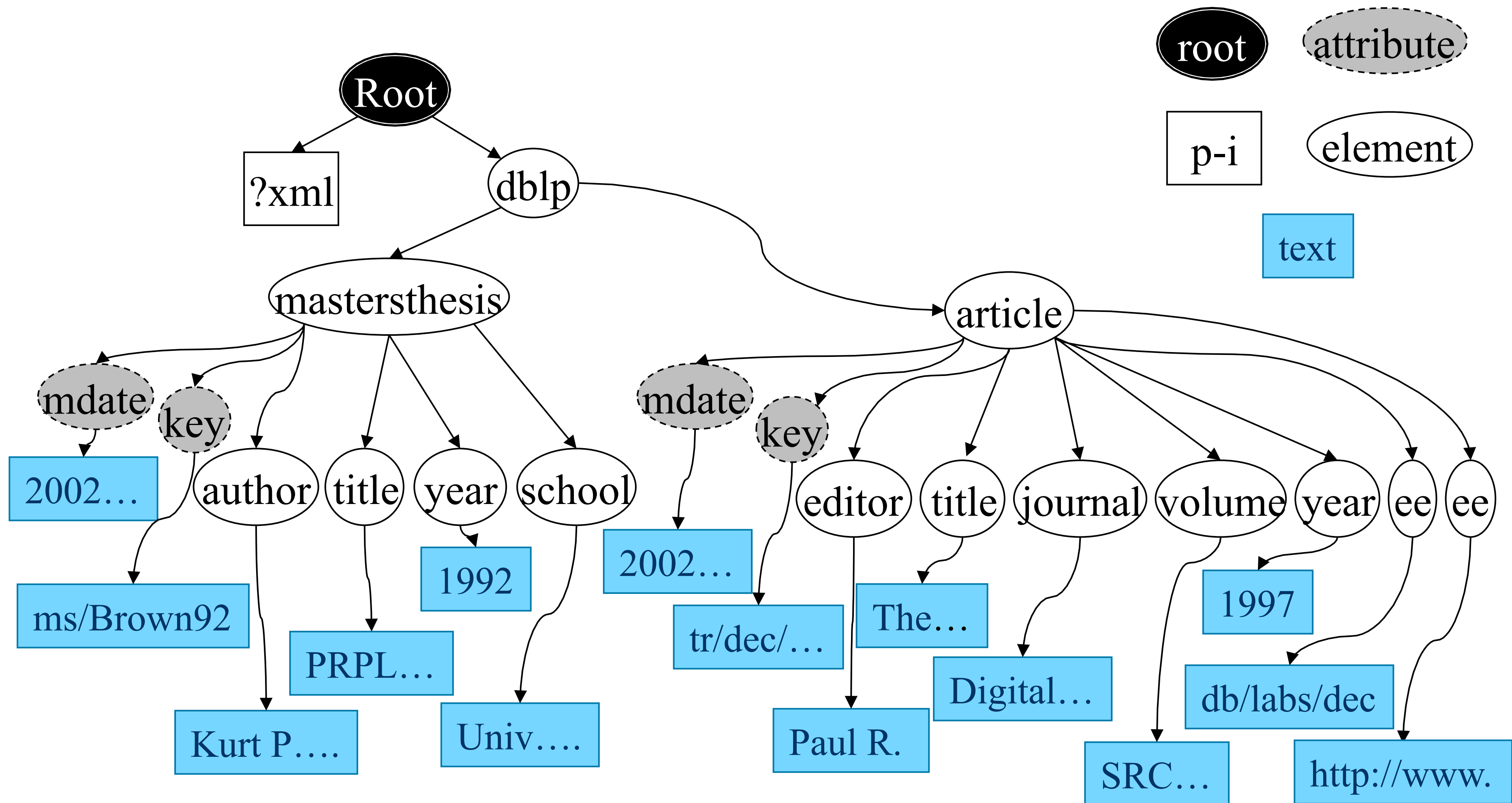
    <tag/> shortcut for empty tags

    (equivalent to <tag></tag>)
  - Attributes (which are unordered, in contrast to elements) only appear once in an element
  - There's a single root element
  - XML is case-sensitive

XML as a Data Model

- XML "information set" includes 7 types of nodes:
  - Document (root)
  - Element
  - Attribute
  - Processing instruction
  - Text (content)
  - Namespace
  - Comment
- XML data model includes this, plus typing info, plus order info and a few other things

XML Data Model Visualized



Multiple Sources with Same Tags

- *Namespaces* allow us to specify a context for different tags
- Two parts:
  - Binding of namespace to URI
  - Qualified names

```
<root xmlns="http://www.first.com/aspace"
   xmlns:otherns="…">
 <tag xmlns:myns="http://www.fictitious.com/mypath">
   <thistag>is in the default namespace (aspace)</thistag>
   <myns:thistag>is in myns</myns:thistag>
   <otherns:thistag>is a different tag in otherns</
   otherns:thistag>
 </tag>
</root>
```

XML Isn't Enough on Its Own

It's too unconstrained for many cases!

- How will we know when we're getting garbage?
- How will we query?
- How will we understand what we got?

We also need:

- Some idea of the structure
  - XMLSchema
- Query Language
  - XPath, XQuery
- Application-specific dialects
- Presentation, in some cases
  - XSLT

