XQuery

What is XQuery?

- **XQuery** for XML is **like** SQL for databases
- Language for **querying XML data**
- Built on **XPath expressions**
- Defined by the W3C
- Supported by all the major database engines (IBM, Oracle, Microsoft, etc.)
- XQuery is a W3C recommendation (14 Dec 2010) thus a standard

XML Query Language

- Advantages
  - Query selective portions of the document (no need to transport entire document)
  - Smaller data size mean **lesser communication cost**

Some examples of usage

- Extract information to use in a Web Service
- Generate summary reports
- Transform XML data to XHTML
- Search Web documents for relevant information

XQuery compared to XPath

- **XQuery 1.0** and **XPath 2.0** share the same data model and support the same functions and operators.
- XQuery 1.0 is a ***strict superset*** of XPath 2.0
- The extra expressive power is the ability to:
  - ***Join*** information from different sources and
  - ***Generate*** new XML fragments

XQuery

- Prolog
  - XQuery expressions are evaluated relatively to a context
  - explicitly provided by a prolog (header)
    - header with definitions
- Body
  - The actual query
    - Generate
    - Join
    - Select

Selecting Nodes with XQuery

- **Functions**
  - XQuery uses functions to **extract data** from XML documents.
- **(X)Path Expressions**
  - XQuery uses path expressions to **navigate** through elements in an XML document.
- **Predicates**
  - XQuery uses predicates to **limit** the extracted data from XML documents.

Functions

- **doc()**
  - function to open a file
- Example:
  - **doc(**"books.xml"**)**

Path Expressions

- Example:

  select all the title elements in the "books.xml" file:

  doc("books.xml")**/bookstore/book/title**

Predicates

- Example:

  select all the book elements under the bookstore element that have a price element with a value that is less than 30 :

  doc("books.xml")/bookstore/book**[price<30]**

XQuery Comments

(: XQuery Comments :)

The output document (LibraryOut.xml)

```
<?xml version=”1.0” encoding=”UTF-8”?>
<library>
    <book  year=”2007”>
    <title>Beginning XML, 4th Edition</title>
    </book>
    <book  year=”2006”>
    <title>Beginning Beginning XML Databases</title>
    </book>
</library>
```

Computed Constructors

Library.xquery: create a simple library using element and attribute constructors

```
element library{
element book {
    attribute year {2007},
    element title {
        "Beginning XML, 4thEdition"
    }
},
element book {
    attribute year {2006},
    element title {
        "Beginning XML Databases"
    }
}
}
```

FLWOR

- The **main engine** of XQuery is the FLWOR expression:
- For, Let, Where, Order by, Return
- Pronounced "**flower**"
- Generalizes **SELECT-FROM-WHERE-HAVING from SQL**
- Similar to SQL syntax

FLWOR by comparison with Path expressions

- Select from books.xml all the **title elements** under the book elements that are under the bookstore element that have a price element with a value that is higher than 30.

- Path expression:

```
doc("books.xml")/bookstore/book[price>30]/title
```

- FLWOR expression:

```
for $x in doc("books.xml")/bookstore/book
where $x/price>30
return $x/title
```

for vs. let

- **for: bind multiple variables**
- **let: bind single variables**
- for $x in list-expr
  - Binds $x in turn to each value in the list expr

- let $x := list-expr
  - Binds $x to the entire list expr
  - Useful for common sub-expressions and aggregations

The Difference between for and let

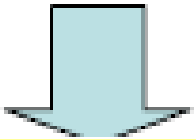
```
for $x in (1, 2, 3, 4)
let $y := ("a", "b", "c")
return ($x, $y)
```

The Difference between for and let

```
let $x := (1, 2, 3, 4)
for $y in ("a", "b", "c")
return ($x, $y)
```

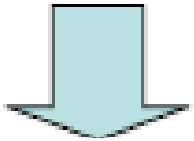
The Difference between for and let

```
for $x in (1, 2, 3, 4)
for $y in ("a", "b", "c")
return ($x, $y)
```



The Difference between for and let

```
let $x := (1, 2, 3, 4)
let $y := ("a", "b", "c")
return ($x, $y)
```



Filtering with where Clause

Suppose you wanted to find any books in BibAdapted.xml that were published by Wrox Press.

