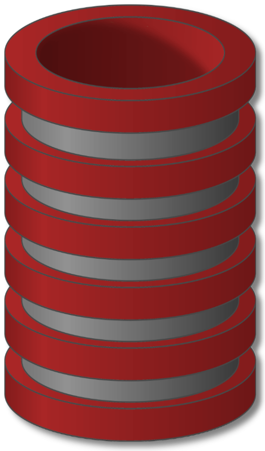
Announcements

- Group Project #1 due (today)
  - CSIF handin
- Individual Homework #1 due (Monday)
  - Homework box
  - SmartSite

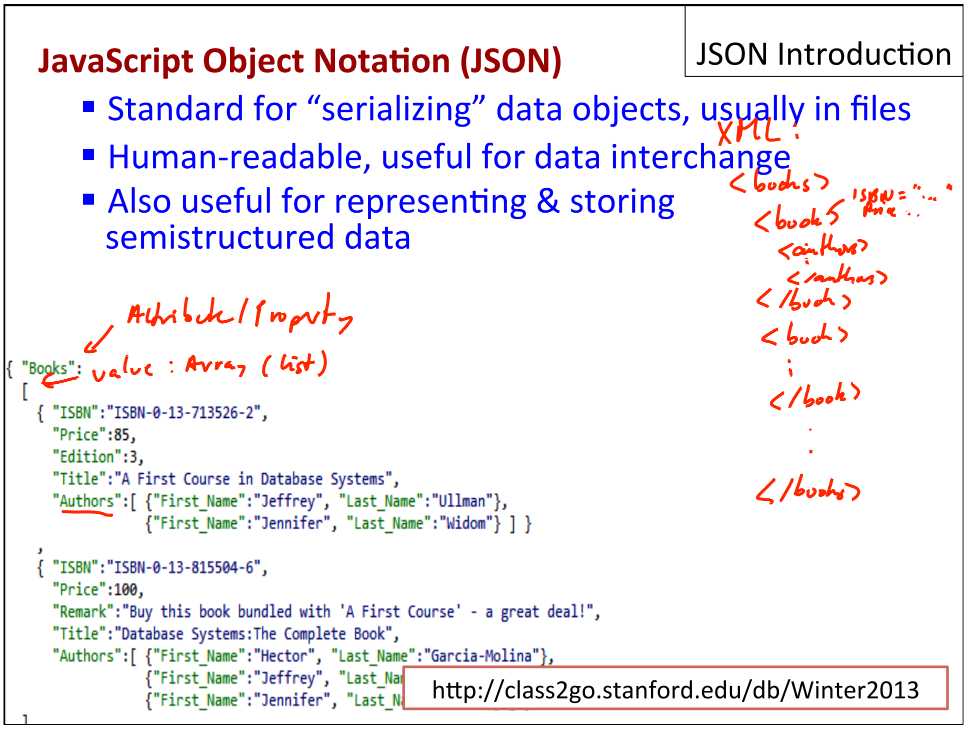
- Wrapping up XML/JSON
- Next up:
  - Relational Design Theory (normal forms)
  - Decision Support (OLAP)
  - Big Data & Map-Reduce
  - …



JSON Data

Introduction

http://class2go.stanford.edu/db/Winter2013



JSON Introduction

JavaScript Object Notation (JSON)

- Standard for "serializing" data objects, usually in files
- Human-readable, useful for data interchange
- Also useful for representing & storing semistructured data

XML:

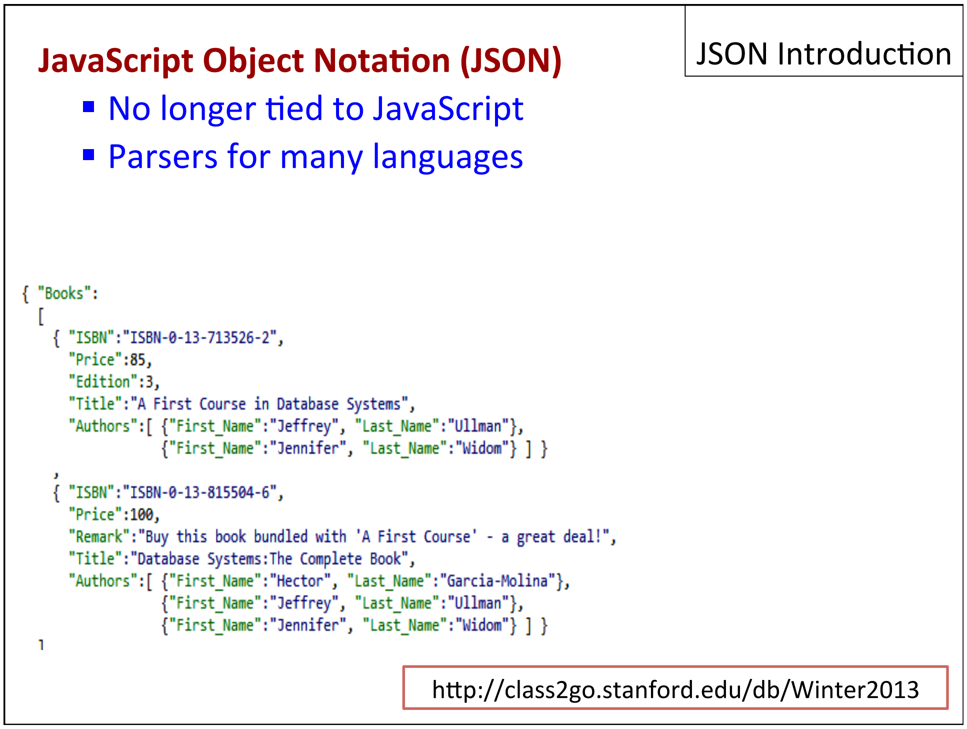
```
<books>
  <book ISBN="..."
        Price ..
    <authors>
     :
    </authors>
  </book>
  <book>
     :
  </book>
     :
     .
</books>
```

Attribute/Property

value : Array (list)

```
{ "Books":
  [
    { "ISBN":"ISBN-0-13-713526-2",
      "Price":85,
      "Edition":3,
      "Title":"A First Course in Database Systems",
      "Authors":[ {"First_Name":"Jeffrey", "Last_Name":"Ullman"},
                  {"First_Name":"Jennifer", "Last_Name":"Widom"} ] }
    ,
    { "ISBN":"ISBN-0-13-815504-6",
      "Price":100,
      "Remark":"Buy this book bundled with 'A First Course' - a great deal!",
      "Title":"Database Systems:The Complete Book",
      "Authors":[ {"First_Name":"Hector", "Last_Name":"Garcia-Molina"},
                  {"First_Name":"Jeffrey", "Last_Na
                  {"First_Name":"Jennifer", "Last_N
  ]
```

http://class2go.stanford.edu/db/Winter2013



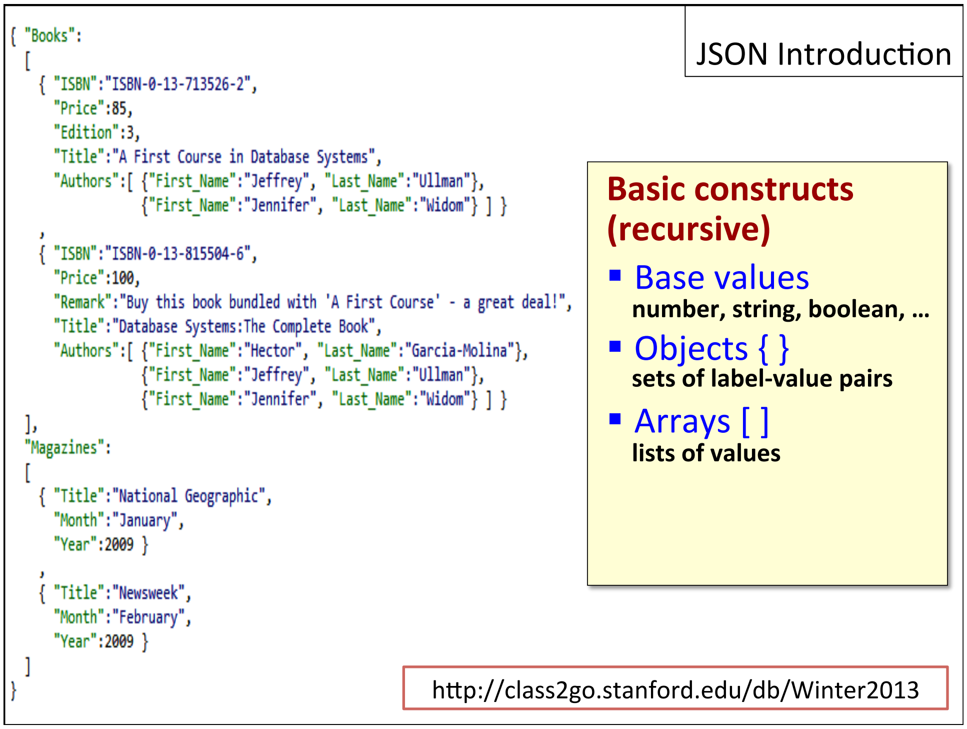
JSON Introduction

JavaScript Object Notation (JSON)

