

Computer Science 411a/538a

Tutorial Notes on Galax

revised fall 2005

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

Galax is installed in the Gaul network in CSD, and is available on any workstation on the network.

Galax was downloaded from <http://www.galaxquery.org>

We now have version 0.5

You could download your own version to run on Windows from this web site.

XML definitions and documents can be found on the w3c web site:

<http://www.w3.org>

You can find more information in `/usr/local/Galax` on gaul, particularly look at the usecases, and files in the `/doc` directory.

General structure of XQuery

In general, XQuery's 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.

When we use Galax, we put the prologue in one file known as the context, and the query in another.

Things to Note about Galax

- 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
- The way of referring to the documents is different from the Don Chamberlain tutorial and from Altova Spy.
- You can have it validate an XML Schema with the `mapschema` command. We don't seem to need this to run queries.
- Comments in Galax 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 /doc 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>{ $i }</itemno>  
      <n bids>{ count($b) }</n bids>  
    </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>
  </bid_tuple>
</bids>
```

```

    <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 Gaul)

```
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.

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>

```

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. `$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. 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>U04</userid></output>,
<output><userid>U05</userid></output>,
<output><userid>U05</userid></output>

```

4. This query produces the same answer as just above;

```

for $u in $users
for $b in $bids
where $b//bid_tuple[userid = $u//userid]
return $b//userid

```

5. This query, however,

```

for $u in $users
for $b in $bids
where $b//bid_tuple[userid = $u//userid]
return $u//userid

```

returns:

```

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

```

Why?

\$u is ranging over \$users, i.e. the root of the users.xml document. To get rid of U06, have to be more precise in the for statement for \$u.

```

6. 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>

```



```

    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>

```

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 valid 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")
```

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)"