

# COMP67340 Introduction to XML Databases

**Goran Nenadic**

School of Computer Science  
University of Manchester

1

## Aims

- Understand the need for managing semi-structured data
- Learn basic principles of design, storage, querying and retrieval in XML DBs
- Understand problems and challenges of XML databases
- Understand how XML can be used for data integration and sharing

2

## Plan

- Introduction
  - need for XML
- XML overview
  - syntax, DTD, XSD
- XML and data modelling
- Storing and querying XML data
  - XML-enabled and XML-native databases
  - XPath and XQuery
- Examples and applications

3

## Introduction – integration

- Different databases differ in:
  - model (relational, object-oriented)
  - schema (normalised vs. un-normalised)
  - terminology
    - e.g. are consultants employees?
  - conventions (meters versus feet)
  - etc.

4

continued

## Introduction – integration

- Example: every pub has a database
  - One may use a relational DBMS; another keeps the menu in an MS-Word document.
  - One stores phones of distributors, another does not.
  - One distinguishes ales from other beers, another does not.
  - One counts beer inventory by bottles, another by cases.

5

continued

## Introduction – integration

- Two solutions: data-warehousing and mediation
- **Warehousing**
  - make copies of the data sources at a central site and transform it to a **common schema**
- **Mediation**
  - create a view of all sources, as if they were integrated
  - Answer a view query by translating it to terminology of the sources and querying them

6

## Introduction – integration

- Neither approach is flexible
- Can we represent the data from independent sources more flexibly than either relational or object-oriented models do?
- Think of entities/objects, but with the type of each entity/object, not only that of its “class.”
- Labels to indicate meaning of substructures
  - rich syntax for description of individual entities
  - can also help with multimedia data

7

## Introduction – integration

- Relational data does not have a “syntax”
  - I can’t “give” you my relational database (if we don’t use the same DBMS and share the schema)
  - need to import it from other syntax, like CSV (comma-separated-values)
- We need a **flexible model** that supports rich syntax for complex data, and
  - can map any (existing) data into it
  - can store data in files, so exchange on the Web, etc.
  - query it directly

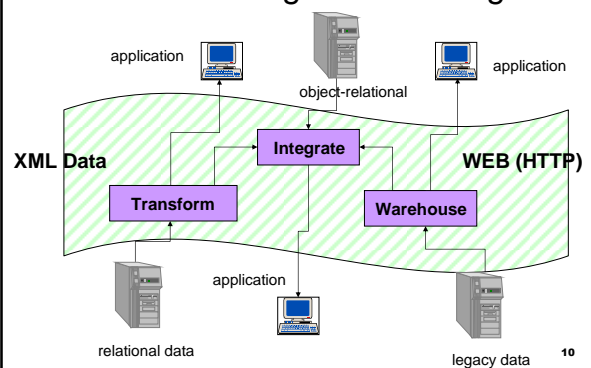
8

## Solution: XML

- XML = eXtensible Mark-up Language
- Entity = Data + Mark-up
  - data entities can be strings, images, video, pubs, trains, visits, etc.
  - mark-up = assign annotation to data
    - explicit “structure”, “semantics”, “meaning”
- XML 1.0
  - recommendation from W3C, 1998

9

## XML data sharing and exchange



10

## XML overview

11

## HTML vs. XML

- HTML uses tags for formatting (e.g., “italic”)
  - describes the layout
- XML uses tags for structure and semantics (e.g., “this is an address, and it contains house number, street, and postcode”)

12

## XML: main concepts

- XML document
- Schema descriptions
  - defined by
    - DTD = Document Type Definition
    - XSD = XML Schema Definition
- Elements, including their
  - structure, and
  - attributes (describe elements)

13

continued

## XML: schema and instances

- A schema is a model for describing the **structure** of data
- The actual document or data that is represented through the schema is called an “**instance**”
- Key idea: create tag sets (i.e. schema) for a domain (e.g., multimedia, genomics, pubs), and translate all data into properly tagged XML documents

14

## XML example (1)

```
<addrBook>
  <person NI =“111-22-3333”>
    <name> Caesar </name>
    <greet> Caesar Imperator </greet>
    <addr> The Capitol </addr>
    <addr> Rome, OH 98765 </addr>
    <tel> (321) 786 2543 </tel>
    <fax> (321) 786 2543 </fax>
    <email> jc@forum.rome.org </email>
  </person>
</addrBook>
```

15

## XML example (2)

```
<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>

<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>
```

16

## 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> or <red/>
- **attributes**: add additional features to data  
    <book year="1992">...  
alternative: <book><year>1992</year>...

