SFU

Database Systems I

CMPT 354 Summer 2024

Zhengjie Miao

26 June 2024

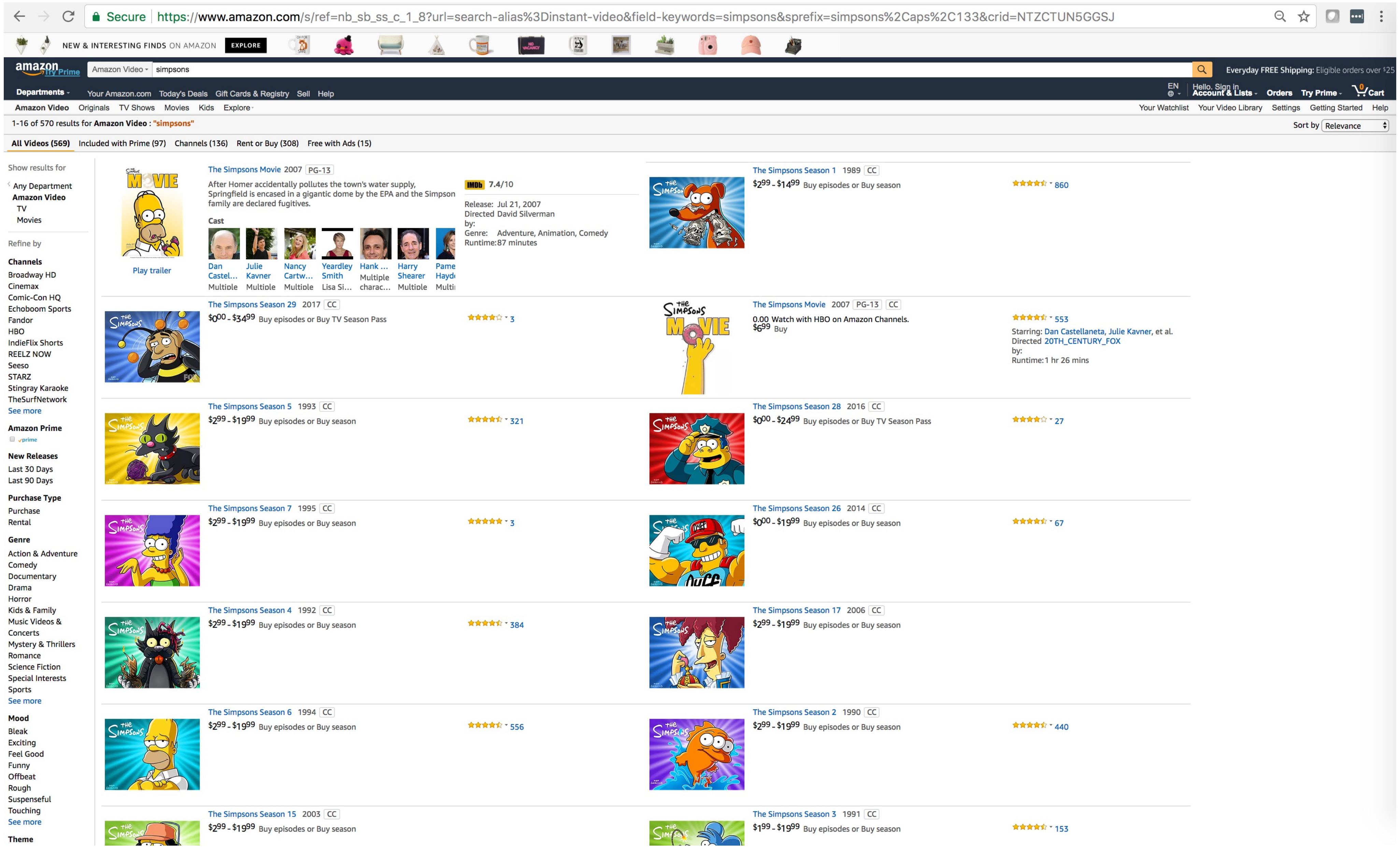
Announcements (Wed. June 26)

- Assignment 4 to be assigned tomorrow; due July 7

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?



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"

Sort by Relevance

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.

