XQuery

Arantza Irastorza Goñi

Informazioaren Kudeaketa Aurreratua

Gradua Ingeniaritza Informatikoan

Esp. Software Ingeniaritza

Lengoaiak eta Sistema Informatikoak saila

2017 urtarrila

eman ta zabal zazu

informatika fakultatea

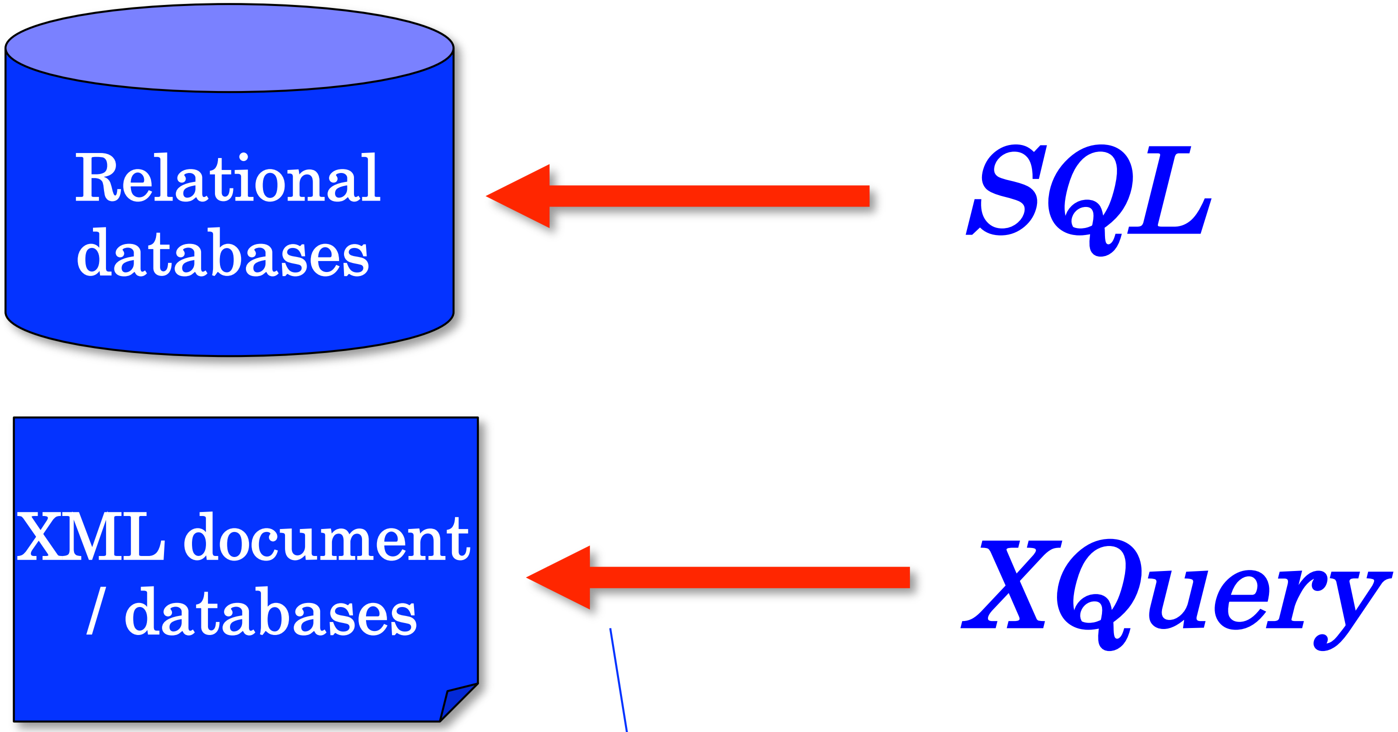
facultad de informática

Contents

- What is XQuery?
- Syntax and examples
  - Path expressions
  - Element constructors
  - FLWOR expressions
  - Selection: conditions
  - Order by, Join and Nested queries
  - Functions (built-in and user-defined)
- Type checking and validation
- Application: “screen-scraping”

© Acknowledgments. Part of these slides have been prepared using slides of Zaniolo, Hung-chih Yang, Ling-Jyh Chen

What is XQuery?



A query in XQuery is an expression that:

- Reads a number of XML documents or fragments
- Returns a sequence of well-formed XML fragments

What is XQuery?

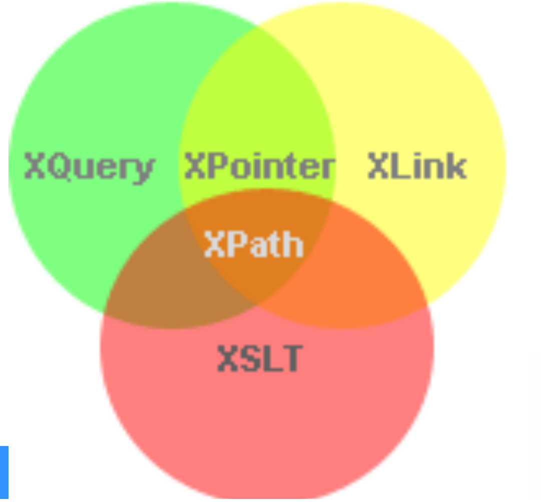
- The SQL for the XML world
- A query language that allows:
  - Selecting attributes/elements of a document
  - “Joining” nodes from several documents
  - Adding new attributes/elements to the result
  - Modifying the data
  - Calculating new data
  - Ordering the result

What is XQuery? (2)

- It is a typed functional language, where each query is represented as an expression
- It allows nested expressions *à la SQL*
  - Thus, allowing for query design based on stepwise refinement
- The input and output of an XQuery expression are instances of the XML data model
  - Thus, the output can be an XML document!!
- It has been designed to be legible, instead of using the XML notation, which is more verbose

XQuery vs XSLT

- XSLT is document-driven
  - XQuery is program driven
- XSLT is written in XML
  - XQuery is not
- An assertion (unproven): *XSLT 2.0 can do everything XQuery can do*



XML queries

- An XQuery unit:
  - a prolog + an expression
- Role of the prolog:
  - Populate the context where the expression is compiled and evaluated
- Prolog contains:
  - namespace definitions
  - schema imports
  - default element and function namespace
  - function definitions
  - collations declarations
  - function library imports
  - global and external variables definitions
  - Etc.

