XML

Introduction to Databases

CompSci 316 Spring 2020

DUKE
COMPUTER SCIENCE

Updates

- HW5 problems on gradescope and gradiance
- (One) more coming on XML
    - There may be an extra credit problem

- Let us know if you have problems in collaboration for project/HW or accessing material

- Video-watch assignments to be submitted by ALL (2 days after lecture + 2 bonus days)

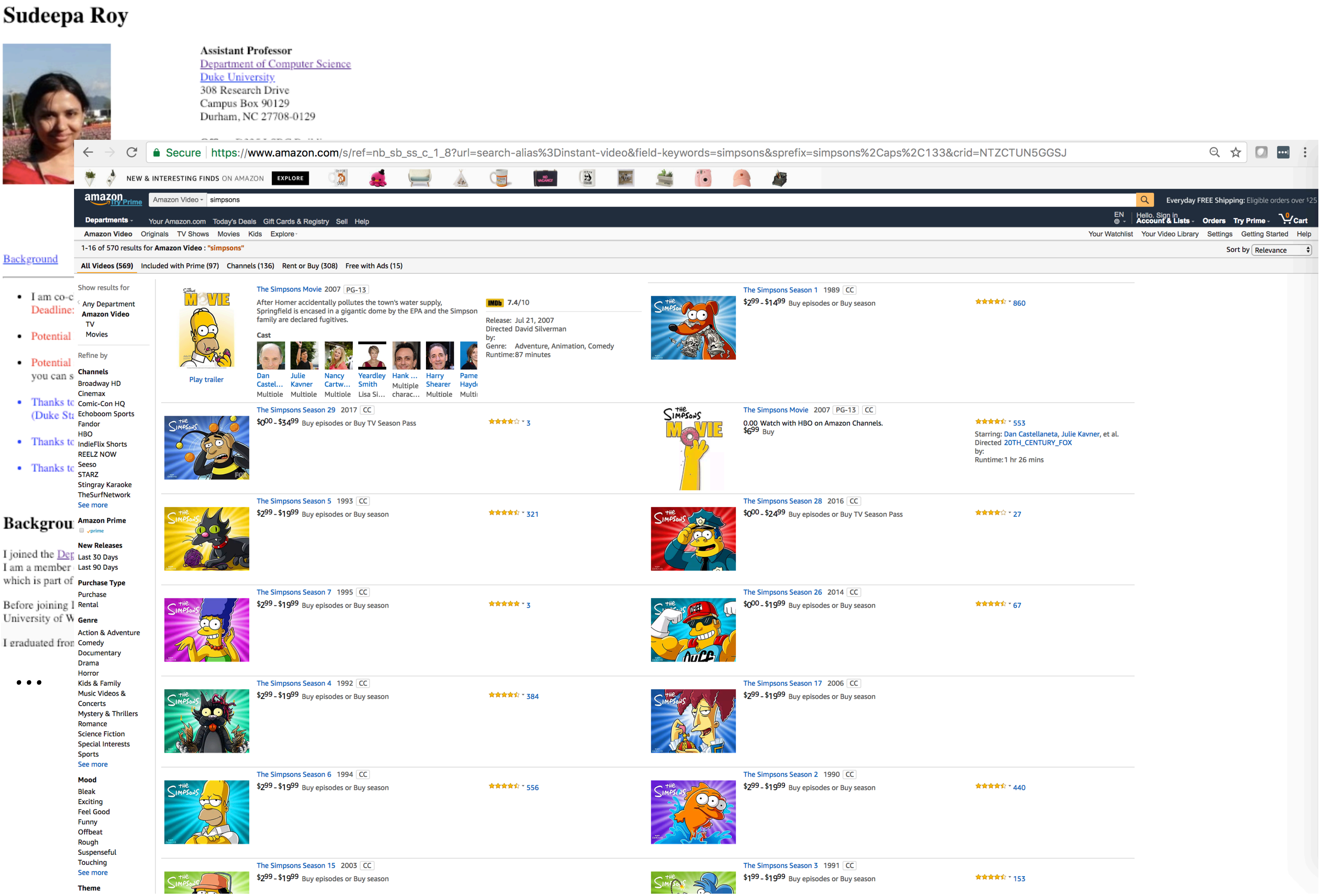
Lecture 11a:

XML Basics

Structured vs. unstructured data

- Relational databases are highly structured
  - All data resides in tables
  - You must define schema before entering any data
  - Every row confirms to the table schema
  - Changing the schema is hard and may break many things
- Texts are highly unstructured
  - Data is free-form
  - There is no pre-defined schema, and it’s hard to define any schema
  - Readers need to infer structures and meanings

What’s in between these two extremes?



Sudeepa Roy

Assistant Professor
Department of Computer Science
Duke University
308 Research Drive
Campus Box 90129
Durham, NC 27708-0129

Background

- I am co-c
Deadline:
- Potential
- Potential
you can s
- Thanks to
(Duke Sta
- Thanks to
- Thanks to

Backgrou

I joined the Dep
I am a member
which is part of

Before joining I
University of W

I graduated fron

• • •

Secure https://www.amazon.com/s/ref=nb_sb_ss_c_1_8?url=search-alias%3Dinstant-video&field-keywords=simpsons&sprefix=simpsons%2Caps%2C133&crid=NTZCTUN5GGSJ

1-16 of 570 results for **Amazon Video** : "simpsons"

All Videos (569) Included with Prime (97) Channels (136) Rent or Buy (308) Free with Ads (15)

Show results for
Any Department
Amazon Video
TV
Movies

Refine by

Channels
Broadway HD
Cinemax
Comic-Con HQ
Echoboom Sports
Fandor
HBO
IndieFlix Shorts
REELZ NOW
Seeso
STARZ
Stingray Karaoke
TheSurfNetwork
See more

Amazon Prime
□ prime

New Releases
Last 30 Days
Last 90 Days

Purchase Type
Purchase
Rental

Genre
Action & Adventure
Comedy
Documentary
Drama
Horror
Kids & Family
Music Videos & Concerts
Mystery & Thrillers
Romance
Science Fiction
Special Interests
Sports
See more

