Querying XML Documents
**XPath**

- [http://www.w3.org/xpath](http://www.w3.org/xpath)
- Building block for other W3C standards:
  - XSL Transformations (XSLT)
  - XML Query
- Was originally part of XSL
Example doc for XPath 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 price="55">
    <publisher> Freeman </publisher>
    <author> Jeffrey D. Ullman </author>
    <title> Principles of Database and Knowledge Base Systems </title>
    <year> 1998 </year>
  </book>
</bib>
**XPath: Simple Expressions**

/bib/book/year

Result:  
\[
\text{<year> 1995 </year> }
\]

\[
\text{<year> 1998 </year> }
\]

/bib/paper/year

Result: empty  
*(there were no papers)*
**XPath: Restricted Kleene Closure**

```xml
//author

Result: <author> Serge Abiteboul </author>
  <author> <first-name> Rick </first-name>
    <last-name> Hull </last-name>
  </author>
  <author> Victor Vianu </author>
  <author> Jeffrey D. Ullman </author>

/bib//first-name

Result: <first-name> Rick </first-name>
```
**Xpath: Wildcard**

`//author/*`

**Result:** `<first-name> Rick </first-name>`

`<last-name> Hull </last-name>`

* Matches any element

`/author/*`

"authors at 3rd level"
**Xpath: Local Info About Nodes**

\[ /\text{bib}/\text{book}/\text{author}/\text{text()} \]

Result: “Serge Abiteboul”
       “Victor Vianu”
       “Jeffrey D. Ullman”

*Rick Hull doesn’t appear because he has firstname, lastname*

**Functions in XPath:**

- `text()` = matches a text value
  
  `text()` returns a string for each text element that is a direct child of the context element.

- `name()` = returns the name of the current tag

\[ /\text{bib}/\text{book}/*/\text{name()}! \rightarrow \text{“author”} \]
**Xpath: Attribute Nodes**

/\bib/book/@price

Result: “55”

@price means that there is a price attribute with a value present
**Xpath: Qualifiers**

/bib/book/author[firstname]

[firstname] means *‘has firstname element’*

Result: `<author> <first-name> Rick </first-name> <last-name> Hull </last-name> </author>`
XPath: Combining Qualifiers

/bib/book/author[firstname][address[//zip][city]]/lastname

Result: <lastname> … </lastname>

“lastname of author
(which has firstname and address (which has zip below and city))”
Xpath: Qualifiers with conditions on values

/bib/book[@price < “60”]

/bib/book[author/@age < “25”]

/bib/book[author/text()]
XPath: more tree traversal

- current node: .
- parent node: ..
- siblings?

General axes:

- self::path-step
- parent::path-step  child::path-step
- descendant::path-step  ancestor::path-step
- descendant-or-self::path-step  ancestor-or-self::path-step
- preceding-sibling::path-step  following-sibling::path-step
- preceding::path-step  following::path-step

- (previous XPaths we saw were in “abbreviated form”)

/bib//last-name//preceding::*  ~~~> <first-name>hull</fitst-name>

( /bib//last-name/  ~~~> /child::bib//descendant-or-self::last-name/ )
**Xpath: Summary**

- `bib` matches a bib element
- `*` matches any element
- `/` matches the root element
- `/bib` matches a bib element under root
- `bib/paper` matches a paper in bib
- `bib//paper` matches a paper in bib, at any depth
- `//paper` matches a paper at any depth
- `//paper/..` matches the parent of paper at any depth
- `paper | book` matches a paper or a book
- `@price` matches a price attribute
- `bib/book/@price` matches price attribute in book, in bib
- `bib/book[@price<"55"]/author/lastname` matches…
XQuery

- Based on Quilt
  (which is based on XML-QL)
- [http://www.w3.org/TR/xquery/](http://www.w3.org/TR/xquery/)

- Try out queries at
  [http://www.w3.org/TR/xquery-use-cases/](http://www.w3.org/TR/xquery-use-cases/)
- You can download your own XQuery interpreter from [http://basex.org](http://basex.org) (also available on iLab machines)
FLWOR ("Flower") Expressions

FOR ... LET... FOR... LET...
WHERE...
ORDER BY
RETURN...
Find all book titles published after 1995:

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

Result:

```
<title> abc </title>
<title> def </title>
<title> ghi </title>
```
XQuery: make better use of Xpath

Find all book titles published after 1995:

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

or even shorter

document("bib.xml")/bib/book[year > 1995]/title

Result:
<title> abc </title>
<title> def </title>
<title> ghi </title>
**XQuery: constructing answers**