- No longer tied to JavaScript
- Parsers for many languages

```
{ "Books":
  [
    { "ISBN":"ISBN-0-13-713526-2",
      "Price":85,
      "Edition":3,
      "Title":"A First Course in Database Systems",
      "Authors":[ {"First_Name":"Jeffrey", "Last_Name":"Ullman"},
                  {"First_Name":"Jennifer", "Last_Name":"Widom"} ] }
    ,
    { "ISBN":"ISBN-0-13-815504-6",
      "Price":100,
      "Remark":"Buy this book bundled with 'A First Course' - a great deal!",
      "Title":"Database Systems:The Complete Book",
      "Authors":[ {"First_Name":"Hector", "Last_Name":"Garcia-Molina"},
                  {"First_Name":"Jeffrey", "Last_Name":"Ullman"},
                  {"First_Name":"Jennifer", "Last_Name":"Widom"} ] }
  ]
```

http://class2go.stanford.edu/db/Winter2013



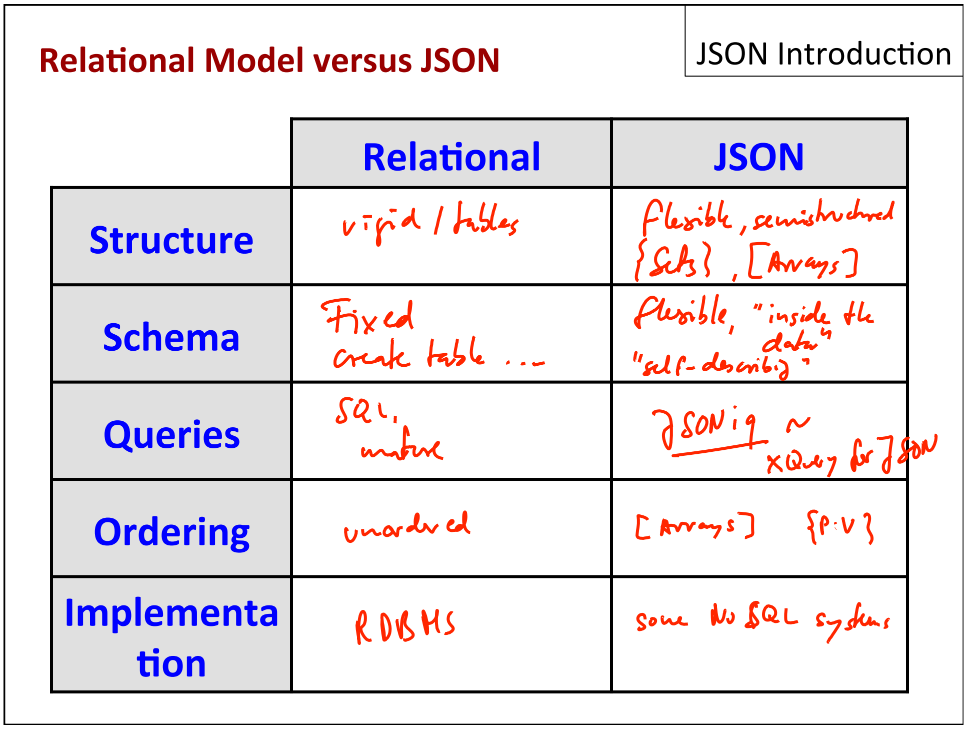
JSON Introduction

```
{ "Books":
  [
    { "ISBN":"ISBN-0-13-713526-2",
      "Price":85,
      "Edition":3,
      "Title":"A First Course in Database Systems",
      "Authors":[ {"First_Name":"Jeffrey", "Last_Name":"Ullman"},
                  {"First_Name":"Jennifer", "Last_Name":"Widom"} ] }
    ,
    { "ISBN":"ISBN-0-13-815504-6",
      "Price":100,
      "Remark":"Buy this book bundled with 'A First Course' - a great deal!",
      "Title":"Database Systems:The Complete Book",
      "Authors":[ {"First_Name":"Hector", "Last_Name":"Garcia-Molina"},
                  {"First_Name":"Jeffrey", "Last_Name":"Ullman"},
                  {"First_Name":"Jennifer", "Last_Name":"Widom"} ] }
  ],
  "Magazines":
  [
    { "Title":"National Geographic",
      "Month":"January",
      "Year":2009 }
    ,
    { "Title":"Newsweek",
      "Month":"February",
      "Year":2009 }
  ]
}
```

Basic constructs (recursive)

- Base values
  number, string, boolean, …
- Objects { }
  sets of label-value pairs
- Arrays [ ]
  lists of values

http://class2go.stanford.edu/db/Winter2013



JSON Introduction

Relational Model versus JSON

	Relational	JSON
Structure	rigid / tables	flexible, semistructured {Sets}, [Arrays]
Schema	Fixed create table ...	flexible, "inside the data" "self-describing"
Queries	SQL, mature	JSONiq ~ XQuery for JSON
Ordering	unordered	[Arrays] {P:V}
Implementation	RDBMS	some NoSQL systems

XML versus JSON

JSON Introduction

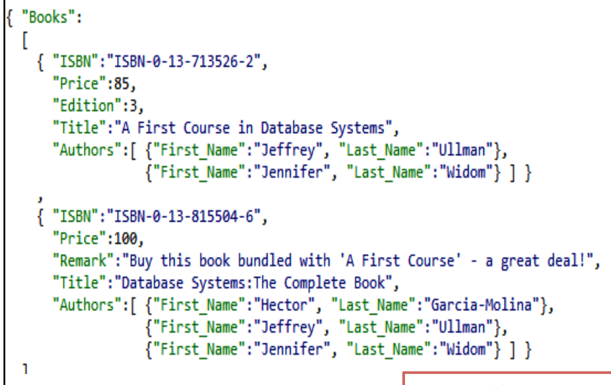
	XML	JSON
Verbosity	more	less
Complexity	more	less
Validity	DTD XML Schema	JSON schema
Prog. Interface	"impedance mismatch" ASCII → objects in PL	closer to internal format
Querying	XPath XQuery XSLT	..JSONiq..

Syntactically valid JSON

JSON Introduction

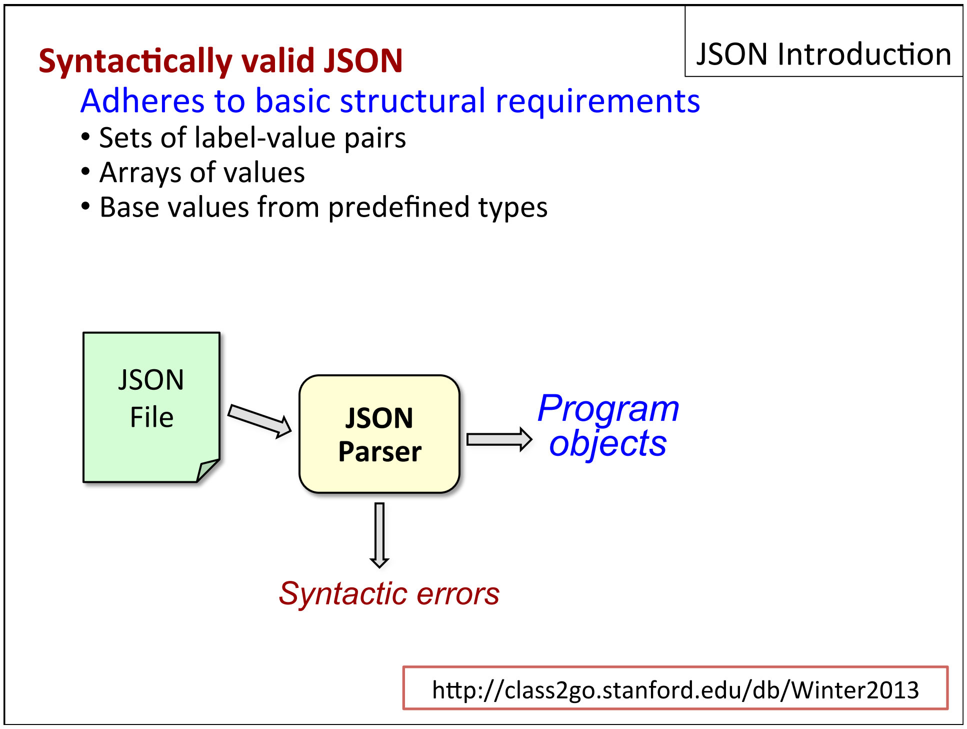
Adheres to basic structural requirements

