Spring 2009

01001100 01010100 01000101 01010010
LTER DataBits
Information Management Newsletter of
The Long Term Ecological Research Network
01001100 01010100 01000101 01010010

Feature: Getting started with eXist and XQuery

- Wade Sheldon (GCE)

Introduction

Two recent LTER workshops were convened to plan and develop ProjectDB, a cross-site research project description language and database (Walsh and Downing, 2008). During the first workshop participants agreed to use eXist, an open source native XML database (http://exist.sourceforge.net/), as the back-end system for storing and retrieving research project documents. This database was primarily chosen to leverage ongoing software development work at CAP LTER that uses eXist, but excellent query performance, built-in support for both REST and SOAP web service interfaces, and simplicity of configuration and administration were also influential factors.

The combination of eXist and XQuery (the XML query language used by eXist) proved to be extremely effective for ProjectDB, exceeding all expectations. A working group of six Information Managers and a CAP software developer designed and implemented a complete system for storing, querying, summarizing and displaying research project documents in just a few days, including RESTful web services to support all the primary use cases identified during the planning workshop (Gries et al.). The rapid success of this development effort has sparked interest in eXist and XQuery across the LTER IM community, and this article presents an overview and brief guidelines on how to get started using this new XML technology.

eXist Database

Many options are currently available for storing and searching XML documents. Recent versions of major relational database management systems support XML storage and retrieval natively, and software packages like the KNB Metacat (http://knb.ecoinformatics.org/software/metacat/) XML-enable conventional relational databases by decomposing nodes into text strings and providing XML-centric document models and supporting technology. In contrast, eXist represents a new breed of native XML databases (NXD) that use XML documents as logical units of storage rather than rows in a table, and employ XML-based specifications and technology for all operations (e.g. XQuery, XPath, XML Infoset, XSLT). Other NXDs include Apache Xindice (http://xml.apache.org/xindice/), BaseX (http://basex.org/) and OZONE (http://www.ozone-db.org/), but eXist stands out due to its excellent XQuery implementation, built-in support for a broad range of web technology standards, and active developer community.

The eXist database is implemented entirely in Java, so it can be run on most computer platforms. The only dependency is that a complete JDK (Java Platform) 1.4.2 or higher must be present on the system prior to running the installer .jar file. A simple Java web server (Jetty, http://www.mortbay.org/jetty/) is installed by default to provide web-based access and REST/SOAP support, but eXist can also be integrated into an existing Apache Tomcat installation using a provided .war file. Once installed, the database can be administered using XQuery-based web forms or a Java client application. Fine-grained access control is supported, and users and groups can be defined and managed internally by eXist or optionally retrieved from an LDAP server. In addition to the supplied interfaces, eXist can be configured as a data source in the oXygen XML editor (see http://www.oxygenxml.com/demo/eXist/eXist.html), which proved to be invaluable during the ProjectDB development workshop.

Storing and retrieving XML data using eXist is extremely easy and fast compared to relational databases, because documents are kept intact. XML files are simply uploaded or downloaded using the management interfaces, HTTP GET or PUT commands, or by clicking on files in the oXygen database connection view. Documents can be stored in a single root directory or organized into hierarchical “collections”, similar to directories in a file system. Besides organizing documents, collections are useful for restricting or “scoping” queries to include a specific subset of documents and for providing different levels of access for specific users or groups. XML documents must be well formed to be stored, but are not validated by default; however, both implicit validation (rejection of invalid documents) and explicit validation (validation report after insertion) can be configured, if desired. Non-XML documents can also be stored in eXist, including XQuery files, XSL and CSS style sheets, HTML and JavaScript files, and binary files like images, making eXist ideally suited as a back-end for web applications and content management systems.

As already mentioned, one of eXist's strengths is built-in support for many web technologies, including XQuery, XSLT (1.0 and 2.0), REST, SOAP, XMLRPC and WebDAV. For ProjectDB, we focused on eXist's REST interface which allows documents to be stored and retrieved, XQueries run, and XSLT transforms performed using parameterized URLs (i.e. HTTP GET requests) or other standard HTTP commands (i.e. POST, PUT, DELETE) and HTTP authentication. Collections are automatically mapped as virtual directories below the base directory (i.e. /exist/rest/db/), providing logical access to any files stored in the system. For example, the following URL retrieves a GCE research project document from the LNO eXist database using the REST API:http://amble.lternet.edu:8080/exist/rest/db/projects/data/gce/gce_hammoc...

XQuery Language

As mentioned above, XQuery is the primary language used to search and manipulate data stored in eXist and other native XML databases. The XQuery 1.0 specification was accepted as a formal W3C recommendation in January 2007, along with 7 related specifications including XQueryX 1.0, XSLT 2.0 and XPath 2.0. XQuery provides many of the same capabilities for working with XML documents and collections that the SQL Data Modeling Language provides for relational databases, including searching, filtering, joining and re-factoring data dynamically. XQuery 1.0 leverages the document model, document navigation syntax and function library of XPath 2.0, as does XSLT 2.0. Unlike XSLT, however, XQuery is not itself an XML dialect so queries are simpler to read and write and special characters do not need to be encoded.

