XML Databases and XQuery

Adam Retter

adam@evolvedbinary.com
@adamretter

Who are you?

- Programmer / Consultant
  - XQuery / XSLT
  - Scala / Java
  - Concurrency

- Core contributor to eXist XML Database
- Contributor to Facebook's RocksDB
- W3C XQuery WG Invited expert
- www.adamretter.org.uk
Learning Objectives

This class looks at XML Databases and XQuery. We will use the eXist XML Database to demonstrate Web Application programming in XQuery.

1. Understand what an XML Database is
2. Understand how an XML Database works
3. Introduction to XQuery
4. Learn the basics of XQuery
5. Test your knowledge
6. Practice by building a Simple XQuery App
7. Review and Improve the App
Lots of material, but this year we will focus on...

Session 1 (~90 Minutes)
1. What is an XML Database?
2. Introduction to XQuery
3. Hands-on. XQuery Basics (30 Minutes)

(: Break (30 Minutes) :)

Session 2 (~90 Minutes)
4. Advanced XQuery
5. XML Web Applications
6. Hands-on. Adding features to the XML Application (30 Minutes)
Introduction to 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. Apache Solr?
    • No direct XPath access!
  – How to Update?
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).”

What is an XML Database?

• More than just a filesystem!

• Unit of storage is the Document

• It ingests (and *may* return) XML documents

• Node aware, e.g. across and within document access

• CRUD operations on document(s)/node(s)

• Some form of query facility/language
What is an XML Database?

- Full-Text capabilities
- Indexes defined for document queries
- Often defines “Collection”s
- May support non-XML content
  - e.g. Key/Value, Tabular, JSON, Binary, Graph etc.
- Single or Multi-user: Client/Server and/or Embedded
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
    - Typically used when only small amounts of XML are involved

- **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 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
Example XML -

```xml
<email>
  <mail>
    <envelope>
      <from>adam.retter@googlemail.com</from>
      <to>someone@somewhere.com</to>
      <date>2015-09-14T15:38:00.687+01:00</date>
      <subject>Hello there</subject>
    </envelope>
    <body>Hey someone...</body>
    <attachment>...</attachment>
  </mail>
  <mail>
    ...
  </mail>
  ...
</email>
```

A possible normalized Entity Relationship model:

Q: Is this the best model? What other options?
Example - Manually Shredding XML 3/4

• ER model with de-duplicated attachments:

  mail
  - mailId
  - envelopeld
  - bodyId

  envelope
  - envelopeld
  - from
  - to
  - date
  - subject

  attachments
  - mailId
  - attachmentId

  attachment
  - attachmentId

• Q: Again, is this the best model?
• One possible (simplified) DDL -

```sql
create table env (  
    envId       integer not null generated always as identity primary key,  
    from        varchar(100),  
    to          varchar(100),  
    date        varchar(30),  
    subject     varchar(100)  
); 

create table body (  
    bodyId       integer not null generated always as identity primary key,  
    body         varchar(30000)  
); 

create table attach (  
    attachId     integer not null generated always as identity primary key,  
    attachment   varchar(100)  
); 
```
### Example - Manually Shredding XML 3/4

#### Populated Tables -

<table>
<thead>
<tr>
<th>email</th>
<th>bodyId</th>
<th>attachId</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>envid</td>
<td>bodyId</td>
<td>attachId</td>
</tr>
<tr>
<td>765</td>
<td>461</td>
<td>NULL</td>
</tr>
<tr>
<td>342</td>
<td>987</td>
<td>NULL</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>env</th>
<th>from</th>
<th>to</th>
<th>date</th>
<th>subject</th>
</tr>
</thead>
<tbody>
<tr>
<td>envid</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>765</td>
<td><a href="mailto:adam.retter@googlemail.com">adam.retter@googlemail.com</a></td>
<td><a href="mailto:someone@somewhere.com">someone@somewhere.com</a></td>
<td>2015-09-14T15:38:00.687+01:00</td>
<td>Hello there</td>
</tr>
<tr>
<td>342</td>
<td><a href="mailto:other@unknown.com">other@unknown.com</a></td>
<td><a href="mailto:someone@somewhere.com">someone@somewhere.com</a></td>
<td>2015-09-01T12:15:00.234+01:00</td>
<td>Whats on?</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>body</th>
<th>bodyId</th>
<th>body</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>461</td>
<td>How you doing? What's...</td>
</tr>
<tr>
<td></td>
<td>987</td>
<td>Dear Sir/Madam, I am writing...</td>
</tr>
</tbody>
</table>

