Semi Structured Data

3rd year computer science (license)

2025/2026

Chapter 5
XQuery

XQuery Introduction

What Is XQuery?

XQuery is a language for gathering data from XML documents.

- It should do the same job for XML like **SQL** for relational databases.
- Documentation http://www.w3.org/XML/Query/

Example:

```
for $x in doc("exercises.xml")/exercises/exercise
where $x/number>5
order by $x/answer
return $x/answer
```

6.2.1 XQuery Syntax
6.2.1.1 XQuery Expressions

Path Expressions

XPath forms the foundation of XQuery navigation:

(: Absolute path :)

- /bookstore/book/title

(: Relative path (within context) :)

- book/author

(: Wildcards :)

- /bookstore/*/price *(: Any element under bookstore :)*
- /bookstore/book/* *(: Any child of book :)*

6.2.1 XQuery Syntax
6.2.1.1 XQuery Expressions

(: Multiple paths :)

- /bookstore/book/title | /bookstore/magazine/title

(: Filtered paths :)

- /bookstore/book[price > 30]
- /bookstore/book[@category = 'fiction']
- /bookstore/book[contains(title, 'XML')]

FLWOR Expressions

FLWOR is the heart of XQuery (pronounced "flower"):

- for → Iteration (like SQL's FROM)
- let → Variable binding
- where → Filtering (like SQL's WHERE)
- order by → Sorting (like SQL's ORDER BY)
- return → Result construction (like SQL's SELECT)

Basic FLWOR Example:

```
(: Find all books > $30, sorted by price :)
for $book in doc("library.xml")//book
let $price := xs:decimal($book/price)
where $price > 30
order by $price descending
return
    <expensive-book>
        {$book/title}
        <price>{$price}</price>
    </expensive-book>
```

Multiple FOR Clauses (Cartesian Product):

```
(: All combinations of books and magazines :)
for $book in doc("catalog.xml")//book
for $mag in doc("catalog.xml")//magazine
return
   <pair>
      <book-title>{$book/title/text()}</book-title>
      <mag-title>{$mag/title/text()}</mag-title>
   </pair>
```

LET Clause (Intermediate Variables):

```
(: Calculate average price and compare :)
for $book in doc("catalog.xml")//book
let $avg-price := avg(doc("catalog.xml")//book/price)
let $price-diff := $book/price - $avg-price
order by $price-diff descending
return
   <book-comparison>
      {$book/title}
      <price>{$book/price}</price>
      <average>{$avg-price}</average>
      <difference>{$price-diff}</difference>
   </book-comparison>
```

WHERE Clause with Complex Conditions:

```
for $book in doc("library.xml")//book
let $author-count := count($book/author)
let $review-count := count($book/reviews/review)
where $book/price > 30
  and $author-count >= 2
  and $review-count > 5
  and contains($book/title, 'XML')
return $book
```

ORDER BY with Multiple Keys:

```
for $book in doc("catalog.xml")//book
order by $book/@category ascending,
        $book/price descending
return
   <book category="{$book/@category}" price="{$book/price}">
      {$book/title}
   </book>
```

Conditional Expressions

```
(: Simple if-then-else :)
for $book in doc("catalog.xml")//book
return
   <book-status>
      {$book/title}
      <status>
         {
            if ($book/price < 20) then "Budget"
            else if ($book/price < 50) then "Standard"
            else "Premium"
         }
      </status>
   </book-status>
```

```
(: Typeswitch (pattern matching on types) :)
for $item in doc("catalog.xml")//(book | magazine)
return
   typeswitch($item)
      case element(book) return
         <book-item>{$item/title}</book-item>
      case element(magazine) return
         <mag-item>{$item/title}</mag-item>
      default return
         <unknown>{$item}</unknown>
```

Quantified Expressions

```
(: SOME (existential quantification) :)
if (some $book in doc("catalog.xml")//book satisfies $book/price > 100)
then "Has expensive books"
else "No expensive books"

(: EVERY (universal quantification) :)
if (every $book in doc("catalog.xml")//book satisfies $book/price > 0)
then "All prices are positive"
else "Some prices are invalid"
```

```
(: Complex quantified conditions :)
for $author in distinct-values(doc("catalog.xml")//book/author)
let $books := doc("catalog.xml")//book[author = $author]
where every $book in $books satisfies $book/price < 50
return
   <affordable-author>
      <name>{$author}</name>
      <book-count>{count($books)}</book-count>
   </affordable-author>
```

