Student Online Teaching Advice Notice

The materials and content presented within this session are intended solely for use in a context of teaching and learning at Trinity.

Any session recorded for subsequent review is made available solely for the purpose of enhancing student learning.

Students should not edit or modify the recording in any way, nor disseminate it for use outside of a context of teaching and learning at Trinity.

Please be mindful of your physical environment and conscious of what may be captured by the device camera and microphone during videoconferencing calls.

Recorded materials will be handled in compliance with Trinity's statutory duties under the Universities Act, 1997 and in accordance with the University's policies and procedures.

Further information on data protection and best practice when using videoconferencing software is available at https://www.tcd.ie/info_compliance/data-protection/.

CSU22041: Information Management I

XQuery

… an **art of making information accessible**.

2020-2021

Gaye Stephens gaye.stephens@tcd.ie

Querying XML Documents

XQuery

What is XQuery?

- Originally focused on **retrieval** of information from XML documents
  - **Update** features added in 2011
    https://www.w3.org/TR/xquery-update-10/

- XQuery is a language for finding and extracting elements and attributes from XML documents.
- Used in conjunction with Xpath

For-Let-Where-OrderBy-Return:“FLWOR” expressions

(pronounced “FLOWER”)

1. One or more FOR and/or LET expressions
   - For gathering nodes into sets from a series of XPath queries to operate upon in other clauses
2. Optional WHERE clause
   - For filtering nodes in the sets to be operated upon in other clauses
3. Optional ORDER BY clause
   - For returning nodes in the sets in particular order in other clauses
4. RETURN clause
   - How to return the identified nodes in the sets

FOR Clause

FOR <variable> IN <xpath expression>, <xpath expression>, <xpath expression>,…

- Variable (starting with $) "binds to" **in turn each member in the set** returned by Xpath expression(s)
- For each variable binding the rest of FLOWR expression is executed
- More than one variable/path expression binding can be specified by separating with comma (,)

Example FOR Clause

XML Source

```
<?xml version="1.0"?>
<assessments>
  <student name="Smith">
      <mark thecourse="4BA5"> 99
        </mark>
      <mark thecourse="4BA1"> 75
        </mark>
  </student>
  <course name="4BA1"
          takenby="Smith,Jones">
     <mark>60</mark>
  </course>
  <course name="4BA5"
          takenby="Smith,Bond">
     <mark>70</mark>
  </course>
</assessments>
```

XQuery

```
for $j in
doc("data/tcd.xml")/assessments/co
urse
return
("Course Node:”,$j)
```

Result

```
Course Node: <course name="4BA1"
takenby="Smith,Jones">
    <mark>60</mark>
</course>
Course Node: <course name="4BA5"
takenby="Smith,Bond">
    <mark>70</mark>
</course>
```

LET Clause

- LET <variable> := <xpath expression>, <xpath expression>, …
  - Variable (starting with $) "binds to" **the set** returned by xpath expression
  - Does not iterate over set like the FOR clause does
  - It cannot be redefined within the scope of the function
  - More than one variable/path expression binding can be specified by separating with comma (,)

Example LET Clause

XML Source

```
<?xml version="1.0"?>
<assessments>
  <student name="Smith">
      <mark thecourse="4BA5"> 99
        </mark>
      <mark thecourse="4BA1"> 75
        </mark>
  </student>
  <course name="4BA1"
          takenby="Smith,Jones">
     <mark>60</mark>
  </course>
  <course name="4BA5"
          takenby="Smith,Bond">
     <mark>70</mark>
  </course>
</assessments>
```

XQuery

```
let $c:=
doc("data/tcd.xml")/assessments/co
urse/mark
return
   <list_of_avg_course_marks>
   {$c}
   </list_of_avg_course_marks>
```

Curly brackets {} are used for enclosed expressions and indicate that the expression enclosed in the return clause needs to be evaluated by the Xquery processor

Result

```
<list_of_avg_course_marks>
    <mark>60</mark>
    <mark>70</mark>
</list_of_avg_course_marks>
```

RETURN Clause

- One limitation of Xpath is that it can only operate on existing elements/attributes within the document

