



XQuery and XML Applications

Adam Retter

 adam.retter@googlemail.com

 [@adamretter](https://twitter.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

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



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!

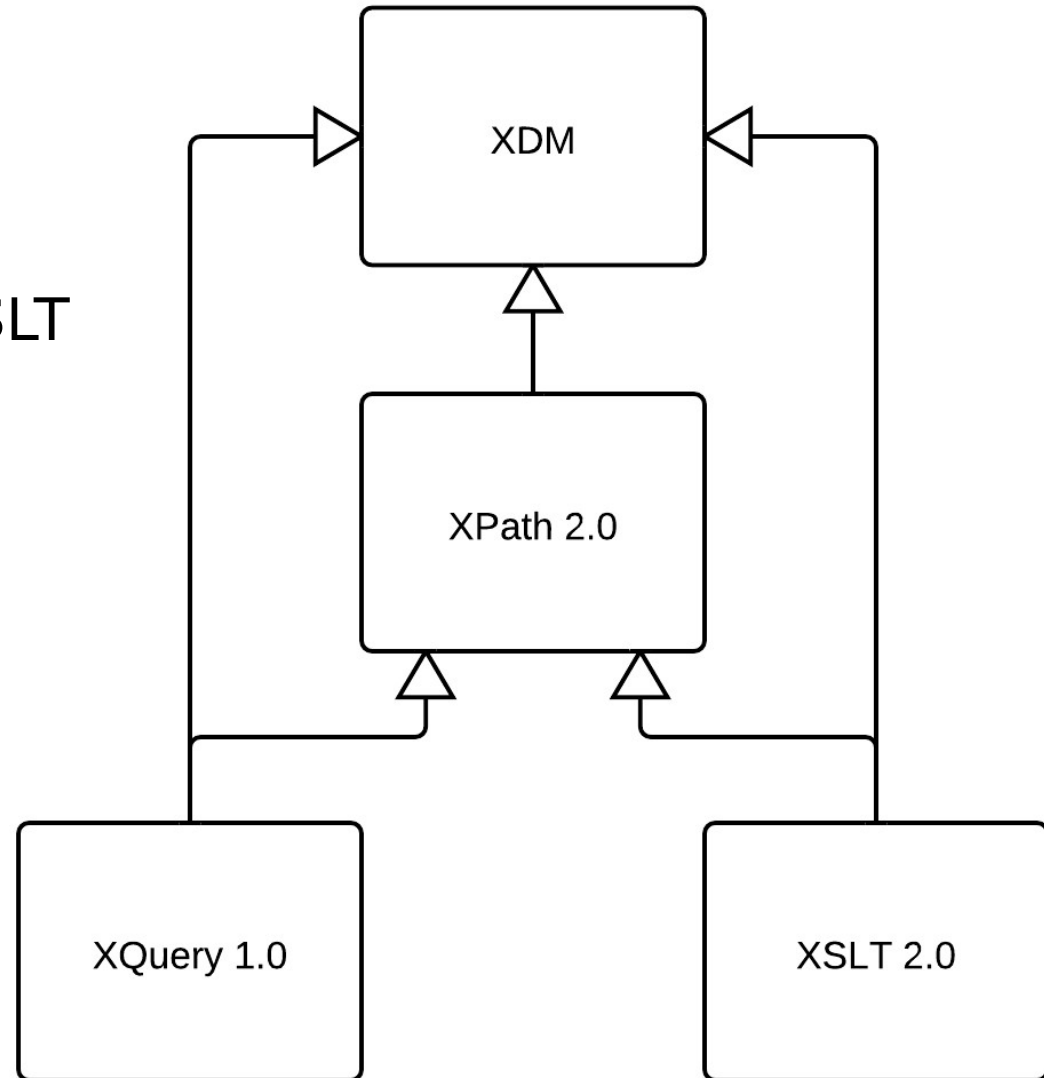


XQuery Design Goals

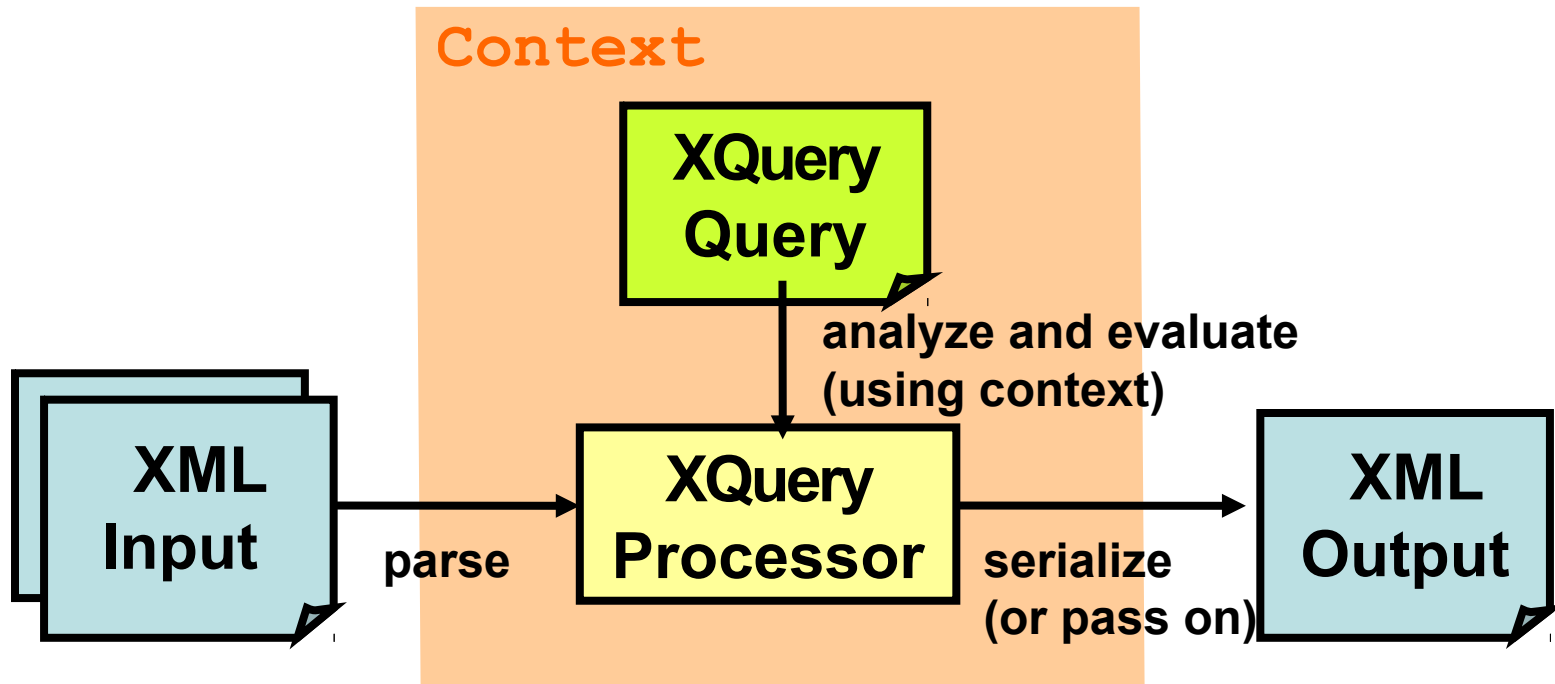
- Queries are concise and easily understood
- Suitable for both structured and unstructured data
- Platform/Protocol agnostic with predictable results
- Declarative rather than Procedural (What vs. How).
 - Strongly typed (optimisation and error detection)
- Able to process collections of documents
-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