Mood
Bleak
Exciting
Feel Good
Funny
Offbeat
Rough
Suspenseful
Touching
See more

Theme

The Simpsons Movie 2007 PG-13
After Homer accidentally pollutes the town's water supply, Springfield is encased in a gigantic dome by the EPA and the Simpson family are declared fugitives.
Play trailer
Cast
Dan Castel… Multiple
Julie Kavner Multiple
Nancy Cartw… Multiple
Yeardley Smith Lisa Si…
Hank … Multiple charac…
Harry Shearer Multiple
Pame Hayd… Multi…
IMDb 7.4/10
Release: Jul 21, 2007
Directed by: David Silverman
Genre: Adventure, Animation, Comedy
Runtime: 87 minutes

The Simpsons Season 1 1989 CC
$2.99 - $14.99 Buy episodes or Buy season
860

The Simpsons Season 29 2017 CC
$0.00 - $34.99 Buy episodes or Buy TV Season Pass
3

The Simpsons Movie 2007 PG-13 CC
0.00 Watch with HBO on Amazon Channels.
$6.99 Buy
553
Starring: Dan Castellaneta, Julie Kavner, et al.
Directed by: 20TH_CENTURY_FOX
Runtime: 1 hr 26 mins

The Simpsons Season 5 1993 CC
$2.99 - $19.99 Buy episodes or Buy season
321

The Simpsons Season 28 2016 CC
$0.00 - $24.99 Buy episodes or Buy TV Season Pass
27

The Simpsons Season 7 1995 CC
$2.99 - $19.99 Buy episodes or Buy season
3

The Simpsons Season 26 2014 CC
$0.00 - $19.99 Buy episodes or Buy season
67

The Simpsons Season 4 1992 CC
$2.99 - $19.99 Buy episodes or Buy season
384

The Simpsons Season 17 2006 CC
$2.99 - $19.99 Buy episodes or Buy season
556

The Simpsons Season 6 1994 CC
$2.99 - $19.99 Buy episodes or Buy season
556

The Simpsons Season 2 1990 CC
$2.99 - $19.99 Buy episodes or Buy season
440

The Simpsons Season 15 2003 CC
$2.99 - $19.99 Buy episodes or Buy season

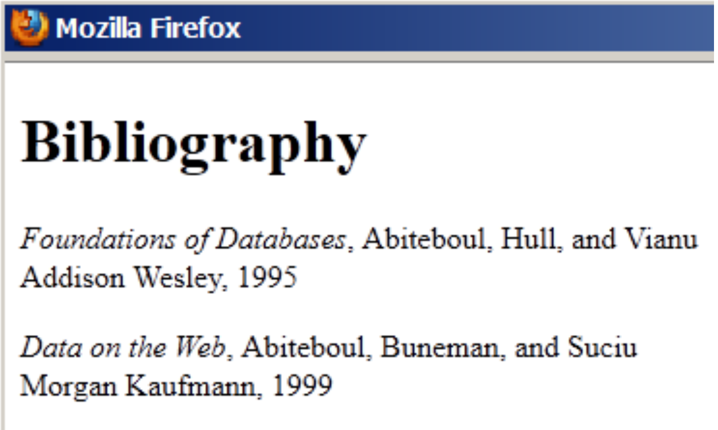
The Simpsons Season 3 1991 CC
$1.99 - $19.99 Buy episodes or Buy season
153

Semi-structured data

- Observation: most data have some structure, e.g.:
  - Book: chapters, sections, titles, paragraphs, references, index, etc.
  - Item for sale: name, picture, price (range), ratings, promotions, etc.

XML: eXtensible Markup Language

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



Bibliography

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

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

- Text-based
- Capture data (content), not presentation
  - Similar but different from HTML
- Data self-describes its structure
  - Names and nesting of tags have meanings!

Other nice features of XML

- Portability: Just like HTML, you can ship XML data across platforms
    - Relational data requires heavy-weight API's
- Flexibility: You can represent any information (structured, semi-structured, documents, …)
    - Relational data is best suited for structured data
- Extensibility: Since data describes itself, you can change the schema easily
    - Relational schema is rigid and difficult to change

XML terminology

```
<bibliography>
  <book ISBN="ISBN-10" price="80.00">
    <title>Foundations of Databases</title>
    <author>Abiteboul</author>
    <author>Hull</author>
    <author>Vianu</author>
    <publisher>Addison Wesley</publisher>
    <year>1995</year>
  </book>…
</bibliography>
```

- Tag names: book, title, …
- Start tags: <book>, <title>, …
- End tags: </book>, </title>, …
- An element is enclosed by a pair of start and end tags: <book>…</book>
  - Elements can be nested: <book>…<title>…</title>…</book>
  - Empty elements: <is_textbook></is_textbook>
    - Can be abbreviated: <is_textbook/>
- Elements can also have attributes: <book ISBN="…" price="80.00">
- Many other features