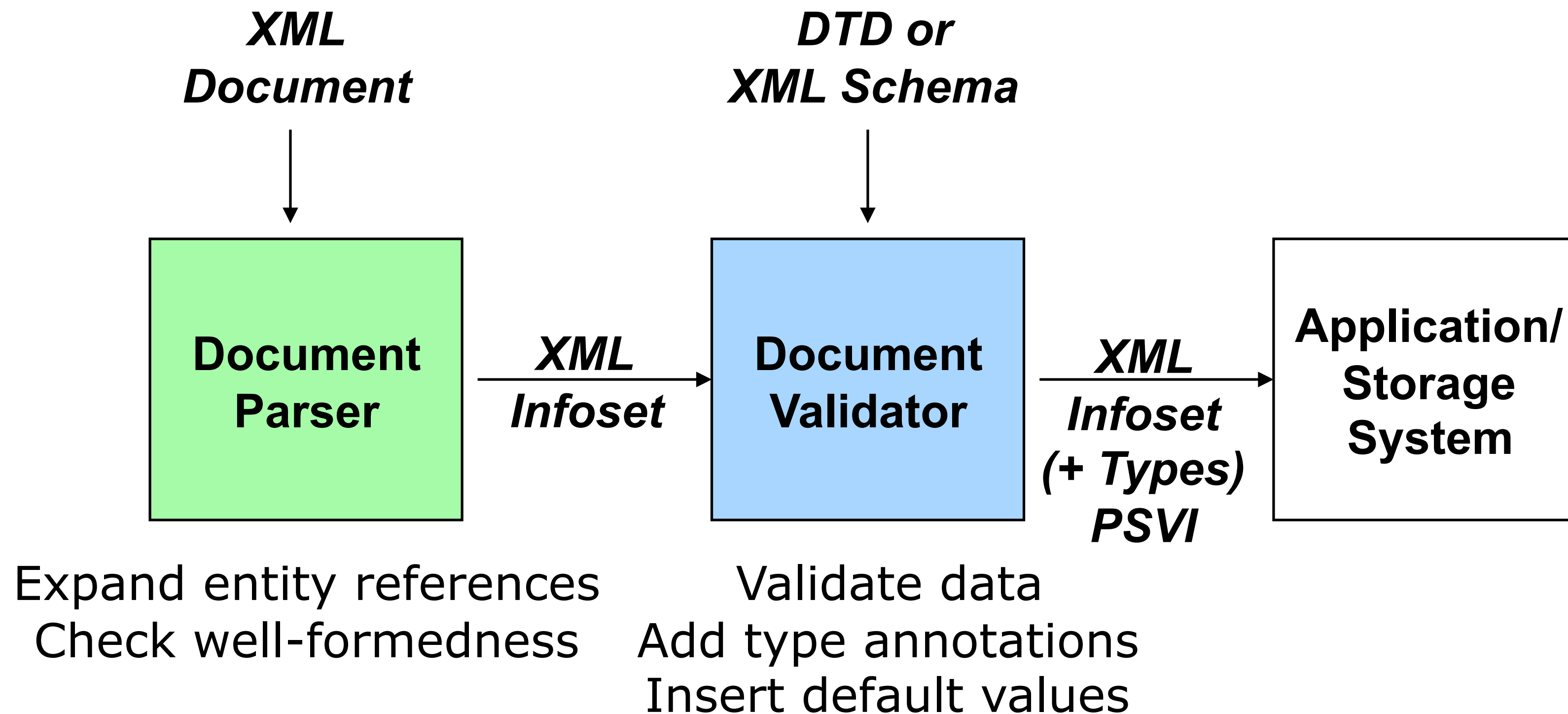
XML APIs and Relational Analogues

XML	Relational
XSLT, XQuery, XPath	*SQL*
DOM API / SAX API	*JDBC/ODBC*
XML Schema	*Relational Schema / SQL*
XPath Data Model/ XML Infoset	*Relational Data Model*
XML Document	*Relational Database*

Generic XML Processing Model

- XML Information Set

  per-character, per-entity model of XML document



Parsing

- XML Document » XML Information Set
- Checks well-formedness

  <person><initials>I.L.</person></initials>

- Doesn’t check that information conforms to any structural rules

```
<person>
  <person name="Joe">
    <cat><price>Fluffy</price></cat>
  </person>
</person>
```

- Doesn’t check that data matches expected type

  <price year="Nine Hundred">seventy cents</price>

Navigational Access: DOM

- Language-independent, programmatic API
- Application requirements
  - Full navigational access to document
  - Dynamic update, add, & delete document content

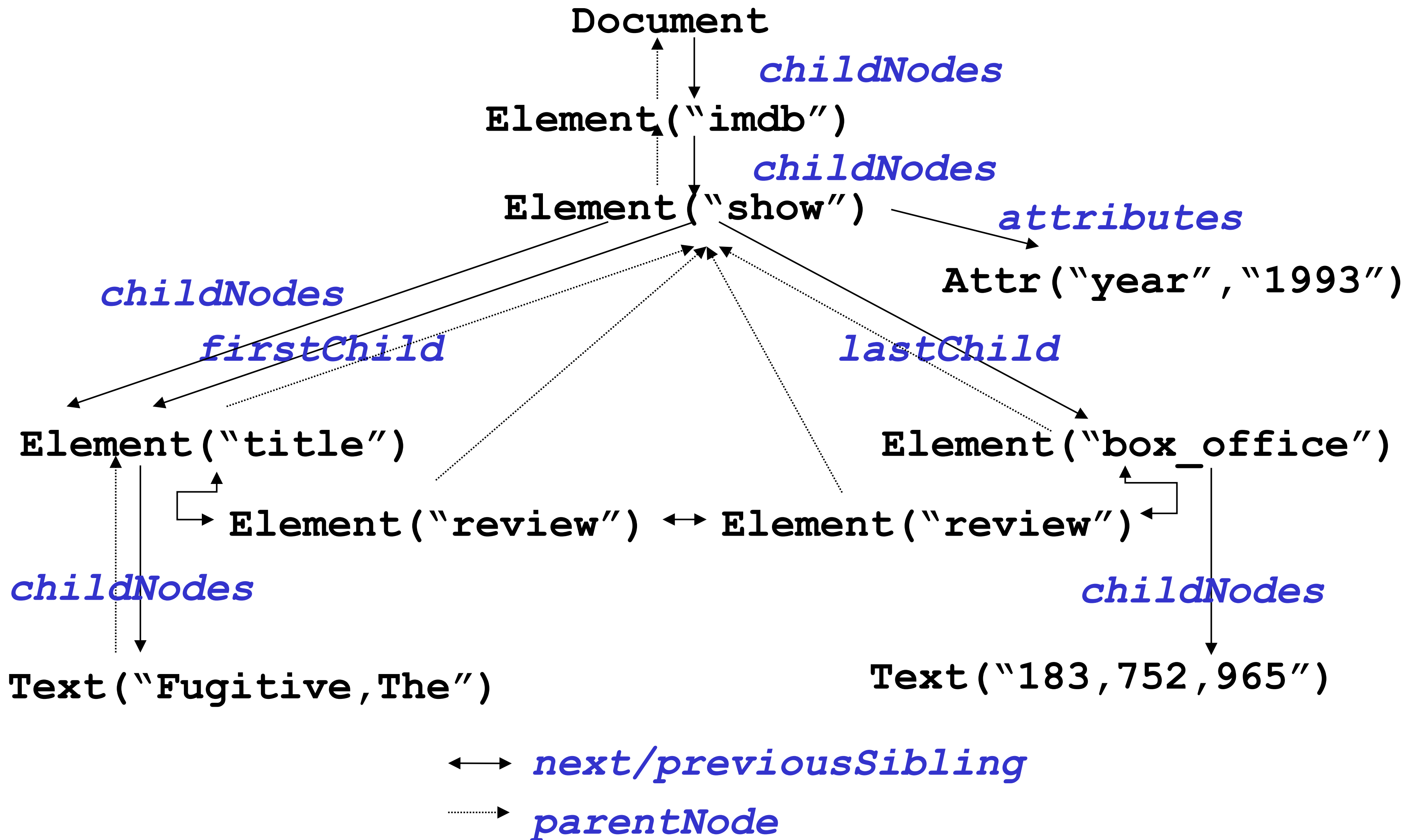
  Ex: Client-side browser apps; Plumbing of Dynamic HTML