17

example

## XML terminology

- **Well formed** XML document
  - if it has matching tags (start/end)
  - tags are properly nested
  - single root element
  - and more constraints, e.g. on names
- **Valid** XML document
  - involves a DTD (Document Type Definition) or XSD which limits the labels and gives a grammar for their use

18

## Document type definition (DTD)

- Essentially, a context-free grammar for describing XML tags and their nesting
- Each domain of interest (e.g., electronic components, bars-beers-drinkers) creates a DTD that describes all the documents (data) this group will share
- an XML document **may** have a DTD
  - not obligatory
  - but, validation is useful in data exchange

19

example

## A simple DTD

```
<!DOCTYPE company [
  <!ELEMENT company ((person|product)*)>
  <!ELEMENT person (NI, name, office, phone?)>
  <!ELEMENT NI (#PCDATA)>
  <!ELEMENT name (#PCDATA)>
  <!ELEMENT office (#PCDATA)>
  <!ELEMENT phone (#PCDATA)>
  <!ELEMENT product (pid, name, description?)>
  <!ELEMENT pid (#PCDATA)>
  <!ELEMENT description (#PCDATA)>
]>
```

20

## A simple DTD: an instance

Example of a valid XML document

```
<company>
  <person> <NI> 123456789 </NI>
    <name> John </name>
    <office> B432 </office>
    <phone> 1234 </phone>
  </person>
  <person> <NI> 987654321 </NI>
    <name> Jim </name>
    <office> B123 </office>
  </person>
  <product> ... </product>
</company>
```

21

## DTD structure

- Root and other elements
 

```
<!DOCTYPE <root tag> [
  <!ELEMENT <name> ( <components> )
  <!ELEMENT <name> ( <components> )
  <!ATTRIBUTE <el> <name> <type> <value> )
  ...
]>
```
- Description of an element consists of its *name* (tag), and a parenthesized description of any nested tags (= content), and its attributes

22

## DTD: elements

```
<!ELEMENT tag (CONTENT)>
```

content model

- Content model
  - complex = a regular expression over other elements
  - text-only = #PCDATA (parsed character data)
  - character data = #CDATA (character data)
  - empty (no other elements) = EMPTY
  - any = ANY
  - mixed content = (#PCDATA | A | B | C)\*

PCDATA is text that will be parsed by a parser.  
CDATA is text that will NOT be parsed by a parser.

23

continued

## DTD: elements

- Complex elements: sub-tags must appear in **order shown**
- A tag may be followed by a symbol to indicate its multiplicity (regular expressions):
  - \* = zero or more
  - + = one or more
  - ? = zero or one
- Symbol | can connect alternative sequences of tags

24

example

## DTD: elements

sequence

DTD

```
<!ELEMENT name (firstName, lastName)>
```

XML

```
<name>
  <firstName> ... </firstName>
  <lastName> ... </lastName>
</name>
```

optional

```
<!ELEMENT name (firstName?, lastName)>
```

```
<name>
  <lastName> ... </lastName>
</name>
```

star (repeated occurrence)

```
<!ELEMENT person (name, phone*)>
```

```
<person>
  <name> ... </name>
  <phone> ... </phone>
  <phone> ... </phone>
  <phone> ... </phone>
  ...
</person>
```

alternation

```
<!ELEMENT person (name, (phone|email))>
```

```
<person>
  <name> ... </name>
  <email> ... </email>
</person>
```

## DTD: attributes

```
<!ATTLIST element-name attribute-name attribute-type default-value>
```

| Value               | Explanation                            |
|---------------------|--|
| CDATA               | The value is character data            |
| (eval   eval   ...) | The value must be an enumerated value  |
| ID                  | The value is a unique id               |
| IDREF               | The value is the id of another element |
| IDREFS              | The value is a list of other ids       |
| ENTITY              | The value is an entity                 |
| ENTITIES            | The value is a list of entities        |
| xml:                | The value is predefined                |
| ...                 |  |

| Value          | Explanation   |
|----------------|---|
| #DEFAULT value | The attribute has a default value                   |
| #REQUIRED      | The attribute value must be included in the element |
| #IMPLIED       | The attribute does not have to be included          |
| #FIXED value   | The attribute value is fixed                        |

26

example

## DTD: attributes

- DTD**

```
<!ELEMENT person (NI, name, office, phone?)>
<!ATTLIST person age CDATA #REQUIRED "18"
  birthdate CDATA #IMPLIED
  nationality CDATA #FIXED "UK"
  gender (male|female) "female">
```

mandatory

optional

default
- Document (instance)**

```
<person age="24" nationality="UK" gender="male">
  <NI> ... </NI> ... <phone> ... </phone>
</person>
```

enumeration

27

## DTD: ID and IDREF attributes

- IDREF type allows the value of one attribute to be an element elsewhere in the document provided that the value of the IDREF is the ID value of the referenced element
- Example**

```
<!ELEMENT part (#PCDATA)>
<!ATTLIST part id ID #REQUIRED>
<!ELEMENT partref EMPTY>
<!ATTLIST partref idref IDREF #REQUIRED>
```