- Sets of label-value pairs { P:V }
- Arrays of values [V]
- Base values from predefined types



```
{ "Books":
  [
    { "ISBN":"ISBN-0-13-713526-2",
      "Price":85,
      "Edition":3,
      "Title":"A First Course in Database Systems",
      "Authors":[ {"First_Name":"Jeffrey", "Last_Name":"Ullman"},
                  {"First_Name":"Jennifer", "Last_Name":"Widom"} ] }
    ,
    { "ISBN":"ISBN-0-13-815504-6",
      "Price":100,
      "Remark":"Buy this book bundled with 'A First Course' - a great deal!",
      "Title":"Database Systems:The Complete Book",
      "Authors":[ {"First_Name":"Hector", "Last_Name":"Garcia-Molina"},
                  {"First_Name":"Jeffrey", "Last_Name":"Ullman"},
                  {"First_Name":"Jennifer", "Last_Name":"Widom"} ] }
  ]
```

http://class2go.stanford.edu/db/Winter2013



JSON Introduction

Syntactically valid JSON

Adheres to basic structural requirements

- Sets of label-value pairs
- Arrays of values
- Base values from predefined types

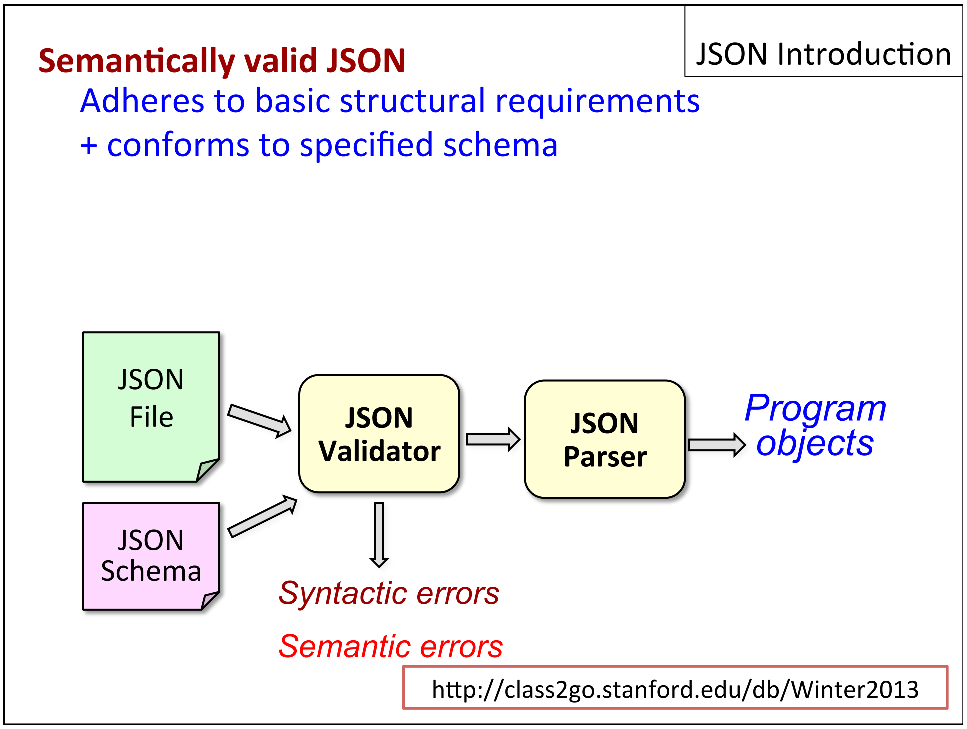
JSON File

JSON Parser

Program objects

Syntactic errors

http://class2go.stanford.edu/db/Winter2013



JSON Introduction

Semantically valid JSON

Adheres to basic structural requirements
+ conforms to specified schema

JSON File

JSON Schema

JSON Validator

JSON Parser

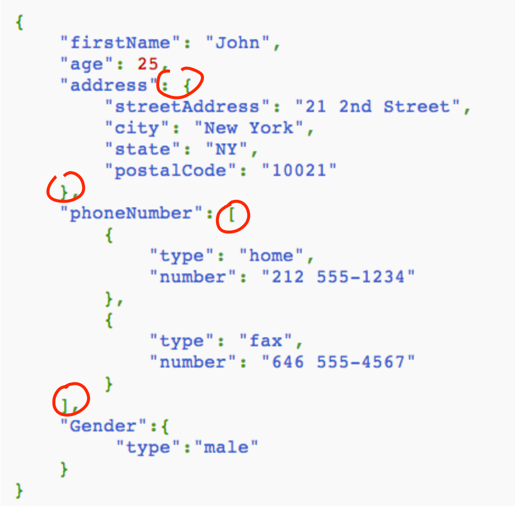
Program objects

Syntactic errors

Semantic errors

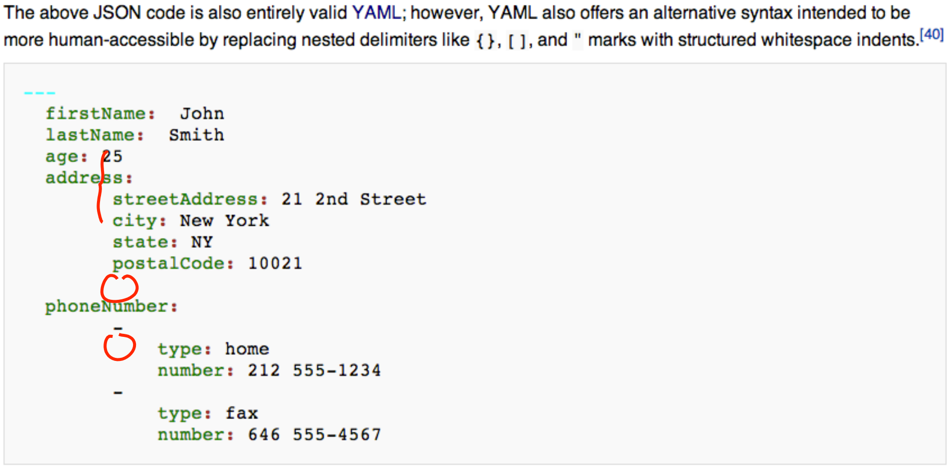
http://class2go.stanford.edu/db/Winter2013

JSON Example



```
{
    "firstName": "John",
    "age": 25,
    "address": {
        "streetAddress": "21 2nd Street",
        "city": "New York",
        "state": "NY",
        "postalCode": "10021"
    },
    "phoneNumber": [
        {
            "type": "home",
            "number": "212 555-1234"
        },
        {
            "type": "fax",
            "number": "646 555-4567"
        }
    ],
    "Gender":{
         "type":"male"
    }
}
```

