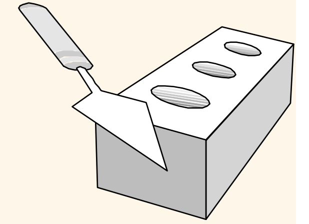
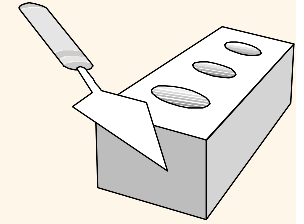


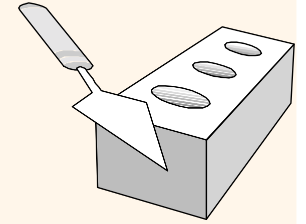
Querying XML

XQuery References



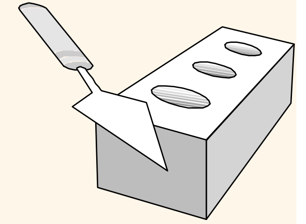
- XQuery 1.0: An XML Query Language
  http://www.w3.org/TR/xquery/
- XML Query Use Cases
  http://www.w3.org/TR/xmlquery-use-cases
- Qexo: The GNU Kawa implementation of XQuery
  http://www.gnu.org/software/qexo/
- XQuery Tutorial by Møller & Schwartzbach
  http://www.brics.dk/~amoeller/XML/querying/
- Xquery Tutorial by Fankhauser & Wadler
  http://homepages.inf.ed.ac.uk/wadler/papers/xquery-tutorial/xquery-tutorial.pdf
- Galax: an open-source XQuery implementation
  http://www.galaxquery.org/

XQuery Parsers



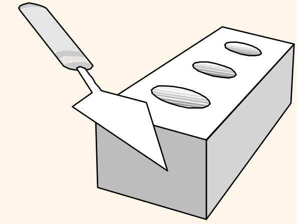
- Monetdb: http://monetdb.cwi.nl/XQuery/
- *XML Spy:* http://www.softandco.com/a/4295/xml-spy.html
- Galax: http://www.galaxquery.org/
- Qexo: http://www.gnu.org/software/qexo/Running.html
- Stylus: http://www.stylusstudio.com/xml_download.html

Why a new query language?



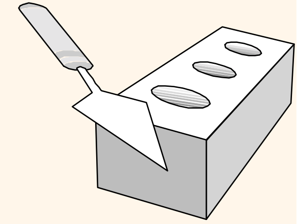
- Semi-structured: XML data is not rigidly structured
- Self-describing: schema exists with data
- Can naturally model irregularities
  - Missing elements (e.g., bestseller?)
  - Multiple occurrences of the same element (reviews*)
  - Elements w/ atomic values in some data items and structured values in others
  - Collections of elements with heterogeneous structure

XQuery 1.0

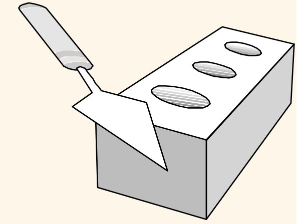


- Based on XML query language Quilt, with borrowed features from XPath, XQL, XML-QL, Lorel, YATL, SQL, and OQL.
- XQuery 1.0 relies on XPath 2.0 and XML Schema datatypes. The same expressions will generate the same results.
- Basic definitions:
  - It is a functional language and strongly typed
  - Basic building block: **expression**
  - The value of an expression is always a **sequence**: a collection of zero or more **items**
  - An item is either an **atomic value** or a **node**
  - A node conforms to one of **7 node types:**
    - Element, attribute, namespace, text, comment, processing-instruction, and document (root) node

XQuery Concepts



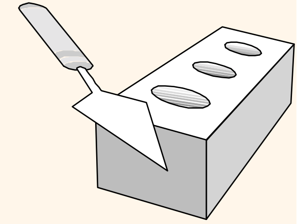
- A **query** in XQuery is an expression that:
  - reads a sequence of XML fragments or atomic values
  - returns a sequence of XML fragments or atomic values
  - Fragment is a general term to refer to part of an XML document
- The **principal forms** of XQuery expressions are:
  - path expressions
  - element constructors
  - FLWOR ("flower") expressions
  - list expressions
  - conditional expressions
  - quantified expressions
  - datatype expressions



Example XML Document

```
<BOOKS>
  <BOOK YEAR="1999 2003">
    <AUTHOR>Abiteboul</AUTHOR>
    <AUTHOR>Buneman</AUTHOR>
    <AUTHOR>Suciu</AUTHOR>
    <TITLE>Data on the Web</TITLE>
    <REVIEW>A <EM>fine</EM> book.</REVIEW>
  </BOOK>
  <BOOK YEAR="2002">
    <AUTHOR>Buneman</AUTHOR>
    <TITLE>XML in Scotland</TITLE>
    <REVIEW><EM>The <EM>best</EM> ever!</EM></REVIEW>
  </BOOK>
</BOOKS>
```

XQuery Examples



- Titles of all books published before 2000:

```
/BOOKS/BOOK[@YEAR < 2000]/TITLE
```