- Query Access

  Ex: Reviews of shows with title "Fugitive, The" in IMDB

```
for s in documentElement.getElementsByTagName
  ("show")
    if (s.getAttribute("title") = "Fugitive, The")
    then s.getElementsByName("review")
```

DOM Example



Stream Access : SAX

- Language-independent, programmatic API
- Stream of elements, attributes, text

  Call-backs into application triggered by start/end tags
- Applications
  - Content-based routing of XML messages

    Ex: filter stock quotes, network alerts, …
  - Read-once processing of large documents

    Ex: load XML document into storage system
- Read-only access
  - No update-in-place -- Stream transformation

Validation

- XML Info Set + XML Schema »
  Post-Schema Validation Info Set (PSVI)
- PSVI includes *type information*
- An Info Set *passes* validation if it conforms to the schema
- Checks for legal tag & attributes, proper nesting & ordering of tags, and proper types
- Why do we care?

  Query optimization, hand editing, storage, transferring between applications, mapping to programming languages

XML Schema

- Defines:
  - vocabulary (element and attribute names)
  - content model (relationships and structure)
  - data types
- Written in XML
- Often uses namespace abbreviated as xs or xsd
- Namespace declaration:

  <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

XML Schema Example

```
<?xml version="1.0"?>
<purchaseOrder orderDate="1999-10-20">
  <shipTo country="US">
    <name>Alice Smith</name>
    <street>123 Maple Street</street>
    <city>Mill Valley</city>
    <state>CA</state>
    <zip>90952</zip>
  </shipTo>
  <billTo country="US">
    <name>Robert Smith</name>
    <street>8 Oak Avenue</street>
    <city>Old Town</city>
    <state>PA</state>
    <zip>95819</zip>
  </billTo>
  <comment>Hurry, my lawn is going wild!
     </comment>
<items>
    <item partNum="872-AA">
      <product>Lawnmower</product>
      <quantity>1</quantity>
      <USPrice>148.95</USPrice>
      <comment>Confirm this is electric</
     comment>
    </item>
    <item partNum="926-AA">
      <product>Baby Monitor</product>
      <quantity>1</quantity>
      <USPrice>39.98</USPrice>
      <shipDate>1999-05-21</shipDate>
    </item>
  </items>
</purchaseOrder>
```

XML Schema Header

- Schema uses a namespace
- Annotations can be inlined into the schema for documentation
- Example:

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
 <xsd:annotation>
  <xsd:documentation xml:lang="en">
   Purchase order schema for Example.com.
   Copyright 2000 Example.com. All rights reserved.
  </xsd:documentation>
 </xsd:annotation>
```

XML Schema Types

- Simple and complex element types

  Simple: <shipDate>2007-10-16</shipDate>

  Complex:

```
<purchaseOrder orderDate="2007-10-15">
  <shipTo>…</shipTo>
  …
</purchaseOrder>
```

- An element with attributes is always complex
- Attributes are unordered
- Can *restrict* attribute or element values

XML Schema Simple Types

- XML Schema defines primitive types
  - Examples: string, boolean, int, boolean, date, anyType, anySimpleType
- anyType allows any type, anySimpleType allows any primitive type
- Examples:

XML: <comment>Hurry, my lawn is going wild!</comment>

Schema: <xsd:element name="comment" type="xsd:string"/>

XML: <shipDate>1999-05-21</shipDate>

Schema: <xsd:element name="shipDate" type="xsd:date"/>

XML Schema Complex Types

- XML Schema supports nested types
- Can choose to reference type definition or use an *anonymous* complex type
- Example:

XML:

```
<purchaseOrder orderDate="2007-10-15">
  <shipTo>…</shipTo>…
</purchaseOrder>
```

Schema (Reference):

```
<xsd:element name="purchaseOrder" type="PurchaseOrderType"/>
<xsd:complexType name="PurchaseOrderType">
  <xsd:sequence>
    <xsd:element name="shipTo" type="USAddress"/>
    ...
  </xsd:sequence>
  <xsd:attribute name='orderDate' type=xsd:date/>
</xsd:complexType>
```

XML Schema Complex Types

- XML Schema supports nested types
- Can choose to reference type definition or use an *anonymous* complex type
- Example:

XML:

```
<purchaseOrder orderDate="2007-10-15">
  <shipTo>…</shipTo>…
</purchaseOrder>
```

Schema (Anonymous):

```
<xsd:element name="purchaseOrder">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="shipTo" type="USAddress"/>
      ...
    </xsd:sequence>
    <xsd:attribute name='orderDate' type=xsd:date/>
  </xsd:complexType>
</xsd:element>
```

Number of Occurrences

- Number of times an element appears in a document: minOccurs and maxOccurs
- Default values:
  - minOccurs: 1
  - maxOccurs: 1
- <xsd:element name="comment" minOccurs="0"/>
- <xsd:element name="item" minOccurs="0" **maxOccurs="unbounded"**/>
- maxOccurs can be *unbounded,* allowing an unlimited number of those elements

XML Schema Restrictions

- Define restrictions for elements/attributes

```
<xsd:element name="quantity">
 <xsd:simpleType>
  <xsd:restriction base="xsd:positiveInteger">
   <xsd:maxExclusive value="100"/>
  </xsd:restriction>
 </xsd:simpleType>
</xsd:element>

<xsd:simpleType name="SKU">
  <xsd:restriction base="xsd:string">
   <xsd:pattern value="\d{3}-[A-Z]{2}"/>
  </xsd:restriction>
</xsd:simpleType>
```

XML Schema Restrictions

- We can even enumerate all possible values:

```
<xsd:simpleType name="USState">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="AK"/>
    <xsd:enumeration value="AL"/>
    <xsd:enumeration value="AR"/>
    <!-- and so on ... -->
  </xsd:restriction>
</xsd:simpleType>
```

XML Schema Grouping

- Order of nodes matters in XML
- Elements of a complex type definition inside
  <xsd:sequence>…</xsd:sequence>
  must appear in XML documents in that order
- If you don't care about order, use
  <xsd:all>…</xsd:all>
- If you want the schema to include one type of element from a given group, use
  <xsd:choice>…</xsd:choice>
  inside xsd:sequence or xsd:all

Example

```
<xsd:complexType name="PurchaseOrderType">
  <xsd:sequence>
    <xsd:choice>
      <xsd:group ref="shipAndBill"/>
      <xsd:element name="singleUSAddress" type="USAddress"/>
    </xsd:choice>
    <xsd:element ref="comment" minOccurs="0"/>
    <xsd:element name="items"  type="Items"/>
  </xsd:sequence>
  <xsd:attribute name="orderDate" type="xsd:date"/>
</xsd:complexType>

<xsd:group id="shipAndBill">
  <xsd:sequence>
    <xsd:element name="shipTo" type="USAddress"/>
    <xsd:element name="billTo" type="USAddress"/>
  </xsd:sequence>
</xsd:group>
```

IMDB Example : Data

```
<imdb>
 <show year="1993"> <!-- Example Movie -->
  <title>Fugitive, The</title>
  <review>
   <suntimes>
    <reviewer>Roger Ebert</reviewer> gives <rating>two thumbs
    up</rating>! A fun action movie, Harrison Ford at his best.
   </suntimes>
  </review>
  <review>
   <nyt>The standard Hollywood summer movie strikes back.</nyt>
  </review>
  <box_office>183,752,965</box_office>
 </show>
 <show year="1994"> <!-- Example Television Show -->
  <title>X Files,The</title>
  <seasons>4</seasons>
 </show> . . .
</imdb>
```

IMDB Example : Data

```
<imdb>
 <show year="1993"> <!-- Example Movie -->
  <title>Fugitive, The</title>
  <review>
   <suntimes>
    <reviewer>Roger Ebert</reviewer> gives <rating>two thumbs
    up</rating>! A fun action movie, Harrison Ford at his best.
   </suntimes>
  </review>
  <review>
   <nyt>The standard Hollywood summer movie strikes back.</nyt>
  </review>
  <box_office>183,752,965</box_office>
 </show>
 <show year="1994"> <!-- Example Television Show -->
  <title>X Files,The</title>
  <seasons>4</seasons>
 </show>  . . .
</imdb>
```

IMDB Example : Schema

```
<element name="show">
 <complexType>
  <sequence>
    <element name="title" type="xs:string"/>
    <sequence minoccurs="0" maxoccurs="unbounded">
      <element name="review" mixed="true"/>
    </sequence>
    <choice>
      <element name="box_office" type="xs:integer"/>
      <element name="seasons" type="xs:integer"/>
    </choice>
  </sequence>
  <attribute name="year" type="xs:integer" use="optional"/>
 </complexType>
</element>
```

Common Querying Tasks

- Filter, select XML values
  - Navigation, selection, extraction
- Merge, integrate values from multiple XML sources
  - Joins, aggregation
- *Transform XML values from one schema to another*
  - *XML construction*

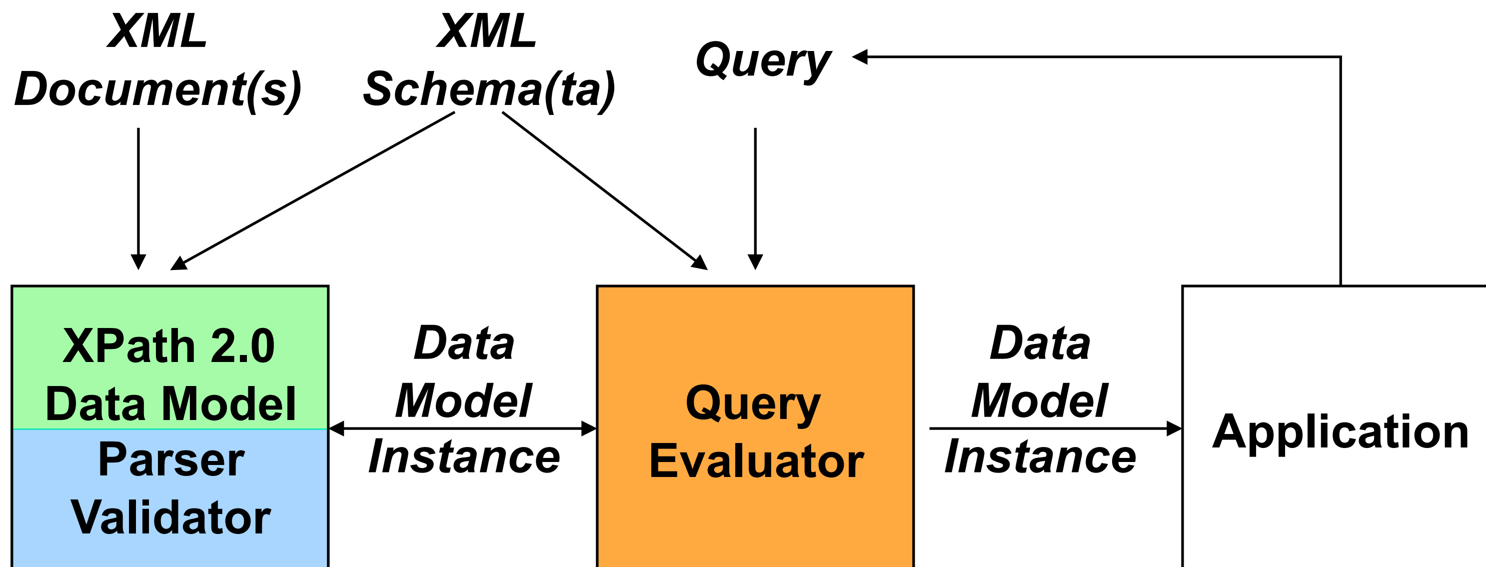
- Programmatic interfaces (DOM/SAX) specify how
- Query languages specify what, not how
  - Provide abstractions for common tasks
  - Easier than programmatic interfaces

Query Languages

- XPath 2.0
  - Common language for navigation, selection, extraction
  - Used in XSLT, XQuery, XPointer, XML Schema, XForms, et al
- XSLT 2.0: XML $\Rightarrow$ XML, HTML, Text
  - Loosely-typed scripting language
  - Format XML in HTML for display in browser
  - Must be highly tolerant of variability/errors in data
- XQuery 1.0: XML $\Rightarrow$ XML
  - Strongly-typed query language
  - Large-scale database access
  - Must guarantee safety/correctness of operations on data
- Over time, XSLT & XQuery may both serve needs of many application domains

Query Processing Model

- Other models possible



XPath

- Syntax for navigating XML
- Looks similar to file paths
- Used by XML Schema, XSLT, XQuery
- Searches by structure and text
- Guarantees same syntactic expression has same semantics
- Navigation, selection, value extraction
- Arithmetic, logical, comparison expressions

XPath

- In its simplest form, an XPath is like a path in a file system:

  /mypath/subpath/*/morepath

  - The XPath returns a *node set* representing the XML nodes (and their subtrees) at the end of the path
  - XPaths can have *node tests* at the end, returning only particular node types, e.g., text(), processing-instruction(), comment(), element(), attribute()
  - XPath is fundamentally an ordered language: it can query in order-aware fashion, and it returns nodes in order

XPath

- **XPath = sequence of location steps**
- A *location step* is:
  axis-name::*node-test*[*predicate*]
- Example: descendant::book[@title="XML"]
- axes: self, child, parent, descendant, ancestor, descendant-or-self, ancestor-or-self, following, preceding, following-sibling, preceding-sibling
- Steps are joined by forward slashes
- Example: root()/child::imdb/descendant-or-self::node()/child::title
- Many syntax shortcuts: /imdb//title

XPath Syntax

- */node-name* == /child::*node-name*
- Relative paths work as expected
  - /imdb == /imdb/show/title/../..
  - /imdb == /imdb/./././.
- // == descendant-or-self
- Predicate tests (filter node set)
  - [Inside brackets]
  - Prefix attributes by @
  - //show[title = "Seinfeld"] == //show[./title/text() = "Seinfeld"]
  - Standard comparisons:
    //show[@year > 2005]
  - Comparisons based on ordering:
    //surgery[//anesthesia[1] before //incision[1]]

XPath Functions

- Library of functions available
- Use fn namespace
- Ordering: fn::position, fn::first, fn::last
- String Operations: fn::substring, fn::starts-with, fn::matches
- Numeric Operations: fn::abs, fn::floor
- Many more:
  - http://www.w3.org/TR/xpath-functions/
  - http://www.w3schools.com/xpath/xpath_functions.asp

Variability in XML Data

- Problem: Replication or absence of XML values
  - Demands flexible semantics for selection
- Selection:

  //show[**year >= 2000**]

  Explicit expression:

  //show[**some $v in ./child::year satisfies data($v) ge 2000**]

  - matches all shows that contain *at least* one year child whose numeric content is greater than 2000
- Existence/absence of value:

  //show/reviewer[**following-sibling::rating**]

  Explicit expression:

  //show/reviewer[**not empty(./following-sibling::rating)**]

Variability in Schemas

- Documents may contain fragments with strongly typed values and un-typed text
- Demands flexible, but consistent semantics

