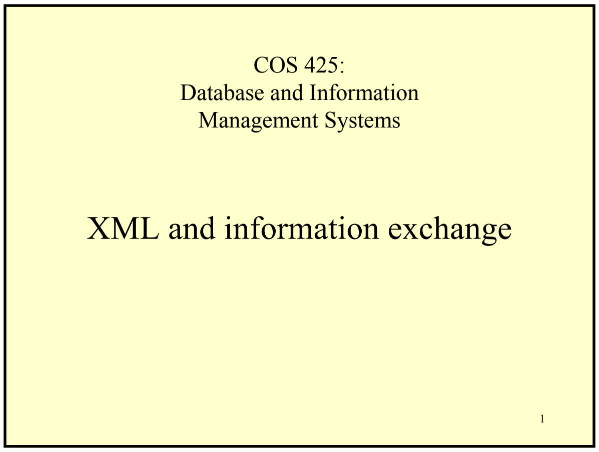
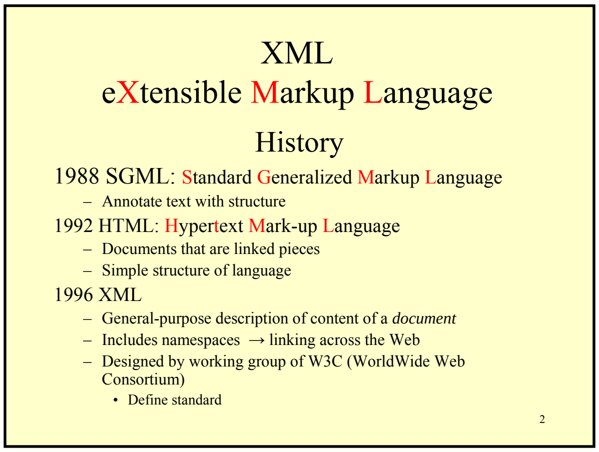


COS 425:
Database and Information
Management Systems

XML and information exchange



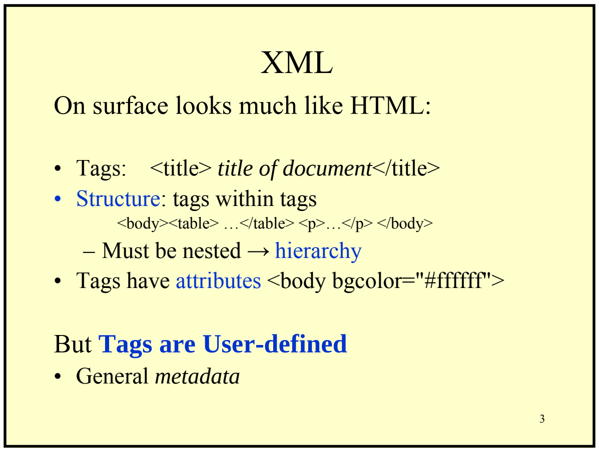
XML
eXtensible Markup Language

History

1988 SGML: Standard Generalized Markup Language
- Annotate text with structure

1992 HTML: Hypertext Mark-up Language
- Documents that are linked pieces
- Simple structure of language

1996 XML
- General-purpose description of content of a *document*
- Includes namespaces → linking across the Web
- Designed by working group of W3C (WorldWide Web Consortium)
  - Define standard

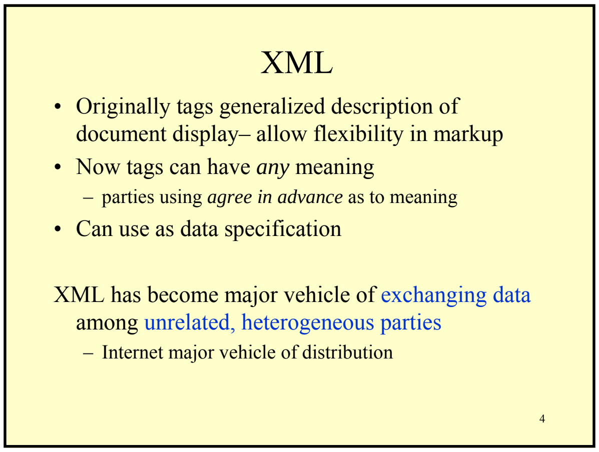


XML

On surface looks much like HTML:

- Tags: <title> *title of document*</title>
- Structure: tags within tags
  <body><table> …</table> <p>…</p> </body>
  - Must be nested → hierarchy
- Tags have attributes <body bgcolor="#ffffff">

But **Tags are User-defined**
- General *metadata*



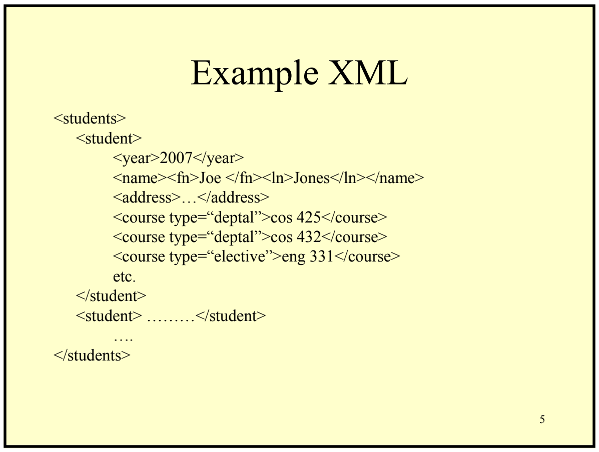
XML

- Originally tags generalized description of document display– allow flexibility in markup
- Now tags can have *any* meaning
  - parties using *agree in advance* as to meaning
- Can use as data specification

XML has become major vehicle of exchanging data among unrelated, heterogeneous parties

- Internet major vehicle of distribution

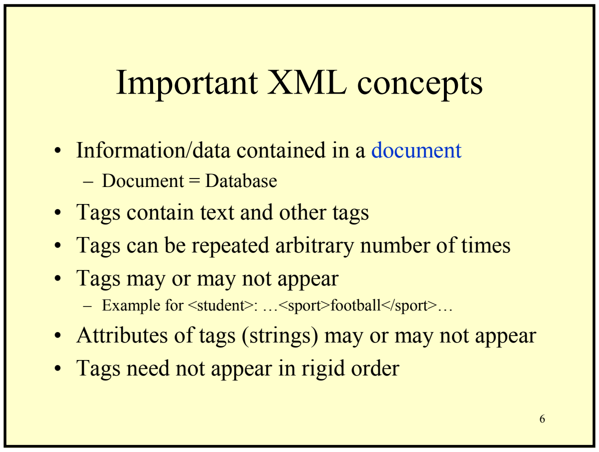
4



Example XML

```
<students>
    <student>
           <year>2007</year>
           <name><fn>Joe </fn><ln>Jones</ln></name>
           <address>…</address>
           <course type="deptal">cos 425</course>
           <course type="deptal">cos 432</course>
           <course type="elective">eng 331</course>
           etc.
    </student>
    <student> ………</student>
           ….
</students>
```

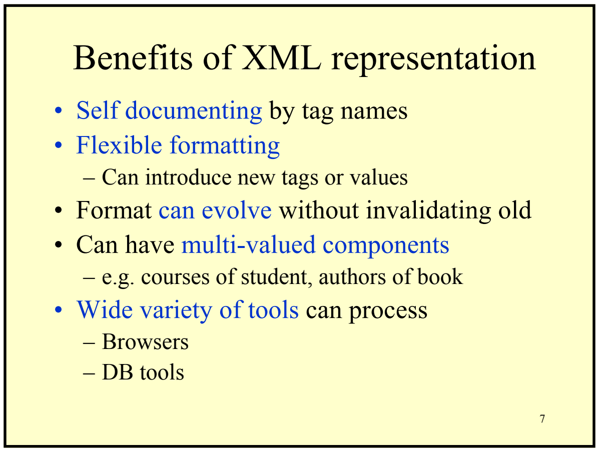
5



Important XML concepts

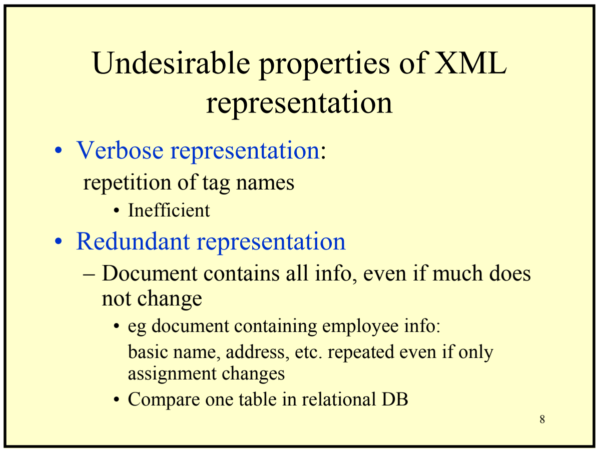
- Information/data contained in a document
  - Document = Database
- Tags contain text and other tags
- Tags can be repeated arbitrary number of times
- Tags may or may not appear
  - Example for <student>: …<sport>football</sport>…
- Attributes of tags (strings) may or may not appear
- Tags need not appear in rigid order

