XML & Allied Technologies

XQuery

What is Xquery?

- Xquery is a language used to query XML and process and integrate XML data.
- Xquery is often thought of as a native XML ***functional* programming language** .
- W3C Standard:
  - ✓ **http://www.w3.org/TR/xquery/**
- XQuery is a superset of XPath.
- Xquery can easily search any XML structure with path expressions.

What is Xquery?(cont.)

- XQuery for XML is like SQL for databases.
- XQuery is built on XPath expressions.
- XQuery is supported by all major databases.
- XQuery can be used to:
  - Extract information to use in a Web Service
  - Generate summary reports
  - Transform XML data to XHTML
  - Search Web documents for relevant information

What is Xquery?(cont.)

- Xquery creates any XML structure using constructors, and transform XML structures using **FLWOR expressions.**

- Xquery can handle both ordinary XML data (untyped),XML associated with XML schema(typed XML)

- Xquery can be used in aggregation of data.

Why XQuery?

- **Query Languages Versus Programming Languages**
  - ✓ Existing programming languages (C#, Java) allow complex ideas to be expressed in a few lines of code.
  - ✓ Treat XML as any other API, instead of as a first-class part of the language.
  - ✓ Single line of an XML query language like XSLT & XQuery can accomplish the equivalent of hundreds of lines of C, C#, Java, or some other general-purpose languages.

Why XQuery?(cont.)

Why do we need a new Query Language although we can use

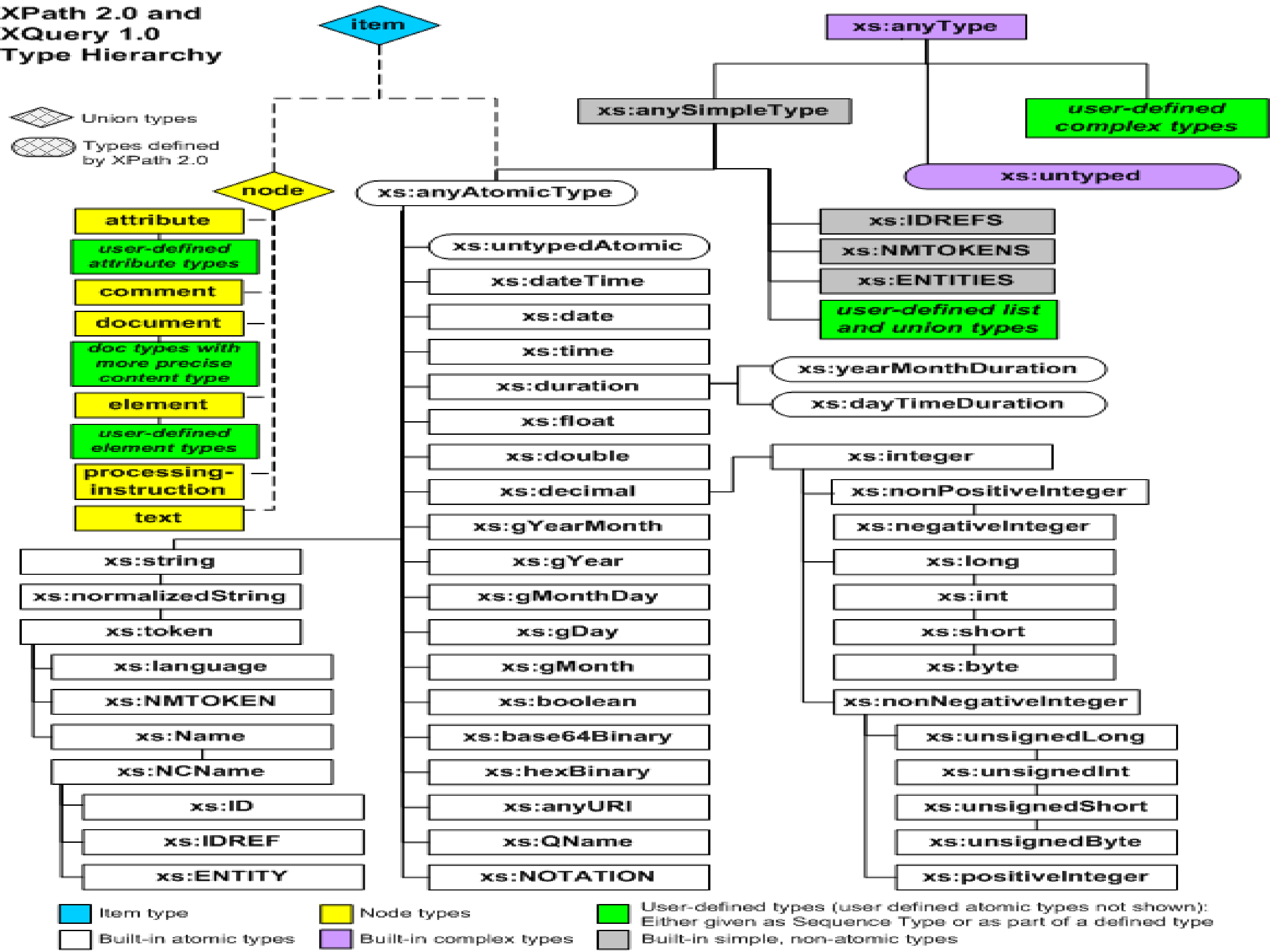
(XPATH) ?!!!!