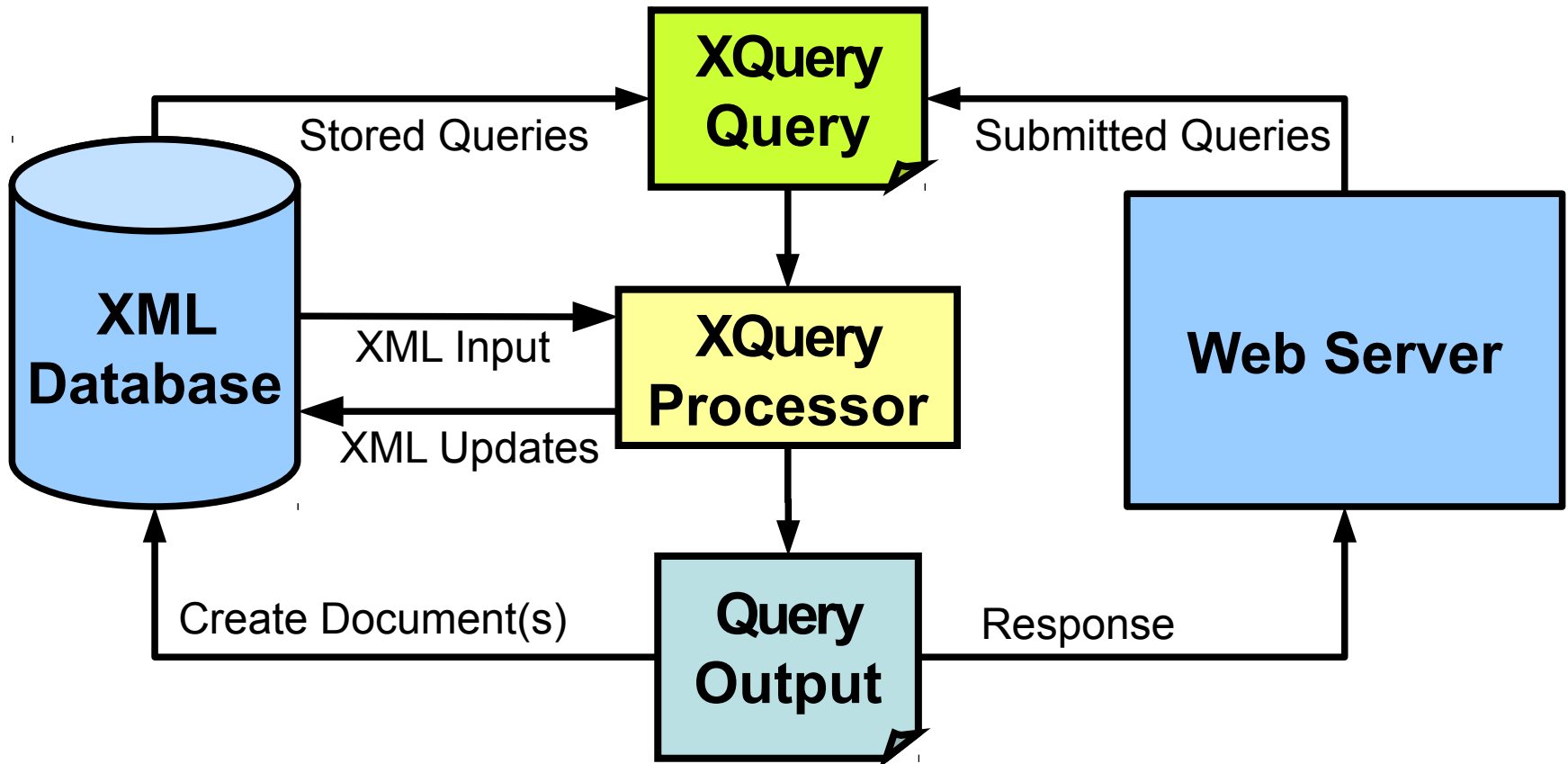


XQuery Processing Model (Simple)



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

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 RDBMS 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	<code><journal></code> <code> <name>Developmental Biology</name></code> <code> <publisher>Elsevier</publisher></code> <code></journal></code>
2	8756-8233	Drugs Soc.	11	<code><journal></code> <code> <name>Drugs and Society</name></code> <code> <publisher>Taylor & Francis</publisher></code> <code></journal></code>

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

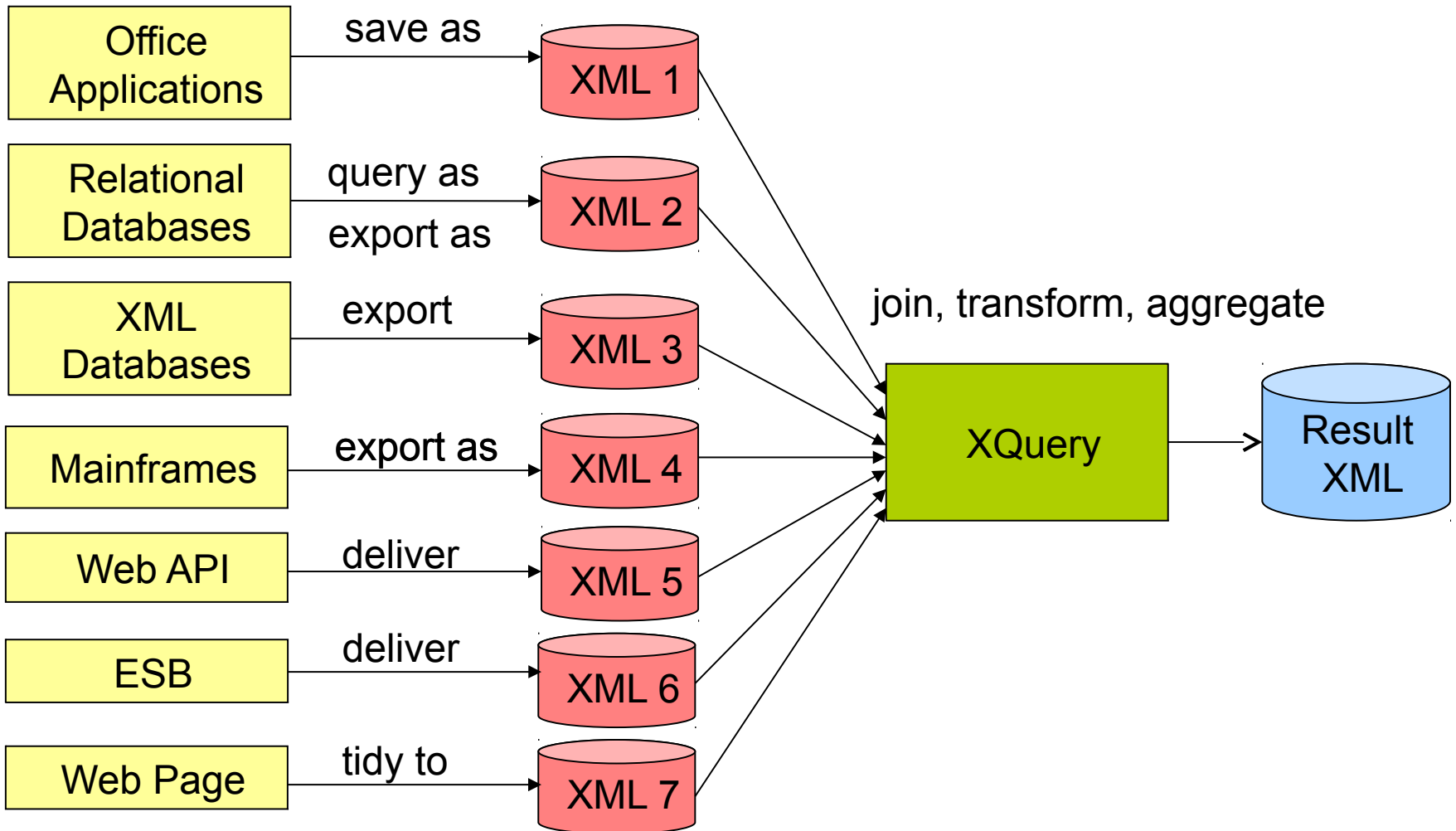


id	vol	name
1	369	<code><name>Developmental Biology</name></code>



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 Public Working Draft: Feb 2001
 - Recommendation: Jan 2007
- Related XML Query Standards
 - XDM (XPath and XQuery Data Model)
 - XQueryX
 - XQuery and XPath Full-Text
 - XQuery Update
 - XQuery Scripting