DTD

Note: empty element

document instance

```
<part id="bolt-1573">...</part>
...
<part id="nut-44456">This nut is compatible with <partref idref="bolt-1573"/></part>
```

28

## DTD: entities

- DTD**

```
<!ENTITY address SYSTEM "address.xml">
<!ENTITY name "<name>Tim Berners Lee</name>">
```
- Document**

```
<celebrity>
  &name;
  &address;
</celebrity>
```

internal entity

external entity

29

## Inclusion of DTD in documents

External DTD declaration

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE test PUBLIC "-//Test AG//DTD test V1.0//EN"
  SYSTEM "http://www.test.org/test.dtd">
<test> "test" is a document element </test>
```

Internal DTD declaration

```
<!DOCTYPE test [ <!ELEMENT test EMPTY > ]>
<test/>
```

Mixed usage

```
<!DOCTYPE test SYSTEM "http://www.test.org/test.dtd" [
  <!ENTITY hello "hello world">
]>
<test>&hello;</test>
```

30

## Example

```
<? XML VERSION = "1.0" STANDALONE = "no" ?>
<!DOCTYPE Bars [
  <!ELEMENT BARS (BAR*)>
  <!ELEMENT BAR (NAME, BEER+)>
  <!ELEMENT NAME (#PCDATA)>
  <!ELEMENT BEER (NAME, PRICE)>
  <!ELEMENT PRICE (#PCDATA)>
]>
<BARS>
  <BAR><NAME>Joe's Bar</NAME>
  <BEER><NAME>Bud</NAME> <PRICE>2.50</PRICE></BEER>
  <BEER><NAME>Miller</NAME> <PRICE>3.00</PRICE></BEER>
</BAR>
<BAR> ...
</BARS>
```

DTD

example: valid document

31

## DTD: problems

- DTDs define structure, but do not provide type constraints
- DTDs do not allow un-order elements
  - sub-tags must appear in order shown
- DTDs have their own syntax
  - not XML-like
  - not easy to process/parse

important

32

## XML schema definition (XSD)

- XSD = XML schema definition
- XML schema is a W3C standard for data modelling using XML
- An XML schema definition is itself an XML document – there is an XML schema for XML Schema!

33

continued

## XML schema definition (XSD)

- XSD can specify
  - which elements are mandatory/optional
  - which attributes are mandatory/optional
  - element/attribute **types**
  - **cardinalities**
  - **relative** ordering

34

## XSD example

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="Employee"
    minOccurs="0"
    maxOccurs="unbounded">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="NI" type="xsd:string">
        <xsd:element name="Name" type="xsd:string"/>
        <xsd:element name="DateOfBirth" type="xsd:date"/>
        <xsd:element name="EmployeeType" type="xsd:string"/>
        <xsd:element name="Salary" type="xsd:long"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

name spaces

cardinality

data type

35

## XML namespaces

- XML namespace is a collection of names identified by a prefix
- Elements and attributes from different namespaces are distinguished using the syntax **prefix:name**, to give a qualified name
  - **name** is the "normal" element/attribute name
  - **prefix** is a name given to a namespace
- A namespace is a URL acting as a unique string
- Namespaces support simultaneous use of definitions from multiple files

36

## XSD elements

- Elements are defined  
`<element name = "the_name" ...>`
- Attributes associated with elements:
  - **type**: specifies the kind of content that an element with no attributes or sub-elements can have. Default imposes no constraints.
  - **minOccurs**, **maxOccurs**: the number of times an element can occur. A value of unbounded allows open-ended cardinality. Default is once and only once.

37

## XSD element types

- There are many built-in types
  - *string, integer, positiveInteger, negativeInteger, short, long, date, dateTime, time, id, idref, anyURI*
- Complex elements
  - *sequence*: the sub-elements must appear in the given order
  - *choice*: a selection is made from the sub-elements
  - both sequence and choice can have *minOccurs* and *maxOccurs*

38

## XSD complex types: syntax

example

```
<xs:element name="visit">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="name" type="..." />
      <xs:element name="time" type="..." />
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

39

## XSD attributes

- Attributes can be defined within complex types
- Attributes are optional unless *use="required"*

```
<xs:complexType name="TrainType">
  <xs:sequence>
    <xs:element name="tno" type="xs:string" />
    ...
  </xs:sequence>
  <xs:attribute name="engine" type="xs:string" />
</xs:complexType>
```