- XQuery allows the generation of new elements/attributes nodes
  - The element's content (if any) is either literally given between start- and end-tag, or provided as an “enclosed expression”, or as a mixture of both.
  - Curly brackets {} are used for enclosed expressions in the return clause and indicate that the expression enclosed needs to be evaluated by the Xquery processor

Example RETURN Clause

XML Source

```
<?xml version="1.0"?>
<assessments>
  <student name="Smith">
      <mark thecourse="4BA5"> 99
        </mark>
      <mark thecourse="4BA1"> 75
        </mark>
  </student>
  <course name="4BA1"
          takenby="Smith,Jones">
     <mark>60</mark>
  </course>
  <course name="4BA5"
          takenby="Smith,Bond">
     <mark>70</mark>
  </course>
</assessments>
```

XQuery

```
for $j in
        doc("data/tcd.xml")/assess
ments/course/@name
return
        <one_of_courses_is>
        {$j}
        </one_of_courses_is>
```

Example of Xquery node generation

Result

```
<one_of_courses_is name="4BA1"/>
<one_of_courses_is name="4BA5"/>
```

WHERE Clause

- Filters the binding tuples produced by the FOR and LET clauses
- If the filter expression evaluates to true then the RETURN clause is executed

Example WHERE Clause

XML Source

```
<?xml version="1.0"?>
<assessments>
  <student name="Smith">
      <mark thecourse="4BA5"> 99
        </mark>
      <mark thecourse="4BA1"> 75
        </mark>
  </student>
  <course name="4BA1"
          takenby="Smith,Jones">
     <mark>60</mark>
  </course>
  <course name="4BA5"
          takenby="Smith,Bond">
     <mark>70</mark>
  </course>
</assessments>
```

XQuery

```
for $j in
doc("data/tcd.xml")/assessments/course
where contains($j/@takenby,"Bond")
return
        <Bond_courses_is>
        {string($j/@name)}
        </Bond_courses_is>
```

Result

```
<Bond_courses_is>4BA5</Bond_courses_is>
```

Querying over several interlinked documents

XML Source
Tcd.xml

```
<?xml version="1.0"?>
<assessments>
  <student name="Smith">
      <mark thecourse="4BA5"> 99
        </mark>
      <mark thecourse="4BA1"> 75
        </mark>
  </student>
  <course name="4BA1"
takenby="Smith,Jones">
     <mark>60</mark>
  </course>
  <course name="4BA5"
           takenby="Smith,Bond">
     <mark>70</mark>
  </course>
</assessments>
```

XML Source
details.xml

```
<?xml version="1.0"?>
<studentdetails>
 <student name="Smith">
    <address> 101 Pine </address>
    <enrolled> 2001 </enrolled>
  </student>
<student name="Bond">
    <address> 007 Fleming </address>
    <enrolled> 2002 </enrolled>
  </student>
```

XQuery

```
for $w in
doc("data/details.xml")/studentdet
ails/student,
$x in
doc("data/tcd.xml")/assessments/st
udent
where $x/@name = $w/@name
return
<studentpercourse>
   {$w/@name}
   {$w/address}
   {$x/mark/@thecourse}
</studentpercourse>
```

Result

```
<studentpercourse name="Smith"
thecourse="4BA5" thecourse="4BA1">
    <address> 101 Pine </address>
</studentpercourse>
```

X Query- Sorting

- The return clause of a FLWOR expression is evaluated once for each tuple in the tuple stream, and the results of these evaluations are concatenated to form the result of the FLWOR expression.
  - If no order by clause is present, the order of the tuple stream is determined by the orderings of the sequences returned by the expressions in the for clauses.
  - If an order by clause is present, it determines the order of the tuple stream

Example ORDER BY clause

XML Source

```
<?xml version="1.0"?>
<studentdetails>
 <student name="Smith">
    <address> 101 Pine </address>
    <enrolled> 2011 </enrolled>
  </student>
<student name="Bond">
    <address> 007 Fleming
</address>
    <enrolled> 2012 </enrolled>
  </student>
```

XQuery

```
for $x in
doc("data/details.xml")/studentdet
ails/student
order by $x/enrolled descending
return
  <byyear>
   {$x}
  </byyear>
```

Result

```
<byyear>
    <student name="Bond">
        <address> 007 Fleming
</address>
        <enrolled> 2012 </enrolled>
    </student>
</byyear>
<byyear>
    <student name="Smith">
        <address> 101 Pine </address>
        <enrolled> 2011 </enrolled>
    </student>
</byyear>
```

Sequence Operations

- A union of two node sequences is a sequence containing all the nodes that occur in either of the operands.
- The intersect operator produces a sequence containing all the nodes that occur in both of its operands.
- The except operator produces a sequence containing all the nodes that occur in its first operand but not in its second operand.

