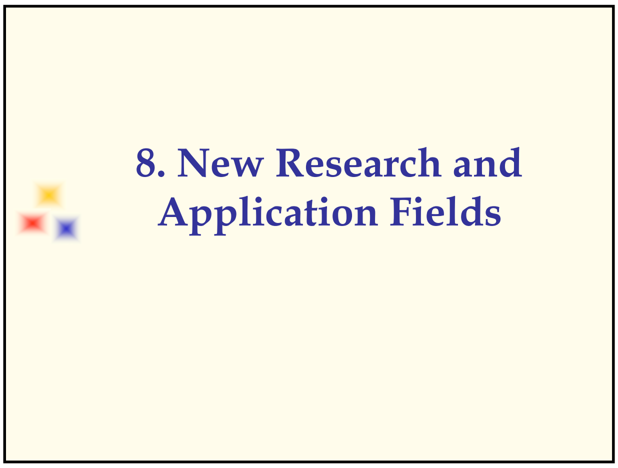
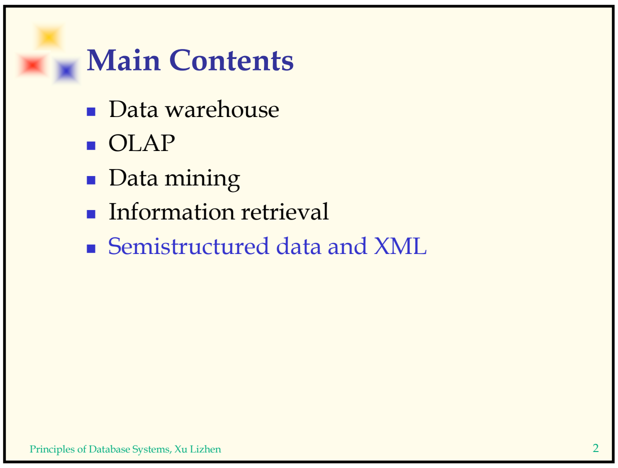
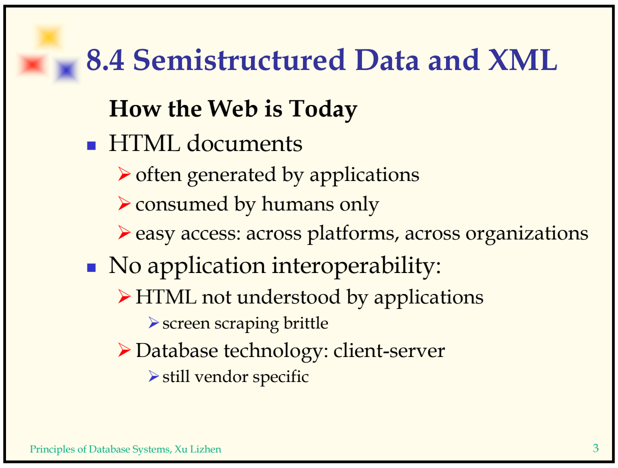


8. New Research and Application Fields



Main Contents

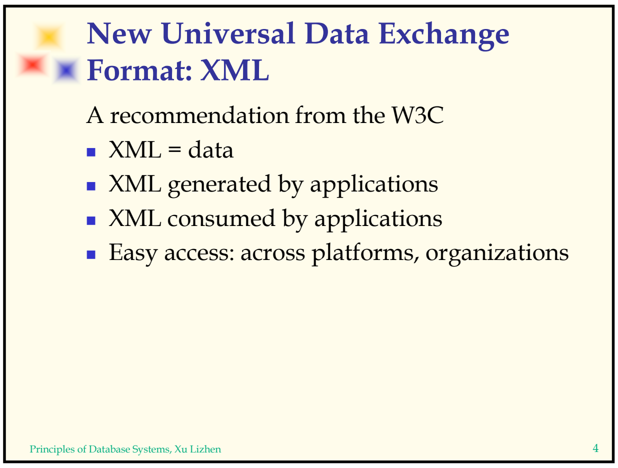
- Data warehouse
- OLAP
- Data mining
- Information retrieval
- Semistructured data and XML



