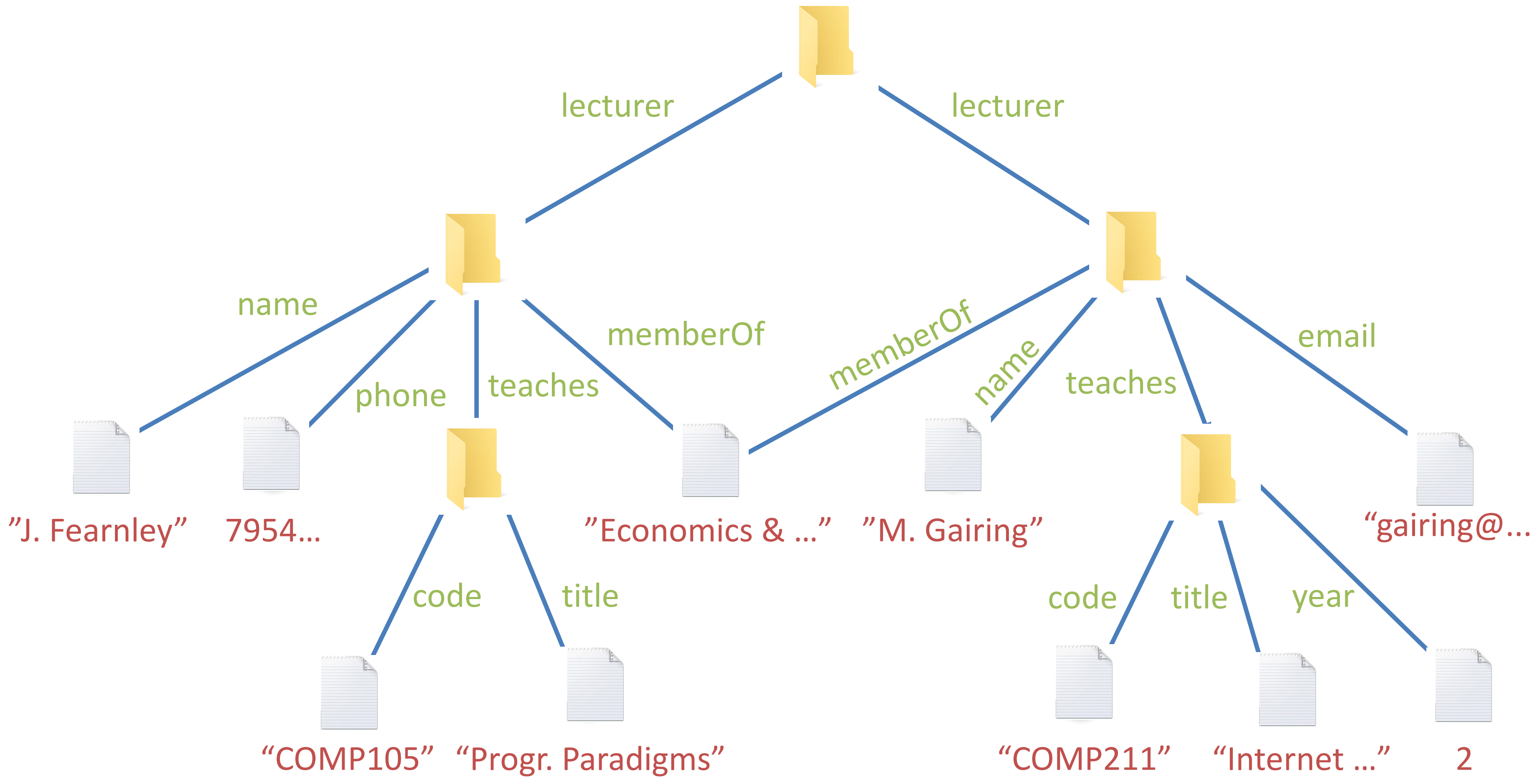
COMP207
Database Development

Lecture 23

Beyond Relational Data:
XQuery & SQL

XML

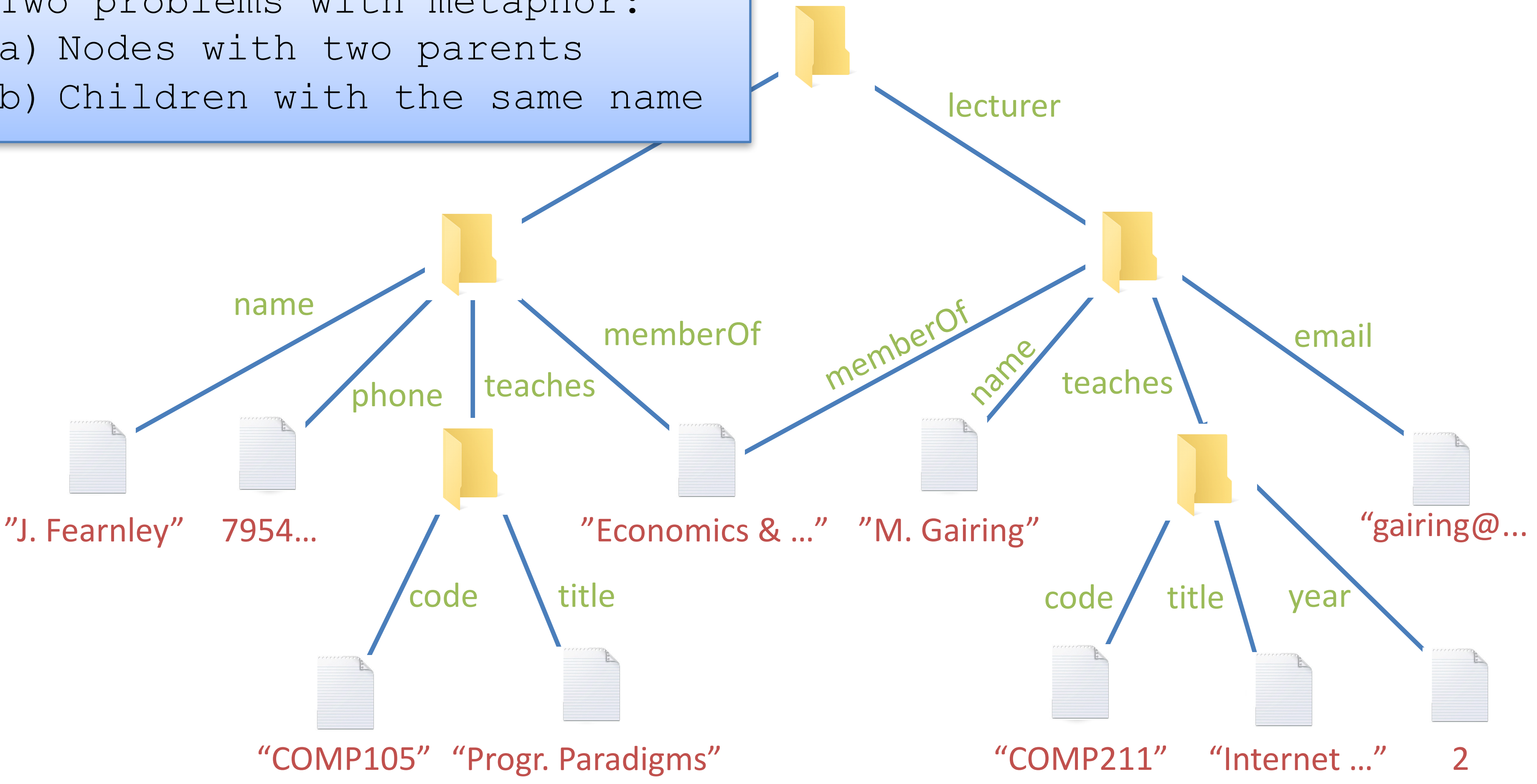
(from two times ago)



XML

(from two times ago)

```
Two problems with metaphor:
a) Nodes with two parents
b) Children with the same name
```



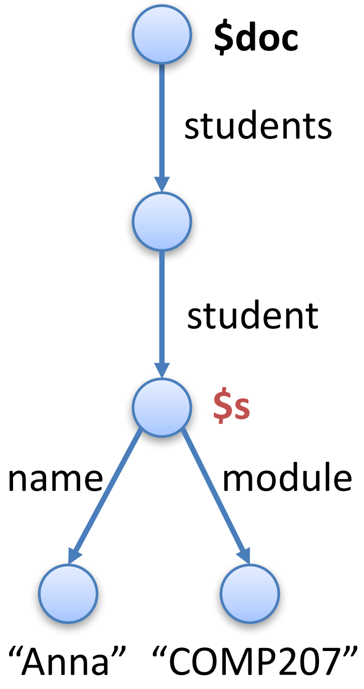
XQuery (Review)

- Extension of XPath by SQL-like features
  - Every XPath expression is an XQuery expression
- More general XQueries: FLWR expressions

Let clause
For clause
Where clause
Return clause

```
let $doc := doc(”mydoc.xml")
for $s in $doc/students/student
where $s/module = "COMP207"
return $s/name
```

Case sensitive!