“For all books published after 1995 return title and authors”

```xquery
FOR $b IN document("bib.xml")/bib/book[year > 1995]
RETURN <result>
  <the-title>{ $b/title/text() } </the-title>
  <authors>
    { $b/author }
  </authors>
</result>
```

Beware of forgetting the { and }; they mean “evaluate nested expression”

If you left { } out, you’ll get

```xml
<authors> $b/author </author>
```
XQuery: nested queries

“For each author of a book by AW, list all books she published:”

```
FOR $a IN document("bib.xml")
    /bib/book[publisher="AW"]/author
RETURN <result>
    { $a,
        FOR $t IN /bib/book[author=$a]/title
        RETURN $t
    }
</result>
```
**XQuery**

Result:

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

```
<result>
  <author>Smith</author>
  <title>ghi</title>
</result>
```
**XQuery: LET expressions**

- **FOR $x$ IN expr** -- binds $x$ *in turn to each value in the list expr*

- **LET $x := expr** -- binds $x$ *once to the entire sequence expr*
  
  – Useful for common subexpressions and for aggregations
XQuery

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

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

“Find books whose price is larger than average”:

\[
\text{LET } a := \text{avg}( \text{document("bib.xml")/bib/book/@price} )
\]
\[
\text{FOR } b \text{ in document("bib.xml")/bib/book}
\]
\[
\text{WHERE } b/@price > a
\]
\[
\text{RETURN } b
\]
XQuery

Summary:
- **FOR-LET-WHERE-RETURN** = FLWR

Diagram:

```
FOR/LET Clauses ➔ List of tuples
  ➔ WHERE Clause ➔ List of tuples
  ➔ RETURN Clause ➔ Instance of Xquery data model
```
FOR vs. LET

FOR
  • Binds *node variables* → iteration

LET
  • Binds *collection variables* → one value

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

Returns:
  <result> <book>... </book></result>
  <result> <book>... </book></result>
  <result> <book>... </book></result>
  ...

LET $x$ := document("bib.xml")/bib/book
RETURN <result> { $x$ } </result>

Returns:
  <result> <book>... </book></result>
  <book>... </book>
  <book>... </book>
  ...
  </result>
Collections in XQuery

- Ordered and unordered collections
  - `/bib/book/author` ~~~> an ordered collection
  - `distinct_values(/bib/book/author/text())` ~~~> an unordered collection
- `LET $b := /bib/book` ~~~> $b is a collection
- `$b/author` ~~~> a collection (authors of all books)

Returns:
<result>
<author>...</author>
<author>...</author>
<author>...</author>
...
</result>
**distinct_values($arg)**

- The $arg sequence can contain atomic values or nodes, or a combination of the two. The nodes in the sequence have their typed values extracted. This means that only the *contents* of the nodes are compared, not any other properties of the nodes (for example, their names).

  e.g. Let $in-xml := <in-xml> <a>3</a> <b>5</b> <b>3</b> </in-xml>  

  Then distinct-values($in-xml/*) = (3, 5)
**Sequences in Xquery**

- \(1,2,3 = (1,2,3) = (1, (2,3), () )\)
- () can be used as sort of a null; \((() + 2 = ()\)
- but boolean logic is 2-valued: \((() \text{ and true()} \) yields \(false()\)
- although there are automatic coercions for tests
  
  ```
  if x then ... else
  x  a sequence ~> check for non-null
  x  a number ~> check for non-zero
  (yuck!)
  ```
If-Then-Else

FOR $h$ IN //catalogoue
RETURN <catalogue>
{
  $h/title,
  IF $h/@type = "Journal"
    THEN $h/editor
    ELSE $h/author
}
</catalogue>
Existential Quantifiers

“Books which have some paragraph containing both the words sailing and windsurfing”

FOR $b$ IN //book

WHERE SOME $p$ IN $b$/para SATISFIES

contains($p$, "sailing")

AND contains($p$, "windsurfing")

RETURN $b$/title
Universal Quantifiers

"Books in which all paragraphs contain the word sailing"

FOR $b$ IN //book
WHERE EVERY $p$ IN $b$//para SATISFIES contains($p$, "sailing")
RETURN $b$/title
Widom’s DTD