- What happens if your document model changes?
Example - XMLType and SQL

<table>
<thead>
<tr>
<th>id</th>
<th>issn</th>
<th>short_name</th>
<th>vol</th>
<th>journal</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0012-1606</td>
<td>Dev. Biol.</td>
<td>369</td>
<td>&lt;journal&gt;Developmental Biology&lt;/name&gt;</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>&lt;publisher&gt;Elsevier&lt;/publisher&gt;</td>
</tr>
<tr>
<td>2</td>
<td>8756-8233</td>
<td>Drugs Soc.</td>
<td>11</td>
<td>&lt;journal&gt;Drugs and Society&lt;/name&gt;</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>&lt;publisher&gt;Taylor &amp; Francis&lt;/publisher&gt;</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>id</th>
<th>vol</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>369</td>
<td>&lt;name&gt;Developmental Biology&lt;/name&gt;</td>
</tr>
</tbody>
</table>
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?
Use a Native XML Database!

- Stores/Retrieves/Queries Documents

- Defines Collection's

- Indexes optimised for XML

- Supports XPath / XQuery
  - Possibly: XSLT, XQ Full Text, XQ Scripting, JSONiq...

- Document Management Platform capabilities

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

Licensed under a Creative Commons Attribution-Noncommercial-Share Alike 3.0 Unported License
Advantages of Native XML Database

• Compared to a Filesystem
  – Manage Document Access
  – Indexing and then Querying (and vice-versa)
  – 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
Choosing an XML Database
Expected Features of a NXDB 1/3

• Query – XPath and XQuery
  – Full text search – XQuery Full Text vs proprietary
  – Updates - XUpdate and/or XQuery Update
  – Programming - XQuery Scripting
  – JSONiq?
  – Extensions API
    • Custom functions in lower-level language
    • EXPath
Expected Features of a NXDB 2/3

• Transform and Processing – XSLT and XProc
  – Optimised for the db?

• Forms – XForms
  – or... maybe just HTML5 + JavaScript
    • JSON serialization helps here

• Metadata – Search documents/collections

• Versioning

• Scalability – Sharding and/or Replication
Expected Features of a NXDB 3/3

- Polyglot – What about binaries, JSON, K/V etc.

- APIs
  - REST, RESTXQ
  - WebDav, FTP,
  - XML-RPC, XML:DB
  - XQJ
  - Language specific – C++, Java, Python, Ruby, etc?

- **Clear ACID compliance statement**
Selecting an XML Database 1/6

• Why do you want an XML database?
  – Repository
    • Accountability?
    • Archival... How about retrieval?
    • Data Warehouse? What about reporting?
  – Querying
    • Online – indexes, scalability and performance!
    • Batch – perhaps tools like HDFS and Hadoop?
  – Publishing
    • Repurposing content
    • Transformation
Selecting an XML Database 2/6

• Enabled vs. Native vs. Polyglot Persistence
  – What is your primary data model?
    • Is XML your primary/only concern?
  – What about Tabular, Key/Value, Graph? etc
    – Could also be stored as XML vs Efficient Query
  • JSON?
    – 28msec Zorba has JSONiq
    – MarkLogic has JavaScript embedded
    – eXist-db, XQuery 3.1 maps/arrays/parse/serialize

– Corporate IT enforcement?
  • Expectation to use Oracle RDBMS or MySQL?
  • Ability to support multiple databases platforms?
Selecting an XML Database 3/6

• Embedded vs. Client/Server
  – Is this part of a larger self contained application?
    • e.g. Oracle Berkley DB XML, eXist, Sedna etc.
  – Does your database need to support multiple clients?
    • How do your clients expect to access your database?
      – XQJ vs JDBC etc.
      – WebDAV?
  – Does your database need to serve data via the web?
    • What is the security model?
• Scalability
  – How much data now and...
  – Clustering
    • Sharding and/or Replication
    • Introduces network issues
      – Consistency guarantees – Eventual?
      – Split-brain

• CAP Therom

  “perfect availability and consistency in the presence of partitions, which are rare.”

  -- http://www.infoq.com/articles/cap-twelve-years-later-how-the-rules-have-changed
• Query Performance?

  – Everyone asks!