8.4 Semistructured Data and XML

How the Web is Today

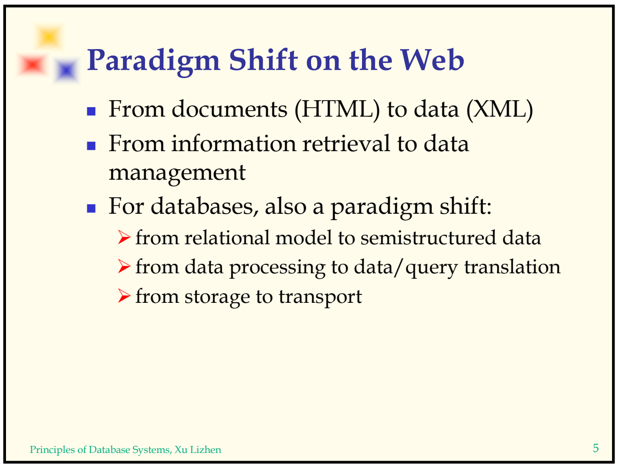
- HTML documents
  - often generated by applications
  - consumed by humans only
  - easy access: across platforms, across organizations
- No application interoperability:
  - HTML not understood by applications
    - screen scraping brittle
  - Database technology: client-server
    - still vendor specific



New Universal Data Exchange Format: XML

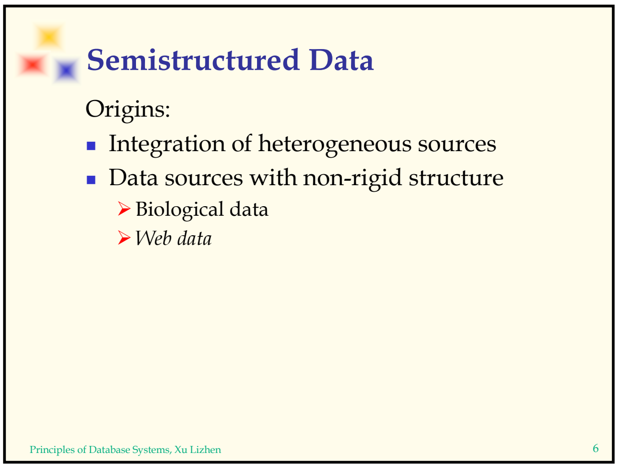
A recommendation from the W3C

- XML = data
- XML generated by applications
- XML consumed by applications
- Easy access: across platforms, organizations



Paradigm Shift on the Web

- From documents (HTML) to data (XML)
- From information retrieval to data management
- For databases, also a paradigm shift:
  - from relational model to semistructured data
  - from data processing to data/query translation
  - from storage to transport

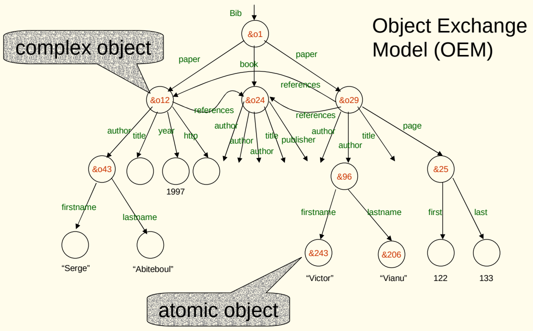


Semistructured Data

Origins:

- Integration of heterogeneous sources
- Data sources with non-rigid structure
  - Biological data
  - *Web data*

The Semistructured Data Model



Object Exchange Model (OEM)

Syntax for Semistructured Data

```
Bib: &o1 { paper: &o12 { … },
           book:  &o24 { … },
           paper: &o29
              { author: &o52 "Abiteboul",
                author: &o96 { firstname: &243 "Victor",
                               lastname: &o206 "Vianu"},
                title: &o93 "Regular path queries with constraints",
                references: &o12,
                references: &o24,
                pages: &o25 { first: &o64 122, last: &o92 133}
              }
         }
```

Observe: Nested tuples, set-values, oids !

Syntax for Semistructured Data

May omit oids:

```
{ paper: { author: "Abiteboul",
           author: { firstname: "Victor",
                     lastname: "Vianu"},
           title: "Regular path queries …",
           page: { first: 122, last: 133 }
         }
}
```

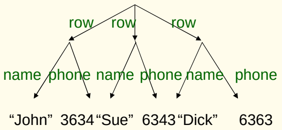
Characteristics of Semistructured Data

- Missing or additional attributes
- Multiple attributes
- Different types in different objects
- Heterogeneous collections

Self-describing, irregular data, no a priori structure

Comparison with Relational Data

name	phone
John	3634
Sue	6343
Dick	6363



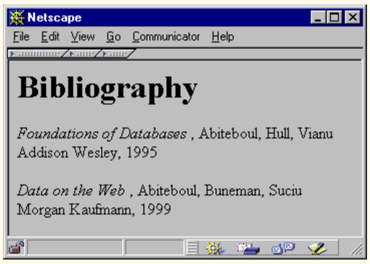
```
{ row: { name: "John", phone: 3634 },
  row: { name: "Sue",  phone: 6343 },
  row: { name: "Dick", phone: 6363 }
}
```