summer school

XQuery Standards

- XQuery 3.0
 - Originally named 1.1... Marketing!
 - First Public Working Draft: July 2008
 - Recommendation: April 2014
 - Higher Order Functions, Group By, Windowing, Annotations, try/catch, Simple Map Operator
- XQuery 3.1
 - First Public Working Draft: April 2014
 - We *just* want to add: Maps and Arrays.
 - Scope Creep: Invoke/Import XSLT, Dynamic XQuery Module Import, etc.





XQuery Basics

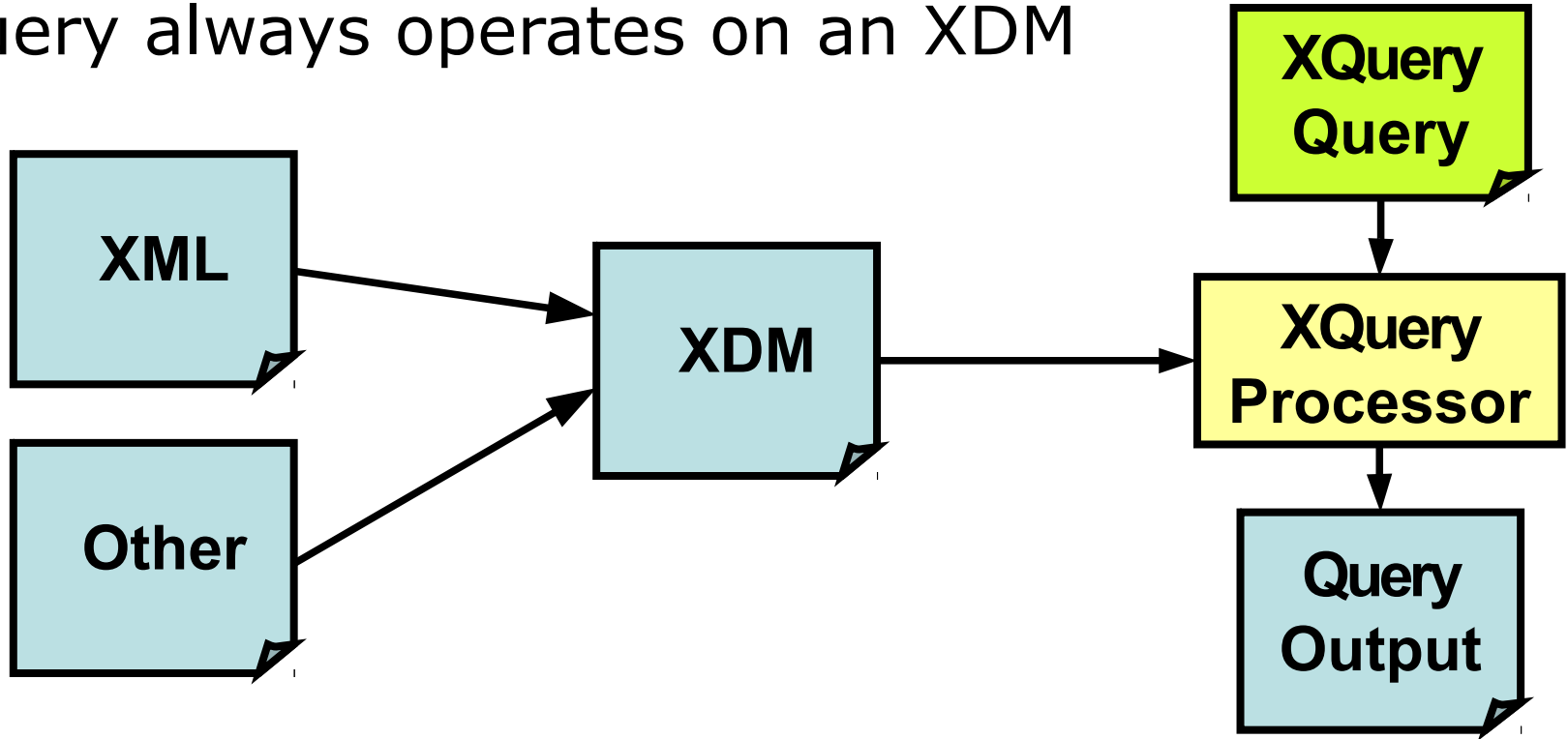
- Aimed at Users Getting Started Quickly
- Some aspects of XQuery we will examine:
 - Data/Type Model
 - XPath 2.0 Expressions
 - FLWOR Expressions



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 (3.0 adds Function Item Type)
- Nodes
 - XML Documents are made of these!
 - Different types of nodes:
 - document, element, attribute, element

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

sing-





XDM - Node Trees

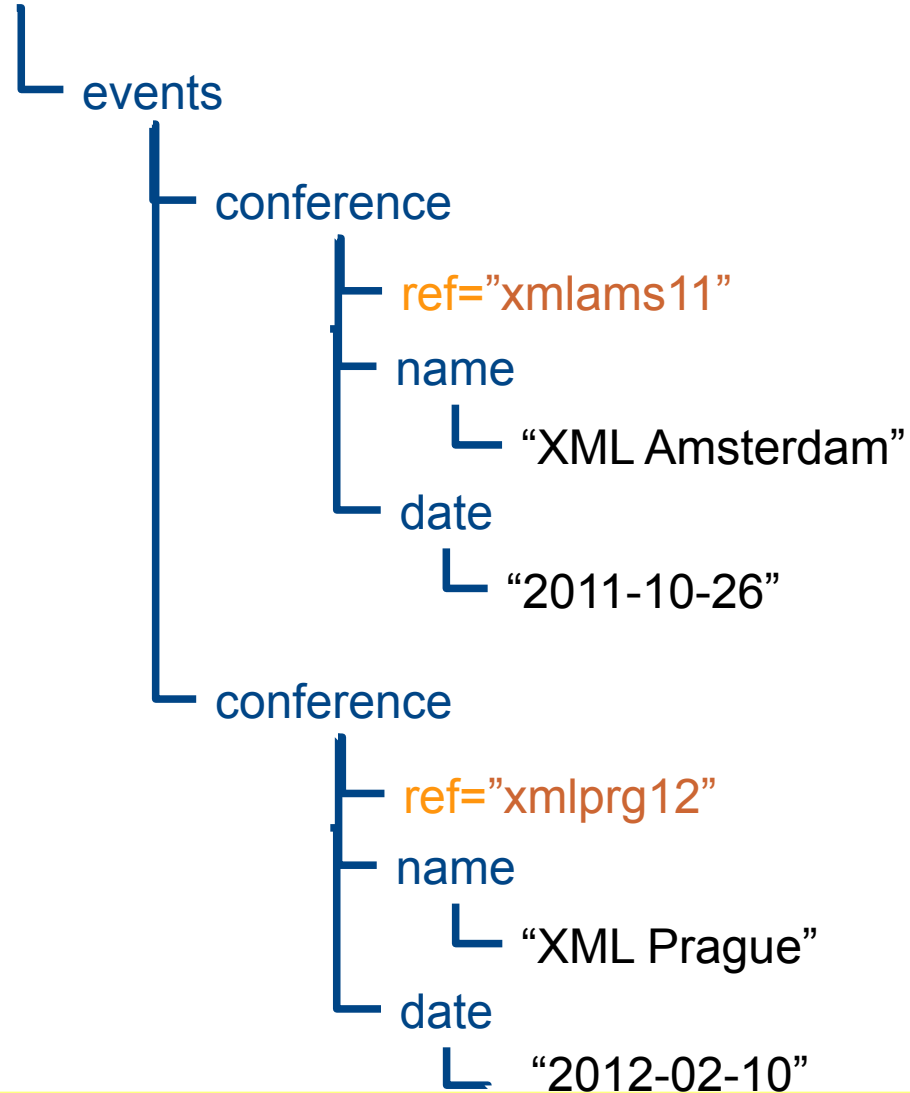
XML: Its a Tree of Nodes!

```

<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>