```
<book isbn="ISBN 10-111">
  <price>45.50</price>
</book>
```

- For un-typed text, permissive correction from PCDATA to typed values

  /book/price * 0.07 SUCCEEDS!

- For typed values, strict interpretation of typed values and type error is fatal

  /book/@isbn * 0.07 FAILS!

Beyond XPath 2.0

- Limitations
  - Constructing new XML
  - Recursive processing of recursive XML data
- Differences between XSLT & XQuery
  - Safety: XQuery enforces input & output types
  - Compositionality:
    - XQuery maps XML to XML, XSLT maps XML to anything
    - Important feature for XML publishing

Supported by XSLT & XQuery

Remember closure?

XQuery 1.0

- Functional, strongly typed query language
- XQuery 1.0 = XPath 2.0 + …
  - A few more expressions
  - **FLWOR**
  - Sort-by
  - XML construction (Transformation)
  - Operators on types (Compile & run-time type tests)
  - **User-defined functions**
    - Modularize large queries
    - Process recursive data
  - **Strong typing**
    - Guarantees result value conforms to output type
    - Enforced statically or dynamically

XQuery FLWOR

- SQL:

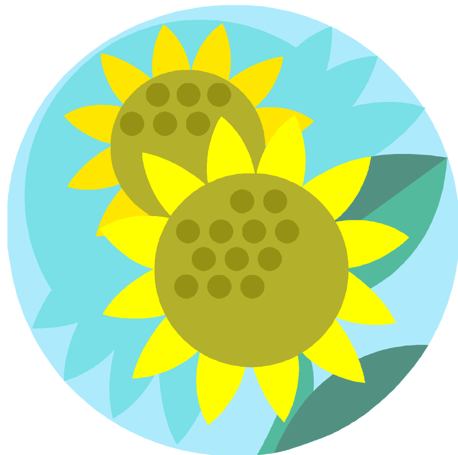
  SELECT <attribute list>

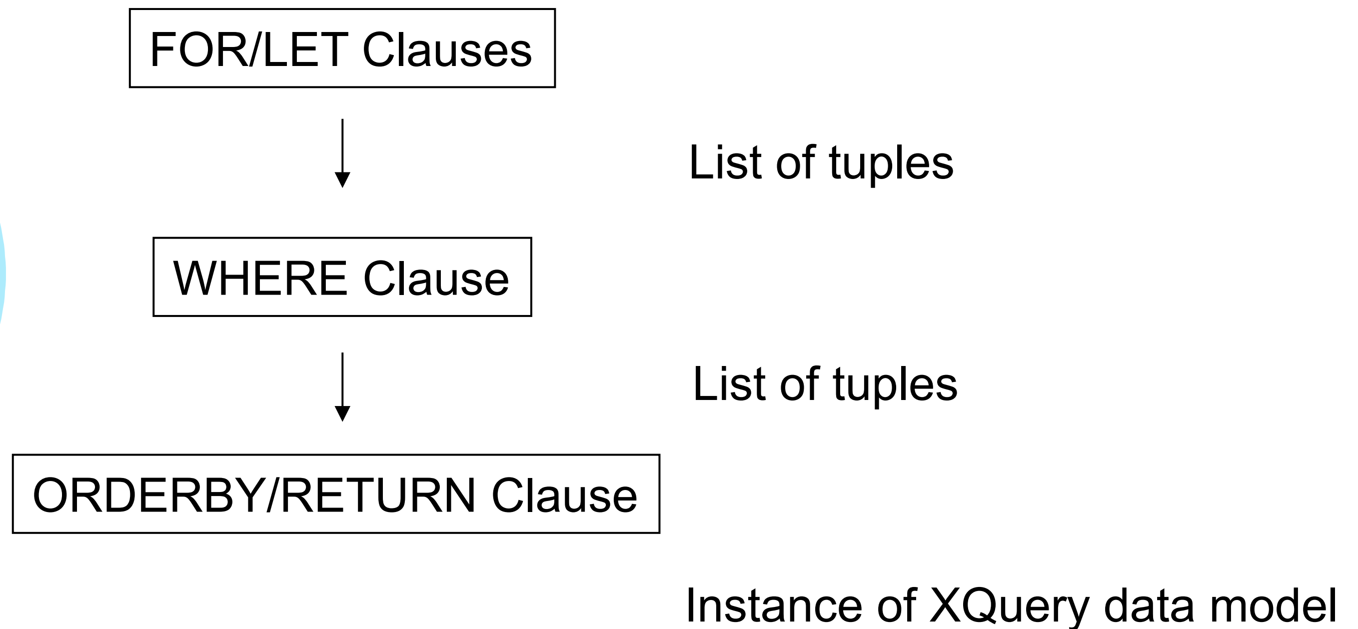
  FROM <set of tables>

  WHERE <set of conditions>

  ORDER BY <attribute list>

- XQuery: **F**OR-**L**ET-**W**HERE-**O**RDERBY-**R**ETURN





XQuery: Example

For each actor, return box office receipts of films in which they starred in past 2 years