YAML Example

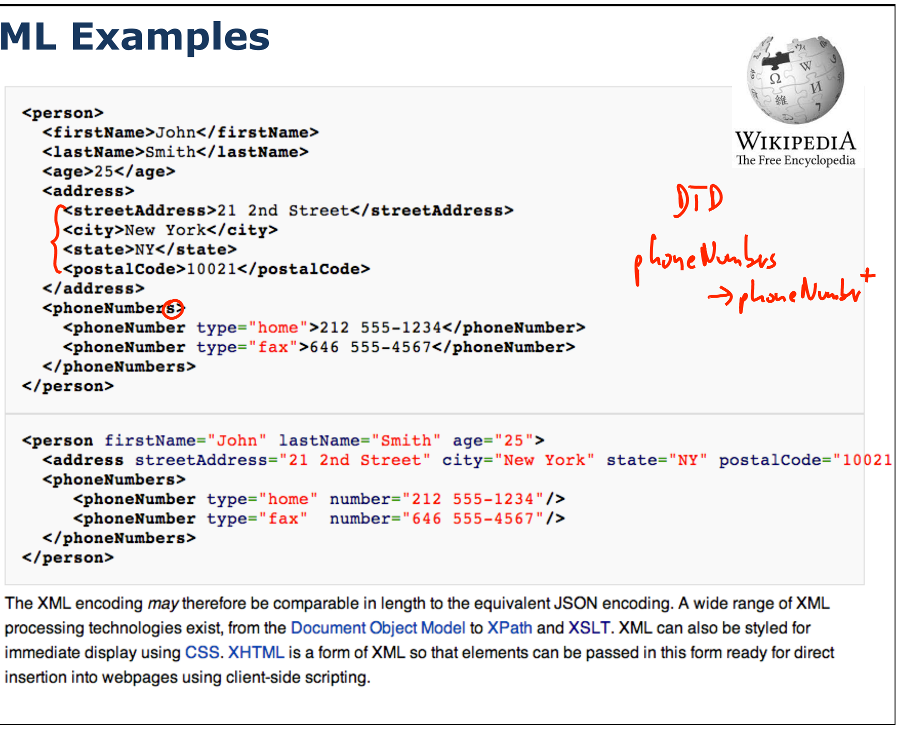


The above JSON code is also entirely valid YAML; however, YAML also offers an alternative syntax intended to be more human-accessible by replacing nested delimiters like { }, [ ], and " marks with structured whitespace indents.[40]

```
---
  firstName:  John
  lastName:  Smith
  age: 25
  address:
        streetAddress: 21 2nd Street
        city: New York
        state: NY
        postalCode: 10021

  phoneNumber:
        -
            type: home
            number: 212 555-1234
        -
            type: fax
            number: 646 555-4567
```

XML Examples



```
<person>
  <firstName>John</firstName>
  <lastName>Smith</lastName>
  <age>25</age>
  <address>
    <streetAddress>21 2nd Street</streetAddress>
    <city>New York</city>
    <state>NY</state>
    <postalCode>10021</postalCode>
  </address>
  <phoneNumbers>
    <phoneNumber type="home">212 555-1234</phoneNumber>
    <phoneNumber type="fax">646 555-4567</phoneNumber>
  </phoneNumbers>
</person>
```

```
<person firstName="John" lastName="Smith" age="25">
  <address streetAddress="21 2nd Street" city="New York" state="NY" postalCode="10021
  <phoneNumbers>
     <phoneNumber type="home"  number="212 555-1234"/>
     <phoneNumber type="fax"   number="646 555-4567"/>
  </phoneNumbers>
</person>
```

The XML encoding *may* therefore be comparable in length to the equivalent JSON encoding. A wide range of XML processing technologies exist, from the Document Object Model to XPath and XSLT. XML can also be styled for immediate display using CSS. XHTML is a form of XML so that elements can be passed in this form ready for direct insertion into webpages using client-side scripting.

DTD

phoneNumbers

→ phoneNumber+

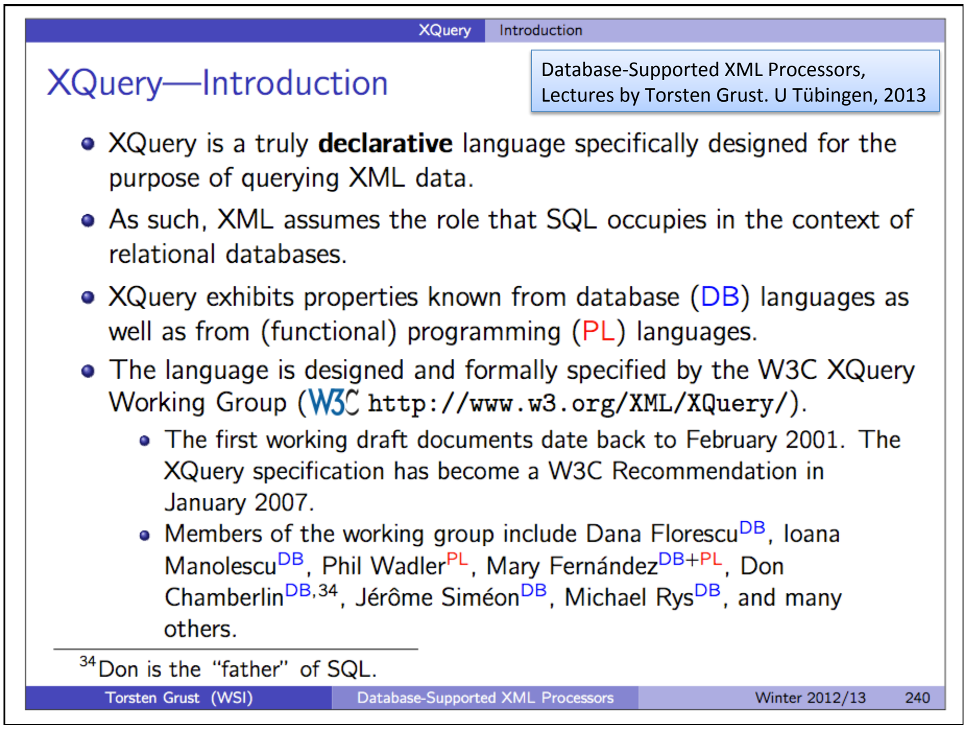
XQuery's FLOWR Power

- Recall the basic SQL clause:

SELECT..FROM..WHERE..GROUP BY..ORDER BY $\underset{select}{\Pi_{A..}}\ \underset{where}{\sigma_{Cond}}\ \underbrace{(R_1 \times R_2 .. R_n)}_{From}$

- In XML we have a similar (but different) model:
  - bind nodes (or node sequences) to variables;
  - operate over each legal combination of bindings;
  - produce a sequence of nodes
- "FLWOR" statement:

for {iterators that bind variables}
let {assignment / collections}
where {conditions}
order by {order-conditions} Group by (XQuery 3.0)
return {output constructor}