<!ELEMENT Bookstore (Book | Magazine)*>  
<!ELEMENT Book (Title, Authors, Remark?)>  
<!ATTLIST Book ISBN CDATA #REQUIRED  
    Price CDATA #REQUIRED  
    Edition CDATA #IMPLIED>  
<!ELEMENT Magazine (Title)>  
<!ATTLIST Magazine Month CDATA #REQUIRED  
    Year CDATA #REQUIRED>  
<!ELEMENT Title (#PCDATA)>  
<!ELEMENT Authors (Author+)>  
<!ELEMENT Remark (#PCDATA)>  
<!ELEMENT Author (First_Name, Last_Name)>  
<!ELEMENT First_Name (#PCDATA)>  
<!ELEMENT Last_Name (#PCDATA)>
<?xml version="1.0" standalone="no"?>
<!DOCTYPE Bookstore SYSTEM "bookstore.dtd">
<Bookstore>
    <Title>A First Course in Database Systems</Title>
    <Authors>
      <Author>
        <First_Name>Jeffrey</First_Name>
        <Last_Name>Ullman</Last_Name>
      </Author>
      <Author>
        <First_Name>Jennifer</First_Name>
        <Last_Name>Widom</Last_Name>
      </Author>
    </Authors>
  </Book>
</Bookstore>
Comparisons

- If one operand is a single value and the other is a sequence, the result of the comparison is true if there exists some member of the sequence for which the comparison with the single operand is true.
- If both operands are sequences, the comparison is true if there exists some member of the first sequence and some member of the second sequence for which the comparison is true.
Value Comparisons (=, !=, <, <=, >, and >=)

- If both operands are simple values of the same type, the result is straightforward.
- If one operand is a node and the other is a simple value, the content of the node is extracted by an implicit invocation of the “data” function before the comparison is performed. (“data” of a node [basically] returns the concatenated contents of all its descendant text nodes, in document order, as untypedAtomic.)
- If both operands are nodes, the string-values of the nodes are compared. (The string-value of a node is the concatenated contents of all its descendant text nodes, in document order, as string.)
Node Identity Comparison (== and !=)

- Defined only for nodes or sequences of nodes
- If both operands of == are nodes, the comparison is true only if both operands are the same node (not just nodes with the same name and value)
- If either or both operands is a node sequence, the rules stated above apply.
**Flattening**

- “Flatten” the authors, i.e. return a list of (author, title) pairs

```
FOR $b IN document("bib.xml")/bib/book,
    $x IN $b/title,
    $y IN $b/author
RETURN <answer>
    <title> {data($x)} </title>
    <author> {data($y)} </author>
</answer>
```

Result:
```
<answer>
    <title> abc </title>
    <author> efg </author>
</answer>
<answer>
    <title> abc </title>
    <author> hkj </author>
</answer>
```
Re-grouping

- “For each author, return all titles of her/his books”

```sql
FOR $b IN document("bib.xml")/bib,
   $x IN $b/book/author
RETURN

<answer>
   <author> {data($x)} </author>
   { FOR $y IN $b/book[author=$x]/title
      RETURN $y }
</answer>
```

Result:
```xml
<answer>
   <author> efg </author>
   <title> abc </title>
   <title> klm </title>
   . . . .
</answer>
```

What about duplicate authors?
• Same, but eliminate duplicate authors:

```xml
FOR $b$ IN document("bib.xml")/bib
LET $a := \text{distinct-values}(\text{$b$/book/author/text()} )
FOR $x$ IN $a$
RETURN
  <answer>
    <author> { $x$ } </author>
    { FOR $y$ IN $b$/book[author=$x$]/title
      RETURN $y$ }
  </answer>
```

*distinct-values* eliminates duplicates  (but must be applied to a collection of *text* values, not of *elements*)
Re-grouping

- Same thing:

```xml
FOR $b$ IN document("bib.xml")/bib,
    $x$ IN distinct-values($b/book/author/text())
RETURN
  <answer>
    <author> { $x$ } </author>
    { FOR $y$ IN $b/book[author=$x]/title
      RETURN $y$ }
  </answer>
```
Another Example

“Find book titles by the coauthors of ‘Database Theory’”

FOR $x$ IN bib/book[title/text() = “Database Theory”]/author
   $y$ IN bib/book[author/text() = $x$/text()]/title
RETURN <answer> { $y$/text() } </answer>

Result:
<answer> abc </answer>
<answer> def </answer>
<answer> abc </answer>
<answer> ghk </answer>

The answer will contain duplicates!
Distinct-values

Same as before, but eliminate duplicates:

\[
\text{FOR } \$x \text{ IN } \text{bib/book[title/text() = “Database Theory”]/author} \\
\text{\hspace{1em} $y \text{ IN distinct-values}(\text{bib/book[author/text() = $x$/text()]/title/text())} \\
\text{RETURN <answer> \{ $y \} </answer>}
\]