XPath

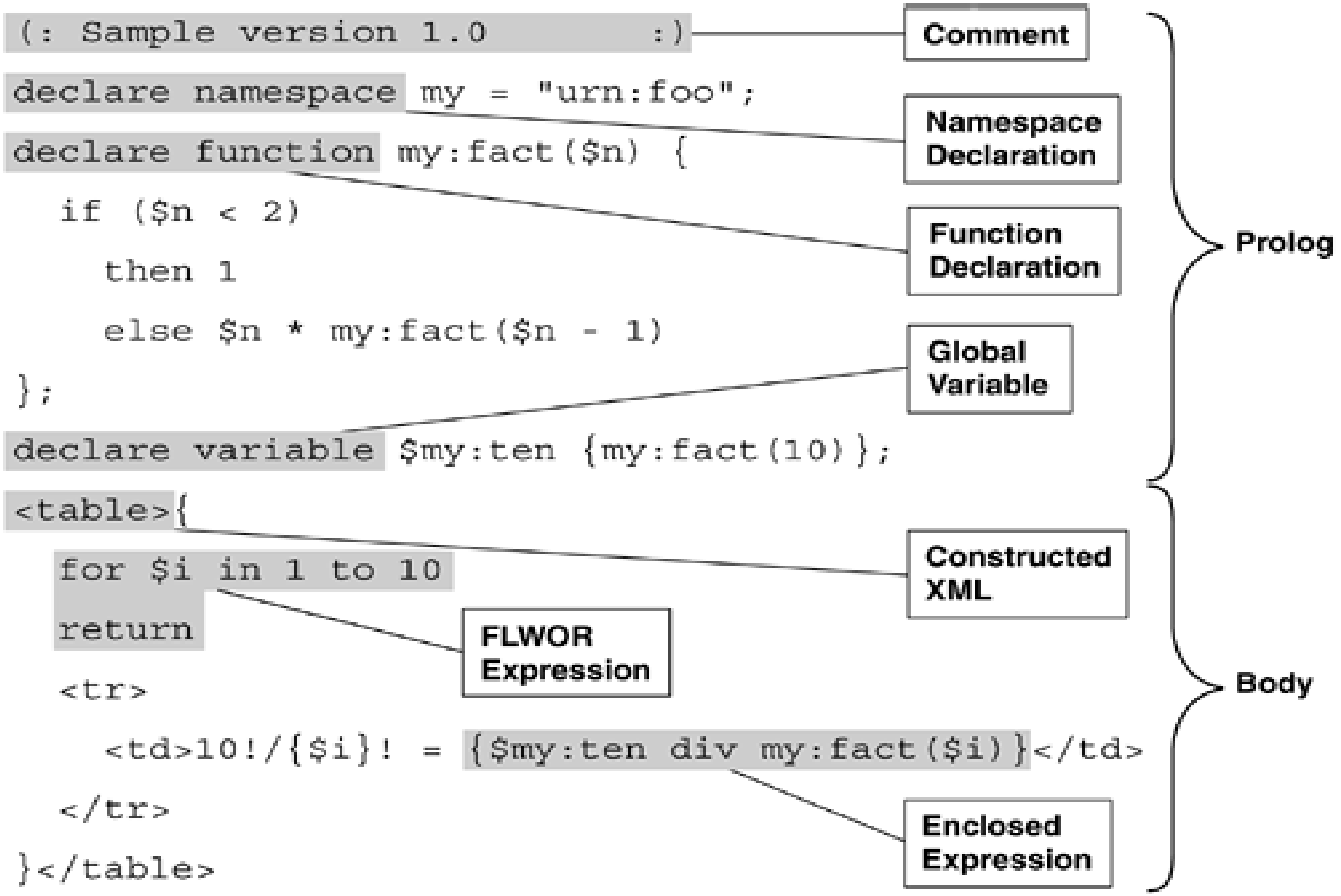
- XPath → addressing parts of an XML document.
- XPath can't create new XML document.
- XPath has a very simple type system (string, boolean, double, and nodeset)