PublisherOut.xml

```
<?xml version=”1.0” encoding=”UTF-8”?>
<books>
    <book year=”2004”>
    <title>XSLT 2.0 Programmer’s Reference</title>
    </book>
    <book year=”2006”>
    <title>Professional Web 2.0 Programming</title>
    </book>
    <book year=”2007”>
    <title>Beginning XML, 4th Edition</title>
    </book>
</books>
```

XQuery

Suppose you wanted to find any books in BibAdapted.xml that were published by Wrox Press.

Publisher.xquery

```
<books>{
for $book in doc("BibAdapted.xml")/bib/book
    where $book/publisher = "Wrox Press" return
element book {
   attribute year {$book/@year},element title {$book/title/text()}
   }
}
</books>
```

Sorting in FLWOR

OrderByTitle.xquery : the output is sorted alphabetically by title

```
<books>{
        for $book in doc("BibAdapted.xml")/bib/book
        let $t := $book/title/text() order by $t
        return
        <book><title>{$t}</title></book>
        }
</books>
```

OrderByTitleOut.xml

```
<?xml version=”1.0” encoding=”UTF-8”?>
<books>
    <book>
        <title>Beginning XML, 4th Edition</title>
        </book>
    <book>
        <title>Professional Web 2.0 Programming</title>
    </book>
    <book>
        <title>The C Programming Language</title>
    </book>
    <book>
        <title>The Economics of Technology and Content for Digital TV</title>
    </book>
    <book>
        <title>XSLT 2.0 Programmer’s Reference</title>
    </book>
</books>
```

reverse alphabetical order

- order by $t descending

Conditional Expressions

- Conditional expressions in XQuery use **if** keyword.

count()

- count() function: calculate the number of elements present
- More functions:

  www.w3.org/tr/xpath-functions.

Query (MultiAuthor.xquery)

Write a query on BibAdapted.xml that outputs a book's title and a count of its authors only if the number of authors exceeds two.

output (MultiAuthorOut.xml)

```
<?xml version=”1.0” encoding=”UTF-8”?>
<MultiAuthor>
    <book>
        <title>Professional Web 2.0 Programming</title>
        <NumberOfAuthors>4</NumberOfAuthors>
    </book>
    <book>
        <title>Beginning XML, 4th Edition</title>
        <NumberOfAuthors>6</NumberOfAuthors>
    </book>
</MultiAuthor>
```

Query (MultiAuthor.xquery)

Write a query on BibAdapted.xml that outputs a book's title and a count of its authors only if the number of authors exceeds two

```
<MultiAuthor>
    {for $book in doc("BibAdapted.xml")/bib/book
    return
    if (count($book/author) gt 2)
      then <book>
    <title>{$book/title/text()}</title>
    <NumberOfAuthors>{count($book/author)}</NumberOfAuthors>
    </book>
    else ()
    }
</MultiAuthor>
```

The concat() Function

The concat() function is used to concatenate strings.

Parts.xml

```
<?xml version=”1.0”?>
<Parts>
    <Part>To be or not to be,</Part>
    <Part>that is the question!</Part>
</Parts>
```

ASayingOut.xml

```
<?xml version=”1.0” encoding=”UTF-8”?>
<ASaying>To be or not to be, that is the question!</ASaying>
```

Asaying.xquery

```
<ASaying>{
    for $a in doc("Parts.xml")/Parts/Part[1]
    for $b in doc("Parts.xml")/Parts/Part[2]
    return concat($a, " ", $b)
}</ASaying>
```

- //**:**Select any descendant, not only children

  article//index (any index element in articles)

Query

```
<doubles>
{
 for $s in  doc("students.xml")//student
 let $m := $s/major
 where count($m) ge 2
 order by $s/@enrollno
 return <double>
          { $s/name/text()}
        </double>
}
</doubles>
```

Above query selects names of students from the students.xml document that have at least two majors **and** orders the results by student enroll number

Joining documents

XQuery allows the joining of documents

XML files: taxpayers.xml, neighbors.xml (2 files)

o/p xml file:

1. aadharno,
2. name (from the neighbors.xml database), and
3. income (from the taxpayers.xml database).

