XQuery Query Language

Intro

- XQuery extends XPath.
- It is a Turing-complete language.
- It uses the same data model.
  - A document is a tree.
  - A query result is a sequence of items from the document.
- XQuery is an expression language.
  - Any XQuery expression can be an argument of any other XQuery expression.
  - This is like relational algebra and unlike SQL.

FLWOR expressions

Example:

```
let $d := fn:doc(“bank.xml”)
for $tfq in $d//TFQuestion
let $qid := $tfq/@QID
where $tfq/@answer=”True”
order by $qid
return $tfq/Text
```

FLWOR expressions

- The semantics of return is surprising:
  - It does not terminate the FLWOR expression!
  - It specifies the value produced by the current iteration.
  - The sequence of these is the result of the FLWOR expression.
- For is like

  ```
  for x in [99, 42, 101, 5]
  ```

  - It iterates over the items in a sequence.
  - Each time, the variable gets a new value.
- Let is like

  ```
  x = [99, 42, 101, 5]
  ```

  - No iteration occurs.

FLWOR expressions

- Keywords are case-sensitive.
- Variables begin with $.
- Rule: (for | let)+ where? order-by? return
- Any subexpression could itself be a FLWOR expression of other complex expression

```
let $d := fn:doc(“bank.xml”)
for $tfq in $d//TFQuestion
let $qid := $tfq/@QID
where $tfq/@answer=”True”
order by $qid
return $tfq/question
```

Branching expressions

- Form: `if («E1») then «E2» else «E3»`
- All three parts are required.
- Any expression has an "effective boolean value" (EBV), so can be an if-condition.
- Value of the if expression is
    - *E2* if the EBV of *E1* is true, and
    - *E3* if the EBV of *E1* is false.
- Example:

```
if ($q/@solution="True")
then $q/question else ()
```

Quantifier expressions

- Form: `some` *«variable»* `in` *«E1»* `satisfies` *«E2»*
- Meaning
  - Evaluate *E1*, yielding a sequence.
  - Let the variable be each item in the sequence, and evaluate *E2* for each.
  - The value of the whole expression is true if *E2* has EBV true at least once.
- Form: `every` *«variable»* `in` *«E1»* `satisfies` *«E2»*
- Meaning is analogous.

Set operator expressions

- Form:

```
«E1» union «E2»
«E1» intersect «E2»
«E1» except «E2»
```

- Meaning is analogous to SQL.
- Result does not include duplicates.
- Result appears in document order.

Nesting expressions arbitrarily

- Remember that XQuery is an expression language.
- So any subexpression can be an arbitrarily complex XQuery expression.

Mixing static output and evaluated expressions

Can construct new XML structures with our code!

```
<title>Facts about Canada</title>
<contents>
<title>Facts about Canada</title>
<truth>
{
  let $d := fn:doc("bank.xml")
  return $d//TFQuestion[@answer = "True"]/Text
}
</truth>
<lies>
{
  let $d := fn:doc("bank.xml")
  return $d//TFQuestion[@answer = "False"]/Text
}
</lies>
</contents>
```

What's evaluated and what's not?

- The default: don't evaluate.
  - Example:
    `<title>$x</title>`
  - This evaluates to a title element with value "`$x`"
- To override the default and force evaluation, surround with braces.
  - Example:
    `<title>{$x}</title>`

Return has the opposite default

- Return’s default is to evaluate the expression.
  - Example:
    ```
    return $x
    ```
- To override the default and treat the value literally, surround with quotes.
  - Example:
    ```
    return “$x”
    ```

A larger example of XQuery, in steps

Step 1

```
fn:doc("class.xml")/ClassResponses/Student/@sid
```

Step 2

```
for $student in
   fn:doc("class.xml")/ClassResponses/Student
return
   <STUDENT>
     { data($student/@sid) }
   </STUDENT>
```

Step 3

```
let $qdoc := fn:doc("quiz.xml")
for $question in $qdoc/Quiz/Question/@QID
return $question
```

Step 4

```
let $qdoc := fn:doc("quiz.xml")
let $bdoc := fn:doc("bank.xml")
let $banktfquestions := $bdoc//TFQuestion/@QID
for $question in $qdoc/Quiz/Question/@QID
where $question = $banktfquestions
return $question
```

Step 5

```
let $qdoc := fn:doc("quiz.xml")
let $bdoc := fn:doc("bank.xml")
let $cdoc := fn:doc("class.xml")
let $banktfquestions := $bdoc//TFQuestion/@QID
let $quiztfquestions :=
   for $question in $qdoc/Quiz/Question/@QID
   where $question = $banktfquestions
   return $question
for $student in $cdoc/ClassResponses/Student
where true()
return
   <STUDENT>
     { data($student/@sid) }
   </STUDENT>
```

Step 6

```
(: Three lets to parse the 3 docs. :)
let $banktfquestions := $bdoc//TFQuestion/@QID
let $quiztfquestions :=
   for $question in $qdoc/Quiz/Question/@QID
   where $question = $banktfquestions
   return $question
for $student in $cdoc/ClassResponses/Student
let $studentanswered :=
   $student/QuestionResponse/@QID
where every $q in $quiztfquestions
   satisfies $q = $studentanswered
return
   <STUDENT>
     { data($student/@sid) }
   </STUDENT>
```

Step 7 . . .

```
<COMPLETE>
{
  (: Three lets to parse the 3 docs. :)
  let $banktfquestions := $bdoc//TFQuestion/@QID
  let $quiztfquestions :=
     for $question in $qdoc/Quiz/Question/@QID
     where $question = $banktfquestions
     return $question
  for $student in $cdoc/ClassResponses/Student
  let $studentanswered :=
     $student/QuestionResponse/@QID
  where every $q in $quiztfquestions
     satisfies $q = $studentanswered
  return
     <STUDENT>
       { data($student/@sid) }
     </STUDENT>
}
</COMPLETE>
```