XQuery Approach

- XQuery is especially great at expressing **joins** and **sorts**.
- XQuery can manipulate sequences of values and nodes in arbitrary order.
- XQuery takes a **procedural approach** to query processing making it easy to write **user-defined functions.**


XPath 2.0 and XQuery 1.0 Type Hierarchy
Union types
Types defined by XPath 2.0
item
node
attribute
user-defined attribute types
comment
document
doc types with more precise content type
element
user-defined element types
processing-instruction
text
xs:anyType
xs:anySimpleType
user-defined complex types
xs:untyped
xs:anyAtomicType
xs:IDREFS
xs:NMTOKENS
xs:ENTITIES
user-defined list and union types
xs:untypedAtomic
xs:dateTime
xs:date
xs:time
xs:duration
xs:yearMonthDuration
xs:dayTimeDuration
xs:float
xs:double
xs:decimal
xs:integer
xs:nonPositiveInteger
xs:negativeInteger
xs:long
xs:int
xs:short
xs:byte
xs:nonNegativeInteger
xs:unsignedLong
xs:unsignedInt
xs:unsignedShort
xs:unsignedByte
xs:positiveInteger
xs:gYearMonth
xs:gYear
xs:gMonthDay
xs:gDay
xs:gMonth
xs:boolean
xs:base64Binary
xs:hexBinary
xs:anyURI
xs:QName
xs:NOTATION
xs:string
xs:normalizedString
xs:token
xs:language
xs:NMTOKEN
xs:Name
xs:NCName
xs:ID
xs:IDREF
xs:ENTITY
Item type
Built-in atomic types
Node types
Built-in complex types
User-defined types (user defined atomic types not shown): Either given as Sequence Type or as part of a defined type
Built-in simple, non-atomic types