Example UNION clause

XML Source
Tcd.xml

```
<?xml version="1.0"?>
<assessments>
  <student name="Smith">
      <mark thecourse="4BA5"> 99
        </mark>
      <mark thecourse="4BA1"> 75
        </mark>
  </student>
  <course name="4BA1"
takenby="Smith,Jones">
     <mark>60</mark>
  </course>
  <course name="4BA5"
           takenby="Smith,Bond">
     <mark>70</mark>
  </course>
</assessments>
```

XML Source
tcd2.xml

```
<?xml version="1.0"?>
<assessments>
  <student name="Ledwidge">
      <mark thecourse="4BA5"> 45
</mark>
      <mark thecourse="4BA1"> 55
</mark>
  </student>
  <student name="ONeill">
      <mark thecourse="4BA5"> 85
</mark>  </student> </assessments>
```

XQuery

```
<recent_good_results>
{doc("data/tcd.xml")/assessments/s
tudent[mark > 80]
union
doc("data/tcd2.xml")/assessments/s
tudent[mark > 80]}
</recent_good_results>
```

Result

```
<recent_good_results>
    <student name="Smith">
        <mark thecourse="4BA5"> 99 </mark>
        <mark thecourse="4BA1"> 75 </mark>
    </student>
    <student name="ONeill">
        <mark thecourse="4BA5"> 85 </mark>
    </student>
</recent_good_results>
```

Conditional Clauses

- if (test expression) then expression else expression
- The result of a conditional expression depends on the value of the test expression in the if clause
  - If the value of the test expression is the Boolean value true, or a sequence containing at least one node (serving as an “existence test”), the then clause is executed.
  - If the value of the test expression is the Boolean value false or an empty sequence, the else clause is executed.
- **All three clauses (if, then, and else) are required**
- Nesting of if clauses possible

Example Conditional clause

XML Source

```
<?xml version="1.0"?>
<studentdetails>
 <student name="Smith">
    <address> 101 Pine </address>
    <enrolled> 2012</enrolled>
  </student>
<student name="Bond">
    <address> 007 Fleming
</address>
    <enrolled> 2011 </enrolled>
  </student>
```

XQuery

```
for $x in
doc("data/details.xml")/studentdet
ails/student
return
 <status>
{$x/@name}
{if ($x/enrolled = 2012)
then "new student"
else if ($x/enrolled < 2008)
        then "should be finished"
else "student"}
</status>
```

Result

```
<status name="Smith">
new student</status>
 <status name="Bond">
Should be finished</status>
```

Quantified Expression

- Allows a variable to iterate over the items in a sequence.
  For each variable binding, a test expression is evaluated.
  - A quantified expression that begins with some keyword returns the value true if the test expression is true for some variable binding
  - A quantified expression that begins with every keyword, returns the value true if the test expression is true for every variable binding

Example Quantified Expression clauses

XML Source
tcd2.xml

```
<?xml version="1.0"?>
<assessments>
  <student name="Ledwidge">
      <mark thecourse="4BA5"> 45
</mark>
      <mark thecourse="4BA1"> 55
</mark>
  </student>
  <student name="ONeill">
      <mark thecourse="4BA5"> 85
</mark>  </student> </assessments>
```

XQuery

```
<results>
{if (every $m in
doc("data/tcd2.xml")/assessments/s
tudent satisfies $m/mark > 60)
then "excellent results"
else if (some $t in
doc("data/tcd2.xml")/assessments/s
tudent satisfies $t/mark > 50)
then "average results"
else "bad results"
}
</results>
```