6



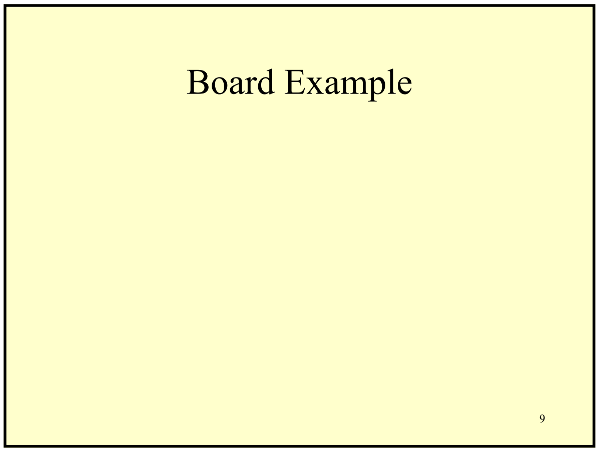
Benefits of XML representation

- Self documenting by tag names
- Flexible formatting
  - Can introduce new tags or values
- Format can evolve without invalidating old
- Can have multi-valued components
  - e.g. courses of student, authors of book
- Wide variety of tools can process
  - Browsers
  - DB tools

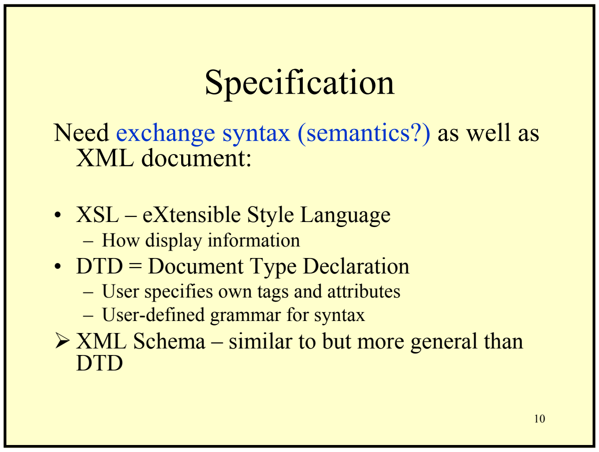


Undesirable properties of XML representation

- Verbose representation:
  repetition of tag names
  - Inefficient
- Redundant representation
  - Document contains all info, even if much does not change
    - eg document containing employee info:
      basic name, address, etc. repeated even if only assignment changes
    - Compare one table in relational DB



Board Example



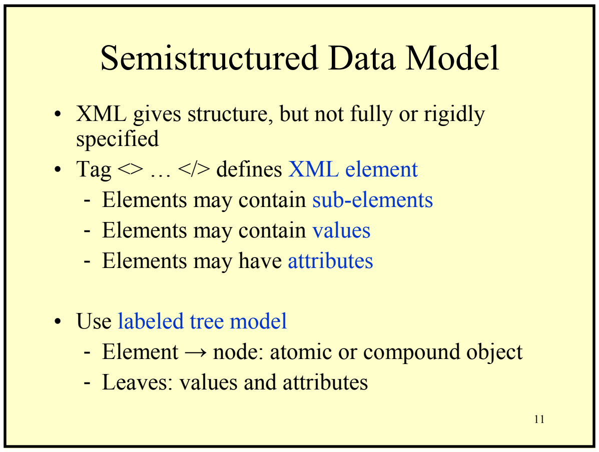
Specification

Need exchange syntax (semantics?) as well as XML document:

- XSL – eXtensible Style Language
  - How display information
- DTD = Document Type Declaration
  - User specifies own tags and attributes
  - User-defined grammar for syntax

