summer school

19th September 2013

XQuery and XML Applications

Adam Retter

adam.retter@googlemail.com / @adamretter

Learning Objectives

The class looks at XQuery, XML Databases and building XML Applications in XQuery and applying these for the Web.

1. Understand the purpose and scope of XQuery
2. Learn the basics of XQuery
3. Test your knowledge
4. Introduction to XML Databases
5. Practice by building a Simple XQuery App
6. Review and Improve the App

summer school

Contents

Lecture (90 Minutes)

1. XQuery Background
2. XQuery Basics
3. Advanced XQuery

* **Break** (30 Minutes) *

Tutorial Session (90 Minutes)

4. XML Databases
5. XML Applications
6. Building an XML Application
7. Web Enabling an XML Application
8. Hands-on. Adding features to the XML Application

summer school

XQuery Background

So what is XQuery?
...and what's it good for?

XQuery is...

- XML Query Language
  - A W3C Standard
  - Superset of XPath 2.0
  - Closely related to XSLT 2.0
  - Is NOT written in XML
- A Query Language!
  - Pull information from one or more XML documents
  - The "SQL of XML"
- A Transformation Language
  - Transform data (XML, HTML, Text, etc.) from one form or structure to another

XQuery is also...

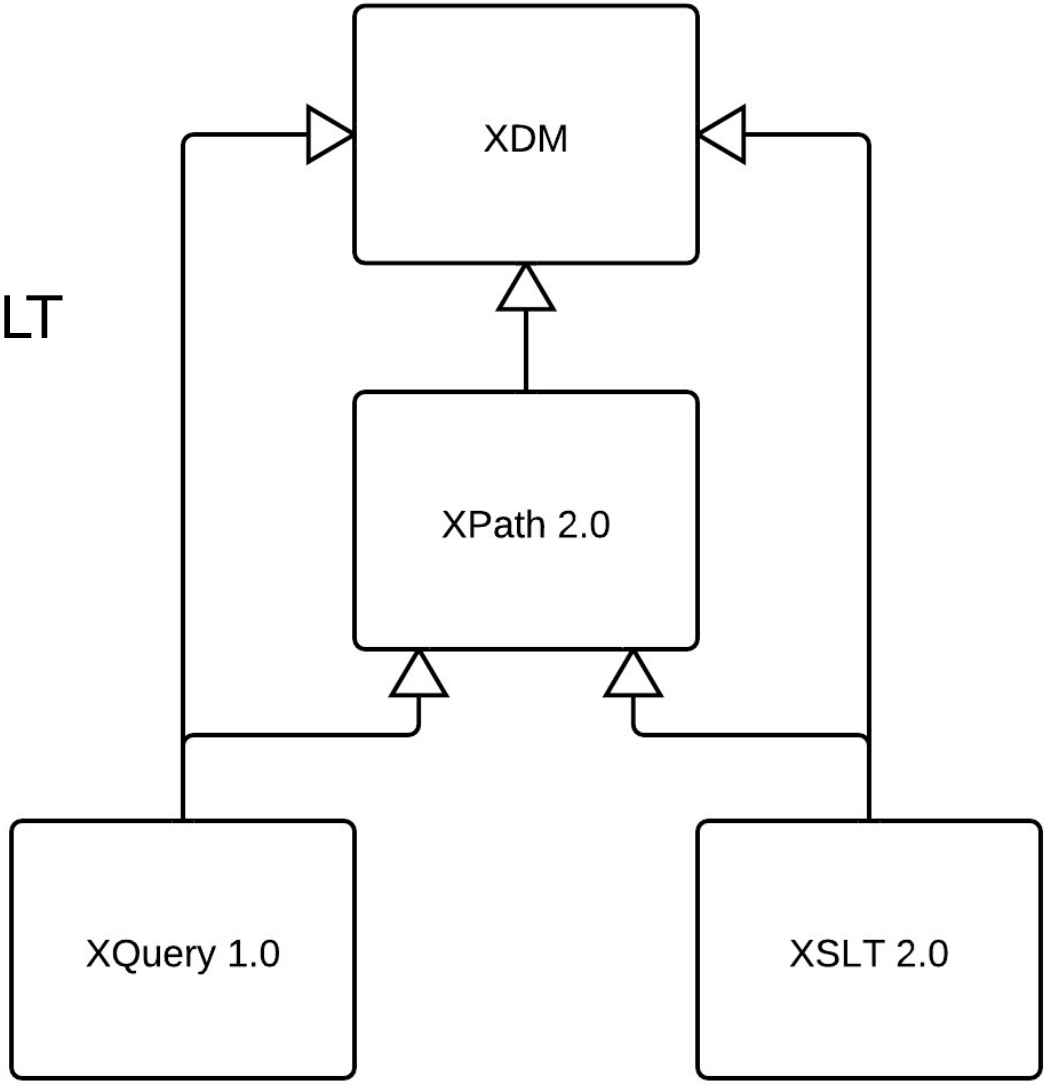
- Not Just Queries
  - Can update XML documents
  - Can create new XML documents
- An Application Programming Language?
  - Turing Complete
  - Functional Programming (esp. 3.0) + Modules
  - XML Data Model Type System (data + code)
  - Suited to the Web
- Easy to learn!

XQuery Design Goals

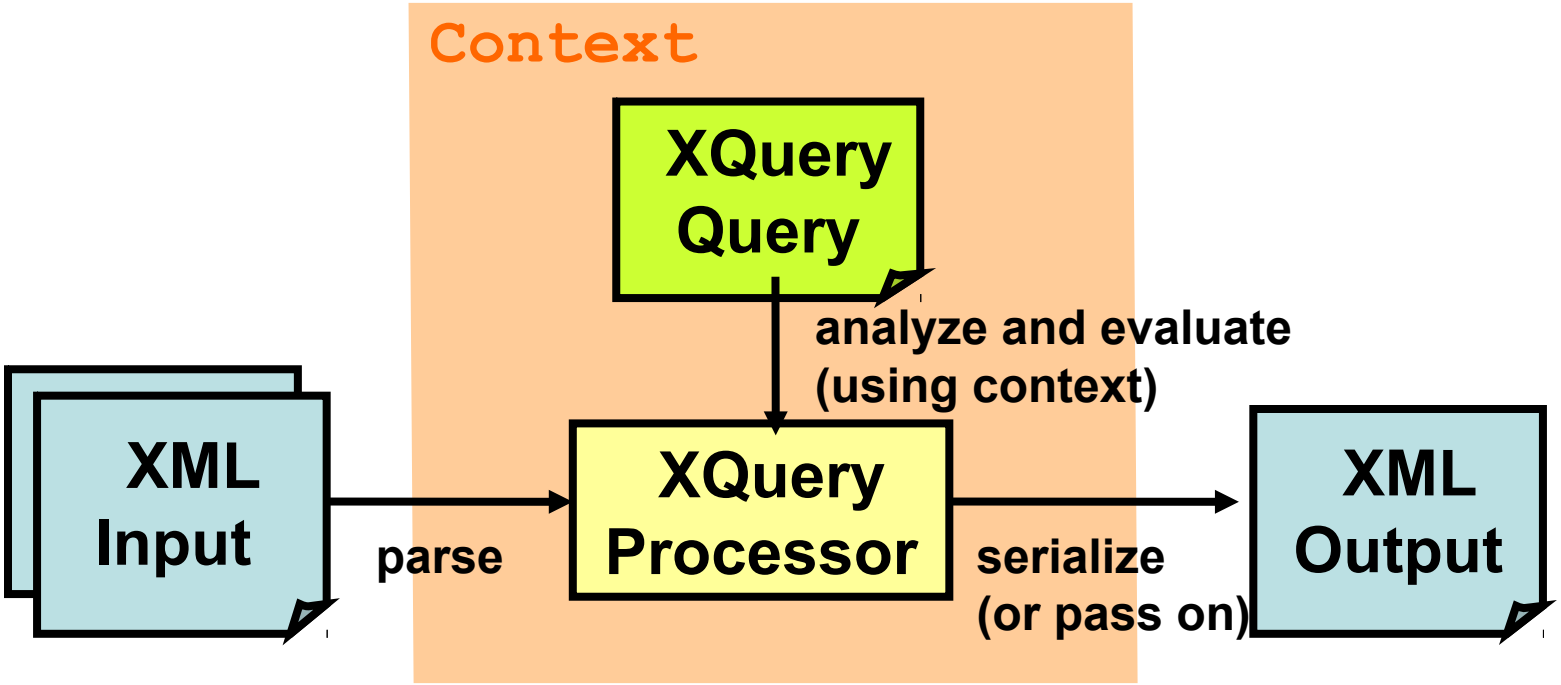
- Queries are concise and easily understood
- Suitable for both structured and unstructured data
- Platform/Protocol agnostic with predictable results
- Declarative rather than Procedural (What vs. How).
  - Strongly typed (optimisation and error detection)
- Able to process collections of documents
- Compatible with other W3C standards
    - XML 1.1, Namespaces, XML Schema, XPath

Where does XQuery fit?