☞Ordering generally matters, except for attributes

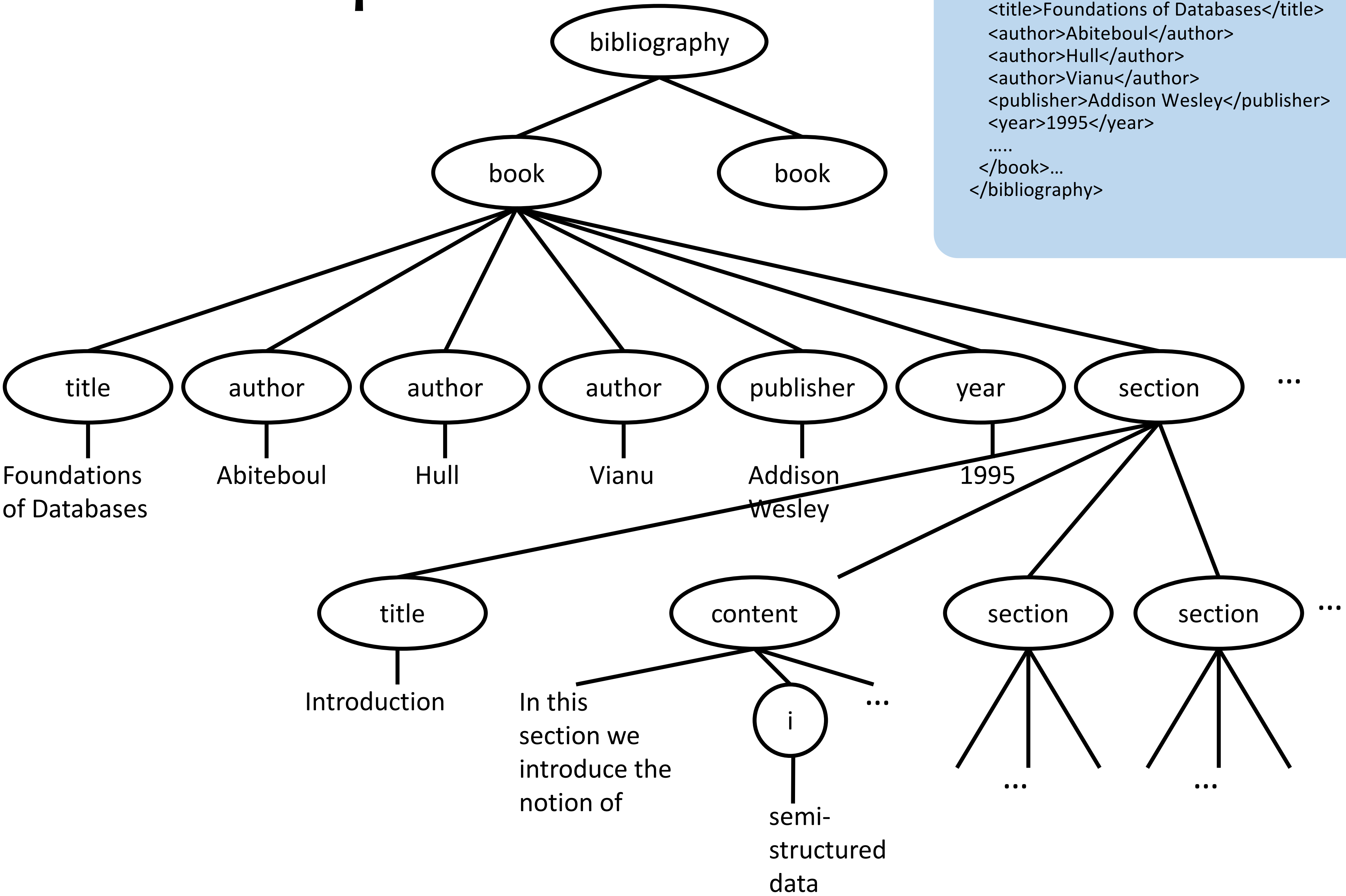
Well-formed XML documents

A well-formed XML document

- Follows XML lexical conventions
  - Wrong: <section>We show that x < 0…</section>
  - Right: <section>We show that x &lt; 0…</section>
    - Other special entities: > becomes &gt; and & becomes &amp;
- Contains a single root element
- Has properly matched tags and properly nested elements (like parentheses matching)
  - Right: <section>…<subsection>…</subsection>…</section>
  - Wrong: <section>…<subsection>…</section>…</subsection>
  - Think of {{()}([])} matching!

A tree representation

```
<bibliography>
  <book ISBN="ISBN-10" price="80.00">
    <title>Foundations of Databases</title>
    <author>Abiteboul</author>
    <author>Hull</author>
    <author>Vianu</author>
    <publisher>Addison Wesley</publisher>
    <year>1995</year>
    …..
  </book>…
</bibliography>
```



Optional slide

DTD and Schema (details omitted)

- DTD (Document Type Definitions)
- Specifies Schema and constraints for XML
- Specifies a grammar (e.g. +, ? for one or more, zero or 1 etc.)
- Another option XML schema (.xsd)

```
<?xml version="1.0"?>

<!DOCTYPE bibliography [
      <!ELEMENT bibliography (book+)>
      <!ELEMENT book (title, author*, publisher?, year?, section*)>
      <!ATTLIST book ISBN ID #REQUIRED>
      <!ATTLIST book price CDATA #IMPLIED>
      <!ELEMENT title (#PCDATA)>
      <!ELEMENT author (#PCDATA)>
      <!ELEMENT publisher (#PCDATA)>
      <!ELEMENT year (#PCDATA)>
      <!ELEMENT i (#PCDATA)>
      <!ELEMENT content (#PCDATA|i)*>
      <!ELEMENT section (title, content?, section*)>
    ]>

<bibliography>
      <book ISBN="ISBN-10" price="80.00">
        <title>Foundations of Databases</title>
        <author>Abiteboul</author>
        <author>Hull</author>
        <author>Vianu</author>
        <publisher>Addison Wesley</publisher>
        <year>1995</year>
        <section>…</section>…
      </book>
      …
    </bibliography>
```

XML versus relational data

Relational data

- Schema is always fixed in advance and difficult to change
- Simple, flat table structures
- Ordering of rows and columns is unimportant
- Exchange is problematic
- “Native” support in all serious commercial DBMS

XML data

- Well-formed XML does not require predefined, fixed schema
- Nested structure; ID/IDREF(S) permit arbitrary graphs
- Ordering forced by document format; may or may not be important
- Designed for easy exchange
- Often implemented as an “add-on” on top of relations

Case study

- Design an XML document representing cities, counties, and states
  - For states, record name and capital (city)
  - For counties, record name, area, and location (state)
  - For cities, record name, population, and location (county and state)
- Assume the following:
  - Names of states are unique
  - Names of counties are only unique within a state
  - Names of cities are only unique within a county
  - A city is always located in a single county
  - A county is always located in a single state

A possible design

Design an XML document representing cities, counties, and states

For states, record name and capital (city)

For counties, record name, area, and location (state)

For cities, record name, population, and location (county and state)

Assume the following:

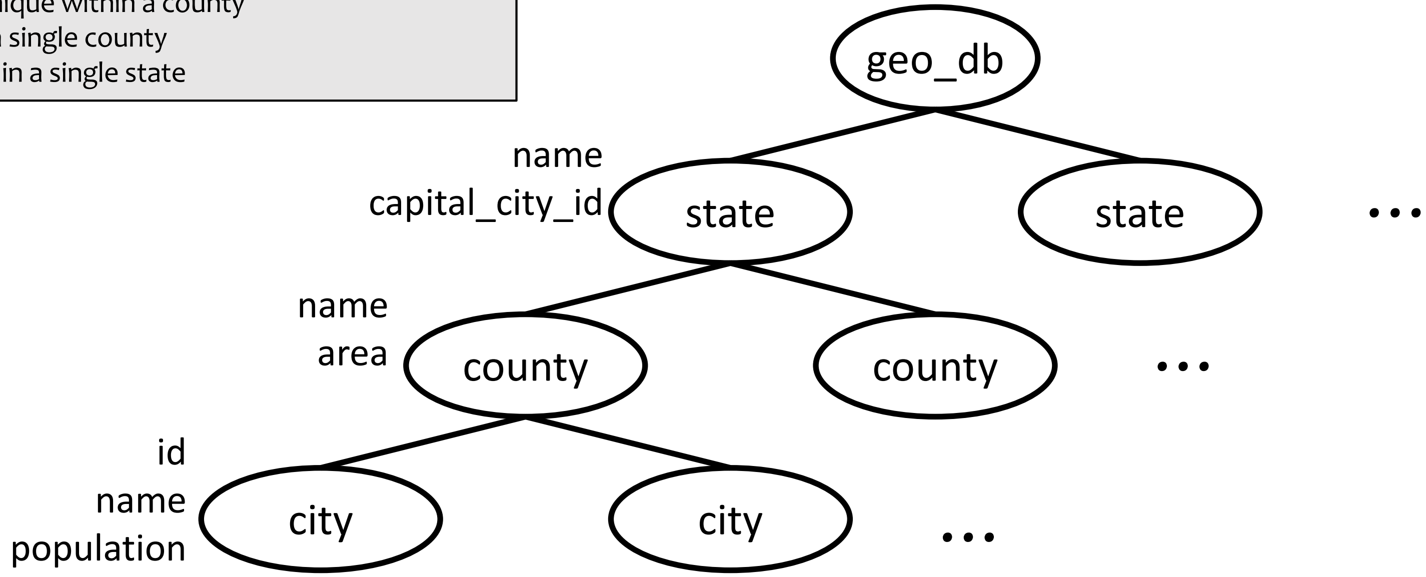
Names of states are unique

Names of counties are only unique within a state

Names of cities are only unique within a county

A city is always located in a single county

A county is always located in a single state



Lecture 11b:

XPath and XQuery

Query languages for XML

- XPath
  - Path expressions with conditions

  ☞ Building block of other standards (XQuery, XSLT, XLink, XPointer, etc.)

- XQuery
  - XPath + full-fledged SQL-like query language

- Also XSLT (not covered)

- We would cover only simple queries

Try the queries in this lecture online

- There are many online Xpath/Xquery testers e.g.
- http://codebeautify.org/Xpath-Tester (XPATH)
- http://videlibri.sourceforge.net/cgi-bin/xidelcgi (XQUERY)
- Try with this example (or change it for different queries)
- Caveats
  - if you see bad characters, you might have to replace them like " or .
  - Not everything works all the time! Try different websites and config

```
<bibliography>
<book ISBN="ISBN-10" price="70">
<title>Foundations of Databases</title>
<author>Abiteboul</author>
<author>Hull</author>
<author>Vianu</author>
<publisher>Addison Wesley</publisher>
<year>1995</year>
<section>abc</section>
</book>
<book ISBN="ISBN-11"price="20">
 <title>DBSTS</title>
 <author>Ramakrishnan</author>
 <author>Gehrke</author>
 <publisher>Addison Wesley</publisher>
 <year>1999</year>
 <section>abc</section>
 </book>
</bibliography>
```

XPath

- XPath specifies path expressions that match XML data by navigating down (and occasionally up and across) the tree
- Example
  - Query: /bibliography/book/author
    - Like a file system path, except there can be multiple “subdirectories” with the same name
  - Result: all author elements reachable from root via the path /bibliography/book/author

```
<bibliography>
<book ISBN="ISBN-10" price="70">
<title>Foundations of Databases</title>
<author>Abiteboul</author>
<author>Hull</author>
<author>Vianu</author>
<publisher>Addison Wesley</publisher>
<year>1995</year>
<section>abc</section>
</book>
<book ISBN="ISBN-11"price="20">
 <title>DBSTS</title>
 <author>Ramakrishnan</author>
 <author>Gehrke</author>
 <publisher>Addison Wesley</publisher>
 <year>1999</year>
 <section>abc</section>
 </book>
</bibliography>
```

Basic XPath constructs

/ separator between steps in a path

name matches any child element with this tag name

* matches any child element

@*name* matches the attribute with this name

@* matches any attribute

// matches any descendent element or the current element itself

. matches the current element

.. matches the parent element

Simple XPath examples

```
<bibliography>
<book ISBN="ISBN-10" price="70">
<title>Foundations of Databases</title>
<author>Abiteboul</author>
<author>Hull</author>
<author>Vianu</author>
<publisher>Addison Wesley</publisher>
<year>1995</year>
<section>abc</section>
</book>
<book ISBN="ISBN-11"price="20">
 <title>DBSTS</title>
 <author>Ramakrishnan</author>
 <author>Gehrke</author>
 <publisher>Addison Wesley</publisher>
 <year>1999</year>
 <section>abc</section>
 </book>
</bibliography>
```

- All book titles

  /bibliography/book/title

- All book ISBN numbers

  /bibliography/book/@ISBN

- All title elements, anywhere in the document

  //title

- All section titles, anywhere in the document

  //section/title

- Authors of bibliographical entries (suppose there are articles, reports, etc. in addition to books)

  /bibliography/*/author

Predicates in path expressions

[condition]   matches the "current" element if *condition* evaluates to true on the current element

- Books with price lower than $50

  /bibliography/book[@price<50]

  - XPath will automatically convert the price string to a numeric value for comparison

```
<bibliography>
<book ISBN="ISBN-10" price="70">
<title>Foundations of Databases</title>
<author>Abiteboul</author>
<author>Hull</author>
<author>Vianu</author>
<publisher>Addison Wesley</publisher>
<year>1995</year>
<section>abc</section>
</book>
<book ISBN="ISBN-11"price="20">
 <title>DBSTS</title>
 <author>Ramakrishnan</author>
 <author>Gehrke</author>
 <publisher>Addison Wesley</publisher>
 <year>1999</year>
 <section>abc</section>
 </book>
</bibliography>
```

Predicates in path expressions – contd.

