XQuery

Bob Brown

bob@transentia.com.au

transentia

http://www.transentia.com.au

Thought for XQuery…

mao tse tung

"Let a hundred flowers bloom…"

"Let a hundred FLWRs bloom…"

Overview

- **From the W3C…**

  "The mission…is to provide flexible query facilities to extract data from real and virtual documents on the World Wide Web, therefore finally providing the needed interaction between the Web world and the database world. Ultimately, collections of XML files will be accessed like databases. … to develop the first world standard for querying web documents…"

- **In Plain English…**

  XQuery is a query language specification under development by the World Wide Web Consortium (W3C) that's designed to query collections of XML data—not just XML files, but anything that can appear as XML, including relational databases.

Positioning Quotes

"[X]Query may cause one of the biggest changes to server-side programming since Java servlets."
— Jason Hunter

"Like it or not, the SQL standard is in its twilight years, with XQuery poised to overtake it in terms of major new applications by 2010."
— Jim Melton

'XQuery brings "SQL-like" querying power to applications that require access, selection, integration and transformation of information from one or more XML sources.'
— BEA

Importance for the Enterprise

- **Today**
  - **Data stored in many formats; accessed by many protocols; manipulated by many programming environments; presented via many technologies**
- **Tomorrow**
  - **Data stored in XML-format; accessed via WebServices (XML); manipulated via XQuery; presented via XSLT (XML)**

“It’s the *Data*, Dummies!”

- **(…don’t worry about J2EE/.Net; COBOL; Ada; …make sure that you can operate with/on XML and everything will turn out smelling like a bunch of flowers!)**

XQuery

- **Started in 1998**
- **Now nearly a full W3C recommendation**
  - **In “Last Call” stages**
    - **Most recent: 23 July 2004**
- **Well supported by developers**
  - **Commercial and Open Source**
  - **Lots of toolkits around**

XQuery/2

"So what have we been doing for the last 4 years?

1. *Real innovation...we've had to keep both traditional document mungers and traditional database people happy, since XML completely blurs the old distinctions between documents and data...*
2. *Syntax issues. [The syntax] is intuitive for users, but has required a lot of work on the grammar side.*
3. *Compatibility with existing complex standards....our cooperation with other Working Groups has cost very significant time....*
4. *Inefficiencies and trying to invent our own process.*

It's easier to work quickly when any of the following are true:

- *Nobody cares about what you are doing*
- *There's not a lot of prior work*
- *There's a lot of prior art that solves just exactly the problem you want to solve*
- *There's a homogeneous user community, so you can attend to just their needs*
- *You don't have to prove interoperable implementations*
- *You're allowed to just ignore public comments you don't agree with*
- *All implementations operate in the same environment, and look fairly similar*
- *It's a small, simple problem*

Jonathan Robie, on xml-dev list

As it happens, none of this is true for XQuery."

Important Question

- **How does XQuery compare to SQL?**
  - XQuery is to XML what SQL has been to relational data…with significant advantages
    - SQL is a language for retrieving from relational sources
    - XQuery provides retrieval from XML sources, as well as advanced processing capabilities
      - integrate, manipulate, transform, and filter
  - XQuery and XML allows a developer to morph a wide range of input data into a specific application-friendly form
  - The nested and flexible structure of XML may meet processing requirements better than the flat relational structures returned by SQL

Important Question/2

- **How does XQuery compare to XPath and XSLT?**
  - Before XQuery, developers used the XPath query language to locate items in an XML document and the Extensible Stylesheet Language Transformation engine (XSLT) to transform XML information
  - XPath and XSLT work primarily on XML files
  - XQuery addresses all the data needs of a Web-based application: access multiple sources, select information from them, join it, and transform it to meet the application needs
  - XQuery is a declarative language that is amenable to query optimization of the form that SQL query processors employ

Three Facets

- **XQuery is really three languages in one**
  - **The "surface" syntax**
    - **The most visible language that users are most likely to come into contact with**
  - **XQueryX**
    - **An alternative XML-based syntax that replaces the surface language with one that's more tractable to machine processing**
      - **Now on hold, at least unofficially, until a better replacement can be devised**
  - **A formal algebraic language**
    - **Describes the inner workings of an XQuery processor in quite a bit of detail**

