Computer Science 4411/9538
Tutorial Notes on Galax
with a tiny bit on Altova Spy

These notes should be read with the powerpoint lecture notes on XML, XPath and XQuery

There is a wiki book on XQuery, accessible at:
http://en.wikibooks.org/wiki/XQuery

XML definitions and documents can be found on the w3c web site: http://www.w3.org

These examples have been run using a system called Galax.
Galax is installed on the gaul network (obelix), and also on the linux machines:
lxg-001.gaul.csd.uwo.ca through lxg-024.gaul.csd.uwo.ca
hack.gaul.csd.uwo.ca
joust.gaul.csd.uwo.ca
pong.gaul.csd.uwo.ca
rogue.gaul.csd.uwo.ca
tron.gaul.csd.uwo.ca
zork.gaul.csd.uwo.ca
Students connecting from home should SSH to obelix.gaul.csd.uwo.ca. You can either work directly on obelix, or you can SSH from there to one of the MC 10 systems, if you prefer to work on Linux.

Galax can be downloaded from http://galax.sourceforge.net. You could download your own version from this web site (the linux version is a binary).
There is a 30 day free trial of Altova Spy available at www.altova.com.

There are other free downloadable XQuery engines mentioned on the www.w3.org/XML/Query website: scroll down to the bottom.

General structure of XQuery

In general, XQuery's queries consist of a prologue (i.e. a preamble) and a query.
The prologue is supposed to provide namespace definitions and can be used to define function definitions.
In our Galax examples, the prologue is in one file known as the context, and the query in another.

Things to Note about Galax

- The examples in this tutorial were run on galax with a prologue or context file. The prologue tells Galax where to find the file or files with the data in it.

  A query can refer to one or more documents.

- It is also possible to put the doc("path to the data") function call right in the XQuery, which also works with other implementations of XQuery.

- You do not need an XML Schema to define the documents you wish to query.

- Comments in XQuery are delimited by

  ```
  (:     :)
  ```

- Error messages are very cryptic.

File Extensions

.xsd an XML schema file
.xml an XML document file
.xq an XML query (or prologue in Galax)

Examples of XPATH and XQuery

For these examples, I took the "relational" usecase, and put the file rel_context.xq in a directory, with a /docs subdirectory containing the files bids.xml, items.xml, users.xml. Here is rel_context.xq:

```
(: ----------------------------------------
    Use Case "R" : Access to Relational Data
   ---------------------------------------- :)

declare variable $users := doc("docs/users.xml");
declare variable $items := doc("docs/items.xml");
declare variable $bids  := doc("docs/bids.xml");

declare function local:bid_summary() as element()*
{
     for $i in distinct-values($bids//itemno)
     let $b := $bids//bid_tuple[itemno = $i]
     return
         <bid_count>
             <itemno>{ $i }</itemno>
             <nbids>{ count($b) }</nbids>
         </bid_count>
};
```