```
let $imdb := document("www.imdb.com/imdb.xml")
for $actor in $imdb//actor
let $films :=
  $imdb//show[box_office and @year >= 2000
    and $actor/name = .//actor[@role="star"]/name]
return
 <receipts>
   { $actor }
   <total> { sum($films/box_office) } </total>
 </receipts>
```

Iteration

Join

XML Construction

Aggregation

XQuery

- FOR $x in expr -- binds $x to each value in the list expr

- LET $x := expr   -- binds $x to the entire list expr
  - Useful for common subexpressions and for aggregations

FOR vs. LET

Returns:

<result> <show>...</show></result>
<result> <show>...</show></result>
<result> <show>...</show></result>
...

```
FOR $x IN document("imdb.xml")//show
RETURN <result> $x </result>
```

```
LET $x := document("imdb.xml")//show
RETURN <result> $x </result>
```

Returns:

<result> <show>...</show>
<show>...</show>
<show>...</show>
...
</result>

Aggregates

Find movies whose box office proceeds are larger than average:

```
LET $a := avg(document("imdb.xml")//box_office)
FOR $s in document("imdb.xml")//show
WHERE $s//box_office > $a
RETURN $s
```

Collections in XQuery

- Ordered and unordered collections
  - /bib/book/author = an ordered collection
  - Distinct(/bib/book/author) = an unordered collection
- LET $s := /imdb/show → $s is a collection
- $s/title → a collection (several titles...)

```
RETURN <result> $s/title </result>
```

Returns:

```
<result> <title>...</title>
         <title>...</title>
         <title>...</title>
         ...
</result>
```

If-Then-Else

```
FOR $s IN //show
ORDERBY $s/year
RETURN <show>
          $s/title,
           IF $s/box_office
           THEN <movie> …</movie>
           ELSE <tv_show> … </tv_show>
       </show>
```

Existential Quantifiers

```
FOR $s IN //show
WHERE SOME $a IN $s/aka SATISFIES
    contains($a, "Term")
    OR contains($p, "T3")
RETURN $s/title
```

Universal Quantifiers

```
FOR $s IN //show
WHERE EVERY $a IN $s//aka SATISFIES
   contains($a, "Term")
RETURN $s/title
```

XML Transformation

- User-defined functions
  - Signatures specify types of arguments & return values
  - Types enforced statically or dynamically
  - Same expressiveness as XSLT templates + parameters

```
define function show2movie(element show $show)
                returns element movie?
{ // Convert a show (that is a movie) to a movie
  if ($show/box_office) then <movie> { $show/* } </movie>
  else ()
}
let $imdb := document("www.imdb.com/imdb.xml")
return <movies>
         for $show in $imdb/show return show2movie($show)
       </movies>
```

Recursive XML Data

- Recursive functions support recursive data

```
<Part id="001">
  <Part id="002">
    <Part id="003"/>
  </Part>
  <Part id="004"/>
</Part>
```

```
<PartCt count="2" id="001">
  <PartCt count="1" id="002"/>
    <PartCt count="0" id="003"/>
  </PartCt>
  <PartCt count="0" id="004"/>
</PartCt>
```

```
define function partCount(element Part $p1)
   returns element PartCt
{
   <PartCt count="{ count($p1/Part) }" { $p1/@id }>
   {
      for $p2 in $p1/Part return partCount($p2)
   }
   </PartCt>
}
```

Challenge Question

Are the following queries equivalent?

A.
```
FOR $show IN document("www.imdb.com/imdb.xml")//show,
      $review IN $show/review
WHERE
$show/@year >= 2002
RETURN
<show> <t>$show/title</t> <r>$review</r> </show>
```

B.
```
FOR $show IN document("www.imdb.com/imdb.xml")//show
 WHERE
$show/@year >= 2002
 RETURN
 <show> <t>$show/title</t> <r>$show/review</r> </show>
```

Safety

- Shared schema ($\mathbf{S}_{\mathbf{shared}}$) is ***contract*** between producers & consumers
- Producer writes query to transform input data into output data

$$\mathbf{D}_{\mathbf{input}} : \mathbf{S}_{\mathbf{input}} \Rightarrow \mathbf{Q}_{\mathbf{producer}} \Rightarrow \mathbf{D}_{\mathbf{output}} : \mathbf{S}_{\mathbf{output}}$$

- Static Type Checking takes $\mathbf{S}_{\mathbf{input}}$ & $\mathbf{Q}_{\mathbf{producer}}$
  - *Infers* $\mathbf{S}_{\mathbf{output}}$: schema of output data
  - *Checks* that $\mathbf{S}_{\mathbf{output}}$ is “subtype” of $\mathbf{S}_{\mathbf{shared}}$
  - *Guarantees* $\mathbf{D}_{\mathbf{output}}$: $\mathbf{S}_{\mathbf{shared}}$

XQuery vs XSLT

- XSLT is primarily a language for describing *XML transformation*; XQuery is primarily a language to *query XML data* and documents.
- XQuery: XML→ XML; XSLT: XML → {XML, HTML, text, …}
- XSLT uses XML-based syntax; XQuery 1.0 doesn't
- *XPath is at the core for both*, XSLT and XQuery.
- XSLT 1.0 turned W3C recommendation on November 16, 1999. XQuery 1.0 turned W3C recommendation on December 14, 2010. Many tools, APIs, and vendors have excellent support for XSLT. XQuery support is introduced by many vendors/toolkits; it is being rapidly improved.

XQuery vs XSLT

- XQuery 1.0 has a concept of user-defined functions, which can be modeled in XSLT 1.0 as named templates.
- XQuery 1.0 is strongly typed language, XSLT 1.0 is not.
- XQuery provides FLWOR expression for looping, sorting, filtering; XSLT 1.0's xsl:for-each instruction (and XSLT 2.0's for expression) allows to do the same.
- XQuery does not support all the XPath axes; XSLT does.

XQuery vs XSLT (cont.)

- **XQuery: Reinventing the Wheel?** http://www.xmlportfolio.com/xquery.html
- An interesting discussion: http://lists.xml.org/archives/xml-dev/200102/msg00483.html

Xquery vs. XSLT: Example