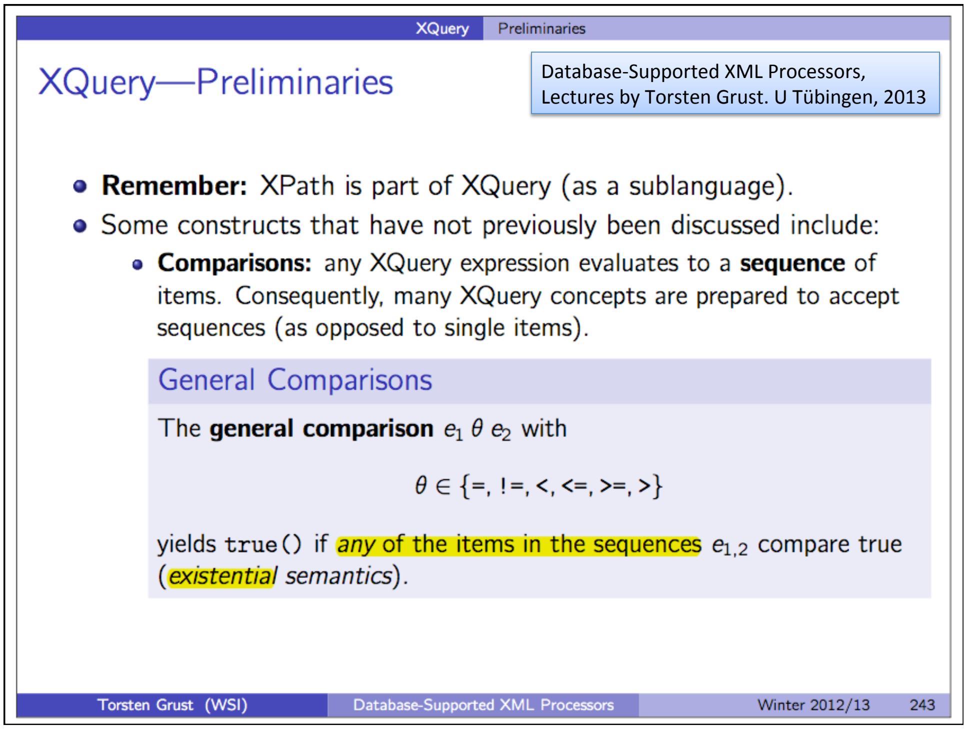


XQuery—Introduction

Database-Supported XML Processors,
Lectures by Torsten Grust. U Tübingen, 2013

- XQuery is a truly **declarative** language specifically designed for the purpose of querying XML data.
- As such, XML assumes the role that SQL occupies in the context of relational databases.
- XQuery exhibits properties known from database (DB) languages as well as from (functional) programming (PL) languages.
- The language is designed and formally specified by the W3C XQuery Working Group (W3C `http://www.w3.org/XML/XQuery/`).
  - The first working draft documents date back to February 2001. The XQuery specification has become a W3C Recommendation in January 2007.
  - Members of the working group include Dana Florescu$^{\text{DB}}$, Ioana Manolescu$^{\text{DB}}$, Phil Wadler$^{\text{PL}}$, Mary Fernández$^{\text{DB+PL}}$, Don Chamberlin$^{\text{DB}}$[34], Jérôme Siméon$^{\text{DB}}$, Michael Rys$^{\text{DB}}$, and many others.

[34] Don is the "father" of SQL.



XQuery—Preliminaries

Database-Supported XML Processors,
Lectures by Torsten Grust. U Tübingen, 2013

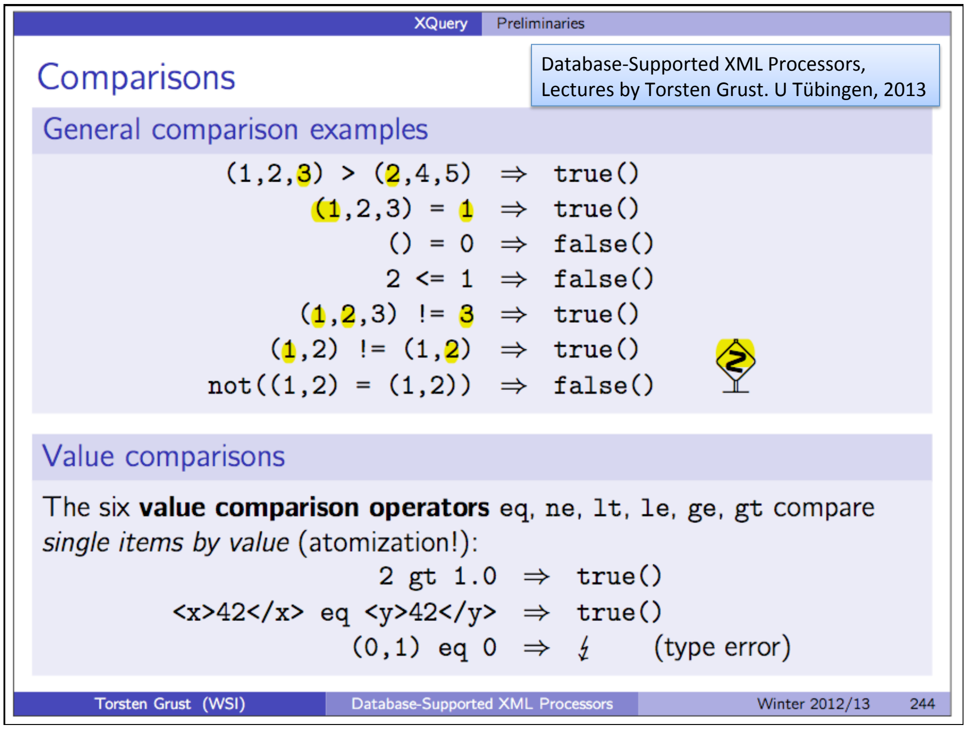
- **Remember:** XPath is part of XQuery (as a sublanguage).
- Some constructs that have not previously been discussed include:
  - **Comparisons:** any XQuery expression evaluates to a **sequence** of items. Consequently, many XQuery concepts are prepared to accept sequences (as opposed to single items).

General Comparisons

The **general comparison** $e_1\ \theta\ e_2$ with

$$\theta \in \{=, !=, <, <=, >=, >\}$$

yields `true()` if *any of the items in the sequences* $e_{1,2}$ compare true (*existential semantics*).



XQuery | Preliminaries

Comparisons

Database-Supported XML Processors, Lectures by Torsten Grust. U Tübingen, 2013

General comparison examples

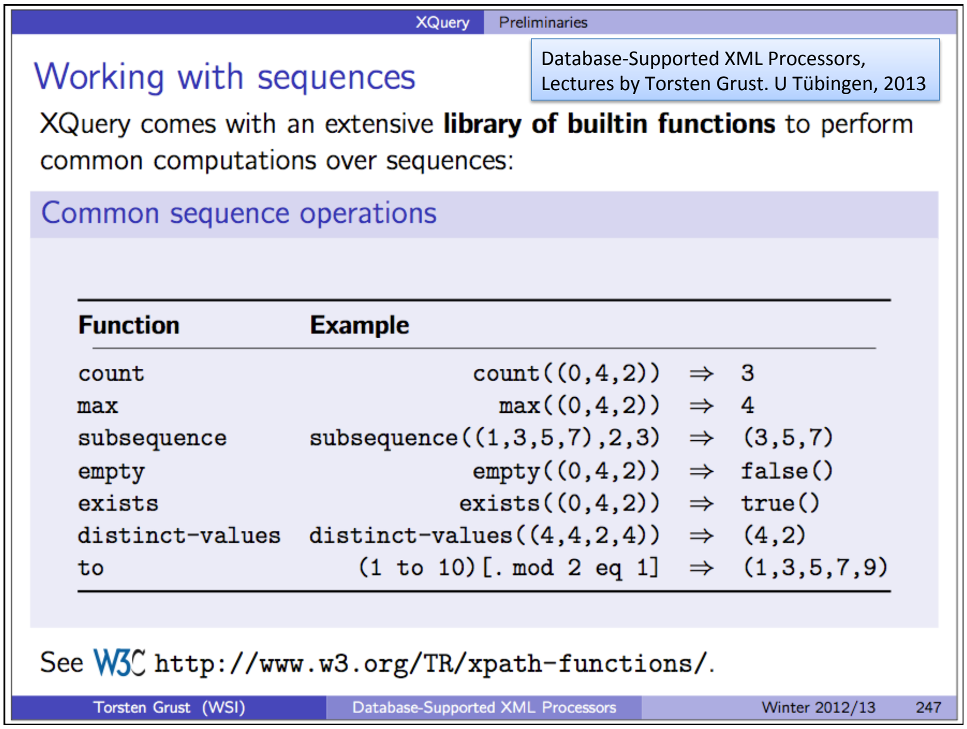
```
      (1,2,3) > (2,4,5)  ⇒  true()
          (1,2,3) = 1    ⇒  true()
               () = 0    ⇒  false()
               2 <= 1    ⇒  false()
         (1,2,3) != 3    ⇒  true()
       (1,2) != (1,2)    ⇒  true()
    not((1,2) = (1,2))   ⇒  false()
```

Value comparisons

The six **value comparison operators** `eq`, `ne`, `lt`, `le`, `ge`, `gt` compare *single items by value* (atomization!):

```
              2 gt 1.0  ⇒  true()
  <x>42</x> eq <y>42</y>  ⇒  true()
            (0,1) eq 0  ⇒  ↯     (type error)
```

