Advanced Databases

Querying XML - XQuery

Dr. George Mertzios
Michaelmas Term

george.mertzios@durham.ac.uk
Room 2066, MCS Building
Tel: 42 429

Querying XML data

How would you query a directed tree?

- A common approach: define a template describing traversals from the root
  - simple navigation through the tree
- XPath: the basis of this template
  - navigates through the tree to select data
- XQuery: the complete XML query language
  - "the SQL of XML"
  - selects / combines data (using XPath) and constructs output

XPath and XQuery

- XQuery (for XML, i.e. semi-structured data):
  - is based on XPath
  - is similar to SQL (for structured data)
- The result of a path expression:
  - ordered list of nodes
  - including their descendant nodes

    e.g.: `/bib/book/year`
  - may contain duplicates:
    - i.e. multiple nodes with the same content

XPath and XQuery

- XQuery (for XML, i.e. semi-structured data):
  - is based on XPath
  - is similar to SQL (for structured data)

XQuery basic form:

FLWR ("Flower") Expressions

FOR ...
LET ...
WHERE ...
RETURN ...

SQL basic form:

SELECT ...
FROM ...
WHERE ...

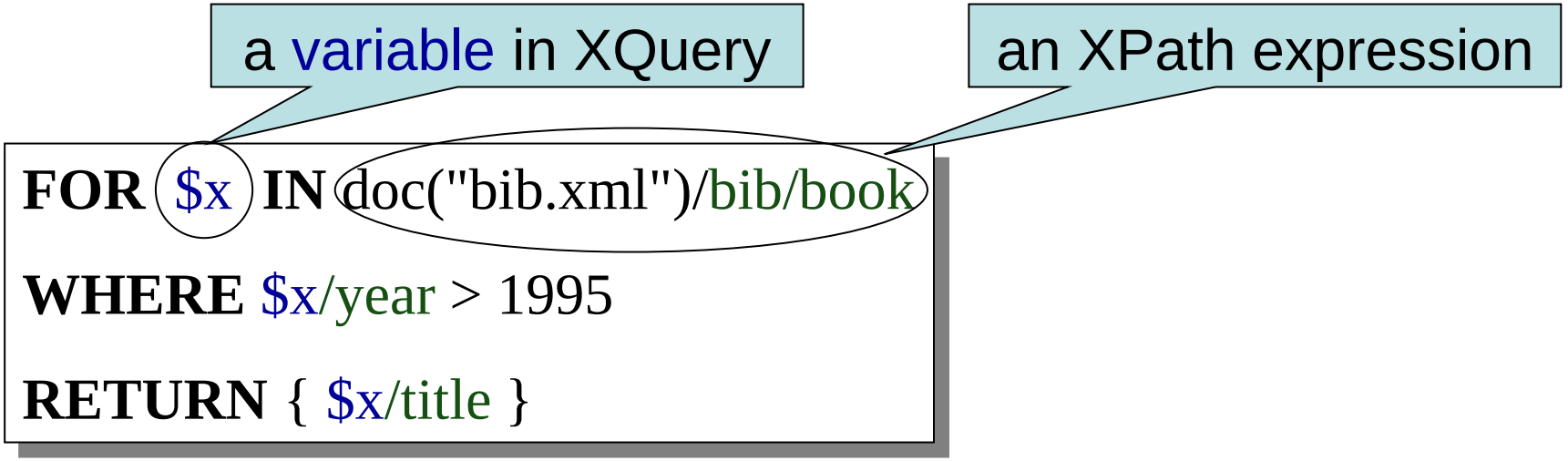
FLWR Expressions

- The four main clauses of XQuery have the following form:
- FOR <variable bindings to individual nodes (elements)>
- LET <variable bindings to collections of nodes (elements)>
- WHERE <qualifier conditions>
- RETURN <query result specification>

XQuery

- "FOR / WHERE / RETURN": basic syntax

"Find all book titles published after 1995":



Result:

<title> Database System Implementation </title>
<title> Introduction to Databases </title>

XQuery vs. XPath and SQL

XQuery:

```
FOR  $x  IN  doc("bib.xml")/bib/book
WHERE  $x/year > 1995
RETURN  { $x/title }
```

Equivalent statement in XPath:

```
doc("bib.xml") / bib / book [year > 1995] / title
```

Similarly in SQL
(for structured data):

```
SELECT   x.title
FROM   book  x
WHERE   x.year > 1995
```

FOR vs. LET

- FOR: binds node *variables*
  → iteration over many values

Example: `FOR $S IN /STAFFLIST/STAFF`

- returns many variables `$S`
  (one for each `STAFF` element in the `STAFFLIST`)

- LET: binds collection *variables*
  → one value (consisting of a list of values)

Example: `LET $S := /STAFFLIST/STAFF`

- returns one variable `$S`
  (the list containing all `STAFF` elements)

FOR vs. LET

```
FOR $x IN doc("bib.xml")/bib/book

RETURN <result> { $x } </result>
```