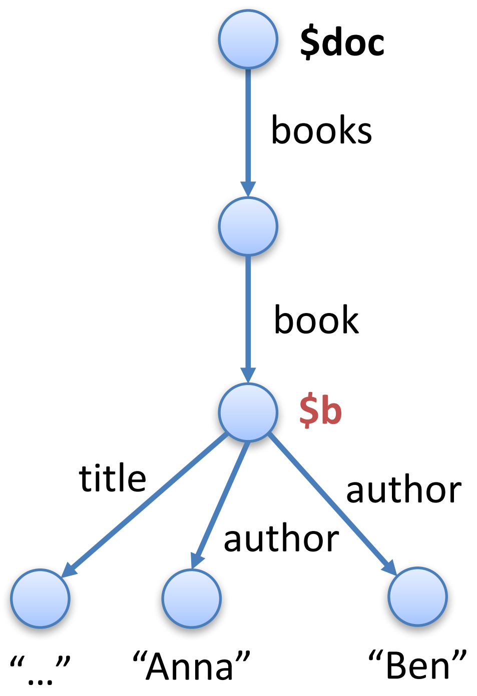


- Return lists of values/nodes … document-order

Example

- Goal: return all pairs of title and author

```
let $doc := doc(“mydoc.xml")

for $b in $doc/books/book

return <pair>{$b/title}, {$b/author}</pair>
```



Example

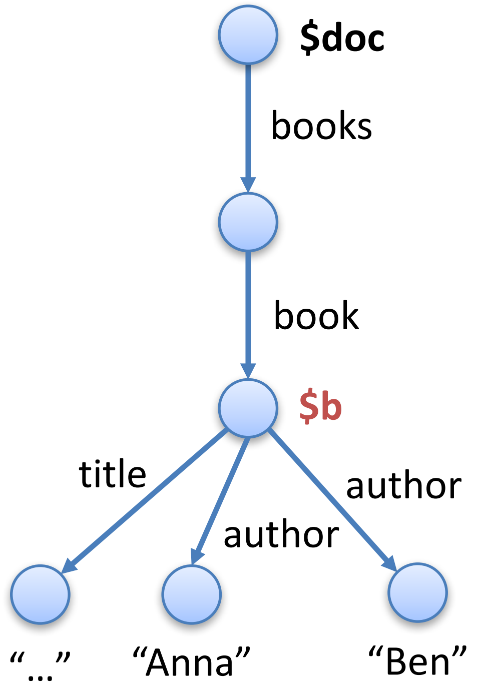
- Goal: return all pairs of title and author

```
let $doc := doc(“mydoc.xml")

for $b in $doc/books/book

return <pair>{$b/title}, {$b/author}</pair>
```

Result:

```
<pair>
   <title>…</title>,
   <author>Anna</author><author>Ben</author>
</pair>
```

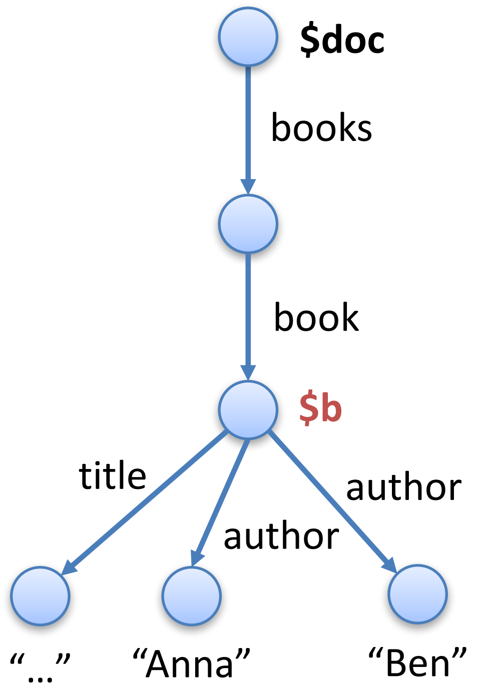
What is the problem?



Example

- Goal: return all pairs of title and author

```
let $doc := doc(”mydoc.xml")
for $b in $doc/books/book
for $author in $b/author
return <pair>{$b/title}, {$author}</pair>
```



Has the desired effect:

```
<pair><title>…</title>, <author>Anna</author></pair>
<pair><title>…</title>, <author>Ben</author></pair>
```

Careful With Conditions

- So far… conditions of the form **expression * constant**

  =, !=, <, …

  - Examples: $s/module = "COMP207"
    $s/year >= 2
  - **Existential semantics:** e.g., $s/module = "COMP207" is true if and only if there exists an item returned by $s/module whose text is equal to "COMP207"
  - Tags around an element are removed before comparison

- What about **expression 1 * expression 2**
  - E.g., $s1/name = $s2/name
  - Existential as well
  - But tags around elements are not necessarily removed, so comparisons might be at the level of elements

Example

- We want to perform a join of two parts of an XML file:

```
let $uni_doc := doc(”mydoc.xml")
for $s in $uni_doc/university/student
    for $l in $uni_doc/university/lecturer
        where $s/module = $l/teaches/code
            return <pair>{$s/name}, {$l/name}</pair>
```

might be false

- Solution: apply **data(…)** to $s/module and $l/teaches/code
  - Returns the text associated with the elements
  - Text values (strings) can then be compared