Torsten Grust (WSI) Database-Supported XML Processors Winter 2012/13 244



XQuery | Preliminaries

Working with sequences

Database-Supported XML Processors, Lectures by Torsten Grust. U Tübingen, 2013

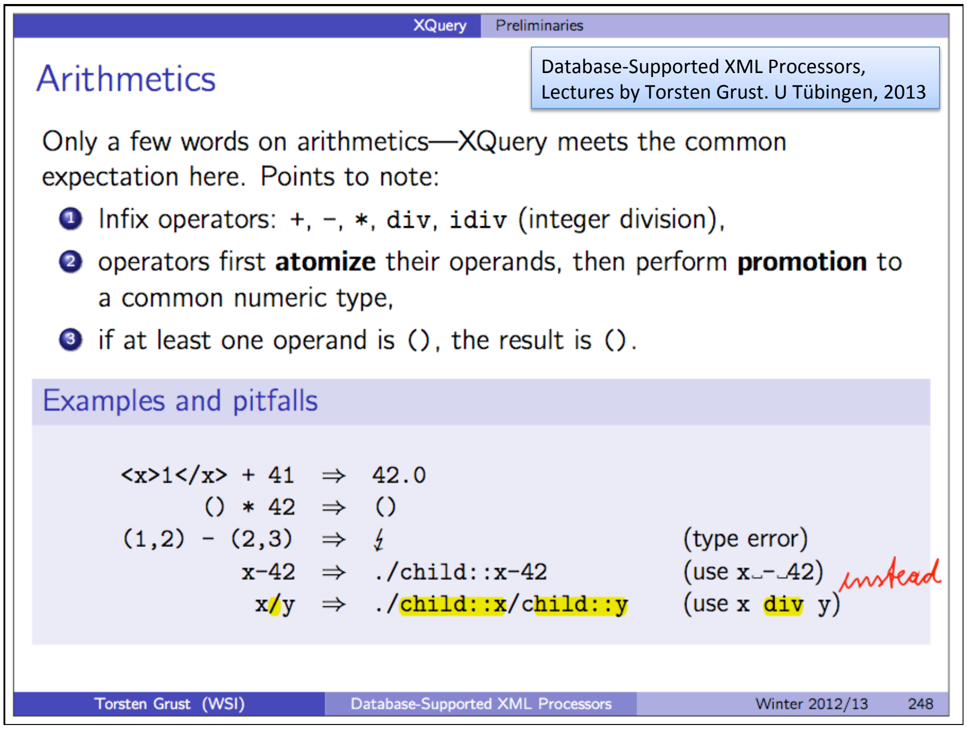
XQuery comes with an extensive **library of builtin functions** to perform common computations over sequences:

Common sequence operations

Function	**Example**
`count`	`count((0,4,2))` ⇒ `3`
`max`	`max((0,4,2))` ⇒ `4`
`subsequence`	`subsequence((1,3,5,7),2,3)` ⇒ `(3,5,7)`
`empty`	`empty((0,4,2))` ⇒ `false()`
`exists`	`exists((0,4,2))` ⇒ `true()`
`distinct-values`	`distinct-values((4,4,2,4))` ⇒ `(4,2)`
`to`	`(1 to 10)[. mod 2 eq 1]` ⇒ `(1,3,5,7,9)`

See W3C `http://www.w3.org/TR/xpath-functions/`.

Torsten Grust (WSI) Database-Supported XML Processors Winter 2012/13 247



Arithmetics

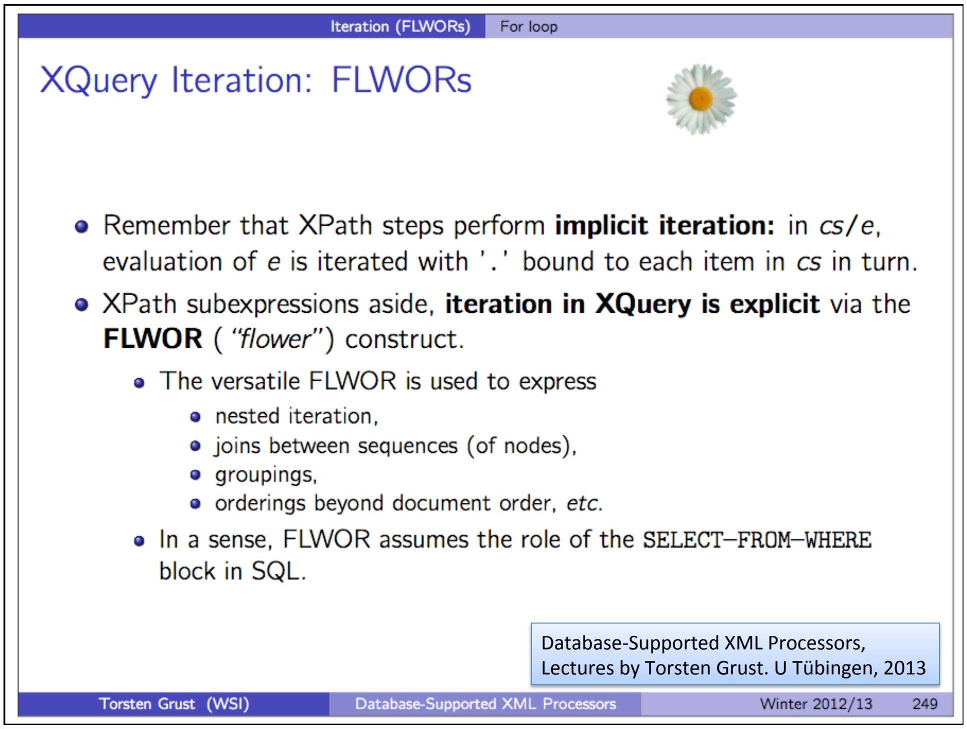
Database-Supported XML Processors, Lectures by Torsten Grust. U Tübingen, 2013

Only a few words on arithmetics—XQuery meets the common expectation here. Points to note:

1. Infix operators: `+`, `-`, `*`, `div`, `idiv` (integer division),
2. operators first **atomize** their operands, then perform **promotion** to a common numeric type,
3. if at least one operand is `()`, the result is `()`.

Examples and pitfalls

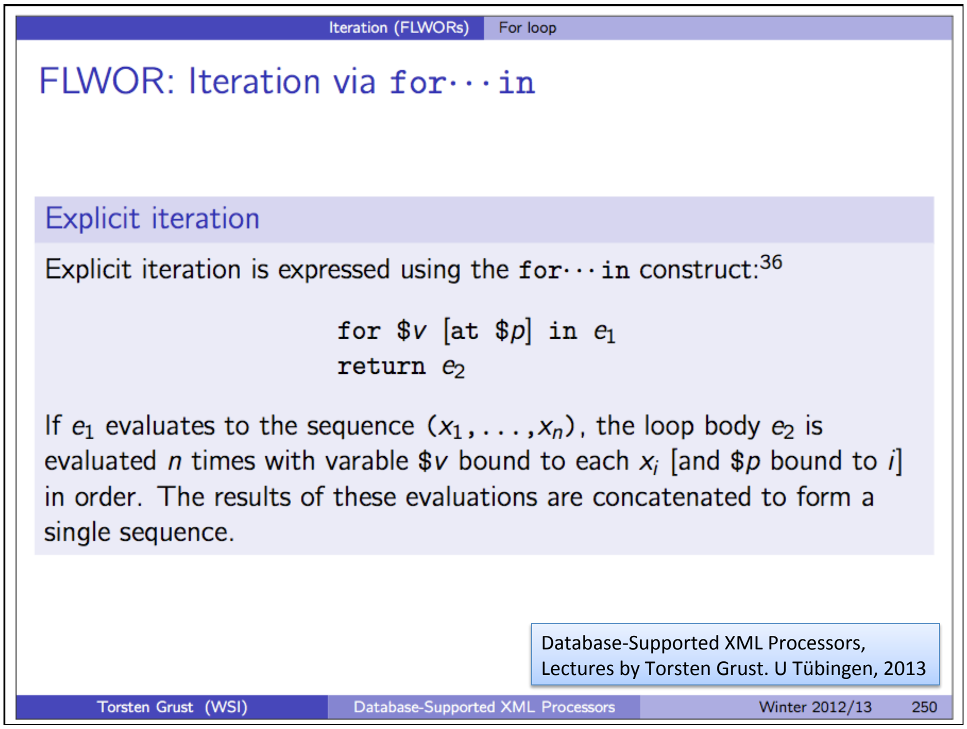
```
<x>1</x> + 41  ⇒  42.0
      () * 42  ⇒  ()
(1,2) - (2,3)  ⇒  ↯                       (type error)
         x-42  ⇒  ./child::x-42           (use x␣-␣42) instead
          x/y  ⇒  ./child::x/child::y     (use x div y)
```



XQuery Iteration: FLWORs

- Remember that XPath steps perform **implicit iteration:** in *cs*/*e*, evaluation of *e* is iterated with '`.`' bound to each item in *cs* in turn.
- XPath subexpressions aside, **iteration in XQuery is explicit** via the **FLWOR** (*"flower"*) construct.
  - The versatile FLWOR is used to express
    - nested iteration,
    - joins between sequences (of nodes),
    - groupings,
    - orderings beyond document order, *etc.*
  - In a sense, FLWOR assumes the role of the `SELECT-FROM-WHERE` block in SQL.