Joining documents

```
for $p in doc("www.incometax.gov.in/taxpayers.xml")//person
for $n in doc("neighbors.xml")//neighbor[aadharno= $p/ aadharno]
return
  <person>
    < aadharno > { $p/aadharno } </ aadharno >
    <name>{ $n/name }</name>
    <income> { $p/income } </income>
</person>
```

Two-way join in where Clause

```
for $item in doc("ord.xml")//item,
    $product in doc("cat.xml")//product
where $item/@num = $product/number
return
  <item num="{$item/@num}"
        name="{$product/name}"
        quan="{$item/@quantity}" />
```

Sample result:

```
<item num="557" name="Samsung TV" quan="1" />
<item num="563" name="Diamond Ring" quan="1" />
```

Aggregation

- Summary calculations on grouped data
- Functions: sum, avg, max, min, count

Conditionals

```
for $b in doc("bibadapted.xml")/book
    return
       <short>
          {$b/title}
          <authors>
           {if ( count($b/author) < 3 )
             then    $b/author
             else
                ( $b/author[1], <author>and others</author>)
           }
         </authors>
       </short>
```

Nesting Conditional Expressions

- Conditional expressions can be nested
- ‘else if’ functionality is provided

- **if (** count($b/author) = 1)
  then $b/author
  else if (count($b/author) = 2 **)**then (: .. :)
  else ($b/author[1], <author>and others</author>)

Logical Expressions

- and, or operators:

if ($isDiscounted **and** ($discount > 5 **or** $discount < 0))
then 5 else $discount

- not function for negations:

if (**not**($isDiscounted)) then 0 else $discount

XQuery Built-in Functions

XQuery function namespace URI is:
http://www.w3.org/2005/02/xpath-functions

default prefix: **fn:**.

- **E.g.:** fn:string() or fn:concat().
- fn: is the default prefix of the namespace, the function names **does not** need to be prefixed when called.

Built-in Functions

- String-related
  - substring, contains, matches, concat, normalize-space, tokenize
- Date-related
  - current-date, month-from-date, adjust-time-to-timezone
- Number-related
  - round, avg, sum, ceiling
- Sequence-related
  - index-of, insert-before, reverse, subsequence, distinct-values

Built-in Functions (2)

- Node-related
  - data, empty, exists, id, idref
- Name-related
  - local-name, in-scope-prefixes, QName, resolve-QName
- Error handling and trapping
  - error, trace, exactly-one
- Document and URI-related
  - collection, doc, root, base-uri

Function calls

```
doc("books.xml")//book[substring(title,1,5)='Harry']

let $name := (substring($booktitle,1,4))

<name>{upper-case($booktitle)}</name>
```

fn:data

- let $cat := doc('http://www.functx.com/input/catalog.xml') return

XQuery Example	Results
data($cat//product[1]/number)	557
data($cat//number)	(557, 563, 443, 784)
data($cat//product[1]/@dept)	WMN
data($cat//product[1]/colorChoices)	navy black
data($cat//product[1])	557 Fleece Pullover navy black
data($cat//product[4]/desc)	Our favorite shirt!

```
for $x in doc(“books.xml")//book/title
for $y in data($x)
for $name in (substring($y,1,4))

return $name
```

T2 Starts here...

User Defined Functions

We can define functions in the prolog, and then use them in the body of the program

```
declare function prefix:function_name($parameter
  AS datatype)
  AS returnDatatype
    {     (: ...function code here... :)    };
```

User-defined Functions

```
declare function local: minPrice(   $price as xs:decimal,
                                    $discount as xs:decimal)
 AS xs:decimal
{
let $disc := ($price * $discount)/100
return ($price - $disc)
};
```

(: Below is an example of how to call the function above :)

<minPrice>{local:minPrice($book/price, $book/discount)}
</minPrice>

Existential and Universal Quantifiers

- for $b in doc("bib.xml")/book
  where **some** $author in $b/author
    satisfies $author/text() = "Ullman"
  return $b