- Its kinda just XPath++
  - If you know XPath...
- Much in common with XSLT
  - XDM and XPath



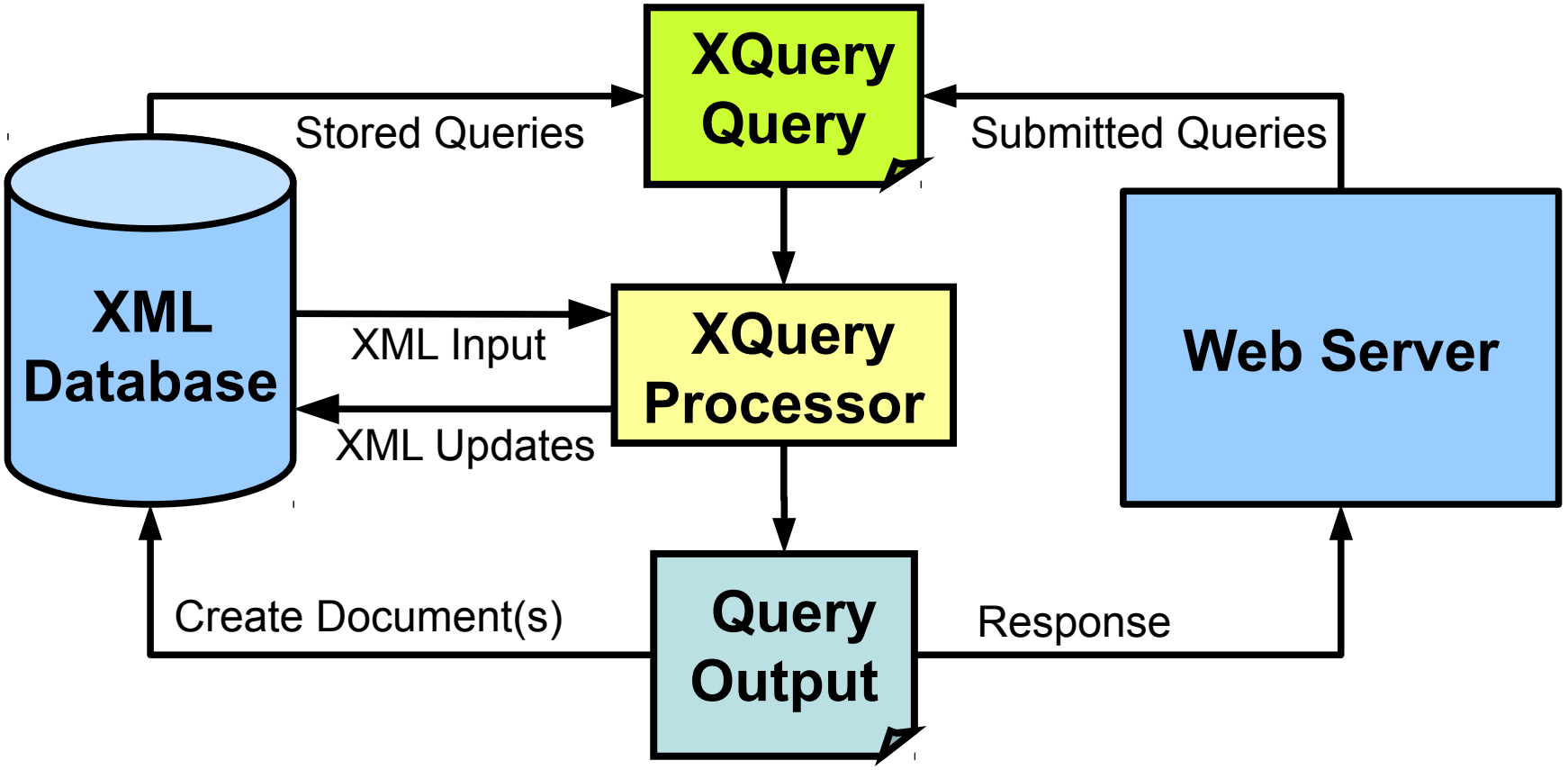
XQuery Processing Model (Simple)



- XQuery typically operates on Document(s) from either:
  - Sources bound to the Processor
  - Pulled in during the query (e.g. doc(), collection())

summer school

XQuery Processing Model (Platform)



Why XQuery?

- Why not just use XSLT?
  - Well you could!
- XSLT is best suited to Transformation
  - Typically: Document → XSLT → Document
- XQuery is best suited to query/search
  - Designed to work well over many documents
  - XSLT does not have Update extensions
  - XSLT does not have Full Text extensions

Use Case #1: Search and Browse

- Searching through documents
  - Usually narrative, semi-structured (mixed content)
  - e.g. Medical Journals, Manuscripts, Web Content
- Multiple document aware
  - Search may need to rank results across documents
  - Content Store (Filesystem, XML Database)?
- Browse
  - Present results to Application/API as XML
  - Present results to user as a Web Page

Use Case #1: Search and Browse

"What medical journal articles since 2004 mention 'artery' and 'plaque' within 3 words of each other?"

- Can be implemented in pure XQuery 1.0*
  - Difficult'ish. No native FT, just string funcs.
  - Not very efficient?
  - Most likely XSLT 2.0

- XQuery and XPath Full Text 1.0
  - An Extension specification to XQuery 1.0
  - Stemming, Thesaurus, Distance, Scoring, Weighting, Occurrence

*see: http://www.adamretter.org.uk/blog/entries/xquery-matching-based-on-word-distance.xml

Use Case #1: Search and Browse

"What medical journal articles since 2004 mention 'artery' and 'plaque' within 3 words of each other?"

- XQuery 1.0 with Full Text extensions Example:

```
/journal[xs:date(@date) ge xs:date("2004-01-01")]
  contains text "artery" ftand "plaque" distance at most 3 words
```

Use Case #1: Search and Browse

- But... Vendor-specific extensions
  - XQuery and XPath Full Text 1.0 is not widely implemented
  - Typically equivalent but proprietary functions are available
  - Also may be available:
    - Functions to extract and search the textual content of non-xml (binary) resources e.g. .doc, PDF etc.

- eXist-db specific XQuery 1.0 Example:

```
/journal[xs:date(@date) ge xs:date("2004-01-01")]
  [ft:query(., '"artery plaque"~3')]
```

Use Case #2: XML in RDBMS

- Sometimes most of your data is highly structured
  - And you just need to add some more flexible data or metadata
- Supported by major relational RDMBS Vendors
  - ISO/IEC SQL/XML Standards
  - SQL Server, Oracle, IBM DB2, Postgres 9
- Features
  - An XML Type for columns
  - Embed XQuery into SQL and pass/project values in/out
  - Possible Schema Validation or Path Indexing

Use Case #2: XML in RDBMS

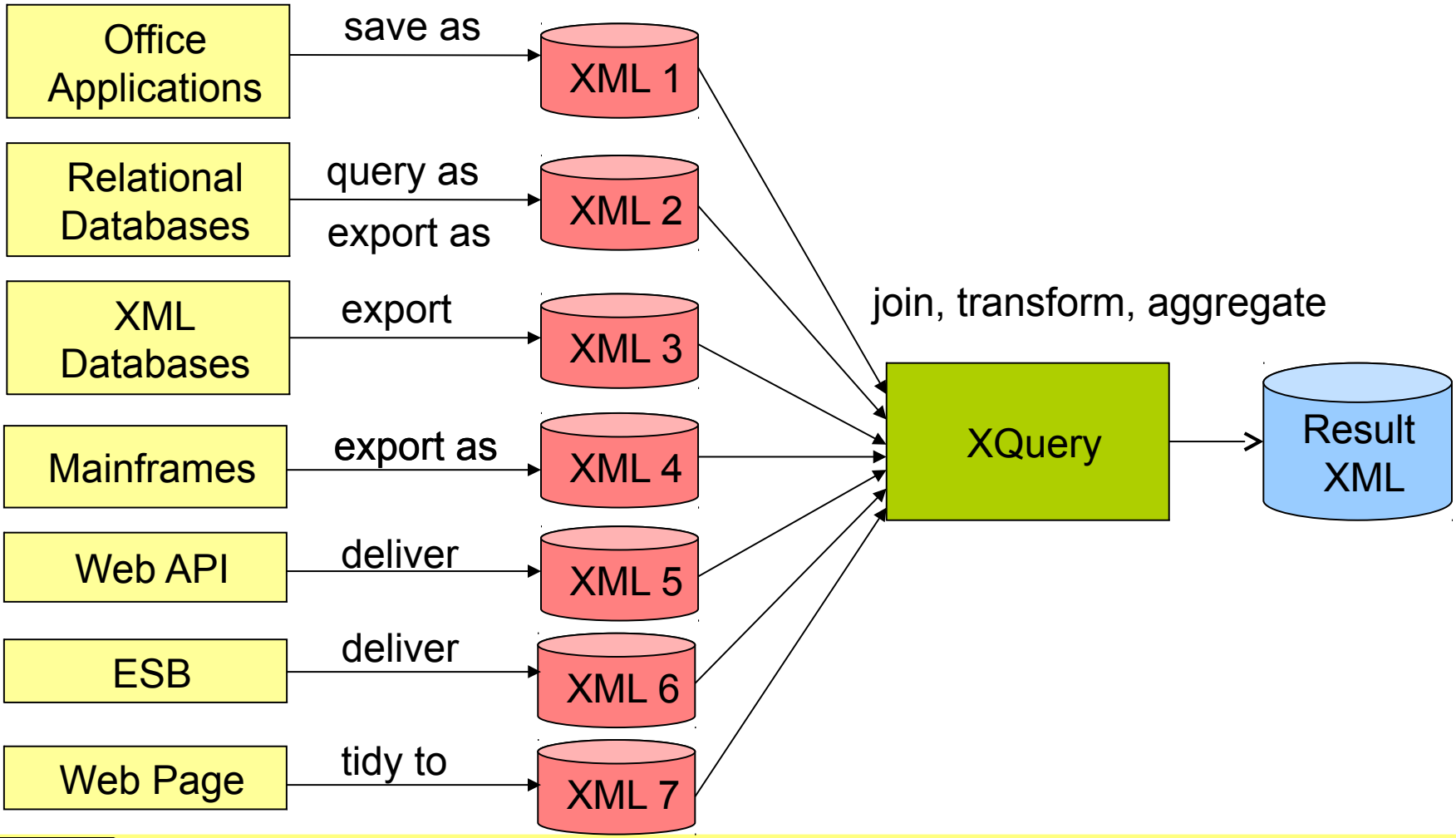
id	issn	short_name	vol	journal
1	0012-1606	Dev. Biol.	369	<journal> <name>Developmental Biology</name> <publisher>Elsevier</publisher> </journal>
2	8756-8233	Drugs Soc.	11	<journal> <name>Drugs and Society</name> <publisher>Taylor & Francis</publisher> </journal>

```
select id, vol, xmlquery('$j/name', passing journal as "j") as name
from journals
where
   xmlexists('$j[publisher="Elsevier"]', passing journal as "j")
```