Other Types of Conditions

- XPath/XQuery expressions can be used as conditions
  - Interpreted as true if the result is non-empty
  - Example:

Like XPath

```
…
where $s/module
return $s/name
```

Return all names of students who have at least one module associated with them

Other Types of Conditions

- XPath/XQuery expressions can be used as conditions
  - Interpreted as true if the result is non-empty
  - Example:

```
…
where $s/module
return $s/name
```

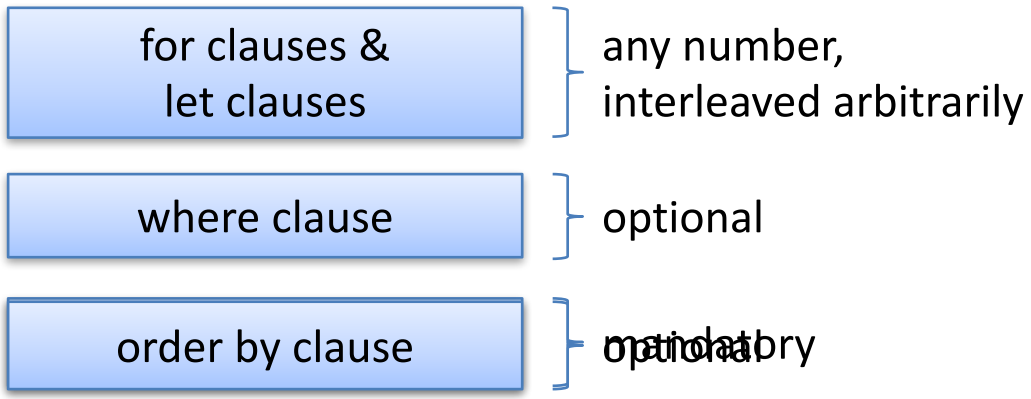
Return all names of students who have at least one module associated with them

- some **$var** in **XQuery expression** satisfies **condition**
  every **$var** in **XQuery expression** satisfies **condition**

```
let $uni_doc := doc(”mydoc.xml")
for $l in $uni_doc/university/lecturer
where every $m in $l/teaches satisfies $m/year <= 2
return $l/name
```

Order By

- One can also do Order By, like in SQL



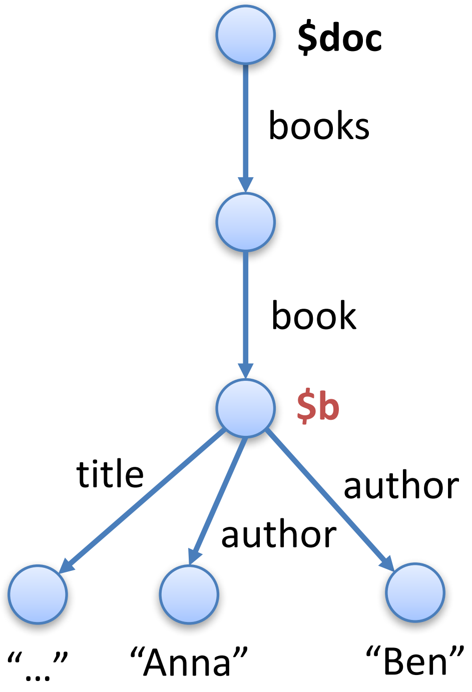
Example

- Goal: return all pairs of title and author, **sorted by author name descending (ascending is default)**

```
let $doc := doc("mydoc.xml")
for $b in $doc/books/book
for $author in $b/author
order by $author descending
return <pair>{$b/title}, {$author}</pair>
```

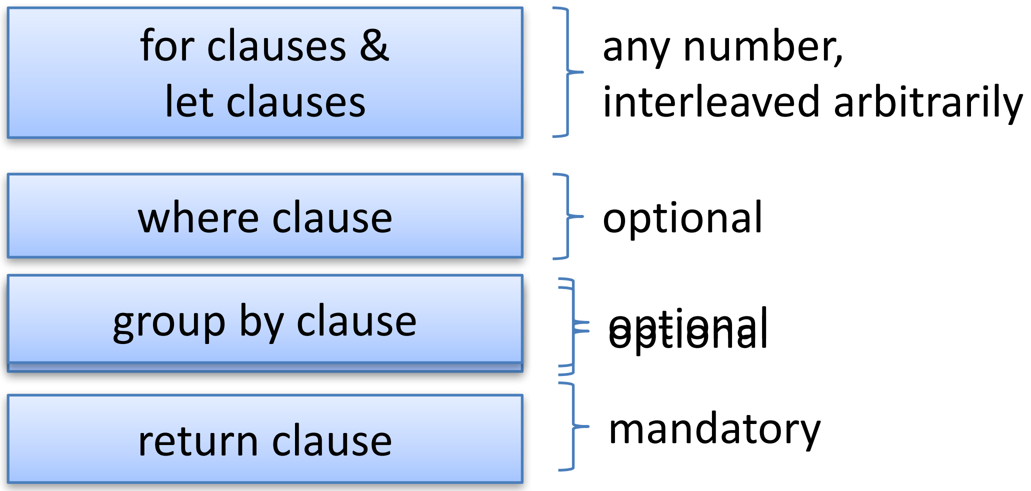
Has the desired effect:

```
<pair><title>…</title>, <author>Ben</author></pair>
<pair><title>…</title>, <author>Anna</author></pair>
```