XQuery Sample



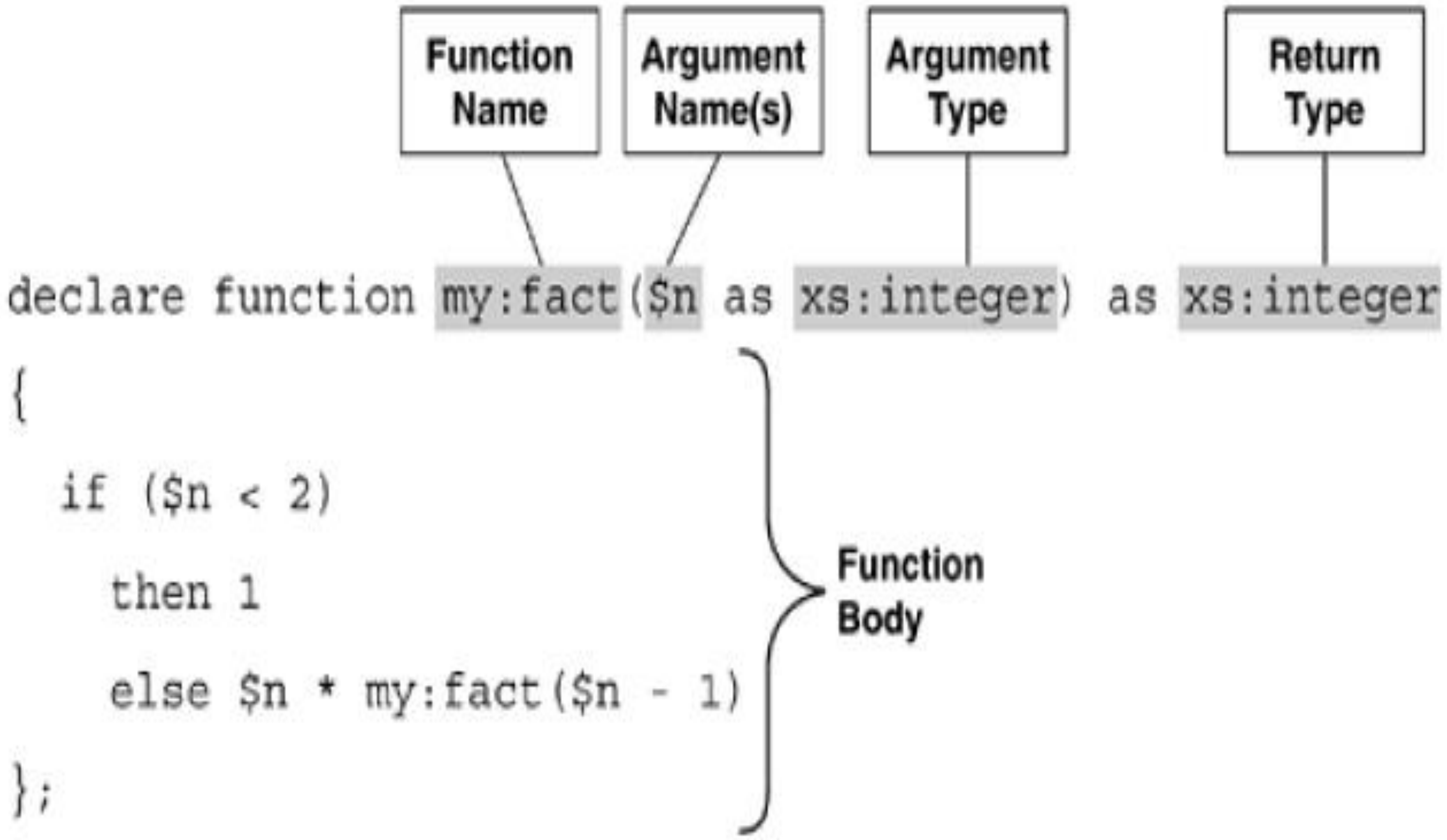
```
(: Sample version 1.0 :)
declare namespace my = "urn:foo";
declare function my:fact($n) {
  if ($n < 2)
    then 1
    else $n * my:fact($n - 1)
};
declare variable $my:ten {my:fact(10)};
<table>{
  for $i in 1 to 10
  return
  <tr>
    <td>10!/{$i}! = {$my:ten div my:fact($i)}</td>
  </tr>
}</table>
```

XQuery Prolog

- The prolog sets up the compile-time context for the rest of the query.

- **It includes things like :**
  - ✓ Default namespace.
  - ✓ User-defined functions.
  - ✓ External Variables.
  - ✓ Global variables.