id	vol	name
1	369	<name>Developmental Biology</name>

Use Case #3: Data Integration

- XML can be a clean format for Transfer/Interoperability



Use Case #3: Data Integration

- When all Data is in XML
  - XQuery makes it very easy to join datasets together
    - Query across Silos for new insight
    - Mash-up values into new datasets
    - Transform and Publish

- Many XQuery Vendors also offer:
  - SQL Queries from XQuery, results as XML
  - HTTP Queries from XQuery, results as XML or...
  - Invoking XSLT Transformations from XQuery
  - Various Serializers: XHTML, HTML5, JSON, Text, etc.

Use Case X...

- Alternative to DOM, SAX, StaX, VTD, XPath, XSLT, XProc e.g.
  - As a pipeline to perform several operation on XML
  - To narrow down the results from a Web Service
  - To create/query/update a Configuration file stored as XML.
- To create Office Documents (.docx, .odt, .epub, etc.).
- To Create a Web Application
  - XML output can be XHTML (Vendors: HTML5/JSON)
  - Get advanced Full-Text search for free
  - When dataset is XML or disparate

XQuery Standards

- XQuery 1.0
  - Developed by the W3C XML Query Working Group
    http://www.w3.org/XML/Query
  - First W.D. Feb 2001 / Rec. Jan 2007

- Related XML Query Standards
  - XDM (XPath and XQuery Data Model)
  - XQueryX
  - XQuery and XPath Full-Text
  - XQuery Update
  - XQuery Scripting

- XQuery 3.0 → First W.D. July 2008 / Rec. 2012???

summer school

XQuery Basics

XQuery Basics

- Aimed at Users Getting Started Quickly

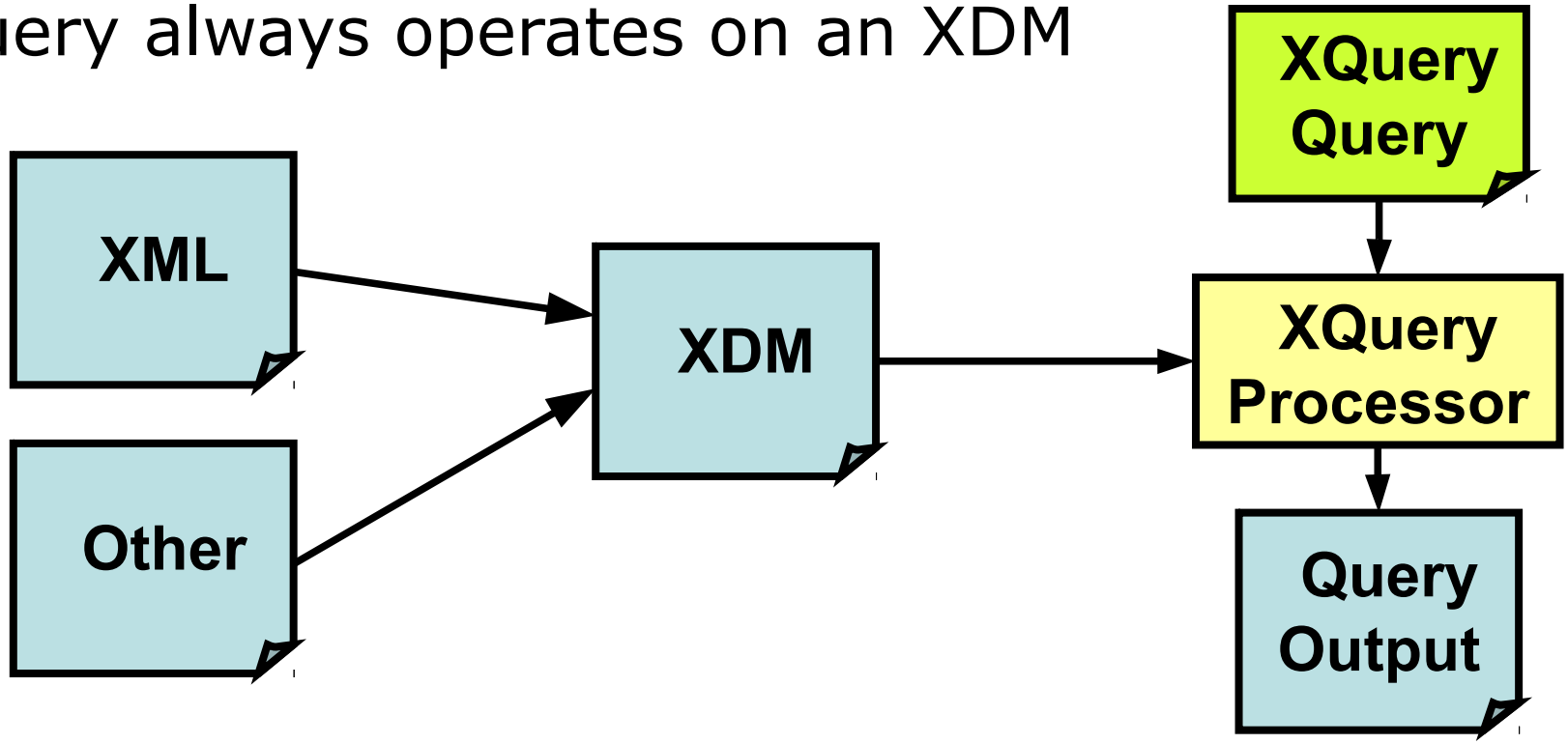
- Some aspects of XQuery we will examine:
  - Data/Type Model
  - XPath 2.0 Expressions
  - FLWOR Expressions

summer school

XPath and XQuery Data Model (XDM)

What is XDM?

- XQuery always operates on an XDM



- XDM is the Data Model for XPath and XQuery
- Understanding basics of XDM is key!

XDM Basics

- An XDM consists of Items and Sequences
  - Builds on XML Infoset and XML Schema
- Items are of two main types:
  - Node or Atomic Value
- Nodes
  - XML Documents are made of these!
  - Different types of nodes:
    - document, element, attribute, text, comment, processing-element
  - Have a Unique Identity!