Database-Supported XML Processors, Lectures by Torsten Grust. U Tübingen, 2013



FLWOR: Iteration via for···in

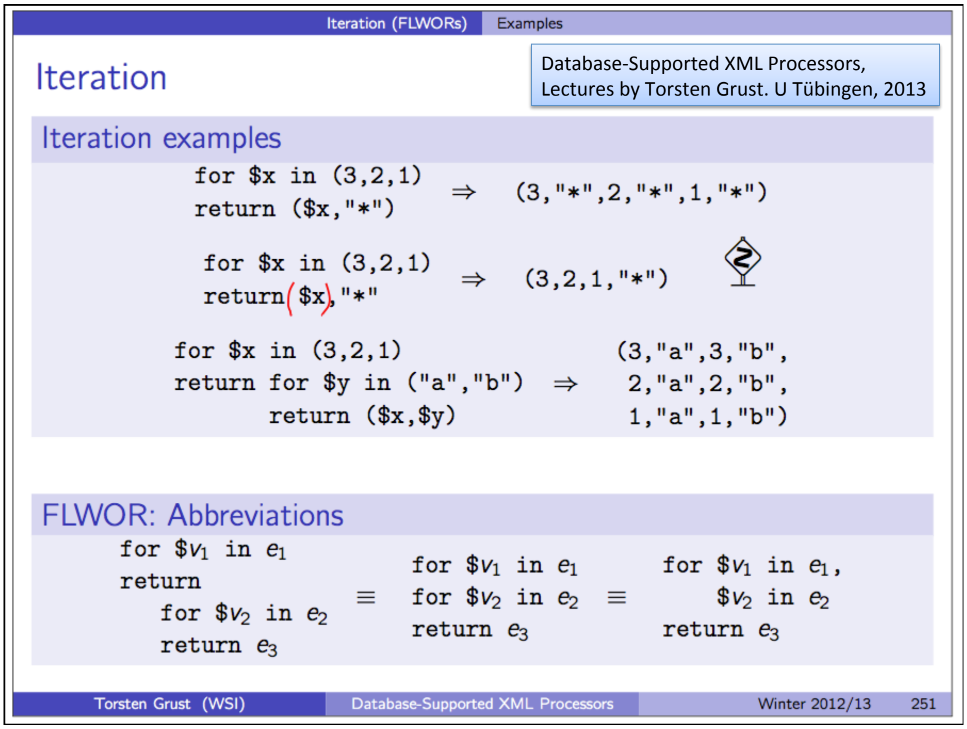
Explicit iteration

Explicit iteration is expressed using the for···in construct:[36]

```
for $v [at $p] in e1
return e2
```

If e_1 evaluates to the sequence $(x_1, \ldots, x_n)$, the loop body e_2 is evaluated n times with varable $v bound to each x_i [and $p bound to i] in order. The results of these evaluations are concatenated to form a single sequence.

Database-Supported XML Processors,
Lectures by Torsten Grust. U Tübingen, 2013



Iteration

Database-Supported XML Processors,
Lectures by Torsten Grust. U Tübingen, 2013

Iteration examples

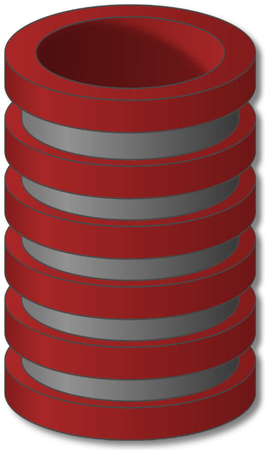
```
for $x in (3,2,1)
return ($x,"*")          ⇒  (3,"*",2,"*",1,"*")

for $x in (3,2,1)
return($x),"*"           ⇒  (3,2,1,"*")

for $x in (3,2,1)                            (3,"a",3,"b",
return for $y in ("a","b")   ⇒               2,"a",2,"b",
       return ($x,$y)                         1,"a",1,"b")
```

FLWOR: Abbreviations

```
for $v1 in e1
return                  for $v1 in e1        for $v1 in e1,
   for $v2 in e2    ≡   for $v2 in e2    ≡       $v2 in e2
   return e3            return e3            return e3
```



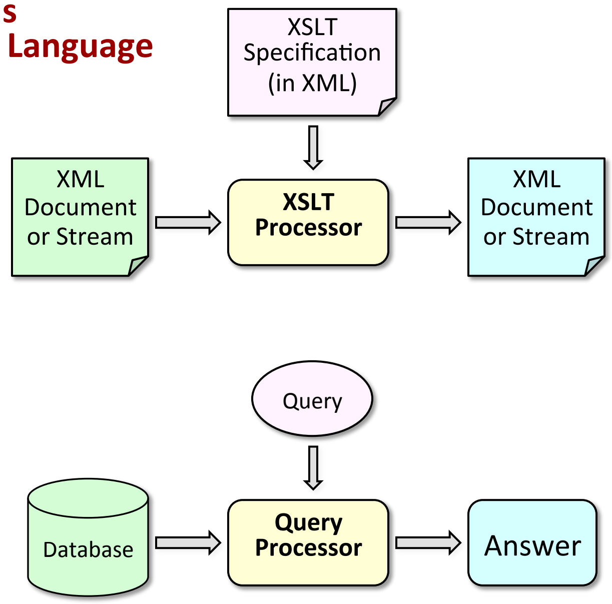
Querying XML

XSLT

http://class2go.stanford.edu/db/Winter2013

XSLT

XSLT as Query Language



XSLT Specification (in XML)