  – Difficult to predict, dependent on:
    • Your XML Database system
    • Your Hardware
    • Your Data Model
    • Complexity of evaluating your queries (joins!)
    • Correct Index Configurations
    • Volume of Data

• Requires careful testing and tuning
• Isolation and Transaction Model
  – **MOST IMPORTANT!**
  – **LEAST UNDERSTOOD**

• What is the isolation level of your database transactions?
  – What about across a cluster?
  – ACID vs BASE?

• Are you using the correct level?
Native XML Database Options 1/2

- **BaseX**
  - Open Source. BSD License
  - XQuery 3.1*, XQuery Update 1.0, RESTXQ, EXPath, XQuery Full-Text 1.0

- **eXist**
  - Open Source. LGPL v2.1
  - XQuery 3.1*, XSLT 2.0, XForms 1.1, RESTXQ, XQuery Update, XProc, EXPath, Bespoke Full-Text, Customisable Extension Modules
  - Master-Slave Replication with Slave promotion.

- **Marklogic**
  - Commercial
  - XQuery 1.0/3.0*, XSLT 2.0, XForms 1.1, Bespoke Full-Text
  - Shared-Nothing Clustering

- **Others:** Sedna / EMC Documentum xDB / Zorba / etc...
Native XML Database Options 2/2

• **BaseX**
  - Serializable
  - Auto (short) locking of database for multiple readers / single-writer
  - System Transactions
  - Manual locking prolog options - `query:read-lock / query:write-lock`

• **eXist**
  - Dirty Reads
  - Auto (short) locking of resources for multiple readers / single writer
  - No PUL for XQuery Update
  - System Transactions for Write Ahead Journal

• **MarkLogic**
  - MVCC – Snapshot Isolation
  - Auto or User controllable Transactions
  - Manual locking functions - `xdmp:lock-acquire / xdmp:lock-for-update`
Getting started with eXist-db
• Native XML Database written in Java 8. Established in 2000

• Open Source, LGPL. Commercial Support: existsolutions.com

• Hierarchical Collections of Documents

• Supports XML and Binary Documents. WebDAV + REST + RESTXQ

• XQuery 3.1, XQuery Update*, Proprietary Full-Text (Lucene)

• Also: XSLT 2.0, XForms, XProc, XInclude, JSON, XHTML, HTML5

• Full Web App platform with XQuery extensions and EXPath
How to get setup?

- eXist-db is written in Java
  - You need Oracle/Open JRE 8

- Download and install v3.0-RC1 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
Collections

• Documents are stored in Collections

• Root collection is /db

• Collections can contain sub-collections

• The collection hierarchy is inherited!

Quiz
How do I get all of the marketing collection?
What does `collection("/db/journals")` return?
What does `collection("/db/books/blogs")` return?
Shameless plug:

http://shop.oreilly.com/product/0636920026525.do
Store/Retrieve with an XML Database (eXist-db)
How to store data into a NXDB?

• Storing an XML Document into eXist-db
  – Upload to the database via API

  – For Web Developers
    • Demo REST

  – For Authors/Editors
    • Demo WebDAV

  – For Programmers
    • Demo Java
    • Demo Python

  – Many other options available...
How to store data into a NXDB?

• Where has my XML Document gone?

- **Demo** – webapp/WEB-INF/data
- Highly Optimised storage format and indexes
How to retrieve data from a NXDB?

• Retrieving an XML Document from eXist-db
  – Download from the database via API
  – **Canonical form!**

  – For Web Developers
    • Demo REST

  – For Authors/Editors
    • Demo WebDAV

  – For Programmers
    • Demo Java

  – Many other options available...
Basic Database Queries

• 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>
  ]]>]]>
</query>
Querying the Database

- **Demo** - GUI Admin Client

- **Demo** - eXide

- SOAP / WebDAV / XML-RPC / Java / PHP / etc.

- Stored Queries
  - XQueries can be stored into the database
  - Executed later e.g. REST Server by URI
  - Demo
How does a NXDB really store data?

• We are about to go down the rabbit hole!

• Let us remember that XML can be modelled as either a Stream or Tree!

• XML Databases are predominantly concerned with supporting XQuery efficiently

...first we need to understand a bit about XQuery!
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!**
Where does XQuery fit?

- It's kinda just XPath++
  - If you know XPath...