The W3C Documents

- **XML Query Requirements**
  - The planning document for the working group. A list of XQuery desiderata.
- **XML Query Use Cases**
  - A number of real-world scenarios and XQuery snippets solving specific problems.
- **XQuery 1.0: An XML Query Language**
  - The central document, introducing the language itself and overviews everything else.
- **XQuery 1.0 and XPath 2.0 Data Model**
  - Describes the data items a query implementation must understand.
- **XQuery 1.0 and XPath 2.0 Formal Semantics**
  - The underlying algebra formally defining the language.
- **XML Syntax for XQuery 1.0 (XQueryX)**
  - An alternative syntax for those who prefer XML. Preferred by machines everywhere.
- **XQuery 1.0 and XPath 2.0 Functions and Operators Version 1.0**
  - Basic functions and operators on Schema datatypes and XQuery nodes and node sequences.
- **XML Path Language (XPath) 2.0**
  - The XPath documentation, broken out separately.
- **XPath Requirements Version 2.0**
  - The requirements document for XPath.
- **XSLT 2.0 and XQuery 1.0 Serialization**
  - A first look at the considerations involved in outputting serialized "angle-bracket" XML from the XQuery 1.0 and XPath 2.0 Data Model.
- **XML Query and XPath Full-Text Requirements**
  - A description of feature requests that a Full-Text Recommendation needs to be able to comply with.
- **XML Query and XPath Full-Text Use Cases**
  - Real-world scenarios that a Full-Text specification is expected to be able to handle.

"A nascent publishing empire…"

Syntax Overview

- **Simple Syntax**
  - Like SQL
  - Hence should become as popular
- **Major concepts**
  - FLWR Expressions
  - Data Types and Operators
  - Functions

FLWR Expressions

- **FOR**
  - Iterates over a sequence of elements ('tuples')
    - Typically selected via XPath expression
- **LET**
  - Establishes a pointer to a given position/sequence
- **WHERE**
  - Establishes a predicate that selects items
- **RETURN**
  - Copy/gathers selected elements for further processing
  - "RETURN is not Return"
    - ***"The return clause of a FLWOR expression is evaluated once for each tuple in the tuple stream, and the results of these evaluations are concatenated to form the result of the FLWOR expression."***

"Realize that FLWR expressions are just as powerful as SQL SELECT queries. FLWRs are capable of joins, subqueries, and set-based operations, just like SELECT queries."

FLWR Example: XML Data

```
<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>
    </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>
    </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>
    </book>

    <book year="1999">
        <title>The Economics of Technology and Content for Digital TV</title>
        <editor>
            <last>Gerbarg</last><first>Darcy</first>
            <affiliation>CITI</affiliation>
        </editor>
            <publisher>Kluwer Academic Publishers</publisher>
        <price>129.95</price>
    </book>
</bib>
```

FLWR Example: XQuery/Result

```
<result>
{
    for $book1 in doc("http://bstore1.example.com/bib.xml")//book,
        $book2 in doc("http://bstore1.example.com/bib.xml")//book
    let $aut1 := for $a in $book1/author
                 order by $a/last, $a/first
                 return $a
    let $aut2 := for $a in $book2/author
                 order by $a/last, $a/first
                 return $a
    where $book1 << $book2
          and not($book1/title = $book2/title)
          and deep-equal($aut1, $aut2)
    return
        <book-pair>
            { $book1/title }
            { $book2/title }
        </book-pair>
}
</result>
```