Group By

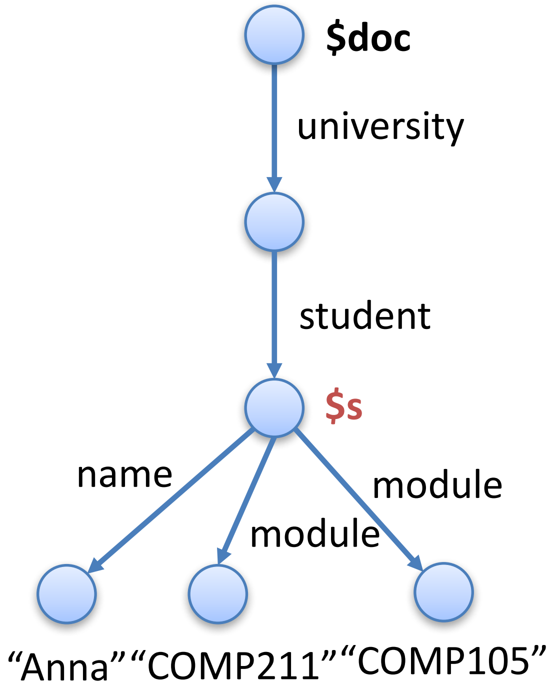
- Again: Similar to SQL



Example

- Goal: count the number of students in each course

```
let $doc := doc("mydoc.xml")
for $s in $doc/university/student
group by $mod:=$s/module
return <pair>{$mod}, {count($s)}</pair>
```

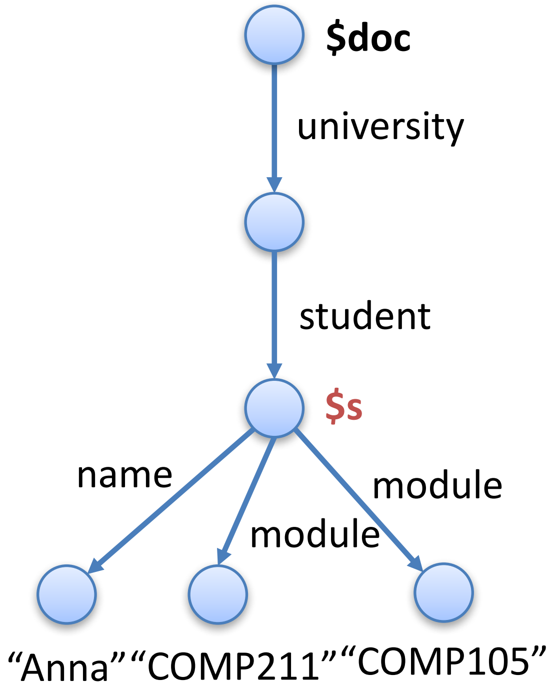


Example

- Goal: count the number of students in each course

```
let $doc := doc("mydoc.xml")
for $s in $doc/university/student
group by $mod:=$s/module
return <pair>{$mod}, {count($s)}</pair>
```

Produces error, why?

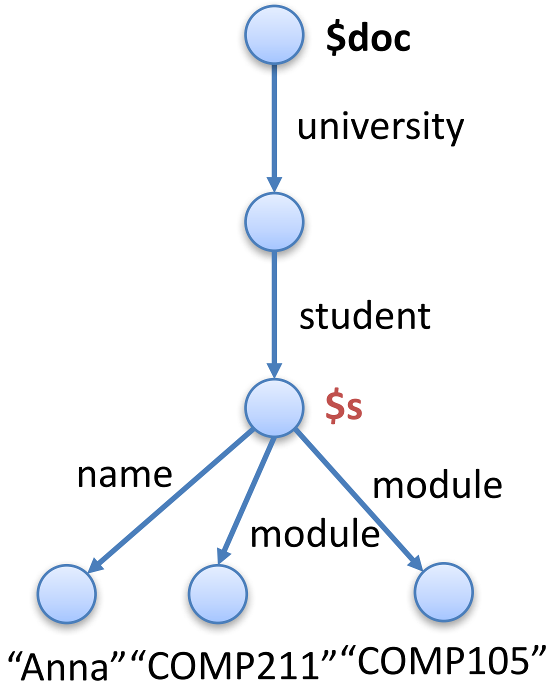


Example

- Goal: count the number of students in each course

```
let $doc := doc("mydoc.xml")
for $m in $doc/university/student/module
group by $mod:=$m
return <pair>{$mod}, {count($m)}</pair>
```

