summer school

13th September 2018

EVOLVED BINARY

XQuery and XML Databases

Adam Retter

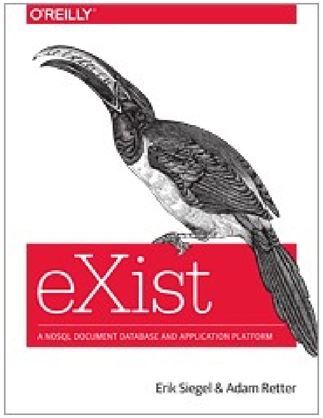
✉ adam@evolvedbinary.com

@adamretter

http://static.adamretter.org.uk/xmlss-18.pdf

Who are you?

- Programmer / Consultant
  - XQuery / XSLT
  - Scala / Java / C++
  - Concurrency
  - Long time tinkerer and XML geek
- Core contributor to eXist-db XML Database (13 yrs.)
- Contributor to Facebook's RocksDB (3 yrs.)
- Creator of “Granite” polyglot database
- W3C XQuery WG Invited expert
- https://www.adamretter.org.uk


O'REILLY
eXist
A NOSQL DOCUMENT DATABASE AND APPLICATION PLATFORM
Erik Siegel & Adam Retter

Today’s Plan

Part 1 (60 minutes)

- Quick Introduction to XQuery
- Lab 1: Our First XQuery
- XDM: XQuery and XPath Data Model
- Lab 2: XQuery and XDM

Break (30 minutes)

Part 2 (60 minutes)

- Introduction to XML Databases
- Choosing an XML Database
- Lab 3: Storing and Querying XML
- XRX / Building XML Web Applications
- Lab 4: Our First XQuery for the Web
- Lab 5: Client/Server Interraction with XQuery
- Lab 6: Full Text Queries with XQuery

summer school

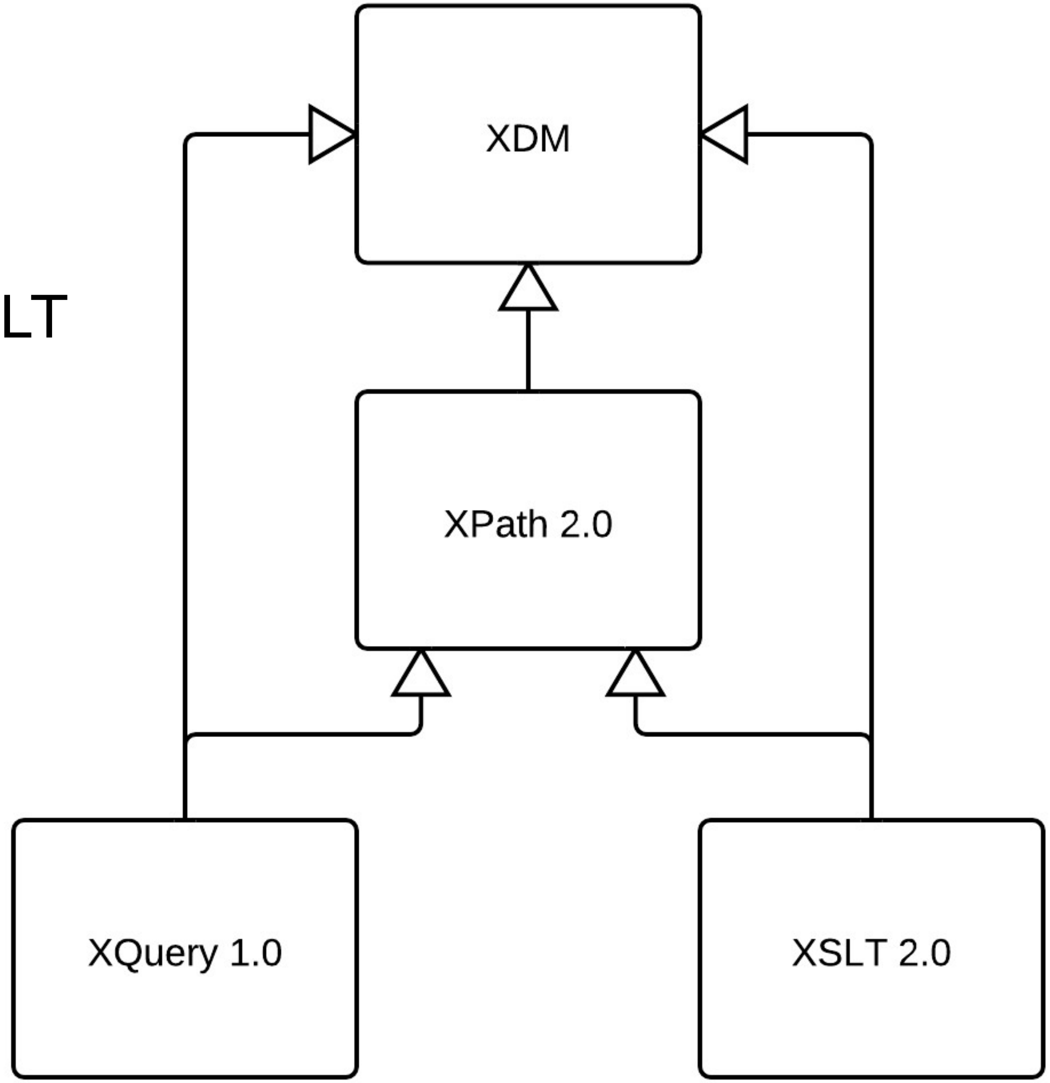
Quick Introduction to XQuery

XQuery is...

- XML Query Language
  - A W3C Standard
  - Superset of XPath
  - 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, JSON, Text, etc.) from one form or structure to another

Where does XQuery fit?

- Its kinda just XPath++
  - If you know XPath...
- Much in common with XSLT
  - XDM and XPath
- XDM / XPath 2.0
  - XQuery 1.0 / XSLT 2.0
- XDM 3.0 / XPath 3.0
  - XQuery 3.0 / XSLT 3.0
- XDM 3.1 / XPath 3.1
  - XQuery 3.1 / ...?



XQuery is also...

- Not Just for Reporting
  - Can update XML documents
  - Can create new XML documents
  - Extensions offer many things – Image Resizing, HTTP, etc.

- An Application Programming Language?
  - Turing Complete
  - Functional Programming (esp. 3.0)
  - XML Data Model Type System (data + code)
  - Suited to the Web

- Easy to learn!

Why use XQuery?

- Why not just use XSLT?
- 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
- More like a (simple) programming language

Lab 1: Our First XQuery

Our First XQuery

- We will use eXist-db
  - *You* have been provided with server access details!

- Open the eXide XQuery IDE in your Web Browser
  - Copy and Paste the Following XQuery:

```
<dates>
    <today>{current-date()}</today>
    <nice>{fn:format-date(current-date(),
            "[FNn], [D1o] [MNn] [Y]")}</nice>
</dates>
```

  - Run the query by pressing the “Eval” button
  - What is the result?

Your First XQuery

- Challenges for you to implement in XQuery:

1) What was the date one week ago?

2) What was the date one month ago?

3) What day was the 4th February 1981?

4) How many days between my talk at last years Summer School and this years Summer School?

Hint: xs:dayTimeDuration() and friends!

Lab 1 Solution for Challenges

1) What was the date one week ago?

```
current-date() - xs:dayTimeDuration("P7D")
```

2) What was the date one month ago?

```
current-date() - xs:yearMonthDuration("P1M")
```

3) What day was the 4th February 1981?

```
fn:format-date(xs:date("1981-02-04"), "[FNn]")
```

4) How many days between my talk at last years Summer School and this years Summer School?

```
xs:date("2018-09-13") - xs:date("2017-09-21")
```

Where is the XML?

- So far we have just run the simplest XQuery!

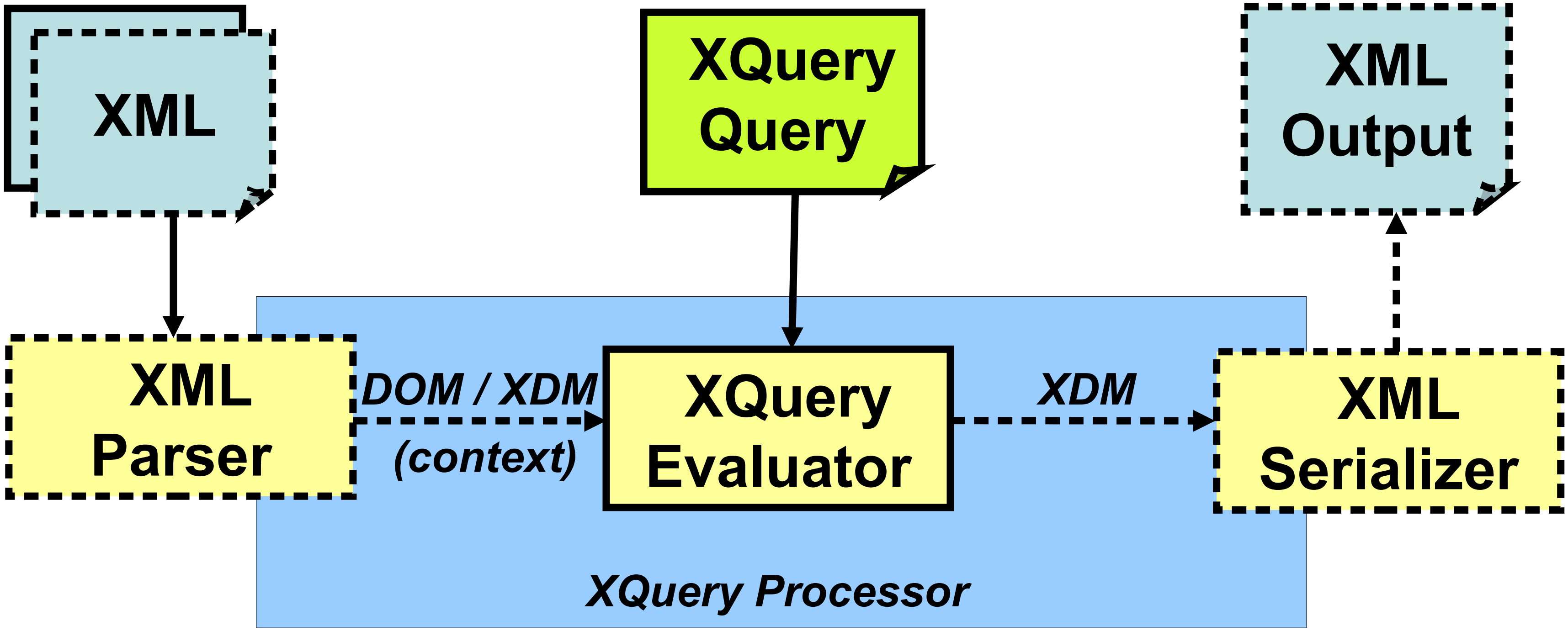
- Yet, we have already encountered:
  - Function Calls
  - Type Constructors
  - Arithmetic
  - (Direct) Element Constructors