XQuery query = prolog + expression

```
(: an example :)
declare namespace  ok =“http://www.onekin.org/”
declare function ok:position($param) {…};
declare variable $cero {0};
```

prolog:
- comment
- namespace
- function
- variable

```
<bib>
 { for $b in doc("bib.xml")/bib/book
   where $b/publisher = "Addison-Wesley" and $b/@year > 1991
   return
        <libro año="{ $b/@year }">
             { $b/title }
         </libro>}
</bib>
```

Expression (between brackets): returns an XML data element

The Principal Forms of XQuery Expressions

Primary

- Literals, variables, function calls and parentheses (for control precedence)

Path

- Locates nodes within a tree, and returns a sequence of distinct nodes in document order

Sequence

- An ordered collection of zero or more items, where an item may be an atomic value or a node
- An item is identical to a sequence of length one containing that item. Sequences are never nested

The Principal Forms of XQuery Expressions (2)

Arithmetic

- Arithmetic operators for addition, subtraction, multiplication, division, and modulus

Comparison

- Four kinds of comparisons: value, general, node, and order comparisons

Logical

- A logical expression is either an AND-expression or an OR-expression
- The value of a logical expression is always a Boolean value

The Principal Forms of XQuery Expressions (3)

Constructor

- Constructors can create XML structures within a query.
- There are constructors for elements, attributes, CDATA sections, processing instructions, and comments

FLWOR

- Expression for iteration and for binding variables to intermediate results
- Useful for computing joins between two or more documents and for restructuring data
- Pronounced "flower", stands for the keywords FOR, LET, WHERE, ORDER BY and RETURN, the five clauses found in a FLWOR expression

The Principal Forms of XQuery Expressions (4)

Sorting expressions

- Provides a way to control the order of items in a sequence

Conditional expressions

- Based on the keywords IF, THEN, and ELSE

Quantified expressions

- support existential and universal quantification
- The value of a quantified expression is always true or false

The Principal Forms of XQuery Expressions (5)

Data types

- Runtime type checking and manipulation

Validate

- A validate expression validates its argument with respect to the in-scope schema definitions, using the schema validation process described in XML Schema

Contents

- What is XQuery?
- Syntax and examples
  - Path expressions
  - Element constructors
  - FLWOR expressions
  - Selection: conditions
  - Order by, Join and Nested queries
  - Functions (built-in and user-defined)
- Type checking and validation
- Application: “screen-scraping”

© Acknowledgments. Part of these slides have been prepared using slides of Zaniolo, Hung-chih Yang, Ling-Jyh Chen

Sample document: *bib.xml*

```
<?xml version="1.0"?>
<bib>
    <book year="1994">
        <title>TCP/IP Illustrated</title>
        <author>
            <last>Stevens</last>
            <first>W.</first>
        </author>
        <publisher>Addison-Wesley</publisher>
        <price>65.95</price>
    </book>

    <book year="1992">
        <title>Advanced Programming in the Unix environment</title>
        <author>
            <last>Stevens</last>
            <first>W.</first>
        </author>
        <publisher>Addison-Wesley</publisher>
        <price>65.95</price>
    </book>
```

cont. →

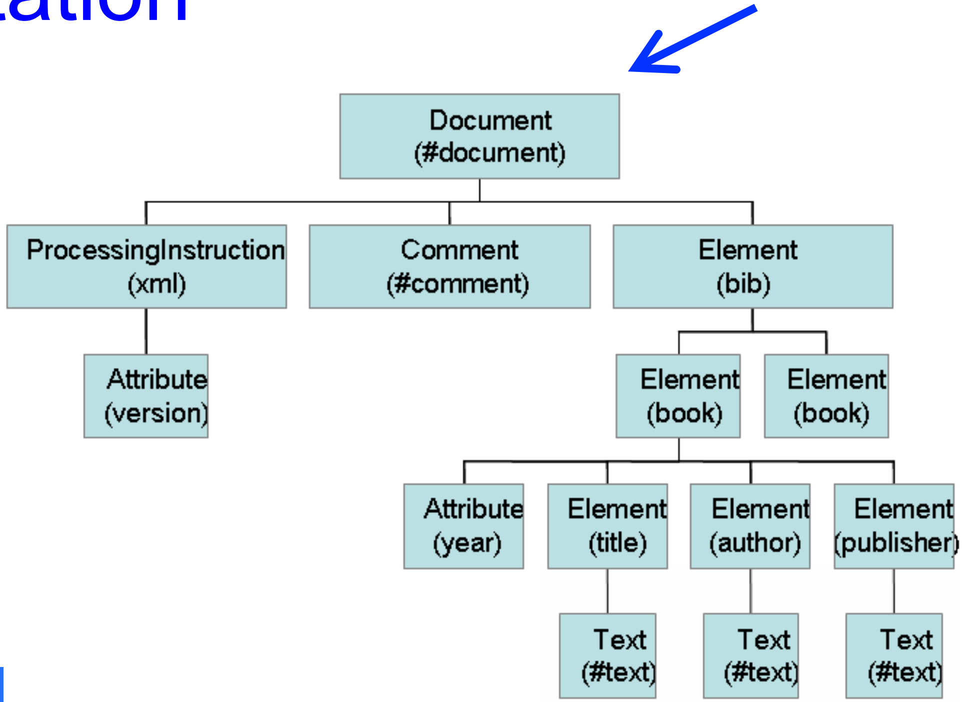
```
<book year="2000">
    <title>Data on the Web</title>
    <author>
        <last>Abiteboul</last>
        <first>Serge</first>
    </author>
    <author>
        <last>Buneman</last>
        <first>Peter</first>
    </author>
    <author>
        <last>Suciu</last>
        <first>Dan</first>
    </author>
    <publisher>Morgan Kaufmann Publishers</publisher>
    <price>39.95</price>
</book>

<book year="1999">
    <title>The Economics of Technology and Content for Digital TV</title>
    <editor>
        <last>Gerbarg</last>
        <first>Darcy</first>
        <affiliation>CITI</affiliation>
    </editor>
    <publisher>Kluwer Academic Publishers</publisher>
    <price>129.95</price>
</book>
</bib>
```

Path expression

- Locates nodes within a tree, and returns a sequence of distinct nodes in document order

- Based on XPath notation
  - Queries can refer to specific documents using the XQuery *doc()* function



Path expression. Example

- Find all books with a price of $39.95

```
doc("bib.xml")/bib/book[price = 39.95]
```

Result:

```
<book year="2000">
   <title>Data on the Web</title>
   <author><last>Abiteboul</last><first>Serge</first></author>
   <author><last>Buneman</last><first>Peter</first></author>
   <author><last>Suciu</last><first>Dan</first></author>
   <publisher>Morgan Kaufmann Publishers</publisher>
   <price> 39.95</price>
</book>
```

Path expression. Example

- Find the title of all books published before 1995

```
doc("bib.xml")/bib/book[@year < 1995]/title
```