Other aggregate functions include min, max, avg and sum



Has the desired effect:

```
<pair>COMP105, 2</pair>
<pair>COMP211, 1</pair>
```

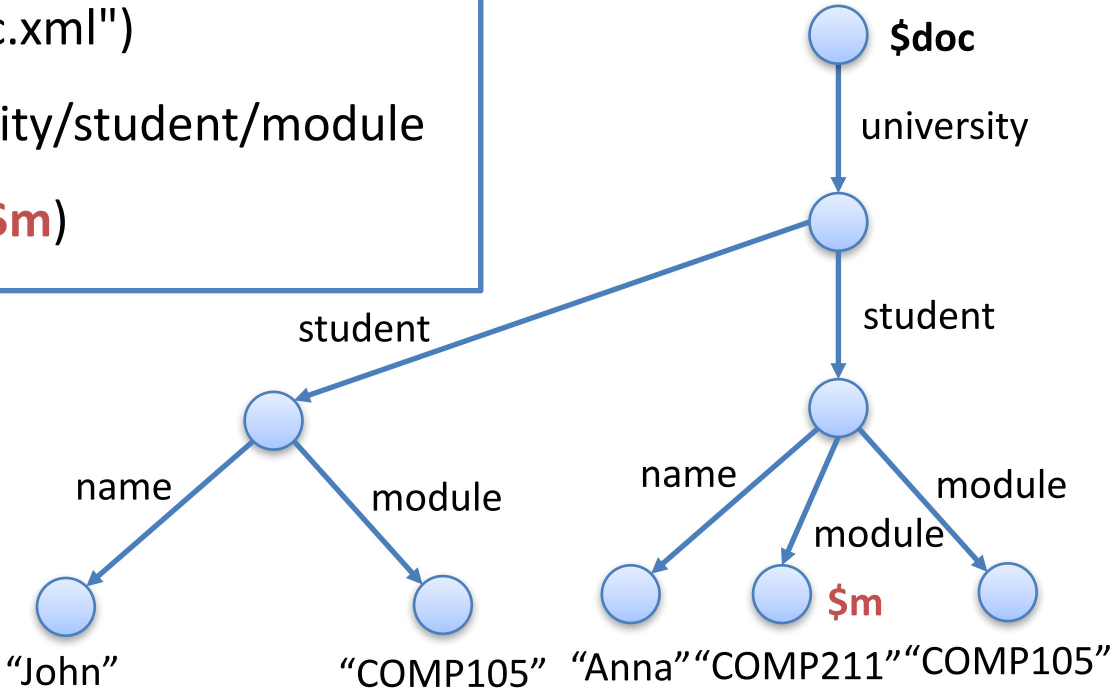
Distinct-values

- In SQL it was a key word (DISTINCT)
- In XQuery, it is a function, called distinct-values
  - Converts elements to strings

Example

- Goal: find the different courses attended by students

```
let $doc := doc("mydoc.xml")
for $m in $doc/university/student/module
return distinct-values($m)
```



Why?

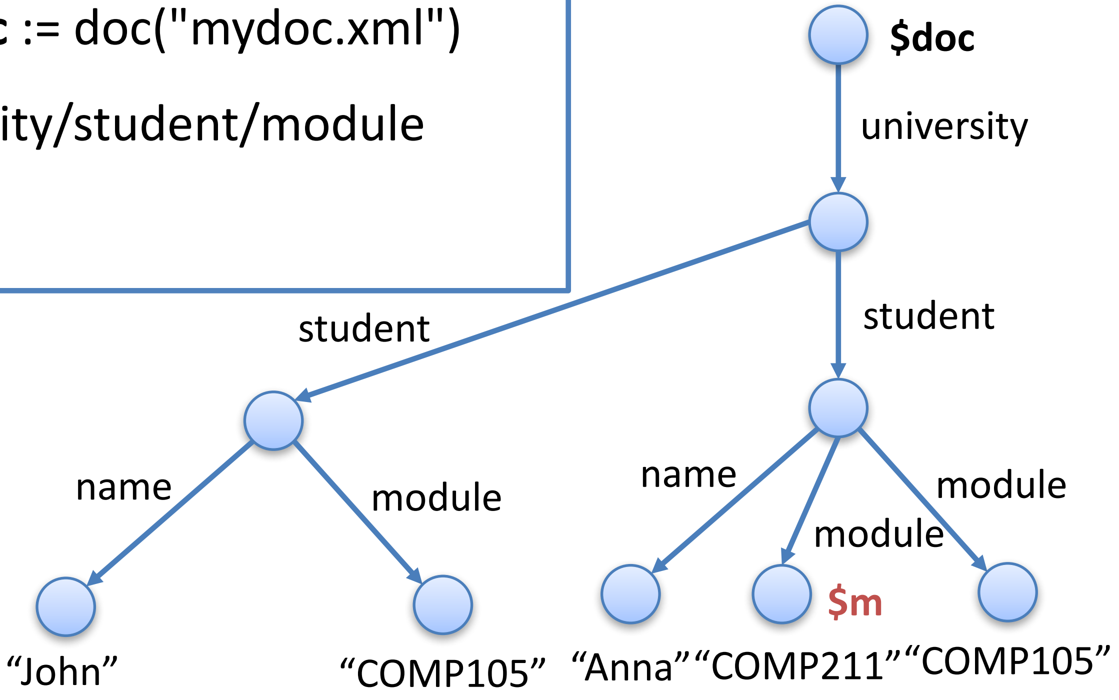
Returns:

```
COMP105 COMP211 COMP105
```

