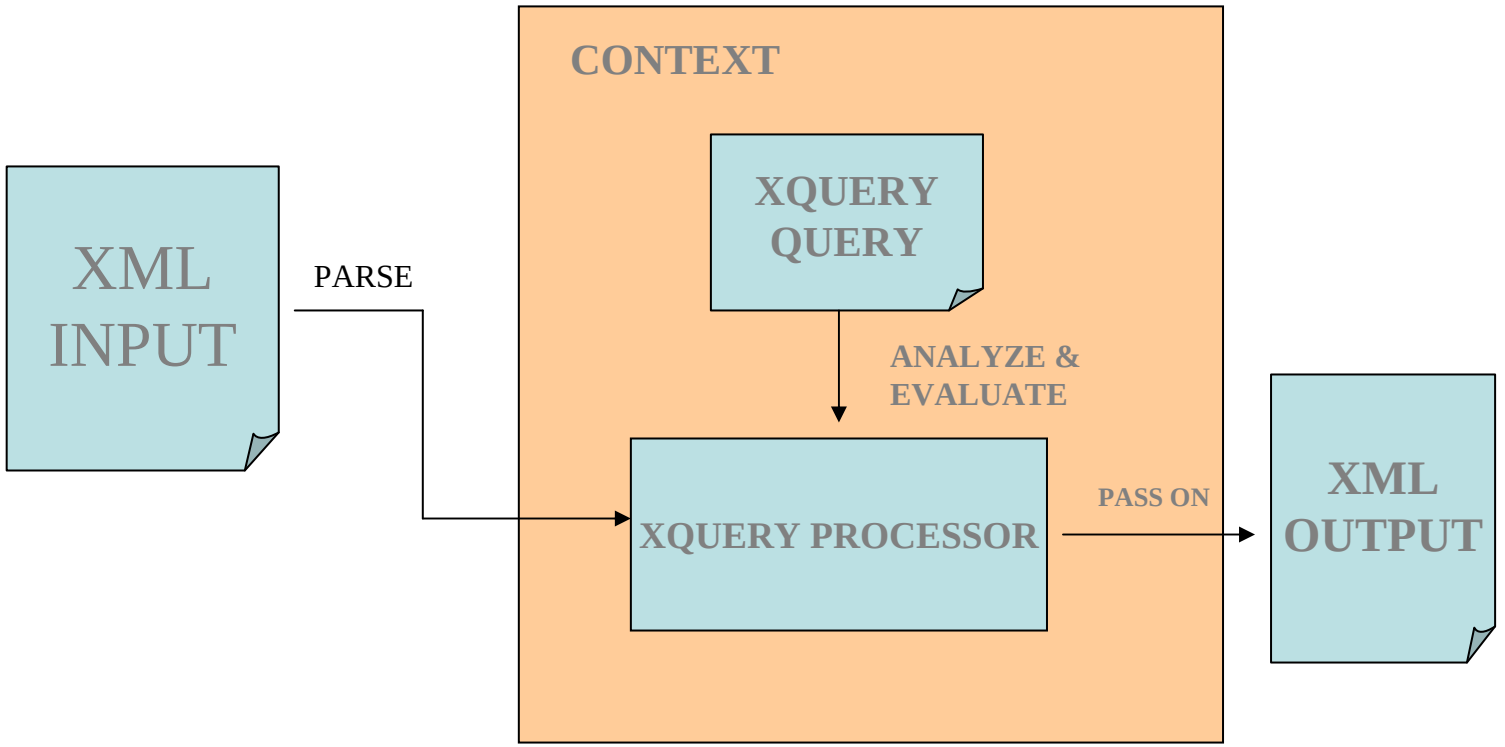
X-QUERY

WHAT IS XQUERY?

- **Select Elements/Attributes From Input Documents**
- **Join Data From Multiple Input Documents**
- **Calculate New Data**
- **Add New Elements/Attributes To The Result**
- **Sort our Results**
- **Xquery For XML Is Like SQL For Databases**

THE XQUERY PROCESSING MODEL



POINTS TO REMEMBER

- XQuery 1.0 queries are not written using XML syntax
- XQuery is a case-sensitive language. All the XQuery keywords are in lower-case.
- XQuery is a functional language comprised of several kinds of expressions that return values. Every query is an expression to be evaluated. These expressions can be nested and composed with full generality..

POINTS TO REMEMBER

- XQuery allows locating nodes (like XPath), constructing and re-structuring XML (like XSLT, but XQuery has a different syntax), sorting, filtering, and so on. One of the most commonly used XQuery expressions is the FLWOR (for-let-where-order by-return) expressions, pronounced "flower expressions".
- XQuery offers the ability to do JOINs across documents.
- XQuery offers a comprehensive set of functions and operators.