Result

```
<results>average results</results>
```

Built-in Functions

- Over 100 XPath built-in functions
- Functions that operate on Basic Types
  - manipulation of dates, strings, numbers etc
  - E.g. fn:string-join for joining strings together
- Functions that operate on Nodes
  - E.g. fn:name() returns name of node
- Functions that operate on Sequences
  - A sequence is an ordered collection of zero or more items
  - E.g. fn:distinct-values returns sequence with all duplicates removed
- Functions that operate on Context
  - obtain information from the evaluation context
  - E.g. fn:last returns the number of items in the sequence being processed

Useful summary of InBuilt functions provided by Oracle

Function	Commentary
Math: +, -, *, div, idiv, mod, =, !=, <, >, <=, >= floor(), ceiling(), round(), count(), min(), max(), avg(), sum()	Division is done using div rather than a slash because a slash indicates an XPath step expression. idiv is a special operator for integer-only division that returns an integer and ignores any remainder.
Strings and Regular Expressions: compare(), concat(), starts-with(), ends-with(), contains(), substring(), string-length(), substring-before(), substring-after(), normalize-space(), upper-case(), lower-case(), translate(), matches(), replace(), tokenize()	compare() dictates string ordering. translate() performs a special mapping of characters. matches(), replace(), and tokenize() use regular expressions to find, manipulate, and split string values.
Date and Time: current-date(), current-time(), current-dateTime() +, -, div eq, ne, lt, gt, le, gt	XQuery has many special types for date and time values such as duration, dateTime, date, and time. On most you can do arithmetic and comparison operators as if they were numeric. The two-letter abbreviations stand for equal, not equal, less than, greater than, less than or equal, and greater than or equal.
XML node and QNames: node-kind(), node-name(), base-uri() eq, ne, is, isnot, get-local-name-from-QName(), get-namespace-from-QName() deep-equal() >>, <<	node-kind() returns the type of a node (i.e. "element"). node-name() returns the QName of the node, if it exists. base-uri() returns the URI this node is from. Nodes and QName values can also be compared using eq and ne (for value comparison), or is and isnot (for identity comparison). deep-equal() compares two nodes based on their full recursive content. The << operator returns true if the left operand preceeds the right operand in document order. The >> operator is a following comparison.
Sequences: item-at(), index-of(), empty(), exists(), distinct-nodes(), distinct-values(), insert(), remove(), subsequence(), unordered().position(), last()	item-at() returns an item at a given position while index-of() attempts to find a position for a given item. empty() returns true if the sequence is empty and exists() returns true if it's not. dictinct-nodes() returns a sequence with exactly identical nodes removed and distinct-values() returns a sequence with any duplicate atomic values removed. unordered() allows the query engine to optimize without preserving order. position() returns the position of the context item currently being processed. last() returns the index of the last item.
Type Conversion: string(), data(), decimal(), boolean()	These functions return the node as the given type, where possible. data() returns the "typed value" of the node.
Booleans: true(), false(), not()	There's no "true" or "false" keywords in XQuery but rather true() and false() functions. not() returns the boolean negation of its argument.
Input: document(), input(), collection()	document() returns a document of nodes based on a URI parameter. collection() returns a collection based on a string parameter (perhaps multiple documents). input() returns s general engine-provided set of input nodes.

User Functions

- XQuery allows users to define functions of their own
  - A function may take zero or more parameters.
  - A function definition must specify the name of the function and the names of its parameters if they exist
    - It may optionally specify types for the parameters
      - If no type is specified for a function parameter, that parameter accepts values of any type.
    - It may optionally specify types for the result of the function.
      - If no type is specified for the result of the function, the function may return a value of any type.
  - Body of the function is an expression enclosed in curly braces with a semicolon following the closing bracket.