IMDb 7.4/10

Release: Jul 21, 2007
Directed by: David Silverman
Genre: Adventure, Animation, Comedy
Runtime: 87 minutes

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…

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

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

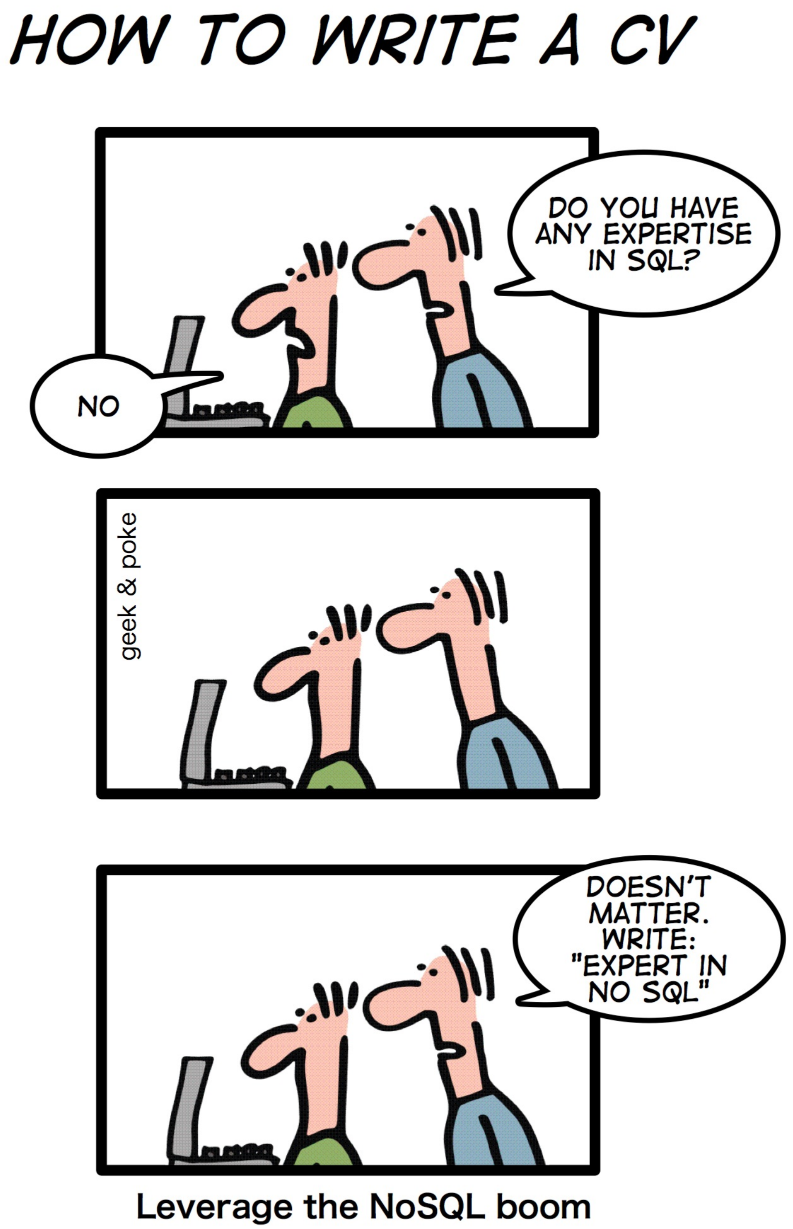
Semi-structured data

- 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.
  - Web page: HTML
- Ideas:
  - Ensure data is “well-formatted”
  - If needed, ensure data is also “well-structured”
    - But make it easy to define and extend this structure
  - Make data “self-describing”

Every time the data changes in a relational database, you need to change the schema
For a non-relational database, you can add the changes to the new data and do not need to change everything

SQL vs. NoSQL

- SQL's rigidity in face of semi-structured data is one of the reasons behind the rise of (some) NoSQL systems
  - NoSQL has other motivations, which we hope to get to in a later part of this course