Result

```
<title>TCP/IP Illustrated</title>
<title>Advanced Programming in the Unix environment</title>
```

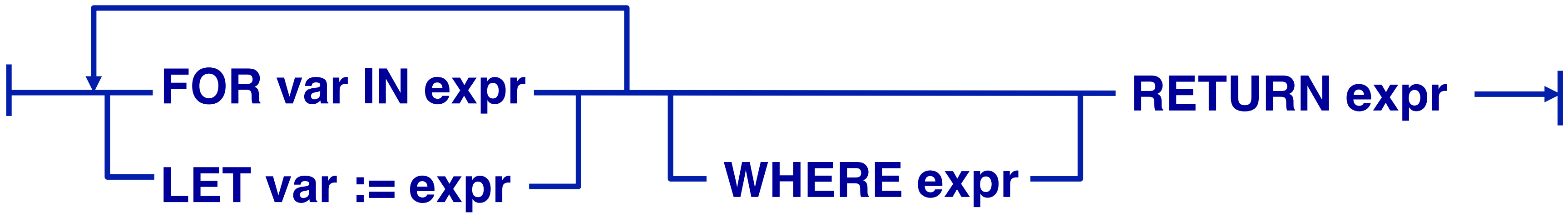
FLWOR (*Flower*) expressions

- FLWOR: For Let Where Order by Return
- Similar to SQL’s SELECT/FROM/WHERE

```
FOR $b in doc(“bib.xml”)//book
WHERE $b/author/first=“John”
      and $b/@year > 2000
RETURN $b/title
```

- …but adds LET to store intermediate results

FLWOR (Flower) expressions (2)



FOR and LET generate a list of linked expression tuples, preserving the order of the document

WHERE filters tuples that do not satisfy the predicate

RETURN is applied to each tuple that satisfies the predicate, generating an ordered list of elements

FLWOR expressions. Example

- Title of books

```
<bib>
 {
  for $b in doc("bib.xml")/bib/book/title
  return
    <item>
        { $b }
    </item>
 }
</bib>
```

A kind of template you fill up at run time

Result:

```
<bib>
     <item>
          <title>TCP/IP Illustrated</title>
     </item>
     <item>
          <title>Advanced Progr…</title>
     </item>
    ….
</bib>
```

FLWOR expressions. Example

- Title of books

```
<bib>
 {
  for $b in doc("bib.xml")/bib/book/title
  return
    <item>
        { data($b) }
    </item>
 }
</bib>
```

data(): retrieves the content of the node. When applied to a structured node, it retrieves the concatenation of the data from the leaf nodes

Result:

```
<bib>
    <item>
        TCP/IP Illustrated
    </item>
    <item>
        Advanced Programming …
    </item>
    …
</bib>
```

FLWOR expressions. Example

- Year and title of books published by Addison-Wesley after 1991

```
<bib>
 {
  for $b in doc("bib.xml")/bib/book
  where $b/publisher = "Addison-Wesley" and $b/@year > 1991
  return
    <book year="{ $b/@year }">
        { $b/title }
    </book>
 }
</bib>
```

Result:

```
<bib>
    <book year="1994">
        <title>TCP/IP Illustrated</title>
    </book>
    <book año="1992">
        <title>Advanced </title>
    </book>
</bib>
```

FLWOR expressions. Element constructor

- The constructor consists of
  - a start tag,
  - an end tag,
  - In between, a list of expressions that return the elements (between { })
- Noted that an XML document is a valid XQuery expression

```
<bib>
 {
  for $b in doc(”/docs/bib.xml")/bib/book
  where $b/publisher = "Addison-Wesley" and $b/@year > 1991
  return
    <book year="{ $b/@year }">
       { $b/title }
    </book>
 }
</bib>
```

{ } indicates a function to evaluate

```
<bib>
     <book year="1994">
              <title>TCP/IP Illustrated</title>
     </book>
     <book year="1992">
              <title>Advanced </title>
     </book>
</bib>
```

FLWOR expressions. Significance of { }

```
<books>{
        for $b in doc(‘bib.xml’)//book
        where $b/author/first = ‘John’
           and $b/author/last = ‘Smith’
        return <book>
                   $b/title,
                   $b/price
               </book>
   }</books>
```

As the brackets { } are missing, the system does not interpret it, it takes them as data

Result

```
<books>
      <book>$b/title,$b/price</book>
      <book>$b/title,$b/price</book>
   </books>
```

FLWOR expressions. Example

- Obtain a list of <*publisher*> elements with the editor’s name and the average price of its books

```
for $p in distinct-values(doc("bib.xml")//publisher)

let $a := avg(doc("bib.xml")//book[publisher = $p]/price)

return

      <publisher>

         <name> {$p} </name>

         <avgprice> {$a} </avgprice>

      </publisher>
```

Removes nodes with the same content. It uses the data() function for it. Notice, the node (i.e. <*title*> *house* </*title*>) is transformed into data (i.e. *house*)

Assigns the average price to $a

FOR vs. LET

- **for $x in** ***exp*** **[where** ***pred*****] return** ***body***
  - both pred and body may depend on the value of $x
  - if expression exp returns the sequence (v1,v2,...,vn), then
    - variable $x is bound to v1 first; if pred is true, then evaluate the body
    - variable $x is bound to v2 next; if pred is true, then evaluate the body, etc
    - ...; finally, variable $x is bound to vn; if pred is true, then evaluate the body
  - all the results of evaluating the body are concatenated

```
for $a in (1,2,3,4) return $a, $a
```

returns → 1,1 2,2 3,3 4,4

```
for $a in (1,2,3,4) return <b>{$a+10}</b>
```

→

FOR vs. LET (2)

- **let $x :=** *exp* **return** *body*
  - if the expression exp returns the sequence of values (v1,v2,...,vn), then $x is bound to the entire sequence

```
let $a := (1,2,3,4) return $a,$a
```

$\xrightarrow{\textbf{returns}}$ (1 2 3 4,1 2 3 4)