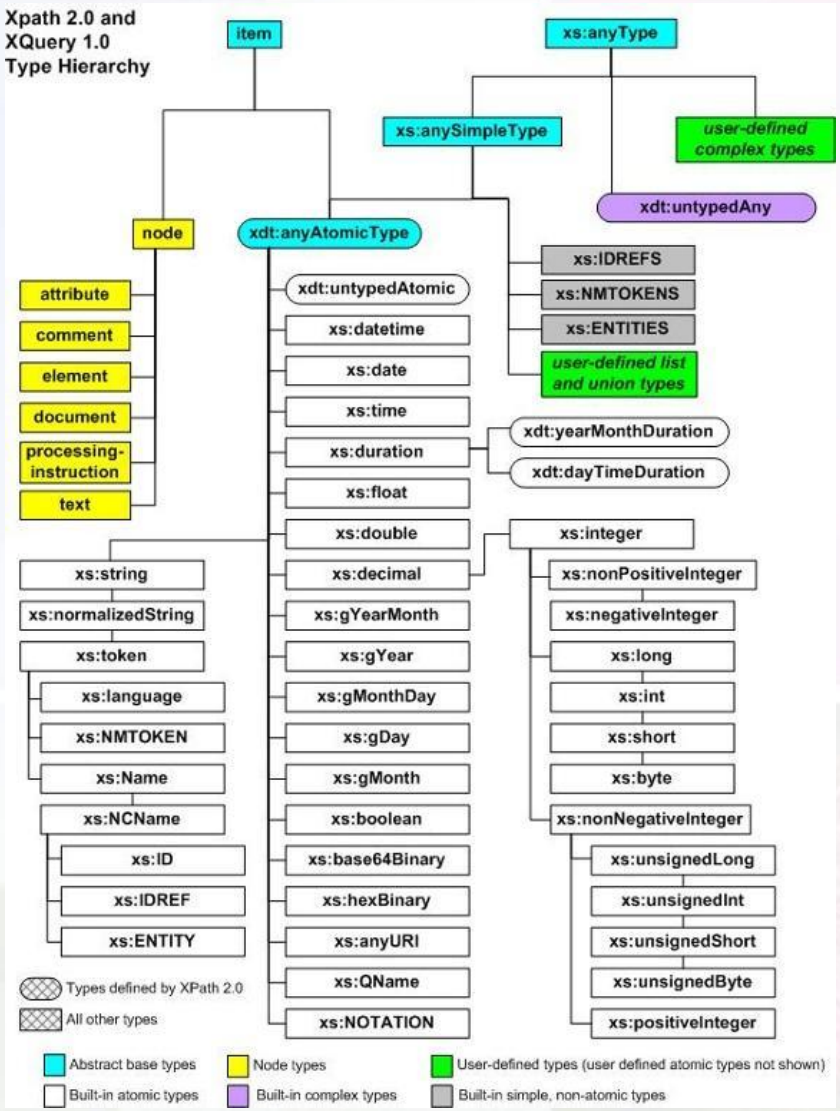
A FLWR Expression

Find pairs of books that have different titles but the same set of authors

```
<result>
    <book-pair>
        <title>TCP/IP Illustrated</title>
        <title>Advanced Programming in the Unix environment</title>
    </book-pair>
</result>
```

Data Types/Operators

- **Based around XPath…**
  - XQuery has driven the development of XPath 2.0, just as XSLT drove development of XPath 1.0
- **Groks**
  - Schemas, DTDs and Namespaces



Functions

- Constructor Functions
- Standard Functions
- Sequence Functions
- External

```
xs:string(12345)

import schema namespace bib="urn:examples:xmp:bib"
bib:isbn("012345678Y")

subtract-dayTimeDuration-from-time(xs:time("11:12:00"), xdt:dayTimeDuration("P3DT1H15M"))
  returns a normalized xs:time value corresponding to the lexical representation "09:57:00"

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

define function outtie($v as xs:integer) as xs:integer external

index-of (("a", "sport", "and", "a", "pastime"), "a")
  returns (1, 4)
```

The W3C Use Cases

- **"XMP": Experiences and Exemplars**
  - Several example queries that illustrate requirements gathered from the database and document communities
- **"TREE": Queries that preserve hierarchy**
  - XML document-types have a very flexible structure in which text is mixed with elements and many elements are optional; show a wide variation in structure; the ways in which elements are ordered and nested are usually quite important
- **"SEQ": Queries based on Sequence**
  - Illustrates queries based on the sequence in which elements appear in a document
- **"R": Access to Relational Data**
  - Treats a database table as an XML document; each tuple inside the table is represented by a nested element. Inside the tuple-elements, each column is in turn represented by an additional nested element
- **"STRING": String Search**
- **"STRONG": Queries that exploit strongly typed data**
  - Explores XQuery's support for types, using data that is governed by a strongly typed XML Schema