Sequence Expressions

(: Range expressions :)

- 1 to 10 *(: Sequence of 10 integers :)*
- $books[1 to 5] *(: First 5 books :)*

(: Union, intersect, except :)

- $books1 union $books2 *(: All books from both sequences :)*
- $books1 intersect $books2 *(: Books in both sequences :)*
- $books1 except $books2 *(: Books only in first sequence :)*

(: Combining sequences :)

- ($books1, $books2) *(: Concatenation :)*

6.2.1.2 Functions and Operations
Built-in Functions
String Functions:

(: String manipulation :)

- concat("Hello", " ", "World") *(: "Hello World" :)*
- string-join(("a", "b", "c"), ", ") *(: "a, b, c" :)*
- tokenize("a,b,c", ",") *(: ("a", "b", "c") :)*
- upper-case("hello") *(: "HELLO" :)*
- lower-case("WORLD") *(: "world" :)*
- substring("Hello World", 7, 5) *(: "World" :)*
- string-length("Hello") *(: 5 :)*
- contains("Hello World", "World") *(: true() :)*
- starts-with("Hello", "He") *(: true() :)*
- ends-with("Hello", "lo") *(: true() :)*
- replace("Hello World", "World", "XQuery") *(: "Hello XQuery" :)*
- normalize-space(" Hello World ") *(: "Hello World" :)*
- matches("abc123", "[a-z]+[0-9]+") *(: true() - regex matching :)*

(: Examples in context :)

```
for $book in doc("library.xml")//book
where contains(upper-case($book/title), 'XML')
return $book
```

Numeric Functions:

(: Basic arithmetic :)

- abs(-5) *(: 5 :)*
- ceiling(4.2) *(: 5 :)*
- floor(4.8) *(: 4 :)*
- round(4.567, 2) *(: 4.57 :)*
- round-half-to-even(2.5) *(: 2 - banking rounding :)*

(: Aggregate functions :)

- sum(//book/price) *(: Total of all prices :)*
- avg(//book/price) *(: Average price :)*
- min(//book/price) *(: Minimum price :)*
- max(//book/price) *(: Maximum price :)*
- count(//book) *(: Number of books :)*

(: Statistical :)

```
fn:avg($prices)          (: Mean :)
fn:stddev($prices)       (: Standard deviation (XQuery 3.0) :)
fn:variance($prices)     (: Variance (XQuery 3.0) :)
```

(: Example: statistical analysis :)

```
let $prices := doc("catalog.xml")//book/price/xs:decimal(.)
let $avg := avg($prices)
let $stddev := standard-deviation($prices)
return
   <price-analysis>
      <average>{$avg}</average>
      <std-deviation>{$stddev}</std-deviation>
      <above-average>
         { for $price in $prices where $price > $avg return $price
}
      </above-average>
   </price-analysis>
```

Date and Time Functions:

(: Current date/time :)

- current-date() *(: 2024-04-08 :)*
- current-time() *(: 14:30:25.123 :)*
- current-dateTime() *(: 2024-04-08T14:30:25.123 :)*

(: Date arithmetic :)

- let $date := xs:date("2024-01-01")
- return $date + xs:yearMonthDuration("P1Y") *(: 2025-01-01 :)*

(: Duration calculations :)

- let $start := xs:date("2024-01-01")
- let $end := xs:date("2024-12-31")
- return days-from-duration($end - $start) *(: 365 :)*

Date and Time Functions:

```
(: Date components :)
let $date := xs:date("2024-04-08")
return (
   year-from-date($date),     (: 2024 :)
   month-from-date($date),    (: 4 :)
   day-from-date($date),      (: 8 :)
   format-date($date, "[MNn] [D], [Y]")  (: "April 8, 2024" :)
)

(: Example: late fee calculation :)
declare function local:late-fee($due-date as xs:date,
                                $return-date as xs:date) as xs:decimal {
   let $days-late := days-from-duration($return-date - $due-date)
   return if ($days-late > 0) then $days-late * 0.50 else 0.00
};
```