Return books where at least one author is "Ullman"

- for $b in doc("bib.xml")/book
  where **every** $author in $b/author
    satisfies $author/text() = "Ullman"
  return $b

Return books where all authors are "Ullman"

Comparisons

- Value comparisons

  eq, ne, lt, le, gt, ge

  Used to compare **individual values**

  Each operand must be a single atomic value (or a node containing a single atomic value)

- General comparisons

  =, !=, <, <=, >, >=

  Can be used with sequences of **multiple items**

Example

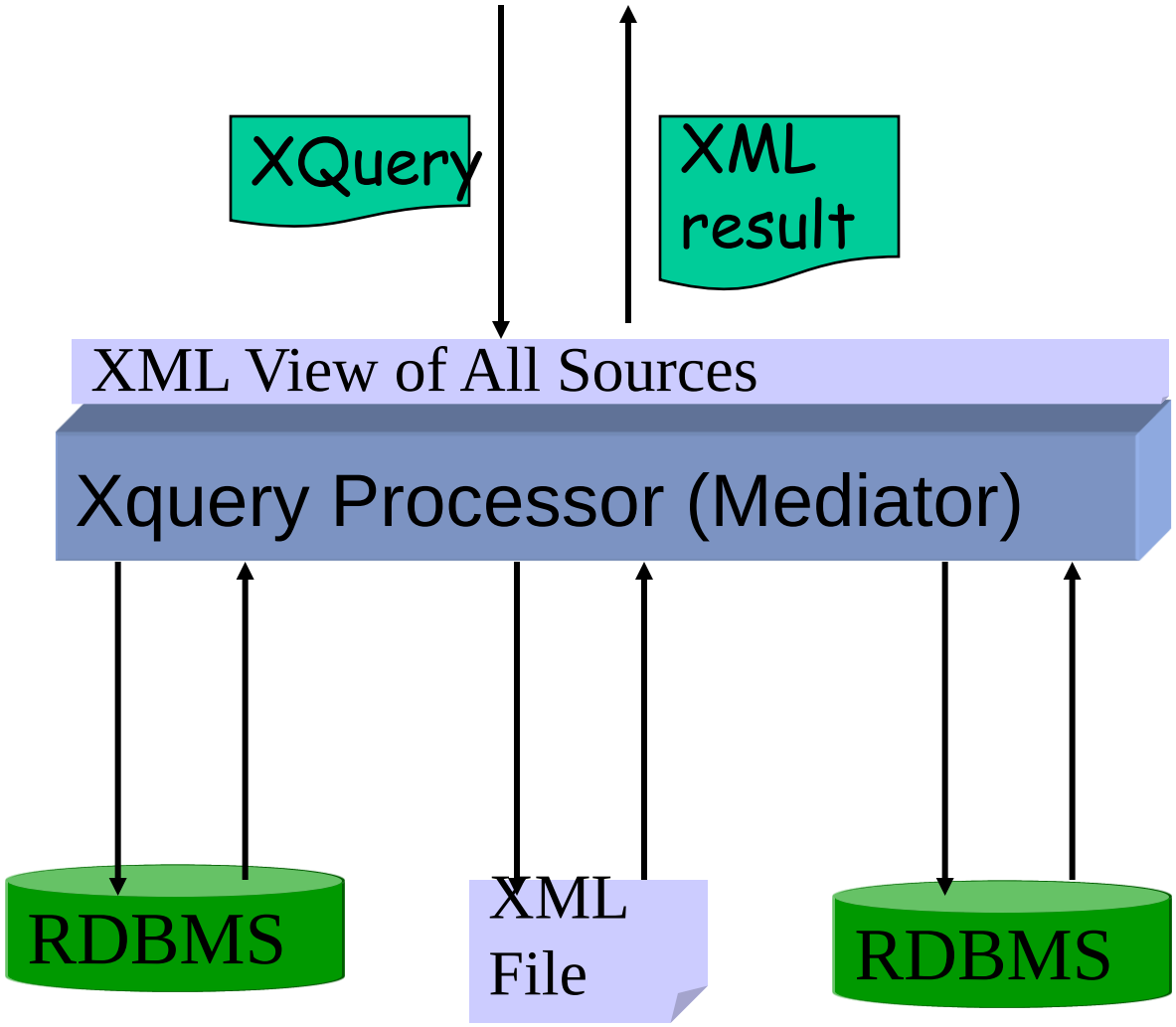
```
doc("ord.xml")//item/@quantity > 1
```