HOW TO WRITE A CV
DO YOU HAVE ANY EXPERTISE IN SQL?
NO
geek & poke
DOESN'T MATTER. WRITE: "EXPERT IN NO SQL"
Leverage the NoSQL boom

https://xdba.files.wordpress.com/2014/07/201301-nosql-cv.jpg

Our roadmap thru the NoSQL land

```
<congress>
  <people>
    <person birthday="1952-11-09" gender="M" id="B000944" name="Sherrod Brown">
      <role district="[illegible]" enddate="1995-01-03" party="Democrat" startdate="1993-01-05" state="OH" type="rep"/>
      <role district="[illegible]" enddate="1997-01-03" party="Democrat" startdate="1995-01-04" state="OH" type="rep"/>
      <role district="[illegible]" enddate="[illegible]-01-03" party="Democrat" startdate="1997-01-07" state="OH" type="rep"/>
      <role district="[illegible]" enddate="2001-[illegible]3" party="Democrat" startdate="1999-01-06" state="OH" type="rep"/>
      <role district="[illegible]" enddate="[illegible]01-03" party="Democrat" startdate="2001-01-03" state="OH" type="rep"/>
      <role district="13" enddate="2005-01-03" party="Democrat" startdate="2003-01-07" state="OH" type="rep"/>
      <role district="13" enddate="2007-01-03" party="Democrat" startdate="2005-01-04" state="OH" type="rep"/>
      <role enddate="2013-01-03" party="Democrat" startdate="2007-01-04" state="OH" type="sen"/>
      <role current="1" enddate="2019-01-03" party="Democrat" startdate="2013-01-03" state="OH" type="sen"/>
    </person>
    <person birthday="1958-10-13" gender="F" id="C000127" name="Maria Cantwell">
```

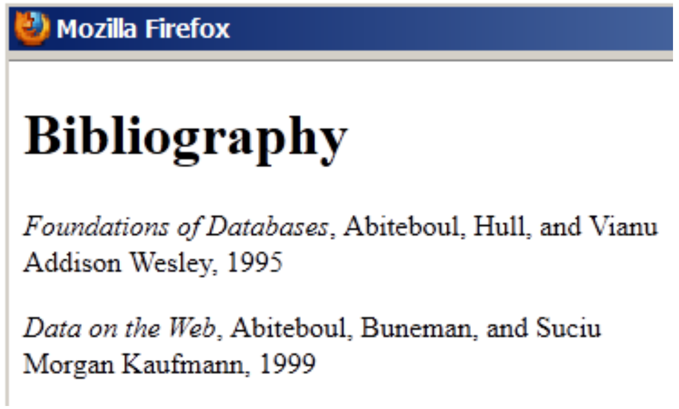
- But can’t relational databases do XML?

Json does not have many query languages which work specifically with it

```
{ "_id" : "B000944", "birthday" : ISODate("1952-11-09T00:00:00Z"), "gender" : "M", "[illegible]", "roles" : [ {
"district" : 13, "enddate" : ISODate("1995-01-03T00:00:00Z"), "party" : "Democrat", "startdate" : ISODat[illegible]993-01-05T00:00:00Z"),
"state" : "OH", "type" : "rep" }, { "district" : 13, "enddate" : ISODate("1997-01-03[illegible]0:00:00Z"), "party[illegible]emocrat", "startdate"
: ISODate("1995-01-04T00:00:00Z"), "state" : "OH", "type" : "rep" }, { "district" : [illegible]3, "enddate" : ISO[illegible]9-01-
03T00:00:00Z"), "party" : "Democrat", "startdate" : ISODate("1997-01-07T00:00:00Z"), "state[illegible] "OH", "[illegible]pe" : "[illegible]ep" }, { "district"
: 13, "enddate" : ISODate("2001-01-03T00:00:00Z"), "party" : "Democrat", "startdate"[illegible] ISOD[illegible]e("1[illegible]9-01[illegible]06T00:0[illegible]:00Z"), "state" :
"OH", "type" : "rep" }, { "district" : 13, "enddate" : ISODate("2003-01-03T00:00:00Z[illegible]), "p[illegible]ty" : "Demo[illegible]t", "[illegible]tartdate" :
ISODate("2001-01-03T00:00:00Z"), "state" : "OH", "type" : "rep" }, { "district" : 13[illegible] "endd[illegible]e" : ISODa[illegible]e("2005[illegible]01-03T00:00:00Z"),
"party" : "Democrat", "startdate" : ISODate("2003-01-07T00:00:00Z"), "state" : "OH",[illegible]"type" [illegible] "re[illegible] }, [illegible] "distr[illegible]ct" : 13, "enddate"
: ISODate("2007-01-03T00:00:00Z"), "party" : "Democrat", "startdate" : ISODate("2005[illegible]01-04T0[illegible]00:00Z"), "state" : "OH", "type" :
"rep" }, { "enddate" : ISODate("2013-01-03T00:00:00Z"), "party" : "Democrat", "start[illegible]ate" : ISODate("2007-01-04[illegible]00:00:00Z"),
"state" : "OH", "type" : "sen" }, { "current" : 1, "enddate" : ISODate("2019-01-03T0[illegible]rat", "startdate" :
ISODate("2013-01-03T00:00:00Z"), "state" : "OH", "type" : "sen" } ] }
{ "_id" : "C000127", "birthday" : ISODate("1958-10-13T00:00:00Z"), "gender" : "F", "[illegible]ame" [illegible]aria Ca[illegible]tw[illegible]l", "[illegible]oles" : [ {
"district" : 1, "enddate" : ISODate("1995-01-03T00:00:00Z"), "party" : "Democrat", "[illegible]tartd[illegible]te" [illegible]SOD[illegible]te[illegible]3-[illegible]1-05T00:00:00Z"),
"state" : "WA", "type" : "rep" }, { "enddate" : ISODate("2007-01-03T00:00:00Z"), "pa[illegible]ty[illegible]", "[illegible]tar[illegible]da[illegible]e" : ISODate("2001-
01-03T00:00:00Z"), "state" : "WA", "type" : "sen" }, { "enddate" : ISODate("2013-01-[illegible]emocrat",
"startdate" : ISODate("2007-01-04T00:00:00Z"), "state" : "WA", "type" : "sen" }, { "current" : 1, "enddate" : ISODate("2019-01-
03T00:00:00Z"), "party" : "Democrat", "startdate" : ISODate("2013-01-03T00:00:00Z"), "state" : "WA", "type" : "sen" } ] }
```