- Books with author “Abiteboul”

  /bibliography/book[author='Abiteboul']

- Books with a publisher child element

  /bibliography/book[publisher]

- Prices of books authored by “Abiteboul”

  /bibliography/book[author='Abiteboul']/@price

```
<bibliography>
<book ISBN="ISBN-10" price="70">
<title>Foundations of Databases</title>
<author>Abiteboul</author>
<author>Hull</author>
<author>Vianu</author>
<publisher>Addison Wesley</publisher>
<year>1995</year>
<section>abc</section>
</book>
<book ISBN="ISBN-11"price="20">
 <title>DBSTS</title>
 <author>Ramakrishnan</author>
 <author>Gehrke</author>
 <publisher>Addison Wesley</publisher>
 <year>1999</year>
 <section>abc</section>
 </book>
</bibliography>
```

More complex predicates

Predicates can use and, or, and not

- Books with price between $40 and $50
  - /bibliography/book[40<=@price and @price<=50]
- Books authored by “Abiteboul” or those with price no lower than $50
  - /bibliography/book[author='Abiteboul' or @price>=50]
  - /bibliography/book[author='Abiteboul' or not(@price<50)]
  - Any difference between these two queries?

```
<bibliography>
<book ISBN="ISBN-10" price="70">
<title>Foundations of Databases</title>
<author>Abiteboul</author>
<author>Hull</author>
<author>Vianu</author>
<publisher>Addison Wesley</publisher>
<year>1995</year>
<section>abc</section>
</book>
<book ISBN="ISBN-11"price="20">
 <title>DBSTS</title>
 <author>Ramakrishnan</author>
 <author>Gehrke</author>
 <publisher>Addison Wesley</publisher>
 <year>1999</year>
 <section>abc</section>
 </book>
</bibliography>
```

A tricky example

- Suppose for a moment that price is a child element of book, and there may be multiple prices per book
- Books with some price in range [20, 50]
  - Wrong answer:
    /bibliography/book[price >= 20 and price <= 50]

    (returns true with one price 10 and one 70!)
  - Correct answer:
    /bibliography/book[price[. >= 20 and . <= 50]]

Predicates involving node-sets

/bibliography/book[author='Abiteboul']

- There may be multiple authors, so author in general returns a node-set (in XPath terminology)
- The predicate evaluates to true as long as it evaluates true for at least one node in the node-set, i.e., at least one author is “Abiteboul”
- Another tricky query

  /bibliography/book[author='Abiteboul' and author!='Abiteboul']

  - Will it return any books?

  - (Returns books with at least one “Abiteboul” and one non-Abiteboul as authors!)

Read yourself if needed for HW problems

More XPath operators and functions

Frequently used in conditions:

x + *y*, *x* – *y*, *x* * *y*, *x* div *y*, *x* mod *y*

contains(*x*, *y*) true if string *x* contains string *y*

count(*node-set*) counts the number nodes in *node-set*

position() returns the "context position" (roughly, the position of the current node in the node-set containing it)

last() returns the "context size" (roughly, the size of the node-set containing the current node)

name() returns the tag name of the current element

Books with fewer than 10 sections

```
/bibliography/book[count(section)<10]
```

All elements whose tag names contain "section" (e.g., "subsection")

```
//*[contains(name(), 'section')]
```

Title of the first section in each book

```
/bibliography/book/section[position()=1]/title
```

A shorthand: `/bibliography/book/section[1]/title`

Title of the last section in each book

```
/bibliography/book/section[position()=last()]/title
```

XQuery

- XPath + full-fledged SQL-like query language

- XQuery expressions can be
  - XPath expressions
  - FLWOR expressions
  - Quantified expressions
  - Aggregation, sorting, and more…

Sample online Xquery tester:
http://videlibri.sourceforge.net/cgi-bin/xidelcgi

Use Xquery 3.0, node format = xml, output format = adhoc, and compatibility = Standard Xquery in the settings

- An XQuery expression can return a new result XML document

A simple XQuery based on XPath

Find all books with price lower than $50

```
<result>{
  ~~doc("bib.xml")~~/bibliography/book[@price<50]
}</result>
```

- Things outside {}’s are copied to output verbatim
- Things inside {}’s are evaluated and replaced by the results
    - doc("bib.xml") specifies the document to query
    - Omit this in the online tester
    - The XPath expression returns a sequence of book elements
    - These elements (including all their descendants) are copied to output

FLWR expressions

- Retrieve the titles of books published before 2000, together with their publisher

```
<result>{
 for $b in /bibliography/book
 let $p := $b/publisher
 where $b/year < 2000
 return
  <book>
   { $b/title }
   { $p }
  </book>
}</result>
```

- for: loop
  - $b ranges over the result sequence, getting one item at a time
- let: “assignment”
  - $p gets the entire result of $b/publisher (possibly many nodes)
- where: filtering by condition
- return: result structuring
  - Invoked in the “innermost loop,” i.e., once for each successful binding of all query variables that satisfies where

An equivalent formulation

- Retrieve the titles of books published before 2000, together with their publisher

```
<result>{
 for $b in /bibliography/book[year<2000]
 return
  <book>
   { $b/title }
   { $b/publisher }
  </book>
}</result>
```

Another formulation

- Retrieve the titles of books published before 2000, together with their publisher

```
<result>{
 for $b in /bibliography/book,
     $p in $b/publisher
 where $b/year < 2000
 return
   <book>
     { $b/title }
     { $p }
   </book>
}</result>
```

} Nested loop

- Is this query equivalent to the previous two?
- Yes, if there is one publisher per book
- No, in general
  - Two result book elements will be created for a book with two publishers
  - No result book element will be created for a book with no publishers

Yet another formulation

- Retrieve the titles of books published before 2000, together with their publisher

```
<result>{
 let $b := /bibliography/book
 where $b/year < 2000
 return
   <book>
     { $b/title }
     { $b/publisher }
   </book>
}</result>
```

- Is this query correct?
- No!
- It will produce only one output book element, with all titles clumped together and all publishers clumped together
- All books will be processed (as long as one is published before 2000)

An explicit join

- Find pairs of books that have common author(s)

```
<result>{
 for $b1 in ~~doc("bib.xml")~~//book
 for $b2 in ~~doc("bib.xml")~~//book
 where $b1/author = $b2/author
   and $b1/title > $b2/title
 return
  <pair>
   {$b1/title}
   {$b2/title}
  </pair>
}</result>
```

← These are string comparisons,
not identity comparisons!