- returns true if any `quantity` attributes have values greater than 1

```
doc("ord.xml")//item/@quantity gt 1
```

- returns true if there is only *one* `quantity` attribute returned by the expression, and its value is greater than 1
- if more than one `quantity` is returned, an error is raised

XQuery on Distributed Sources



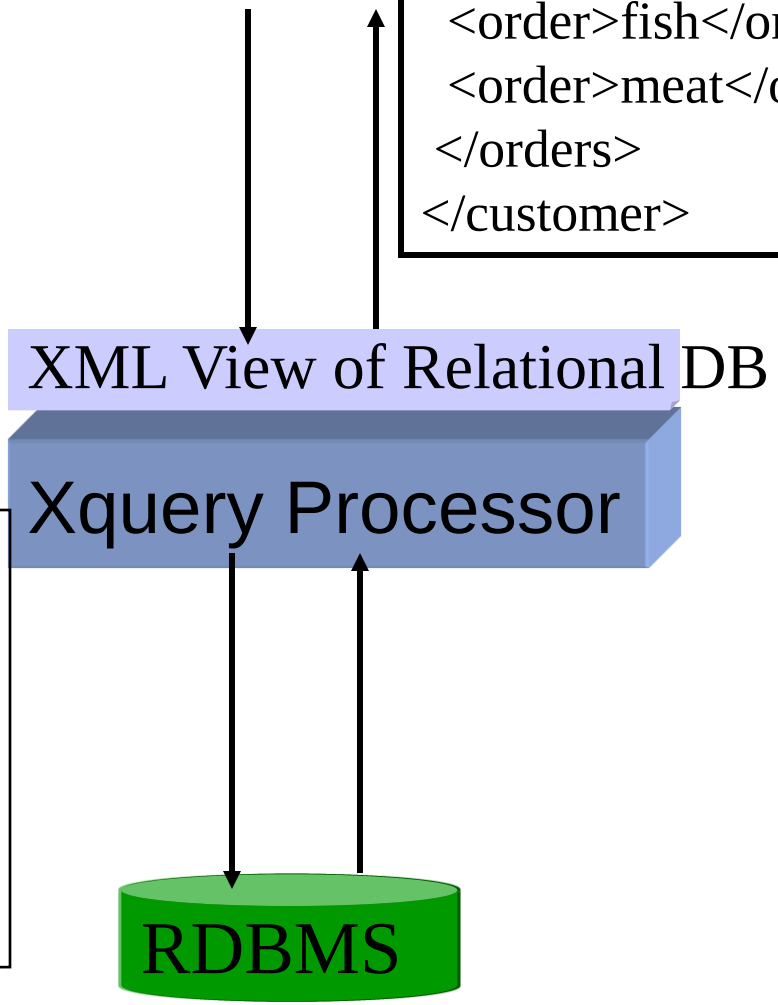
XQuery on Relational Databases

- Single language for accessing database and structuring XML result
- Avoids deficiencies of SQL in dealing with nested structures, optional elements, etc.

XQuery on Relational Databases

```
for $c in db(1)/customers/tuple
where $c/name = “Joe”
return
<customer>  $c/name
 <due>  7.8 * $c/balance  </due>
    <orders>
          for $o in db(1)/orders/tuple
           where $c/name = $o/name
          return $o
    </orders>
</customer>
```

```
<customer>
 <name> Joe </name>
 <due> 780M </due>
 <orders>
  <order>fish</order>
  <order>meat</order>
 </orders>
</customer>
```