HTML: language of the Web

```
<h1>Bibliography</h1>
<p><i>Foundations of Databases</i>,
Abiteboul, Hull, and Vianu
<br>Addison Wesley, 1995
<p>…
```



Bibliography

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

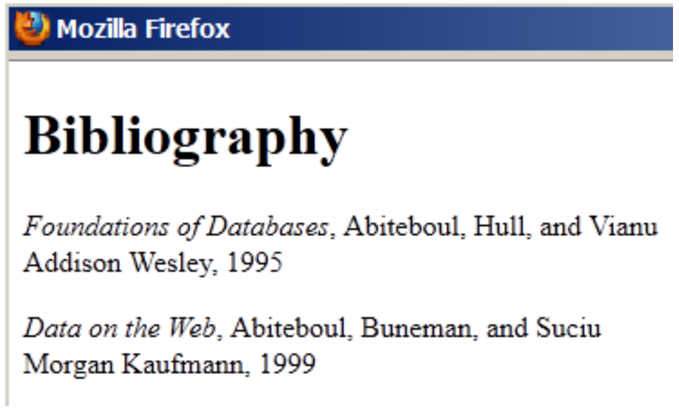
Data on the Web, Abiteboul, Buneman, and Suciu
Morgan Kaufmann, 1999

HTML can be considered a special case of XML

- It started mostly as a “formatting” language
- It mixes presentation and content
  - Hard to change presentation (say, for different displays)
  - Hard to extract content

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
- Data self-describes its structure
  - Names and nesting of tags have meanings!

Other nice features of XML

With transferring data from relational databases, you need to be sure that the other person is working with the same DMBS as you.