Sample document: *bib.xml*

```
<?xml version="1.0"?>
<bib>
    <book year="1994">
        <title>TCP/IP Illustrated</title>
        <author>
            <last>Stevens</last>
            <first>W.</first>
        </author>
        <publisher>Addison-Wesley</publisher>
        <price>65.95</price>
    </book>

    <book year="1992">
          <title>Advanced Programming in the Unix
    environment</title>
        <author>
            <last>Stevens</last>
            <first>W.</first>
        </author>
        <publisher>Addison-Wesley</publisher>
        <price>65.95</price>
    </book>
```

cont. →

```
<book year="2000">
    <title>Data on the Web</title>
    <author>
        <last>Abiteboul</last>
        <first>Serge</first>
    </author>
    <author>
        <last>Buneman</last>
        <first>Peter</first>
    </author>
    <author>
        <last>Suciu</last>
        <first>Dan</first>
    </author>
    <publisher>Morgan Kaufmann Publishers</publisher>
    <price>39.95</price>
</book>

<book year="1999">
    <title>The Economics of Technology and Content for
Digital TV</title>
    <editor>
        <last>Gerbarg</last>
        <first>Darcy</first>
        <affiliation>CITI</affiliation>
    </editor>
    <publisher>Kluwer Academic Publishers</publisher>
    <price>129.95</price>
</book>
</bib>
```

FOR vs. LET. Example (1)

```
<result>
 {
   for $b in doc("bib.xml")/bib/book/title
   return
      <titles>
         { $b }
      </titles>
 }
</result>
```

Result

```
<result>
     <titles>
          <title>TCP/IPIllustrated</title>
     </titles>
     <titles>
          <title>Advanced Programming in..</title>
     </titles>
     <titles>
          <title>Data on the Web</title>
     </titles>
     <titles>
          <title>The Economics of Technol…</title>
     </titles>
</result>
```

FOR vs. LET. Example (1)

```
<result>
 {
   let $b := doc("bib.xml")/bib/book/title
   return
      <titles>
         { $b }
      </titles>
 }
</result>
```

Result

```
<result>
  <titles>
    <title>TCP/IP Illustrated</title>
    <title>Advanced Programming in Unix..</title>
    <title>Data on the Web</title>
    <title>The Economics of Technology ..</title>
  </titles>
<result>
```

FOR vs. LET. Example (2)

```
<results>
 {
   for $b in doc("bib.xml")/bib/book,
       $a in $b/author
   return
      <result>
         { $b/title }
         { $a }
      </result>
 }
</results>
```

Result

```
<results>
   <result>
      <title>Data on the Web</title>
      <author>
         <last>Abiteboul</last>
         <first>Serge</first>
      </author>
   </result>
   <result>
      <title>Data on the Web</title>
      <author>
         <last>Buneman</last>
         <first>Peter</first>
      </author>
   </result>
   <result>
      <title>Data on the Web</title>
      <author>
         <last>Suciu</last>
         <first>Dan</first>
      </author>
   </result>
</results>
```

FOR vs. LET. Example (2)

```
<results>
 {
   for $b in doc("bib.xml")/bib/book
   let $a := $b/author
   return
      <result>
         { $b/title }
         { $a }
      </result>
 }
</results>
```

Result

```
<results>
    <result>
        <title>Data on the Web</title>
        <author>
            <last>Abiteboul</last>
            <first>Serge</first>
        </author>
        <author>
            <last>Buneman</last>
            <first>Peter</first>
        </author>
        <author>
            <last>Suciu</last>
            <first>Dan</first>
        </author>
    </result>
    <result>
        <title>The Economics TV</title>
    </result>
</results>
```

Conditions.
XPath predicate vs FLWOR where

- Books that have at least one *Peter* as authors' name and one *Buneman* as authors' surname. (Could be different authors)

```
<books>{
    for $b in doc('bib.xml')//book
    where  $b/author/first = 'Peter'
        and $b/author/last = 'Buneman'
    return <book>{
            $b/title,
            $b/price
    }</book>
}</books>
```

Existentially qualified

Result:

```
<books>
    <book>
        <title>Data on the web</title>
        <price>39.95</price>
    </book>
</books>
```

Conditions.
XPath predicate vs FLWOR where

- Books where at least one of the authors is *Peter Buneman*

```
<books>{
        for $b in doc(‘bib.xml’)//book
        where $b/author[first = ‘Peter’ and last = ‘Buneman’]
        return <book>{
                $b/title,
                $b/price
                  }</book>
}</books>
```

Alternatives?

Conditions: Existential conditions

- Find books that <u>have lendings over 10 days</u>

$x:\ book(x) \wedge \exists y\ (lending(y) \wedge y.booktit=x.title \wedge y.numdays>10)$

```
<books>{
  for $i in doc('bib.xml')//book
  where some $b in doc('biblendings.xml')//lending[booktit=$i/title]
             satisfies $b/numdays > 10
  return $i/title
}
</books>
```

equivalent

```
<books>{
  for $i in doc('biblio.xml')//book
  where doc('biblendings.xml')//lending[booktit=$i/title] [numdays > 10]
  return $i/title
}
</books>
```

```
<?xml version="1.0"?>
<bib>
    <book year="1994">
        <title>TCP/IP Illustrated</title>
        <author>
            <last>Stevens</last>
            <first>W.</first>
        </author>
        <publisher>Addison-Wesley</publisher>
        <price>65.95</price>
```

```
<lendings>
    <lending>
        <booktit>TCP/IP Illustrated</booktit>
        <member>Serge Abiteboul</member>
        <date>2012-02-01</date>
        <numdays>5</numdays>
    <lending>
        ...
```

Conditions: Universal conditions

- Find books whose lendings are all over 10 days

$x\text{: book}(x) \wedge \forall y\, (\text{lending}(y) \wedge y.\text{booktit}=x.\text{title} \rightarrow y.\text{numdays}>10)$

```
<books>{
   for $i in doc('bib.xml')//book
   where every $b in doc('biblendings.xml')//lending[booktit=$i/title]
               satisfies $b/numdays > 10
   return $i/title
}
</books>
```

equivalent

```
<books>{
   for $i in doc('biblio.xml')//book
   where not(doc('biblendings.xml')//lending[booktit=$i/title] [numdays <= 10])
   return $i/title
}
</books>
```

Conditional expressions

- Create a reference list, ordered by title. If the reference belongs to a *journal*, then the *publisher* will appear. Otherwise, the *author* will appear.

```
<?xml version="1.0"?>
<bib>
  <book year=“…“ type=“…”>
     <isbn> … </isbn>
     <title> … </title>
     <author> … </author>
     <publisher> … </publisher>
      <price> … </price>
   </book>
   ….
</bib>
```

```
for $h in doc("bib.xml")//book
return
        <myReference>
           {$h/title,
               if ($h/@type = "Journal“) then $h/publisher
               else  $h/author
               }
        </myReference>
```

Order by

- Title and year of every book published by Addison-Wesley after 1991, in alphabetical order

```
<bib>
 {
   for $b in doc("bib.xml")//book
   where $b/publisher = "Addison-Wesley" and $b/@year > 1991
   order by $b/title
   return
      <book>
         { $b/@year }
         { $b/title }
      </book>
}
</bib>
```