There are other cases…

“XMP”

```
<prices>
    <book>
        <title>Advanced Programming in the Unix environment</title>
        <source>bstore2.example.com</source>
        <price>65.95</price>
    </book>
    <book>
        <title>Advanced Programming in the Unix environment</title>
        <source>bstore1.example.com</source>
        <price>65.95</price>
    </book>
    <book>
        <title>TCP/IP Illustrated</title>
        <source>bstore2.example.com</source>
        <price>65.95</price>
    </book>
    <book>
        <title>TCP/IP Illustrated</title>
        <source>bstore1.example.com</source>
        <price>65.95</price>
    </book>
    <book>
        <title>Data on the Web</title>
        <source>bstore2.example.com</source>
        <price>34.95</price>
    </book>
    <book>
        <title>Data on the Web</title>
        <source>bstore1.example.com</source>
        <price>39.95</price>
    </book>
</prices>
```

“XMP/1”

```
<results>
  {
    for $b in doc("http://bstore1.example.com/bib.xml")/bib/book
    return
        <result>
            { $b/title }
            { $b/author  }
        </result>
  }
</results>
```

Create a flat list of all the title-author pairs, with each pair enclosed in a "result" element.

```
<results>
    <result>
        <title>TCP/IP Illustrated</title>
        <author>
            <last>Stevens</last>
            <first>W.</first>
        </author>
    </result>
    …
    <result>
        <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>
    </result>
</results>
```

“XMP/2”

```
<results>
  {
    let $a := doc("http://bstore1.example.com/bib/bib.xml")//author
    for $last in distinct-values($a/last),
        $first in distinct-values($a[last=$last]/first)
    order by $last, $first
    return
        <result>
            <author>
               <last>{ $last }</last>
               <first>{ $first }</first>
            </author>
            {
                for $b in doc("http://bstore1.example.com/bib.xml")/bib/book
                where some $ba in $b/author
                      satisfies ($ba/last = $last and $ba/first=$first)
                return $b/title
            }
        </result>
  }
</results>
```

For each author in the bibliography, list the author's name and the titles of all books by that author, grouped inside a "result" element.

```
<results>
    <result>
        <author>
            <last>Abiteboul</last>
            <first>Serge</first>
        </author>
        <title>Data on the Web</title>
    </result>
    …
    <result>
        <author>
            <last>Suciu</last>
            <first>Dan</first>
        </author>
        <title>Data on the Web</title>
    </result>
</results>
```

“TREE”

```
<?xml version="1.0"?>
<!DOCTYPE book SYSTEM "book.dtd">
<book>
  <title>Data on the Web</title>
  <author>Serge Abiteboul</author>
  <author>Peter Buneman</author>
  <author>Dan Suciu</author>
  <section id="intro"
           difficulty="easy" >
    <title>Introduction</title>
    <p>Text ... </p>
    <section>
      <title>Audience</title>
      <p>Text ... </p>
    </section>
    <section>
      <title>Web Data and the Two Cultures</title>
      <p>Text ... </p>
      <figure height="400" width="400">
        <title>Traditional client/server architecture</title>
        <image source="csarch.gif"/>
      </figure>
      <p>Text ... </p>
    </section>
  </section>
  <section id="syntax" difficulty="medium" >
    …
  </section>
  </section>
</book>
```

```
<!ELEMENT book (title, author+, section+)>
 <!ELEMENT title (#PCDATA)>
 <!ELEMENT author (#PCDATA)>
 <!ELEMENT section (title, (p | figure | section)* )>
 <!ATTLIST section
     id         ID    #IMPLIED
     difficulty CDATA #IMPLIED>
 <!ELEMENT p (#PCDATA)>
 <!ELEMENT figure (title, image)>
 <!ATTLIST figure
      width   CDATA   #REQUIRED
      height  CDATA   #REQUIRED >
 <!ELEMENT image EMPTY>
 <!ATTLIST image
      source  CDATA   #REQUIRED >
```

“TREE/1”

```
declare function local:toc($book-or-section as element()) as element()*
{
    for $section in $book-or-section/section
    return
      <section>
         { $section/@* , $section/title , local:toc($section) }
      </section>
};

<toc>
   {
   for $s in doc("book.xml")/book
       return local:toc($s)
   }
</toc>
```

Prepare a (nested) table of contents, listing all the sections and their titles. Preserve the original attributes of each <section> element, if any

```
<toc>
  <section id="intro" difficulty="easy">
      <title>Introduction</title>
      <section>
          <title>Audience</title>
      </section>
      <section>
          <title>Web Data and the Two Cultures</title>
      </section>
  </section>
  <section id="syntax" difficulty="medium">
      <title>A Syntax For Data</title>
      <section>
          <title>Base Types</title>
      </section>
      <section>
          <title>Representing Relational Databases</title>
      </section>
      <section>
          <title>Representing Object Databases</title>
      </section>
  </section>
</toc>
```

“TREE/2”

```
<top_section_count>
 {
   count(doc("book.xml")/book/section)
 }
</top_section_count>
```

```
<top_section_count>2</top_section_count>
```

How many top-level sections are there?

```
<section_count>{ count(doc("book.xml")//section) }</section_count>,
<figure_count>{ count(doc("book.xml")//figure) }</figure_count>
```

```
<section_count>7</section_count>
<figure_count>3</figure_count>
```

How many sections are in there, and how many figures?