- 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">`

- Ordering generally matters, except for attributes

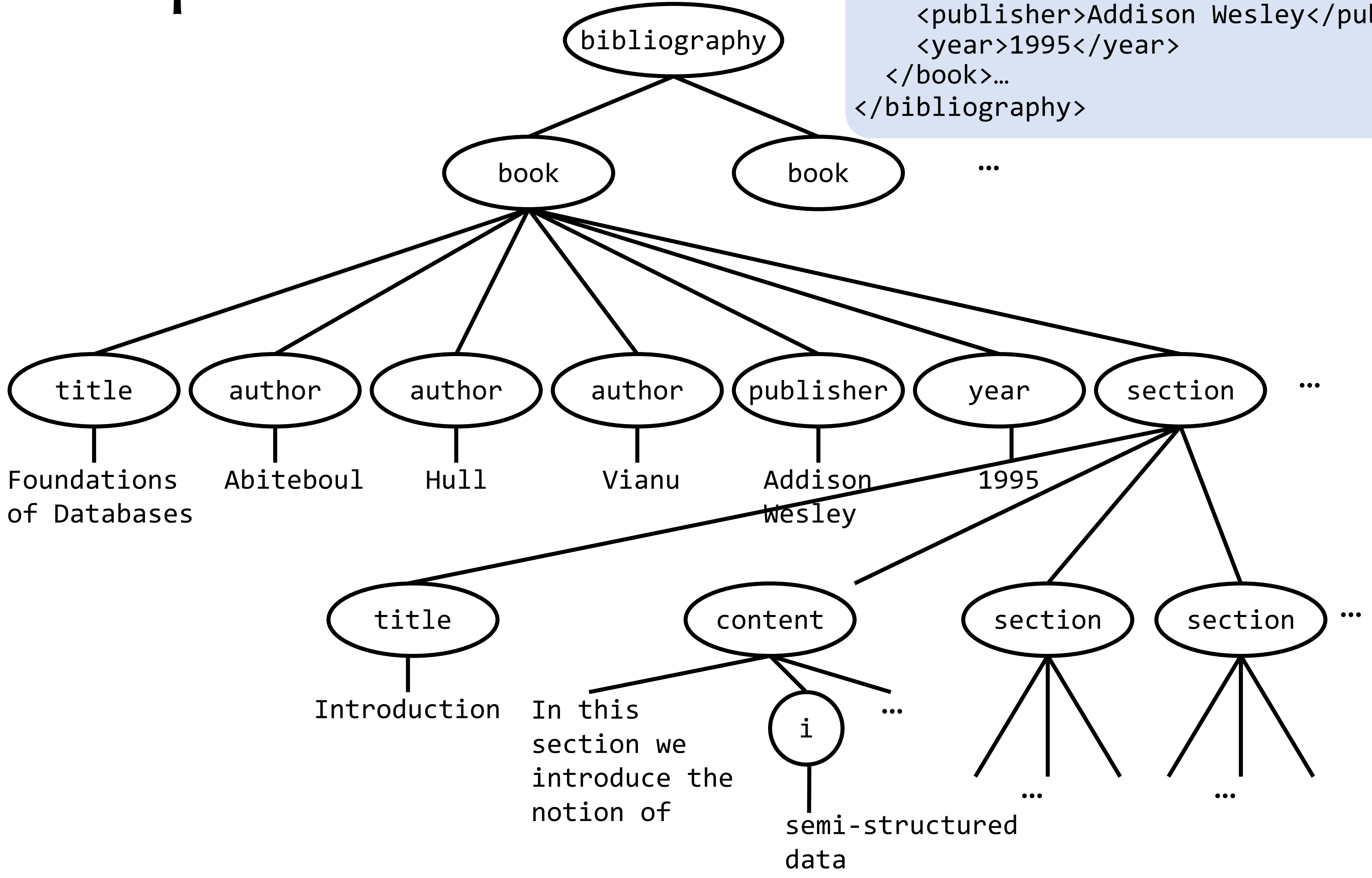
Well-formed XML documents

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
  - Right: `<section>…<subsection>…</subsection>…</section>`
  - Wrong: `<section>…<subsection>…</section>…</subsection>`

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>
```



More XML features

- Processing instructions for apps: `<? … ?>`
  - An XML file typically starts with a version declaration using this syntax: `<?xml version="1.0"?>`
- Comments: `<!-- Comments here -->`
- CDATA section: `<![CDATA[Tags: <book>,…]]>`
- ID's and references
  - ID value must start with a non-digit

CDATA section: escapes everything within the section and treats them as strings

```
<person id="o12"><name>Homer</name>…</person>
<person id="o34"><name>Marge</name>…</person>
<person id="o56" father="o12" mother="o34">
  <name>Bart</name>…
</person>…
```

- Namespaces allow external schemas and qualified names

```
<myCitationStyle:book xmlns:myCitationStyle="http://…/mySchema">
  <myCitationStyle:title>…</myCitationStyle:title>
  <myCitationStyle:author>…</myCitationStyle:author>…
</book>
```

- And more…

Valid XML documents

- A valid XML document conforms to a Document Type Definition (DTD)
  - A DTD is optional
  - A DTD specifies a grammar for the document
    - Constraints on structures and values of elements, attributes, etc.
- Example

Review this section

```
<!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*)>
]>
```

Ask professor about this constraint.
Is it not the same as (#PCDATA)*?