XML

- A W3C standard to complement HTML
- Origins: Structured text SGML
  - ➢ Large-scale electronic publishing
  - ➢ Data exchange on the web
- Motivation:
  - ➢ HTML describes presentation
  - ➢ XML describes content
- http://www.w3.org/TR/2000/REC-xml-20001006 (version 2, 10/2000)

HTML4.0 $\in$ XML $\subset$ SGML

From HTML to XML



Bibliography

Foundations of Databases , Abiteboul, Hull, Vianu
Addison Wesley, 1995

Data on the Web , Abiteboul, Buneman, Suciu
Morgan Kaufmann, 1999

HTML describes the presentation

HTML

```
<h1> Bibliography </h1>
<p> <i> Foundations of Databases </i>
        Abiteboul, Hull, Vianu
        <br> Addison Wesley, 1995
<p> <i> Data on the Web </i>
        Abiteboul, Buneman, Suciu
        <br> Morgan Kaufmann, 1999
```

XML

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

XML describes the content

Why are we DB'ers interested?

- It's data, stupid. That's us.
- Proof by Google:
  - database+XML – 1,940,000 pages.
- Database issues:
  - How are we going to model XML? (graphs)
  - How are we going to query XML? (XQuery)
  - How are we going to store XML (in a relational database? object-oriented? native?)
  - How are we going to process XML efficiently? (many interesting research questions !)

Document Type Descriptors

- Sort of like a schema but not really.

```
<!ELEMENT Book (title, author*) >

<!ELEMENT title #PCDATA>
<!ELEMENT author (name, address,age?)>

<!ATTLIST Book id ID #REQUIRED>
<!ATTLIST Book pub IDREF #IMPLIED>
```

- Inherited from SGML DTD standard
- BNF grammar establishing constraints on element structure and content
- Definitions of entities

Shortcomings of DTDs

Useful for documents, but not so good for data:

- Element name and type are associated globally
- No support for structural re-use
  - Object-oriented-like structures aren't supported
- No support for data types
  - Can't do data validation
- Can have a *single* key item (ID), but:
  - No support for multi-attribute keys
  - No support for foreign keys (references to other keys)
  - No constraints on IDREFs (reference *only* a Section)

XML Schema

- In XML format
- Element names and types associated locally
- Includes primitive data types (integers, strings, dates, etc.)
- Supports value-based constraints (integers > 100)
- User-definable structured types
- Inheritance (extension or restriction)
- Foreign keys
- Element-type reference constraints

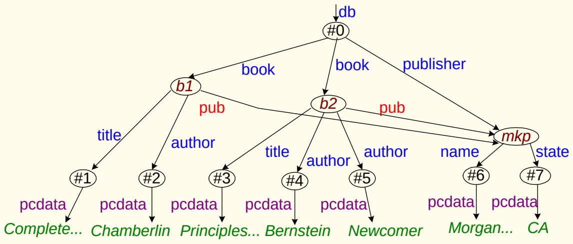
Sample XML Schema

```
<schema version=“1.0” xmlns=“http://www.w3.org/1999/XMLSchema”>
<element name=“author” type=“string” />
<element name=“date” type = “date” />
<element name=“abstract”>
  <type>
    …
  </type>
</element>
<element name=“paper”>
  <type>
    <attribute name=“keywords” type=“string”/>
    <element ref=“author” minOccurs=“0” maxOccurs=“*” />
    <element ref=“date” />
    <element ref=“abstract” minOccurs=“0” maxOccurs=“1” />
    <element ref=“body” />
  </type>
</element>
</schema>
```

Important XML Standards

- XSL/XSLT: presentation and transformation standards
- RDF: resource description framework (meta-info such as ratings, categorizations, etc.)
- Xpath/Xpointer/Xlink: standard for linking to documents and elements within
- Namespaces: for resolving name clashes
- DOM: Document Object Model for manipulating XML documents
- SAX: Simple API for XML parsing
- XQuery: query language

XML Data Model (Graph)



Issues:

- Distinguish between attributes and sub-elements?
- Should we conserve order?

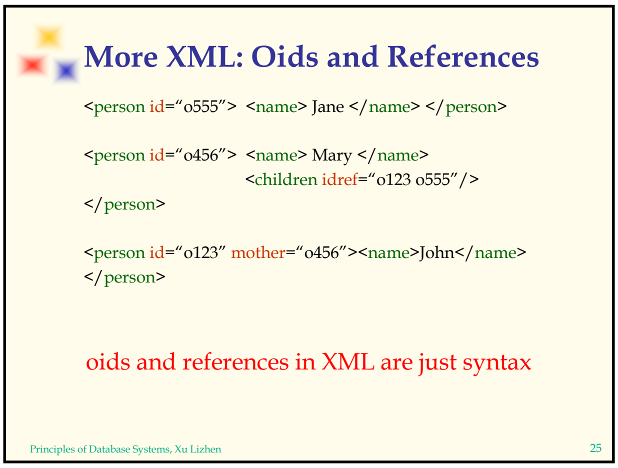
XML Terminology

- Tags: book, title, author, …
  - start tag: <book>, end tag: </book>
- Elements: <book>…<book>,<author>…</author>
  - elements can be nested
  - empty element: <red></red> (Can be abbrv. <red/>)
- XML document: Has a single root element
- Well-formed XML document: Has matching tags
- Valid XML document: conforms to a schema