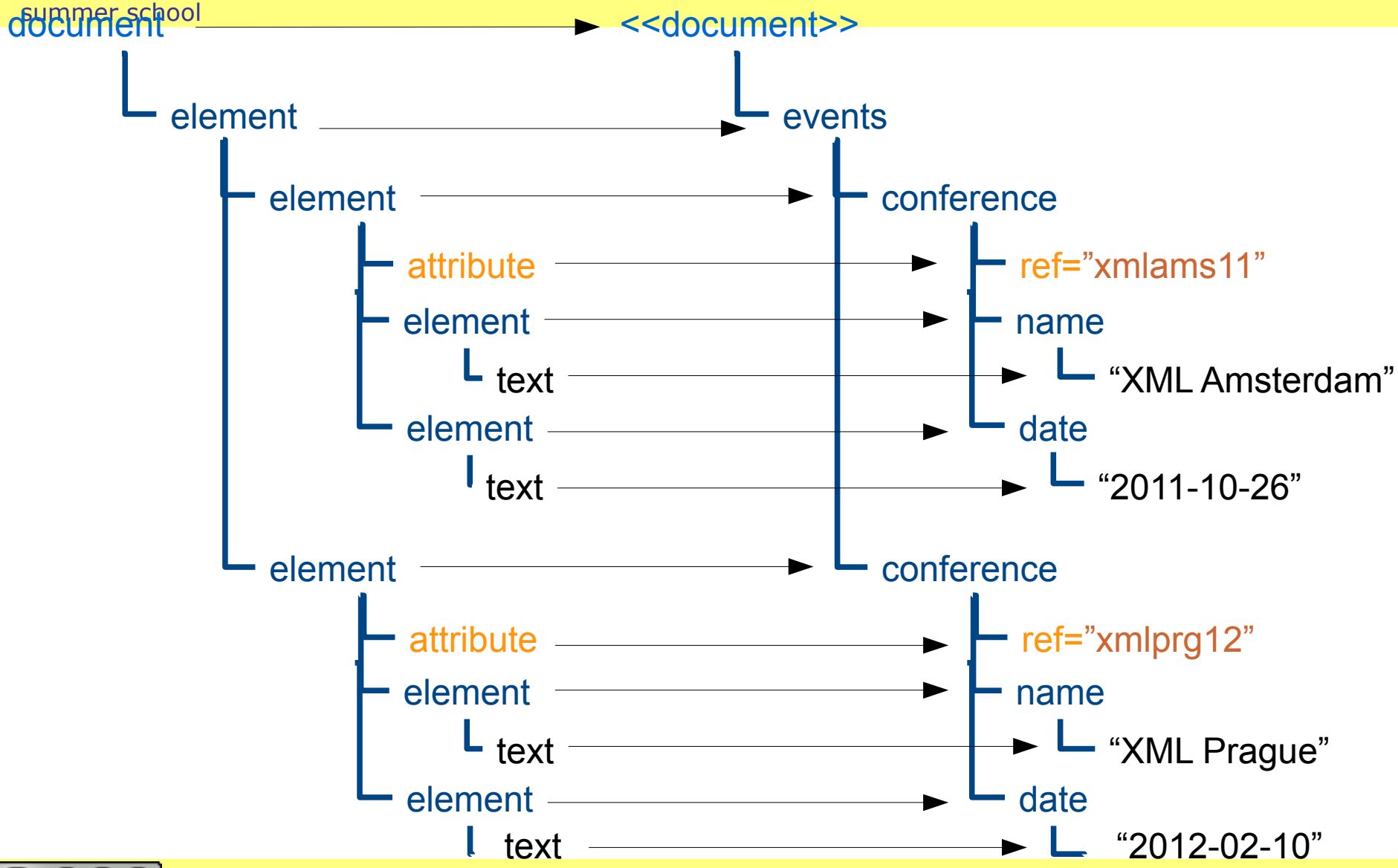
```
<root>
  <hello>world</hello>
  <hello>world</hello>
  ...
```

XDM - Node Trees

XML: Its a Tree of Nodes!

```
<events>
   <conference ref="xmlams11">
      <name>XML Amsterdam</name>
      <date>2011-10-26</date>
   </conference>
   <conference ref="xmlprg12">
      <name>XML Prague</name>
      <date>2012-02-10</date>
   </conference>
</events>
```

XDM – Node Trees



XDM – Atomic Values

- Atomic Values
  - i.e. Literal, Parameter to a function, or Computed Result
  - NOT Nodes!
  - Many different types of Atomic Value:
    - See: XML Schema Part 2: Datatypes
      - xs:string e.g. “I am a String”
      - xs:int e.g. 1234
      - xs:date e.g. xs:date(“2004-03-01”)

- Useful Links:
  - http://www.w3.org/TR/xpath-datamodel/#types-hierarchy
  - http://www.w3.org/TR/xmlschema-2/#built-in-datatypes

XDM – Type Hierarchy

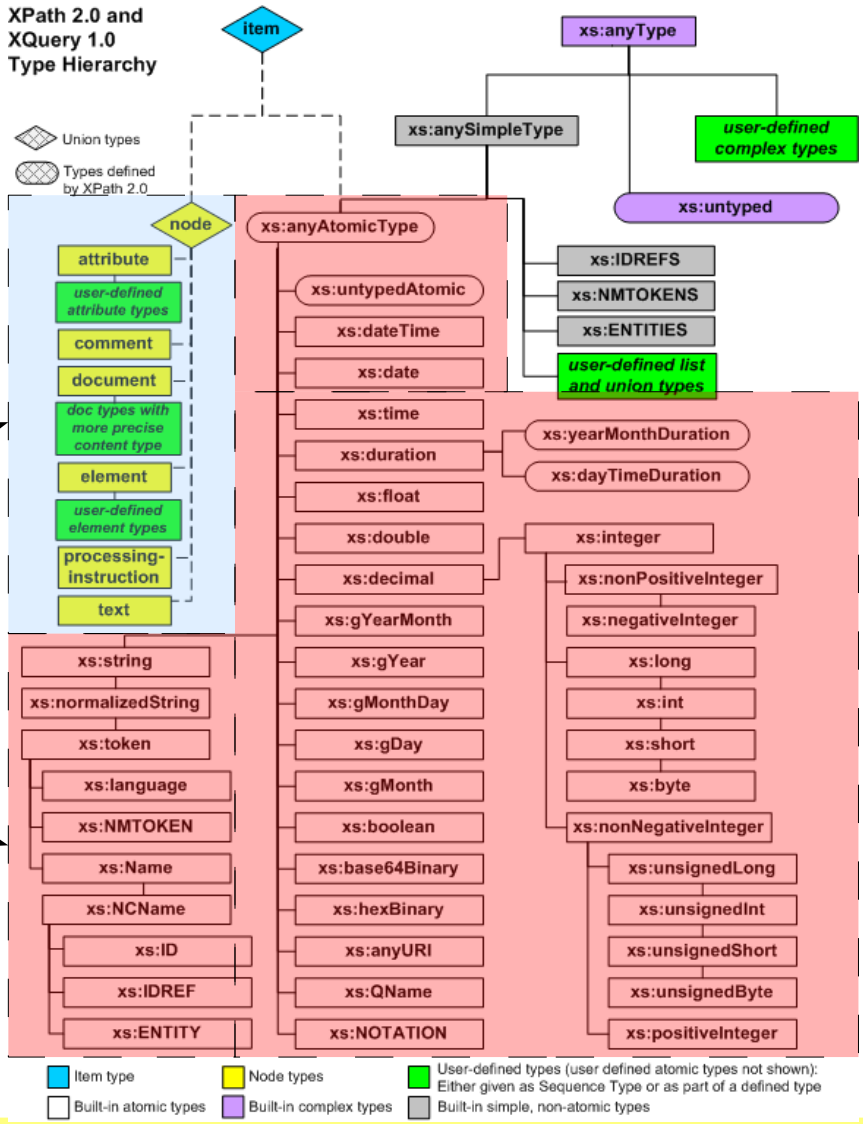
summer school

Simple!

- Probably only use a few of the Atomic Value Types

Nodes

Atomic Values



Modified from:
W3C XQuery 1.0 and XPath 2.0 Data Model (XDM) (Second Edition)

XDM – Nodes Quiz

- Quiz on XDM Nodes

```
<document lang="en_GB">
   <fragment1>Hello there <gn>James</gn> <fn>Smith</fn>, </fragment1>
   <fragment2>how are you today?</fragment2>
</document>
```

1) How many nodes are in the document?

2) What kind of node is 'fragment2'?

3) What are the names of the attributes?

4) How many text nodes are in the document?

5) What does the node tree look like? (Draw it!)

XDM - Sequences

- Sequences
  - An Ordered List
  - Sequence Constructor starts with '*(*' and ends with '*)*'
  - Consist of Zero or More Items

    ("hello", "world")
  - Can be mix of Nodes and Atomic Values

    ("hello", <gn>james</gn>, <fn>smith</fn>)
  - No Nested Sequences!

    ("a", "b", ("c", "d")) *becomes:* ("a", "b", "c", "d")

XDM - Sequences

- Sequences
  - An Item == Sequence containing just that Item

    ("hello") *is the same as:* "hello"
  - A Sequence with Zero Items, is an Empty Sequence

    () *is the Empty Sequence*
  - Can be the parameter to a function, a computed result, or the result of an expression e.g.

    "Find me all the names?"

    //name
    - Returns the Sequence of two Elements:

      (<name>adam</name>, <name>bob</name>)

Comparison Operators

- XQuery has two types of Comparison Operators

	Atomic Values	Sequences
Equal to	eq	=
Not equal to	ne	!=
Greater than	gt	>
Greater than or equal to	ge	>=
Less than	lt	<
Less than or equal to	le	<=

```
("james", "simon", "mark", "bob") = "mark"
```

- True

```
("james", "simon", "mark", "bob") = ("mark", "james")
```

- True

```
("james", "simon", "mark", "bob") = ("mark", "cliff")
```

- ...and this??