Example

- Goal: find the different courses attended by students

```
distinct-values(let $doc := doc("mydoc.xml")
for $m in $doc/university/student/module
return $m)
```



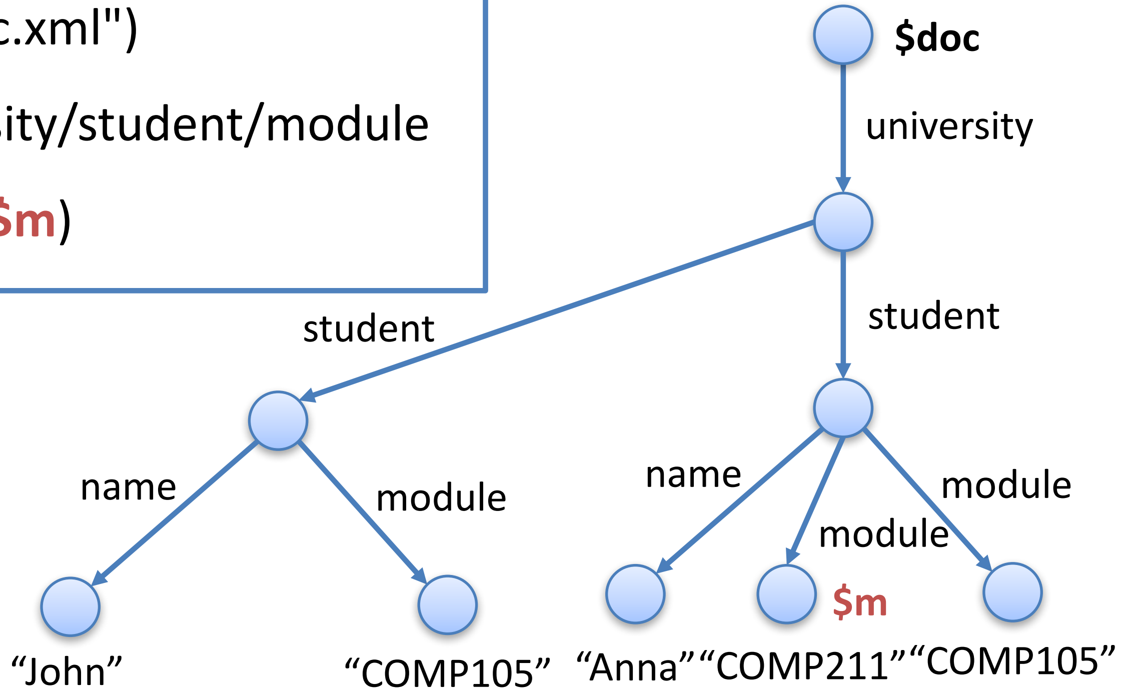
Has the desired effect:

```
COMP105 COMP21
```