40

## XSD IDs and IDREFs

example

```
<xs:element name="station">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="city" type="xs:string" />
      <xs:element name="name" type="xs:ID" />
    </xs:sequence>
  </xs:complexType>
</xs:element>
...
<xs:element name="visit">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="name" type="xs:IDREF" />
      <xs:element name="time" type="xs:time" />
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

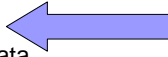
41

## XSD: summary

- XSD provides a wide range of modelling facilities for defining XML documents
- Can be used to define both **structure** and **types**
- However, intended mainly for machine processing
  - easy to parse/validate
  - human readability not considered!
  - typically much longer than related DTDs

42

## Plan

- Introduction
  - need for XML
- XML overview
  - syntax, DTD, XSD
- **XML and data modelling** 
- Storing and querying XML data
- Examples and applications

43

## Data modelling with XML

44

## Semi-structured data

- Semi-structured data
  - no strict format, but can have some structure with optional elements and attributes
  - some attributes may be missing, some repeated, and new ones can be added later: change unpredictably
  - ad-hoc, diverse data sources
  - examples
    - HTML document with titles (e.g. <H1>) and paragraphs (<p>), but no information about author(s), date, etc.
    - multimedia objects

45

example

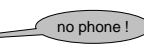
## Semi-structured data

- Missing attributes

```

<person> <name> John</name>
          <phone>1234</phone>
</person>

<person> <name>Joe</name>
</person>
    
```



- Could be represented in a relational table with nulls (-)

| name | phone |
|------|-------|
| John | 1234  |
| Joe  | -     |

46

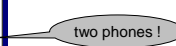
example

## Semi-structured data

- Repeated attributes

```

<person> <name> Mary</name>
          <phone>2345</phone>
          <phone>3456</phone>
</person>
    
```



- Difficult in a table  
(but what about two tables?)

| name | phone |      | ??? |
|------|-------|------|-----|
| Mary | 2345  | 3456 |     |
|      |       |      |     |

47

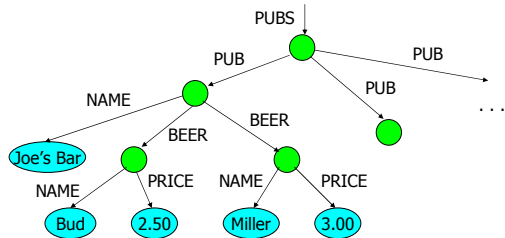
example

## Semi-structured data

- Relational schema and data are separated
  - relational schema: `persons(name,phone)`
- Typically, semi-structured data is **self-describing**
  - mixing data and meta-data (schema)
- Schema elements become part of the data
  - in XML, <person>, <name>, <phone> are part of the data, and are repeated for each entity (as required)
- Semi-structured data can be modeled by trees and/or (directed) graphs

48

## Example: tree/graph model



49

## Data modeling with XML

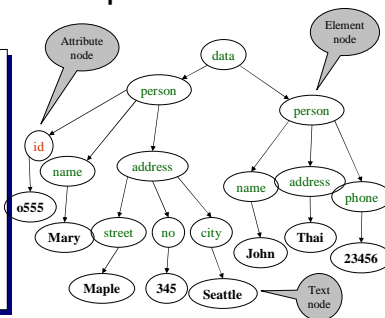
- XML is **self-describing**
  - consequence: XML is much more flexible
- Well-Formed XML with nested tags is exactly the same idea as trees of semi-structured data
- XML also enables non-tree structures, as does the semi-structured data model

XML is a natural choice to model semi-structured data

50

## XML data: tree representation

```
<data>
  <person id="0555" >
    <name> Mary </name>
    <address>
      <street> Maple </street>
      <no> 345 </no>
      <city> Seattle </city>
    </address>
  </person>
  <person>
    <name> John </name>
    <address> Thailand </address>
    <phone> 23456 </phone>
  </person>
</data>
```



51

## Storing and querying XML data

52

## Storing XML data

- XML-data can be stored in the following 3 types of DBMS
  - **traditional relational** database
    - DBMS basically does not 'deal' with XML
    - storage
      - without mapping (XML documents as CLOBs)
      - map an XML schema to relational schema
  - **XML-enabled** database
  - **XML-native** database

53

## XML-enabled databases

- Use object or object-relational model to map XML-schema
  - model XML-elements as classes
  - e.g. element *Person* is modelled as a class *Person*
- Also, new data types provided to support XML
  - **SQL:2003** (SQL/XML) introduced XML extensions including a new native data type (*XML*) and a set of operators
  - DBMS now understands how to deal with XML
  - XMLCLOB, XMLFile (DB2); XMLTYPE (Oracle 9)

54

example

## XML-enabled databases