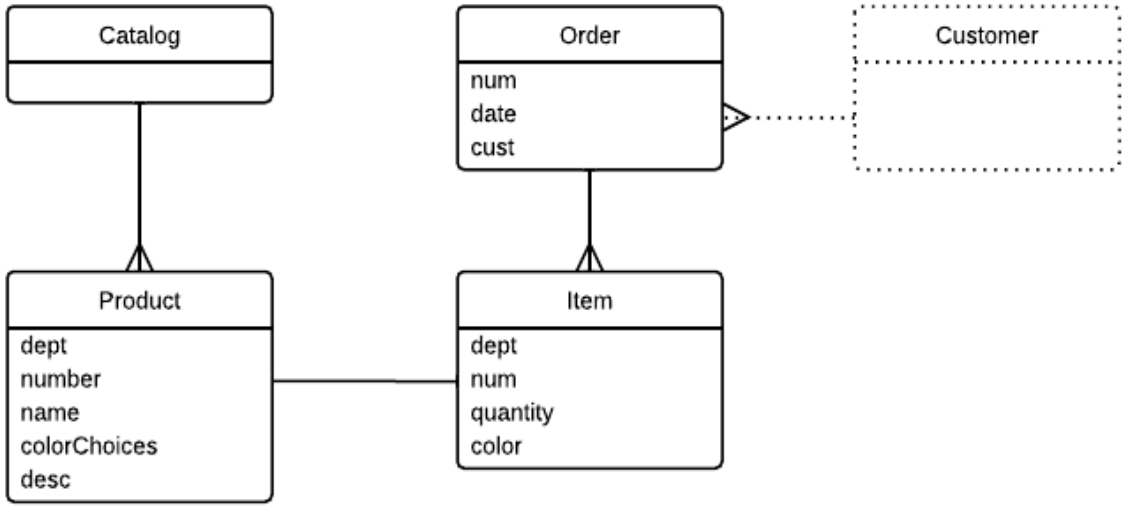
summer school

Practical XQuery

Example Data

- Initially to illustrate some simple XQuery we will use some very simple data centric XML

- Two documents:
  - Products
  - Orders (of Products)



Example Data

catalog.xml

```
<catalog>
   <product dept="WMN" commenced=”2009-02-04”>
      <number>557</number>
      <name language="en">Linen Shirt</name>
      <colorChoices>beige sage</colorChoices>
   </product>
   <product dept="ACC" commenced=”2009-02-04”>
      <number>563</number>
      <name language="en">Ten-Gallon Hat</name>
   </product>
   <product dept="ACC" commenced=”2012-01-01”>
      <number>443</number>
      <name language="en">Golf Umbrella</name>
   </product>
   <product dept="MEN" commenced=”2010-08-09”>
      <number>784</number>
      <name language="en">Rugby Shirt</name>
      <colorChoices>blue/white blue/red</colorChoices>
      <desc>Our <i>best-selling</i> shirt!</desc>
   </product>
</catalog>
```

Example Data

order.xml

```
<order num="00299432" date="2004-09-15" cust="0221A">
   <item dept="WMN" num="557" quantity="1" color="beige"/>
   <item dept="ACC" num="563" quantity="1"/>
   <item dept="ACC" num="443" quantity="2"/>
   <item dept="MEN" num="784" quantity="1" color="blue/white"/>
   <item dept="MEN" num="784" quantity="1" color="blue/red"/>
   <item dept="WMN" num="557" quantity="1" color="sage"/>
</order>
```

XQuery Example

summer school

input document

```
<order num="00299432" date="2004-09-15" cust="0221A">
   <item dept="WMN" num="557" quantity="1" color="beige"/>
   <item dept="ACC" num="563" quantity="1"/>
   <item dept="ACC" num="443" quantity="2"/>
   <item dept="MEN" num="784" quantity="1" color="blue/white"/>
   <item dept="MEN" num="784" quantity="1" color="blue/red"/>
   <item dept="WMN" num="557" quantity="1" color="sage"/>
</order>
```

query

```
for $d in distinct-values(doc("order.xml")//item/@dept)
let $items := doc("order.xml")//item[@dept = $d]
order by $d
return
   <department name="{$d}"
      totalQuantity="{sum($items/@quantity)}"/>
```

results

```
<department name="ACC" totalQuantity="3"/>
<department name="MEN" totalQuantity="2"/>
<department name="WMN" totalQuantity="2"/>
```

The Context

- You need something to process:
  - External. Set by processor
  - Request, by explicit function call
    - A document

```
doc("catalog.xml")
```

    - A collection of documents

```
collection("/shop/products")
```

- Or... XQuery generates original data

Documents and Collections

- doc() and collection() functions take a URI
- URI may or may not be de-referenced
- Both functions return document node(s).
- What is a Collection?
  - Implementation defined
    - A folder? Hierarchical?
    - A label?
- Typically followed by an expression.

Documents and Collections

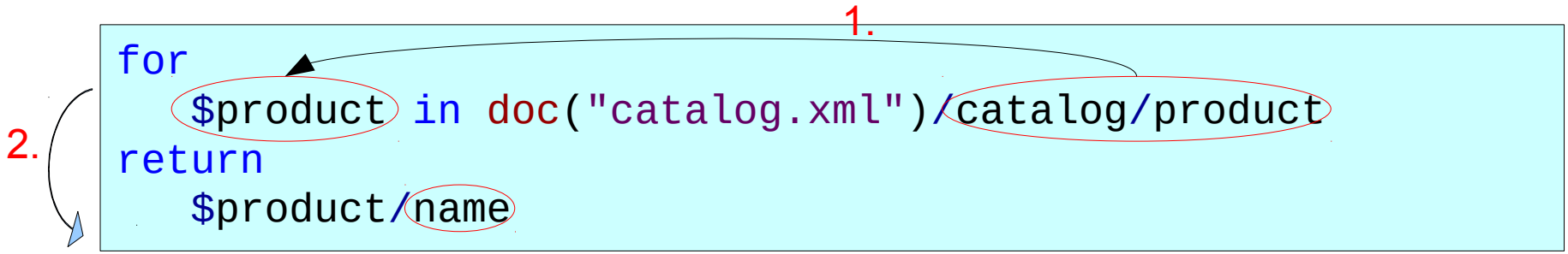
- 1 big document vs. collection of smaller documents
  - Why and When is it best to split?
    - How does your data come to you?
    - How do you deliver your data?
    - What is the unit of demarcation?
  - A Node is a Node is a Node
    - Do you need to know that it was originally X documents?
    - Can Transform back into any shape?
  - How do you update?
  - Processor constraints!

Expressions

- Path Expressions e.g. /xml/summer/school
- FLOWR Expressions:
  - **for**
    - creates a sequence of nodes
  - **let**
    - binds a sequence to a variable
  - **where**
    - filters the nodes on a boolean expression
  - **order by**
    - sorts the nodes
  - **return**
    - gets evaluated once for every node

Simple FLWOR Example

- Just the 'F' and 'R':



1. Bind the $product variable to each /catalog/product node in turn during iteration

2. Return the evaluation of $product/name for each iteration
   i.e. each /catalog/product/name element

Simple FLWOR Example

- Result:

```
<name language="en">Linen Shirt</name>
<name language="en">Ten-Gallon Hat</name>
<name language="en">Golf Umbrella</name>
<name language="en">Rugby Shirt</name>
```

  - An XML Fragment!

Simple FLWOR Example with Position

- Iteration position can be bound with '*at*'

```
for
   $product at $i in doc("catalog.xml")/catalog/product
return
   <product idx="{$i}">{$product/name/text()}</product>
```

- Just like in XSLT, expressions can be evaluated inline using **{***expression***}** notation

- New Nodes can be constructed to change structure
  - Direct Constructors
  - Computed Constructors

```
element product {
    attribute idx { 99 },
    text { "Our New Product" }
}
```

Simple FLWOR Example with Position

- Result:

```
<product idx="1">Linen Shirt</product>
<product idx="2">Ten-Gallon Hat</product>
<product idx="3">Golf Umbrella</product>
<product idx="4">Rugby Shirt</product>
```

- Result is in Document Order!
- 'Product' is directly constructed element, with attr.
- text() node of Product Name was copied
- Re-write the query create valid XML document...

Simple FLWOR Example with Position

summer school

```
<products>{
    for
       $product at $i in doc("catalog.xml")/catalog/product
    return
       <product idx="{$i}">{$product/name/text()}</product>
}</products>
```

```
<products>
   <product idx="1">Linen Shirt</product>
   <product idx="2">Ten-Gallon Hat</product>
   <product idx="3">Golf Umbrella</product>
   <product idx="4">Rugby Shirt</product>
</products>
```

- What about *document{ … }?*

FLWOR - Bindings

```
for
   $product in doc("catalog.xml")/catalog/product
let
   $catalog-age := days-from-duration(
     current-date() - xs:date($product/@commenced)
   )
return
    <product ref="{$product/number}">
       Added to the catalog {$catalog-age} days ago.
    </product>
```

1. Bind the $catalog-age variable to each expression during iteration

2. Return the evaluation of
   $catalog-age expression for the iteration

- A FLWOR expression can have any number of bindings

FLWOR – Where Clauses

```
for
    $product in doc("catalog.xml")/catalog/product
where
    $product/@dept eq "ACC"
return
    $product
```

- The ***where*** clause *may* be evaluated once for each iteration
  - It *can* be more efficient to use a **predicate** instead
  - All variable bindings are available, unlike in a predicate.
    - Not all *where* expressions can be rewritten as predicates.

e.g.

```
for
   $i in (1,2,3), $j in (2,3,4)
where
   $j > $i
return ($i, $j)
```

FLWOR – Ordering Results

```
for
    $product in doc("catalog.xml")/catalog/product
order by
    xs:date($product/@commenced) descending
return
    $product
```

- Ordering may be either ***ascending*** or ***descending***
- You can order on multiple values, e.g.

```
order by
    xs:date($product/@commenced) descending,
    xs:int($product/number) ascending
```