- We have not yet given the XQuery Processor any XML to process!
  - i.e.: The Context Sequence was empty.

summer school

XDM: XQuery and XPath Data Model

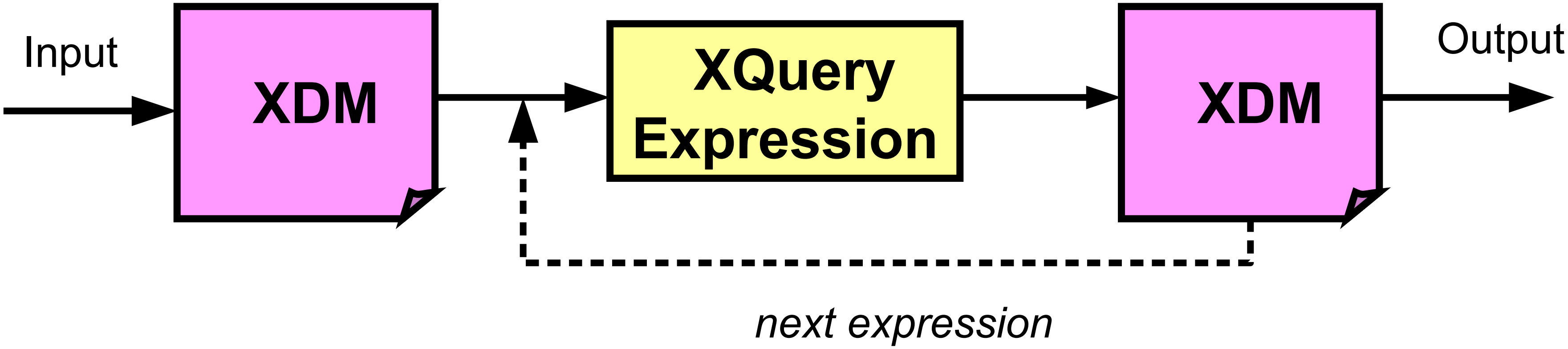
Model of an XQuery Processor



- An XQuery operates on a *Context Sequence (XDM)*
  - e.g. Document(s) from either:
    - Sources bound to the Processor
    - Pulled in during the query (e.g. *fn:doc*(), *fn:collection*(), etc.)

What is XDM?

- XDM (XQuery and XPath Data Model)
- The XDM represents your XML (or...)
- XDM is what XPath and XQuery process!



- Understanding the basics of XDM is key!

XDM Basics

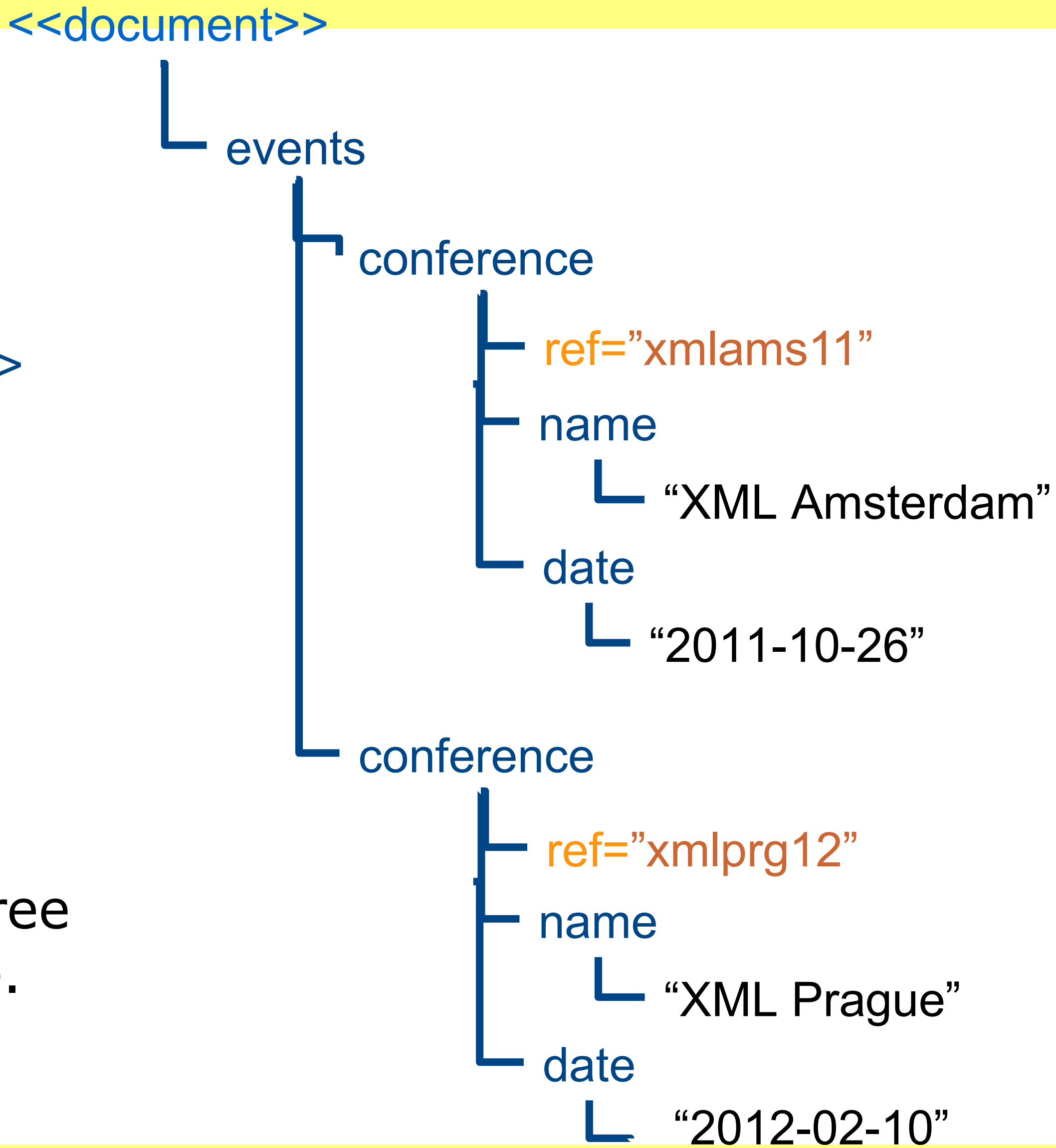
- An XDM consists of Items, and Sequences of Items
  - Builds upon XML Infoset and XML Schema
- There are 3 types of Items:
  - Node types, Atomic types, and Functions types.
- Nodes
  - XML Documents are made of these!
  - Different types of nodes:
    - document, element, attribute, text, comment, processing-element
  - Each has a Unique Identity!

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

XDM - Node Trees

XML is 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>
```

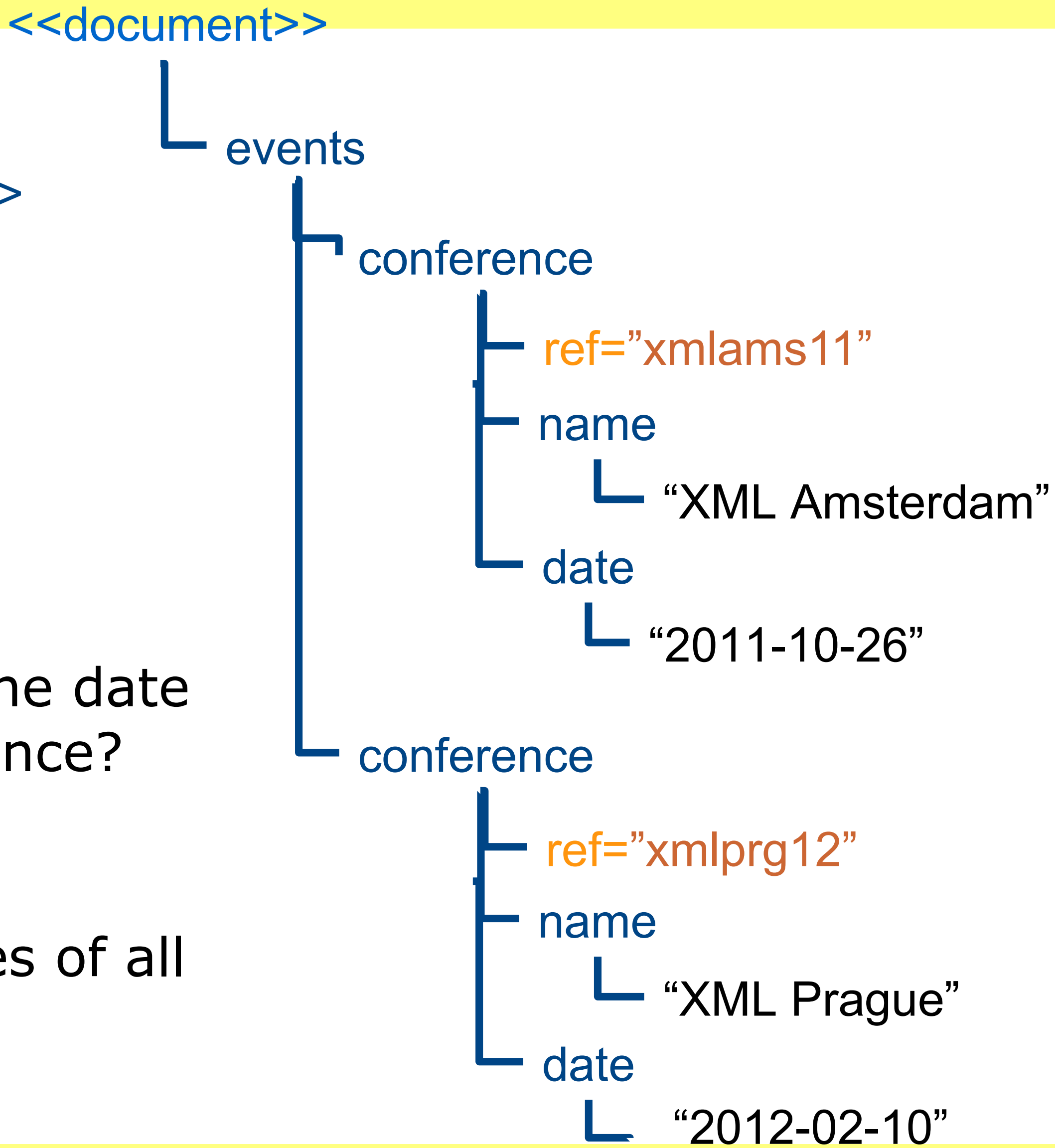


```
<<document>>
└ events
   ├ conference
   │  ├ ref=”xmlams11”
   │  ├ name
   │  │  └ “XML Amsterdam”
   │  └ date
   │     └ “2011-10-26”
   └ conference
      ├ ref=”xmlprg12”
      ├ name
      │  └ “XML Prague”
      └ date
         └ “2012-02-10”
```

We address nodes in the tree using path expressions, i.e. XPath.

XDM - Node Trees