```
CREATE TABLE mail_user (
  user_name  VARCHAR2(20),
  mailbox    SYS.XMLTYPE
);
```

```
<mailbox>
<inbox>
<email id=1><subject text="Urgent info"/><from .../><attach .../> </email>
<email id=2><subject text="Meeting"/><from .../><attach .../></email>
...
</inbox>
<outbox>
<email id=7><subject text="RE:Urgent"/><from .../><attach .../></email>
...
</outbox>
</mailbox>
```

55

example

## XML-enabled databases

### Querying using SQL-like expressions

*return names of users that have any email in their inbox that contains string "Manchester" in its subject*

```
SELECT  user_name
FROM    mail_user
WHERE   mailbox.extract('/mailbox/inbox/email/subject/text()').getStringVal()
LIKE '%Manchester%'
```

↑

Elements  
(full path)

↑

Attribute

56

## XML-native databases

- Designed to store XML documents
  - do not rely on a relational or object model
  - basic unit of logical design is an XML-document
  - easy/natural storage and access to data
- Support for XML query languages
  - XPath = simple path through the tree
  - XQuery = the SQL of XML
- Example XML-native DBMS
  - Tamino, X-Hive (commercial)
  - dbXML, eXist (open-source)

57

## XML query languages: XPath and XQuery

58

file: bib.xml

## Sample data for queries

```
<bib>
<book> <publisher> Addison-Wesley </publisher>
<author> Serge Abiteboul </author>
<author> <first-name> Rick </first-name>
<last-name> Hull </last-name>
</author>
<author> Victor Vianu </author>
<title> Foundations of Databases </title>
<year> 1995 </year>
</book>

<book price="55">
<publisher> Freeman </publisher>
<author> Jeffrey D. Ullman </author>
<title> Principles of Database and Knowledge Base Systems
</title>
<year> 1998 </year>
</book>
</bib>
```

59

## Data model for XPath

Uses node paths to extract data

60

## XPath: simple expressions

- retrieve *titles* of all books from file "bib.xml"  
`doc("bib.xml")/bib/book/title`  
 result  

```
<title>Foundations of Databases</title>
<title>Principles of Database and Knowledge Base Systems</title>
```
- `/bib/book/year` result  

```
<year> 1995 </year>
<year> 1998 </year>
```

61

example

## XPath: simple expressions

- `/bib/paper/year` result: empty  
(there were no papers)
- retrieving attribute nodes:  
`@price` means that *price* is an attribute  
`/bib/book/@price`  
 result: "55"

62

examples

## XPath: predicates

- Specify restrictions/conditions/constraints

- retrieve all books published before 2000  
`doc("bib.xml")/bib/book[year<2000]`
- retrieve *titles* of all books published before 2000  
`doc("bib.xml")/bib/book[year<2000]/title`

63

examples

## XPath: predicates

- `/bib/book[@price < "60"]`
- `/bib/book[author/@age < "25"]`
- `/bib/book[@price<"55"]/author/lastname`

64

## XPath: retrieving text node values

- get the text value of an element  
`/bib/book/author/text()`

Result: Serge Abiteboul  
 Victor Vianu  
 Jeffrey D. Ullman

[Note: Rick Hull doesn't appear because he has *firstname*, *lastname*]

`/bib/book[publisher/text()="Addison-Wesley"]/title`

65

## XPath: brief summary

|                              |   |
|------------------------------|---|
| <code>bib</code>             | matches a <b>bib</b> element                                  |
| <code>@price</code>          | matches a <b>price</b> attribute                              |
| <code>/</code>               | matches the <b>root</b> element                               |
| <code>/bib</code>            | matches a <b>bib</b> element under <b>root</b>                |
| <code>bib/paper</code>       | matches a <b>paper</b> in <b>bib</b>                          |
| <code>bib/book/@price</code> | matches <b>price</b> attribute in <b>book</b> , in <b>bib</b> |

Also,

|                         |   |
|-------------------------|---|
| <code>paper book</code> | matches a <b>paper</b> or a <b>book</b> |
| <code>*</code>          | matches any element                     |

66

## XQuery: motivation

- XPath's expressivity insufficient
  - no join queries (as in SQL)
  - no quantifiers (as in SQL)
  - no aggregation and functions
  - no changes to the XML structure possible
- XQuery uses XPath to express more complex queries

67

## XQuery: basic FLWOR

FOR ...  
LET...  
WHERE...  
ORDER BY...  
RETURN...

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

Retrieve all book titles published after 1995

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

Result:

<title> Principles of Database and Knowledge Base Systems</title> 68

## XQuery: joins and nesting

For each author of a book by *Addison-Wesley*, list all books they published:

```
FOR $b IN document("bib.xml")/bib,
  $a IN $b/book[publisher/text()='Addison-Wesley']/author
RETURN { $a,
        FOR $t IN $b/book[author/text()=$a/text()]/title
        RETURN $t
      }
```

In the RETURN clause, comma concatenates XML fragments

69

## Sample data for queries (2)

file: stores.xml

```
<stores>
  <store sid = "s282">
    <name>Wiz</name> <phone>555-1234</phone>
    <product pid = "233">
      <name>gizmo plus</name> <price>99.99</price>
      <description>more features</description> <markup>25%</markup>
    </product>
  </store>
  <store sid = "s521">
    <name>Econo-Wiz</name> <phone>555-6543</phone>
    <product pid = "323">
      <name>gizmo</name> <price>22.99</price>
      <description>great</description> <markup>10%</markup>
    </product>
    <product pid = "233">
      <name>gizmo plus</name> <price>99.99</price>
      <description>more features</description> <markup>15%</markup>
    </product>
  </store>
</stores>
```

## Example 1

- Retrieve stores (their IDs) that sell some products with a price higher than 50?

```
FOR $x IN /stores/store[product/price/text() >"50"]
RETURN $x/@sid
```

```
FOR $x IN /stores//store[product/price/text() >"50"]/@sid
RETURN $x
```

```
/stores/store[product/price/text() >"50"]/@sid XPath
```

71

## Example 2

- Retrieve the names of all stores that sell the same products as store "Wiz"

```
FOR $x IN /stores/store[name = "Wiz"]/product
FOR $y IN /stores/store[name<>"Wiz"]
WHERE $x = $y/product
RETURN {$y/name}
```

- Can't be expressed in XPath – no joins!

72

## XQuery: aggregates

**count** = a function that counts  
**avg** = computes the average  
**sum** = computes the sum  
**distinct-values** = eliminates duplicates

Find all books with more than 3 authors

```
FOR $x IN document("bib.xml")/bib/book
WHERE count($x/author)>3
RETURN $x
```

```
FOR $x IN document("bib.xml")/bib/book[count(author)>3]
RETURN $x
```

73

examples

## XQuery: aggregates

Retrieve all authors who published more than 3 books

```
FOR $b IN document("bib.xml")/bib,
  $a IN distinct-values($b/book/author/text())
WHERE count($b/book[author/text()=$a])>3
RETURN { $a }
```

74

## XQuery variables

- **FOR \$x in expr** – binds \$x to each value in expr
  - binds node variables: used for iteration
- **LET \$x := expr** – binds \$x to the entire expr
  - binds **collection** variables: one value

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

collection  
of all prices

75

## XML examples and applications

76

## A complete example

- Provide a simple XML model to describe students, academics and support staff in a school
- Provide a sample document for the School of Computer Science
- Write a (XQuery or XPath) query that retrieves
  - names of all academics who supervise students
  - the names of staff members who share an office with another staff member
  - all students supervised by a professor

77

school.dtd

## School DTD

```
<!DOCTYPE School [
  <!ELEMENT School (Academic | Support | Student)+>
  <!ELEMENT Academic (Office, Position, Salary)>
  <!ATTLIST Academic Name ID #REQUIRED>
  <!ELEMENT Support (Office, Salary)>
  <!ATTLIST Support Name ID #REQUIRED>
  <!ELEMENT Student EMPTY>
  <!ATTLIST Student Name ID #REQUIRED Tutor IDREF #REQUIRED>
  <!ELEMENT Office (#PCDATA)>
  <!ELEMENT Salary (#PCDATA)>
  <!ELEMENT Position ("lecturer"|"senior_lecturer"|"professor")>
]
```

78

## A sample document

```
<?xml version="1.0"?>
<!DOCTYPE School SYSTEM "school.dtd">
<School>
<Academic Name = "Name1">
  <Office>H1</Office><Position>lecturer</Position>
  <Salary>Grade2.1</Salary>
</Academic>
<Academic Name = "Name2">
  <Office>H1</Office><Position>professor</Position>
  <Salary>Grade3.2</Salary>
</Academic>
<Support Name = "Name2"> <Office>H2</Office>
  <Salary>Grade1.7</Salary>
</Support>
<Student Name="Student1" Tutor="Name1"/>
<Student Name="Student2" Tutor="Name1"/>
</School>
```

## Sample queries

- Retrieve all academics who are professors  
`doc("CS.doc")/School/Academic[Position/text()="professor"]`
- Retrieve salaries of all professors  
`/School/Academic[Position/text()="professor"]/Salary`
- Retrieve the names of all professors  
`/School/Academic[Position/text()="professor"]/@Name`

[Note: *name* is an attribute in this example]

80

continued

## Sample queries

- names of all academics who supervise students  
 for \$s in /School/Student  
 for \$a in /School/Academic  
 where \$s[@Tutor] = \$a[@Name]  
 return \$a/@Name

81

continued

## Sample queries

- the names of staff members who share an office with another staff member  
 for \$s1 in /School/Support  
 for \$s2 in /School/Support  
 where \$s1/Office/text() = \$s2/Office/text() AND  
 \$s1/@Name <> \$s2/@Name  
 return \$s1/@Name

82

<http://www.davidmoisan.org/>

## Example 2: a DTD for TV schedule