- **Hint:** When ordering, ensure the type of the value

Complete FLWOR Example

```
<products>{
    for
       $product at $i in doc("catalog.xml")/catalog/product
    let
       $catalog-age := days-from-duration(
              current-date() - xs:date($product/@commenced)
       )
    where
       $product/dept eq "ACC"
    order by
       xs:date($product/@commenced) descending,
       xs:int($product/number) ascending
    return
       <product idx="{$i}" ref="{$product/number}">
              <age>{$catalog-age}</age>
              <name>{$product/name/text()}</name>
       </product>
}</products>
```

Beheaded FLWOR

- Can start with Let:

```
let $tomorrow := current-date() + xs:dayTimeDuration("P1D")
return
    <next-date>{$tomorrow}</next-date>
```

- Providing *$product* is bound, the following is valid:

```
let
      $catalog-age := days-from-duration(
            current-date() - xs:date($product/@commenced)
      )
where
      $product/dept eq "ACC"
order by
      xs:date($product/@commenced) descending
return
      <product idx="{$i}" ref="{$product/number}">
            <age>{$catalog-age}</age>
            <name>{$product/name/text()}</name>
      </product>
```

Variable Scope

- XQuery is functional
  - It has immutable variables
  - Variables are "*bound*" to a value

- What result does the following yield?

```
let $y := 1 return
    for $x in (1 to 100)
        let $y := $y + $x return
    $y
```

a) 5051
b) The Sequence (2 to 101)
c) A sequence ending in 5051
d) The Sequence (1 to 100)
e) 5050

Variable Scope

- Variable Bindings have a limited Scope

- Is this valid?

```
<something>
    <today>
    {
        let $now := current-date() return
            $now
    }
    </today>
    <tomorrow>
    {
        $now + xs:dayTimeDuration("P1D")
    }
    </tomorrow>
</something>
```

Variable Scope

- ...and is this valid?

```
<something>
    {
        let $now := current-date() return
            <today>$now</today>
        ,
        <tomorrow>
        {
            $now + xs:dayTimeDuration("P1D")
        }
        </tomorrow>
    }
</something>
```

summer school

Advanced XQuery

Advanced XQuery

- XQuery is a full functional Programming Language
- Aspects of XQuery we will examine
  - Conditional Expressions
  - Functions

Conditional Expressions

- If, then, else syntax:

```
if($date lt current-date())then
    <result>The date {$date} is in the past</result>
else if($date gt current-date())then
    <result>The date {$date} is in the future</result>
else
    <result>The date {$date} is today!</result>
```

- Parentheses must surround the expression for **if**
- **if** expressions can be chained, i.e. **else if**
- **else** is always required
  - Can just use the empty sequence i.e. **else** ()

Effective Boolean Value

- Expression of **`if`** statement must be boolean
  - if not, its *effective boolean value* is found
- effective boolean value is false for:
  - the **`xs:boolean`** value **`false`**
  - the number **`0`** or **`NaN`**
  - a zero-length string
  - the empty sequence
- otherwise it is true (e.g. a list of elements)

```
if(doc("order.xml")//item)then
    <result>Found some items</result>
else
    <result>Error: Zero items in the order</result>
```

Logical Expressions

- Combine boolean values: **and**, **or**
  - **and** has precedence over **or**
  - use parentheses to manage precedence

```
if($is-discounted and ($discount gt 10 or $discount lt 0))then
    10
else
    $discount
```

- Use **not** function to invert boolean value

```
if(not($is-discounted))then
    0
else
    $discount
```

- **not** function will also resolve *effective* boolean value

Type Conditional Expressions

- typeswitch syntax:

```
typeswitch($something)

    case $n as element(name) return
        <result>found the name: {$n/text()}</result>

    case $e as element() return
        <result>found an element: {local-name($e)}</result>

    case $t as text() return
        <result>found the text: {$t}</result>

    case $i as xs:integer return
        <result>found the integer: {$i}</result>

    default return
        ()
```

Type Conditional Expressions

- Identity Transform using typeswitch:

```
declare function local:transform($node) {
    for $n in $node return
        typeswitch($n)

            (: TODO add you overrides here:)

            case document-node() return
                document {
                    local:transform($n/*)
                }

            case element() return
                element {node-name($n)} {
                    local:transform($n/@*|$n/node())
                }

            default return
                $n
};
```

Functions

- XQuery has many built-in functions
  - Defined in W3C Spec:
    - XQuery 1.0 and XPath 2.0 Functions and Operators
      http://www.w3.org/TR/xpath-functions/
  - So far we have seen:

    doc collection distinct-values sum current-date days-from-duration not local-name node-name
  - There are >150 functions available in XQuery 1.0

Extension Functions

- Vendors/Processors may also provide extension functions
- e.g. SQL Queries, Sending Email, DB Management
  - eXist-db provides ~600 functions
- Extension functions are processor specific
  - ...Non-Portable XQuery code
  - EXPath and EXQuery Projects try to standardise

http://www.expath.org
http://www.exquery.org

User Defined Functions

- You can also write your own functions in XQuery

- Functions must have a fully qualified name
  - e.g. **my:function1**
  - *The "local" prefix may be used for functions in a main module. e.g.* **local:function1**

- Functions may be placed in library modules
  - Which are imported by other modules or main module

- Many common functions available at FunctX

  http://www.xqueryfunctions.com/

User Defined Functions

- Function Declaration:

```
declare function local:my-first-function() {

    (: TODO your function body code goes here! :)

};
```

  - Functions cannot have an empty body (i.e. above!)

- Functions may take parameters

```
declare function local:my-first-function($thing, $other) {
    <this>{$thing}</this>
};
```

- Parameters may be explicitly typed

```
declare function local:my-second-function($when as xs:time) {
    ...
```

User Defined Functions

- Parameters may be sequences with constraints
  - Cardinality
    - One

```
declare function local:my-third-function($date as xs:date)
```

    - One or More **+**

```
declare function local:my-third-function($dates as xs:date+)
```

    - Zero or One *****

```
declare function local:my-third-function($dates as xs:date*)
```

User Defined Functions

- Return values may be explicitly typed

```
declare function local:my-third-function($date as xs:date)
 as element(calendar) {

   <calendar>Starting from: {$date}</calendar>
};
```

- Return values may specify a cardinality constraint

```
declare function local:my-third-function($dates as xs:date)
 as element(calendar)+ {

    for $date in $dates return
        <calendar>Starting from: {$date}</calendar>
};
```

Function Modules

- XQuery code lives in Modules
  - Two types of module:
    - Main Module
      - Everything you have seen up to now!
      - Query Body (and maybe functions)
      - Can Import other Libraries
      - .xqy file
    - Library Module
      - Has a namespace!
      - Just functions
      - Can Import other Libraries
      - .xqm file

Function Modules

- Main Module

```
xquery version "1.0";

import module namespace my = "http://my-function-module"
 at "my-funcs.xqm";

<greetings>{
    my:say-hello("Adam Retter")
}</greetings>
```

- Library Module

```
xquery version "1.0";

module namespace my = "http://my-function-module";

declare function my:say-hello($name) {
    <hello>{$name}</hello>
};
```

summer school

XML Databases

What is an XML Database?

- More than just a filesystem
- Unit of storage is the Document
- Node aware, e.g. Indexing
- CRUD operations
- Full-Text capabilities
- May support non-XML content

XML Enabled Database

- XML Enabled Database
  - RDBMS approaches:
    - XML Stored in CLOB
    - XML Shredding into tables. e.g. Oracle XML Schema Table.
    - ISO XML Type for columns
  - Good for small amounts of standalone XML
  - Bad for complex queries across XML and Tables
  - Commercial: Oracle Database, IBM DB2, SQL Server
  - Open Source: PostgreSQL

Native XML Databases

- Native XML Databases
  - Stores/Retrieves/Queries Documents
  - Defines 'Collection's
  - Indexes optimised for XML
  - Supports XQuery (possibly: XSLT, XQ FT, XQ SE, etc.)
  - More like a Document Management Platform
  - NXMLDB++
    - Binary content, REST, Web, etc, etc.

Native XML Database Products

- Commercial:
  - 28msec Sausalito
    - Cloud. XML + JSON DB and App Server
  - MarkLogic Server
    - Clusterable DB
  - EMC xDB

- Open Source:
  - BaseX
    - Compliant standards support
  - eXist-db
    - Well established and feature rich
  - Others: Qizx. Sedna.

Advantages of Native XML Database

- Compared to a Filesystem
  - Manage Document Access
  - Indexing and then Querying
  - Metadata
  - Fine-grained updates

- Compared to RDBMS
  - No need to take apart your dataset
  - Can store Relational and Hierarchical data
  - Better full-text search
  - Support for Metadata and Meta-Metadata etc.
  - Schema Free

- Native XML Database. Established in 2000
- Open Source. LGPL. Commercial Support available!
- Hierarchical Collections of Documents
- Supports XML and Binary Documents. WebDAV + REST
- XQuery, XQuery Update, Proprietary Full-Text
- Also: XSLT 2.0, XForms, XProc, XInclude, JSON, XHTML, HTML5
- Full Web App platform with XQuery extensions

summer school

eXist-db: In Practice

Native XML Databases

How to get setup?