```
<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>
```



Q: What is the XPath for the date of the second conference?

Q: What are the possible XPath(s) for the names of all conferences?

Path Expressions

- Composed of Steps on Axes!
- Four most common: *Child*, *Descendant*, Parent, and *Attribute*
- Child Axis is the simplest:

```
/some/thing
```

  - is just shorthand for:

```
/child::element(some)/child::element(thing)
```

  - Also expressible as:

```
/child::some/child::thing
```

Path Expressions

- Descendant Axis is used for drilling down

-
  ```
  /descendant::thing
  ```

  - is just shorthand for:

    ```
    /descendant::element(thing)
    ```

- Much more convenient is:

  ```
  //thing
  ```

  - Not quite the same as descendant! It is shorthand for:

    ```
    /descendant-or-self::node()/child::thing
    ```

Path Expressions

- Parent Axis is also simple

-
  ```
  /..
  ```
  - is just shorthand for:

    ```
    /parent::element()
    ```

- You can also test/select for the parent by name:

  ```
  /parent::thing
  ```

  - Which is (of course) shorthand for:

    ```
    /parent::element(adam)
    ```

Path Expressions

- Attribute Axis can only be applied to elements

-
```
/@my-attribute
```
  - is just shorthand for:

```
/attribute::my-attribute
```

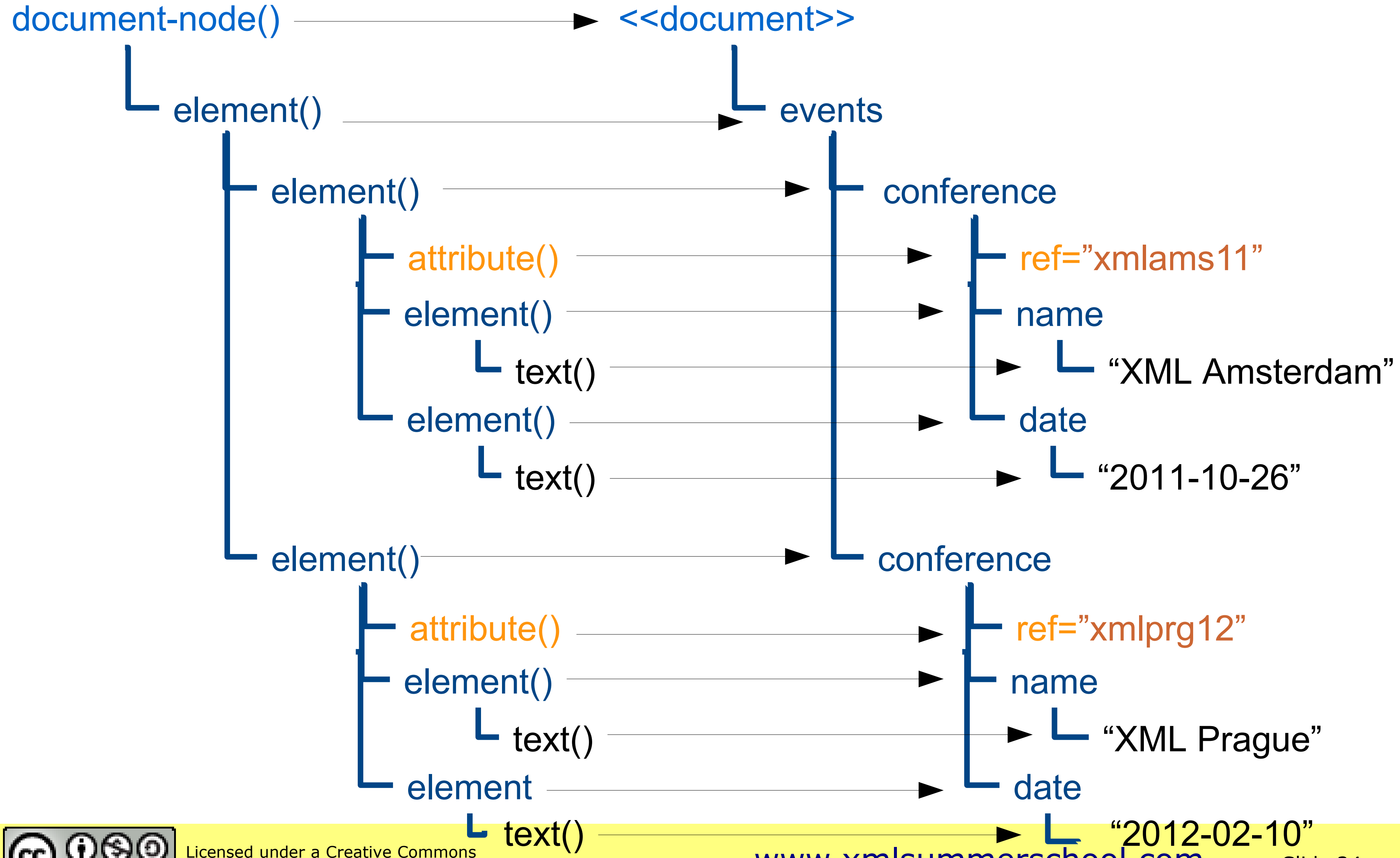
You could alternatively use the node kind test:

```
/attribute(my-attribute)
```

- Attributes are not children! So this will not work:

```
/child::attribute(my-attribute)
```

XDM – Node Trees Types



document-node() → <<document>>

element() → events

element() → conference

attribute() → ref=”xmlams11”

element() → name

text() → “XML Amsterdam”

element() → date

text() → “2011-10-26”

element() → conference

attribute() → ref=”xmlprg12”

element() → name

text() → “XML Prague”

element → date

text() → “2012-02-10”

XDM – Atomic Types

- Atomic Values
  - i.e. A literal value, e.g. “*hello*”
  - These are NOT Nodes!
  - Many different Atomic types:
    - 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”)
      - etc.

- Useful Links:
  - https://www.w3.org/TR/xpath-datamodel-31/#types-hierarchy
  - http://www.w3.org/TR/xmlschema-2/#built-in-datatypes

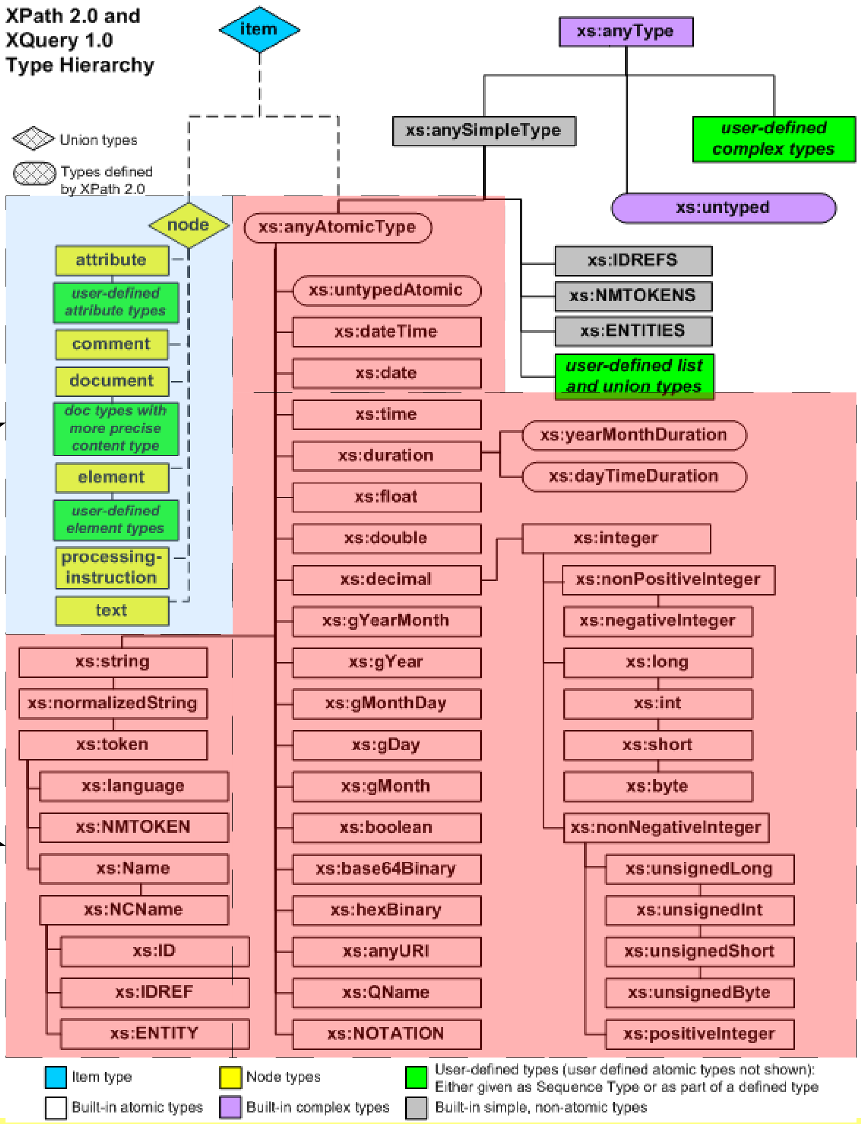
XDM (2nd Ed.) – Type Hierarchy

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)
https://www.w3.org/TR/2010/REC-xpath-datamodel-20101214/

How to determine the type?

- You can use the XPath `*instance of*` expression to test the type:

```
<hello>world</hello> instance of xs:string
```

- You can write a node Kind Test as part of an XPath Expression to select a node:

```
<name><first>world</first></name>/element()/text()
```

XDM - Sequences

- Sequence are returned by Path Expressions and Functions
  - Sequence Constructor starts with '(' and ends with ')'
  - Consists of *Zero or More* Items (in order)

```
("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 identical to => `"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
  - Value Comparisons for Atomic Values
  - General Comparisons for Sequences

	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	<=

- You should use appropriately to enforce intent and avoid bugs...

Lab 2: XQuery and XDM

XML for the Context Sequence

- **NOTE**: For simple queries, we can embed the XML we want to query!

- Context Item as an Element:

```
<person><name>Adam</name></person>
   /name
```

- But… You likely wanted a Document!
  - Wrap using a *Computed Document Constructor*

```
document {<person><name>Adam</name></person>}
   /person/name
```

XDM Nodes Challenge

- Using eXide, write XQuerys to answer the following questions:

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

1) What kind of node is '*fragment2*'?

2) What are the names of the attributes?

3) How many text nodes are in the document?

4) How many nodes are in the document?

5) What is the name of the `*fn*` element's parent?