- Much in common with XSLT
  - XDM and XPath
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)

- XML Database
  - XML Input
  - XML Updates
  - Create Document(s)

- Query Processor
  - Submitted Queries
  - XQuery Query

- Web Server
  - Response

- XML Updates
- Stored Queries
- XML Input

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

• XQuery 3.1 (CR) will be released soon
  – Most people still using 1.0 (and some 3.0)

• Related XML Query Standards
  – XDM (XPath and XQuery Data Model)
  – XQueryX
  – XQuery and XPath Full-Text
  – XQuery Update
  – XQuery Scripting
XDM: XQuery and XPath Data Model
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, text, comment, processing-element
  – Have a Unique Identity!

```xml
<root>
  <hello>world</hello>
  <hello>world</hello>
...</root>
```
XDM - Node Trees

XML: It's a Tree of Nodes!

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

```
<document>
  <element>
    <attribute name="ref">xmlams11</attribute>
    <element>
      <text>XML Amsterdam</text>
      <date>2011-10-26</date>
    </element>
    <element>
      <text>XML Prague</text>
      <date>2012-02-10</date>
    </element>
  </element>
</document>
```
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
Simple!
- Probably only use a few of the Atomic Value Types

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

• Quiz on XDM Nodes

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

1) How many nodes are in the document?

2) What kind of node is 'fragment2'?

3) What are the names of the attributes?

4) How many text nodes are in the document?

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

• Sequences
  – An Ordered List

– Sequence Constructor starts with '(' and ends with ')'

– Consist of Zero or More Items
  ("hello", "world")

– Can be mix of Nodes and Atomic Values
  ("hello", <gn>james</gn>, <fn>smith</fn>)

– No Nested Sequences!
  ("a", "b", ("c", "d")) becomes: ("a", "b", "c", "d")
XDM - Sequences

- 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:

    \(<\text{name}>adam</\text{name}>, \ <\text{name}>bob</\text{name}>\)
Comparison Operators

- XQuery has two types of Comparison Operators

<table>
<thead>
<tr>
<th></th>
<th>Atomic Values</th>
<th>Sequences</th>
</tr>
</thead>
<tbody>
<tr>
<td>Equal to</td>
<td>eq</td>
<td>=</td>
</tr>
<tr>
<td>Not equal to</td>
<td>ne</td>
<td>!=</td>
</tr>
<tr>
<td>Greater than</td>
<td>gt</td>
<td>&gt;</td>
</tr>
<tr>
<td>Greater than or equal to</td>
<td>ge</td>
<td>&gt;=</td>
</tr>
<tr>
<td>Less than</td>
<td>lt</td>
<td>&lt;</td>
</tr>
<tr>
<td>Less than or equal to</td>
<td>le</td>
<td>&lt;=</td>
</tr>
</tbody>
</table>

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

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

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

- is ?
- is ?
- ...and this??
Practical XQuery
Example Data

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

• Two documents:
  – Catalog (of Products)
  – Orders (of Products)
Example Data

catalog.xml

```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

```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>
```
Question 1...

1. How do we get a list of departments?
XQuery Example

input document

```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>
  </product>
</catalog>
```

query

```xml
doc("/db/shop/catalog.xml")/catalog/product/string(@dept)
```

results

WMN ACC ACC MEN
Improved XQuery Example

**input document**

```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>
  </product>
</catalog>
```

**query**

```xml
distinct-values(doc("/db/shop/catalog.xml")/catalog/product/@dept)
```

**results**

WMN ACC MEN

Q: What is distinct-values?
Q: Why don’t we need string()?
Q: What other functions are available?
Question 2…

2. How do we calculate total orders by department?
XQuery Example

input document

```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>
```

query

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

results

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

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

• A document
  ```
  doc("/db/shop/catalog.xml")
  ```

• A collection of documents
  ```
  collection("/db/shop/products")
  ```

• Or... XQuery generates original data
Documents and Collections

- doc() and collection() functions take a URI

- URI may or may not be de-referenced

- Both functions return document node(s).

- What is a Collection?
  - Implementation defined
    - A folder? Hierarchical?
    - URI! A label?

- Typically followed by an expression.
Expressions

• Path Expressions e.g. /xml/summer/school

• FLOWR Expressions:
  – for
    • creates a sequence of nodes
  – let
    • binds a sequence to a variable
  – where
    • filters the nodes on a boolean expression
  – order by
    • sorts the nodes
  – return
    • gets evaluated once for every node
Simple FLWOR Example

• Just the 'F' and 'R':

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

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

• Result:

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

– An XML Fragment!
Simple FLWOR Example with Position

• Iteration position can be bound with 'at'

```xml
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

```xml
element product {
  attribute idx { 99 },
  text { "Our New Product" }
}
```
Simple FLWOR Example with Position

• Result:

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

• Result is in Document Order!