Result:

\[
\text{distinct-values} = \text{a function that eliminates duplicates}
\]

\[
\text{Need to apply to a collection of text values, not of elements – note how query has changed}
\]
**SQL and XQuery Side-by-side**

Find all product names, prices

Product(pid, name, maker, price)

```sql
SELECT x.name, x.price
FROM Product x
```

```xml
<db>
  <Product>
    <row>
      <row>
        <pid 1234 /> <name 'bulb'/> <maker
    </row>
  </Product>
</db>
```

```xquery
FOR $x in document("db.xml")/db/Product/row
RETURN <answer>
  { $x/name, $x/price }
</answer>
```
**Xquery’s Answer**

```xml
<answer>
  <name> abc </name>
  <price> 7 </price>
</answer>
<answer>
  <name> def </name>
  <price> 23 </price>
</answer>

 Notice: this is NOT a well-formed document!
(WHY ???)
```
Producing a Well-Formed Answer

<myQuery>
{ FOR $x$ in document("db.xml")/db/Product/row
  RETURN <row>
    { $x/name, $x/price } 
  </row>
} 
</myQuery>
Xquery’s Answer

<myQuery>
  <row>
    <name> abc </name>
    <price> 7 </price>
  </row>
  <row>
    <name> def </name>
    <price> 23 </price>
  </row>
  ....
</myQuery>

Now it is well-formed!
SQL and XQuery Side-by-side

“Find all product names, prices sorted by price”

Product(pid, name, maker, price)

**SQL**

```
SELECT x.name, x.price
FROM Product x
ORDER BY price
```

**XQuery**

```
FOR $x in $db/Product/row
ORDER BY $x /price/text()
RETURN <a>
{ $x/name, $x/price }</a>
```
\textbf{Answer:}

\begin{verbatim}
<answer>
  <name> abc </name>
  <price> 7 </price>
</answer>

<answer>
  <name> def </name>
  <price> 23 </price>
</answer>

\ldots
\end{verbatim}

\textbf{Notice: this is NOT a well-formed document! (WHY ???)}
Producing well formed doc

```xml
<result>
  { FOR $x$ in document(“db.xml”)/db/Product/row
      ORDER BY $x$/price/text()
      RETURN <a>
        { $x$/name, $x$/price }
    </a>
  }
</result>
```

```
<result>
  <a>
    <name> abc </name>
    <price> 7 </price>
  </a>
  <a>
    <name> def </name>
    <price> 23 </price>
  </a>
  ...
</result>
```
**SQL and XQuery Side-by-side**

"Find all products made in Seattle"

Product(pid, name, maker, price)  
Company(cid, name, city, revenues)

**SQL**

```
SELECT x.name  
FROM Product x, Company y  
WHERE x.maker=y.cid  
and y.city="Seattle"
```

**XQuery**

```
FOR $x in $db/Product/row,  
  $y in $db/Company/row  
WHERE  
  $x/maker=$y/cid  
and $y/city = "Seattle"  
RETURN { $x/name }  
```

**Compact XQuery**

```
FOR $y in /db/Company/row[city="Seattle"],  
  $x in /db/Product/row[maker=$y/cid]  
RETURN $x/name  
```
<product>
  <row> <pid> 123 </pid>
      <name> abc </name>
      <maker> efg </maker>
  </row>
  <row> …. </row>
  …
  <product>
    <product>
      . . .
    </product>
  </product>
  . . .
**SQL and XQuery Side-by-side**

For each company with revenues < 1M count the products over $100

```
SELECT c.name, count(*)
FROM Product p, Company c
WHERE p.price > 100 and p.maker=c.cid and c.revenue < 1000000
GROUP BY c.cid, c.name
```

```
FOR $r in document(“db.xml”)/db,
  $c in $r/Company/row[revenue<1000000]
RETURN
  <proudCompany>
    <companyName> { $c/name } </companyName>
    <numberOfExpensiveProducts>
      { count($r/Product/row[maker=$c/cid][price>100]) }    
    </numberOfExpensiveProducts>
  </proudCompany>
```
**SQL and XQuery Side-by-side**

Find companies with at least 30 products, and their average price

```sql
SELECT y.name, avg(x.price)
FROM Product x, Company y
WHERE x.maker=y.cid
GROUP BY y.cid, y.name
HAVING count(*) > 30
```

```xquery
FOR $r in document("db.xml")/db,
    $y in $r/Company/row
LET $p := $r/Product/row[maker=$y/cid]
WHERE count($p) > 30
RETURN
    <theCompany>
        <companyName> {$y/name } </companyName>
        <avgPrice> avg($p/price) </avgPrice>
    </theCompany>
```