- eXist-db is written in Java 6
  - You need Oracle/Open JDK 6 or 7
- Download and install 2.0 Tech Preview from
  - http://www.exist-db.org/exist/download.html
- Two apps:
  - Database and Web Server
  - Simple GUI Admin Client

Working with Documents

- Documents can be added to the database by:
  - GUI Admin Client
  - Web Admin Client
    http://localhost:8080/exist/admin
  - WebDAV
    http://localhost:8080/exist/webdav/db
  - REST
    http://localhost:8080/exist/rest/db
  - XQuery (xmldb:store)
  - Java/PHP/.net/APIs: XML-RPC, SOAP, Eclipse Plugin. Etc.

Collections

- Documents are stored in Collections
- Root collection is **/db**
- Collections can contain sub-collections
- The collection hierarchy is inherited!

Quiz

How do I get all of the marketing collection?

What does **collection**("/db/journals") return?

What does **collection**("/db/books/blogs") return?

Working with the Database

- Where are my documents now?
  - Stored in optimised BTrees and Indexes
  - Stored in .dbx files in *webapp/WEB-INF/data*
- How do I get my documents back?
  - Same; Admin Clients, WebDAV, REST, doc(), etc.
- You can also store non-XML files
- Database logs are in *webapp/WEB-INF/logs*

Querying the Database

- REST API
  - HTTP GET

```
http://localhost:8080/exist/rest/db/?
_query=<date>{current-dateTime()}</date>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<exist:result xmlns:exist="http://exist.sourceforge.net/NS/exist" exist:hits="1"
exist:start="1" exist:count="1">
   <date>2012-09-07T15:44:23.275+01:00</date>
</exist:result>
```

  - HTTP POST

```
http://localhost:8080/exist/rest/db/
```

```
<query xmlns="http://exist.sourceforge.net/NS/exist">
   <text><![CDATA[
         <date>{current-date()}</date>
      ]]></text>
</query>
```

Querying the Database

- GUI Admin Client 
- eXide
- SOAP / WebDAV / XML-RPC / Java / PHP / etc.
- Stored Queries
  - XQuerys can be stored into the database
  - Executed later e.g. REST Server by URI
  - See Demo.

summer school

XML Applications

What is an XML Application?

- Developed entirely with XML Technologies
  - W3C Standards
    - **XML**
      - Storage and Transfer
    - **XPath, XQuery, XSLT, XSL-FO**
      - Process, Query, Transform
    - **XForms**
      - Capture and Update
  - Processor and Vendor extensions
- What about JavaScript/JSON?

Why build XML Applications?

- Easier/faster than traditional approaches
  - Higher Level Languages
  - Empower non-programmers
- No data conversion in and out of database
  - No mapping Objects to Tables
  - No Java (or others) types to SQL types
- Less Programming!
  - 55% - 80% reduction in code is possible
- Common thread: XPath and Schema Data Types

Why build XML Applications?

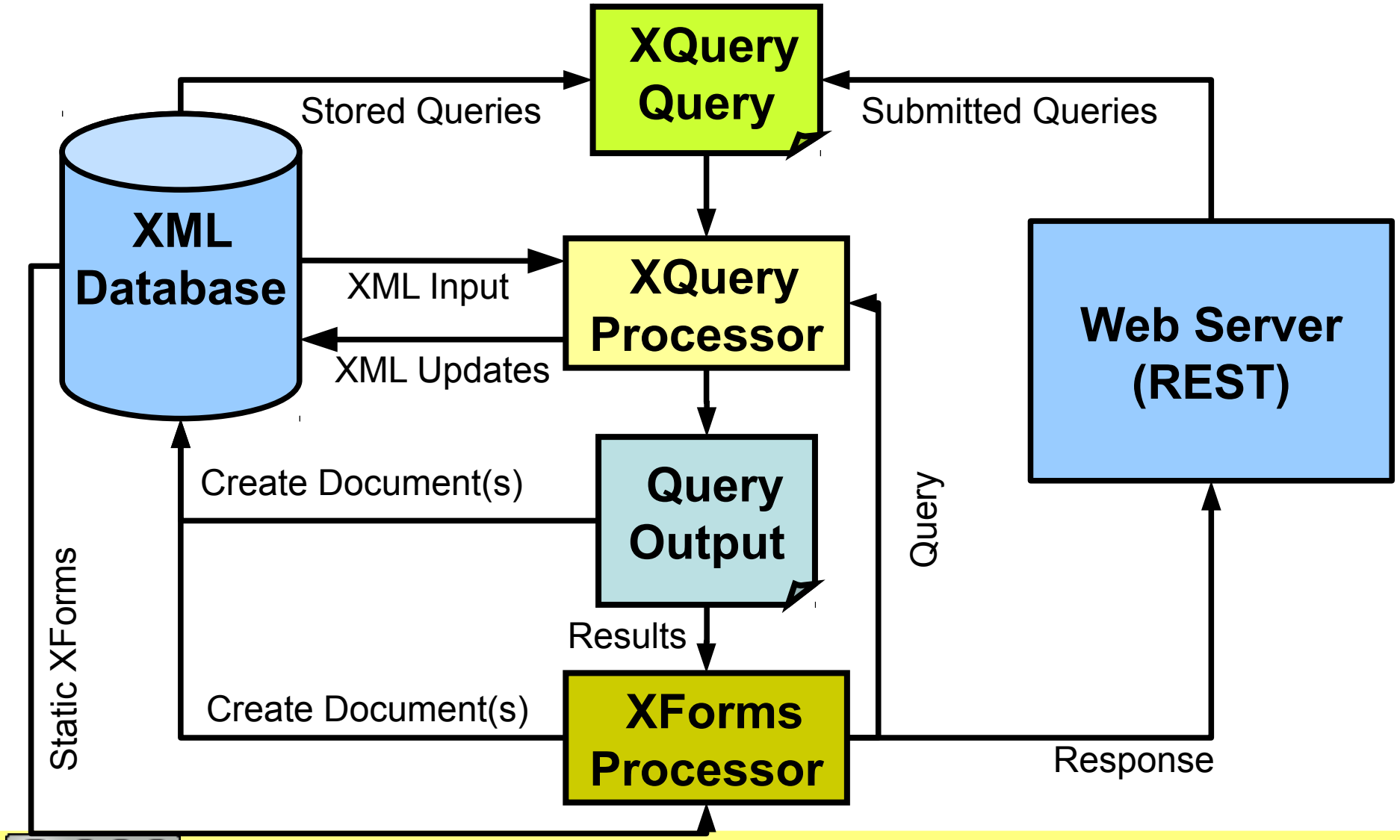
- The Web!
  - SGML 'History Lesson':
    - HTML was inspired by SGML.
    - HTML 4 is a subset of SGML.
    - **XML** is a subset of SGML!
    - XML is HTML:
      - XHTML 1.0 and 1.1
      - HTML 5 (XHTML)
  - Vendors: Other Serializers (JSON, HTML4, HTML5, etc)

XRX

- XRX is an architecture for XML Applications
  - **X**Forms
  - **R**EST
  - **X**Query
  - Its an approach to MVC for Web Apps
    - Not just XForms, REST and XQuery
      - XSLT, XSL-FO, XUpdate, Xinclude, etc.
  - Majority of code is typically XQuery
    - Why not XProc?

summer school

Typical XRX Architecture



summer school

Building an XML Application

What are we building?

- Dataset is PubMed
  - Just 2012 / 691,122 Articles / 4.81 GB XML File
  - Details of Articles published in Medical Journals
    - http://www.ncbi.nlm.nih.gov/pubmed
- Application Should:
  - Search Articles
    - By Journal
    - By Author
    - By Date
    - By Keyword
  - Browse / Summarise Results

PubMed Structure

```
<PubmedArticleSet>

    <PubmedArticle>
        <MedlineCitation Owner="NLM" Status="In-Process">
            <PMID Version="1">22905362</PMID>
            <DateCreated> [4 lines]
            <Article PubModel="Print">
                <Journal> [11 lines]
                <ArticleTitle>Feasibility and acceptability of rapid HIV screening in a labour ward in Togo.</Art
                <Pagination> [2 lines]
                <Abstract>
                    <AbstractText Label="BACKGROUND" NlmCategory="BACKGROUND">HIV screening in a labour ward is th
                    <AbstractText Label="FINDINGS" NlmCategory="RESULTS">A cross-sectional survey was conducted in
                    <AbstractText Label="CONCLUSIONS" NlmCategory="CONCLUSIONS">This study is the first to show in
                </Abstract>
                <Affiliation>Département des Sciences Biologiques et Santé Publique, Faculté Mixte de Médecine et
                <AuthorList CompleteYN="Y"> [61 lines]
                <Language>eng</Language>
                <PublicationTypeList>
                    <PublicationType>Journal Article</PublicationType>
                    <PublicationType>Research Support, Non-U.S. Gov't</PublicationType>
                </PublicationTypeList>
            </Article>
            <MedlineJournalInfo> [5 lines]
            <CitationSubset>IM</CitationSubset>
            <CitationSubset>X</CitationSubset>
        </MedlineCitation>
        <PubmedData> [28 lines]
    </PubmedArticle>
```

summer school

Step 1: Composing the Queries

Building an XML Application

Search by Journal

- Find Articles by Journal ISSN

```
xquery version "1.0";

declare function local:by-journal-issn($issn) {
  collection("/db/pubmed")/PubmedArticleSet/PubmedArticle
    [MedlineCitation/Article/Journal/ISSN eq $issn]
};

let $issn := "1758-2652" return
      local:by-journal-issn($issn)
```

eXist-db Index Configuration

- PubMed is a big dataset!
- Databases rely on Indexes for performance
- Indexes can be used for:
  - Comparisons
  - Full Text Search
  - Geospatial Calculation
- Indexes are configured in:
  - /db/system/config/db
  - Collection Configuration (collection.xconf)

eXist-db Index Configuration

- `Find Articles by Journal ISSN` comparison is:

```
[MedlineCitation/Article/Journal/ISSN eq $issn]
```

- Index Configuration for /db/pubmed collection
  - /db/system/config/db/pubmed/collection.xconf:

```
<collection xmlns="http://exist-db.org/collection-config/1.0">
    <index>

        <!-- journal -->
        <create qname="ISSN" type="xs:string"/>

    </index>
</collection>
```

- **Note**: Need to Re-Index!