Hint: Explore some standard XPath functions

Lab 2 Solution for XDM Nodes Challenge

summer school

1) What kind of node is '*fragment2*'?

```
//fragment2 instance of element()

//element(fragment2)/exists(.)
```

2) What are the names of the attributes?

```
//attribute()/local-name(.)

//attribute::*/local-name(.)
```

3) How many text nodes are in the document?

```
count(document { ... }//text())?
```

Lab 2 Solution for XDM Nodes Challenge

4) How many nodes are in the document?

```
count(document { ... }//node())?
```

5) What is the name of the `fn` element’s parent?

```
//fn/local-name(..)

//fn/../local-name(.)

//element()[fn]/local-name(.)
```

With Just Embedded XML

- We have now encountered:
  - XDM Nodes
  - Axes
  - Node Name Tests, and Node Kind Tests
    - Wildcard Tests for Names
  - Functions: *fn:count*, *fn:local-name*, *fn:exists*
  - Sequences (e.g. input to *fn:count*!)
  - A basic Predicate!
  - (Direct) Element Constructors
- We have still not yet given the XQuery Processor any external XML to process!

Comparison Operators – Quiz!

- Do the following evaluate to *true* or *false*?

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

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

```
"mark" eq ()
```

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

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

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

```
("james", "simon", "mark", "bob") != ("hannah", "laura")
```

Lab 2 Comparison Operators – Quiz! Solutions

- Do the following evaluate to *true* or *false*?

`("james", "simon", "mark", "bob") = "mark"`	: true()
`("james", "simon", "mark", "bob") eq "mark"`	XPTY0004
`"mark" eq ()`	: ()
`("james", "simon", "mark", "bob") = ("mark", "james")`	: true()
`("james", "simon", "mark", "bob") = ("mark", "cliff")`	: true()
`("james", "simon", "mark", "bob") != ("mark", "james")`	: true()
`("james", "simon", "mark", "bob") != ("hannah", "laura")`	: true()

Comparison Operators !=

- So, what is going on with *!=* ?

> *“The result of the comparison is true if and only if there is a pair of atomic values, one in the first operand sequence and the other in the second operand sequence, that have the required magnitude relationship.”*

- https://www.w3.org/TR/xpath-31/#id-general-comparisons

```
("a", "b", "a", "b") != ("a", "b")
```

: true()

- So first, "a" != "a" : false()
- But then, "a" != "b" : true()

```
("a", "a", "a") != ("a", "a")
```

: false()

- For, most cases, you probably want *fn:not* instead of *!=*

Lab 2 Subset Challenge

- Using eXide, write an XQuery to determine if sequence B is a subset of sequence A:

```
let $seq-a := ("abigail", "lesley", "faye", "jess")
let $seq-b := ("lesley", "nicola")
return
    (: your code goes here :)
```

Hint: You *could* use an XPath *Quantified Expression*

Lab 2 Subset Challenge – Solution

```
every $b in $seq-b satisfies $seq-a[. eq $b]
```

- Also valid:

  -
    ```
    not(
       (
          for $b in $seq-b
          return $seq-a = $b
       ) = false()
    )
    ```

  -
    ```
    not(($seq-b ! ($seq-a = .)) = false())
    ```

Through Exploring Comparisons

- We have now encountered:
  - General vs. Value Comparisons
  - The unintuitive but consistent *!=* operator
  - Functions: *fn:not* and *fn:false*
  - Quantified Expressions e.g. *some*/*every satisifes*
  - For expression (precursor to FLWOR)
  - Simple map operator

- Next we get into the Database and XML...

Introduction to XML Databases

Why XML Databases?

Why might you need an XML Database?

Why use an XML Database?

- Why a database, 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. Solr / Elastic?
    - No direct XPath access!
  - How to Update?

summer school

Why XML Databases?

What is an XML Database?

What is an XML Database?

“An XML database is a data persistence software system that allows data to be specified, and sometimes stored, in XML format.

These data can then be queried, transformed, exported and returned to a calling system. XML databases are a flavor of document-oriented databases which are in turn a category of NoSQL database (meaning Not (only) SQL).”

-- https://en.wikipedia.org/wiki/XML_database

What is an XML Database?

- More than just a filesystem!
  - Look at how eXist-db stores XML, e.g. dbx files
- Unit of storage is the "Document"
- It ingests (and *may* return) XML documents or Nodes
- Node aware, e.g. search within and across documents
- CRUD operations on document(s)/node(s)
- Some form of query facility/language, e.g. XPath and/or XQuery

What is an XML Database?

- Full-Text capabilities
- Indexes defined for Document queries
- Often defines "*Collection*"s
- May also support non-XML content
  - e.g. Key/Value, Tabular, JSON, Binary, Graph etc.
- Single or Multi-user: Embedded Library or Client/Server

Types of XML Database

- XML Enabled Database
  - Existing database product which added support for XML
  - Predominant Data Model and purpose is NOT XML
  - Heterogenous data models
    - Useful with small amounts of XML as part of a larger non-XML dataset.

- Native XML Database (NXDB)
  - Designed for XML storage/retrieval/query from the start
  - Primary concern and data model is hierarchical (tree)
  - Highly optimised for XML storage and query
    - Typically used when the majority (or all) of the data is XML

- Polyglot Persistence - i.e. 'Use the Right Tool for the Job'

XML Enabled/Hybrid 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 RDBMS, IBM DB2, SQL Server

– Open Source: PostgreSQL

DB2 Example – XMLType and SQL

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>

Native XML Databases

- Reasons not to use an RDBMS
  - XML is **NOT** "*just text*"! (varchar / BLOB / CLOB)
  - Shredding
    - Every set of children is a table. Many *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?

summer school

Choosing an XML Database

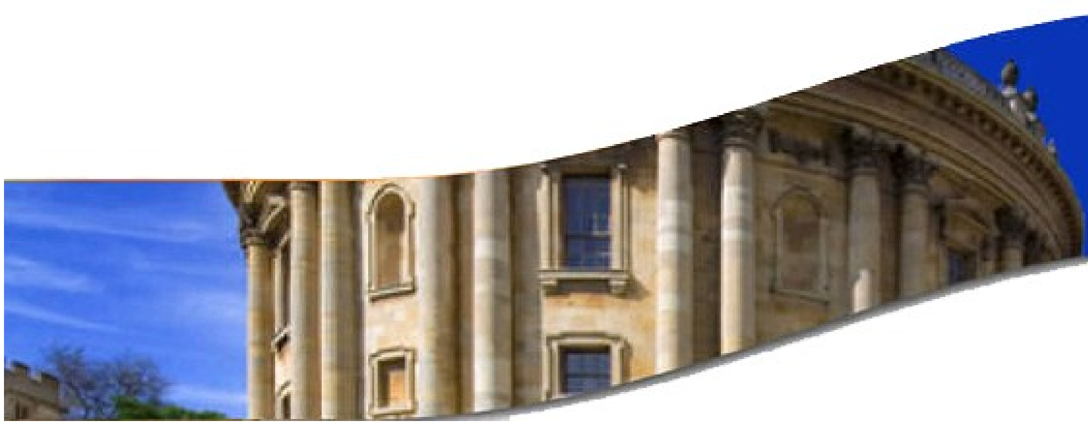
What to look for?

- Open Source vs. Commercial. Total Cost?
- Features
  - XQuery / XSLT / XForms / XProc / JavaScript / JSONiq
  - Indexes, Full Text Search facilities, and Update facilities
  - REST / XML-RPC / WebDAV / SOAP / Language Integration
- Performance
  - Test, Test, Test! Highly dependent on your data and queries
- Scalability
- Support
  - Self / Community / Paid / 3rd Party / Combination

Popular Native XML Databases

- BaseX
  - Open Source. BSD License. Commercial support available
  - XQuery 3.1*, XSLT 1.0/3.0, XQuery Update 1.0, RESTXQ, EXPath, XQuery Full-Text 1.0
  - Java API
- eXist-db (Used in this lecture and labs)
  - Open Source. LGPL v2.1. Commercial support available
  - XQuery 3.1*, XSLT 3.0, XQuery Update, RESTXQ, EXPath, Bespoke Full-Text, XProc, XForms 1.1, Customisable Extension Modules.
  - Java, Python, Scala APIs
  - Master-Slave Replication with Slave promotion.
- MarkLogic
  - Proprietary. Commercial
  - XQuery 1.0/3.0*, XSLT 2.0, Bespoke Update, Bespoke Full-Text, XForms 1.1
  - Shared-Nothing Clustering