More XML: Attributes

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

Attributes are alternative ways to represent data



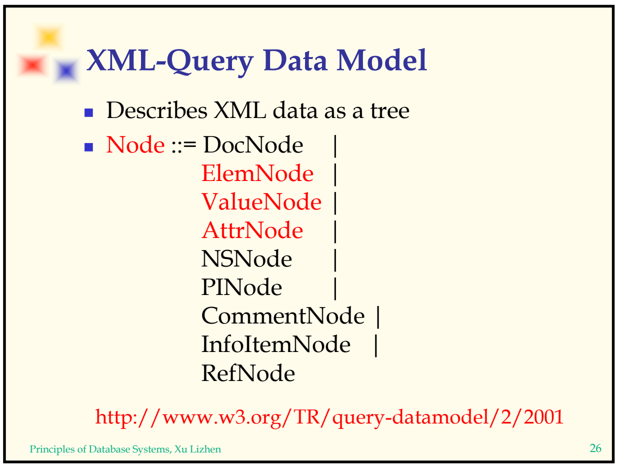
More XML: Oids and References

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

<person id="o456"> <name> Mary </name>
                        <children idref="o123 o555"/>
</person>

<person id="o123" mother="o456"><name>John</name>
</person>
```

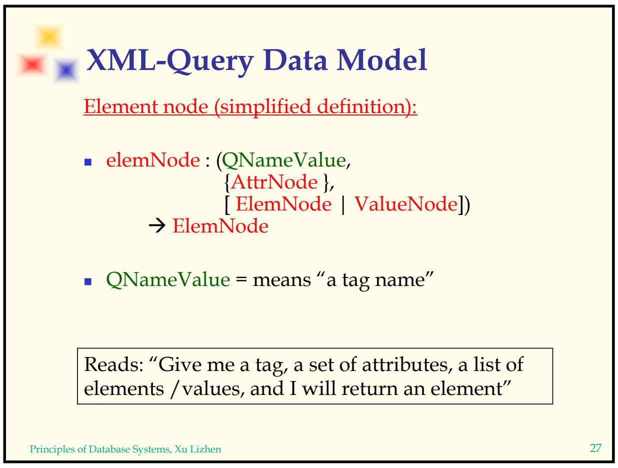
oids and references in XML are just syntax



XML-Query Data Model

- Describes XML data as a tree
- Node ::= DocNode |
  ElemNode |
  ValueNode |
  AttrNode |
  NSNode |
  PINode |
  CommentNode |
  InfoItemNode |
  RefNode

http://www.w3.org/TR/query-datamodel/2/2001

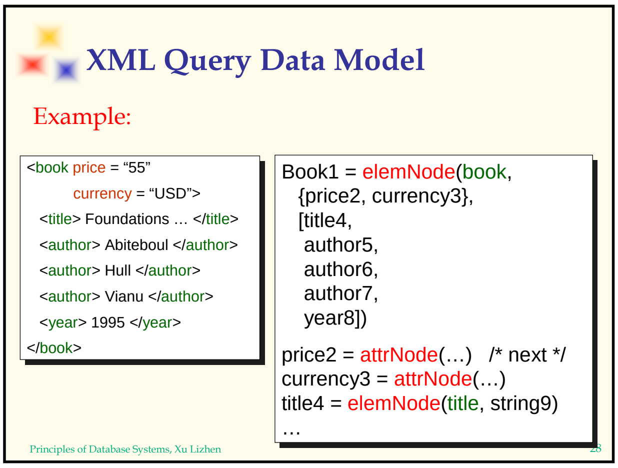


XML-Query Data Model

Element node (simplified definition):

- elemNode : (QNameValue,
  {AttrNode },
  [ElemNode | ValueNode])
  → ElemNode
- QNameValue = means "a tag name"

Reads: "Give me a tag, a set of attributes, a list of elements /values, and I will return an element"

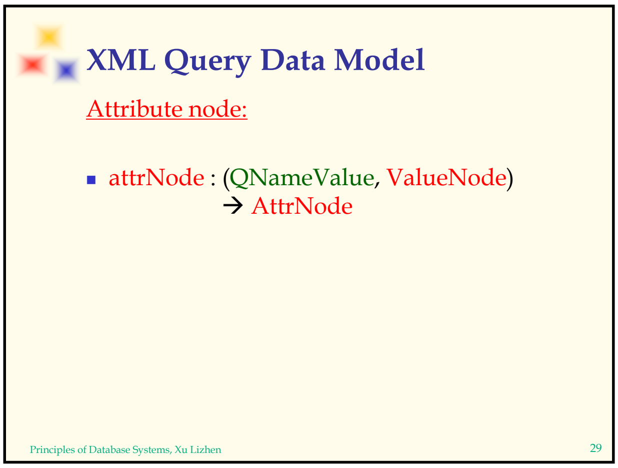


XML Query Data Model

Example:

```
<book price = "55"
      currency = "USD">
  <title> Foundations … </title>
  <author> Abiteboul </author>
  <author> Hull </author>
  <author> Vianu </author>
  <year> 1995 </year>
</book>
```

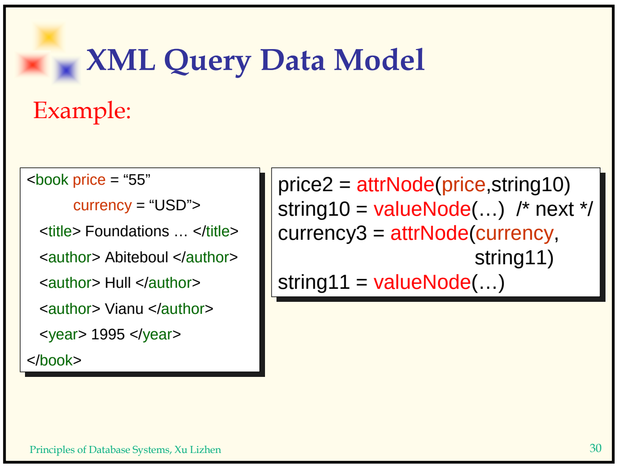
```
Book1 = elemNode(book,
   {price2, currency3},
   [title4,
    author5,
    author6,
    author7,
    year8])
price2 = attrNode(…)   /* next */
currency3 = attrNode(…)
title4 = elemNode(title, string9)
…
```



XML Query Data Model

Attribute node:

- attrNode : (QNameValue, ValueNode)
  → AttrNode



XML Query Data Model

Example:

```
<book price = "55"
      currency = "USD">
  <title> Foundations … </title>
  <author> Abiteboul </author>
  <author> Hull </author>
  <author> Vianu </author>
  <year> 1995 </year>
</book>
```

```
price2 = attrNode(price,string10)
string10 = valueNode(…)  /* next */
currency3 = attrNode(currency,
                     string11)
string11 = valueNode(…)
```

XML Query Data Model

Value node:

- ValueNode = StringValue |
  BoolValue |
  FloatValue …
- stringValue : string → StringValue
- boolValue : boolean → BoolValue
- floatValue : float → FloatValue

XML Query Data Model

Example:

```
<book price = “55”
      currency = “USD”>
   <title> Foundations … </title>
   <author> Abiteboul </author>
   <author> Hull </author>
   <author> Vianu </author>
   <year> 1995 </year>
</book>
```

```
price2 = attrNode(price,string10)
string10 = valueNode(stringValue(“55”))
currency3 = attrNode(currency, string11)
string11 = valueNode(stringValue(“USD”))

title4 = elemNode(title, string9)
string9 =
valueNode(stringValue(“Foundations…”))
```

XML vs. Semistructured Data

- Both described best by a graph
- Both are schema-less, self-describing
- XML is ordered, ssd is not
- XML can mix text and elements:

  <talk> Making Java easier to type and easier to type
  <speaker> Phil Wadler </speaker>
  </talk>
- XML has lots of other stuff: attributes, entities, processing instructions, comments

Management of XML and Semistructured Data

Based upon slides by Dan Suciu

Path Expressions

Examples:

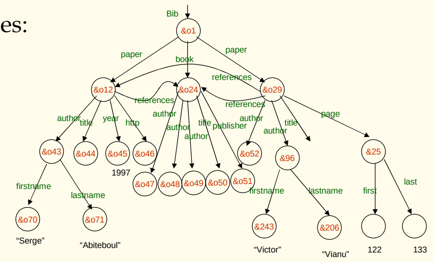
- Bib.paper
- Bib.book.publisher
- Bib.paper.author.lastname

Given an OEM instance, the *value* of a path expression *p* is a set of objects

Path Expressions

Examples:

DB =

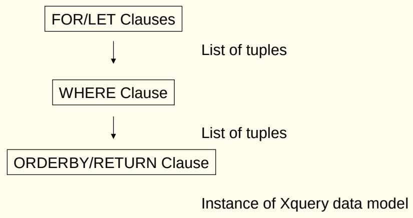


Bib.paper={&o12,&o29}
Bib.book.publisher={&o51}
Bib.paper.author.lastname={&o71,&206}

XQuery

Summary:

- FOR-LET-WHERE-ORDERBY-RETURN = FLWOR



FOR/LET Clauses

List of tuples

WHERE Clause

List of tuples

ORDERBY/RETURN Clause

Instance of Xquery data model

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 v.s. LET

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

Returns:

```
<result> <book>...</book></result>
<result> <book>...</book></result>
<result> <book>...</book></result>
...
```

```
LET $x IN document("bib.xml")/bib/book
RETURN <result> $x </result>
```

Returns:

```
<result> <book>...</book>
         <book>...</book>
         <book>...</book>
         ...
</result>
```

Path Expressions

- Abbreviated Syntax
  - ➢ /bib/paper[2]/author[1]
  - ➢ /bib//author
  - ➢ paper[author/lastname="Vianu"]
  - ➢ /bib/(paper|book)/title
- Unabbreviated Syntax
  - ➢ child::bib/descendant::author
  - ➢ child::bib/descendant-or-self::*/child::author
  - ➢ parent, self, descendant-or-self, attribute

XQuery

Find all book titles published after 1995:

```
FOR $x IN document("bib.xml")/bib/book
WHERE $x/year > 1995
RETURN $x/title
```

Result:

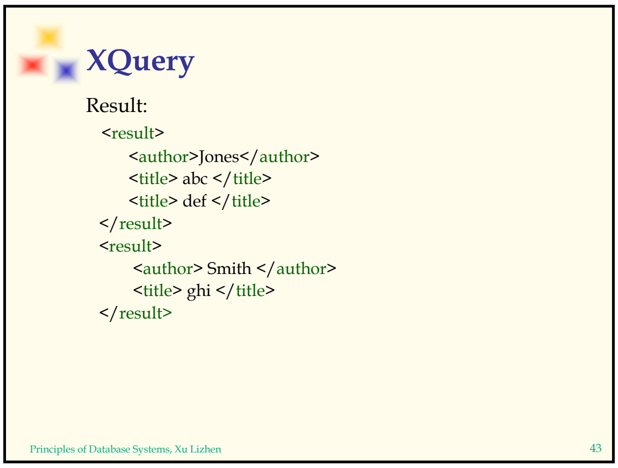
```
<title> abc </title>
<title> def </title>
<title> ghi </title>
```

XQuery

For each author of a book by Morgan Kaufmann, list all books she published:

```
FOR $a IN distinct(document("bib.xml")
                   /bib/book[publisher="Morgan Kaufmann"]/author)
RETURN <result>
           $a,
           FOR $t IN /bib/book[author=$a]/title
           RETURN $t
       </result>
```

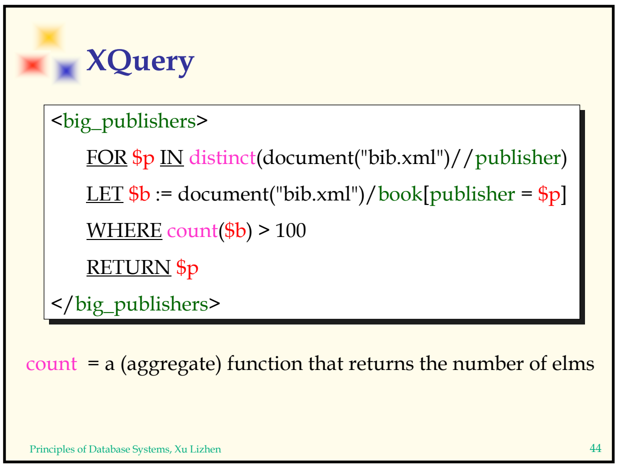
distinct = a function that eliminates duplicates



XQuery

Result:

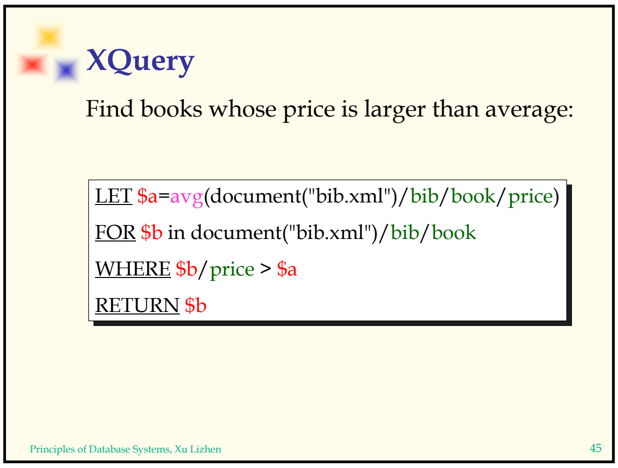
```
<result>
    <author>Jones</author>
    <title> abc </title>
    <title> def </title>
</result>
<result>
     <author> Smith </author>
     <title> ghi </title>
</result>
```



XQuery