“SEQ”

```
<report>
  <section>
    <section.title>Procedure</section.title>
     <section.content>
      The patient was taken to the operating room where she was placed
      in supine position and
      <anesthesia>induced under general anesthesia.</anesthesia>
      <prep>
        <action>A Foley catheter was placed to decompress the bladder</action>
        and the abdomen was then prepped and draped in sterile fashion.
      </prep>
      <incision>
        A curvilinear incision was made
        <geography>in the midline immediately infraumbilical</geography>
        and the subcutaneous tissue was divided
        <instrument>using electrocautery.</instrument>
      </incision>
      The fascia was identified and
      <action>#2 0 Maxon stay sutures were placed on each side of the midline.</action>
      <incision>
        The fascia was divided using <instrument>electrocautery</instrument>
        and the peritoneum was entered.
      </incision>
      <observation>The small bowel was identified.</observation>
      and <action>
        the <instrument>Hasson trocar</instrument>
        was placed under direct visualization.
      </action>
      <action>
        The <instrument>trocar</instrument>
        was secured to the fascia using the stay sutures.
      </action>
    </section.content>
  </section>
</report>
```

“SEQ/1”

```
let $i2 := (doc("report1.xml")//incision)[2]
for $a in (doc("report1.xml")//action)[. >> $i2][position()<=2]
return $a//instrument
```

What Instruments were used in the first two Actions after the second Incision?

```
<instrument>Hasson trocar</instrument>
<instrument>trocar</instrument>
```

```
for $p in doc("report1.xml")//section[section.title = "Procedure"]
where not(some $a in $p//anesthesia satisfies
        $a << ($p//incision)[1] )
return $p
```

Find "Procedure" sections where no Anesthesia element occurs before the first Incision

(No sections satisfy the query, *thankfully*.)

“R”

```
<items>
  <item_tuple>
    <itemno>1001</itemno>
    <description>Red Bicycle</description>
    <offered_by>U01</offered_by>
    <start_date>1999-01-05</start_date>
    <end_date>1999-01-20</end_date>
    <reserve_price>40</reserve_price>
  </item_tuple>
  <!-- !!! Snip !!! -->

<users>
  <user_tuple>
    <userid>U01</userid>
    <name>Tom Jones</name>
    <rating>B</rating>
  </user_tuple>
  <!-- !!! Snip !!! -->

<bids>
  <bid_tuple>
    <userid>U02</userid>
    <itemno>1001</itemno>
    <bid>35</bid>
    <bid_date>1999-01-07</bid_date>
    </bid_tuple>
  <bid_tuple>
  <!-- !!! Snip !!! -->
```

“R/1”

```
<result>
  {
    for $i in doc("items.xml")//item_tuple
    let $b := doc("bids.xml")//bid_tuple[itemno = $i/itemno]
    where contains($i/description, "Bicycle")
      order by $i/itemno
    return
        <item_tuple>
            { $i/itemno }
            { $i/description }
            <high_bid>{ max($b/bid) }</high_bid>
        </item_tuple>
  }
</result>
```

```
<result>
  <item_tuple>
    <itemno>1001</itemno>
    <description>Red Bicycle</description>
    <high_bid>55.0</high_bid>
  </item_tuple>
  <item_tuple>
    <itemno>1003</itemno>
    <description>Old Bicycle</description>
    <high_bid>20.0</high_bid>
  </item_tuple>
  <item_tuple>
    <itemno>1007</itemno>
    <description>Racing Bicycle</description>
    <high_bid>225</high_bid>
  </item_tuple>
  <item_tuple>
    <itemno>1008</itemno>
    <description>Broken Bicycle</description>
    <high_bid></high_bid>
  </item_tuple>
</result>
```