```
<!DOCTYPE TVSCHEDULE [
  <!ELEMENT TVSCHEDULE (CHANNEL+)>
  <!ELEMENT CHANNEL (BANNER.DAY+)>
  <!ELEMENT BANNER (#PCDATA)>
  <!ELEMENT DAY (DATE,(HOLIDAY|PROGRAMSLOT+)+)>
  <!ELEMENT HOLIDAY (#PCDATA)>
  <!ELEMENT DATE (#PCDATA)>
  <!ELEMENT PROGRAMSLOT (TIME,TITLE,DESCRIPTION?)>
  <!ELEMENT TIME (#PCDATA)>
  <!ELEMENT TITLE (#PCDATA)>
  <!ELEMENT DESCRIPTION (#PCDATA)>
  <!--
  <!ATTLIST TVSCHEDULE NAME CDATA #REQUIRED>
  <!ATTLIST CHANNEL CHAN CDATA #REQUIRED>
  <!ATTLIST PROGRAMSLOT VTR CDATA #IMPLIED>
  <!--
  <!ATTLIST TITLE RATING CDATA #IMPLIED>
  <!--
  <!ATTLIST TITLE LANGUAGE CDATA #IMPLIED>
  -->
]>
```

83

example

## Example 2: a sample document

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE TVSCHEDULE SYSTEM
"http://www1.shore.net/~dmoisan/comp/xml/tvschedule.dtd" [ ]>
<TVSCHEDULE NAME="Salem Access Television">
  <CHANNEL CHAN="3">
    <BANNER>Channel 3 Program Schedule
    Monday, April 27th--Sunday, May 2nd</BANNER>
    <DAY>
      <DATE>Monday, April 27th</DATE>
      <PROGRAMSLOT>
        <TIME>3:00 PM</TIME>
        <TITLE>Salem Recreation Department Basketball</TITLE>
        <DESCRIPTION>Vs. Brookline</DESCRIPTION>
      </PROGRAMSLOT>
    </DAY>
  </CHANNEL>
</TVSCHEDULE>
```

84

## Example 3: multimedia

- XML can be used to describe multimedia objects
- Element and attribute values can be
  - “free” textual descriptions
  - ontological descriptions (controlled vocabulary)
- Multimedia XML databases
  - use a XSD to define a schema for multimedia DB
  - query multimedia XML representations
- Digital libraries (see Lecture 8)
- Semantic Web (RDFs, OWL – see Lecture 9)

85

example

## Multimedia object representation

```
<Description xsi:type="ContentEntityType">
<MultimediaContent xsi:type="VideoType">
  <Video>
    <TemporalDecomposition>
      <VideoSegment>
        <TextAnnotation type="scene" relevance="1" confidence="1">
          <FreeTextAnnotation>Indoors</FreeTextAnnotation>
          <FreeTextAnnotation>Laboratory</FreeTextAnnotation>
          <FreeTextAnnotation>Person_Speaking</FreeTextAnnotation>
        </TextAnnotation>
        <MediaTime>
          <MediaTimePoint>T00 00 00 0F25</MediaTimePoint>
          <MediaIncrDuration mediaTimeUnit="PT1N25F">9 16</MediaIncrDuration>
        </MediaTime>
      </VideoSegment>
    </TemporalDecomposition>
  </Video>
</MultimediaContent>
</Description>
```