The file users.xml is:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<users>
  <user_tuple>
    <userid>U01</userid>
    <name>Tom Jones</name>
    <rating>B</rating>
  </user_tuple>
  <user_tuple>
    <userid>U02</userid>
    <name>Mary Doe</name>
    <rating>A</rating>
  </user_tuple>
  <user_tuple>
    <userid>U03</userid>
    <name>Dee Linquent</name>
    <rating>D</rating>
  </user_tuple>
  <user_tuple>
    <userid>U04</userid>
    <name>Roger Smith</name>
    <rating>C</rating>
  </user_tuple>
  <user_tuple>
    <userid>U05</userid>
    <name>Jack Sprat</name>
    <rating>B</rating>
  </user_tuple>
  <user_tuple>
    <userid>U06</userid>
    <name>Rip Van Winkle</name>
    <rating>B</rating>
  </user_tuple>
</users>
```

The file bids.xml is:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<bids>
  <bid_tuple>
    <userid>U02</userid>
    <itemno>1001</itemno>
    <bid>35</bid>
    <bid_date>99-01-07</bid_date>
  </bid_tuple>
  <bid_tuple>
    <userid>U04</userid>
    <itemno>1001</itemno>
    <bid>40</bid>
    <bid_date>99-01-08</bid_date>
  </bid_tuple>
  <bid_tuple>
    <userid>U02</userid>
    <itemno>1001</itemno>
    <bid>45</bid>
    <bid_date>99-01-11</bid_date>
  </bid_tuple>
  <bid_tuple>
    <userid>U04</userid>
    <itemno>1001</itemno>
    <bid>50</bid>
    <bid_date>99-01-13</bid_date>
  </bid_tuple>
  <bid_tuple>
    <userid>U02</userid>
    <itemno>1001</itemno>
    <bid>55</bid>
    <bid_date>99-01-15</bid_date>
  </bid_tuple>
  <bid_tuple>
    <userid>U01</userid>
    <itemno>1002</itemno>
    <bid>400</bid>
    <bid_date>99-02-14</bid_date>
  </bid_tuple>
  <bid_tuple>
    <userid>U02</userid>
    <itemno>1002</itemno>
    <bid>600</bid>
    <bid_date>99-02-16</bid_date>
  </bid_tuple>
  <bid_tuple>
```

```
  <userid>U03</userid>
  <itemno>1002</itemno>
  <bid>800</bid>
  <bid_date>99-02-17</bid_date>
</bid_tuple>
<bid_tuple>
  <userid>U04</userid>
  <itemno>1002</itemno>
  <bid>1000</bid>
  <bid_date>99-02-25</bid_date>
</bid_tuple>
<bid_tuple>
  <userid>U02</userid>
  <itemno>1002</itemno>
  <bid>1200</bid>
  <bid_date>99-03-02</bid_date>
</bid_tuple>
<bid_tuple>
  <userid>U04</userid>
  <itemno>1003</itemno>
  <bid>15</bid>
  <bid_date>99-01-22</bid_date>
</bid_tuple>
<bid_tuple>
  <userid>U05</userid>
  <itemno>1003</itemno>
  <bid>20</bid>
  <bid_date>99-02-03</bid_date>
</bid_tuple>
<bid_tuple>
  <userid>U01</userid>
  <itemno>1004</itemno>
  <bid>40</bid>
  <bid_date>99-03-05</bid_date>
</bid_tuple>
<bid_tuple>
  <userid>U03</userid>
  <itemno>1007</itemno>
  <bid>175</bid>
  <bid_date>99-01-25</bid_date>
</bid_tuple>
<bid_tuple>
  <userid>U05</userid>    <itemno>1007</itemno>
  <bid>200</bid>
  <bid_date>99-02-08</bid_date>
</bid_tuple>
```

```
  <bid_tuple>
    <userid>U04</userid>
    <itemno>1007</itemno>
    <bid>225</bid>
    <bid_date>99-02-12</bid_date>
  </bid_tuple>
</bids>
```

And items.xml is:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<items>
  <item_tuple>
    <itemno>1001</itemno>
    <description>Red Bicycle</description>
    <offered_by>U01</offered_by>
    <start_date>99-01-05</start_date>
    <end_date>99-01-20</end_date>
    <reserve_price>40</reserve_price>
  </item_tuple>
  <item_tuple>
    <itemno>1002</itemno>
    <description>Motorcycle</description>
    <offered_by>U02</offered_by>
    <start_date>99-02-11</start_date>
    <end_date>99-03-15</end_date>
    <reserve_price>500</reserve_price>
  </item_tuple>
  <item_tuple>
    <itemno>1003</itemno>
    <description>Old Bicycle</description>
    <offered_by>U02</offered_by>
    <start_date>99-01-10</start_date>
    <end_date>99-02-20</end_date>
    <reserve_price>25</reserve_price>
  </item_tuple>
  <item_tuple>
    <itemno>1004</itemno>
    <description>Tricycle</description>
    <offered_by>U01</offered_by>
    <start_date>99-02-25</start_date>
    <end_date>99-03-08</end_date>
    <reserve_price>15</reserve_price>
  </item_tuple>
  <item_tuple>
    <itemno>1005</itemno>
```