```
<big_publishers>
    FOR $p IN distinct(document("bib.xml")//publisher)
    LET $b := document("bib.xml")/book[publisher = $p]
    WHERE count($b) > 100
    RETURN $p
</big_publishers>
```

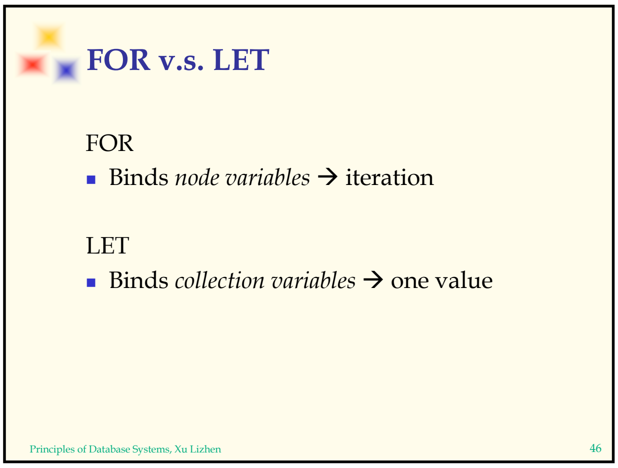
count = a (aggregate) function that returns the number of elms



XQuery

Find books whose price is larger than average:

```
LET $a=avg(document("bib.xml")/bib/book/price)
FOR $b in document("bib.xml")/bib/book
WHERE $b/price > $a
RETURN $b
```



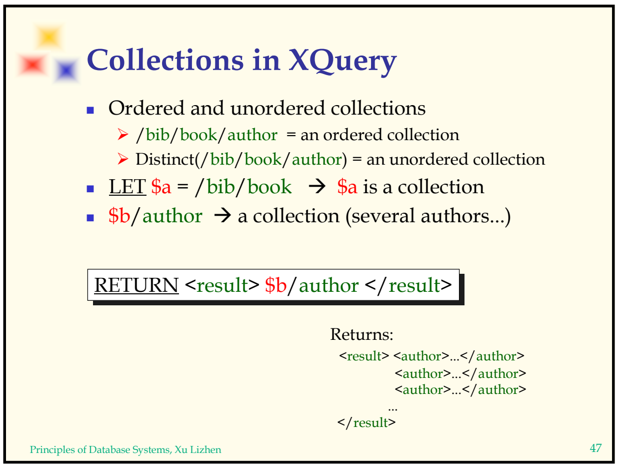
FOR v.s. LET

FOR

- Binds *node variables* → iteration

LET

- Binds *collection variables* → one value



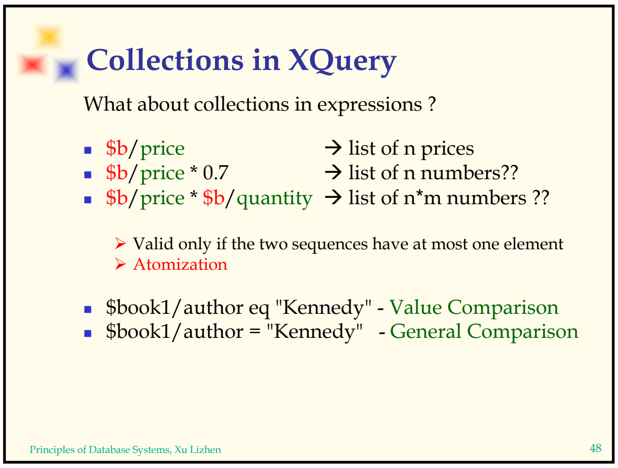
Collections in XQuery

- Ordered and unordered collections
  - /bib/book/author = an ordered collection
  - Distinct(/bib/book/author) = an unordered collection
- LET $a = /bib/book → $a is a collection
- $b/author → a collection (several authors...)