ORDER BY number($b/price)

Inner Join

- For books at both *bn.xml* and *amazon.xml* … …list the title of the book and its price from each source

```
<books-with-prices>
 {
   for $b in doc("bn.xml")//book,
            $a in doc("amazon.xml")//entry
   where $b/isbn = $a/isbn
   return
      <book>
         { $b/title}
         <price-amazon>{ $a/price }</price-amazon>
         <price-bn>{ $b/price }</price-bn>
      </book>
 }
</books-with-prices>
```

```
<?xml version="1.0"?>
<bib>
  <book year="…">
      <isbn> … </isbn>
      <title> … </title>
      <author> … </author>
      <publisher> … </publisher>
       <price> … </price>
   </book>
   ….
</bib>
```

bn.xml

```
<?xml version="1.0"?>
<bib>
  <entry year="…">
      <isbn> … </isbn>
      <title> … </title>
      <author> … </author>
      <publisher> … </publisher>
       <price> … </price>
   </entry>
   ….
</bib>
```

amazon.xml

Left-outer join

- For books at Amazon … All Amazon’s book should be outputed

```
<books-with-prices>
 {
  for  $a in doc(‘amazon.xml’)//entry
  return
      <book>
          {$a/title}
          <price-amazon>{$a/price}</price-amazon>

           {  for $b in doc(‘bn.xml’)//book
              where $b/isbn=$a/isbn
              return
                     <price-bn>{$b/price}</price-bn>
          }
        </book>
     }
 </books-with prices>
```

Full-outer join

- For all books at either Amazon or Bn …

```
let $allISBNs := distinct-values( doc('amazon.xml')//entry/isbn
                                    union doc('bn.xml')//book/isbn )
return
 <books-with-prices>
    {  for  $isbn in $allISBNs
       return
              <book>
                  {  for $a in doc('amazon.xml')//entry [isbn=$isbn]
                     return  <price-amazon>{$a/price}</price-amazon>
                  }

                  {  for $b in doc("bn.xml")//book[isbn=$isbn]
                     return  <price-bn>{$b/price}</price-bn>
                  }
             </book>
     }
 </books-with prices>
```

Group-by and Having

- For authors with more than 10 books …

  … output their first 10 books

```
for $a in distinct-values(doc(‘bib.xml’)//author/last)
let $books := doc(‘bib.xml’)//book[some $y in author satisfies $y/last=$a]
where count($books)>10
return <result  lastname=“{$a}”>
                    { $books[position()=1 to 10]/title }
          </result>
```

Nested XQueries

- For each book from Amazon …

  …obtain title and price, and the BN price, if this price is lower

```
<prices>{
 for $a in doc('www.amazon.com/books.xml')//book
 return
    <book>
              { $a/title, $a/price }
              { for $b in doc('www.bn.com/books.xml')//book
               where $b/@isbn=$a/@isbn
                   and $b/price < $a/price
               return $b/price }
    </book>
}</prices>
```

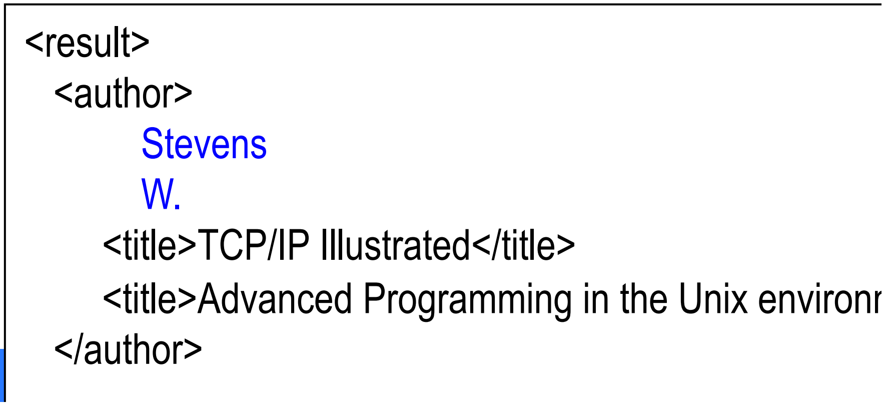
Any place an element's content can appear, a FLWOR expression can also appear

```
<prices>{
 for $a in doc('www.amazon.com')//boo
 for $b in doc('www.bn.com')//book
 where $b/@isbn=$a/@isbn
        and $b/price < $a/price
  return
    <book>
          { $a/title, $a/price , $b/price }
    </book>
}</prices>
```

Nested XQueries (2)

- In the original document, we had the authors for each book. Now we want it the other way around: for each author, his/her books

```
<result>{
        for $a in distinct-values(doc(‘bib.xml’)/bib/book/author)
        return
           <author>
                   {$a}
                   { for $t in doc(‘bib.xml’)/bib/book[author=$a]/title
                     return $t
                   }
           </author>
}</result>
```



```
<result>
  <author>
         Stevens
         W.
     <title>TCP/IP Illustrated</title>
     <title>Advanced Programming in the Unix environr
  </author>
```

Nested XQueries (3)

- For each book that at least has one author, obtain the first two authors and an empty *<et-al/>* element, if there are more authors

```
<bib>
 {
   for $b in doc("bib.xml")//book
   where count($b/author) > 0
   return
       <book>
          { $b/title }
          { for $a in $b/author[position()<=2]
            return $a
          }
          { if (count($b/author) > 2)
               then <et-al/>
               else ()
          }
       </book>
 }
</bib>
```

The query is made using a stepwise approach

```
<bib>
    <book>
        <title>Advanced … </title>
        <author>
            <last>Stevens</last>
            <first>W.</first>
        </author>
    </book>
    <book>
        <title>Data on the Web</title>
        <author>
            <last>Abiteboul</last>
            <first>Serge</first>
        </author>
        <author>
            <last>Buneman</last>
            <first>Peter</first>
        </author>
        <et-al/>
    </book>
</bib>
```

Nested XQueries (4)

- In the "prices.xml“ document, find the lowest price and extract a <*minprice*> element with the title as attribute and <*price*> as child element

```
<results>
 {
   let $doc := doc("/XQuery/prices.xml")
   for $t in distinct-values($doc//book/title)
   let $p := $doc//book[title = $t]/price
   return
     <minprice title="{ $t }">
             <price>{ min($p) }</price>
     </minprice>
 }
</results>
```

Returns a collection of <price> elements, then *min* aggregate function is applied to them

```
<results>
      <minprice title="Data Web">
              <price>34.95</price>
      </minprice>
      <minprice title=" TCP/IP ">
              <price>65.95</price>
      </minprice>
      <minprice title="Advanced">
              <price>65.95</price>
      </minprice>
</results>
```

