XQuery for fun and profit

I've always liked the idea of being able to serve serialized content on-demand whenever a user requests it. It wasn't until I read about XQuery that I started thinking about it being possible.

In this post I will explore XQuery, eXistdb as a specialized XML database and one possible way to serve serialized content created with Docbook XML, an XML vocabulary created for technical documentation, that has also been used to create books and other types of content.

Introducing XQuery

XQuery is a query language for structured and semi structured data. It uses a syntax similar to SQL with added tweak that we can use it with HTML tags to build our content.

Using the following XML content:

```
<catalog>
  <product dept="WMN">
    <number>557</number>
    <name language="en">Fleece Pullover</name>
    <colorChoices>navy black</colorChoices>
  </product>
  <product dept="ACC">
    <number>563</number>
    <name language="en">Floppy Sun Hat</name>
  </product>
  <product dept="ACC">
    <number>443</number>
    <name language="en">Deluxe Travel Bag</name>
  </product>
  <product dept="MEN">
    <number>784</number>
    <name language="en">Cotton Dress Shirt</name>
    <colorChoices>white gray</colorChoices>
```

```
    <desc>Our <i>favorite</i> shirt!</desc>
  </product>
</catalog>
```

We can query the XML to insert the resulting content into a `ul` element.

```
<ul>{
  for $prod in doc("catalog.xml")/catalog/product
  where $prod/@dept='ACC'
  order by $prod/name
  return $prod/name
}</ul>
```

And produces the following HTML/XML content.

```
<ul>
  <name language="en">Deluxe Travel Bag</name>
  <name language="en">Floppy Sun Hat</name>
</ul>
```

Rather than go through an entire XQuery Syntax Course, I'll cover one basic syntax construct, the FLOWR expression and any additional syntax as needed.

```
for $prod in doc("catalog.xml")//product
let $prodDept := $prod/@dept
where $prodDept = "ACC" or $prodDept = "WMN"
return $prod/name
```

The basic structur of a FLOWR expression has the following elements:

for

This clause sets up an iteration through the product elements returned by the path expression. The variable `$prod` is bound, in turn, to each product in the sequence. The rest of the FLWOR is evaluated once for each product, in this case, four times

let
: This clause binds the `$prodDept` variable to the value of the dept attribute

where
: This clause selects elements whose dept attribute is equal to ACC or WMN.

return
: This clause returns the name child of each of the three product elements that pass the where clause.

The first clause in a FLWOR must be a `for`, `let`, or `window` clause. After that, any number of any of the clauses listed may appear, in any order. The final clause is the required `return` clause.

The `for` clause

The `for` clause sets up an iteration that allows the rest of the FLWOR to be evaluated multiple times, once for each item in the sequence returned by the expression after the in keyword. This sequence, also known as the binding sequence, can evaluate to any sequence of zero, one, or more items. In our example, it was a sequence of product elements, but it could also be atomic values, attribute nodes, or indeed items of any kind, or a mixture of items. If the binding sequence is the empty sequence, the rest of the FLWOR is simply not evaluated (it iterates zero times).

You can use multiple for clauses in a FLWOR, which is similar to nested loops in a programming language. The result is that the rest of the FLWOR is evaluated for every combination of the values of the variables.

The `let` clause

A let clause is a convenient way to bind a variable to a value. Unlike a for clause, a let clause does not result in iteration; it binds the whole sequence to the variable rather than binding each item in turn. The let clause serves as a programmatic convenience that avoids repeating the same expression multiple times. With some implementations, it may improve performance, because the expression is evaluated only once instead of each time it is needed.

One or more let clauses can be intermingled with one or more for clauses. Each of the let and for clauses may reference a variable bound in any previous clause.

The `where` clause

The where clause is used to specify criteria that filter the results of the FLWOR.

Starting in XQuery 3.0, it is also possible to have multiple where clauses in the same FLWOR. In previous versions, only one was allowed.

Note that when using paths within the where clause, they need to start with an expression that sets the context. For example, it has to say $prod/number > 100 rather than just number > 100. Otherwise, the processor does not know where to look for the number child.

The effective boolean value of the where clause is calculated. This means that if the where clause evaluates to a Boolean value false a zero-length string, the number 0 or NaN, or the empty sequence it is considered false, and the return clause of the FLWOR is not evaluated for that iteration.

The `return` clause

The return clause consists of the return keyword followed by the single expression that is to be returned. It is evaluated once for each iteration, assuming the where clause evaluated to true. The result value of the entire FLWOR is a sequence of items returned by each evaluation of the return clause.

Further Reading

For more thorough coverage of the subject I'll refer you to Priscilla Walmsley's XQuery (2nd Edition) from O'Reilly.

The project

The idea for this project is to test if we can create custom serialized content on demand using XQuery and eXists-DB a nosql database that provides tools to create XML based web applications.

The hypothesis for the project are:

- I've defined multiple use cases for the tool
- I can create a database of XML serialized stories one of the following:
    - Docbook

        - TEI
        - XHTML as defined in the HTML specification
        - A custom vocabulary
    - I can run queries against the database to generate tailored XML documents containing the body of all documents matching the query
    - I can convert those documents to HTML and PDF using XML technologies

Use Cases

These are the use cases I've come up with.

- Course packets of supplementary material for college courses
- Serial magazines for short story content. Think Analog, Isaac Asimov or Ellery Queen Mystery magazines
- Material updated periodically where we can build updated copies based on search results

Which Syntax?

Which syntax to use depends on a few things:

- Are we starting from scratch?
- What format is our legacy content in?
- How easy are the transformation processes?
- Do we have an existing workflow to process content?

Because we're starting from scratch I will work with a customized version of Docbook with an additional metadata set to provide the custom metadata we'll search by.

Query interfaces

eXist provides full application development tools, including front-end and server. We'll leverage these tools to build the UI for the application.

Format Conversion

One of the reasons why I chose Docbook is that it has a powerful set of XSLT stylesheets to convert XML documents to both HTML(using Saxon or XSLTProc) and PDF (using Apache FOP or AntennaHouse) without having to create new

stylesheets. It should also be possible to create Paged Media stylesheets but I wonder if the effort is worth it when you already have FOP alternatives.

Setting up eXist

The rest of the document assumes you've installed eXist for your platform.

I'm also assuming that you've prepared your editor to work with eXist. I'm using oXygen XML.