```
<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>
```

Text

Why use DTD or XML Schema?

- Benefits of not using them
  - Unstructured data is easy to represent
  - Overhead of validation is avoided
- Benefits of using them
  - Serve as schema for the XML data
    - Guards against errors
    - Helps with processing
  - Facilitate information exchange
    - People can agree to use a common DTD or XML Schema to exchange data (e.g., XHTML)

Text

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

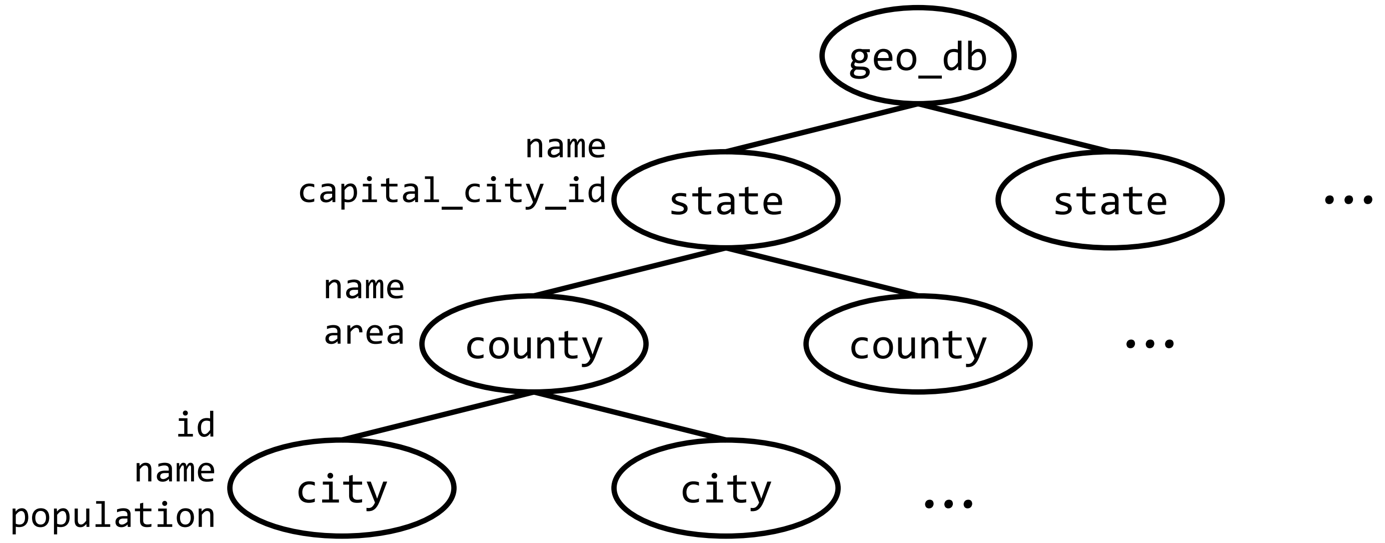
Should there not be a note saying how there may be a schema?

- 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
- Native XML database systems, or "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



Query languages for XML

- XPath
  - Path expressions with conditions
  - Building block of other standards (XQuery, XSLT, XLink, XPointer, etc.)

XQuery is powerful, but what else? Ask professor

- XQuery

Exams may contain question about writing XPath, but not about writing XQuery. Only need to be able to read XQuery

  - XPath + full-fledged SQL-like query language
- XSLT: mostly used a stylesheet language
  - XPath + transformation templates
  - We are not going to cover it in this course

Query languages for XML (online tool)

There are many online Xpath/Xquery testers, e.g.,

- https://codebeautify.org/Xpath-Tester (XPATH)
- https://videlibri.sourceforge.net/cgi-bin/xidelcgi (XQUERY)

Try with this example (or change it for different queries)

- Be careful with quotes (“” -> ")
- Not everything works all the time! Try different websites and config