Simple queries can be performed using XPath syntax alone (i.e. called path queries), but most XQueries are written as FLWOR (pronounced flower) expressions to provide more control over the data returned.

FLWOR is an acronym for For, Let, Where, Order by and Return, which are the keywords used for statements in the query definition. The For statement is used to define a set of nodes to iterate through when evaluating the expression (i.e. collection, document, or specified part of a document or collection of documents). The Let statement is used to bind XML document nodes or other data to a variable, for programming convenience and more efficient evaluation. Note that multiple For and Let statements can be used in a single query, and that Let statements can precede For statements unless they reference nodes in the iteration. The Where and Order by statements restrict nodes based on specified criteria and control ordering of the results, resp., similarly to their SQL counterparts. The Return statement, which is the only statement that is actually required in any XQuery, specifies the structure of the output document. XQueries can return an XML fragment, XML document, HTML, or plain text depending on the query implementation. Specific examples from the ProjectDB development effort are described below, and many more examples are provided in the XQuery Wikibook online (http://en.wikibooks.org/wiki/XQuery) and various XQuery references (Walmsley, 2007).

ProjectDB Examples

A number of XQuery files were written during and after the ProjectDB development workshop to support the targeted web services. These files are stored in the LNO eXist database and archived in the LNO SVN server, and are all available for review and customization (see http://intranet.lternet.edu/im/project/LTERProjectDatabase/userguide for more information). These queries range from simple path queries to complex multi-parameter searches with derived return documents, and all contain code documentation and comments.

For example, getProjectById.xql searches all research project documents and retrieves the document with an lter:researchProject id attribute matching a specified id parameter, using the FLWOR expression below (abbreviated for illustration):

```
 xquery version "1.0"; let $id:= request:get-
parameter("id",””) for $researchProject in
collection('/db/projects/data')/lter:researchProject[@i
d = $id] return $researchProject
```

In this query, the `let $id`… statement binds the variable $id to the query input parameter `$id`, defaulting to “” (empty string) if no id parameter is provided as input. Note that `request:get-parameter` is an eXist function used to interact with the web server session to retrieve parameters from the HTTP request. The `for $researchProject` … statement iterates through the collection of documents specified by the path statement, binding nodes from each iteration to the variable `$researchProject` . The XPath `collection` function is used in this query to search across all XML documents in the corresponding collection in the eXist database. Note that in this query, `[@id = $id]` defines a restriction in the XPath, so that only documents with an id attribute in `lter:researchProject` matching the input “id” will be returned. The `return` … statement simply references the `$researchProject` variable, resulting in the entire node set (research project document in this case) being returned without modification as output.

A more complex example is provided by getProjectsByKeyword.xql, which returns a derived XML document summarizing content from all research project documents matching site and keyword search parameters:

```
 xquery version "1.0";  (: get input parameters from
http request :) let $keyword := request:get-
parameter("keyword","") let $keywordSet := request:get-
parameter("keywordSet","") let $siteId := request:get-
parameter("siteId", "")  return  if (string-
length($keyword) > 0) then ( <projects> {   for $p in
collection(concat('/db/projects/data/',   lower-
case($siteId)))/lter:researchProject      let $title :=
$p/title/text()     let $idstr := $p/@id     let $time
:= $p/coverage/temporalCoverage      let $kw :=
if(string-length($keywordSet)>0)      then
$p/keywordSet[@name=$keywordSet]      else
$p/keywordSet      where
matches($kw/keyword,$keyword,'i')      order by
$idstr      return      <project
id="{$idstr}">     <title>{$title}</title>      {for $c
in $p/creator     let $individual :=
$c/individualName     let $userid :=
$c/userId     return     <creator>{$individual}{$userid
}</creator>}      {for $ap in
$p/associatedParty     let $ap_name :=
$ap/individualName     let $ap_id := $ap/userId     let
$role :=
$ap/role     return     <associatedParty>{$ap_name}{$ap
_id}{$role}     </associatedParty>}      <keywordSet>{f
or $k in $p/keywordSet/keyword     let $kwd :=
$k/text()     return     <keyword>{$kwd}</keyword>}
 </keywordSet>      {$time}     </project> } </projects
>) else (<projects/>)
```

In this query, three input parameters are supported ("site", "keywordSet" and "keyword"), but only the "keyword" parameter is required as dictated by the if-then-else logic and empty string test. The "site" parameter is used in the XPath to restrict the `for $p in …` statement to a specific site collection in the LNO eXist database, by concatenating the `$site` string to the base collection path (i.e. because a separate collection in `/db/projects/data` was created for each participating site during the workshop, e.g. `/db/projects/data/cap`). If the "site" parameter is omitted or blank a double slash appears in the path, effectively removing the site restriction so that all documents in `/db/projects/data` are searched (i.e. because in XPath syntax, a double slash denotes a child node at any depth level below the parent node).