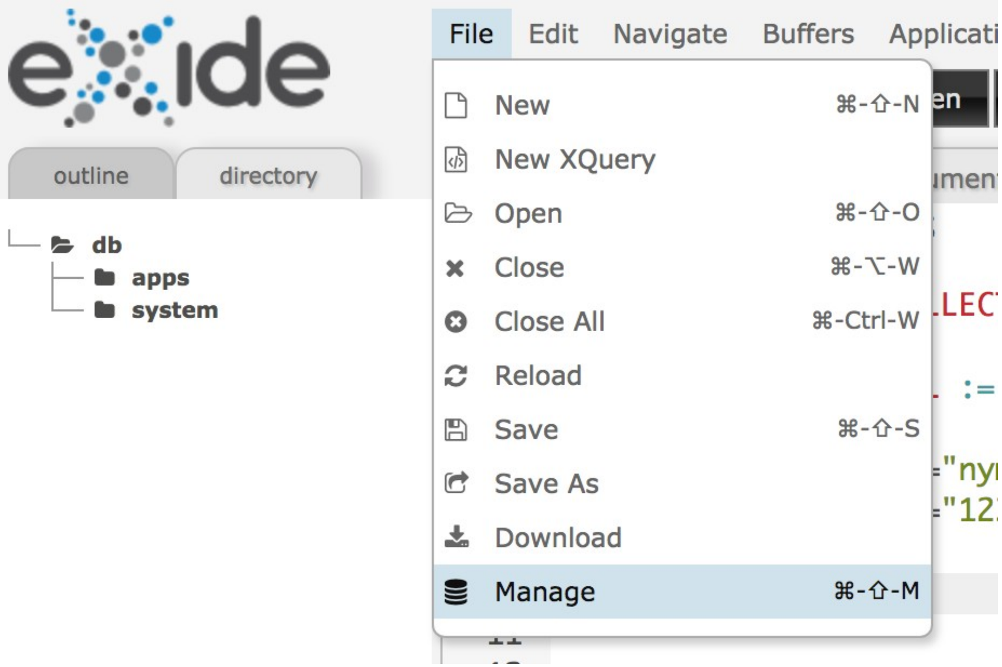
summer school



Lab 3: Storing and Querying XML

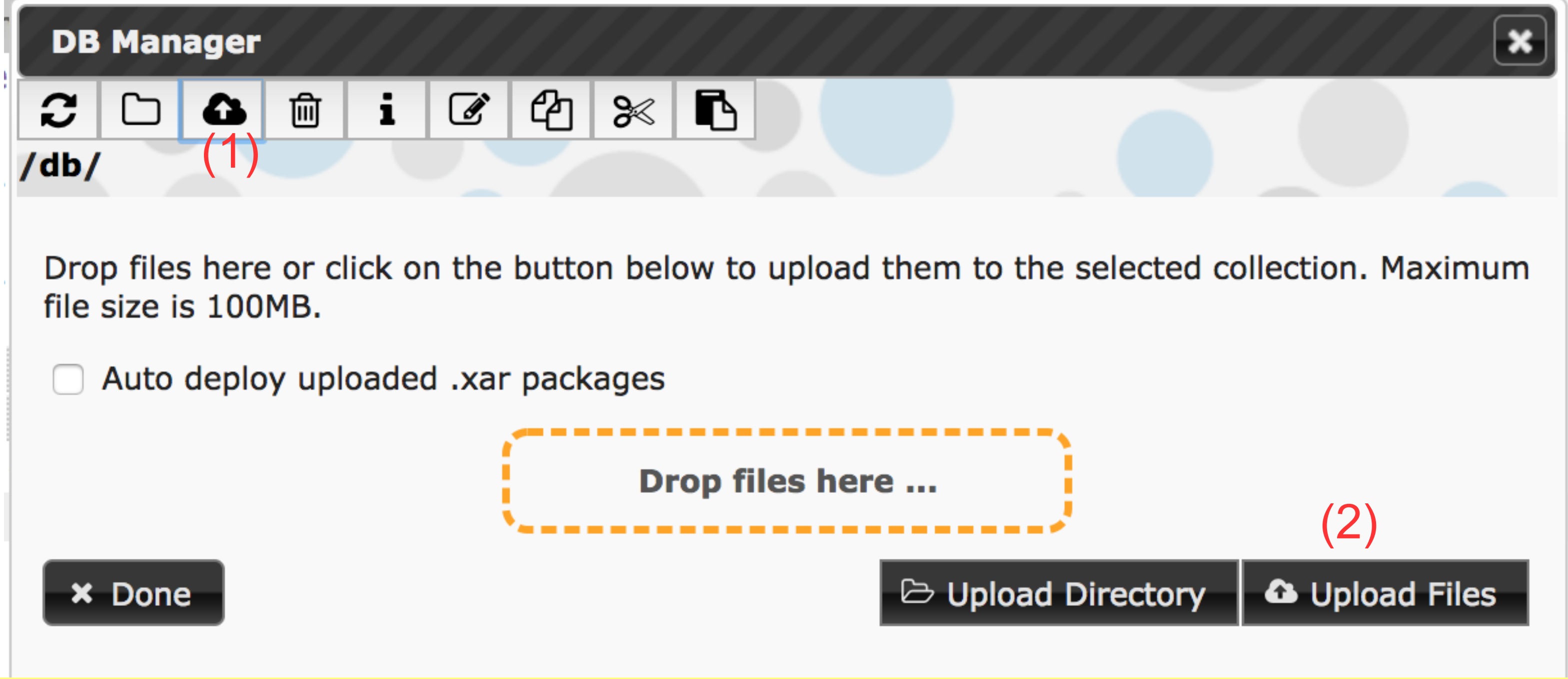
Storing your first XML Document into eXist-db

- First we will store a single document into eXist-db using eXide
  - Download this file to your computer: http://static.adamretter.org.uk/hindawi-example.xml
  - From the File Menu in eXide, enter the DB Manager:



Storing your first XML Document into eXist-db

1) Click the Upload Cloud icon
2) Click the "Upload Files" button
3) Choose the *hindawi-example.xml* file from your Computer
4) Click Close

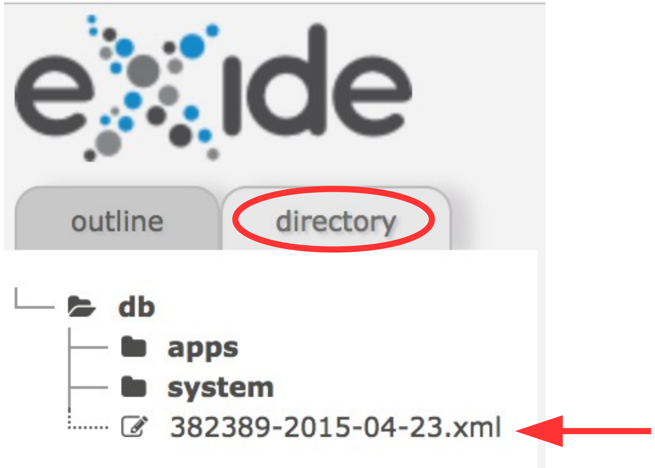


Drop files here or click on the button below to upload them to the selected collection. Maximum file size is 100MB.

□ Auto deploy uploaded .xar packages

Storing your first XML Document into eXist-db

1) Refresh the Directory side-panel in eXide



2) Click the */db/hindawi-example.xml document to open it from the database*

3) Spend a couple of minutes examining the XML and understanding the Document representation.

Lab 3 First FLWOR

- Using eXide, write an XQuery for the Journal Article
  - Query */db/hindawi-example.xml*
  - Produce an XML report of the authors names, similar to:

```
<authors>
    <author>Priscilla Walmsley</author>
    <author>Adam Retter</author>
    <author>Michael Kay</author>
    <author>Florent Georges</author>
</authors>
```

Hint: XQuery components you *could* use include:

- *For Binding*
- *Predicate* or *Where Clause*
- *fn:concat* function

Lab 3 First FLWOR – Solution 1

XQuery:

```
<authors>
{
for $contrib in //contrib-group/contrib
where $contrib/@contrib-type eq "author"
return
    <author>{concat($contrib/name/given-names, " ", $contrib/name/surname)}</author>
}
</authors>
```

Produces the report:

```
<authors>
    <author>Bo Zhang</author>
    <author>Wenxu Xie</author>
    <author>Yong Xiang</author>
</authors>
```

Lab 3 First FLWOR – Solution 1 details

```
<authors>
{
for $contrib in //contrib-group/contrib
where $contrib/@contrib-type eq "author"
return
   <author>{concat($contrib/name/given-names, " ", $contrib/name/surname)}</author>
}
</authors>
```

1) For Binding: **for** *each* **contrib** *child element present* **in** *the* //**contrib-group** *element(s), bind each in turn to the variable* **$contrib**

2) Where Clause: *but only* **where** *each* **contrib** *element (referenced by* **$contrib***) has a* ***contrib-type*** *attribute with the value equal to* "author"

3) Return Clause: *for each thing we have bound, evaluate the next expression*

4) Concat Function: *concatenates one or more strings together into a single string*

summer school

Lab 3 First FLWOR – Improved, Solution 2

```
<authors>
{
for $name in
  doc("/db/hindawi-example.xml")//contrib-group/contrib[@contrib-type eq "author"]/name
return
    <author>{$name/given-names || " " || $name/surname}</author>
}
</authors>
```

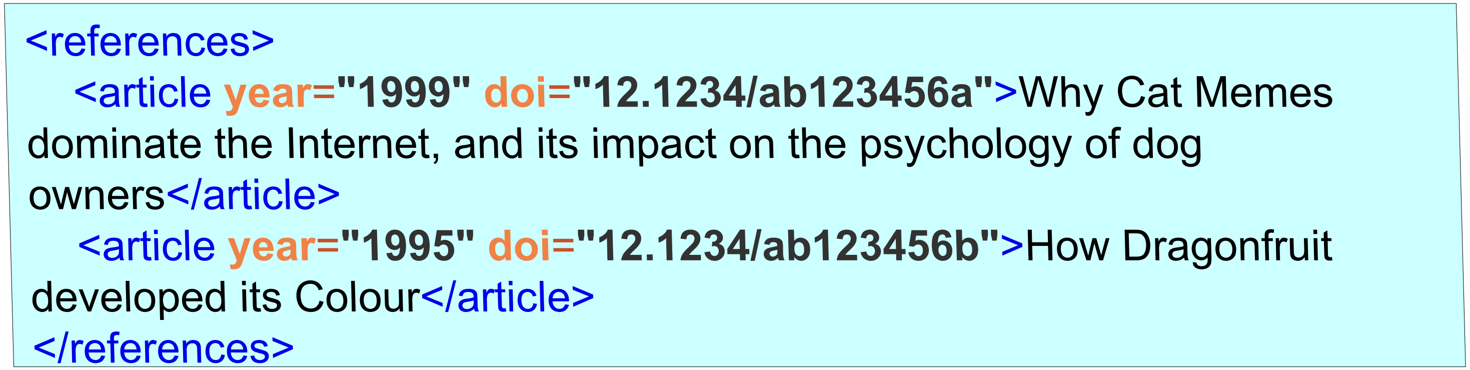
1) Doc function: *retrieves a document using the URI ("/db/hindawi-example.xml") specified, and typically returns it's document-node()*

2) Predicate: *Only where the context item (***contrib** *element)* has *a* ***contrib-type*** *attribute with the value equal to* "author"

3) String Concatenation Expression: syntactic sugar for *fn:concat(a, b)*

Lab 3 Full FLWOR

- Using eXide, write an XQuery for the Journal Article
  - Query */db/hindawi-example.xml*
  - Produce an XML report of the references to articles ordered by year (most recent first) and title:



```
<references>
    <article year="1999" doi="12.1234/ab123456a">Why Cat Memes dominate the Internet, and its impact on the psychology of dog owners</article>
    <article year="1995" doi="12.1234/ab123456b">How Dragonfruit developed its Colour</article>
</references>
```

Hint: XQuery components you *could* use include:

- *Let Binding*
- *Order By Clause*
- *Text Node Kind Test*
- *xs:int type constructor*

Lab 3 Full FLWOR - Solution

XQuery:

```
<references>
{
for $cite in doc("/db/hindawi-example.xml")//ref[@content-type eq "article"]/nlm-citation
let $year := if($cite/year) then xs:int($cite/year) else 0
order by $year descending, $cite/article-title ascending
return
    <article year="{$year}" doi="{$cite/pub-id[@pub-id-type eq "doi"]}">
    {
        $cite/article-title/text()
    }
    </article>
}
</references>
```

Lab 3 FLWOR Join

- Using eXide, write an XQuery:
  - Query */db/hindawi-example.xml*
  - From the articles authors, produce an XML report of the references which were also likely authored by the same people. Similar to:

```
<article year="2014" title="Key Observations of Retterpotomus Social Interactions">
    <authors>
        <author surname="Retter" given-names="Adam"/>
    </authors>
    <self-refs>
        <article year="2011" title="Upon the Discovery of The Retterpotomus">
            <author surname="Retter" given-names="A."/>
        </article>
    </self-refs>
</article>
```