Nested XQueries (5)

- Find book pairs with different titles but same authors (they should be in the same order)

```
<bib>
{
    for $book1 in doc("/docs/bib.xml")//book,
         $book2 in doc(“/docs/bib.xml")//book
    let $aut1 :=  for $a in $book1/author
                  order by $a/last, $a/first
                  return $a
    let $aut2 :=  for $a in $book2/author
                  order by $a/last, $a/first
                  return $a
    where $book1 << $book2
          and not($book1/title = $book2/title)
          and deep-equal($aut1, $aut2)
    return
        <book-pair>
            { $book1/title }
            { $book2/title }
        </book-pair>
}
</bib>
```

If B1 has the same title as B2, B2 has the same title as B1. This comparison avoids duplication due to this commutativity

Comparison between variables that contain structured nodes: **<<**, **deep-equal** ...

sequence-node-equal-any-order()

When authors can be in different order

Functions. Built-in

- URI of the function namespace

  http://www.w3.org/2005/02/xpath-functions

- The default prefix fn:
  - The function names do not need to be prefixed when called

```
<name>{upper-case($booktitle)}</name>
```

```
doc("books.xml")/bookstore/book[substring(title,1,5)='Harry’]
```

```
let $name := (substring($booktitle,1,4))
```

http://www.xqueryfunctions.com/

User-defined functions

- User-defined functions can be defined in the query or in a separate library

```
declare namespace prefix= "http://www.w3.org/2005/02/xpath-functions";
declare function prefix:function_name($parameter as datatype) as returnDatatype
{
 ...function code here...
};
```

```
declare function local:minPrice($p as xs:decimal, $d as xs:decimal)
as xs:decimal
{
     let $disc := ($p * $d) div 100
     return ($p - $disc)
};
```

```
<minPrice>{local:minPrice($book/price, $book/discount)}</minPrice>
```

Contents

- What is XQuery?
- Syntax and examples
  - Path expressions
  - Element constructors
  - FLWOR expressions
  - Selection: conditions
  - Order by, Join and Nested queries
  - Functions (built-in and user-defined)
- ➡ Type checking and validation
- Application: “screen-scraping”

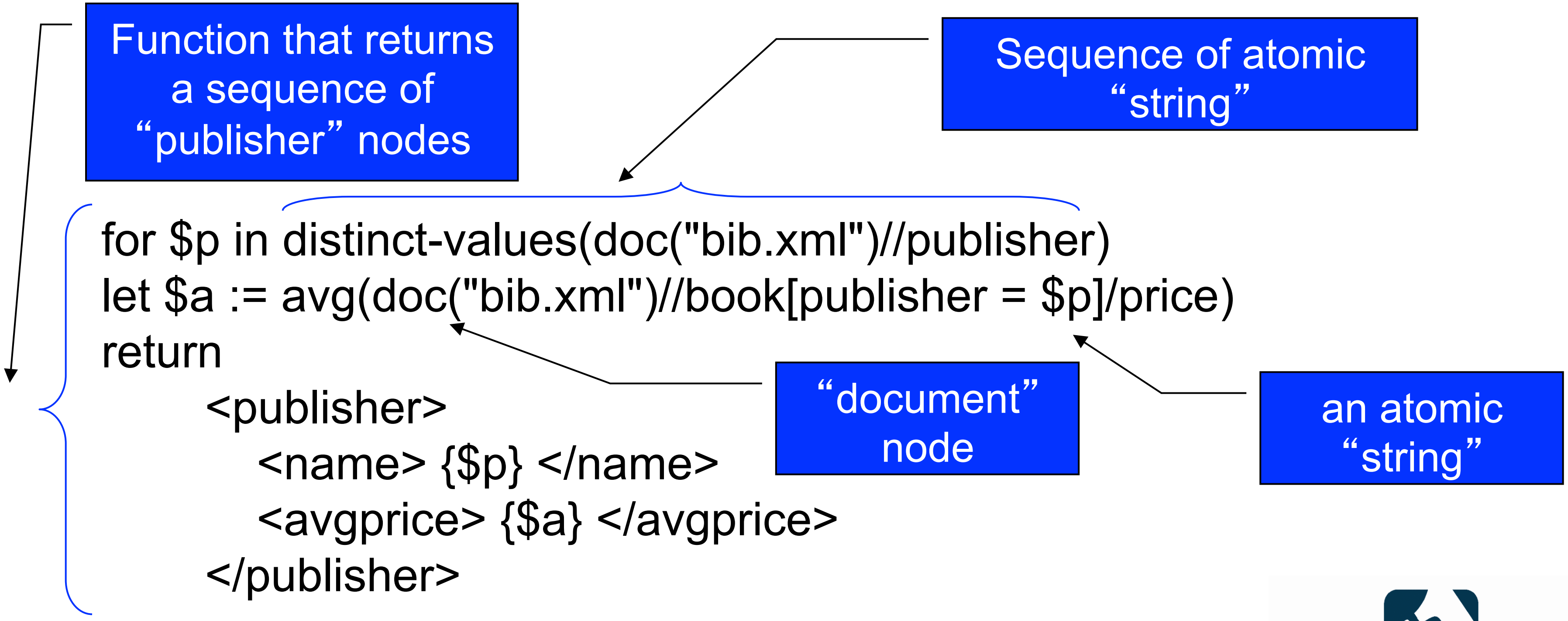
© Acknowledgments. Part of these slides have been prepared using slides of Zaniolo, Hung-chih Yang, Ling-Jyh Chen

Type checking and validation

- XQuery is strongly typed
  - Every expression has a given type, based on XML Schema 1.0
- Types
  - Pre-defined types: *xs:string*, *xs:date*, *xs:boolean*
  - Types imported from a user schema
  - XQuery's additional types (for schemaless documents)
    - with no type: ***xdt:untyped***
    - with no type but atomic: ***xdt:untypedAtomic***
    - generic atomic type: ***xdt:anyAtomicType***

Pre-defined types

- Data nodes (leaves) have TYPE
- Element nodes and attributes have TYPE ANNOTATIONS



Function that returns a sequence of “publisher” nodes

Sequence of atomic “string”

```
for $p in distinct-values(doc("bib.xml")//publisher)
let $a := avg(doc("bib.xml")//book[publisher = $p]/price)
return
        <publisher>
            <name> {$p} </name>
            <avgprice> {$a} </avgprice>
        </publisher>
```

“document” node

an atomic “string”

Constructors and *casting*

- Element and attribute constructors

```
element author { “Jennifer Widom” }
```