XML Document or Stream

XSLT Processor

XML Document or Stream

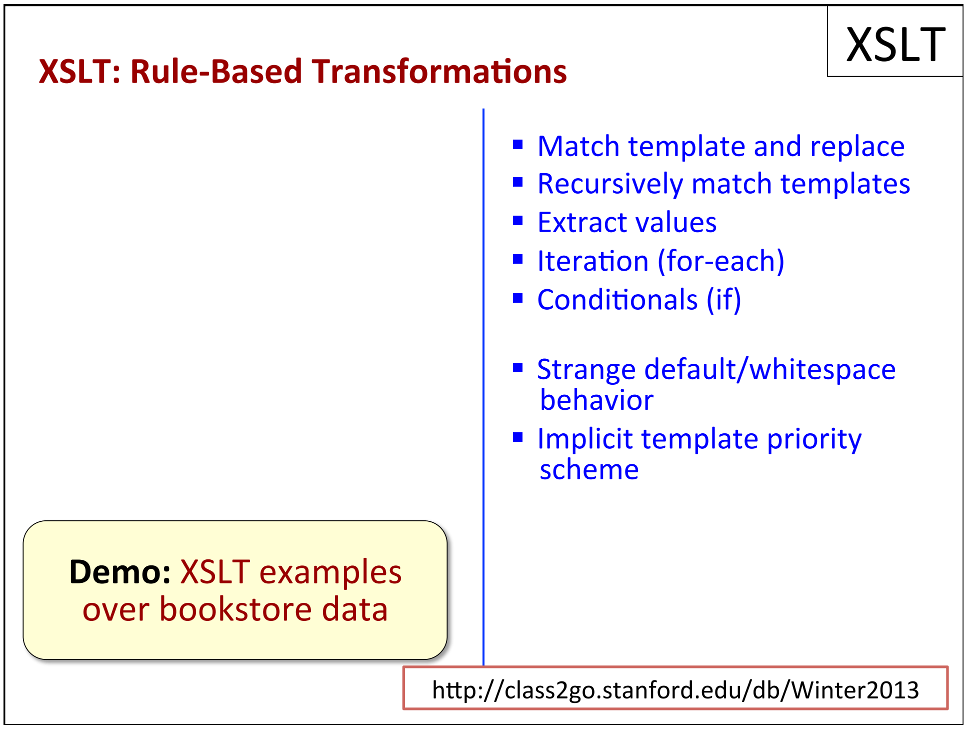
Query

Database

Query Processor

Answer

http://class2go.stanford.edu/db/Winter2013



XSLT

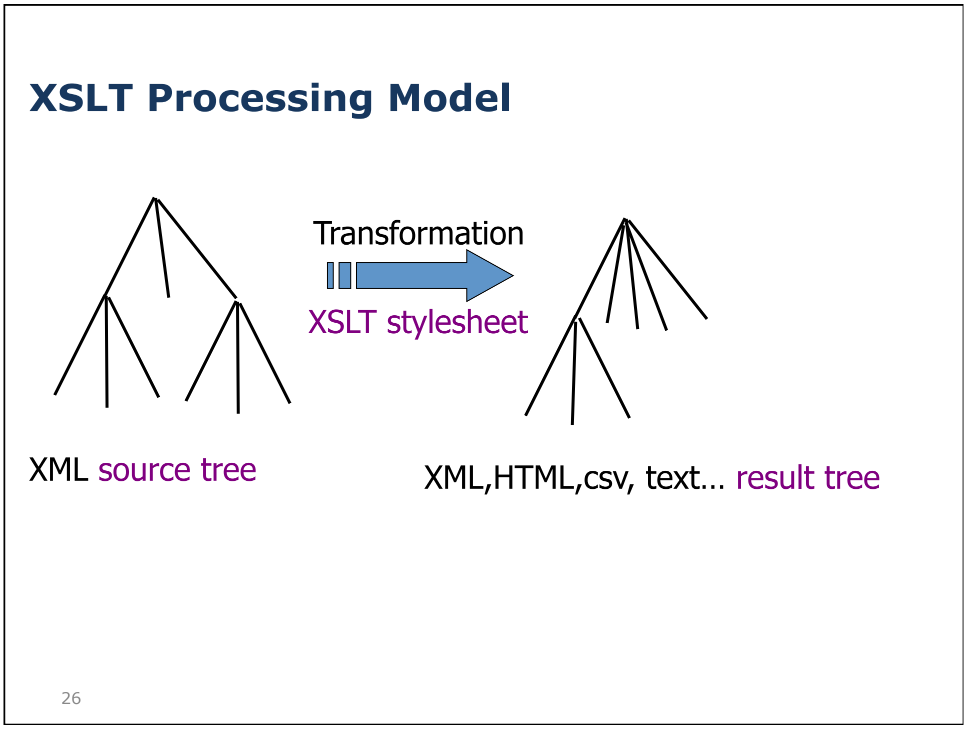
XSLT: Rule-Based Transformations

- Match template and replace
- Recursively match templates
- Extract values
- Iteration (for-each)
- Conditionals (if)

- Strange default/whitespace behavior
- Implicit template priority scheme

Demo: XSLT examples over bookstore data

http://class2go.stanford.edu/db/Winter2013



XSLT Processing Model

Transformation

XSLT stylesheet

XML source tree

XML,HTML,csv, text… result tree

XSLT Elements

- <xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  - root element of an XSLT stylesheet "program"
- <xsl:template match=*pattern* name=*qname* priority=*number* mode=*qname*>
  ...*template*...
  </xsl:template>
  - declares a rule: (*pattern* => *template*)
- <xsl:apply-templates select = *node-set-expression* mode = *qname*>
  - apply templates to selected children (default=all)
  - optional mode attribute
- <xsl:call-template name=*qname*>

XSLT Processing Model

- XSL stylesheet: collection of template rules
- template rule: (pattern ⇒ template)
- main steps:
  - match pattern against source tree
  - instantiate template (replace current node "." by the template in the result tree)
  - select further nodes for processing
- control can be a mix of
  - recursive processing ("push": **<**xsl:apply-templates**>** ...)
  - program-driven ("pull": **<**xsl:foreach**> ...)**

pattern

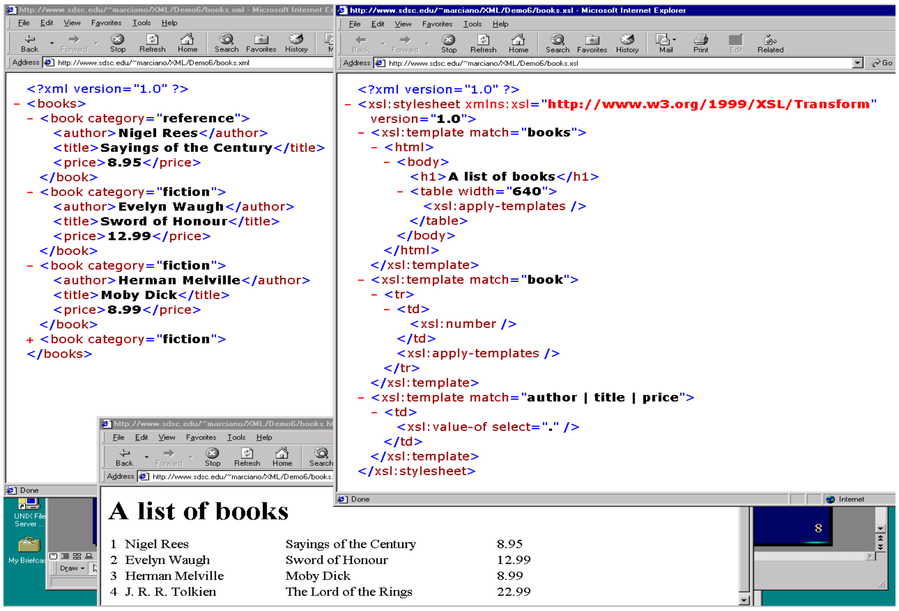
mplate Rule: Example

template

```
<xsl:template match="product">
  <table>
     <xsl:apply-templates select="sales/domestic"/>
  </table>
  <table>
   <xsl:apply-templates select="sales/foreign"/>
  </table>
</xsl:template>
```

(i) match pattern: process <product> elements
(ii) instantiate template: replace each product element with two HTML tables
(iii) select the <product> grandchildren (“sales/domestic”, “sales/foreign”) for further processing

XSLT Example


http://www.sdsc.edu/~marciano/XML/Demo6/books.xml - Microsoft Internet E
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search Favorites History
Address http://www.sdsc.edu/~marciano/XML/Demo6/books.xml
Done
http://www.sdsc.edu/~marciano/XML/Demo6/books.xsl - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search Favorites History Mail Print Edit Related
Address http://www.sdsc.edu/~marciano/XML/Demo6/books.xsl
Go
Done
Internet
http://www.sdsc.edu/~marciano/XML/Demo6/books.h
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search
Address http://www.sdsc.edu/~marciano/XML/Demo6/books.
UNIX File Server ...
My Briefca
Draw
8

```
  <?xml version="1.0" ?>
- <books>
  - <book category="reference">
      <author>Nigel Rees</author>
      <title>Sayings of the Century</title>
      <price>8.95</price>
    </book>
  - <book category="fiction">
      <author>Evelyn Waugh</author>
      <title>Sword of Honour</title>
      <price>12.99</price>
    </book>
  - <book category="fiction">
      <author>Herman Melville</author>
      <title>Moby Dick</title>
      <price>8.99</price>
    </book>
  + <book category="fiction">
  </books>
```

```
  <?xml version="1.0" ?>
- <xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    version="1.0">
  - <xsl:template match="books">
    - <html>
      - <body>
          <h1>A list of books</h1>
        - <table width="640">
            <xsl:apply-templates />
          </table>
        </body>
      </html>
    </xsl:template>
  - <xsl:template match="book">
    - <tr>
      - <td>
          <xsl:number />
        </td>
        <xsl:apply-templates />
      </tr>
    </xsl:template>
  - <xsl:template match="author | title | price">
    - <td>
        <xsl:value-of select="." />
      </td>
    </xsl:template>
  </xsl:stylesheet>
```

A list of books

1	Nigel Rees	Sayings of the Century	8.95
2	Evelyn Waugh	Sword of Honour	12.99
3	Herman Melville	Moby Dick	8.99
4	J. R. R. Tolkien	The Lord of the Rings	22.99