```
<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>abc</section>
  </book>
  <book ISBN="ISBN-11" price="20.00">
    <title>DBSTS</title>
    <author>Ramakrishnan</author>
    <author>Gehrke</author>
    <publisher>Addison Wesley</publisher>
    <year>1999</year>
    <section>abc</section>
  </book>
</bibliography>
```

XPath

```
<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>
```

- 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`

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

```
<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>
```

Simple XPath examples

- 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

```
<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>
```

[*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

- 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`

This condition implicitly converts the author element to a string in order to compare.
If there are multiple authors, then you are comparing a set of authors against a single author.

More complex predicates

```
<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>
```

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?

If there is no value for the price, then the first query will fail.
The second query will work since you take the negation.
Review this section.

A tricky example

```
<bibliography>
  <book ISBN="ISBN-10">
    <title>Foundations of Databases</title>
    <author>Abiteboul</author>
    <author>Hull</author>
    <author>Vianu</author>
    <publisher>Addison Wesley</publisher>
    <year>1995</year>
    <price>70.00</price>
  </book>…</bibliography>
```

This slide differs from the previous one since it has that the price is a element of the book, and not an attribute. Therefore, we can have that there can be multiple prices for a book.

- 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!)

    This is comparing a value with a set
  - 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 to true for at least one node in the node-set, i.e., at least one author is "Abiteboul"
- Tricky query

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

  - Will it return any books?

Yes. In the collection of author names, you only need one author to be Abiteboul and another author to not have this name.

If you have book A authored by Abiteboul and John, this book will be returned

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

More XPath examples

- 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`
- Books with fewer than 10 sections
  `/bibliography/book[count(section)<10]`
- All elements whose parent’s tag name is not “book”
  `//*[name()!='book']/*`

De-referencing IDREF's

`id(`*identifier*`)` returns the element with *identifier*

- Suppose that books can reference other books

```
<section><title>Introduction</title>
  XML is a hot topic these days; see <bookref ISBN="ISBN-10"/> for more
details…
</section>
```

- Find all references to books written by "Abiteboul" in the book with "ISBN-10"

```
/bibliography/book[@ISBN='ISBN-10']
  //bookref[id(@ISBN)/author='Abiteboul']
```

  Or simply:

```
id('ISBN-10')//bookref[id(@ISBN)/author='Abiteboul']
```

XQuery

Sample online Xquery tester:
https://videlibri.sourceforge.net/cgi-bin/xidelcgi
Use Xquery 3.1, node format = xml, output format = adhoc, and compatibility = Standard Xquery in the settings.
Drop doc("bib.xml") in the query

- XPath + full-fledged SQL-like query language
- XQuery expressions can be
  - XPath expressions
  - FLWR expressions
  - Quantified expressions
  - Aggregation, sorting, and more…
- An XQuery expression in general can return a new result XML document
  - Compare with an XPath expression, which always returns a sequence of nodes from the input document or atomic values (boolean, number, string, etc.)

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
    - Can be omitted if there is a default context document
  - 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 doc("bib.xml")/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 doc("bib.xml")/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 doc("bib.xml")/bibliography/book,
      $p in $b/publisher
  where $b/year < 2000
  return
    <book>
      { $b/title }
      { $p }
    </book>
}</result>
```

Variables must start with dollar sign

} Nested loop

Curly brackets indicate that you want to evaluate the value inside them

- 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 := doc("bib.xml")/bibliography/book
  where $b/year < 2000
  return
    <book>
      { $b/title }
      { $b/publisher }
    </book>
}</result>
```

This condition does not return all the books published before 200, but instead gets all books if there exists one published before 2000

- 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)

Subqueries in `return`

- 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>
```

What happens if we replace it with `$a`?

Ask professor to explain again what the string function does

- `normalize-space(`*string*`)` removes leading and trailing spaces from string, and replaces all internal sequences of white spaces with one white space

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!