➢ XML Schema – similar to but more general than DTD

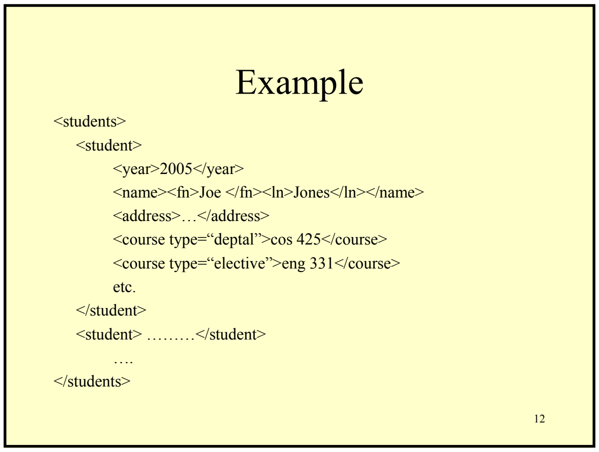
10



Semistructured Data Model

- XML gives structure, but not fully or rigidly specified
- Tag <> … </> defines XML element
  - Elements may contain sub-elements
  - Elements may contain values
  - Elements may have attributes
- Use labeled tree model
  - Element → node: atomic or compound object
  - Leaves: values and attributes

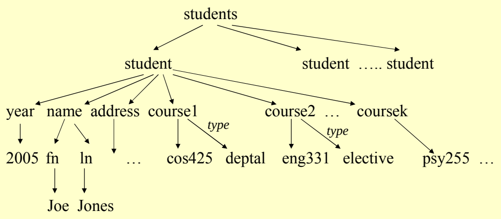
11



Example

```
<students>
    <student>
            <year>2005</year>
            <name><fn>Joe </fn><ln>Jones</ln></name>
            <address>…</address>
            <course type=“deptal”>cos 425</course>
            <course type=“elective”>eng 331</course>
            etc.
    </student>
    <student> ………</student>
            ….
</students>
```

12



students

student student ….. student

year name address course1 course2 … coursek

type *type*

2005 fn ln … cos425 deptal eng331 elective psy255 …

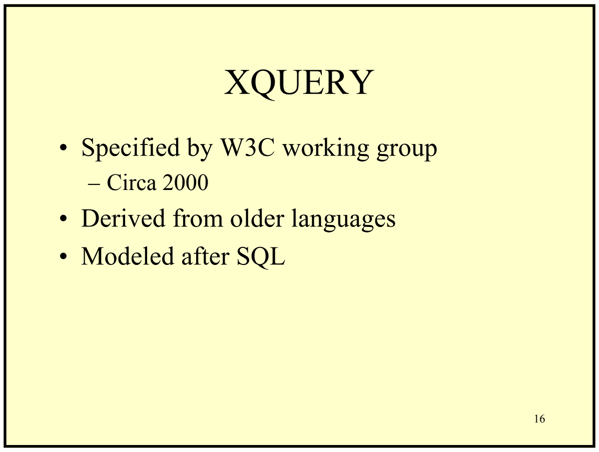
Joe Jones

XML Tools

- Display
  - Very flexible what and how display
- Convert to different represenation
  - Example: put in relational database?
- Extract information from XML document
  - ➢Querying

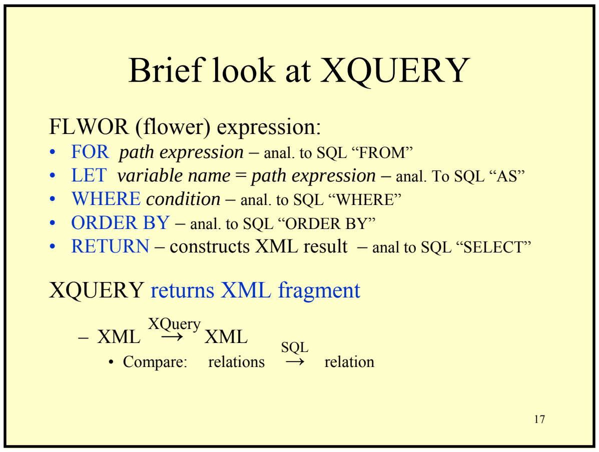
Querying XML

- Storing data in XML; want to query
- Could map to relational model, but then must restructure data
- Several querying languages
  - XPath : now building block
  - Quilt : historic
  - XQuery
  - XSLT : designed for style sheets but general



XQUERY

- Specified by W3C working group
  - – Circa 2000
- Derived from older languages
- Modeled after SQL