• 'Product' is directly constructed element, with attr.

• `text()` node of Product Name was copied

• Re-write the query create valid XML document...
Simple FLWOR Example with Position

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

- What about `document{ ... }`?
FLWOR - Bindings

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

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

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

- The **where** clause will be evaluated once for each iteration
  - Familiar for SQL users

- On some implementations it can be more efficient to use a **predicate** instead

```xml
doc("catalog.xml")/catalog/product[@dept eq "ACC"]
```
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

```xml
<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>
```
Variable Scope

• XQuery is functional
  – It has immutable variables
  – Variables are “bound” to a value

• What result does the following yield?

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

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

• More comprehensible when rewritten to:

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

Answer) The Sequence (2 to 101)
Variable Scope

• Variable Bindings have a limited Scope

• Is this valid?

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

• ...and is this valid?

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

• XQuery is a full functional Programming Language

• Aspects of XQuery we will examine
  – Conditional Expressions
  – Functions
Conditional Expressions

• If, then, else syntax:

```xml
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)

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

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

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

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

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

• **not** function will also resolve *effective* boolean value
Type Conditional Expressions

• typeswitch syntax:

```xml
<result>found the name: {$n/text()}</result>
```

```xml
<result>found an element: {local-name($e)}</result>
```

```xml
<result>found the text: {$t}</result>
```

```xml
<result>found the integer: {$i}</result>
```

```xml
()```

Type Conditional Expressions

- Identity Transform using typeswitch:

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

    (: TODO add your overrides here:)

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

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

default return
    $n
};
```
Functions

• XQuery has many built-in functions

  – Defined in W3C Spec:
    • XQuery 1.0 and XPath 2.0 Functions and Operators
      http://www.w3.org/TR/xpath-functions/

  – So far we have seen:
    doc  collection  distinct-values  sum
    current-date  days-from-duration  not
    local-name  node-name

  – There are >150 functions available in XQuery 1.0
  – There are >220 functions available in XQuery 3.0
Extension Functions

• Vendors/Processors may also provide extension functions

• e.g. SQL Queries, Sending Email, DB Management
  – eXist-db provides ~600 functions!

• Extension functions are processor specific
  – …Non-Portable XQuery code
  – EXPath and EXQuery Projects try to standardise

http://www.expath.org
http://www.exquery.org
User Defined Functions

- You can also write your own functions in XQuery

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

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

- Many common functions available at FunctX
User Defined Functions

• Function Declaration:

```xml
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

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

• Parameters may be explicitly typed

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

• Parameters may be sequences with constraints
  – Cardinality
    • One
      ```xml
      declare function local:my-third-function($date as xs:date)
      ```
    • Zero or One ?
      ```xml
      declare function local:my-third-function($date as xs:date?)
      ```
    • One or More +
      ```xml
      declare function local:my-third-function($dates as xs:date+)
      ```
    • Zero or More *
      ```xml
      declare function local:my-third-function($dates as xs:date*)
      ```
User Defined Functions

• Return values may be explicitly typed

```xml
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

```xml
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...
**Main Module**

```xquery
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
xquery version "1.0";

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

declare function my:say-hello($name) {
  <hello>{$name}</hello>
};
```
Building Applications in XML
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 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 Processing

• 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

External Variable Definitions

XML Documents

Statically known Documents

XQuery

XQuery Processor

fn:collection

fn:doc

sql:query

XML Documents

XML Document

RDBMS

Result (XML?)

Licensed under a Creative Commons Attribution-Noncommercial-Share Alike 3.0 Unported License
XQuery and the Web