```
    <description>Tennis Racket</description>
    <offered_by>U03</offered_by>
    <start_date>99-03-19</start_date>
    <end_date>99-04-30</end_date>
    <reserve_price>20</reserve_price>
  </item_tuple>
  <item_tuple>
    <itemno>1006</itemno>
    <description>Helicopter</description>
    <offered_by>U03</offered_by>
    <start_date>99-05-05</start_date>
    <end_date>99-05-25</end_date>
    <reserve_price>50000</reserve_price>
  </item_tuple>
  <item_tuple>
    <itemno>1007</itemno>
    <description>Racing Bicycle</description>
    <offered_by>U04</offered_by>
    <start_date>99-01-20</start_date>
    <end_date>99-02-20</end_date>
    <reserve_price>200</reserve_price>
  </item_tuple>
  <item_tuple>
    <itemno>1008</itemno>
    <description>Broken Bicycle</description>
    <offered_by>U01</offered_by>
    <start_date>99-02-05</start_date>
    <end_date>99-03-06</end_date>
    <reserve_price>25</reserve_price>
  </item_tuple>
</items>
```

Some XPath examples

to run XQuery, we type (on one of the linux boxes) (on gaul, it is in /usr/local/Galax)

```
galax-run -context context_file.xq query_file.xq
```

or

```
/usr/local/Galax/bin/galax-run -context context_file.xq
  query_file.xq
```

and the answer comes out on the screen.
Or you can set your PATH variable in Unix.

If the .xq file or data file contains any errors, you will be told at this point.

1. If the query file contains:

   ```
   $users
   ```

   this returns the whole users file.

2. `$users//userid`

   returns

   ```
   <userid>U01</userid>,
   <userid>U02</userid>,
   <userid>U03</userid>,
   <userid>U04</userid>,
   <userid>U05</userid>,
   <userid>U06</userid>
   ```

   note the commas

3. `$items//item_tuple/description[contains(.,"Bicycle")]`

   is an example of a path expression with a comparison. . refers to "this node". It returns just the description elements containing the word "Bicycle".

```
<description>Red Bicycle</description>,
<description>Old Bicycle</description>,
<description>Racing Bicycle</description>,
<description>Broken Bicycle</description>
```

4. `$items//item_tuple[contains(description,"Bicycle")]`

   returns the item_tuples where the description contains "Bicycle"

```
<item_tuple>
    <itemno>1001</itemno>
    <description>Red Bicycle</description>
    <offered_by>U01</offered_by>
    <start_date>1999-01-05</start_date>
    <end_date>1999-01-20</end_date>
    <reserve_price>40</reserve_price>
  </item_tuple>,
<item_tuple>
    <itemno>1003</itemno>
    <description>Old Bicycle</description>
    <offered_by>U02</offered_by>
    <start_date>1999-01-10</start_date>
    <end_date>1999-02-20</end_date>
    <reserve_price>25</reserve_price>
  </item_tuple>,
<item_tuple>
    <itemno>1007</itemno>
```