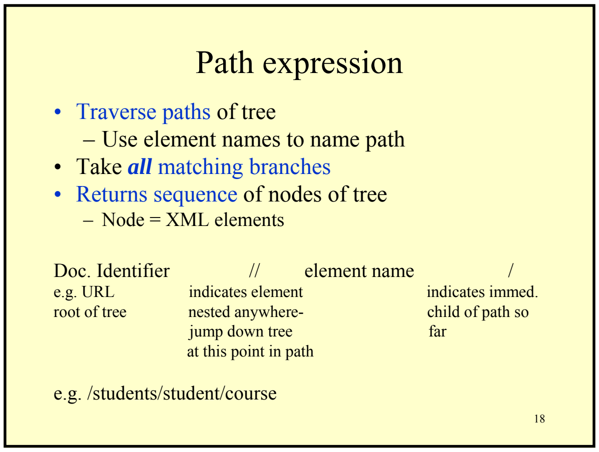
Brief look at XQUERY

FLWOR (flower) expression:

- FOR *path expression* – anal. to SQL “FROM”
- LET *variable name* = *path expression* – anal. To SQL “AS”
- WHERE *condition* – anal. to SQL “WHERE”
- ORDER BY – anal. to SQL “ORDER BY”
- RETURN – constructs XML result – anal to SQL “SELECT”

XQUERY returns XML fragment

- – XML $\xrightarrow{\text{XQuery}}$ XML
  - Compare: relations $\xrightarrow{\text{SQL}}$ relation

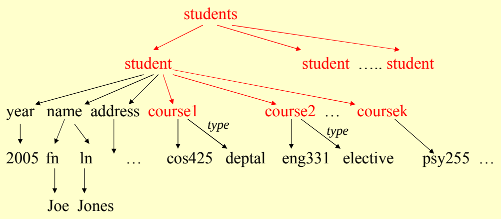


Path expression

- Traverse paths of tree
  - – Use element names to name path
- Take ***all*** matching branches
- Returns sequence of nodes of tree
  - – Node = XML elements

Doc. Identifier	//	element name	/
e.g. URL root of tree	indicates element nested anywhere- jump down tree at this point in path		indicates immed. child of path so far

e.g. /students/student/course


students

student

student ….. student

year name address course1 course2 … coursek

type

type

2005 fn ln … cos425 deptal eng331 elective psy255 …

Joe Jones

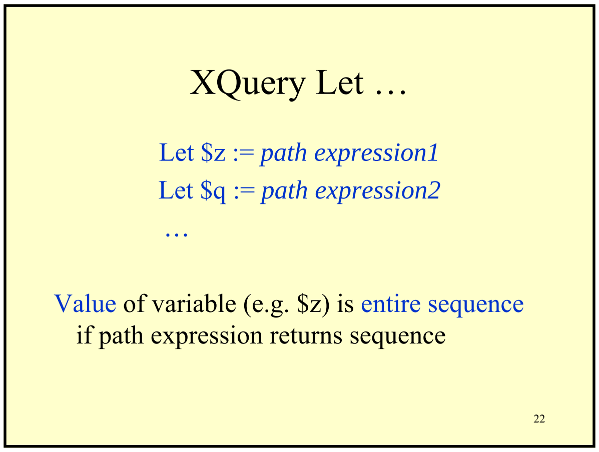
Path expressions – *some* details

- Returns sequence of matching elements
  - Includes tags of those elements
  - Sequence ordered by appearance in document
- Attributes can be accessed: @attribute_name
- … /* denotes *all children* of elements …/
- Predicates at any point in path
  - Prunes out paths
  - e.g. /students/student/course[@type=‘deptal’]
- Doc(*document name*) returns root of a named document
  - File name
  - URL (URI)

XQuery FOR …

For $x in *path expression 1,*
  $y *in path expression 2,*
…

- $ precedes variable name
- Each variable ranges over sequence of elements returned by its path expression
- Multiple variables => Cartesian product