Lab 3 FLWOR Join - Solution

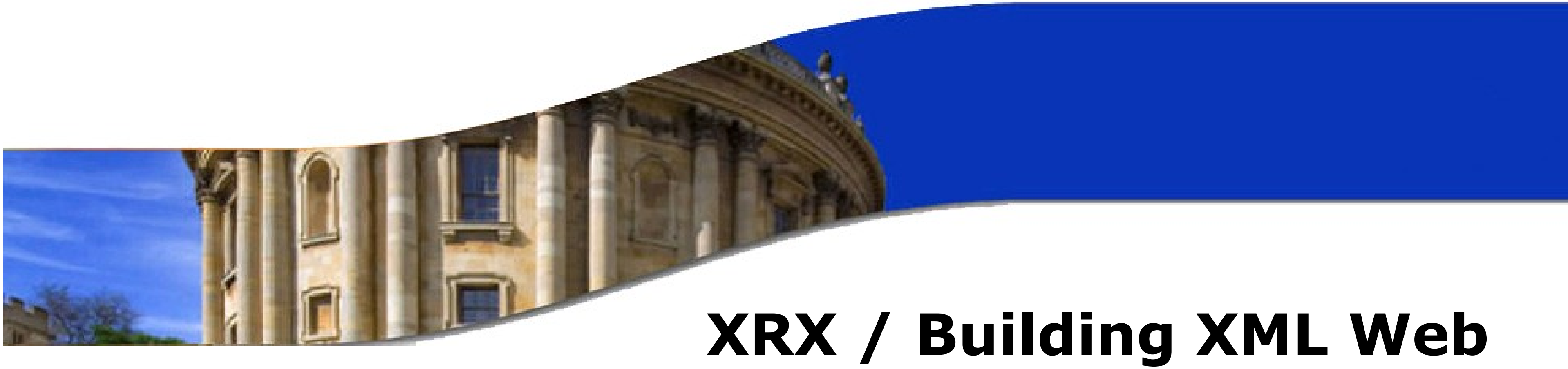
- XQuery:

```
let $article := doc("/db/hindawi-example.xml")/article
let $authors := $article//contrib-group/contrib[@contrib-type eq "author"]/name
return
    <article year="{$article//pub-date[@pub-type eq "publication-year"]/year}"
      title="{$article//article-meta//article-title}">
    { $authors ! <author surname="{surname}" given-names="{given-names}"/> }
    <self-refs>
    {
      for $cite in
        $article//ref[@content-type eq "article"]
          /nlm-citation[.//surname = $authors/surname]
      return
        <article year="{$cite/year}" title="{$cite/article-title/text()}">
        { $cite/person-group[@person-group-type eq "author"]/name !
            <author surname="{surname}" given-names="{given-names}"/> }
        </article>
    }
    </self-refs>
  </article>
```

FLWOR’ing the Database

- We have now encountered:
  - Explicitly providing the Context Sequence with the function *fn:doc*
  - FLWOR components:
    - For Bindings – more powerful than XPath’s simple For expr.
    - Let Bindings – to bind variables to values
    - Where Clause – constraints. Also compared to Predicates
    - Order By Clause - multiple key, ascending/descending
    - Return Clause – Operating on tuples!
  - If/then/else expressions – like any other returns a sequence.
  - String Concatenation. Function vs. expression.
  - Joining queries on sequences

summer school

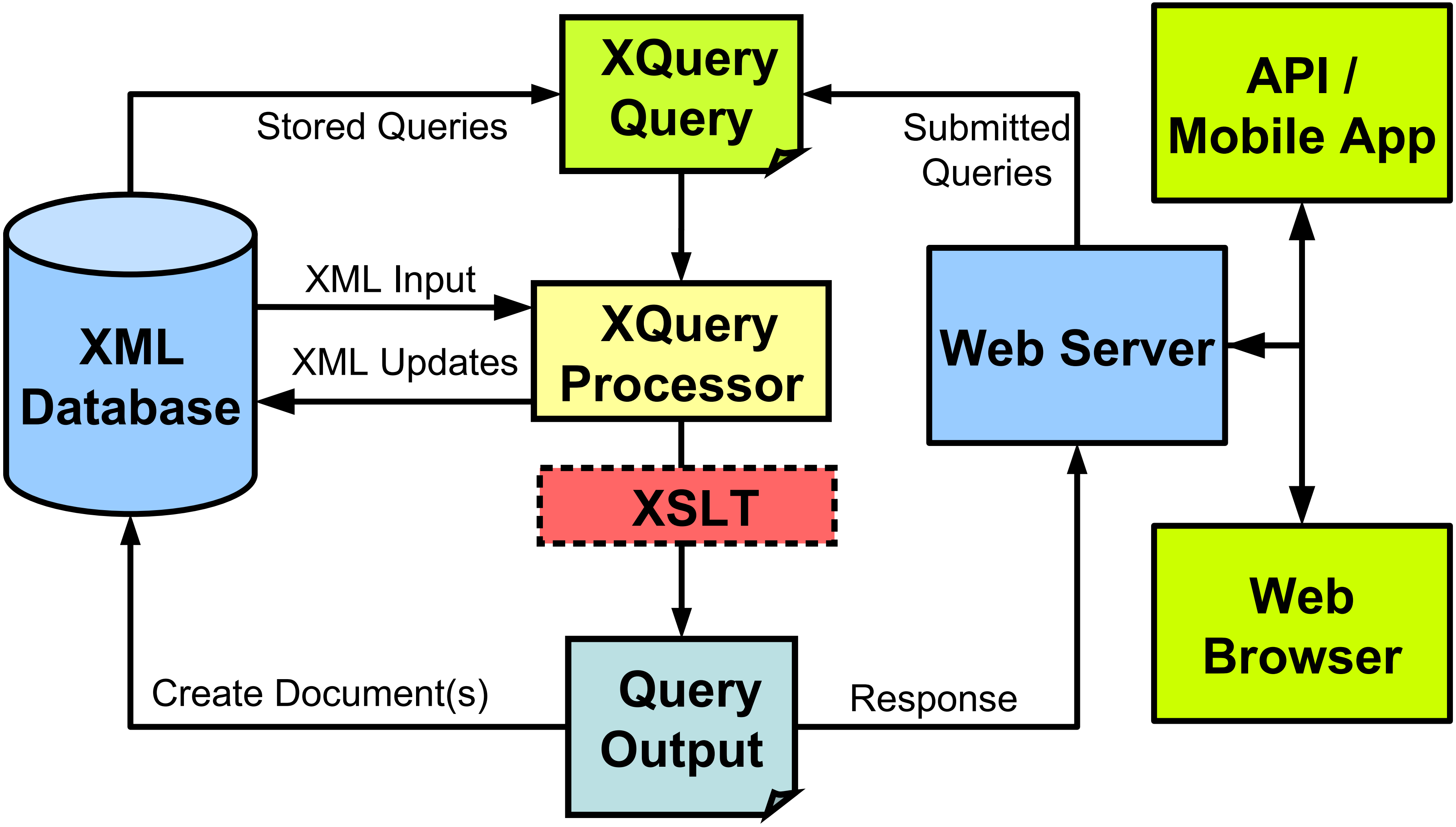


XRX / Building XML Web Applications

What is XRX?

- Originally, XRX = XForms → REST → XQuery
- A zero-translation architecture, e.g. Orbeon with eXist-db. i.e., XML end-to-end
- Now more popular:
  - XML as the storage
  - XQuery and XSLT as the backend processing
  - Maybe some XML Templating Framework
  - Delivering XML/HTML/JSON over REST
  - JavaScript on the client. Web-browser / API client.

XQuery Processing Model (Web Platform)



Working for the Web from XQuery

- Output – XSLT and XQuery Serialization 3.1
  - XML / XHTML / HTML 5 / Text / JSON

```
declare namespace output = "https://www.w3.org/2010/xslt-xquery-serialization";
declare option output:method "xhtml";
```

- More Context!
  - Accessing the HTTP Request
  - Controlling the HTTP Response
    - RESTXQ (EXQuery) – eXist-db, BaseX, and MarkLogic?
    - Vendor extensions
      - XQuery Functions - e.g. eXist-db’s Request, Response and Session Modules.
      - URI Routing. e.g. eXist-db’s XQuery URL Rewriting

eXist-db’s Request Module

- Provides a set of XQuery Functions
  - Read data from Java’s *HttpServletRequest*
  - Allows us to read all parts of the HTTP Request
- See: *http://exist-db.org/exist/apps/fundocs/view.html?uri=http://exist-db.org/xquery/request*

Given the URI:

```
http://localhost:8080/exist/rest/db/myquery.xq?abbrev=xmlss&year=2018
```

Get the value of the “abbrev” parameter:

```
request:get-parameter("abbrev", ())
```

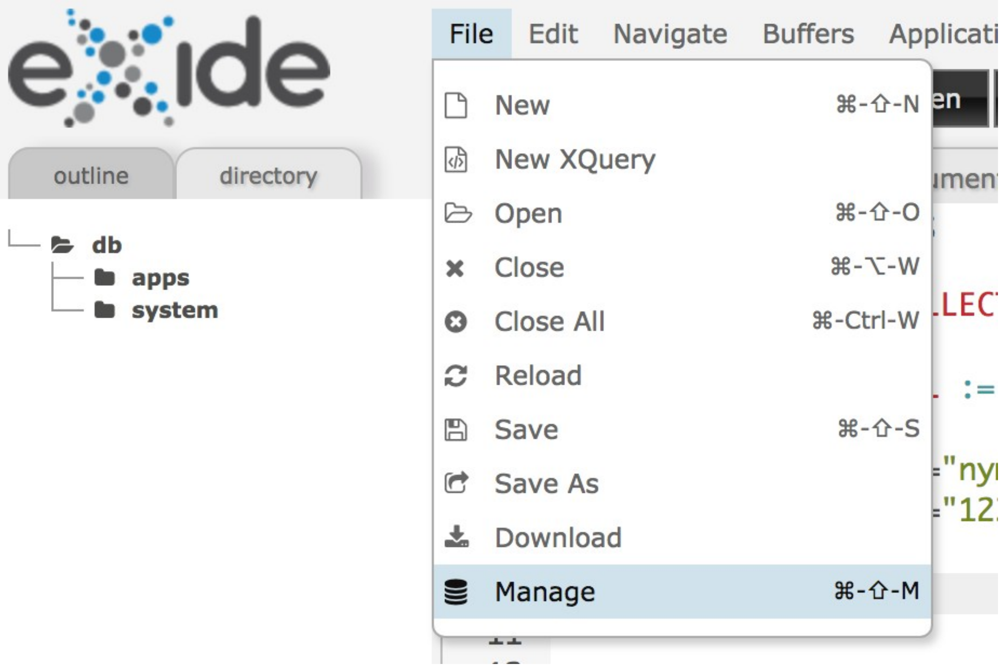
Get the name and value of all parameters:

```
request:get-parameter-names()
   ! <param name="{.}" value="{request:get-parameter(., ())}"/>
```