```

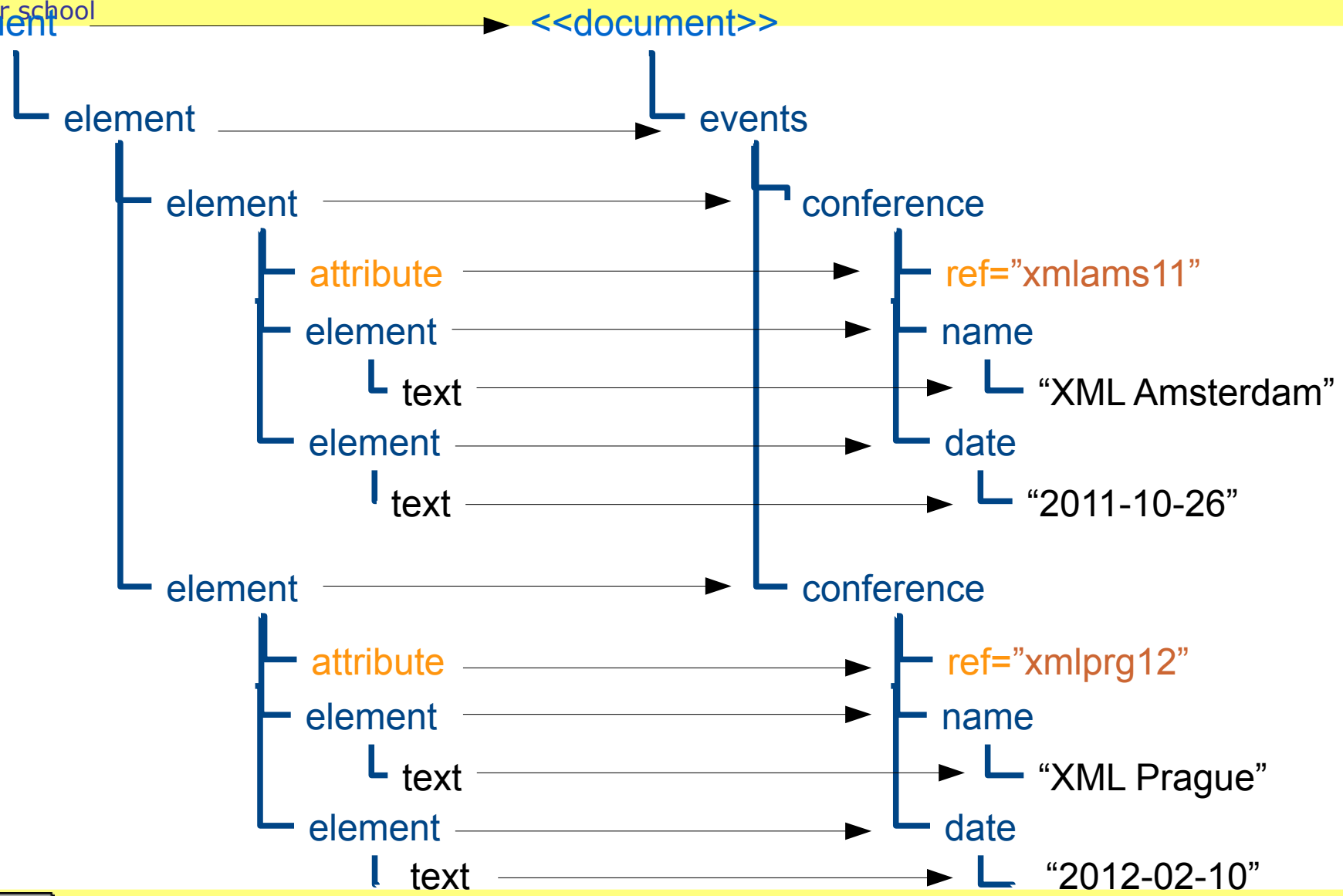
<<document>>





summer school
document

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>

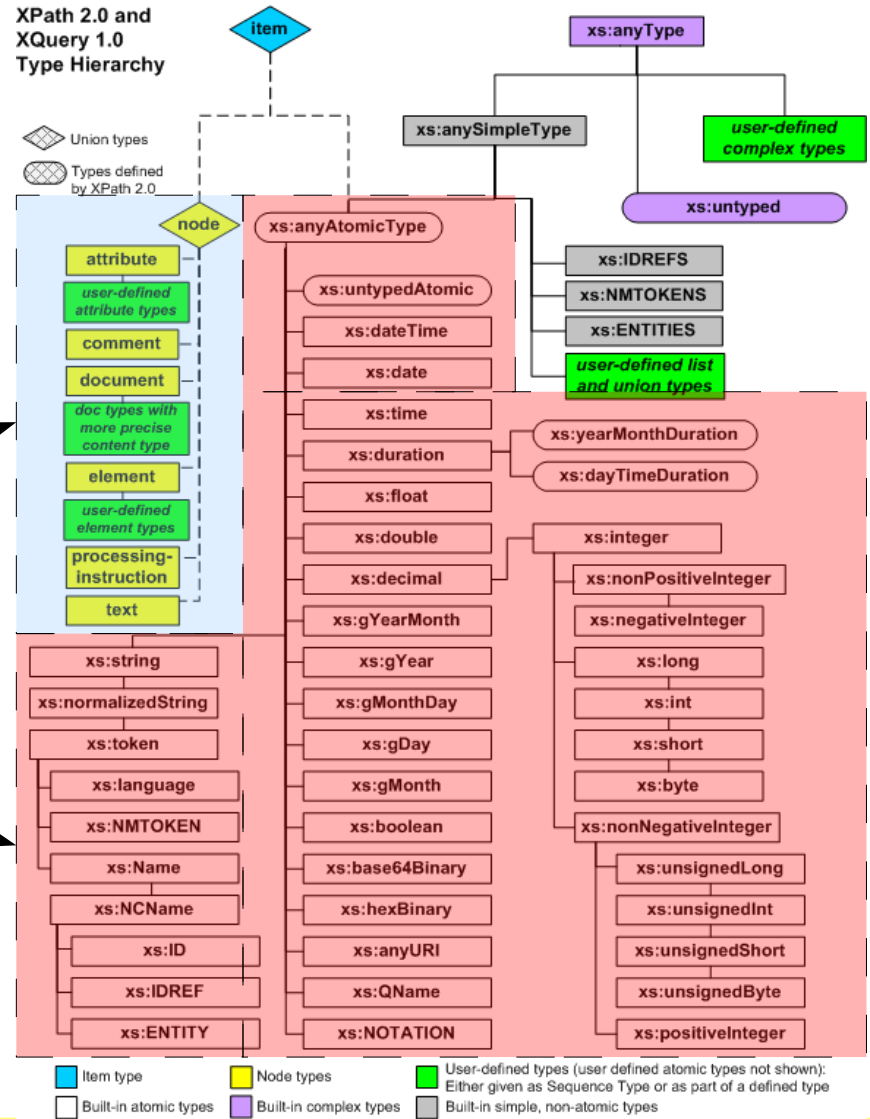
XDM – Type Hierarchy

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)



- 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")

- 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"`

- is ?