For all bicycles, list the item number, description, and highest bid (if any), ordered by item number

“R/2”

```
<result>
  {
   unordered (
    for $seller in doc("users.xml")//user_tuple,
        $buyer in doc("users.xml")//user_tuple,
        $item in doc("items.xml")//item_tuple,
        $highbid in  doc("bids.xml")//bid_tuple
    where $seller/name = "Tom Jones"
      and $seller/userid  = $item/offered_by
      and contains($item/description , "Bicycle")
      and $item/itemno  = $highbid/itemno
      and $highbid/userid  = $buyer/userid
      and $highbid/bid =
            max(
                doc("bids.xml")//bid_tuple
                  [itemno = $item/itemno]/bid
                )
    return
        <jones_bike>
            { $item/itemno }
            { $item/description }
            <high_bid>{ $highbid/bid }</high_bid>
            <high_bidder>{ $buyer/name }</high_bidder>
        </jones_bike>
   )
  }
</result>
```

For bicycle(s) offered by Tom Jones that have received a bid, list the item number, description, highest bid, and name of the highest bidder, ordered by item number

```
<result>
    <jones_bike>
        <itemno>1001</itemno>
        <description>Red Bicycle</description>
        <high_bid>
            <bid>55</bid>
        </high_bid>
        <high_bidder><name>Mary Doe</name></high_bidder>
    </jones_bike>
</result>
```

“R/3”

```
declare function local:bid_summary()
  as element()*
{
    for $i in distinct-values(doc("bids.xml")//itemno)
    let $b := doc("bids.xml")//bid_tuple[itemno = $i]
    return
        <bid_count>
            <itemno>{ $i }</itemno>
            <nbids>{ count($b) }</nbids>
        </bid_count>
};

<result>
 {
    let $bid_counts := local:bid_summary(),
        $maxbids := max($bid_counts/nbids),
        $maxitemnos := $bid_counts[nbids = $maxbids]
    for $item in doc("items.xml")//item_tuple,
        $bc in $bid_counts
    where $bc/nbids =  $maxbids and $item/itemno = $bc/itemno
    return
        <popular_item>
            { $item/itemno }
            { $item/description }
            <bid_count>{ $bc/nbids/text() }</bid_count>
        </popular_item>
 }
</result>
```

List the item number and description of the item(s) that received the largest number of bids, and the number of bids it (or they) received

```
<result>
    <popular_item>
        <itemno>1001</itemno>
        <description>Red Bicycle</description>
        <bid_count>5</bid_count>
    </popular_item>
    <popular_item>
        <itemno>1002</itemno>
        <description>Motorcycle</description>
        <bid_count>5</bid_count>
    </popular_item>
</result>
```

“String”

```
declare function local:partners($company as xs:string) as element()*
{
    let $c := doc("company-data.xml")//company[name = $company]
    return $c//partner
};

let $foobar_partners := local:partners("Foobar Corporation")

for $item in doc("string.xml")//news_item
where
  some $t in $item//title satisfies
    (contains($t/text(), "Foobar Corporation")
    and (some $partner in $foobar_partners satisfies
      contains($t/text(), $partner/text())))
  or (some $par in $item//par satisfies
   (contains(string($par), "Foobar Corporation")
     and (some $partner in $foobar_partners satisfies
        contains(string($par), $partner/text()))))
return
    <news_item>
        { $item/title }
        { $item/date }
    </news_item>
```

Find news items where the Foobar Corporation and one or more of its partners are mentioned in the same paragraph and/or title. List each news item by its title and date

```
<news_item>
    <title> Gorilla Corporation acquires YouNameItWeIntegrateIt.com </title>
    <date>1-20-2000</date>
</news_item>
<news_item>
    <title>Foobar Corporation releases its new line of Foo products
today</title>
    <date>1-20-2000</date>
</news_item>
```

“STRONG”

```
import schema namespace ipo="http://www.example.com/IPO";
import schema namespace pst="http://www.example.com/postals";
import schema namespace zips="http://www.example.com/zips";

import module namespace zok="http://www.example.com/xq/zips";
import module namespace pok="http://www.example.com/xq/postals";

declare function local:address-ok($a as element(*, ipo:Address))
 as xs:boolean
{
  typeswitch ($a)
      case $zip as element(*, ipo:USAddress)
           return zok:zip-ok($zip)
      case $postal as element(*, ipo:UKAddress )
           return pok:postal-ok($postal)
      default return false()
};

for $p in doc("ipo.xml")//element(ipo:purchaseOrder)
where not( local:address-ok($p/shipTo) and local:address-ok($p/billTo))
return $p
```