• 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

• 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 examples 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

```
<?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: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>
```

```
<x http:request xmlns:x=http="http://expath.org/ns/http">
  <http:header name="Accept" value="application/xhtml+xml"/>
  <http:header name="User-Agent" value="Your Browser"/>
</x>
```

```
<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

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

- **XML Database**
  - XML Input
  - XML Updates
  - Create Document(s)

- **XQuery Processor**
  - Submitted Queries
  - Stored Queries
  - XQuery
  - Query Output
  - Results
  - Create Document(s)

- **Web Server (REST)**
  - Response
  - Query

- **XForms Processor**
XQuery on the Web extras

- 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.
XML Web App Frameworks*

- 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*

<table>
<thead>
<tr>
<th>Framework</th>
<th>Created</th>
<th>Server Arch.</th>
<th>App Framework</th>
<th>Portability</th>
</tr>
</thead>
<tbody>
<tr>
<td>REST Server</td>
<td>2003</td>
<td>REST + XQY execute</td>
<td>n/a</td>
<td>eXist</td>
</tr>
<tr>
<td>HTTP App Server</td>
<td>???</td>
<td>XQ exec.</td>
<td>n/a</td>
<td>MarkLogic</td>
</tr>
<tr>
<td>XQuery URL Rewrite</td>
<td>2008</td>
<td>n/a</td>
<td>URL Routing + View Pipelines</td>
<td>eXist</td>
</tr>
<tr>
<td>URL Rewrite</td>
<td>2009</td>
<td>n/a</td>
<td>URL Routing</td>
<td>MarkLogic</td>
</tr>
<tr>
<td>XQ MVC</td>
<td>2009</td>
<td>n/a</td>
<td>MVC</td>
<td>eXist, MarkLogic</td>
</tr>
<tr>
<td>Corona</td>
<td>2010</td>
<td>REST</td>
<td>n/a</td>
<td>MarkLogic</td>
</tr>
<tr>
<td>REST</td>
<td>2011</td>
<td>REST + XQY execute</td>
<td>n/a</td>
<td>BaseX</td>
</tr>
<tr>
<td>REST Endpoint</td>
<td>2011</td>
<td>n/a</td>
<td>URL Routing</td>
<td>MarkLogic</td>
</tr>
<tr>
<td>mustache.xq</td>
<td>2011</td>
<td>n/a</td>
<td>View Templating</td>
<td>BaseX, eXist, MarkLogic</td>
</tr>
<tr>
<td>Roxy</td>
<td>2012</td>
<td>n/a</td>
<td>mVC</td>
<td>MarkLogic</td>
</tr>
<tr>
<td>REST API</td>
<td>2012</td>
<td>REST + XQY execute</td>
<td>n/a</td>
<td>MarkLogic</td>
</tr>
<tr>
<td>RESTXQ</td>
<td>2012</td>
<td>REST API</td>
<td>URL Routing View Serialization</td>
<td>BaseX, eXist, MarkLogic</td>
</tr>
<tr>
<td>templates.xql</td>
<td>2012</td>
<td>n/a</td>
<td>View Templating</td>
<td>eXist*</td>
</tr>
</tbody>
</table>
(3.1) Direct XQuery Function Call (RESTXQ)

```xml
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>
}
```

XML Web App Frameworks*

- 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 Web Application
What are we building?

- Dataset is PubMed
  - Details of Articles published in Medical Journals

- Application Should:
  - Search Articles
    - By Journal
    - By Author
    - By Date
    - By Keyword
  - Browse / Summarise Results
<PubmedArticleSet>

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

</PubmedArticle>
</PubmedArticleSet>
Step 1 – Composing the Queries

Building an XML Web Application
• Find Articles by Journal ISSN

```xml
xquery version "1.0";

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

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

- PubMed is a big dataset!

- Databases rely on Indexes for performance

- Indexes can be used for:
  - Comparisons
  - Full Text Search
  - Geospatial Calculation

- Indexes are configured in:
  - /db/system/config/db
  - Collection Configuration (collection.xconf)
eXist-db Index Configuration

- `Find Articles by Journal ISSN` comparison is:
  
  ```xml
  [MedlineCitation/Article/Journal/ISSN eq $issn]
  ```

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

  ```xml
  <collection xmlns="http://exist-db.org/collection-config/1.0">
    <index>
      <!-- journal -->
      <create qname="ISSN" type="xs:string"/>
    </index>
  </collection>
  ```

- **Note**: Need to Re-Index!
Search by Journal

• Find Articles by Journal Title

```xml
xquery version "1.0";

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

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

• What if I don't know the name of the Journal?
Full Text Search in eXist-db

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

- You must establish an Index

```
<collection xmlns="http://exist-db.org/collection-config/1.0">
  <index>
    <!-- journal -->
    <create qname="ISSN" type="xs:string"/>
    <create qname="Title" type="xs:string"/>
    <text qname="Title"/>
  </index>
</collection>
```
Full Text Search in eXist-db

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

• Find Articles by Journal Title (Full Text)

```xml
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)
```
• Find Articles by Author

```xml
<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)

```
• Find Articles by Author

```xml
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)
```
• 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 Web Application
From Queries to Application

• It's 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

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

- RESTful Annotations for XQuery 3.0
  
  http://localhost:8080/exist/restxq

```xml
xquery version "3.0";

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

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

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

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

```xml
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?