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

- is ?

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

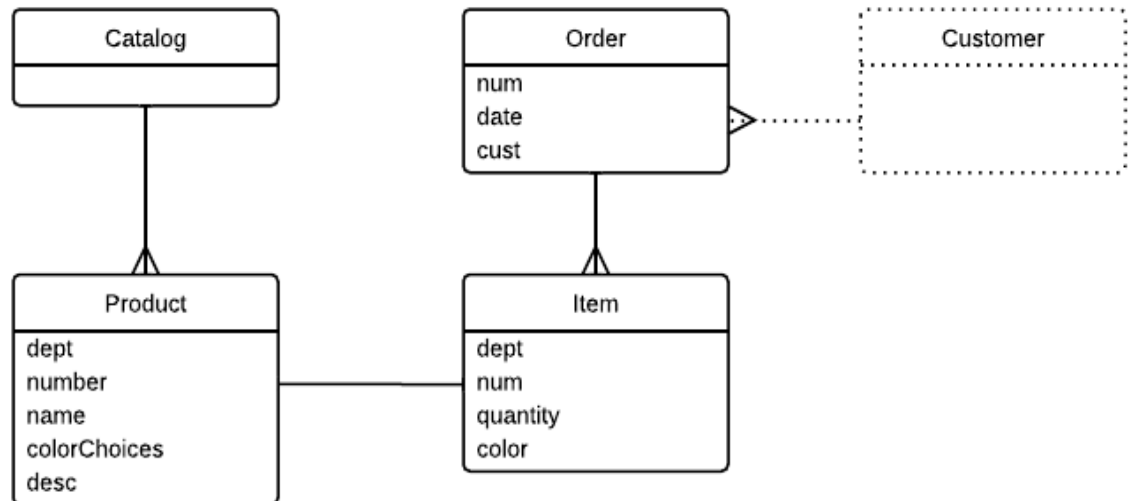
- ...and this??



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

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>
```

```
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)}"/>
```

query

results

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


- You need something to process:
 - External. Set by processor (also external var's!)
 - Request, by explicit function call

- A document

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

- A collection of documents

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

- Or... XQuery generates original data

- `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?
 - URI! A label?
- Typically followed by an expression.

- 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!

- 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.
for
  $product in doc("catalog.xml")/catalog/product
return
  $product/name
2.
```

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" }
}
```

- 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

```
<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
  1. $catalog-age := days-from-duration(
      current-date() - xs:date( $product / @commenced )
  )
return
  2. <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>
```

- 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 Bindings have a limited Scope
- Is this valid?

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


- ...and is this valid?

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



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>
```

- 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  
};
```



- 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
 - There are >220 functions available in XQuery 3.0

- 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)
```

- Zero or 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 More *

```
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, or...
 - Library Module
 - Has a namespace!
 - Just functions
 - Can Import other Libraries
 - .xqm file, or...



Function Modules

- Main Module

```
xquery version "1.0";

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

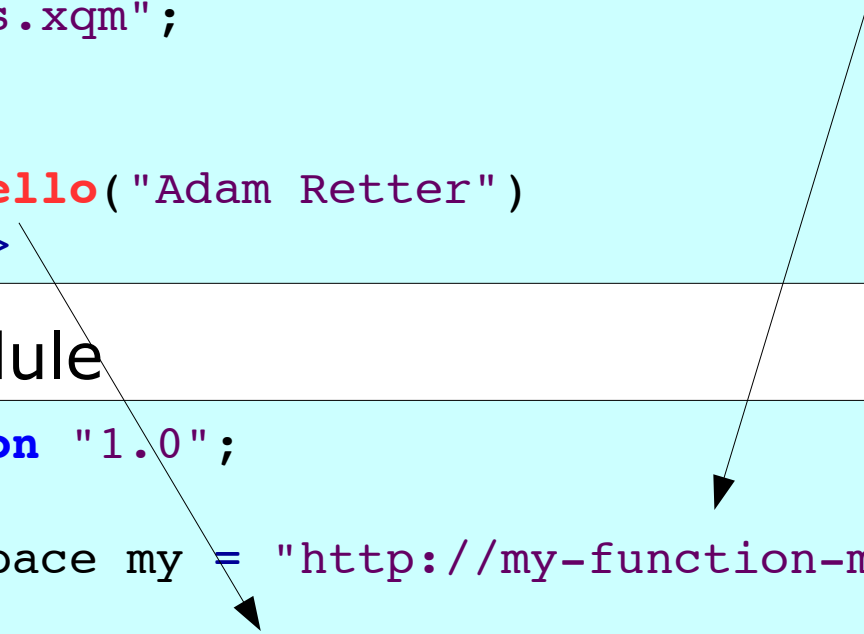
<greetings>{
  my:say-hello("Adam Retter")
}</greetings>
```

- Library Module

```
xquery version "1.0";

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

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





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
 - 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
 - NXDB++
 - Binary content, REST, Web, etc, etc.



- Why not use a File System?
 - How to retrieve?
 - By file-path or some sort of lookup table?
 - i.e. Is a 'Directory' the same as a 'Collection'?
 - Where to keep metadata?
 - How to Query?
 - grep?
 - Integrate a search-engine (full-text), e.g. Apache Solr?
 - No direct XPath access!
 - How to Update?



Why Native XML Database?

- Why not use an RDBMS?
 - XML is just Text?!? (varchar / BLOB / CLOB)
 - Shredding
 - Every set of children is a table. Many tables!
 - Manual vs. Auto.
 - How to Query/Transform/Retrieve doc?
 - Many RDBMS offer XML storage (e.g. XMLType)
 - Oracle shred's behind the scenes, requires XML Schema.
 - Querying is often still driven from SQL
 - Joining XML and non-XML data is hard
 - How to Update? Full-text Search? Aggregate?

- 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

- Marklogic
 - Commercial
 - XQuery 1.0/3.0, XSLT 2.0, XForms 1.1, RESTXQ, Bespoke Full-Text
 - Shared-Nothing Clustering
- eXist
 - Open Source. LGPL v2.1
 - XQuery 3.0, XSLT 2.0, XForms 1.1, RESTXQ, XQuery Update, XProc, EXPath, Bespoke Full-Text, Customisable Extension Modules
 - Master-Slave Replication with Slave promotion.
- BaseX
 - Open Source. BSD License
 - XQuery 3.0, XQuery Update 1.0, RESTXQ, EXPath, XQuery Full-Text 1.0
- Others: Sedna / EMC Documentum xDB / Software AG Tamino / etc...

- Native XML Database. Established in 2000
- Open Source. LGPL. Commercial Support: existsolutions.com
- Hierarchical Collections of Documents
- Supports XML and Binary Documents. WebDAV + REST + RESTXQ
- XQuery 3.0, XQuery Update*, Proprietary Full-Text
- Also: XSLT 2.0, XForms, XProc, XInclude, JSON, XHTML, HTML5
- Full Web App platform with XQuery extensions and EXPath



eXist-db: In Practice

Native XML Databases

How to get setup?