OPTIONAL SLIDE

More features

- Learn if useful in homework, not needed for exams
- Subqueries
  - normalize-space(*string*) removes leading and trailing spaces from string, and replaces all internal sequences of white spaces with one white space
- Existential (some) and Universal (all)
- Conditional
  - Use anywhere you'd expect a value, e.g.:
  - let $foo := if (…) then … else …
  - return <bar blah="{ if (…) then … else … }"/>

Extract book titles and their authors; make title an attribute and rename author to writer

```
            <bibliography>{
 for $b in doc("bib.xml")/bibliography/book
 return
   <book title="{normalize-space($b/title)}">{
     for $a in $b/author
     return <writer>{string($a)}</writer>
   }</book>
}</bibliography>
```

Find titles of books in which XML is mentioned in some section

```
            <result>{
      for $b in doc("bib.xml")//book
      where (some $section in $b//section
    satisfies
           contains(string($section),
    "XML"))
      return $b/title
    }</result>
```

Find titles of books in which XML is mentioned in every section

```
            <result>{
      for $b in doc("bib.xml")//book
      where (every $section in $b//section
    satisfies
           contains(string($section),
    "XML"))
      return $b/title
    }</result>
```

OPTIONAL SLIDE

Aggregation

- Learn if useful in homework, not needed for exams
- List each publisher and the average prices of all its books

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

- distinct-values(*collection*) removes duplicates by value
  - If the collection consists of elements (with no explicitly declared types), they are first converted to strings representing their “normalized contents”
- avg(*collection*) computes the average of *collection* (assuming each item in *collection* can be converted to a numeric value)

Lecture 11c:

XML to Relational Data

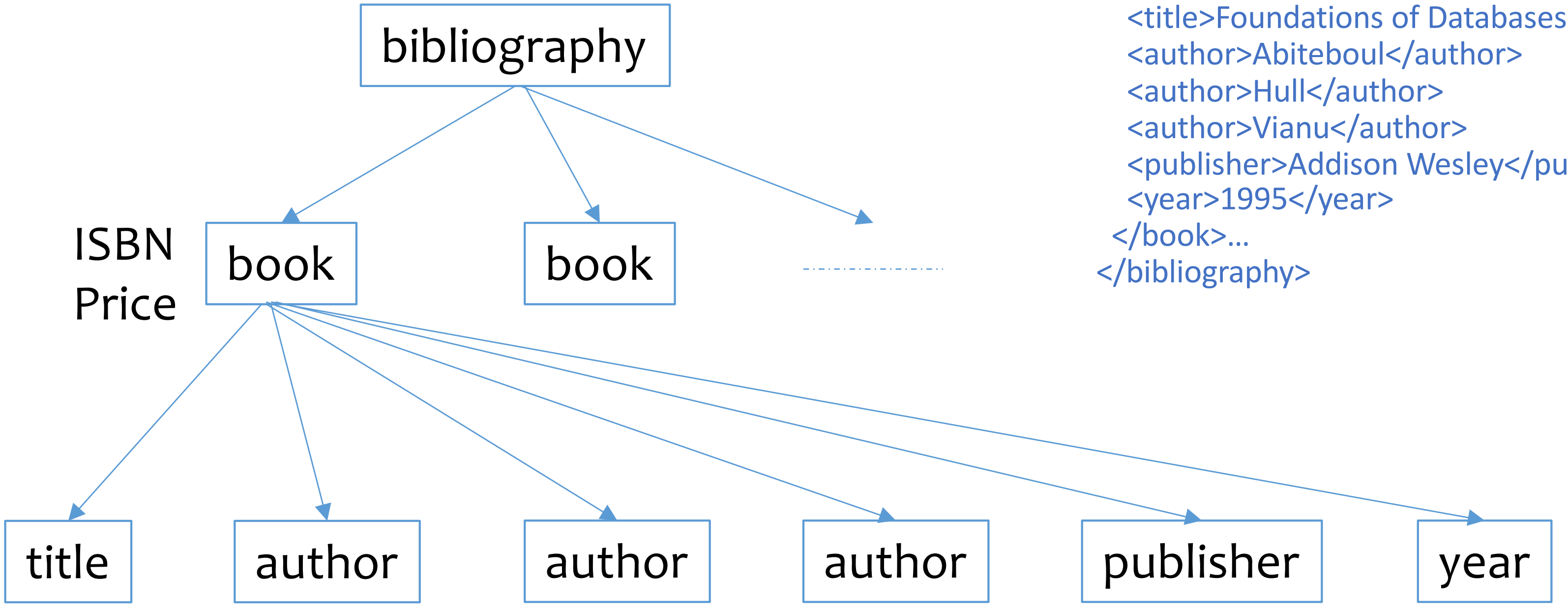
Which one is easier?

- XML to relational?
- Or
- Relational to XML?

Mapping XML to relational

- Store XML in a column
  - CLOB (Character Large OBject) type
  - Not much useful!
- Alternatives?
  - Schema-oblivious mapping:
    well-formed XML → generic relational schema
    - Node/edge-based mapping for graphs
    - Interval-based mapping for trees
    - Path-based mapping for trees (not covered)
  - Schema-aware mapping (not covered):
    valid XML → special relational schema based on DTD

Example – Node/Edge Based



```
<bibliography>
  <book ISBN="ISBN-10" price="80.00">
    <title>Foundations of Databases</title>
    <author>Abiteboul</author>
    <author>Hull</author>
    <author>Vianu</author>
    <publisher>Addison Wesley</publisher>
    <year>1995</year>
  </book>...
</bibliography>
```

- How would you translate it to a relational schema?
  - Element? Attribute? Parent-child relationship?
  - Keys? (Do not see the next slides yet!)