Lab 4: Our First XQuery for the Web

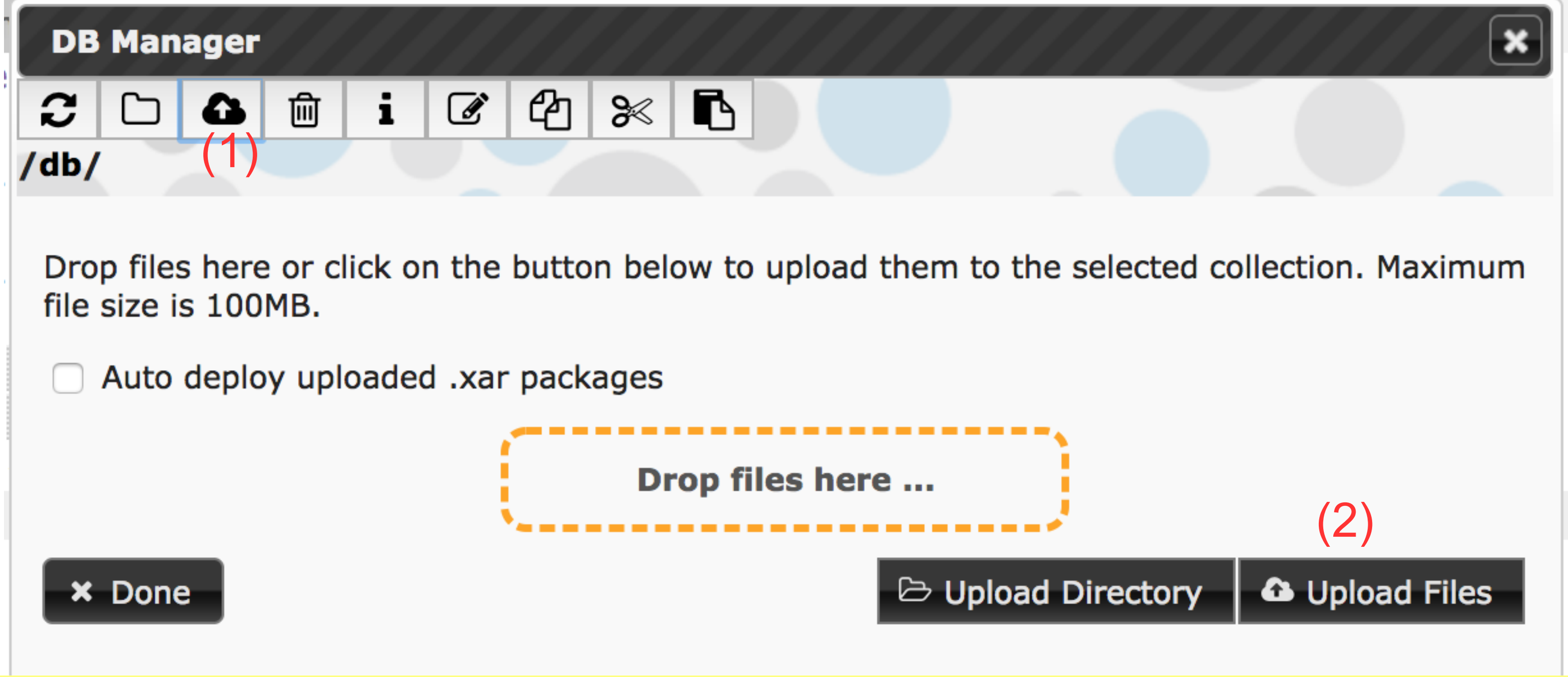
Storing your first XML Collection into eXist-db

- We need more data to work with. We will store a collection of XML files.
  - Download this file to your computer: http://static.adamretter.org.uk/hindawi-small.zip
  - From the File Menu in eXide, enter the DB Manager:



Storing your first XML Collection into eXist-db

1) Click the Upload Cloud icon
2) Click the "Upload Files" button
3) Choose the *hindawi-small.zip* file from your Computer
4) Click Close



DB Manager

/db/

Drop files here or click on the button below to upload them to the selected collection. Maximum file size is 100MB.

□ Auto deploy uploaded .xar packages

summer school

Expanding the Zip File into a Collection

1) Copy and Paste the following XQuery into eXide
2) Click the “Eval” button to run the XQuery.
3) What did the XQuery do?

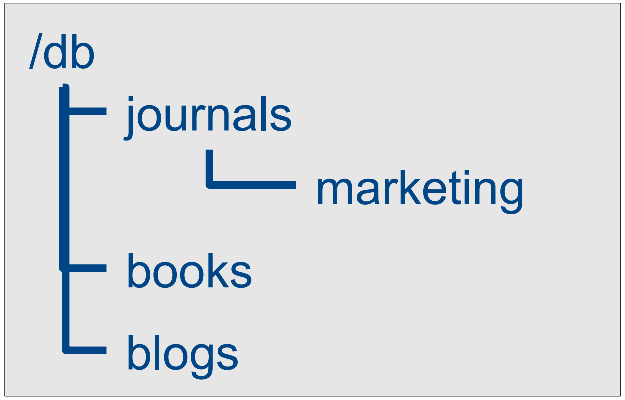
```
let $collection-uri := xmldb:create-collection("/db", "hindawi-data")
let $zip := util:binary-doc("/db/hindawi-small.zip")
return
    compression:unzip($zip, compression:no-filter#3, (), function($path, $dt, $px) {
        $collection-uri || "/" || $path
    }, ())
```

NOTE: Documents and Collections

- *doc()* and *collection()* functions take a URI
  - *fn:doc* returns zero or one document!
  - *fn:collection* return zero or many documents
- 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?

NOTE: Collections in eXist-db

- All Documents are stored in Collections
  - Documents belong to only one Collection!
- Root collection is **/db**
- Collections can contain sub-collections
- The collection hierarchy is inherited!



/db

journals

marketing

books

blogs

Quiz

How do I get all of the marketing collection?

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

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

Lab 4 Subsequence Query

- Using eXide, write an XQuery:
  - Query the collection: */db/hindawi-data*
  - Produce an XML Report listing the first 10 articles. Include title, authors and year. Order by year descending e.g.:

```
<articles>
    <article year="2014" title="Understanding the Colour of Jake the Dog">
        <authors>
            <author surname="The Human" given-names="Finn"/>
        </authors>
    </article>
    <article year="2014" title="Methods for the Hypnotisation of Princesses">
        <authors>
            <author surname="Ice King" given-names="The"/>
        </authors>
    </article>
</articles>
```

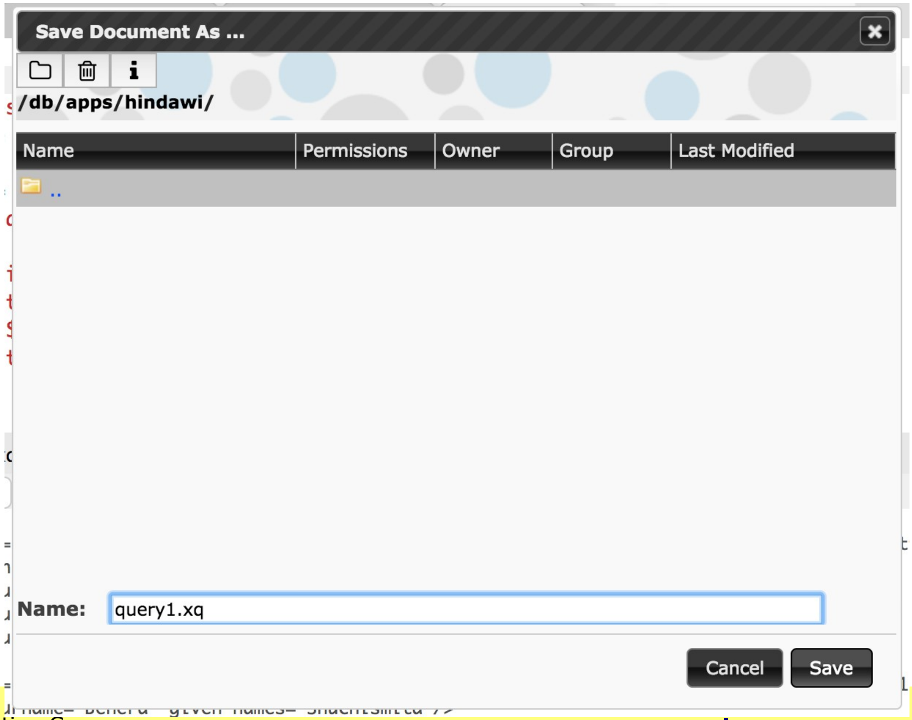
Hint: *fn:subsequence* used on *fn:collection*

Lab 4 Subsequence Query - Solution

```
<articles>
{
let $articles := subsequence(collection("/db/hindawi-data"), 1, 10)/article
for $article in $articles
let $authors := $article//contrib-group/contrib[@contrib-type eq "author"]/name
let $year := $article//pub-date[@pub-type eq "publication-year"]/year
order by $year cast as xs:int descending
return
        <article year="{$year}"
            title="{$article//article-meta//article-title}">
          { $authors ! <author surname="{surname}" given-names="{given-names}"/> }
        </article>
}
</articles>
```

Storing your XQuery into eXist-db

1) Click the “Save” button
2) Navigate into the “apps” collection
3) Click the “Create Collection” button, enter “hindawi”, click Ok.
4) Navigate into the new “hindawi” sub-collection
5) Enter the name “list.xq”, click “Save”



/db/apps/hindawi/

Name: query1.xq

Executing your Query from the Web

- Your XQuery is now stored as a resource in eXist-db at the URI: */db/apps/hindaw/list.xq*

- We will use eXist-db's REST Server to execute the query. The REST Server URI's start */exist/rest* followed by the URI to the resource in the database.