→ `<author>Jennifer Widom</author>`

```
element book {
          attribute year { 1977 },
          element author {
                    element first { “Jennifer" },
                    element last { “Widom" }   },
          element publisher {“ACM Publishing” },
          element price { 14.95 }
}
```

→

Constructors and *casting* (2)

- Each atomic type has its own constructor
  - *xs:date("2004-12-15")* creates an *xs:date* value
- *number()* and *string()* are used for type conversion
  - *number("123")* → 123
  - *string(123)* → "123"

Constructors and casting (3)

- Without a schema, XML data are *“untyped”*
- In most expressions, the system automatically converts the arguments to the expected type
  - product/price + 23
    - price would be untyped
    - System does number(product/price) to transform it into the type the + operator expects
- This does NOT happen with functions (see next slide)

```
for $i in (1 to 3)
return
    <sample> {concat(‘sample’,$i,’.doc’)} </sample>
```

Exception (*concat* function): *$i* is not an string. It is converted: string($i)

Constructors and casting (4)

- The system does not convert the arguments to the expected type

```
declare namespace fun = "http://www.w3.org/2005/02/xpath-functions";
declare function fun:fibo ($n as xs:integer) {
    if ($n = 0)
    then 0
    else if ($n = 1)
         then 1
         else (fun:fibo($n - 1) + fun:fibo($n - 2))
};

let $seq := 1 to 10
for $n in $seq
return <fibonacci n="{$n}">{ fun:fibo($n) }</fibonacci>
```

{ fun:fibo("4") }

Error: *"4" is* not an integer.
It has to be converted: string($i)

Schemas for *type checking*

An “*import schema* …” clause is compulsory if we query a schema-based document

```
import schema
        namespace ipo = “http://www.ehu.es/bib”
                at  “bibschema.xsd“ ;
…
for $x in doc(“bibesk.xml”)//ipo:book
where  $x/ipo:author/ipo:first
return $x/price +3
```

Schemas for *type checking*

Warning!!!
There is not check

```
import schema
        namespace ipo = “http://www.ehu.es/bib”
                at  “bibschema.xsd“ ;
…

for $x in doc(“bibesk.xml”)//ipo:author/book
where  $x/ipo:fist
return $x/prefix +3
```

Structure error!!

Book cannot hang from *author*

Spelling mistake!!
fist ⇒ first

Schemas for *type checking* (2)

```
import schema
        namespace ipo = “http://www.ehu.es/bib”
                at “bibschema.xsd“ ;

<item> {
   for $x in doc(“bib.xml”)//ipo:book/element()
   where  $x instance of  element(ipo:title)
   return $x
}</item>
```

Refers to any element node

Checks that content of *$x* is an element with name *title* (from *ipo* schema)

```
<item>
   <title xmlns="http://www.ehu.es/bib"
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-insta
                TCP/IP Illustrated
   </title>
   <title xmlns="http://www.ehu.es/bib"
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-insta
        Advanced Programming in the Unix environment
   </title>
…
</item>
```

Schemas for *type checking* (3)

```
declare function books-by-author($author)
{
     for $b in doc("bib.xml")/bib/book
     where some $ba in $b/author satisfies
             ($ba/last = $author/last and $ba/first = $author/first)
     order by $b/title
     return $b/title
}
```

ok

But, empty…

```
for $b in doc(“bib.xml")/bib/book
return books-by-author($b)
```

```
import schema default element namespace  "urn:examples:xmp:bib"
at "c:/dev/schemas/eg/bib.xsd"
declare function books-by-author($a as element(author)) as element(title)*
{
        for $b in doc("bib.xml")/bib/book
        where some $ba in $b/author satisfies
                        ($ba/last = $a/last and $ba/first = $a/first)
        order by $b/title
        return $b/title
}
```

error

Schemas for *type checking* (4)

```
import schema namespace b = "urn:examples:xmp:bib“
    at "c:/dev/schemas/eg/bib.xsd"
declare function books-by-author($a as element(b:author))
    as element(b:title)*
{
    for $b in doc("bib.xml")/b:bib/b:book
    where some $ba in $b/b:author satisfies
                    ($ba/b:last=$a/b:last and $ba/b:first=$a/b:first)
    order by $b/b:title
    return $b/b:title
}
```

Schemas for validation

- Validate the input documents as well as the result document

- To make this possible,

  1. We explicitly import the schemas of input and output documents
  2. Input documents: are validated explicitly
  3. Output documents: are implicitly validated as they are being created

Schemas for validation. Example

```
import schema namespace bib="urn:examples:xmp:bib“ at …

for $b in doc("bib.xml")//bib:book
return
    <bib:book>
    {
        $b/bib:title,
        $b//element(bib:creator)
    }
    </bib:book>
```

error

book **element does not have a** ***creator*** **element**

Schemas for validation. Example

```
import schema default element namespace  "http://www.example.com/auction“
        at “auction.xsd”
import schema namespace x = "http://www.w3.org/1999/xhtml" at “xhtml.xsd”;
validate strict {

let $auction := validate { doc(…)//auction }
return
    <x:html>
    <x:body>
          <x:h1>Auctions</x:h1>
              <x:table>
              <x:td>
                    <x:th>Item Name</x:th>
                    <x:th>Seller</x:th>
                    <x:th>Last Bid</x:th>
                    <x:th>Closes On</x:th>
              </x:td>
                   {
                   for $article in $auction/articles/article[start_date <= date()]   ….
```

Validates the input document against the schema specified in the document
explicit

Validates the output document against the “x” schema
implicit

Contents

- What is XQuery?
- Syntax and examples
  - Path expressions
  - Element constructors
  - FLWOR expressions
  - Selection: conditions
  - Order by, Join and Nested queries
  - Functions (built-in and user-defined)
- Type checking and validation
- ➡ Application: “screen-scraping”

© Acknowledgments. Part of these slides have been prepared using slides of Zaniolo, Hung-chih Yang, Ling-Jyh Chen