```
    <description>Racing Bicycle</description>
    <offered_by>U04</offered_by>
    <start_date>1999-01-20</start_date>
    <end_date>1999-02-20</end_date>
    <reserve_price>200</reserve_price>
  </item_tuple>,
<item_tuple>
    <itemno>1008</itemno>
    <description>Broken Bicycle</description>
    <offered_by>U01</offered_by>
    <start_date>1999-02-05</start_date>
    <end_date>1999-03-06</end_date>
    <reserve_price>25</reserve_price>
  </item_tuple>
```

5. You can use 2 sources in an XPath expression, but only the primary one is returned.

   ```
   $users//user_tuple[userid = $bids//bid_tuple/userid]
   ```

   returns:

   ```
   <user_tuple>
     <userid>U01</userid>
     <name>Tom Jones</name>
     <rating>B</rating>
   </user_tuple>,
   <user_tuple>
     <userid>U02</userid>
     <name>Mary Doe</name>
     <rating>A</rating>
   </user_tuple>,
   ```

```
<user_tuple>
  <userid>U03</userid>
  <name>Dee Linquent</name>
  <rating>D</rating>
</user_tuple>,
<user_tuple>
  <userid>U04</userid>
  <name>Roger Smith</name>
  <rating>C</rating>
</user_tuple>,
<user_tuple>
  <userid>U05</userid>
  <name>Jack Sprat</name>
  <rating>B</rating>
</user_tuple>
```

(user U06 has no bids).

6. To just get userids:

```
$users//user_tuple/userid[. = $bids//bid_tuple/userid]
```

returns:

```
<userid>U01</userid>,
<userid>U02</userid>,
<userid>U03</userid>,
<userid>U04</userid>,
<userid>U05</userid>
```

Some XQuery examples

The difference between for and let:

1. The following query:

   ```
   let $u := $users//userid
   where $u[contains(.,"5")]
   return <output> {$u} </output>
   ```

   returns

   ```
   <output>
     <userid>U01</userid>
     <userid>U02</userid>
     <userid>U03</userid>
     <userid>U04</userid>
     <userid>U05</userid>
     <userid>U06</userid>
   </output>
   ```

2. whereas

   ```
   for $u in $users//userid
   where $u[contains(.,"5")]
   return <output> {$u} </output>
   ```

   gives

   ```
   <output><userid>U05</userid></output>
   ```

3. More illustrative version of "let":

```
let $u := $users//userid
return<output> {$u} </output>
```

gives:

```
<output>
  <userid>U01</userid>
  <userid>U02</userid>
  <userid>U03</userid>
  <userid>U04</userid>
  <userid>U05</userid>
  <userid>U06</userid>
</output>
```

This, however, uses the Let statement to bind a variable to a string, as with an assignment statement in a programming language:

```
let $u := $users//userid[contains(.,"5")]
return<output> {$u} </output>
```

and this gives

```
<output><userid>U05</userid></output>
```

4. More illustrative version of "for":

```
for $u in $users//userid
return <output> {$u} </output>
```

```
<output><userid>U01</userid></output>,
<output><userid>U02</userid></output>,
<output><userid>U03</userid></output>,
<output><userid>U04</userid></output>,
<output><userid>U05</userid></output>,
<output><userid>U06</userid></output>
```

5.
```
for $u in $users//userid
for $b in $bids//bid_tuple
where $b//userid = $u
return <output> {$u} </output>
```

returns

```
<output><userid>U01</userid></output>,
<output><userid>U01</userid></output>,
<output><userid>U02</userid></output>,
<output><userid>U02</userid></output>,
<output><userid>U02</userid></output>,
<output><userid>U02</userid></output>,
<output><userid>U02</userid></output>,
<output><userid>U03</userid></output>,
<output><userid>U03</userid></output>,
<output><userid>U04</userid></output>,
<output><userid>U04</userid></output>,
<output><userid>U04</userid></output>,
<output><userid>U04</userid></output>,
<output><userid>U04</userid></output>,
```

```
<output><userid>U05</userid></output>,
<output><userid>U05</userid></output>
```

6. This query, however,

```
for $u in $users
for $b in $bids
where $b//bid_tuple/userid = $u//userid
return <output>{$u//userid}</output>
```

returns:

```
<output>
  <userid>U01</userid>
  <userid>U02</userid>
  <userid>U03</userid>
  <userid>U04</userid>
  <userid>U05</userid>
  <userid>U06</userid>
</output>
```

Why?

$u is the whole users.xml document. To get rid of U06, have to be more precise in the for statement for $u.

7. ```
   for $u in $users//user_tuple
   for $b in $bids
   where $u//userid = $b//userid
   return $u/userid
   ```

   gives:

   ```
   <userid>U01</userid>,
   <userid>U02</userid>,
   <userid>U03</userid>,
   <userid>U04</userid>,
   <userid>U05</userid>
   ```

8. Generating interesting output:

   ```
   for $u in $users//user_tuple
     for $b in $bids//bid_tuple[userid = $u//userid]
     where $u//userid = $b//userid
     return <outputuser>
       {$u//userid}
       <bidson>$b//itemno</bidson>
     </outputuser>
   ```

   gives

   ```
   <outputuser><userid>U01</userid><bidson>$b//itemno</bidson></outputuser>,
   <outputuser><userid>U01</userid><bidson>$b//itemno</bidson></outputuser>,
   <outputuser><userid>U02</userid><bidson>$b//itemno</bidson></outputuser>,
   <outputuser><userid>U02</userid><bidson>$b//itemno</bidson></outputuser>,
   <outputuser><userid>U02</userid><bidson>$b//itemno</bidson></outputuser>,
   <outputuser><userid>U02</userid><bidson>$b//itemno</bidson></outputuser>,
   <outputuser><userid>U02</userid><bidson>$b//itemno</bidson></outputuser>,
   <outputuser><userid>U03</userid><bidson>$b//itemno</bidson></outputuser>,
   <outputuser><userid>U03</userid><bidson>$b//itemno</bidson></outputuser>,
   <outputuser><userid>U04</userid><bidson>$b//itemno</bidson></outputuser>,
   <outputuser><userid>U04</userid><bidson>$b//itemno</bidson></outputuser>,
   <outputuser><userid>U04</userid><bidson>$b//itemno</bidson></outputuser>,
   <outputuser><userid>U04</userid><bidson>$b//itemno</bidson></outputuser>,
   ```
   ```

```
<outputuser><userid>U04</userid><bidson>$b//itemno</bidson></outputuser>,
<outputuser><userid>U05</userid><bidson>$b//itemno</bidson></outputuser>,
<outputuser><userid>U05</userid><bidson>$b//itemno</bidson></outputuser>
```

9. This:

```
for $u in $users//user_tuple
let $b := $bids//bid_tuple[userid = $u//userid]
where $u//userid = $b//userid
return <outputuser>
  {$u//userid}
    <bidset>
    {for $b2 in $bids//bid_tuple[userid = $u//userid]
    return <bidson>{$b2//itemno}</bidson>}
    </bidset>
</outputuser>
```

gives:

```
<outputuser>
  <userid>U01</userid>
  <bidset>
    <bidson><itemno>1002</itemno></bidson>
    <bidson><itemno>1004</itemno></bidson>
  </bidset>
</outputuser>,
<outputuser>
  <userid>U02</userid>
  <bidset>
    <bidson><itemno>1001</itemno></bidson>
    <bidson><itemno>1001</itemno></bidson>
    <bidson><itemno>1001</itemno></bidson>
    <bidson><itemno>1002</itemno></bidson>
```

```
    <bidson><itemno>1002</itemno></bidson>
  </bidset>
</outputuser>,
<outputuser>
  <userid>U03</userid>
  <bidset>
    <bidson><itemno>1002</itemno></bidson>
    <bidson><itemno>1007</itemno></bidson>
  </bidset>
</outputuser>,
<outputuser>
  <userid>U04</userid>
  <bidset>
    <bidson><itemno>1001</itemno></bidson>
    <bidson><itemno>1001</itemno></bidson>
    <bidson><itemno>1002</itemno></bidson>
    <bidson><itemno>1003</itemno></bidson>
    <bidson><itemno>1007</itemno></bidson>
  </bidset>
</outputuser>,
<outputuser>
  <userid>U05</userid>
  <bidset>
    <bidson><itemno>1003</itemno></bidson>
    <bidson><itemno>1007</itemno></bidson>
  </bidset>
</outputuser>
```

10. This leaves in U6 with no bids:

```
for $u in $users//user_tuple
return <outputuser>
  {$u//userid}
    <bidset>
    {for $b2 in $bids//bid_tuple[userid = $u//userid]
    return <bidson>{$b2//itemno}</bidson>}
    </bidset>
</outputuser>
```

11. This:

```
<answer>
{
for $u in $users//user_tuple
let $b := $bids//bid_tuple[userid = $u//userid]
where $u//userid = $b//userid
return <outputuser>
  {$u//userid}
    <bidset>
    {for $b2 in $bids//bid_tuple[userid = $u//userid]
    return <bidson>{$b2//itemno}</bidson>}
    </bidset>
</outputuser>
}
</answer>
```

makes the answer a well formed XML document

12. Example with order by:

```
for $b in $bids//bid_tuple
  for $u in $users//user_tuple
  where $u//.[userid = $b//userid]
  order by $u/userid
  return $u/userid
```

13. An example with attributes: (from an old assignment)

```
for $c in $contestants//Contestant
return $c//@Name
```

gives:

```
attribute Name {"Shania Twain"},
attribute Name {"Paul McCartney"},
attribute Name {"Denise Pelley"},
attribute Name {"Randy McCaulley"}
```

14. Comparing attribute values to element values:

```
for $c in $contestants//Contestant
let $s := $songs//Song[@SongID = $c//Performance/SongRef]
return $s//Title
```

just compares two string values

Other functions:
empty(single argument)
max, min, avg, count
distinct-values
Example:

```
for $u in $users//userid
return distinct-values($u)
```

```
xdt:untypedAtomic("U01"),
xdt:untypedAtomic("U02"),
xdt:untypedAtomic("U03"),
xdt:untypedAtomic("U04"),
xdt:untypedAtomic("U05"),
xdt:untypedAtomic("U06")
```

```
for $b in $bids//userid
return distinct-values($b)
```

```
xdt:untypedAtomic("U02"),
xdt:untypedAtomic("U04"),
xdt:untypedAtomic("U02"),
xdt:untypedAtomic("U04"),
xdt:untypedAtomic("U02"),
xdt:untypedAtomic("U01"),
xdt:untypedAtomic("U02"),
xdt:untypedAtomic("U03"),
xdt:untypedAtomic("U04"),
xdt:untypedAtomic("U02"),
xdt:untypedAtomic("U04"),
xdt:untypedAtomic("U05"),
xdt:untypedAtomic("U01"),
xdt:untypedAtomic("U03"),
xdt:untypedAtomic("U05"),
xdt:untypedAtomic("U04")
```

```
for $b in $bids
return distinct-values($b//userid)
```

```
xdt:untypedAtomic("U02"),
xdt:untypedAtomic("U04"),
xdt:untypedAtomic("U01"),
xdt:untypedAtomic("U03"),
xdt:untypedAtomic("U05")
```

There is an if..then..else which one could use within the return clause.
Can also say "where some ... in ... satisfies (predicate)"
or "where every ... in ... satisfies (predicate)"
One would use the "where every" construct to simulate the universal quantifier.

Final Points

1. There are different ways of connecting to the document with the doc function. You can just put doc("users.xml") right in the for or let statement (assuming users.xml is in the same directory from which you are running. As set up in the examples, users.xml in in a subdirectory called docs, so it would be doc("docs/users.xml")). You can alternatively put a declare variable statement at the begining of the

query, followed by a semicolon.

2. Altova Spy does not put “,” ’s between output elements when the output has no root tag. eXist-db does.