86

example

```
<PubmedArticle>
  <MedlineCitation Owner="NLM" Status="MEDLINE">
    <PMID>15542823</PMID>
    <DateCreated>
      <Year>2004</Year>
      <Month>11</Month>
      <Day>15</Day>
    </DateCreated>
    <DateCompleted>
      <Year>2005</Year>
      <Month>04</Month>
      <Day>18</Day>
    </DateCompleted>
    <Article PubModel="Print">
      <Journal>
        <ISSN IssnType="Print">1532-0464</ISSN>
        <JournalIssue CitedMedium="Print">
          <Volume>37</Volume>
          <Issue>6</Issue>
          <PubDate>
            <Year>2004</Year>
            <Month>Dec</Month>
          </PubDate>
        </JournalIssue>
        <Title>Journal of biomedical informatics. </Title>
        <ArticleTitle>Term identification in the biomedical literature.</ArticleTitle>
        <PageInfo>
          <MedlinePage>512-26</MedlinePage>
        </PageInfo>
        <Abstract>
          <AbstractText>Sophisticated information technologies are needed for effective data acquisition and integration from a growing body of the biomedical literature. Successful term identification is key to getting access to the stored literature information, or it is the terms (and their relationships) that convey knowledge across scientific articles. Due to the complexities of a dynamically changing biomedical terminology, term identification has been recognized as the current bottleneck in text mining, and—as a consequence—has become an important research topic both in natural language processing and biomedical communities. This article overviews state-of-the-art approaches in term identification. The process of identifying terms is analyzed through three steps: term recognition, term classification, and term mapping. For each step, main approaches and general trends, along with the major problems, are discussed. By assessing previous work in context of the overall term identification process, the review also tries to delineate needs for future work in the field.</AbstractText>
        </Abstract>
      </Article>
    </PubmedArticle>
```

### Representation of a textual document (1)

87

example

```
<Author ValidTY="Y">
  <LastName>Benedict</LastName>
  <ForeName>Goran</ForeName>
  <Initials>G</Initials>
</Author>
<AuthorList>
  <Language>eng</Language>
  <PublicationTypeList>
    <PublicationType>Journal Article</PublicationType>
  </PublicationTypeList>
</Article>
<MedlineJournalInfo>
  <Country>United States</Country>
  <MedlineTA>J Biomed Inform</MedlineTA>
  <NlmUniqueID>100970413</NlmUniqueID>
</MedlineJournalInfo>
<CitationSubes>IM</CitationSubes>
<MeshHeadingList>
  <MeshHeading>
    <DescriptorName MajorTopicTY="N">Abbreviations</DescriptorName>
  </MeshHeading>
  <MeshHeading>
    <DescriptorName MajorTopicTY="N">Abstracting and Indexing</DescriptorName>
    <QualifierName MajorTopicTY="Y">methods</QualifierName>
  </MeshHeading>
  <MeshHeading>
    <DescriptorName MajorTopicTY="N">Algorithms</DescriptorName>
  </MeshHeading>
  <MeshHeading>
    <DescriptorName MajorTopicTY="N">Animals</DescriptorName>
  </MeshHeading>
  <MeshHeading>
    <DescriptorName MajorTopicTY="N">Artificial Intelligence</DescriptorName>
  </MeshHeading>
  <MeshHeading>
    <DescriptorName MajorTopicTY="N">Computational Biology</DescriptorName>
    <QualifierName MajorTopicTY="Y">methods</QualifierName>
  </MeshHeading>
</MeshHeadingList>
```

### Representation of a textual document (2)

88

## Wrapping up

89

## XML

- Use XML for
  - representation, storage and content-based retrieval of semi-structured data
    - XML-databases
    - e.g. storing and querying multimedia, text, Semantic Web
  - data exchange (for any kind of data)
    - transform relational/object models into XML models
    - however, unlike relational/object databases, an XML database need not have a schema!
    - An XML schema may not be very prescriptive in terms of what can or cannot be stored

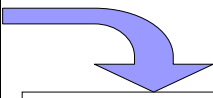
90

example

## From relational to XML data

persons

| name | phone |
|------|-------|
| John | 3634  |
| Sue  | 6343  |
| Dick | 6363  |



```

<persons>
  <row> <name>John</name>
    <phone>3634</phone></row>
  <row> <name>Sue</name>
    <phone>6343</phone>
  <row> <name>Dick</name>
    <phone>6363</phone></row>
</persons>

```

91

## Summary

- Brief overview of XML
  - self-describing
- XML and databases
  - relational, XML-enabled, XML-native
- XPath and XQuery
- Modelling semi-structured data using XML

92

## Reading for this lecture

- Main text: Chapter 26 in [Elmasri & Navathe]
- See also, Chapter 30 in [Connolly & Begg] (30.1, 30.2, 30.4, 30.5, 30.6)
- See many on-line materials
  - e.g. [www.w3.org](http://www.w3.org)
  - <http://www.alphaworks.ibm.com/xml/newto>
  - <http://www.oracle.com/technology/tech/xml/xmlldb/index.html>

93

## Questions for this lecture

1. What are the main concepts in XML?
2. Explain the differences between DTDs and XSDs.
3. Provide DTDs for examples used in the lecture.
4. Explain the notion of semi-structured data? Give an example.
5. Explain the link between tree/graph representation and XML.
6. Explain the three approaches to storing XML data in databases.
7. What are the limits of XPath (compared to XQuery)?
8. Explain how XML can be used to support data integration.
9. Explain how XML can be used to represent and store multimedia objects, and provide an example.

94