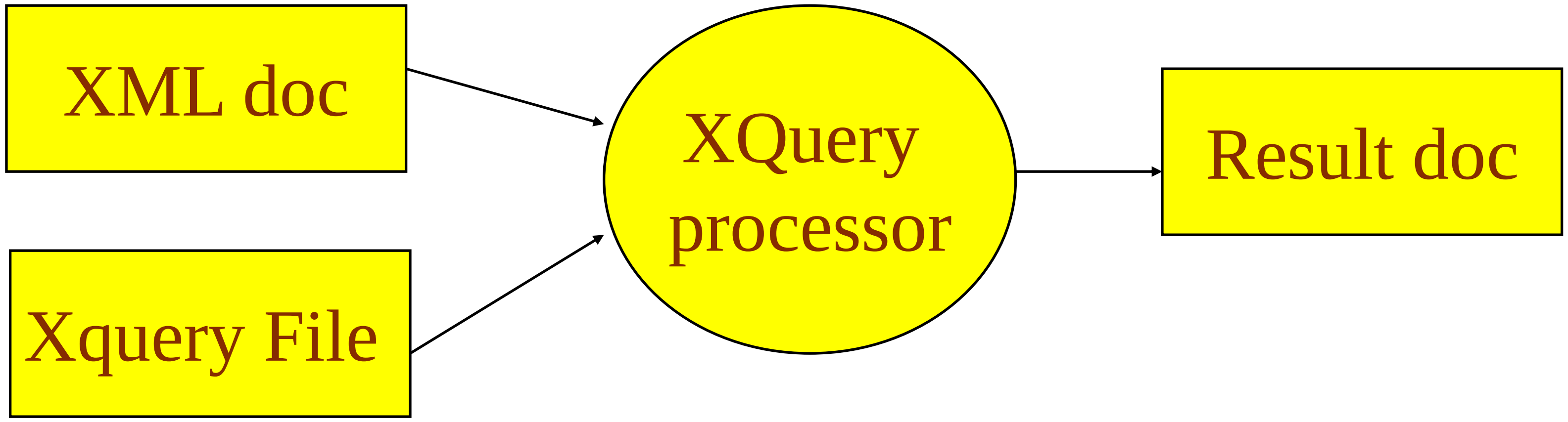
XQuery Functions



```
declare function my:fact($n as xs:integer) as xs:integer
{
  if ($n < 2)
    then 1
    else $n * my:fact($n - 1)
};
```

XQuery

- To execute or Run any of Xquery files ,the editor you are using should have Xquery processor.



Accessing Xml Docs with XQuery

- The XQuery language is designed so that every valid XPath expression is also a valid Xquery query

Ex :

```
//actors

//actors/actor[ends-with(., 'Lisa')]
```

Accessing Xml Docs with Xquery (cont.)

```
//video[actorRef=//actors/actor
                    [ends-with(., 'Lisa')] /@id]/title
```

XQuery FLWOR Expressions

- **FLWOR expression** is equivalent of SQL's SELECT statement

- **It is named after its five clauses:**

  for, let, where, order by, return

- Most of these are optional→the only clause that's always present is the XQuery **return** clause as well as **for** or **let**

XQuery FLWOR Expressions(cont.)

- **Let clause** →simply declares a variable
  - ✓ Variables in Xquery → are written using a dollar sign symbol in front of a name, like so: $variable
- **For clause** → perform looping
- **Where clause** →selects those pairs that we are actually interested in
- **Return clause** → tells the system what information we want to get back

XQuery FLWOR Expressions(cont.)

```
let $doc := .
for $v in $doc//video, $a in $doc//actors/actor
where ends-with($a, 'Lisa‘) and $v/actorRef =
$a/@id
return $v/title
```

```
//video[actorRef=//actors/actor[ends-with(., 'Lisa')]
/@id]/title
```

O in FLWOR expression

O in FLWOR →you can get the results in sorted *order*

```
let $doc := .
for $v in $doc//video,$a in $doc//actors/actor
where ends-with($a, 'Lisa')
and $v/actorRef = $a/@id
order by $v/year
return $v/title
```

LFWOR expression

- Why it isn't a **LFWOR** expression?!
  - ✓ The **for** and **let** clauses can appear in any order, and you can have any number of each.

Declaring a variable in XQuery

- **Variables** in Xquery→ are written using a dollar sign symbol in front of a name, like so ($variable)

- The variable name may consist of only a local-name like this one, or it may be a qualified name consisting of a prefix and local-name, like **$prefix:localname**

- In this case, it behaves like any other XML qualified name. (The prefix must be bound to a namespace in scope, and it is the namespace value that matters, not the prefix.)
- Declaring an **External variable** →a variable that will be taken as input

Declaring a variable in XQuery

- Examples:

  let $doc := .