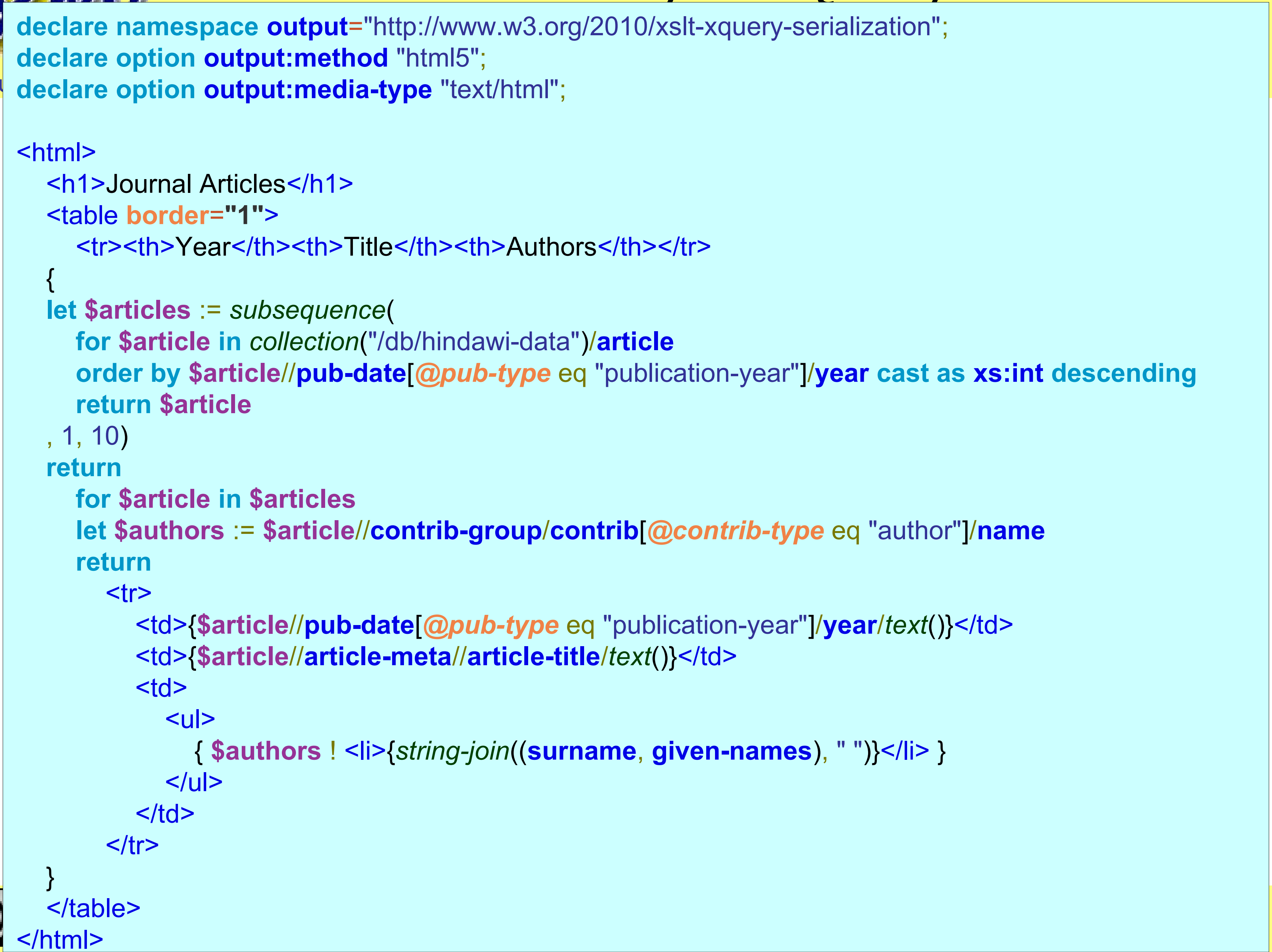
- In your web-browser, visit the URL:
  - http://xmlss*N*.evolvedbinary.com:8080/exist/rest/db/apps/hindawi/list.xq

Lab 4 HTML’ize your Query

- Using eXide, modify your XQuery “*list.xq*”:
  - To output valid HTML instead of XML
  - **HINT:** You will need *output:method* and *output:media-type* from the *XSLT and XQuery Serialization 3.1* spec.
  - Test in the Web Browser!

```
<table>
    <tr><th>Year</th><th>Title</th><th>Authors</th></tr>
    <tr>
        <td>2014</td>
        <td>Understanding the Colour of Jake the Dog</td>
        <td>
            <ul>
                <li>Finn The Human</li>
            </ul>
        </td>
    </tr>
</table>
```

Lab 4 HTML’ize your Query - Solution



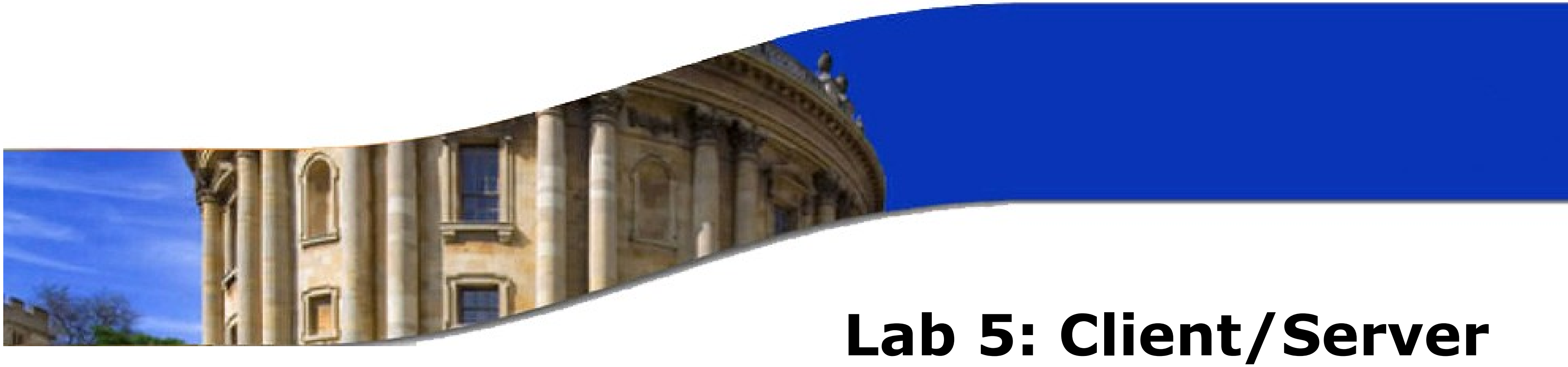
```
declare namespace output="http://www.w3.org/2010/xslt-xquery-serialization";
declare option output:method "html5";
declare option output:media-type "text/html";

<html>
    <h1>Journal Articles</h1>
    <table border="1">
        <tr><th>Year</th><th>Title</th><th>Authors</th></tr>
    {
    let $articles := subsequence(
        for $article in collection("/db/hindawi-data")/article
        order by $article//pub-date[@pub-type eq "publication-year"]/year cast as xs:int descending
        return $article
    , 1, 10)
    return
        for $article in $articles
        let $authors := $article//contrib-group/contrib[@contrib-type eq "author"]/name
        return
            <tr>
                <td>{$article//pub-date[@pub-type eq "publication-year"]/year/text()}</td>
                <td>{$article//article-meta//article-title/text()}</td>
                <td>
                    <ul>
                        { $authors ! <li>{string-join((surname, given-names), " ")}</li> }
                    </ul>
                </td>
            </tr>
    }
    </table>
</html>
```

Our First XQuery for the Web

- We have now encountered:
  - XML Collections
    - How eXist-db implements them
    - Creating Collections – *xmldb:create-collection*
    - Storing XML into Collections – *xmldb:store*
    - Providing the Context Sequence from *fn:collection*
  - Binary Documents – *util:binary-doc*
  - Uncompressing Zip files - *compression:unzip*
  - Function References and Inline Functions
  - Executing a Server-side XQuery from the Web Browser
  - Generating a dynamic HTML page from Xquery
    - Serialization of XML to HTML – *output:method "html"*

summer school



Lab 5: Client/Server Interaction with XQuery

Lab 5 Processing a Form

- Using eXide, modify your `*list.xq*` query:
  - Add a HTML form at the top of the page
  - The form should have a single “year” field
  - When the form is submitted it should call your `*list.xq*` query again, and only return the first 10 results for that year.
- Example HTML Form:

```
<form action="list.xq" method="post">
   <label for="year">Year:</label>
   <input name="year" id="year"/>
   <input type="submit"/>
</form>
```

Hint:
 - eXist-db’s *request:get-parameter* function that we saw previously can be used to get the value of the form field
- A *predicate* or *where clause* can be used to restrict the results to a particular year

Lab 5 Processing a Form - Solution

- The start of our ` *list.xq* ` query now looks like:

```
<html>
    <h1>Journal Articles</h1>
    <form action="list.xq" method="post">
        <label for="year">Year:</label>
        <input name="year" id="year" value="{request:get-parameter("year",())}"/>
        <input type="submit"/>
    </form>
    <hr/>
    <table border="1">
...
```

- Our initial FLWOR becomes:

```
for $article in collection("/db/hindawi-data")/article
let $year := $article//pub-date[@pub-type eq "publication-year"]/year
where $year eq request:get-parameter("year", ())
order by $year cast as xs:int descending
return $article
```

Client/Server Interaction with XQuery

- We have now encountered:
  - The Client (the web-browser)
    - Sends the HTML Form request over HTTP to eXist-db
  - The Server (eXist-db)
    - eXist-db routes the HTTP request to the XQuery
    - The XQuery is executed
    - Our XQuery performs some actions and generates HTML
      - The *request:get-parameter* function gets the form data
    - eXist-db sends the HTML response back to the client

summer school

Lab 6: Full Text Queries with XQuery

Full Text Queries

- We can use XPath to query the structure of a document.
- We can use Full Text queries to query the content of the document.
- Combining structural and content queries together is incredibly powerful!
- XQuery and XPath Full Text 3.0
  - W3C Spec.
  - Some implement, some don’t!
    - BaseX does, eXist-db has something else...

Full Text Queries in eXist-db

- Define indexes for your Collection(s)
  - This is done in XML
  - Stored in a file named `*collection.xconf*` in the system Collection.
  - If your collection is: */db/abc*, then your *collection.xconf* must be located at: */db/system/config/db/abc/collection.xconf*
- Reindex your Collection
- Write full-text queries using the *ft:query* function(s)

- Other index types are available: NGram, Range, QName, Sort, Algolia, and Geospatial.

summer school

Lab 6 Full Text Indexing

- Using eXide, store the following XML file to: */db/system/config/db/hindawi-data/collection.xconf*:
  - eXide should ask you if you want to apply the Configuration, choose yes.
    - If not – ask me!

```
<collection xmlns="http://exist-db.org/collection-config/1.0">
    <index>
        <lucene>
            <analyzer class="org.apache.lucene.analysis.standard.StandardAnalyzer"/>
            <text qname="article-title" boost="2.0"/>
            <text qname="p">
                <inline qname="sup"/>
                <inline qname="sub"/>
            </text>
        </lucene>
    </index>
</collection>
```

Lab 6 Full Text Indexing

- Using eXide, modify your `*list.xq*` query:
  - Add an additional field to your HTML form called "*keywords*"
  - Your list.xq query should when it detects the *keywords* field restrict the results to the first 10 which match the keyword in the article abstract.

Hint:
- You will need to use eXist-db's *ft:query* function inside a predicate on the abstract of the article
- eXist-db's Full Text documentation is here: http://www.exist-db.org/exist/apps/doc/lucene

Lab 6 Full Text Indexing - Solution

- The end of our HTML form of our `*list.xq*` query:

```
...
    <label for="keywords">Keywords:</label>
    <input name="keywords" id="keywords" value="{request:get-parameter("keywords", ())}"/>
    <input type="submit"/>
</form>
```

- Our initial FLWOR becomes:

```
for $article in
    if(request:get-parameter("keywords", ())) then
        collection("/db/hindawi-data")/article[.//abstract[ft:query(p, request:get-parameter("keywords", ()))]]
    else
        collection("/db/hindawi-data")/article
let $year :=
...
```