```
FOR $b IN document("bib.xml")//book
WHERE $b/publisher = "Morgan Kaufmann"
AND $b/year = "1998"
RETURN $b/title
```

```
<xsl:transform version="1.0" xmlns:xsl="http://www.w3.org/
1999/XSL/Transform">
  <xsl:template match="/">
    <xsl:for-each select="document('bib.xml')//book">
      <xsl:if test="publisher='Morgan Kaufmann' and year='1998'">
        <xsl:copy-of select="title"/>
      </xsl:if>
    </xsl:for-each>
  </xsl:template>
</xsl:transform>
```

Feature Summary

	XML Content		Update	Safety	
	What	How		Input	Output
DOM	Entity refs String data	Navigational	In-place Transform	Not Preserved	Not Enforced
SAX	Entity refs String data	Streams		Not Preserved	Not Enforced
XPath 2.0	Typed values	Declarative		Preserved	
XSLT 2.0	Typed values	Declarative	Transform	Preserved	Not Enforced
XQuery 1.0	Typed values	Declarative	Transform	Preserved	Enforced

Implementor's Perspective

- Interface : multiple implementation strategies

XSLT 2.0/XQuery 1.0
XPath 2.0

Custom Query engine

Translate into SQL/OQL/LDAP

XPath Data Model

Implement from scratch

Build on existing storage system

DOM API

SAX API

XML Parser

Special-purpose Streams Processor

XML Information Set
XML Document

References

- XML Use Cases: sample queries
  - http://www.w3.org/TR/xquery-use-cases/
- Galax: an XQuery engine
  - http://www.galaxquery.org/
- Xalan: an XPath + XSL engine
  - http://xml.apache.org/xalan-j/
- XPath tutorials:
  - http://www.w3schools.com/xpath/default.asp
  - http://www.zvon.org/xxl/XPathTutorial/General/examples.html
  - http://www.ibiblio.org/xml/books/xmljava/chapters/ch16.html
- XQuery:
  - http://www.brics.dk/~amoeller/XML/querying/
  - An Introduction to XQuery -- http://www.perfectxml.com/articles/xml/xquery.asp
  - XQuery Tutorial http://www.ipedo.com/html/xquery/xquery_tutorial/

References (cont.)

- DOM

  http://www.w3.org/TR/REC-DOM-Level-1/
- SAX

  http://www.saxproject.org/
- XPath 2.0

  http://www.w3.org/TR/query-datamodel/

  http://www.w3.org/TR/xpath20/

  http://www.w3.org/TR/query-operators/

  http://www.topxml.com/xpathvisualizer/
- XQuery 1.0

  http://www.w3.org/TR/xquery/

XQuery

A strongly-typed, Turing-complete XML manipulation language

- Attempts to do static type-checking against XML Schema
- Based on an object model derived from Schema

Unlike SQL, fully compositional, highly orthogonal:

- Inputs & outputs collections (sequences or bags) of XML nodes
- Anywhere a particular type of object may be used, may use the results of a query of the same type
- Designed mostly by DB and functional language people

Attempts to satisfy the needs of data management *and* document management

- The database-style core is mostly complete (even has support for NULLs in XML!!)
- The document keyword querying features are still in the works – shows in the order-preserving default model

XQuery's Basic Form

- Has an analogous form to SQL's SELECT..FROM..WHERE..GROUP BY..ORDER BY
- The model: bind nodes (or node sets) to variables; operate over each legal combination of bindings; produce a set of nodes
- "FLWOR" statement:

```
for {iterators that bind variables}
let {collections}
where {conditions}
order by {order-conditions}
return {output constructor}
```

"Iterations" in XQuery

A series of (possibly nested) FOR statements assigning the results of XPaths to variables

```
for $root in document("http://my.org/my.xml")
    for $sub in $root/rootElement,
            $sub2 in $sub/subElement, …
```

- Something like a template that pattern-matches, produces a "binding tuple"
- For each of these, we evaluate the WHERE and possibly output the RETURN template
- document() or doc() function specifies an input file as a URI
  - Old version was "document"; now "doc" but it depends on your XQuery implementation

Two XQuery Examples

```
<root-tag> {
   for $p in document("dblp.xml")/dblp/proceedings,
        $yr in $p/yr
   where $yr = “1999”
   return <proc> {$p} </proc>
} </root-tag>
```

```
for $i in document("dblp.xml")/dblp/inproceedings[author/text() = “John
   Smith”]
return <smith-paper>
                 <title>{ $i/title/text() }</title>
                 <key>{ data($i/@key) }</key>
                 { $i/crossref }
        </smith-paper>
```

Nesting in XQuery

Nesting XML trees is perhaps the most common operation

- In XQuery, it's easy – put a subquery in the return clause where you want things to repeat!

```
for $u in document("dblp.xml")/universities
where $u/country = "USA"
return <ms-theses-99>
            { $u/title }  {
             for $mt in $u/../mastersthesis
             where $mt/year/text() = "1999" and ____________
             return $mt/title }
        </ms-theses-99>
```

Collections & Aggregation

In XQuery, many operations return collections

- XPaths, sub-XQueries, functions over these, …
- The let clause assigns the results to a variable

Aggregation simply applies a function over a collection, where the function returns a value (very elegant!)