Date and Time Functions:

```
for $loan in doc("loans.xml")//loan
let $due := xs:date($loan/@dueDate)
let $returned := xs:date($loan/@returnDate)
where $returned > $due
return
   <late-return>
      <book-id>{$loan/@bookId}</book-id>
      <days-late>{days-from-duration($returned - $due)}</days-late>
      <fee>{local:late-fee($due, $returned)}</fee>
   </late-return>
```

Sequence Functions:

(: Position and indexing :)

- $books[1] *(: First book :)*
- $books[last()] *(: Last book :)*
- $books[position() = 3] *(: Third book :)*
- $books[position() mod 2 = 0] *(: Even positions :)*

(: Subsequences :)

- subsequence($books, 2, 3) *(: Books 2, 3, 4 :)*
- insert-before($books, 2, $new-book) *(: Insert before position 2 :)*
- remove($books, 3) *(: Remove third book :)*
- reverse($books) *(: Reverse order :)*
- distinct-values($books/author) *(: Unique authors :)*
- index-of($books, $book) *(: Position of book :)*

Sequence Functions:

```
(: Example: pagination :)
declare function local:paginate($items as item()*,
                                $page as xs:integer,
                                $page-size as xs:integer) {
   let $start := ($page - 1) * $page-size + 1
   return subsequence($items, $start, $page-size)
};

local:paginate(//book, 2, 10)  (: Page 2 of books, 10 per page :)
```

Node Functions:

(: Node information :)

- name($book) *(: Element name :)*
- local-name($book) *(: Local name (no namespace) :)*
- namespace-uri($book) *(: Namespace URI :)*
- node-name($book) *(: QName of node :)*

(: Node tests :)

- $book instance of element() *(: Is it an element? :)*
- $book instance of attribute() *(: Is it an attribute? :)*

```
(: Node creation :)
element book {
   attribute id {"b001"},
   element title {"XML Guide"},
   element price {"49.99"}
}

(: Example: dynamic element creation :)
for $book in doc("catalog.xml")//book
return
   element { concat("book-", $book/@id) } {
      $book/title
   }
```

Context Functions:

```
(: Position in sequence :)
for $book at $pos in doc("catalog.xml")//book
return
   <item position="{$pos}">
      {$book/title}
   </item>

(: Last position :)
for $book at $pos in doc("catalog.xml")//book
return
   if ($pos = last()) then "Last book" else "Book"

(: Current date/time in context :)
current-dateTime()
```

Custom Functions

```
(: Function declaration syntax :)
declare function local:function-name($param1 as type, $param2 as type)
   as return-type {
   function-body
};

(: Example 1: Simple function :)
declare function local:apply-discount($price as xs:decimal,
                                      $discount as xs:decimal)
   as xs:decimal {
   $price * (1 - $discount)
};
```

```
(: Example 2: Recursive function (flatten nested elements) :)
declare function local:flatten($nodes as node()*) as item()* {
   for $node in $nodes
   return (
      if ($node instance of element()) then (
         local:flatten($node/node())
      ) else (
         $node
      )
   )
};
```

```
(: Example 3: Function with multiple returns :)
declare function local:classify-book($book as element(book))
    as xs:string {
    let $price := xs:decimal($book/price)
    let $year := xs:integer($book/year)
    return
        if ($price > 100) then "luxury"
        else if ($year < 2000) then "classic"
        else if ($price < 20) then "budget"
        else "standard"
```

(: Example 4: Higher-order function (XQuery 3.0) :)

```
declare function local:apply-to-books($books as element(book)*,
                                      $func as function(item()) as item())
   as item()* {
   for $book in $books
   return $func($book)
};
```

(: Use the higher-order function :)

```
let $books := doc("catalog.xml")//book
let $price-doubler := function($b as element(book)) as xs:decimal {
   xs:decimal($b/price) * 2
}
return local:apply-to-books($books, $price-doubler)
```