XQUERY OVERVIEW

- Xquery is an expression language
  - Every statement evaluates to some result

    Let $x := 5 let $y := 6 return 10*$x+$y

    Evaluates to 56
- Primitive types
  - Number, boolean, strings, dates, times, durations, and XML types
- Derived types
  - Restrictions of other types (e.g., range types)

TYPES OF EXPRESSIONS

- Arithmetic expressions
- Comparison expressions
- Logical expressions
- Path expressions

  - **XQuery path expressions locate nodes, such as element, attribute, and text nodes, in a document.**
  eg:-*doc("books.xml")/bookstore/book/title*

- FLWOR expressions
- Conditional expressions
- Quantified expressions

FLWOR EXPRESSIONS

- **For-Let-Where-Order-Return**
- **pronounced "flower"**
- **generalizes SELECT-FROM-HAVING-WHERE from SQL**
- **The for clause binds a variable to each item returned by the in expression. The for clause results in iteration.**
- **The let clause allows variable assignments and it avoids repeating the same expression many times. The let clause does not result in iteration.**

  eg: let $x := (1 to 5) return <test>{$x}</test>

  ans:

  <test>1 2 3 4 5</test>

- **The where clause is used to specify one or more criteria for the result:**
- **The order by clause is used to specify the sort order of the result**
- **The return clause specifies what is to be returned.**

EXAMPLE FOR FLWOR EXPRESSIONS

- **for $x in doc("books.xml")/bookstore/book/title**
  order by $x return $x

- **for $d in document("depts.xml")//deptno**
  let $e := document("emps.xml")//employee[deptno = $d]
  where count($e) >= 10
  order by avg($e/salary) descending
  return <big-dept> { $d, <headcount>{count($e)}</headcount>, <avgsal>{avg($e/salary)}</avgsal> } </big-dept>

- **for $b in doc("books.xml")//book**
  let $c := $b/author
  return <book>{ $b/title, <count>{ count($c) }</count>}</book>

QUANTIFIED EXPRESSIONS

- XQuery allows **quantified** expressions, which decide properties for all elements in a list:

➢ some-in-satisfies

eg:for $b in doc("books.xml")//book
where some $a in $b/author
satisfies ($a/last="Stevens" and $a/first="W.")
return $b/title

The some quantifier in the where clause tests to see if there is at least one author that satisfies the conditions given inside the parentheses.

➢ every-in-satisfies

A universal quantifier tests whether every node in a sequence satisfies a condition. If a universal quantifier is applied to an empty sequence, it always returns true,because every item in that (empty) sequence satisfies the condition—even though there are no items.The following query tests to see if every author of a book is named W. Stevens:

Eg: for $b in doc("books.xml")//book
where every $a in $b/author
satisfies ($a/last="Stevens" and $a/first="W.")
return $b/title

SELECTION

- Same as SQL select operation
- Selecting tuples satisfying some conditions.

eg:-doc("bib.xml")/bib/book[price<50]

PROJECTION

- Same as SQL project operation
- Selecting some attributes from the document.

eg:-doc("bib.xml")/bib/book/title

JOINS

- Combining two or more input documents into one result set.
- Another use of multiple for clauses
- Eg: <books-with-prices>

```
{
 for $b in doc("bib.xml")//book,
   $a in doc("review.xml")//entry
  where $b/title = $a/title
return
<book-with-prices>
  { $b/title },
     <price-amazon> { $a/price/text() } </price-amazon>
     <price-bn> { $b/price/text() } </price-bn>
     <review>{$a/review/text()}</review>
  </book-with-prices>
   } sortby title
 </books-with-prices>
```

Example(“bib.xml”)

```
<bib>
 <book year="1994">
  <title>TCP/IP Illustrated</title>
  <author><last>Stevens</last><first>W.</first></author>
  <publisher>Addison-Wesley</publisher>
  <price>65.95</price>
  <discount>5</discount>
 </book>

 <book year="1992">
  <title>Advanced Programming in the Unix Environment</title>
  <author><last>Stevens</last><first>W.</first></author>
  <publisher>Addison-Wesley</publisher>
  <price>65.95</price>
    <discount>5</discount>
 </book>

 <book year="2000">
  <title>Data on the Web</title>
  <author><last>Abiteboul</last><first>Serge</first></author>
  <author><last>Buneman</last><first>Peter</first></author>
  <author><last>Suciu</last><first>Dan</first></author>
  <publisher>Morgan Kaufmann Publishers</publisher>
  <price>39.95</price>
    <discount>5</discount>
 </book>
</bib>
```