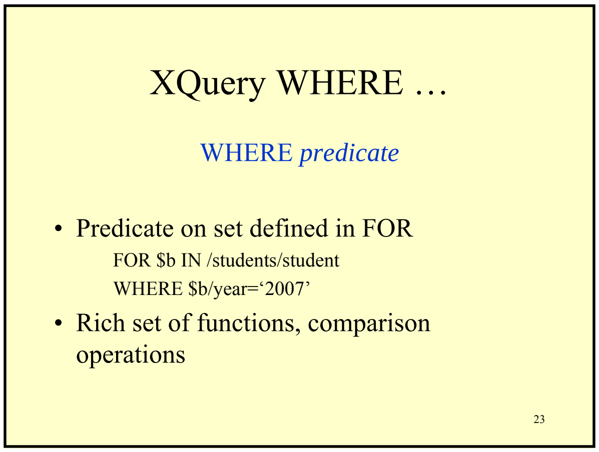
XQuery Let …

Let $z := *path expression1*

Let $q := *path expression2*

…

Value of variable (e.g. $z) is entire sequence if path expression returns sequence



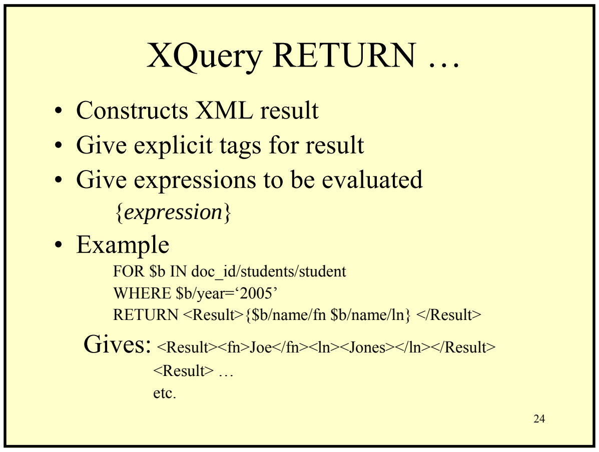
XQuery WHERE …

WHERE *predicate*

- Predicate on set defined in FOR

```
FOR $b IN /students/student
WHERE $b/year=‘2007’
```

- Rich set of functions, comparison operations



XQuery RETURN …

- Constructs XML result
- Give explicit tags for result
- Give expressions to be evaluated

  {*expression*}
- Example

```
FOR $b IN doc_id/students/student
WHERE $b/year=‘2005’
RETURN <Result>{$b/name/fn $b/name/ln} </Result>
```

Gives: <Result><fn>Joe</fn><ln><Jones></ln></Result>

<Result> …

etc.

Example

```
FOR $x IN doc_id//name/ln
RETURN <LastName>{$x}</LastName>
```

Gives: ?

For :

```
<students>
   <student>
         <year>2007</year>
         <name><fn>Joe </fn><ln>Jones</ln></name>
         …
   </student>
   <student>
         <year>2008</year>
         <name><fn>Jane </fn><ln>Smith</ln></name>
         …
   </student>

</students>
```

Examples

```
FOR $x IN doc_id//name/ln
RETURN < LastName >{$x}</LastName >
```

Gives:

```
<LastName><ln>Jones</ln></LastName>
< LastName ><ln>Smith</ln></LastName >
```

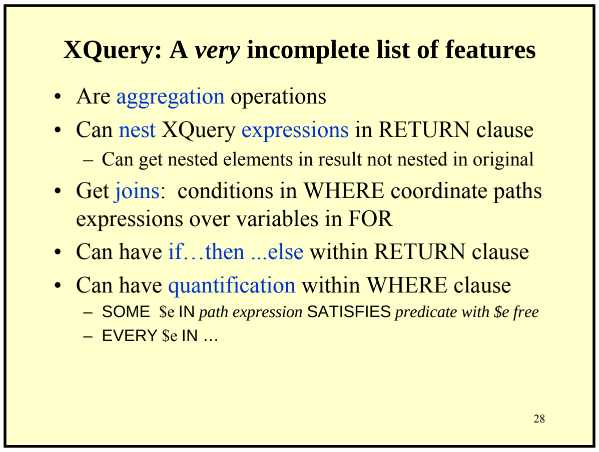
Examples

```
FOR $x IN doc_id//name/ln
RETURN < LastName >{$x/text()}</LastName >
```

Gives:

```
<LastName>Jones</LastName>
< LastName >Smith</LastName >
```

- Many functions



XQuery: A *very* incomplete list of features

- Are aggregation operations
- Can nest XQuery expressions in RETURN clause
  - Can get nested elements in result not nested in original
- Get joins: conditions in WHERE coordinate paths expressions over variables in FOR
- Can have if…then ...else within RETURN clause
- Can have quantification within WHERE clause
  - SOME $e IN *path expression* SATISFIES *predicate with $e free*
  - EVERY $e IN …