```
RETURN <result> $b/author </result>
```

Returns:

```
<result> <author>...</author>
         <author>...</author>
         <author>...</author>
         ...
</result>
```



Collections in XQuery

What about collections in expressions ?

- $b/price → list of n prices
- $b/price * 0.7 → list of n numbers??
- $b/price * $b/quantity → list of n*m numbers ??
  - Valid only if the two sequences have at most one element
  - Atomization
- $book1/author eq "Kennedy" - Value Comparison
- $book1/author = "Kennedy" - General Comparison

Sorting in XQuery

```
<publisher_list>
   FOR $p IN distinct(document("bib.xml")//publisher)
   ORDERBY  $p
   RETURN <publisher> <name> $p/text() </name> ,
                  FOR $b IN document("bib.xml")//book[publisher = $p]
                  ORDERBY $b/price DESCENDING
                  RETURN <book>
                                    $b/title ,
                                    $b/price
                             </book>
              </publisher>
</publisher_list>
```

If-Then-Else

```
FOR $h IN //holding
ORDERBY $h/title
RETURN <holding>

                $h/title,

                IF $h/@type = "Journal"

                       THEN $h/editor

                ELSE $h/author

          </holding>
```

Existential Quantifiers

```
FOR $b IN //book
WHERE SOME $p IN $b//para SATISFIES
   contains($p, "sailing")
   AND contains($p, "windsurfing")
RETURN $b/title
```

Universal Quantifiers

```
FOR $b IN //book
WHERE EVERY $p IN $b//para SATISFIES
   contains($p, "sailing")
RETURN $b/title
```

Other Stuff in XQuery

- If-then-else
- Universal and existential quantifiers
- Sorting
- Before and After
  - for dealing with order in the input
- Filter
  - deletes some edges in the result tree
- Recursive functions

Group-By in Xquery ??

- No GROUPBY currently in XQuery
- A recent proposal (next)
  - What do YOU think ?

Group-By in Xquery ??

```
FOR $b IN document("http://www.bn.com")/bib/book,
      $y IN $b/@year
WHERE $b/publisher="Morgan Kaufmann"
RETURN   GROUPBY $y
              WHERE count($b) > 10
              IN <year> $y </year>
```

← with GROUPBY

Equivalent SQL →

```
SELECT year
FROM Bib
WHERE Bib.publisher="Morgan Kaufmann"
GROUPBY year
HAVING count(*) > 10
```

Group-By in Xquery ??

```
FOR $b IN document("http://www.bn.com")/bib/book,
      $a IN $b/author,
      $y IN $b/@year
RETURN  GROUPBY $a, $y
             IN <result> $a,
                        <year> $y </year>,
                        <total> count($b) </total>
                  </result>
```

← with GROUPBY

Without GROUPBY →

```
FOR $a IN document(“http://www.bn.com”)/ bib/book/author,

      $y IN $a/../@year

LET $b = document("http://www.bn.com")/bib/book[author=$a,@year=$y]
RETURN <result> $a,
                        <year> $y </year>,
                        <total> count($b) </total>
                </result>
```

Correct if the GROUPBY is node-identity based
Not equivalent if the GROUPBY is value-based

Group-By in Xquery ??

```
FOR $b IN document("http://www.bn.com")/bib/book,
      $a IN $b/author,
      $y IN $b/@year
RETURN  GROUPBY $a, $y
             IN <result> $a,
                        <year> $y </year>,
                        <total> count($b) </total>
                  </result>
```

← with GROUPBY

Without GROUPBY →

```
FOR $a IN distinct(document(“http://www.bn.com”)/ bib/book/author)

      $y IN distinct(document(“http://www.bn.com”)/bib/book/@year)

LET $b = document("http://www.bn.com")/bib/book[author=$a,@year=$y]
RETURN

            IF count($b) > 0

              THEN

                 <result> $a,
                        <year> $y </year>,
                        <total> count($b) </total>
                 </result>
```

Group-By in Xquery ??

```
FOR $b IN document("http://www.bn.com")/bib/book,
      $a IN $b/author,
      $y IN $b/@year
RETURN  GROUPBY $a, $y
             IN <result> $a,
                        <year> $y </year>,
                        <total> count($b) </total>
                  </result>
```

← with GROUPBY

Without GROUPBY →

```
FOR $Tup IN distinct (FOR $b IN document("http://www.bn.com")/bib,
                                        $a IN $b/author,
                                        $y IN $b/@year
                                  RETURN <Tup> <a> $a </a> <y> $y </y> </Tup>),
      $a IN $Tup/a/node(),
      $y IN $Tup/y/node()
  LET $b = document("http://www.bn.com")/bib/book[author=$a,@year=$y]
  RETURN <result> $a,
                        <year> $y </year>,
                        <total> count($b) </total>
                  </result>
```

Group-By in Xquery ??

```
FOR $b IN document("http://www.bn.com")/bib/book,
      $a IN $b/author,
      $y IN $b/@year,
      $t IN $b/title,
      $p IN $b/publisher
RETURN
    GROUPBY $p, $y
    IN <result> $p,
            <year> $y </year>,
            GROUPBY $a
             IN <authorEntry>
                      $a,
                      GROUPBY $t
                      IN $t
                  <authorEntry>
        </result>
```

← Nested GROUPBY’s