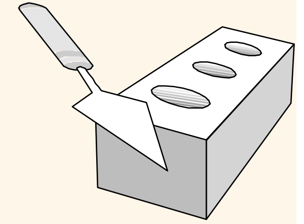
- Year and title of all books published before 2000:

```
for $book in /BOOKS/BOOK
where $book/@YEAR < 2000
return <BOOK>{ $book/@YEAR, $book/TITLE }</BOOK>
```

- Books grouped by author:

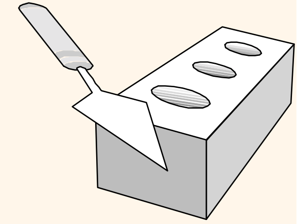
```
for $author in distinct(/BOOKS/BOOK/AUTHOR) return
  <AUTHOR NAME="{ $author }">{
    /BOOKS/BOOK[AUTHOR = $author]/TITLE
  }</AUTHOR>
```

XQuery Nodes & Expressions



- XQuery is an expression language
  - Every statement evaluates to some result
    - Let $x := 5 let $y := 6 return 10*$x+$y
    - Evaluates to 56
- Primitive types
  - Number, boolean, strings, dates, times, durations, and XML types
- Various functions create or return nodes.
  - Document function reads an XML file
    doc("http://infolab.usc.edu/csci585/Spring2004/bib.xml")/bib
  - Element constructor creates a new node:
    return <doc><par>Blah Blah</par></doc>
  - Use curly braces to embed XQuery expressions inside an element constructor:
    return <BOOK>{ $book/@YEAR, $book/TITLE }</BOOK>

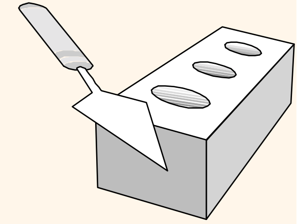
Path Expressions



- The simplest kind of query is just an **XPath 2.0** expression. A simple path expression looks like:

  doc("recipes.xml")//recipe[title="Ricotta Pie"]//ingredient[@amount]

  - The initial context for the path expression is given by **doc("recipes.xml")**
  - // means any child node (not just the immediate child node)
  - The result is all simple ingredients used to prepare Ricotta Pie in the recipe collection (http://www.brics.dk/~amoeller/XML/xml/recipes.xml)
  - The result is given as a list of XML fragments, each rooted with an **ingredient** element
  - The **order** of the fragments respects the document order (order matters! - as opposed to SQL)
  - Only return ingredients with "amount" attribute

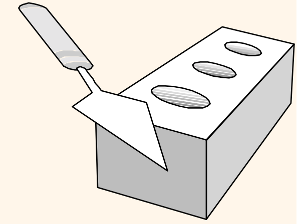


FLWOR Expressions

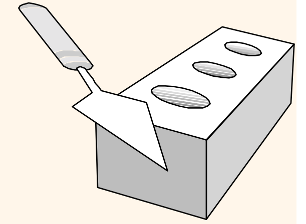
- The **main engine** of XQuery is the FLWOR expression:
- **F**or-**L**et-**W**here-**O**rder-**R**eturn
- Pronounced "flower"
- Generalizes SELECT-FROM-HAVING-WHERE from SQL; example:

```
for $d in doc("depts.xml")//deptno
let $e := doc("emps.xml")//employee[deptno = $d]
where count($e) >= 10
order by avg($e/salary) descending
return
  <big-dept>
    { $d,
      <headcount>{count($e)}</headcount>,
      <avgsal>{avg($e/salary)}</avgsal>
    }
  </big-dept>
```

FLWOR Expressions (cont'd)



- Definitions
  - **for** generates an ordered list of bindings of **deptno** values to **$d**
  - **let** associates to each binding a further binding of the list of **emp** elements with that department number to **$e**
  - at this stage, we have an ordered list of tuples of bindings: **($d,$e)**
  - **where** filters that list to retain only the desired tuples
  - **order** sorts that list by the given criteria
  - **return** constructs for each tuple a resulting value
- The combined result is in this case is a list of departments with at least 10 employees, sorted by average salaries.
- General rules:
  - **for** and **let** may be used many times in any order
  - only one **where** is allowed
  - many different sorting criteria can be specified



FLWOR Expressions (cont'd)

- Note the difference between **for** and **let**:

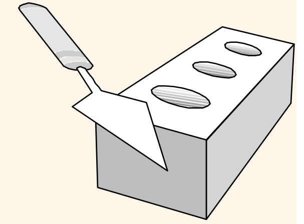
  for $x in /company/employee

  - Generates a list of bindings of **$x** to each **employee** element in the **company**, but:

  let $x := /company/employee

  - Generates a single binding of **$x** to the list of **employee** elements in the **company**.