Node/edge-based: schema

- *Element(<u>eid</u>, tag)*
- *Attribute(eid, attrName, attrValue)* Key: (*eid*, *attrName*)
  - Attribute order does not matter
- *ElementChild(eid, pos, child)* Keys: (*eid*, *pos*), (*child*)
  - *pos* specifies the ordering of children
  - *child* references either *Element*(*eid*) or *Text*(*tid*)
- *Text(<u>tid</u>, value)*
  - *tid* cannot be the same as any *eid*

☞Need to "invent" lots of *id*'s

☞Need indexes for efficiency, e.g., *Element*(*tag*), *Text*(*value*)

Node/edge-based: example

```
<bibliography>
 <book ISBN="ISBN-10" price="80.00">
  <title>Foundations of Databases</title>
  <author>Abiteboul</author>
  <author>Hull</author>
  <author>Vianu</author>
  <publisher>Addison Wesley</publisher>
  <year>1995</year>
 </book>…
</bibliography>
```

Attribute

eid	attrName	attrValue
e1	ISBN	ISBN-10
e1	price	80

Text

tid	value
t0	Foundations of Databases
t1	Abiteboul
t2	Hull
t3	Vianu
t4	Addison Wesley
t5	1995

Element

eid	tag
e0	bibliography
e1	book
e2	title
e3	author
e4	author
e5	author
e6	publisher
e7	year

ElementChild

eid	pos	child
e0	1	e1
e1	1	e2
e1	2	e3
e1	3	e4
e1	4	e5
e1	5	e6
e1	6	e7
e2	1	t0
e3	1	t1
e4	1	t2
e5	1	t3
e6	1	t4
e7	1	t5

Node/edge-based: queries

- //title
  - SELECT eid FROM Element WHERE tag = 'title';
- //section/title
  - SELECT e2.eid
    FROM Element e1, ElementChild c, Element e2
    WHERE e1.tag = 'section'
    AND e2.tag = 'title'
    AND e1.eid = c.eid
    AND c.child = e2.eid;

☞Path expression becomes joins!

- Number of joins is proportional to the length of the path expression

- //bibliography/book[author="Abiteboul"]/@price
  - More complex SQL queries with EXISTS needed
- //book//title
  - Needs recursion (not covered yet)

eid	*attrName*	*attrValue*
e1	ISBN	ISBN-10
e1	price	80

Attribute

eid	*pos*	*child*
e0	1	e1
e1	1	e2
e1	2	e3
e1	3	e4
e1	4	e5
e1	5	e6
e1	6	e7
e2	1	t0
e3	1	t1
e4	1	t2
e5	1	t3
e6	1	t4
e7	1	t5

ElementChild

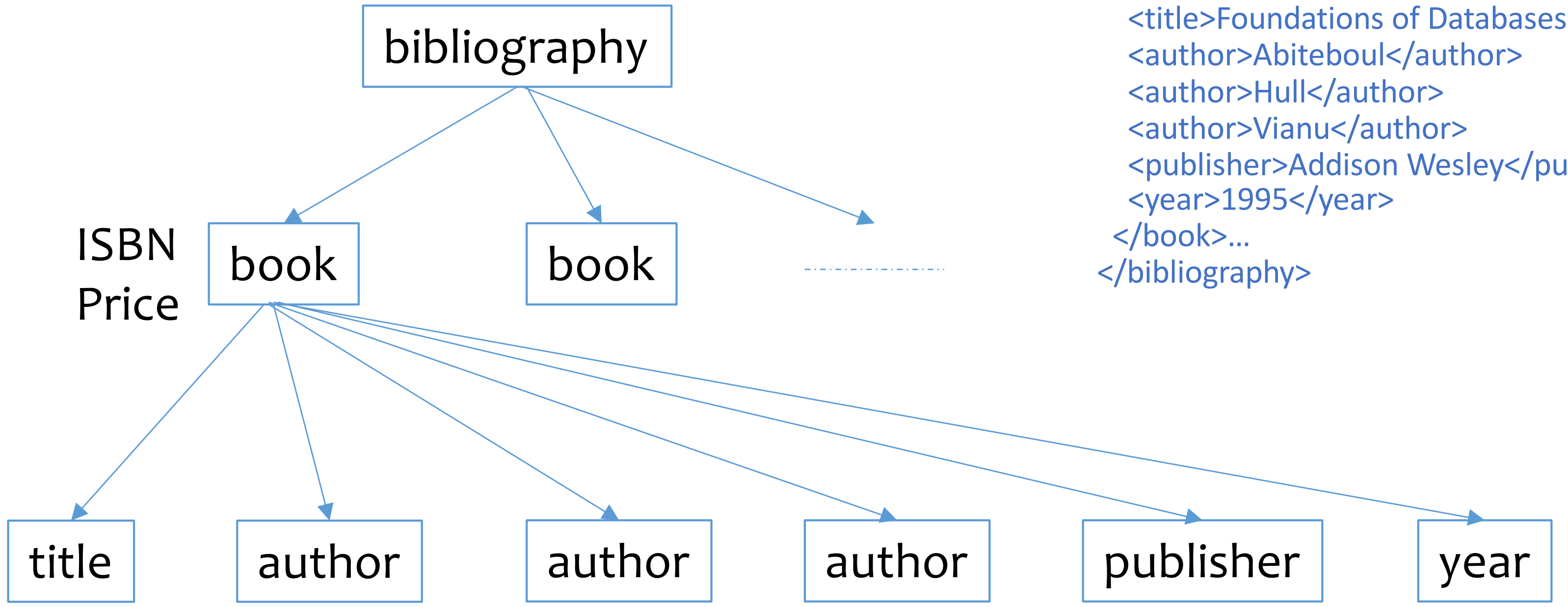
eid	*tag*
e0	bibliography
e1	book
e2	title
e3	author
e4	author
e5	author
e6	publisher
e7	year