Example Function clause

XML Source
Tcd.xml

```
<?xml version="1.0"?>
<assessments>
  <student name="Smith">
      <mark thecourse="4BA5"> 99
        </mark>
      <mark thecourse="4BA1"> 75
        </mark>
  </student>
  <course name="4BA1"
takenby="Smith,Jones">
     <mark>60</mark>
  </course>
  <course name="4BA5"
           takenby="Smith,Bond">
     <mark>70</mark>
  </course>
</assessments>
```

XML Source
tcd2002.xml

```
<?xml version="1.0"?>
<assessments>
  <student name="Ledwidge">
      <mark thecourse="4BA5"> 45
</mark>
      <mark thecourse="4BA1"> 55
</mark>
  </student>
  <student name="ONeill">
      <mark thecourse="4BA5"> 85
</mark>  </student> </assessments>
```

XQuery

```
declare function local:all_students()
   {
for $s in
doc("data/tcd.xml")/assessments/student
union
doc("data/tcd2.xml")/assessments/studen
t
return
<student>
         {$s/@name}
         {$s/mark/string(@thecourse)}
</student>
   };

<all>
{local:all_students()}
</all>
```

Result

```
<all>
    <student name="Smith">4BA5 4BA1<student/>
    <student name="Ledwidge">4BA5
4BA1</student>
    <student name="ONeill">4BA5</student>
</all>
```

Types in Queries

- Sometimes necessary to refer to a particular type in query
- One way to refer to a type is by its qualified name, or QName.
  - A QName may refer to a built-in type such as xs:integer or to a type that is defined in some schema, such as *abc:student*.
  - If the QName has a namespace prefix (the part to the left of the colon), that prefix must be bound to a specific namespace URI using the "declare namespace" clause

Example Function clause with param

XML Source
Tcd.xml

```
<?xml version="1.0"?>
<assessments>
  <student name="Smith">
      <mark thecourse="4BA5"> 99
        </mark>
      <mark thecourse="4BA1"> 75
        </mark>
  </student>
  <course name="4BA1"
takenby="Smith,Jones">
     <mark>60</mark>
  </course>
  <course name="4BA5"
          takenby="Smith,Bond">
     <mark>70</mark>
  </course>
</assessments>
```

XML Source
tcd2.xml

```
<?xml version="1.0"?>
<assessments>
  <student name="Ledwidge">
      <mark thecourse="4BA5"> 45
</mark>
      <mark thecourse="4BA1"> 55
</mark>
  </student>
  <student name="ONeill">
      <mark thecourse="4BA5"> 85
</mark>  </student> </assessments>
```

XQuery

```
declare function local:
find_students($stuname as xs:string)
   {
for $s in
doc("data/tcd.xml")/assessments/student
union
doc("data/tcd2.xml")/assessments/studen
t
where $stuname = string($s/@name)
return
<student>
         {$s/@name}
         {$s/mark/@thecourse}
</student>
   };

local:find_students("Smith")
```

Result

```
<student name="Smith" thecourse="4BA5"
thecourse="4BA1"/>
```

Interesting error in this example- see if you can find it and understand it

Types in Queries

- Another way to refer to a type is by a generic keyword such as element or attribute.
  - May optionally be followed by a QName that further restricts the name or type of the node.
  - For example,
    - element denotes any element;
    - element student denotes an element whose name is *student*;
    - element of type abc:student denotes an element whose type is *student* as declared in the namespace *abc*.
  - A reference to a type may optionally be followed by one of three occurrence indicators:
    - "*" means "zero or more";
    - "+" means "one or more,"
    - "?" means "zero or one."
    - The absence of an occurrence indicator denotes exactly one occurrence of the indicated type.

Example Function clause with output type declared

XML Source
Tcd.xml

```
<?xml version="1.0"?>
<assessments>
  <student name="Smith">
      <mark thecourse="4BA5"> 99
        </mark>
      <mark thecourse="4BA1"> 75
        </mark>
  </student>
  <course name="4BA1"
takenby="Smith,Jones">
     <mark>60</mark>
  </course>
  <course name="4BA5"
           takenby="Smith,Bond">
     <mark>70</mark>
  </course>
</assessments>
```

XML Source
tcd2.xml

```
<?xml version="1.0"?>
<assessments>
  <student name="Ledwidge">
      <mark thecourse="4BA5"> 45
</mark>
      <mark thecourse="4BA1"> 55
</mark>
  </student>
  <student name="ONeill">
      <mark thecourse="4BA5"> 85
</mark>  </student> </assessments>
```