- eXist-db is written in Java
 - You need Oracle/Open JRE 7
- Download and install v2.1 or v2.2-RC2 from
 - <http://exist-db.org/#download>
 - Code: <https://github.com/exist-db/exist>
- Consists of:
 - Database and Web Server
 - Simple GUI Admin Client
 - Web IDE (eXide) and Dashboard



Working with Documents

- Documents can be added to the database by:
 - GUI Admin Client
 - Dashboard
<http://localhost:8080/exist/apps/dashboard>
 - WebDAV
<http://localhost:8080/exist/webdav/db>
 - REST
<http://localhost:8080/exist/rest/db>
 - XQuery (*xml:store*)
 - Java/Scala/Ruby/PHP/.net/APIs: XML-RPC, SOAP, Eclipse Plugin. Etc.



- 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, *fn: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>
```

- 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

eXist-db Guide and Reference

Shameless plug:



<http://shop.oreilly.com/product/0636920026525.do>



Licensed under a Creative Commons
Attribution-Noncommercial-Share Alike 3.0 Unported License

www.xmlsummerschool.com

Slide 89



XML Applications

What is an XML Application?

- Developed entirely with XML Technologies
 - W3C Standards
 - **XML**
 - Storage and Transfer
 - **XPath, XQuery, XSLT, XSL-FO, XProc**
 - 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 (...ETH Zurich)
- 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'ish:
 - XHTML 1.0 and 1.1
 - HTML 5 (XHTML)
 - Vendors: Other Serializers (JSON, HTML4, HTML5, etc)



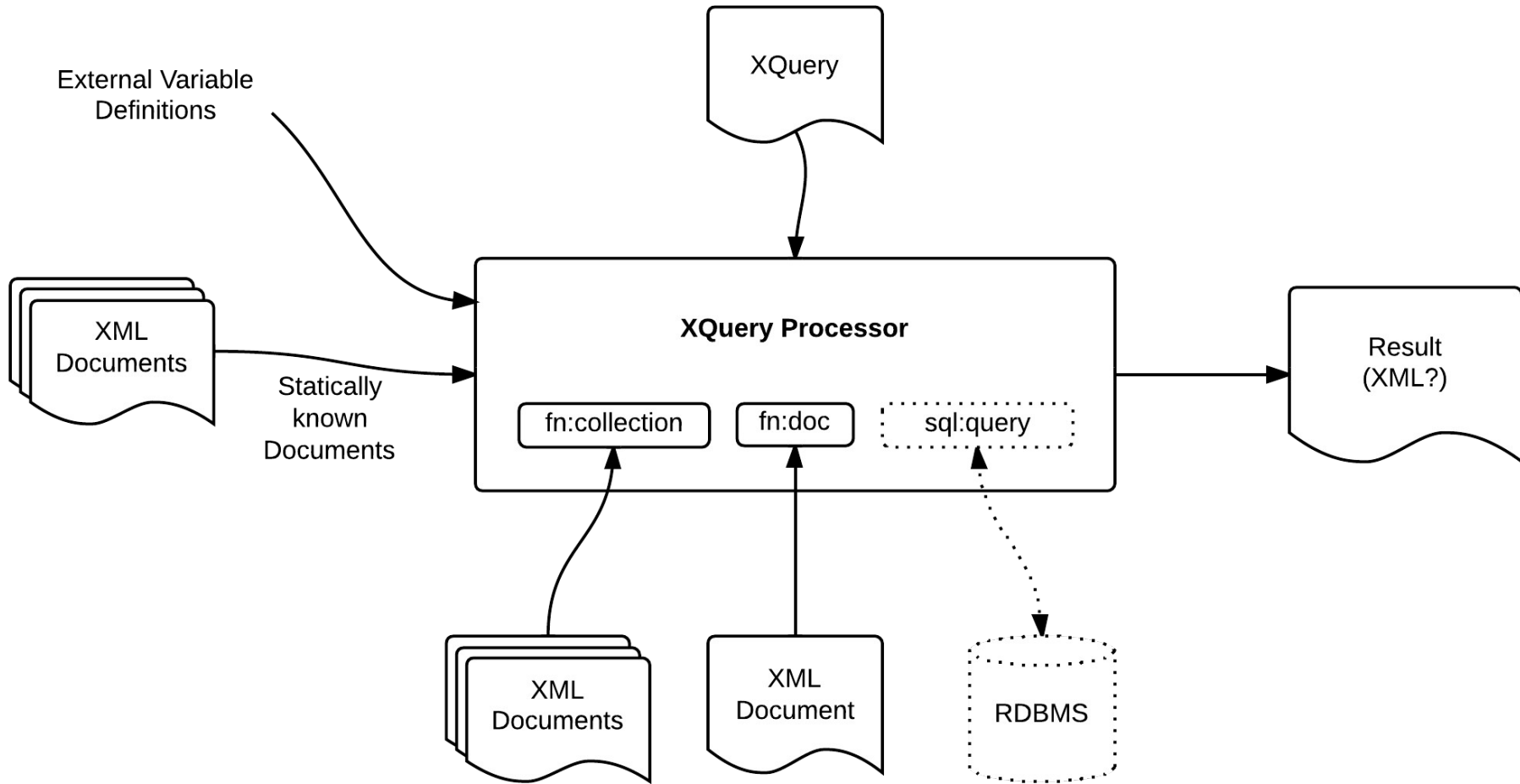
XQuery and the Web

- XQuery as specified by the W3C:
 - Has **no** concept of the Web or HTTP!
 - **Not** designed with Server Side Scripting in-mind!

- ***So... how can we use XQuery with the Web?***

- XQuery can process input from:
 - Static Context
 - Statically known Documents
 - Value of External Variable Declaration
 - Function Call
 - XQuery 1.0 - `fn:doc`, `fn:collection`
 - XQuery 3.0 – `fn:unparsed-text`, `fn:unparsed-text-lines`, etc.
 - Extension function provided by XQuery Processor
 - Many available
 - Varies platform to platform

XQuery Processing



- Two main integration options:
 - Loose: Call XQuery Processor from Web Server
 - Advantage: Easy to do
 - Disadvantage: XQuery Processor has no Web knowledge
 - Tight: XQuery Processor embedded in Web Server
 - Disadvantage: Harder to achieve
 - Advantage: XQuery has some Web knowledge
 - Advantage: Use one of many existing options
 - e.g. Native XML Database or Servlex



HTTP Request/Response with XQuery

summer school

- Tight integration options:
 - 1) Model HTTP Request/Response as XML(s)
 - Request is statically known document
 - Response is result of XQuery
 - 2) Just invoke the main XQuery
 - Provide *extension* functions to the XQuery
 - Functions to access HTTP Request properties
 - Functions to set HTTP Response properties
 - *Body* of Response is result of XQuery
 - 3) Direct XQuery function call
 - Inject HTTP Request (or properties) as function params
 - Response is result of XQuery



- We will now look at example of each of the three approaches from the previous slide:
 - To understand the input and output for the XQuery
 - To understand how the XQuery
 - Processes the input
 - Creates the output



(1) HTTP Request/Response as XML Model

summer school

```
<http:request xmlns:http="http://expath.org/ns/http"
url="http://something.com/thing" method="post">
  <http:header name="Accept" value="application/xhtml+xml"/>
  <http:header name="User-Agent" value="Your Browser"/>
</http:request>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<person>
  <name>John Smith</name>
</person>
```

```
xquery version "1.0";
declare namespace http = "http://expath.org/ns/http";
(
  <http:response status="200">
    <http:header name="Context-Type" value="application/xhtml+xml"/>
  </http:response>
,
  <p>Hello {//name/text()}, I see you are using
    {//http:header[@name eq "User-Agent"]/string(@value)}</p>
)
```



(2) HTTP through Extension Functions

summer school