Example(“review.xml”)

```
<reviews>
 <entry>
   <title>Data on the Web</title>
   <price>34.95</price>
   <review>
     A very good discussion of semi-structured database
     systems and XML.
   </review>
 </entry>
 <entry>
   <title>Advanced Programming in the Unix Environment</title>
   <price>65.95</price>
   <review>
    A clear and detailed discussion of UNIX programming.
   </review>
 </entry>
 <entry>
   <title>TCP/IP Illustrated</title>
   <price>65.95</price>
   <review>
    One of the best books on TCP/IP.
   </review>
 </entry>
</reviews>
```

Output:

- <books-with-prices><book-with-prices><title>TCP/IP Illustrated</title>,
- <price-amazon>65.95</price-amazon><price-bn>65.95</price-bn><review>
- One of the best books on TCP/IP.
- </review></book-with-prices><book-with-prices><title>Advanced Programming in the Unix Environment</title>,
- <price-amazon>65.95</price-amazon><price-bn>65.95</price-bn><review>
- A clear and detailed discussion of UNIX programming.
- </review></book-with-prices><book-with-prices><title>Data on the Web</title>,
- <price-amazon>34.95</price-amazon><price-bn>39.95</price-bn><review>
- A very good discussion of semi-structured database
- systems and XML.
- </review></book-with-prices> sortby title
- </books-with-prices>

BUILT-IN FUNCTIONS

- **XQuery includes over 100 built-in functions. There are functions for string values, numeric values, date and time comparison, node and QName manipulation, sequence manipulation, Boolean values, and more.**
- **Number related:**

  - round(), avg(),sum(),count(),floor(), and ceiling()
- **String related:**

  - concat(), string-length(), starts-with(), ends-with(), substring(),upper-case(), lower-case();
- **The input functions doc() and collection()**
- **Two other frequently used functions are not() and empty().**
- **Date-related:**

  -current-date,month-from-date etc.

USER DEFINED FUNCTIONS

- User-defined functions can be defined in the query or in a separate library.
- **Syntax**

  declare function *prefix:function_name*($*parameter* AS *datatype*) AS *returnDatatype*{(: ...function code here... :) };
- Notes on user-defined functions:
- Use the declare function keyword
- The name of the function must be prefixed
- The data type of the parameters are mostly the same as the data types defined in XML Schema
- The body of the function must be surrounded by curly braces

EXAMPLE-”bib.xml”

```
<?xml version="1.0" encoding="UTF-8"?>
<bib>
 <book year="1994">
   <title>TCP/IP Illustrated</title>
   <author><last>Stevens</last><first>W.</first></author>
   <publisher>Addison-Wesley</publisher>
   <price>65.95</price>
   <discount>5</discount>
 </book>

 <book year="1992">
   <title>Advanced Programming in the Unix Environment</title>
   <author><last>Stevens</last><first>W.</first></author>
   <publisher>Addison-Wesley</publisher>
   <price>65.95</price>
     <discount>5</discount>
 </book>
</bib>
```

EXAMPLE

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

```
for $book in doc("bib.xml")//book
return

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

Output:

- <minPrice>62.6525</minPrice><minPrice>62.6525</minPrice>

ADDING ELEMENTS AND ATTRIBUTES TO THE RESULT

- We can add some HTML elements or other elements and text to the result.

Eg:<html>
<body>
<h1>Bookstore</h1>
<ul>
{
for $x in doc("bib.xml")/bib/book
order by $x/title
**return {data($x/title)}. Year: {data($x/@year)}
**
}
**
**
</body>
</html>

Output:

- **<html><body><h1>Bookstore</h1>
Advanced Programming in the Unix Environment . Year: 1992
Data on the Web . Year: 2000
TCP/IP Illustrated . Year: 1994
The Economics of Technology and Content for Digital TV . Year: 1999
</body></html>**

Example:(adding attributes)

- **<html>**
- **<body>**
- **<h1>Bookstore</h1>**
- **<ul>**
- **{**
- **for $x in doc("bib.xml")/bib/book**
- **order by $x/title**
- **return <li class="{data($x/@year)}">{data($x/title)}
**
- **}**
- **
**
- **</body>**
- **</html>**

Output:

- <html><body><h1>Bookstore</h1>
<li class="1992">Advanced Programming in the Unix Environment
<li class="2000">Data on the Web
<li class="1994">TCP/IP Illustrated
<li class="1999">The Economics of Technology and Content for Digital TV
</body></html>

References:

- www.w3schools.com
- www.ipedo.com/xquery
- http://www.w3.org/TR/xquery/
- http://www.stylusstudio.com/xml_download.html