Returns:

```
<result>  <book> . . . </book>  </result>
<result>  <book> . . . </book>  </result>
<result>  <book> . . . </book>  </result>
```

```
LET $x := doc("bib.xml")/bib/book

RETURN <result> { $x } </result>
```

Returns:

```
<result>  <book> . . . </book>
          <book> . . . </book>
          <book> . . . </book>
</result>
```

Collections in XQuery

- Ordered / unordered collections:
  - /bib/book/author → an ordered collection
  - distinct-values(/bib/book/author) → an unordered collection
- **LET** $a **:=** /bib/book → $a is a collection
- $a/author → a collection (several authors...)

```
RETURN <result> { $a/author } </result>
```

Result:
```
<result>
        <author> . . . </author>
        <author> . . . </author>
        <author> . . . </author>
</result>
```

Collections in XQuery

What about collections in expressions ?

- $b/price → list of n prices
- $b/price * 0.7 → list of n numbers
- $b/price * $b/quantity → list of n x m numbers
- $b/price * ($b/quant1 + $b/quant2) ≠
  $b/price * $b/quant1 + $b/price * $b/quant2 !!

WHERE

- WHERE:
  - one or more conditions to restrict the elements returned by the FOR and LET clauses
- For variables bound by a FOR clause:
  - single elements
  - typically used in scalar predicates, such as:

```
$S/SALARY > 10000
```

- For variables bound by a LET clause:
  - list of elements
  - typically used in list-oriented predicates, such as:

```
avg($S/SALARY) > 10000
```

WHERE

- WHERE:
  - how do we compare elements and values (e.g. integers)?

```
$S/SALARY > 10000
```

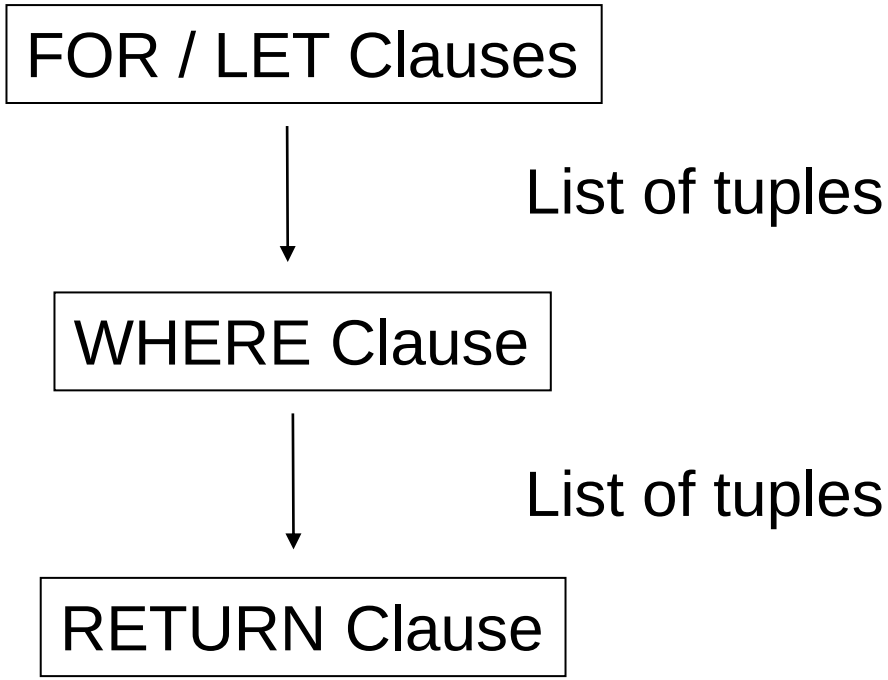
```
avg($S/SALARY) > 10000
```

- In comparison operators, XML performs:
  - *atomization* (convert content of an element to a value)
  - *casting* (convert it to a value of the appropriate type, e.g. integer)

FLWR Expressions

Summary of FLWR:

- FOR – LET – WHERE – RETURN



XQuery

"Find the IDs of staff working at branch B005 with salary more than 15000"

```
FOR $S IN doc("staff_list.xml")//STAFF
WHERE $S/SALARY > 15000 AND $S/@branchNo = "B005"
RETURN <answer> { $S/STAFFNO/text() } </answer>
```

Result:

```
<answer> 1263 </answer>
<answer> 2986 </answer>
<answer> 3451 </answer>
<answer> 9803 </answer>
```

Double iterations

"Find book titles by the coauthors of *Database Theory*":

```
FOR  $x IN bib/book [title/text() = "Database Theory"] / author
     $y IN bib/book [author/text() = $x/text()] / title

RETURN <answer> { $y/text() } </answer>
```

Result:

<answer> abc </answer>
<answer> cde </answer>
<answer> fgh </answer>
<answer> abc </answer>

The answer may contain duplicates !