XQuery

```
declare function local:all_students()
as element()*
   {
for $s in
doc("data/tcd.xml")/assessments/student
union
doc("data/tcd2.xml")/assessments/studen
t
return
<student>
         {$s/@name}
         {$s/mark/@thecourse}
</student>
   };

<all>
{local:all_students()}
</all>
```

Interesting error in this example- see if you can find it and understand it

Result

```
<all>
    <student name="Smith" thecourse="4BA5"
thecourse="4BA1"/>
    <student name="Ledwidge" thecourse="4BA5"
thecourse="4BA1"/>
    <student name="ONeill" thecourse="4BA5"/>
</all>
```

Part 2 eXtensible Markup Language - extract from the assignment sheet on blackboard.

1. XML and DTD documents

   a. From your group's UML Class diagram, pick <u>at least</u> 6 classes and for each create a different XML document. Include the following characteristics <u>for each</u> XML document:

      - At least 6 different XML elements/tags are used.
      - At least one third of the XML elements should have 1 XML attribute
      - Interlinks between some of the documents (reflecting the assocations/relationships between the classes within the UML design), with enough information to allow for interesting cross document XML Queries to be designed

   b. For each XML document create a DTD

2. Design and Document **at minimum 8 interesting XQuery** queries that support some of your UML use cases. **Pesent these queries during online sessions(within your groups on Blackboard).**

   - At least 3 of the queries should retrieve information from two or more interlinked XML documents, using the WHERE clause
   - At least 2 of the queries should use the FOR clause
   - At least 1 of the queries should use the LET clause
   - At least 2 of the queries should use a Built-in XQuery function
   - At least 2 of the queries should use User Defined Functions

3. Present your XML, DTD and XQueries in a **group report** which also includes the following.

   - What (if anything) did you need to change in going from UML design to XML implementation? - Include revised diagrams/ethics canvas, if appropriate.
   - List who did what in the group for XML implementation
   - Strengths and Weaknesses of the XML design and XQueries design
   - For the XML and DTD documents- Use comments to clearly state what is the purpose of the document, and comments describing purpose of each element and for each attribute, and why certain cardinality (*,+ etc.) is used.
   - For each Xquery include: identification of the UML use case that it supports, description of the purpose of the query and provide example outputs that you expect when query is executed.

Past XML Exam Question

2.

(a) Explain using examples what constitute a well formed and valid XML document.

[10 Marks]

(b) Use DTD Notation to fully describe the XML document shown in Figure A above. Provide explanation for your design decisions

[16 Marks]

(c) Define and explain XQuery Statements for each of the following queries posed over the document in Figure A. Show expected results and explain your design decisions

I. Return within a single new element called "Colleagues", all the last name values in the document separated by a "+" sign.

II. Return just the values of the "medicalregnumber"attribute in a new element called "RegNumbers"

III. Return only the first of the firstname for each Doctor in the document

[24 Marks]

[Total 50 Marks]

XML

```
<?xml version='1.0' encoding='UTF-8' ?>
<DoctorDirectory>
<doctor area="Dublin" medicalregnumber="123456">
        <name>
                <firstname>James</firstname>
                <lastname>Murphy</lastname>
        </name>
        <telephone Type ="mobile">
                <number>0871234567</number>
        </telephone>
         <telephone Type ="Landline">
                <number>014587884</number>
        </telephone>
</doctor>
<doctor area="Kildare" medicalregnumber="789112">
        <name>
                <firstname>Freda</firstname>
                <firstname>Anne</firstname>
                <lastname>Hartigan</lastname>
        </name>
        <telephone Type ="mobile">
                <number>0879922345</number>
        </telephone>
         <telephone Type ="Landline">
                <number>045865768</number>
        </telephone>
</doctor>
<doctor  medicalregnumber="223445">
        <name>
                <firstname>Francis</firstname>
                <firstname>Mary</firstname>
                <lastname>Kelly</lastname>
        </name>
         <telephone Type ="Landline">
                <number>04487994</number>
        </telephone>
</doctor>
</DoctorDirectory>
```

Figure A for exam question on previous slide

That’s All Folks Thank You for Listening