```
POST /thing HTTP/1.1
Host: http://something.com
Accept: application/xhtml+xml
User-Agent: Your favourite browser
Content-Length: 86
Content-Type: application/xml

<?xml version="1.0" encoding="UTF-8"?>
<person>
  <name>John Smith</name>
</person>
```

```
xquery version "1.0";

import module namespace request = "http://exist-db.org/xquery/request";
import module namespace response = "http://exist-db.org/xquery/response";

let $body := request:get-data(),
    $null := response:set-header("Content-Type" "application/xhtml+xml")
return
  <p>Hello { $body//name/text() }, I see you are using
    { request:get-header("User-Agent") }</p>
```



(3) Direct XQuery Function Call

summer school

```
<http:request xmlns:http="http://expath.org/ns/http"
url="http://something.com/thing" method="post">
  <http:header name="Accept" value="application/xhtml+xml"/>
  <http:header name="User-Agent" value="Your Browser"/>
</http:request>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<person>
  <name>John Smith</name>
</person>
```

```
xquery version "1.0";

module namespace app = "http://my-app";
declare namespace http = "http://expath.org/ns/http";

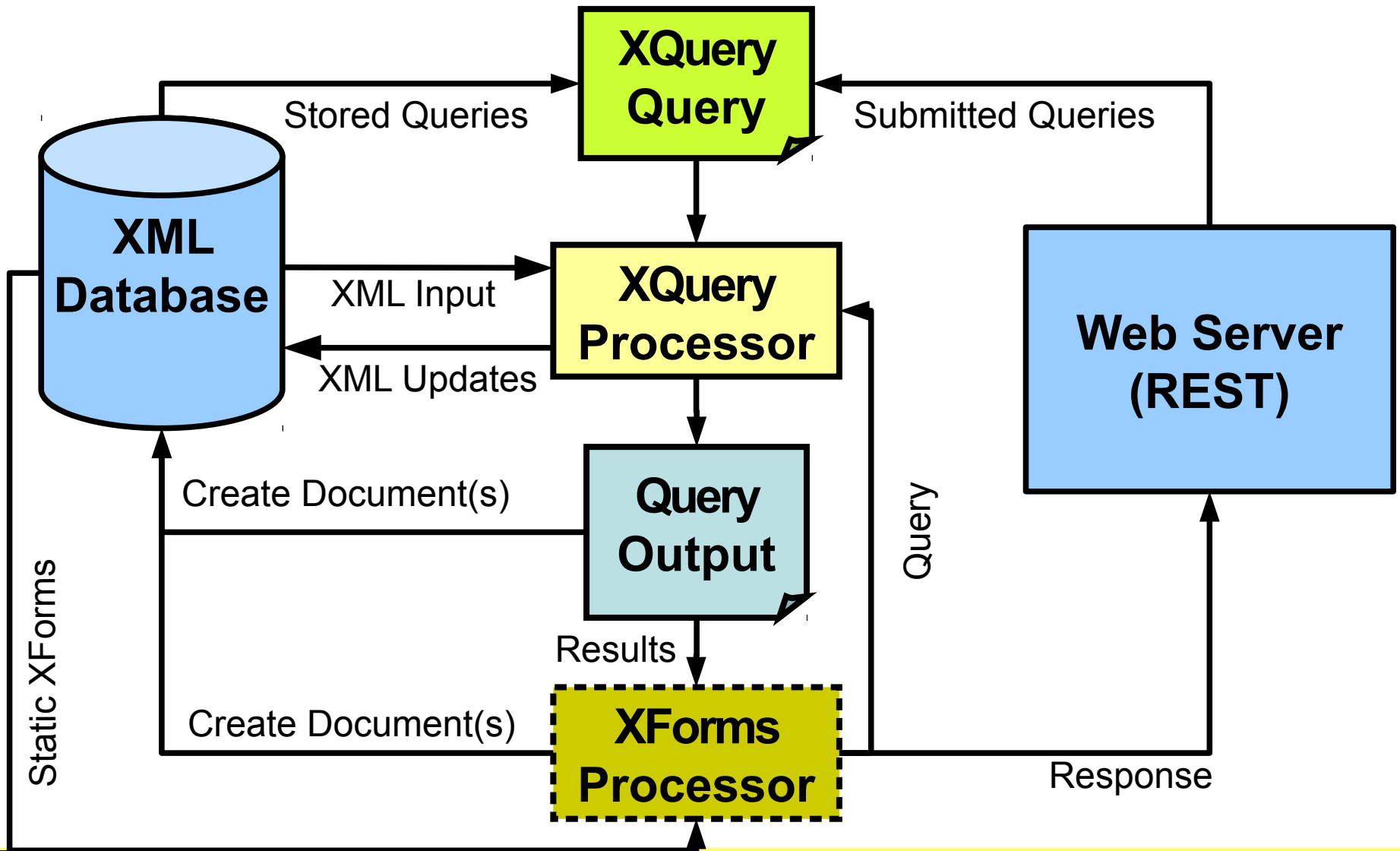
declare function app:some-thing($input as item()+) as item()+ {
  (<http:response status="200">
    <http:header name="Context-Type" value="application/xhtml+xml"/>
  </http:response>
  ,
  <p>Hello { $input//name/text() }, I see you are using
    { $input//http:header[@name eq "User-Agent"]/string(@value) }</p>)
};
```

- So far we have seen:
 - XQuery can be used to generate XHTML
 - Basic Templating with `{code}`
 - XQuery can service HTTP Request/Response
 - Extract and use headers and body from HTTP Request
 - Set headers and body in HTTP Response
 - Requires some sort of **3rd Party** integration!
 - Several possible approaches.

What about XRX?

- XRX (XForms, REST and XQuery)
 - A Client/Server Web Application Architecture
 - 2MVC. MVC in XForms + MVC in XQuery.
 - Zero-Translation architecture. i.e. XML end-to-end
 - **Shallow XRX**: replace XForms with XML consumer (API)
 - XQuery, may be replaced with XSLT or XProc

Classic XRX Architecture



- Exciting things we have yet to consider:
 - HTTP QueryString Parameters
 - Extension Functions or Function Parameter Injection
 - Processing HTML Forms (HTTP QueryString/Body)
 - Extension Functions or Function + Parameter Injection
 - HTTP URI Templates and Routing
 - Web Frameworks!
 - Authentication
 - Serialization – Text, HTML5, JSON, JSON-P, etc.

- Written in XQuery (excl. Servlex[†] and RESTXQ[‡])
- Provide a Higher Level of Abstraction
 - Less glue/boiler-plate for you to write
 - Operate by convention
 - Main Execution / Function + Parameter Injection
- Usually MVC or similar
 - Separation of Concerns
 - Controllers written in XQuery[†]
- URL Routing



XML Web App Frameworks*

	Created	Server Arch.	App Framework	Portability
REST Server	2003	REST + XQY execute	n/a	eXist
HTTP App Server	???	XQ exec.	n/a	MarkLogic
XQuery URL Rewrite	2008	n/a	URL Routing + View Pipelines	eXist
URL Rewrite	2009	n/a	URL Routing	MarkLogic
XQMVC	2009	n/a	MVC	eXist, MarkLogic
Corona	2010	REST	n/a	MarkLogic
REST	2011	REST + XQY execute	n/a	BaseX
REST Endpoint	2011	n/a	URL Routing	MarkLogic
mustache.xq	2011	n/a	View Templating	BaseX, eXist, MarkLogic
Roxy	2012	n/a	mVC	MarkLogic
REST API	2012	REST + XQY execute	n/a	MarkLogic
RESTXQ	2012	REST API	URL Routing View Serialization	BaseX, eXist, MarkLogic
templates.xql	2012	n/a	View Templating	eXist*



(3.1) Direct XQuery Function Call (RESTXQ)