  - ✓ Declaring variable doc and initializing it by the current document.

- for $v in $doc//video
  - ✓ Declaring variable $v in for clause

- Global variables declaration

  declare variable $age as xs:integer :=1 ;

- Declare an external variable that will be taken as input from the user

  declare variable $firstName external;

Declaring a variable in XQuery(cont.)

- Variable values may refer to other variables defined before them.

```
declare variable $userName as xs:string external;

declare variable $userDoc{concat($userName, ".xml") };
```

Xquery built-in Functions

- XQuery defines over 100 built-in functions.
- Some of these functions come from Xpath but most are new to Xquery.
- Every built-in function resides in the namespace **http://www.w3.org/2003/11/xpath-functions**, which is bound to the predefined namespace prefix **fn**.

Xquery built-in Functions(cont.)

- Because this is also the default function namespace in XQuery, this prefix is generally omitted from built-in function names.
- For example, the built-in **count( )** function takes one sequence argument and computes its length.

Xquery built-in Functions(cont.)

- ceiling(numeric?) as numeric?
- compare(xs:string?, xs:string?) as xs:integer?
- concat(xs:string?, xs:string?, ...) as xs:string
- count(item*) as xs:integer
- current-date() as date

Generating XML Output with XQuery

```
declare variable $firstName as xs:string external;
<videos>{
let $doc := .
for $v in $doc//video, $a in $doc//actors/actor
where ends-with($a, $firstName) and $v/actorRef = $a/@id
order by $v/year
return
<video year="{$v/year}">
{$v/title}
</video>
}
</videos>
```

Xquery Output (cont.)

```
declare namespace my ="my";
declare function my:fact($n as xs:integer)
{
  if ($n < 2) then 1
  else $n* my:fact($n -1)
 };
declare variable $f :=my:fact(4);
  <table>
  {     for $i in 1 to 4
        return
               <tr> <td>
                      4! /{$i} ! ={  $f div  my:fact ($i)   }
              </td> </tr>
     }   </table>
```

```
<table>
        <tr>
         <td>
             4! /1 =24</td>
        </tr>
        <tr>
        <td>
             4! /2 =12</td>
        </tr>
        <tr>
        <td>
              4! /3 =4</td>
        </tr>
        <tr>
          <td>
             4! /4 =1</td>
        </tr>
</table>
```

Output

Logical operators &Conditional statements

- Cond1 **or** Cond2.
- Cond1 **and** Cond2 .
- **not** (Cond1).
- **if/then/else** conditional statement.
- Chained conditions

```
if ($x = 'a')  then 1
else
    if ($x = 'b') then 2
    else 0
```

Quantifications

- **some** and **every :**

•Like "**mini-FLWORs**" that contain only **for** and **where** clauses.

- Instead of **for**, these use the keywords **some** and **every**

- Instead of returning a sequence of values, they return a single **boolean value**

```
some $emp in doc("team.xml")//Employee satisfies
$emp/@years > 5
```

```
every $emp in doc("temp.xml")//Employee satisfies
$emp/@years > 5
```

Xquery Joins

```
for $i in (1, 2, 3)
      for $j in (3, 4, 5)
 →where $i = $j
return ($i, $j)
```

(1, 3, 1, 4, 1, 5, 2, 3, 2, 4, 2, 5, 3, 3, 3, 4, 3, 5)

→(3, 3)

Joining XML Docs

```
for $proj in doc("projects.xml")/Projects/Project,
    $emp in doc("team.xml")//Employee
    where $proj/@owner = $emp/@id
return $proj/Name
```

```
<Name>Enter the Tuple Space</Name>
<Name>Cryptic Code</Name>
<Name>XQuery Bandit</Name>
<Name>Micropoly</Name>
```

Sorting

```
for $i in (4, 2, 3, 1)
order by $i descending
return $i
```

1 2 3 4 →Ascending(default)

4 3 2 1 →Descending