```
let $allpapers := document(“dblp.xml”)/dblp/article
return <article-authors>
    <count> { fn:count(fn:distinct-values($allpapers/authors)) } </count>
{   for $paper in doc(“dblp.xml”)/dblp/article
    let $pauth := $paper/author
    return <paper> {$paper/title}
                <count> { fn:count($pauth) } </count>
           </paper>
}   </article-authors>
```

Collections, Ctd.

Unlike in SQL, we can compose aggregations and create new collections from old:

```
<result> {
let $avgItemsSold := fn:avg(
   for $order in document(“my.xml”)/orders/order
   let $totalSold = fn:sum($order/item/quantity)
   return $totalSold)
   return $avgItemsSold
} </result>
```

Sorting in XQuery

- SQL actually allows you to sort its output, with a special ORDER BY clause (which we haven't discussed, but which specifies a sort key list)
- XQuery borrows this idea
- In XQuery, what we order is the sequence of "result tuples" output by the return clause:

```
for $x in document("dblp.xml")/proceedings
order by $x/title/text()
return $x
```

If Order Doesn't Matter

By default:

- SQL is unordered
- XQuery is ordered everywhere!
- But unordered queries are much faster to answer

XQuery has a way of telling the DBMS to avoid preserving order:

- ```
  unordered {
    for $x in (mypath) …
  }
  ```
  ```

Distinct-ness

In XQuery, DISTINCT-ness happens as a function over a collection

- But since we have nodes, we can do duplicate removal according to value or node
- Can do fn:distinct-values(collection) to remove duplicate values, or fn:distinct-nodes(collection) to remove duplicate nodes

```
for $years in fn:distinct-values(doc("dblp.xml")//
  year/text()
return $years
```

Querying & Defining Metadata

Can't do this in SQL!

Can get a node's name by querying node-name():

```
for $x in document("dblp.xml")/dblp/*
return node-name($x)
```

Can construct elements and attributes using computed names:

```
for $x in document("dblp.xml")/dblp/*,
  $year in $x/year,
  $title in $x/title/text(),
element node-name($x) {
  attribute {"year-" + $year} { $title }
}
```

XQuery Summary

Very flexible and powerful language for XML

- Clean and orthogonal: can always replace a collection with an expression that creates collections
- DB and document-oriented (we hope)
- The core is relatively clean and easy to understand

XSL(T): Bridge Back to HTML

- XSL (XML Stylesheet Language) is actually divided into two parts:
  - XSL:FO: formatting for XML
  - XSLT: a special transformation language

- We'll leave XSL:FO for you to read off www.w3.org, if you're interested
- XSLT is actually able to convert from XML → HTML, which is how many people do their formatting today
  - Products like Apache Cocoon generally translate XML → HTML on the server side

A Different Style of Language

- XSLT is based on a series of *templates* that match different parts of an XML document
  - There's a policy for what rule or template is applied if more than one matches (it's not what you'd think!)
  - XSLT templates can invoke other templates
  - XSLT templates can be nonterminating (beware!)
- XSLT templates are based on XPath "match"es, and we can also apply other templates (potentially to "select"ed XPaths)
  - Within each template, we describe what should be output
  - (Matches to text default to outputting it)

An XSLT Stylesheet

```
<xsl:stylesheet version="1.1">
 <xsl:template match="/dblp">
   <html><head>This is DBLP</head>
   <body>
     <xsl:apply-templates />
   </body>
   </html>
 </xsl:template>
 <xsl:template match="inproceedings">
   <h2><xsl:apply-templates select="title" /></h2>
   <p><xsl:apply-templates select="author"/></p>
 </xsl:template>
 …
</xsl:stylesheet>
```

Results of XSLT Stylesheet

```
<dblp>
  <inproceedings>
    <title>Paper1</title>
    <author>Smith</author>
  </inproceedings>
  <inproceedings>
    <author>Chakrabarti</author>
    <author>Gray</author>
    <title>Paper2</title>
  </inproceedings>
</dblp>
```

```
<html><head>This Is DBLP</
   head>
<body>
  <h2>Paper1</h2>
  <p>Smith</p>
  <h2>Paper2</h2>
   <p>Chakrabarti</p>
   <p>Gray</p>
</body>
</html>
```

What XSLT Can and Can't Do

- XSLT is great at converting XML to other formats
  - XML → diagrams in SVG; HTML; LaTeX
  - …
- XSLT doesn't do joins (well), it only works on one XML file at a time, and it's limited in certain respects
  - It's not a query language, really
  - … But it's a very good formatting language
- Most web browsers (post Netscape 4.7x) support XSLT and XSL formatting objects
- But most real implementations use XSLT with something like Apache Cocoon
- You may want to use XSL/XSLT for your projects – see www.w3.org/TR/xslt for the spec

Querying XML

We've seen three XML manipulation formalisms:

- XPath: the basic language for "projecting and selecting" (evaluating path expressions and predicates) over XML
- XQuery: a statically typed, Turing-complete XML processing language
- XSLT: a template-based language for transforming XML documents

- Each is extremely useful for certain applications!

XML Schema

- Use xs, xsd namespace
- Schema composed of elements
- Each element has name & type, and may have minOccurs/maxOccurs & restrictions
- Complex types may have child elements and attributes
- Child elements grouped by xs:sequence/ xs:all/xs:choice
- Attributes have name & type and may have optional flag

XPath

- Series of location steps to filter nodes
- Location step:

  axis-name::*node-test*[*predicate*]
- Shortcuts:
  - */node-test* == /child::*node-test*
  - All nodes pass the *node-test* * Example: /*/*
  - Axes: child, descendant-or-self, following-sibling, preceding-sibling, …
  - Relative paths (/. or /..) work as expected
  - // == descendant-or-self
  - Predicate tests [Inside brackets]
  - Prefix attributes by @
  - Standard comparisons: //show[@year > 2005]

    //show[contains(title, "The")]
  - Comparisons based on ordering:

    //surgery[//anesthesia[1] before //incision[1]]

XQuery

- for $x in *xpath*
- let $x := *expression*
- where *boolean comparison*
- order by *comparison*
- return *xml + variables in brackets*
- Example:

```
for $book in doc(library.xml)//book
let $authors := $book/author
where contains($book/title, "Potter")
order by $book/year
return <book> {$book/title} {$authors} </book>
```

Answers to Questions

- Comments in tags?
  - <tag <!-- comment --> >
  - **Not allowed**
- minOccurs and maxOccurs defaults
  - **1** for both (**not** 0 and unbounded)
- XQuery Updates? **Yes**
  - "Updating XML", Tatarinov et al., SIGMOD 2001
  - XQuery Update Facility: W3C Recommendation, March 17, 2011
  - Not supported by all tools yet