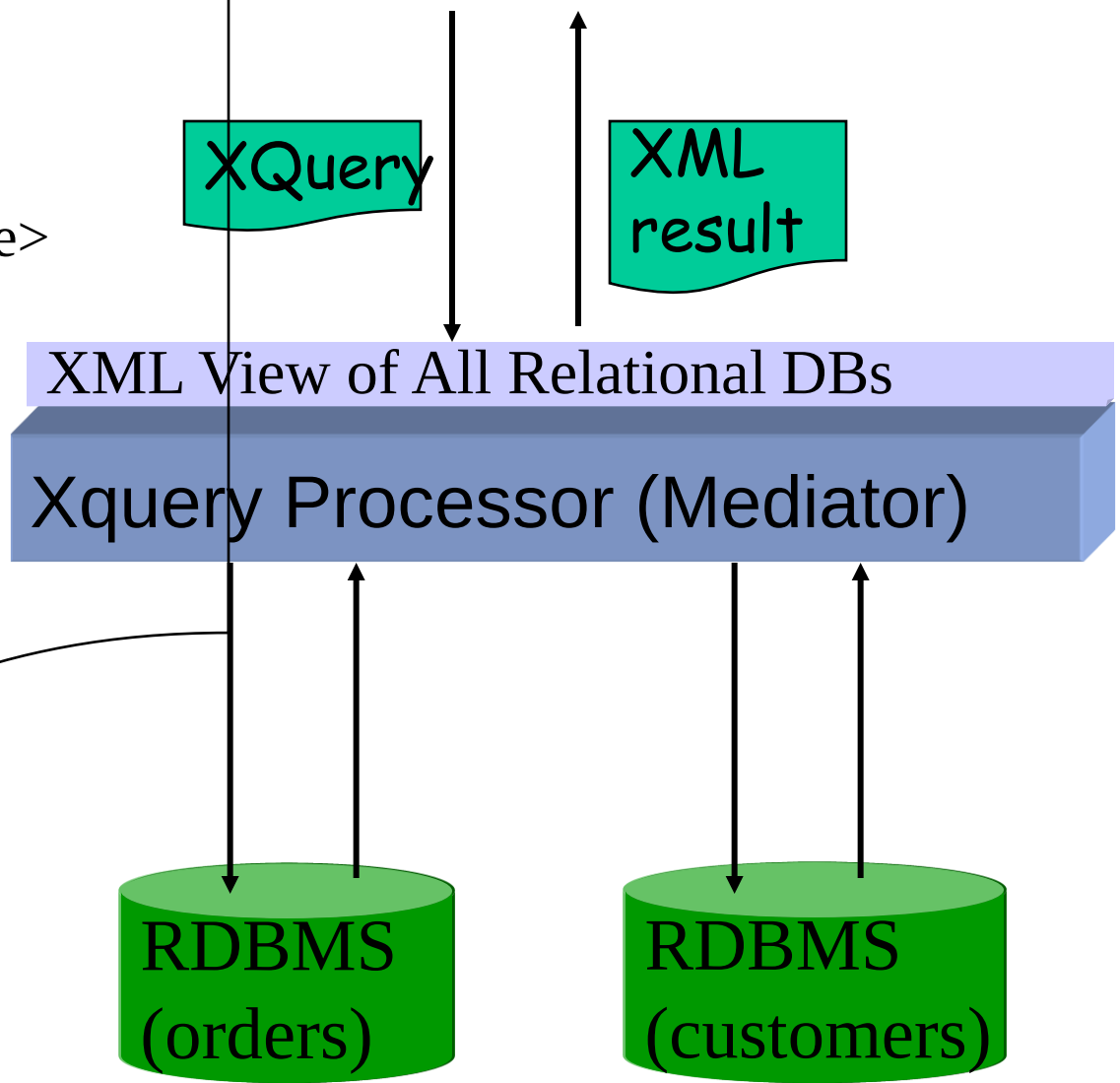



```
SELECT * FROM customers
WHERE name = “Joe”

For each customer #c
 SELECT * FROM orders
 WHERE orders.name = #c.name

Merge results
```