Projection



- Return all authors of all books:

  /BOOKS/BOOK/AUTHOR

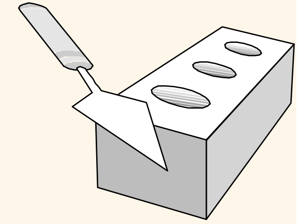
  - Returns:

    <AUTHOR>Abiteboul</AUTHOR>,
    <AUTHOR>Buneman</AUTHOR>,
    <AUTHOR>Suciu</AUTHOR>,
    <AUTHOR>Buneman</AUTHOR>

- The same query can also be written as a for loop:

  for $dot1 in $root/BOOKS return
  $dot1/BOOK/AUTHOR

Selection



- Return the titles of all books published before 2000:

```
/BOOKS/BOOK[@YEAR < 2000]/TITLE
```

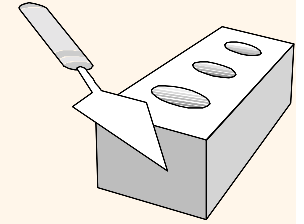
  - Returns:

    <TITLE>Data on the Web</TITLE>

- The above query is equivalent to:

```
for $book in /BOOKS/BOOK
where $book/@YEAR < 2000
return $book/TITLE
```

Selection *(cont'd)*



- Return book with title "Data on the Web":

  /BOOKS/BOOK[TITLE = "Data on the Web"]

  - Returns:

```
<BOOK YEAR="1999 2003">
  <AUTHOR>Abiteboul</AUTHOR>
  <AUTHOR>Buneman</AUTHOR>
  <AUTHOR>Suciu</AUTHOR>
  <TITLE>Data on the Web</TITLE>
  <REVIEW>A <EM>fine</EM> book.</REVIEW>
</BOOK>
```

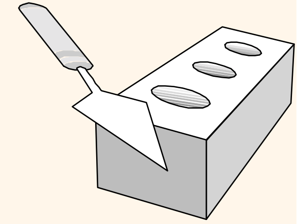
- Return the review of the book with the title "Data on the Web":

  /BOOKS/BOOK[TITLE = "Data on the Web"]/REVIEW

  - Returns:

```
<REVIEW>A <EM>fine</EM> book.</REVIEW>
```

Construction



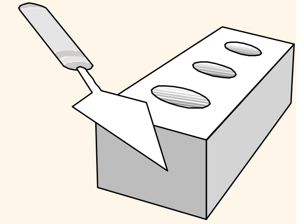
- Return year and title of all books published before 2000

```
for $book in /BOOKS/BOOK
where $book/@YEAR < 2000
return
  <BOOK>{ $book/@YEAR, $book/TITLE }</BOOK>
```

  - Returns:

```
<BOOK YEAR="1999 2003">
<TITLE>Data on the Web</TITLE>
</BOOK>
```

Grouping



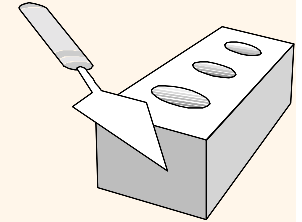
- Return titles for each author:

```
for $author in distinct(/BOOKS/BOOK/AUTHOR) return
  <AUTHOR NAME="{ $author }">{
    /BOOKS/BOOK[AUTHOR = $author]/TITLE
  }</AUTHOR>
```

  - Returns:

```
<AUTHOR NAME="Abiteboul">
  <TITLE>Data on the Web</TITLE>
</AUTHOR>,
<AUTHOR NAME="Buneman">
  <TITLE>Data on the Web</TITLE>
  <TITLE>XML in Scotland</TITLE>
</AUTHOR>,
<AUTHOR NAME="Suciu">
  <TITLE>Data on the Web</TITLE>
</AUTHOR>
```

Join



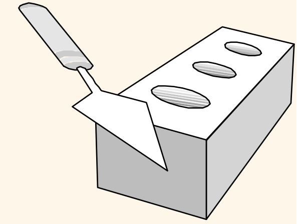
- Return the books that cost more at Amazon than at Fatbrain:

```
let $amazon := doc("http://www.amazon.com/books.xml"),
$fatbrain := doc("http://www.fatbrain.com/books.xml")
for $am in $amazon/BOOKS/BOOK,
  $fat in $fatbrain/BOOKS/BOOK
where $am/ISBN = $fat/ISBN
  and $am/PRICE > $fat/PRICE
return <BOOK>{ $am/TITLE, $am/PRICE, $fat/PRICE }</BOOK>
```

- Return the name and income of your neighbors:

```
for $p in doc("www.irs.gov/taxpayers.xml")//person
for $n in doc("neighbors.xml")//neighbor[ssn = $p/ssn]
return <person> <ssn> { $p/ssn } </ssn> { $n/name } <income> {
$p/income } </income> </person>
```

Other XML Query Languages



- XML-QL
- Lore (Lightweight Object Repository) & Lorel
- XSL, easy to express recursive processing:
  - All `author` elements, regardless of how deep they occur in the data:

```
<xsl:template> <xsl:apply-templates/> </xsl:template>
<xsl:template match="author"> <result> <xsl:value-of/>
  </result> </xsl:template>
```

- XQL: XSL match patterns+some concise syntax for constructing results
- XML-GL: similar in expressiveness power to XML-QL but with a GUI
- WebL: markup algebra + service combinators