Uses of XQuery

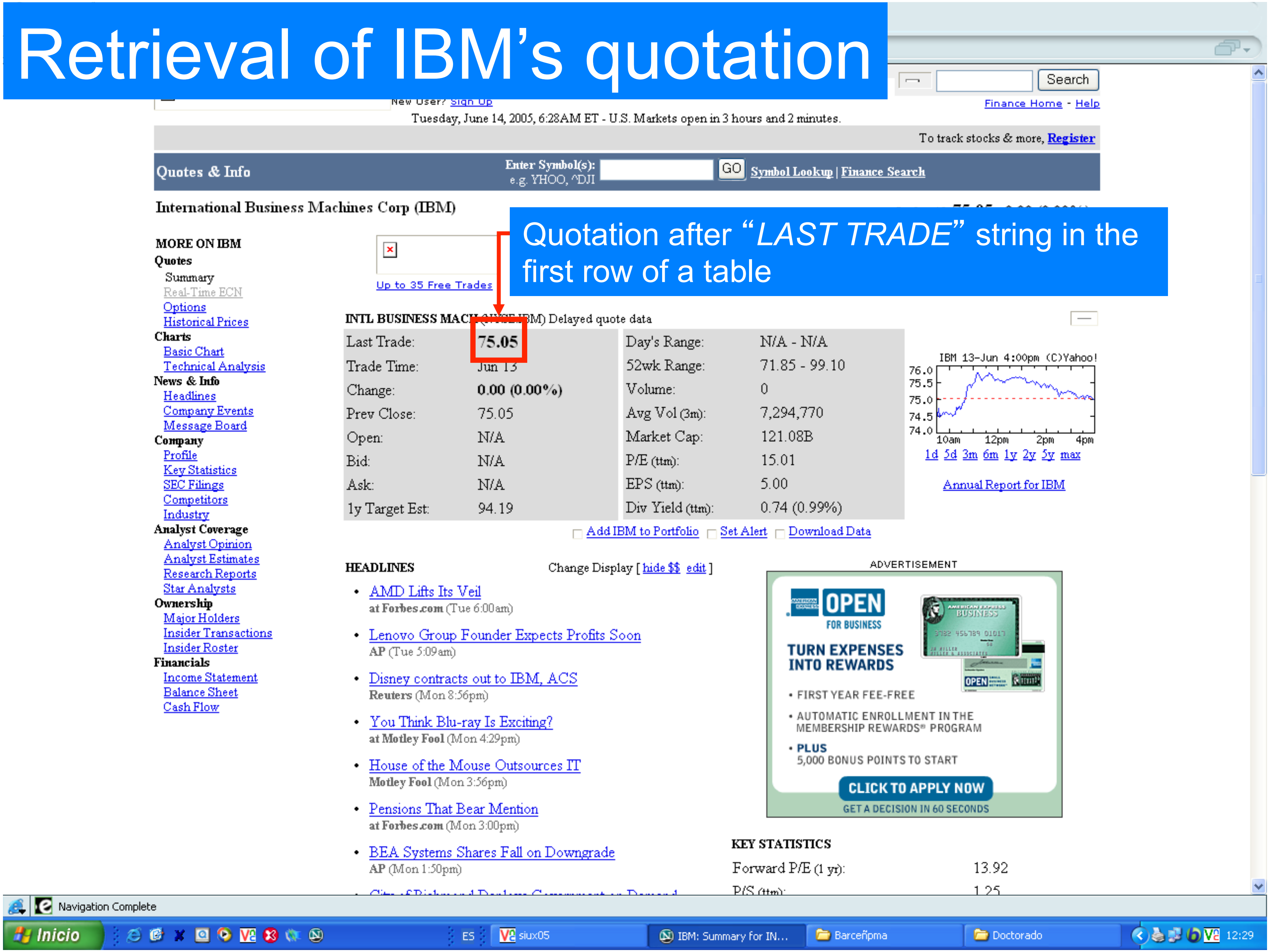
- XQuery can be used to:
  - Extract information to use in a Web Service
  - Generate summary reports
  - Transform XML data to XHTML
  - Search Web documents for relevant information (‘*screen-scraping*’)

XQuery for *screen-scraping*

- An HTML page can be transformed into an XML document (XHTML)
  - Using *JTidy*

- Once in XHTML, XQuery can be used to retrieve and transform the data from the page

Retrieval of IBM’s quotation



Quotation after “*LAST TRADE*” string in the first row of a table

Tuesday, June 14, 2005, 6:28AM ET - U.S. Markets open in 3 hours and 2 minutes.

To track stocks & more, **Register**

Quotes & Info — **Enter Symbol(s):** e.g. YHOO, ^DJI GO **Symbol Lookup** | **Finance Search**

International Business Machines Corp (IBM)

MORE ON IBM

Quotes
- Summary
- Real-Time ECN
- Options
- Historical Prices

Charts
- Basic Chart
- Technical Analysis

News & Info
- Headlines
- Company Events
- Message Board

Company
- Profile
- Key Statistics
- SEC Filings
- Competitors
- Industry

Analyst Coverage
- Analyst Opinion
- Analyst Estimates
- Research Reports
- Star Analysts

Ownership
- Major Holders
- Insider Transactions
- Insider Roster

Financials
- Income Statement
- Balance Sheet
- Cash Flow

Up to 35 Free Trades

INTL BUSINESS MACH (NYSE:IBM) Delayed quote data

Last Trade:	**75.05**	Day's Range:	N/A - N/A
Trade Time:	Jun 13	52wk Range:	71.85 - 99.10
Change:	**0.00 (0.00%)**	Volume:	0
Prev Close:	75.05	Avg Vol (3m):	7,294,770
Open:	N/A	Market Cap:	121.08B
Bid:	N/A	P/E (ttm):	15.01
Ask:	N/A	EPS (ttm):	5.00
1y Target Est:	94.19	Div Yield (ttm):	0.74 (0.99%)

Annual Report for IBM

□ Add IBM to Portfolio □ Set Alert □ Download Data

HEADLINES Change Display [hide $$ edit]

- AMD Lifts Its Veil
  at Forbes.com (Tue 6:00am)
- Lenovo Group Founder Expects Profits Soon
  AP (Tue 5:09am)
- Disney contracts out to IBM, ACS
  Reuters (Mon 8:56pm)
- You Think Blu-ray Is Exciting?
  at Motley Fool (Mon 4:29pm)
- House of the Mouse Outsources IT
  Motley Fool (Mon 3:56pm)
- Pensions That Bear Mention
  at Forbes.com (Mon 3:00pm)
- BEA Systems Shares Fall on Downgrade
  AP (Mon 1:50pm)

KEY STATISTICS

Forward P/E (1 yr):	13.92
P/S (ttm):	1.25

Retrieval of IBM’ s quotation

```
<table>
   {
   FOR $cell IN doc(“page.xhtml")//td[1]
   WHERE CONTAINS($cell/text(), "Last Trade")
   RETURN
       <tr>
         <td>
                 { $cell/following-sibling::td/text() }
         </td>
       </tr>
  }
</table>
```

```
<table>
<tr>
      <td>Last Trade</td>
      <td>75.05</td>
</tr>
<tr>
      <td>Trade Time</td>
      <td>Jun 13</td>
</tr>
</table>
```