Return purchase orders with an erroneous postal code or zip code (depending on whether it is a UK Address or a US Address)

XQueryX Example

Disliked for its verboseness!

```
FOR $p IN distinct(doc("bib.xml")
                   //publisher)
LET $a := avg(doc("bib.xml")
              //book[publisher = $p]/price)
RETURN
    <publisher>
        <name>{ $p/text() }</name>
        <avgprice>{ $a }</avgprice>
    </publisher>
```

```
<q:query xmlns:q="http://www.w3.org/2001/06/xqueryx">
  <q:flwr>
    <q:forAssignment variable="$p">
      <q:function name="distinct">
        <q:step axis="SLASHSLASH">
          <q:function name="document">
            <q:constant datatype="CHARSTRING">bib.xml</q:constant>
          </q:function>
          <q:identifier>publisher</q:identifier>
        </q:step>
      </q:function>
    </q:forAssignment>
    <q:letAssignment variable="$a">
      <q:function name="avg">
        <q:step axis="CHILD">
          <q:function name="document">
            <q:constant datatype="CHARSTRING">bib.xml</q:constant>
          </q:function>
          <q:step axis="CHILD">
            <q:predicatedExpr>
              <q:identifier>book</q:identifier>
              <q:predicate>
                <q:function name="EQUALS">
                  <q:identifier>publisher</q:identifier>
                  <q:variable>$p</q:variable>
                </q:function>
              </q:predicate>
            </q:predicatedExpr>
            <q:identifier>price</q:identifier>
          </q:step>
        </q:step>
      </q:function>
    </q:letAssignment>
    <q:return>
      <q:elementConstructor>
        <q:tagName>
          <q:identifier>publisher</q:identifier>
        </q:tagName>
        <q:elementConstructor>
          <q:tagName>
            <q:identifier>name</q:identifier>
          </q:tagName>
          <q:step axis="CHILD">
            <q:variable>$p</q:variable>
            <q:nodeKindTest kind="TEXT" />
          </q:step>
        </q:elementConstructor>
        <q:elementConstructor>
          <q:tagName>
            <q:identifier>avgprice</q:identifier>
          </q:tagName>
          <q:variable>$a</q:variable>
        </q:elementConstructor>
      </q:elementConstructor>
    </q:return>
  </q:flwr>
</q:query>
```

What about Update?

- **Full CRUD is not here yet…**
  - **Various proposals being readied for XQuery 2.0**
    - **FLWU (“FLU”?)**

```
update replace input()/bib/book[title = "TCP/IP Illustrated"]/@year
  with attribute year {2003}
```

```
update for $a in input()/bib/book
  let $b := $a/@year
  where ($b >= 1990) and ($b <= 1994)
  do replace $b
     with attribute year {2003}
```

Examples from Tamino

```
update insert
  <author>
     <last>Schneider</last> <first>Helge</first>
  </author>
  following input()/bib/book[title='Data on the Web']/author[1]
```