More features (skip)

- Existential (some) and Universal (all)
- Aggregation
- Conditional
  - Use anywhere you'd expect a value, e.g.:
  - `let $foo := if (…) then … else …`
  - `return <bar blah="{ if (…) then … else … }"/>`

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]/@p
rice)
  return
    <publisherpricing>
      <publisher>{$pub}</publisher>
      <avgprice>{$price}</avgprice>
    </publisherpricing>
}</result>
```

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>
```

XQuery vs. SQL

With SQL, you need to work on multiple tables.
With XQuery, you are more likely to work on one document

Xml data does not tell you how the data is represented nor where it is stored. Although it is
navigational, it is still a high level description

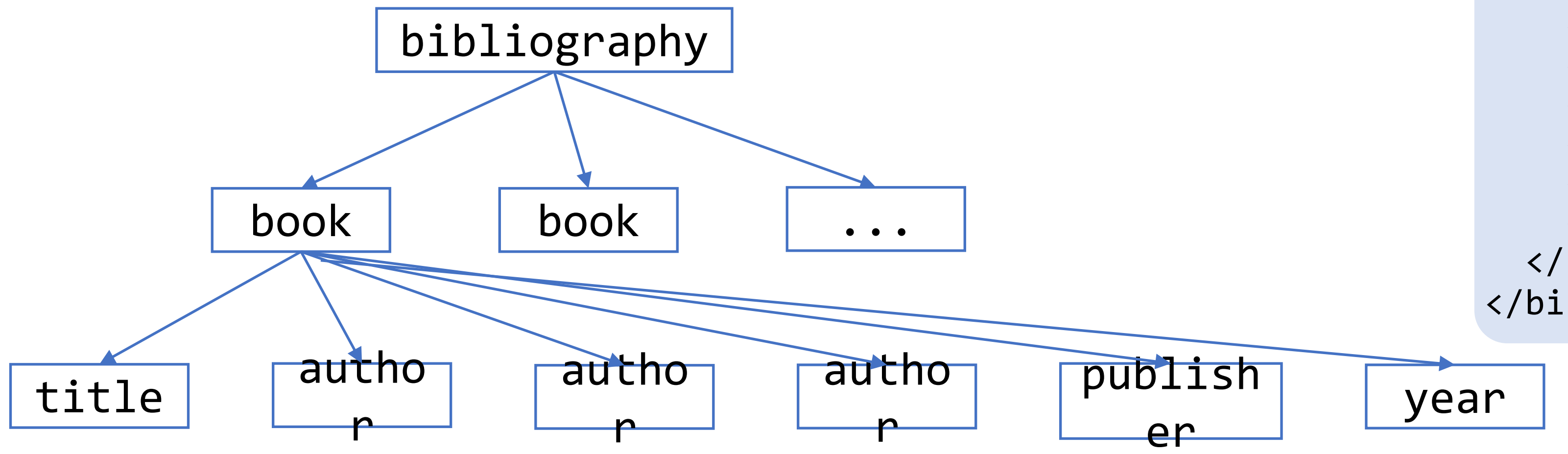
- Where did the join go?
- Is navigational query going to destroy physical data independence?
- Strong ordering constraint
  - Can be overridden by `unordered { for… }`
  - Why does that matter?

Ask professor to go over ordering constraint

Mapping XML to relational

- Store XML in a column
  - Simple, compact
  - CLOB (Character Large OBject) type + full-text indexing, or better, special XML type + functions
  - Poor integration with relational query processing
  - Updates are expensive
- Alternatives?
  - Schema-oblivious mapping:
    well-formed XML → generic relational schema
    - Node/edge-based mapping for graphs ← Focus of this lecture
    - Interval-based or Dewey-order mapping for trees
  - Schema-aware mapping:
    valid XML → special relational schema based on DTD

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>
```

- How would you translate it to a relational schema?
  - Element? Attribute? Parent-child relationship?
  - Keys?

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>
```

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

Text

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

Attribute

eid	attrName	attrValue
e1	ISBN	ISBN-10
e1	price	80

Node/edge-based: simple paths

- `//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

Node/edge-based: complex paths

- //bibliography/book[author="Abiteboul"]/@price

  -
    ```
    SELECT a.attrValue
    FROM Element e1, ElementChild c1,
         Element e2, Attribute a
    WHERE e1.tag = 'bibliography'
    AND e1.eid = c1.eid AND c1.child = e2.eid
    AND e2.tag = 'book'
    AND EXISTS (SELECT * FROM ElementChild c2,
                         Element e3, ElementChild c3, Text t
                WHERE e2.eid = c2.eid AND c2.child = e3.eid
                AND e3.tag = 'author'
                AND e3.eid = c3.eid AND c3.child = t.tid
                AND t.value = 'Abiteboul')
    AND e2.eid = a.eid
    AND a.attrName = 'price';
    ```