```xml
xquery version "1.0";

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

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

- **RESTXQ**
  - Use Templates
    - In the path `%rest:path("/journal/issn/{$issn}"")`

```xml
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]
};
```
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 Web Application
Interfacing with JavaScript

- 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

```xml
<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)`

```json
{
    "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"

```xml
<container>
  <person>
    <name>adam</name>
    <age json:literal="true">33</age>
  </person>
</container>
```
Array Construction is determined by the cardinality of element names. Can be tricky!
  - Can control by using `json:array="true"`

```json
{
    "person": {
        "name":"adam",
        "age":33
    }
}
```
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
        }
    ]
}
• **RESTXQ** - `%output:method("json")`
  - adapts XSLT and XQuery Serialization 3.0
  - Per-function serialization
  - JSON format is implementation-defined

```xml
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
   - Your on your own with mixed-content: node -> string value

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

declare option output:method "json"

map {
    "person" : map {
        "name": "adam",
        "age": 33
    }
}
```
• Your XQuery looks *like* JSON
  – Serialization of XDM permits Nodes
  – Use XDM 3.1 Array and Map items for control of JSON format

```xml
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:
And...

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

- **Server**
  
  http://play.evolvedbinary.com:8080/exist/apps/eXide

- **Collection**
  
  /db/products

- **Functions**
  
  - `xmldb:get-child-collections($collection-path)`
  - `xmldb:get-child-resources($collection-path)`
  - https://www.w3.org/TR/xpath-functions-31/#quickcontents
Challenges - Part 1

• How many documents are in /db/products/examples, what does their structure look like?

• What sub-collections of /db/products exist?

• How many:
  – Tablets? How many reviews for Tablets?
  – Tablets and Laptops?
  – How many total reviews?
Challenges – Part 2

Please test first with the `/db/products/examples` collection

- Which product(s) have the most reviews?
- Which product(s) have the highest average “Overall” score?
- Which year(s) have the most laptop reviews?
XQuery Challenge 2
What you need to know...

- Server
  http://play.evolvedbinary.com:8080/exist/apps/eXide

- Data Collection: /db/pubmed/data
  - Examine: /PubmedArticleSet/PubmedArticle[1]

- Create your own App Collection
  /db/⟨your-app-name⟩

- Functions
  - xmldb:collection-available("/db/⟨your-app-name⟩")
  - xmldb:create-collection("/db", "⟨your-app-name⟩")
Challenges - Part 1

- Functions
  - [https://www.w3.org/TR/xpath-functions-31/#quickcontents](https://www.w3.org/TR/xpath-functions-31/#quickcontents)
  - [http://exist-db.org/exist/apps/fundocs/index.html](http://exist-db.org/exist/apps/fundocs/index.html)
  - **HINT:** request:get-parameter($param-name, $default-val)

- Create and store a HTML form (form1.html) into your Collection
  - Form should take some sort of query, e.g. date, name or keyword

- Create an XQuery that processes the form, queries the /db/pubmed/data collection, and shows the result as XML
Challenges - Part 2

• Functions
  • https://www.w3.org/TR/xpath-functions-31/#quickcontents
  • http://exist-db.org/exist/apps/fundocs/index.html
  • \texttt{HINT: transform:transform($xml-node, $xslt-node, ())}
  • \texttt{HINT: fn:serialize($xml-node, $serialization-parameters)}

• Modify your query from part 1 to optionally return HTML (use XSLT for the transform)

• Modify your query again, to optionally return JSON
  • \texttt{<output:serialization-parameters><output:method value="json"/></output:serialization-parameters>
Challenges - Part 3

- **Functions**
  - [http://exist-db.org/exist/apps/doc/lucene.xml#D2.2.5](http://exist-db.org/exist/apps/doc/lucene.xml#D2.2.5)
  - `ft:query($xml-node, $query-expression)`

- Examine the Full Text index definitions in:
  `/db/system/conf/db/pubmed/data/collection.xconf`

- **Modify your HTML form and XQuery again, to also offer a full-text query option**