Element

Text

tid	*value*
t0	Foundations of Databases
t1	Abiteboul
t2	Hull
t3	Vianu
t4	Addison Wesley
t5	1995

Example – Interval Based

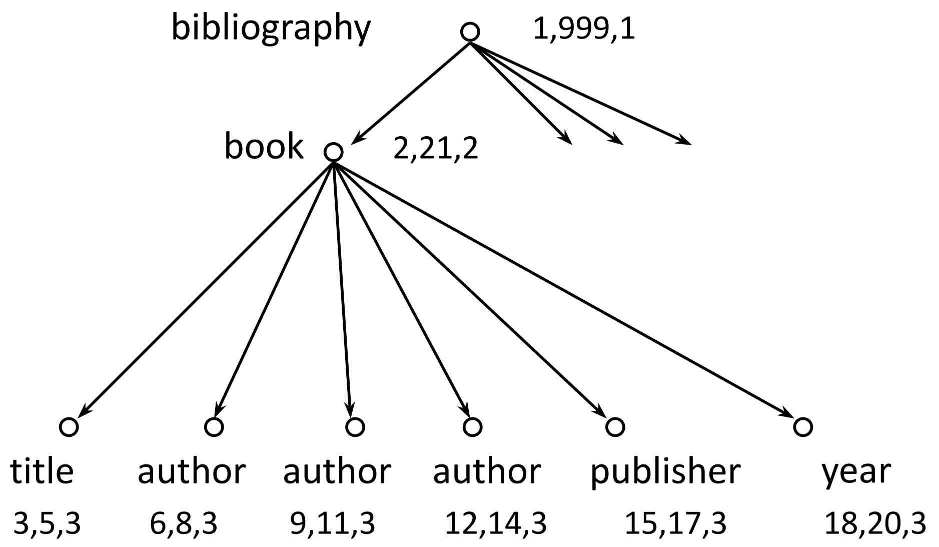


```
<bibliography>
  <book ISBN="ISBN-10" price="80.00">
    <title>Foundations of Databases</title>
    <author>Abiteboul</author>
    <author>Hull</author>
    <author>Vianu</author>
    <publisher>Addison Wesley</publisher>
    <year>1995</year>
  </book>…
</bibliography>
```

- How would you translate it to a relational schema?
  - Using intervals!

Interval-based: example

```
1<bibliography>
 2<book ISBN="ISBN-10" price="80.00">
  3<title>4Foundations of Databases</title>5
  6<author>7Abiteboul</author>8
  9<author>10Hull</author>11
 12<author>13Vianu</author>14
 15<publisher>16Addison Wesley</publisher>17
 18<year>191995</year>20
 </book>21…
</bibliography>999
```



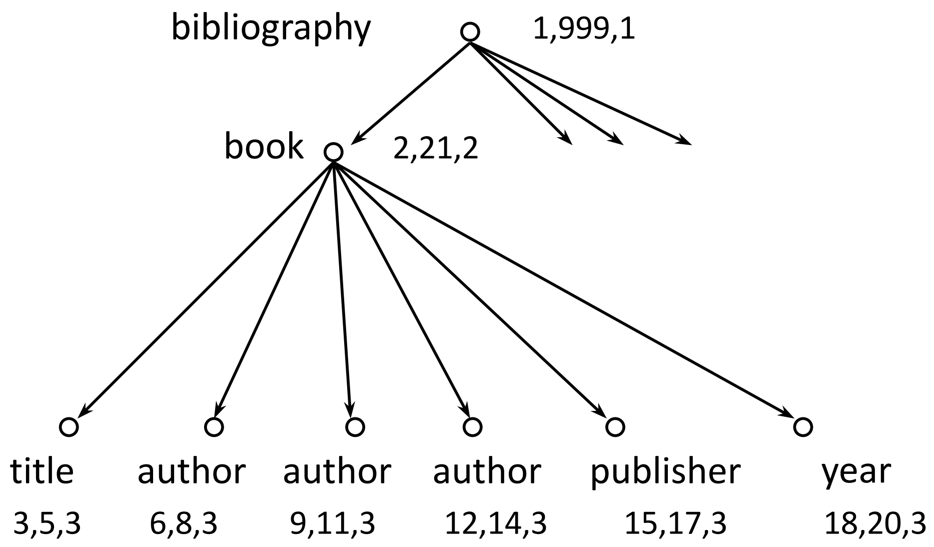
First two fields denote the interval.. see next slide

Interval-based: schema

- *Element(left, right, level, tag)*
  - *left* is the start position of the element
  - *right* is the end position of the element
  - *level* is the nesting depth of the element (strictly speaking, unnecessary)
  - Key is *left*
- *Text(left, right, level, value)*
  - Key is *left*
- *Attribute(left, attrName, attrValue)*
  - Key is (*left*, *attrName*)

Interval-based: example

```
1<bibliography>
 2<book ISBN="ISBN-10" price="80.00">
  3<title>4Foundations of Databases</title>5
  6<author>7Abiteboul</author>8
  9<author>10Hull</author>11
 12<author>13Vianu</author>14
 15<publisher>16Addison Wesley</publisher>17
 18<year>191995</year>20
 </book>21...
</bibliography>999
```



☞Where did *ElementChild* go?

- e_1 is the parent of e_2 iff:

  $[e_1.left, e_1.right] \supset [e_2.left, e_2.right]$, and
  $e_1.level = e_2.level - 1$

Interval-based: queries

Element(left, right, level, tag)
Text(left, right, level, value)
Attribute(left, attrName, attrValue)

- //section/title
  - SELECT e2.left
    FROM Element e1, Element e2
    WHERE e1.tag = 'section' AND e2.tag = 'title'
    AND e1.left < e2.left AND e2.right < e1.right
    AND e1.level = e2.level-1;

  ☞Path expression becomes “containment” joins!

    - Number of joins is proportional to path expression length

- //book//title
  - SELECT e2.left
    FROM Element e1, Element e2
    WHERE e1.tag = 'book' AND e2.tag = 'title'
    AND e1.left < e2.left AND e2.right < e1.right;

  ☞No recursion!