Search by Journal

- Find Articles by Journal Title

```
xquery version "1.0";

declare function local:by-journal-name($name) {
  collection("/db/pubmed")/PubmedArticleSet/PubmedArticle
    [MedlineCitation/Article/Journal/Title eq $name]
};

let $name := "Journal of the International AIDS Society"
return
      local:by-journal-name($name)
```

- What if I don't know the name of the Journal?

Full Text Search in eXist-db

- Full Text Search in eXist-db
  - Not W3C XQuery Full Text
  - Relies on Extension Functions and Lucene Indexes

- You must establish an Index

```
<collection xmlns="http://exist-db.org/collection-config/1.0">
    <index>

        <!-- journal -->
        <create qname="ISSN" type="xs:string"/>
        <create qname="Title" type="xs:string"/>
        <text qname="Title"/>

    </index>
</collection>
```

Full Text Search in eXist-db

- Functions for Full Text:
  - **ft:query**($nodes, $query)
    - Executes a Lucene query against indexed nodes
    - $query is:
      - Keywords
      - Lucence Query Syntax
      - XML Query Description
  - **ft:score**($node)
    - In conjunction with ft:query
    - Used in the `*order by*` clause of a FLWOR

Search by Journal

- Find Articles by Journal Title (Full Text)

```
xquery version "1.0";

declare function local:by-journal-name-ft($name) {
 collection("/db/pubmed")/PubmedArticleSet/PubmedArticle
  [ft:query(MedlineCitation/Article/Journal/Title, $name)]
};

let $name := "international" return
      local:by-journal-name-ft($name)
```

- Wildcard Example:

```
let $name := "bio*" return
      local:by-journal-name-ft($name)
```

Search by Journal

- Find Articles by Journal Title (Full Text)

```
xquery version "1.0";

declare function local:by-journal-name-ft($name) {
 collection("/db/pubmed")/PubmedArticleSet/PubmedArticle
  [ft:query(MedlineCitation/Article/Journal/Title, $name)]
};

let $name := "international" return
      local:by-journal-name-ft($name)
```

- Wildcard Example:

```
let $name := "bio*" return
      local:by-journal-name-ft($name)
```

Search by Author

- Find Articles by Author

```
xquery version "1.0";

declare function local:by-author($name) {
  collection("/db/pubmed")/PubmedArticleSet/PubmedArticle
    [MedlineCitation/Article/AuthorList/Author
     [ForeName eq $name or LastName eq $name]
    ]
};

let $name := "Castellano" return
      local:by-author($name)
```

Search by Author

- Find Articles by Author

```
xquery version "1.0";

declare function local:by-author($name) {
  collection("/db/pubmed")/PubmedArticleSet/PubmedArticle
    [MedlineCitation/Article/AuthorList/Author
     [ForeName eq $name or LastName eq $name]
    ]
};

let $name := "Castellano" return
      local:by-author($name)
```

Search by Date

- Published After Year

```
xquery version "1.0";

declare function local:by-date($year as xs:int) {
  collection("/db/pubmed")/PubmedArticleSet/PubmedArticle
    [MedlineCitation/DateCreated/Year ge $year]
};

let $after-year := 2012 return
      local:by-date($after-year)
```

Search by Keyword

- Article by Keyword
  - ArticleTitle
  - Abstract

```
xquery version "1.0";

declare function local:by-keyword($keyword) {
  collection("/db/pubmed")/PubmedArticleSet/PubmedArticle
    [ft:query(
      (MedlineCitation/Article/Abstract,
       MedlineCitation/Article/ArticleTitle),
     $keyword)]
};

let $keyword := "medical" return
       local:by-keyword($keyword)
```

summer school

Step 2: More than just queries

Building an XML Application

From Queries to Application

- Its not very "*App*" yet...
  - How do people use my queries?
  - How to remove hard-coded parameter values?
  - How can I deliver Web Pages?

How do people use my queries?

- Store them into the database: /db/pubmed
- Execute by:
  - Admin Client (Requires Software Install)
  - Programming API Call (Requires Programming)
  - HTTP call (e.g. Web Browser)
    - Two options:
      - REST Server
      - RESTXQ

REST Server

- REST Server: http://localhost:8080/exist/rest
  - Append database URI to REST URI
    - http://localhost:8080/exist/rest/db/pubmed/
  - RESTful access to DB
    - PUT, POST, DELETE (for document updates)
    - GET (document retrieval)
  - Query execution
    - ?_query= or POST XML doc containing query.
  - Stored query execution
    - http://localhost:8080/exist/rest/db/pubmed/find-by-issn.xql

RESTXQ

- RESTful Annotations for XQuery 3.0

  http://localhost:8080/exist/restxq

```
xquery version "3.0";

declare namespace rest = "http://exquery.org/ns/restxq";

declare variable $local:issn := "1758-2652";

declare
    %rest:GET
    %rest:path("/journal/issn")
function local:by-journal-issn() {
  collection("/db/pubmed")/PubmedArticleSet/PubmedArticle
    [MedlineCitation/Article/Journal/ISSN eq $local:issn]
};
```

Removing hard-coded parameter values

- REST Server
  - Send in Query String of URL or HTML Form
  - Receive with request:get-parameter($name, $def)

```
xquery version "1.0";

declare function local:by-journal-issn($issn) {
  collection("/db/pubmed")/PubmedArticleSet/PubmedArticle
    [MedlineCitation/Article/Journal/ISSN eq $issn]
};

let $issn := request:get-parameter("issn", "1758-2652")
return
      local:by-journal-issn($issn)
```

Removing hard-coded parameter values

- REST Server: What if there is no parameter?

```
xquery version "1.0";

declare function local:by-journal-issn($issn) {
  collection("/db/pubmed")/PubmedArticleSet/PubmedArticle
    [MedlineCitation/Article/Journal/ISSN eq $issn]
};

let $issn := request:get-parameter("issn", ()) return
    if($issn)then
          local:by-journal-issn($issn)
       else
       (
          response:set-status-code(400),
          <error>You must provide an ISSN e.g.
              {request:get-url()}?issn=1758-2652</error>
       )
```

Removing hard-coded parameter values

- RESTXQ
  - Use Templates
    - In the path %rest:path(“/journal/issn/{$issn}”}

```
xquery version "3.0";

declare namespace rest = "http://exquery.org/ns/restxq";

declare
    %rest:GET
    %rest:path("/journal/issn/{$issn}")
function local:by-journal-issn($issn) {
  collection("/db/pubmed")/PubmedArticleSet/PubmedArticle
    [MedlineCitation/Article/Journal/ISSN eq $issn]
};

()
```

Removing hard-coded parameter values

- Use Templates
  - In the Query String
    - %rest:query-param(“issn”, “{$issn}”)
  - From a HTML Form
    - %rest:form-param(“issn”, “{$issn}”)

```
declare
    %rest:GET
    %rest:path("/journal/issn")
    %rest:query-param("issn", "{$issn}", "1758-2652")
function local:by-journal-issn($issn) {
  collection("/db/pubmed")/PubmedArticleSet/PubmedArticle
    [MedlineCitation/Article/Journal/ISSN eq $issn]
};
```

Removing hard-coded parameter values

- RESTXQ: What if there is no parameter?

```
declare
    %rest:GET
    %rest:path("/journal/issn")
    %rest:query-param("issn", "{$issn}")
function local:by-journal-issn($issn) {
    if($issn) then
        (: removed for brevity :)
    else
    (
        <rest:response>
            <http:response status="400"/>
        </rest:response>,
        <error>You must provide an ISSN e.g.
            {rest:uri()}?issn=1758-2652</error>
    )
};
```

How can I deliver Web Pages?

- Use XHTML just as though it was XML
  - Store Images, CSS, JavaScript in db?
- Transform from XML to XHTML
  - XQuery or XSLT
- Serialize appropriately
  - Serializers for HTML, XHTML, HTML5, Text, JSON
  - REST Server
    - declare option exist:serialize "method=html5";
  - RESTXQ
    - %output:method("html5")

And...

- Things a Web App should/may have:
  - Wizzy JavaScript?
  - Security
  - API
    - URIs
    - Content Negotiation
  - Integration with 3rd Party sites