Double iterations

"Find book titles by the coauthors of *Database Theory*":

```
FOR $x IN bib/book [title/text() = "Database Theory"] / author
      $y IN distinct-values( bib/book [author/text() = $x/text()] / title )

RETURN <answer> { $y/text() } </answer>
```

Result:

```
<answer> abc </answer>
<answer> cde </answer>
<answer> fgh </answer>
```

The same as before,
but no duplicates !

distinct-values(): a function that eliminates duplicates

Nesting in XQuery

"For each author of a book by *Morgan Kaufmann*, list all books he/she published":

```
FOR  $a  IN distinct-values(doc("bib.xml")/bib
                    /book [publisher = "Morgan Kaufmann"] / author)
RETURN
      <result>
            { $a,
              FOR $t IN doc("bib.xml")/bib/book [author = $a] / title
              RETURN $t
            }
      </result>
```

Nesting in XQuery

Result:

```
<result>
      <author> Jones Jameson</author>
      <title> Databases </title>
      <title> Systems Theory </title>
      <title> System Design </title>
</result>

<result>
       <author> Smith King </author>
       <title> A story of love </title>
</result>
```

Aggregate functions

"Find all publishers with more than 100 books":

```
<big_publishers>
    FOR $p IN distinct-values(doc("bib.xml")//publisher)
    LET $b := doc("bib.xml")/book[publisher = $p]
    WHERE count($b) > 100
    RETURN { $p }
</big_publishers>
```

count(): an aggregate function
that returns the number of elements

Aggregate functions

"Find books with price larger than the average":

```
LET $a := avg(doc("bib.xml")/bib/book/price)
FOR $b IN doc("bib.xml")/bib/book
WHERE $b/price > $a
RETURN { $b }
```

avg(): an aggregate function
that returns the average of elements

More aggregate functions in XQuery: min(), max(), sum()

Other functions: string-length(), empty(), exists(), . . .

Joins in XQuery

- Similarly to SQL (for relational data):
  - we can also join two XML documents

"List staff along with the address of the branch they are working in"

```
FOR  $S  IN  doc("staff_list.xml") // Staff
     $B  IN  doc("branch.xml") // Branch
WHERE  $S/branchNo = $B/branchNo
RETURN <staff-branch>
            {$S, $B/Address}
       </staff-branch>
```

Ordering in XQuery

"List each branch office and the staff who work at the branch"

```
<branch_list>
FOR $B IN distinct-values(doc("staff_list.xml") // @branchNo)
ORDER BY $B
RETURN
        <BRANCH>
            {$B/text()}
            FOR $S IN doc("staff_list.xml")//Staff
            WHERE $S/@branchNo = $B
            ORDER BY $S/staffNo
            RETURN $S/staffNo, $S/Name, $S/Position, $S/Salary
        </BRANCH>
</branch_list>
```

If – Then – Else

"List the editor of each Journal paper
and the first author of all other papers"

```
FOR $h IN doc("bibliography.xml") // papers
ORDER BY $h/title
RETURN <holding>
          { IF $h/@type = "Journal"
               THEN $h/editor
               ELSE $h/author[1]
          }
       </holding>
```

Existential Quantifiers (∃)

"List the titles of the books that have at least 1 chapter containing "sailing" and having less than 40 pages"

```
FOR $b IN doc("bibliography.xml") // book
WHERE SOME $p IN $b//chapter SATISFIES
                ( contains($p, "sailing") AND $p / length < 40 )
RETURN {$b/title}
```

Universal Quantifiers (∀)

"List the titles of the books, in which all chapters contain "sailing" and have less than 40 pages"

```
FOR  $b  IN  doc("bibliography.xml") // book
WHERE EVERY  $p  IN  $b//chapter  SATISFIES
                ( contains($p, "sailing")  AND  $p / length < 40 )
RETURN  {$b/title}
```

Summary of the Lecture

- XQuery:
  - FLWR (“Flower”) Expressions
    - FOR – LET – WHERE – RETURN
  - Joining documents in XQuery
  - Nesting in XQuery
  - Aggregate functions
  - Ordering in XQuery
  - If – Then – Else
  - Existential Quantifiers (∃)
  - Universal Quantifiers (∀)

Additional Slides

User-defined functions

Create the user-defined function:

"Return the staff at a given branch":

```
DEFINE FUNCTION StaffAtBranch ( $BranchNo )
{
    FOR $S IN doc("staff_list.xml")//STAFF
    WHERE $S/@BranchNo = $BranchNo
    ORDER BY $S/StaffNo
    RETURN $S/StaffNo, $S/Name, $S/Position, $S/Salary
}
```

User-defined functions

⇒ the query of p. 23 becomes:

```
<branch_list>
FOR $B IN distinct-values(doc("staff_list.xml") // @branchNo)
ORDER BY $B
RETURN
        <BRANCH>
            {$B/text()}
            StaffAtBranch ( $B )
        </BRANCH>
</branch_list>
```