Following the `for $p in …` statement that sets up the main iteration, several let statements are defined to bind nodes to variables for use in generating the output document (`$title`, `$idstr`, `$time`). The statement

```
 let $kw := if(string-length($keywordSet)>0) then
$p/keywordSet[@name=$keywordSet] else $p/keywordSet
```

employs if-then-else logic to bind `$kw` to a different node-set depending on whether a `keywordSet` was specified as input or not (i.e. based on non-zero string length of `$keywordSet`). If a `keywordSet` was specified as input then a restriction is included in the path, otherwise no restriction is used so `keywords` in any `keywordSet` will be searched.

The statement `where matches($kw/keyword,$keyword,'i')` applies the main keyword match restriction. Only documents with keyword text nodes matching the input parameter `keyword`, based on a case-insensitive regular expression match (i.e. indicated by the `i` flag), will be returned from the query. The `order by $idstr` statement then specifies that documents should be returned ordered by `researchProject` id.

The code in the return statement defines the structure for the query results, in this case an XML document that contains a subset of content from each matched project document in a separate `<project>` element, nested within a `<projects>` root element. Variables from the `for` and `let` statements in the XQuery are denoted in the return schema by curly braces, and these variables are combined with direct XML element constructors (i.e. literal XML tags) to generate the desired output. Note that additional XQueries are embedded in the return section, also denoted by curly braces, to generate nested elements containing a specific subset of the elements in the original documents (e.g. portions of the creator and associatedParty nodes). In contrast, other nodes (e.g. `coverage/temporalCoverage` denoted by `{$time}`) are copied intact to the output document.

The syntax for running these two example XQueries in the LNO eXist database using the REST API is as follows:

getProjectById.xql:http://amble.lternet.edu:8080/exist/rest/db/projects/util/xquery/getProj...

getProjectsByKeyword.xql:http://amble.lternet.edu:8080/exist/rest/db/projects/util/xquery/getProj...?keyword=Temperature

Note that the queries stored in eXist are configured to return results using an XHTML doctype without an XML declaration for broadest web browser compatibility, but viewing the page source will display the XML output. Also note that query results can be styled on the server if an XSL URL is specified using the "_xsl" parameter, as follows:

http://amble.lternet.edu:8080/exist/rest/db/projects/util/xquery/getProj...

When the XSL file is stored in an collection in the same eXist database as the XQuery, as in this case, the relative database path to the XSL can be used; however, a fully-qualified URL for an external XSL file can also be specified if XSL files are stored on a web server and not in eXist. More complex XSL stylesheets (e.g. containing import directives, parameters, and other advanced options) can also be referenced to display query results in a complete web site template, as in the following GCE example:

http://amble.lternet.edu:8080/exist/rest/db/projects/util/xquery/getProj...

Concluding Remarks

The eXist XML database and XQuery language are powerful tools for working with XML, but they are also relatively easy to use making them well suited to rapid application development and deployment. The simple installation, low resource requirements, and low administrative overhead of eXist also makes it a very practical tool for use in local LTER site information systems. In informal performance testing, queries of thousands of complex research project and EML documents in eXist took only seconds to run, indicating that this database should scale well for production use.

Given these strengths, the question of how eXist compares to Metacat naturally arises. As mentioned in the introduction, Metacat is a hybrid XML-RDBMS system, making it inherently more complex to install and administer than eXist and requiring far more system resources. The back-end RDBMS (e.g. Oracle, Postgres, etc.), a Java Servlet engine (Apache Tomcat) and the Metacat Java software need to be installed, configured and administered separately. In addition, the pathquery syntax used to query the database is unique to Metacat (see http://knb.ecoinformatics.org/software/metacat/metacatquery.html), and considerably less flexible than XQuery. The overall structure of the return document is also fairly rigid, generally requiring an extra XSL transformation step to produce useful output. Lastly, Metacat relies on versioned document ids for managing files and does not support collections or other organizational structures, requiring much more planning and discipline when storing documents and limiting options for restricting searches.

I believe that these characteristics limit the utility of Metacat as a general repository for XML and other files needed to build web applications and web services, and make it less suitable for collaborative application development than eXist (e.g. the LTER Research Projects database). However, Metacat provides very important benefits over eXist for archiving XML documents, including versioning, strong validation, replication, interoperability with other relevant EcoInformatics software (EcoGrid, Morpho, Kepler, SRB) and services (MapServer, LSID), plus support for automated document harvesting. Consequently, Metacat is a superior database for long-term storage of EML metadata and other authoritative XML documents in LTER and other environmental science networks..

For information about all the XQuery-based web services deployed for the ProjectDB database, please visit the LTER Research Project Database page on the LTER Information Management website (Gries et al.).

References

Walsh, J. and Downing, J. 2008. PROJECTDB – Planning and Development of a Collaborative Programming Effort. LTER Databits – Information Management Newsletter of the Long Term Ecological Research Network. Fall 2008 Issue. (http://databits.lternet.edu/node/33)

Gries, C., O'Brien, M., Porter, J., Sheldon, W., Walsh, J. and Bohm, S. Documentation for the LTER Research Project Database. LTER Information Management Website. (http://intranet.lternet.edu/im/project/LTERProjectDatabase/documentation)

Walmsley, P. 2007. XQuery. O'Reilly Media, Inc. Sebastopol, California. 491pp.