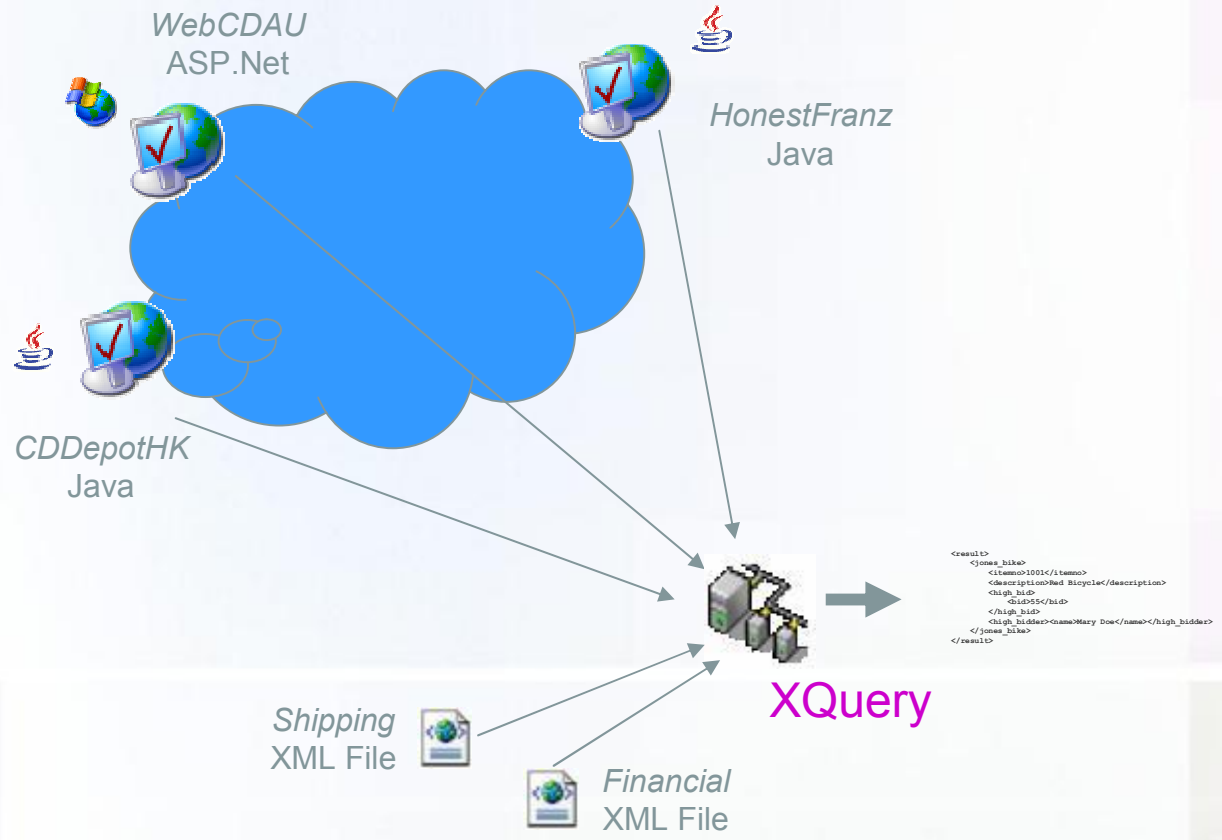
XQJ

- JSR-225, titled "XQuery API for Java (XQJ)."
- A common API for XQuery/Java interaction
  - Proposed by Oracle and IBM
  - Likely to live in the javax.xml.xquery package
- As JDBC is for SQL, so XQJ will be for XQuery
- Some goals:
  - A stylistic similarity with JDBC and Java API for XML Processing (JAXP)
    - Can compile queries for repeated execution
    - Parameterized queries and discovery/binding of input parameters
  - A connection-oriented interface with transactional support
    - (interesting because XQuery 1.0 doesn't have a standard update mechanism)
  - A connectionless interface for single-shot queries
  - Can obtain an XQJ connection from a JDBC connection for engines where that makes sense
  - Provide for easy back-end plugability

Resources

- **Pages**
  - http://www.w3.org/XML/Query
  - http://otn.oracle.com/oramag/oracle/03-may/o33devxml.html?_template=/ocom/technology/content/print
  - http://www.perfectxml.com/XQuery.asp
  - http://www.transentia.com.au
    - This presentation will be available in a day or so
- **Implementations**
  - BEA Liquid Data, http://www.bea.com/framework.jsp?CNT=index.htm&FP=/content/products/liquid_data
  - Bumblebee, http://xquery.com/bumblebee
    - "Bumblebees buzz around FLWRs"
  - Saxon, http://www.saxonica.com/
  - Qizx/open, http://www.xfra.net/qizxopen/
  - Oracle xquery, http://otn.oracle.com
  - Tamino, http://tamino.demozone.softwareag.com/demoXQuery/XQueryDemo/index.jsp;jsessionid=0EF28F7E0761C23140E254771CEAC95D

Demo Time!


WebCDAU
ASP.Net
HonestFranz
Java
CDDepotHK
Java
<result>
<jones_bike>
<itemno>1001</itemno>
<description>Red Bicycle</description>
<high_bid>
<bid>55</bid>
</high_bid>
<high_bidder><name>Mary Doe</name></high_bidder>
</jones_bike>
</result>
XQuery
Shipping
XML File
Financial
XML File