Xquery Processor automatically sends SQL queries to DB and structures XML result

Example: Access to Two Relational Databases

```
FOR $c IN db(1)/customers/tuple
WHERE $c/name = "Joe"
RETURN
<customer>
 $c/name
 <due>   7.8 * $c/balance  </due>
 <orders>
  FOR $o IN db(2)/orders/tuple
  WHERE $c/name = $o/name
  RETURN $o
 </orders>
</customer>
```



data()

- It should be used when you want to **extract the atomic type of a value stored in XML** and defined by a schema.
- Schema that defines this element as xs:dateTime
- *<my-time>2016-01-28T10:30:45.954716-06:00</my-time>*
- Calling data() on <my-time> will return an xs:dateTime type value
-

data()

- The effective boolean value of a node-sequence N is not the same as the effective boolean value of data(N).
- For example, **if (@married)** tests whether the @married **attribute exists,**
- while if (data(@married)) tests whether the **typed value of the @married attribute is true.**
- When constructing element content in XQuery, **nodes are not implicitly atomized,**
- <e>{**@married**}</e> **does something different from** <e>{**data(@married)**}</e>

XQuery Prolog

- The XQuery prolog is a series of declarations and definitions that together create the required environment for query processing.
- Prolog
  - Like XPath, XQuery expressions are evaluated relatively to a context
  - explicitly provided by a prolog (header)

XQuery Ex.: Prolog + Query

```
xquery version "1.0";
declare boundary-space preserve;
declare namespace ord = "http://datypic.com/ord";
declare function local:getProdNums
  ($catalog as element()) as xs:integer*
  {for $prod in $catalog/product
   return xs:integer($prod/number)};

<title>Order Report</title>,
(for $item in doc("ord.xml")//item
 order by $item/@num
 return $item)
```

prolog

query body

XQuery Prolog

- Each declaration or import is followed by a **semicolon**.
- Settings define various parameters for the XQuery processor language, such as:
  - xquery version “1.0”;
  - declare xml:space preserve;
  - declare xml:space strip;

xml:space declaration

- The **xml:space declaration** controls whether, boundary whitespace is preserved during execution of the query.
- If **xml:space preserve is specified**, **boundary whitespace is preserved**.
- If **xml:space strip** is specified or if no xml:space declaration is present, boundary whitespace is stripped (deleted).

XQuery Prolog

- Settings define various parameters for the XQuery processor language, such as:

```
xquery version "1.0";
declare base-uri "http://example.org";
declare default element namespace "http://example.org/names";
declare namespace xs= "http://www.w3.org/2001/XMLSchema";
import module "http://www.w3.org/2003/05/xpath-functions" at
    "logo.xq";

declare variable $x as xs:integer := 7;
declare function addLogo($root as node()) as node();
```

base uri declaration

A **base URI declaration** specifies the **base URI property of the static context**, which is **used when resolving relative URIs within a module resource**. Only one base URI declaration is allowed.

Example

```
declare base-uri "http://example.org";
```

default namespace

```
xquery version "1.0";
declare base-uri "http://example.org";
declare default element namespace "http://example.org/names";
declare namespace xs= "http://www.w3.org/2001/XMLSchema";
```

default namespace declaration

- A **default element namespace declaration** declares a namespace URI that is associated with unprefixed names of elements and types.
- This declaration is recorded as the default element/type namespace in the static context.
- A prolog may contain at most one default element/type namespace declaration.
- **Example:**
  - **declare default element namespace "http://example.org/names";**

Module definition

```
xquery version “1.0”;
module namespace mylib = “http://www.example.com/test_library”;
declare variable $mylib:foo as xs:string := “foo”;
```

module declaration

- A **module declaration** serves to identify a module resource as part of a library module.
- A module declaration consists of the **keyword** module **followed by a namespace prefix** and a string literal which must contain a valid URI.
- The URI identifies the target namespace of the library module, which is the **namespace for all variables and functions imported by the library module.**

import module

```
xquery version "1.0";
declare base-uri "http://example.org";
declare default element namespace "http://example.org/names";
declare namespace xs= "http://www.w3.org/2001/XMLSchema";
import module "http://www.w3.org/2003/05/xpath-functions" at
    "logo.xq";
```

module import

- A **module import** imports the function declarations and variable declarations from one or more library modules into the function signatures and in-scope variables.