```
POST /thing HTTP/1.1
Host: http://something.com
Accept: application/xhtml+xml
User-Agent: Your favourite browser
Content-Length: 86
Content-Type: application/xml
```

```
<?xml version="1.0" encoding="UTF-8"?>
<person>
  <name>John Smith</name>
</person>
```

```
xquery version "3.0";

import module namespace rest = "http://exquery.org/ns/restxq";
declare namespace output = "http://www.w3.org/2010/xslt-xquery-
serialization";

declare
  %rest:POST("${body}") %rest:path("/thing")
  %rest:header-param("User-Agent", "{$user-agent}")
  %output:method("xhtml") %output:media-type("application/xhtml+xml")
function bc:author($body, $user-agent) {
  <p>Hello {$body//name/text()}, I see you are using {$user-agent}</p>
};
```



- Lots of Server and Framework options available
 - Each takes a different approach.
 - Very few are portable!
- Building an API? Use RESTXQ
 - Can build apps, but extra framework would help
 - mustache.xq is cross-platform (too simple?)
- Platform Specific
 - eXist: RESTXQ or URL Rewrite / betterForm or templates.xq
 - MarkLogic: Roxy / RESTXQ with XSLTForms



Building an XML Application



What are we building?

- Dataset is PubMed
 - 64,089 Articles of Lymphoma / 500 MB 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

```
1 <PubmedArticleSet>
2
3 <PubmedArticle>
4 <MedlineCitation Owner="NLM" Status="In-Process">
5 <PMID Version="1">22905362</PMID>
6 <DateCreated> [4 lines]
11 <Article PubModel="Print">
12 <Journal> [11 lines]
24 <ArticleTitle>Feasibility and acceptability of rapid HIV screening in a labour ward in Togo.</Art
25 <Pagination> [2 lines]
28 <Abstract>
29 <AbstractText Label="BACKGROUND" NlmCategory="BACKGROUND">HIV screening in a labour ward is th
30 <AbstractText Label="FINDINGS" NlmCategory="RESULTS">A cross-sectional survey was conducted in
31 <AbstractText Label="CONCLUSIONS" NlmCategory="CONCLUSIONS">This study is the first to show in
32 </Abstract>
33 <Affiliation>Département des Sciences Biologiques et Santé Publique, Faculté Mixte de Médecine et
34 <AuthorList CompleteYN="Y"> [61 lines]
96 <Language>eng</Language>
97 <PublicationTypeList>
98 <PublicationType>Journal Article</PublicationType>
99 <PublicationType>Research Support, Non-U.S. Gov't</PublicationType>
100 </PublicationTypeList>
101 </Article>
102 <MedlineJournalInfo> [5 lines]
108 <CitationSubset>IM</CitationSubset>
109 <CitationSubset>X</CitationSubset>
110 </MedlineCitation>
111 <PubmedData> [28 lines]
140 </PubmedArticle>
141
```





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)
```

- 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)

- `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!

- 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
 - 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>
```


- 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)
```



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 developers)
 - HTTP call (e.g. Web Browser)
 - Two options:
 - REST Server
 - RESTXQ

- 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>

- 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]
};
```

- 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>
    )
```

- 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

summer school

- 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")`



Step 3: JSON and JavaScript

Building an XML Application

- JavaScript has taken over the Web Browser
- JavaScript (AJAX) can process XML from the Server, but:
 - Most Web developers today want JSON
 - JSON is fine if you don't have Mixed-Content
- Most NXDB can produce JSON:
 - XQJSON: <https://github.com/joewiz/xqjson>
 - `output:method("json")` e.g. RESTXQ or XSLT and XQuery Serialization 3.1

- Just an XQuery Library Module
 - Created by John Snelson / Up: Joe Wicentowski
 - Pure XQuery 3.0 = Platform independent :-)
- Uses a specific XML DSL to describe JSON

```
<json type="object">
  <pair name="article" type="object">
    <pair name="journal" type="object">
      <pair name="title" type="string">World
journal of surgical oncology</pair>
      <pair name="issn" type="string">1477-
7819</pair>
    </pair>
    <pair name="title" type="string">Synchronous
ipsilateral carcinoma of the...</pair>
  </pair>
</json>
```

- Bidirectional lossless conversion
 - XML to JSON: `xqjson:serialize-json($element)`

```
{
  "article":{
    "journal":{
      "title":"World journal of surgical oncology",
      "issn":"1477-7819"
    },
    "title":"Synchronous ipsilateral carcinoma of
the..."
  }
}
```

- JSON to XML: `xqjson:parse-json($json)`



JSON Serialization: eXist

- eXist:

declare option exist:serialize "method=json media-type=application/json"

```
declare namespace exist =  
  "http://exist.sourceforge.net/NS/exist";  
declare namespace json = "http://www.json.org";  
  
declare option exist:serialize  
  "method=json media-type=application/json";  
  
<container>  
  <person>  
    <name>adam</name>  
    <age json:literal="true">33</age>  
  </person>  
</container>
```

```
{  
  "person": {  
    "name": "adam",  
    "age": 33  
  }  
}
```

- Array Construction is determined by the cardinality of element names. Can be tricky!
 - Can control by using `json:array="true"`



summer school

JSON Serialization: eXist

```
declare namespace exist =  
  "http://exist.sourceforge.net/NS/exist";  
declare namespace json = "http://www.json.org";  
  
declare option exist:serialize  
  "method=json media-type=application/json";  
  
<container>  
  <person json:array="true">  
    <name>adam</name>  
    <age json:literal="true">33</age>  
  </person>  
</container>
```

```
{  
  "person": [  
    {  
      "name": "adam",  
      "age": 33  
    }  
  ]  
}
```



JSON Serialization: RESTXQ

- RESTXQ - %output:method("json")
 - adapts XSLT and XQuery Serialization 3.0
 - Per-function serialization
 - JSON format is implementation-defined

```
declare namespace rest =  
    "http://exquery.org/ns/restxq";  
declare namespace output =  
    "http://www.w3.org/2010/xslt-xquery-serialization";  
  
declare  
    %rest:GET  
    %output:method("json")  
function search:by-journal-issn() {  
    <PubmedArticles>  
        <blah>...</blah>  
    </PubmedArticles>  
};
```



- XSLT and XQuery Serialization 3.1
 - Explicitly specifies how to convert XDM types to JSON
 - You're on your own with mixed-content: node -> string value

```
declare namespace output =
  "http://www.w3.org/2010/xslt-xquery-serialization";

declare option output:method "json"

map {
  "person" : map {
    "name": "adam",
    "age": 33
  }
}
```



JSON Serialization: W3C

- Your XQuery looks *like* JSON
 - Serialization of XDM permits Nodes
 - Use XDM 3.1 Array and Map items for control of JSON format

```
declare namespace output =  
"http://www.w3.org/2010/xslt-xquery-serialization";
```

```
declare option output:method "json"
```

```
map {  
  "person" : [  
    {  
      "name": <name>adam</name>,  
      "age": 33  
    }  
  ]  
}
```



Integrating with JavaScript

- Simpler to use a JavaScript data-binding framework e.g. Dojo, Angular.js, Ember.js etc...
- BootStrap + Angular.js TODO:

- Things a Web App should/may have:
 - Templating
 - Unit Tests
 - Wizzy JavaScript?
 - PDF
 - Security
 - API
 - URIs
 - Content Negotiation
 - Integration with 3rd Party sites (also internal SQL etc?)