Example

- Goal: find the different courses attended by students

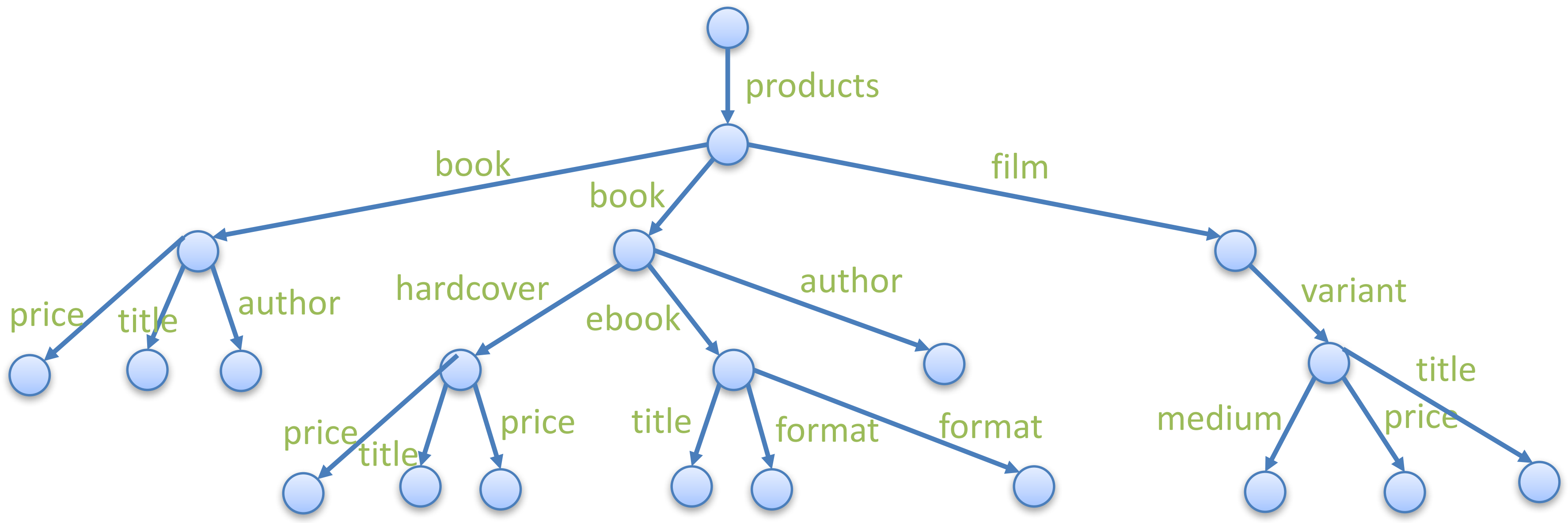
```
let $doc := doc("mydoc.xml")
for $m in $doc/university/student/module
return distinct-values($m)
```



Returns:

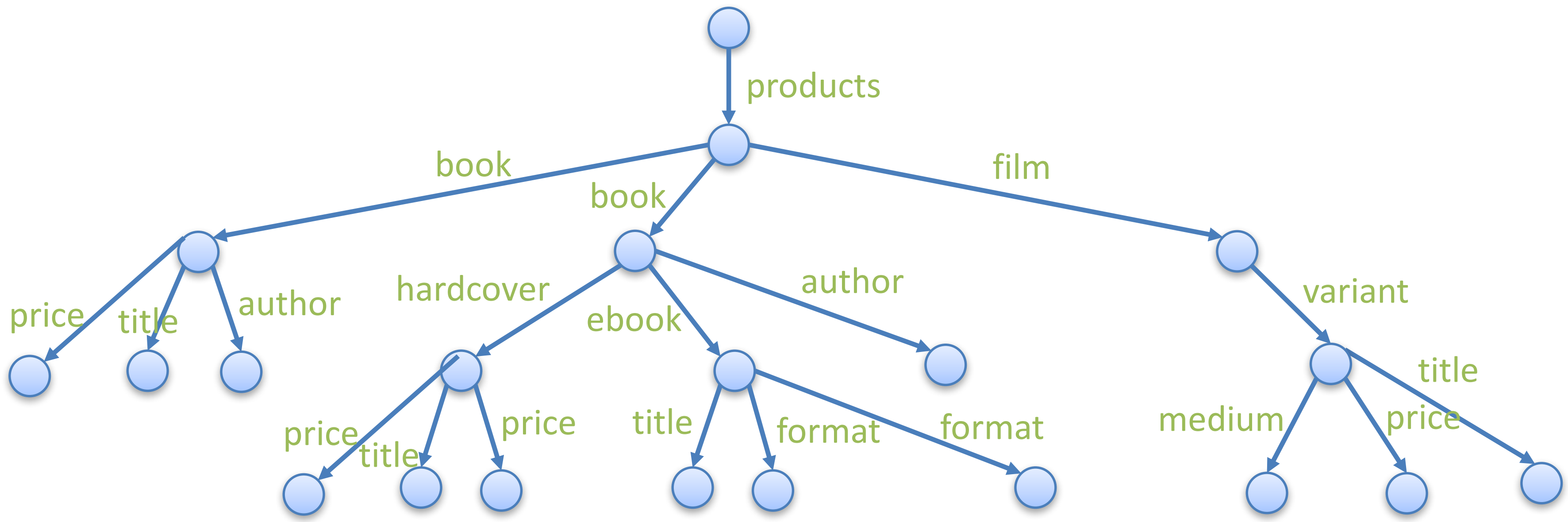
```
COMP105 COMP211 COMP105
```

Exercise (5 min)



- Write XQuery expressions that:
  - return the different distinct formats
  - return pairs of titles such that the former is cheaper than the latter

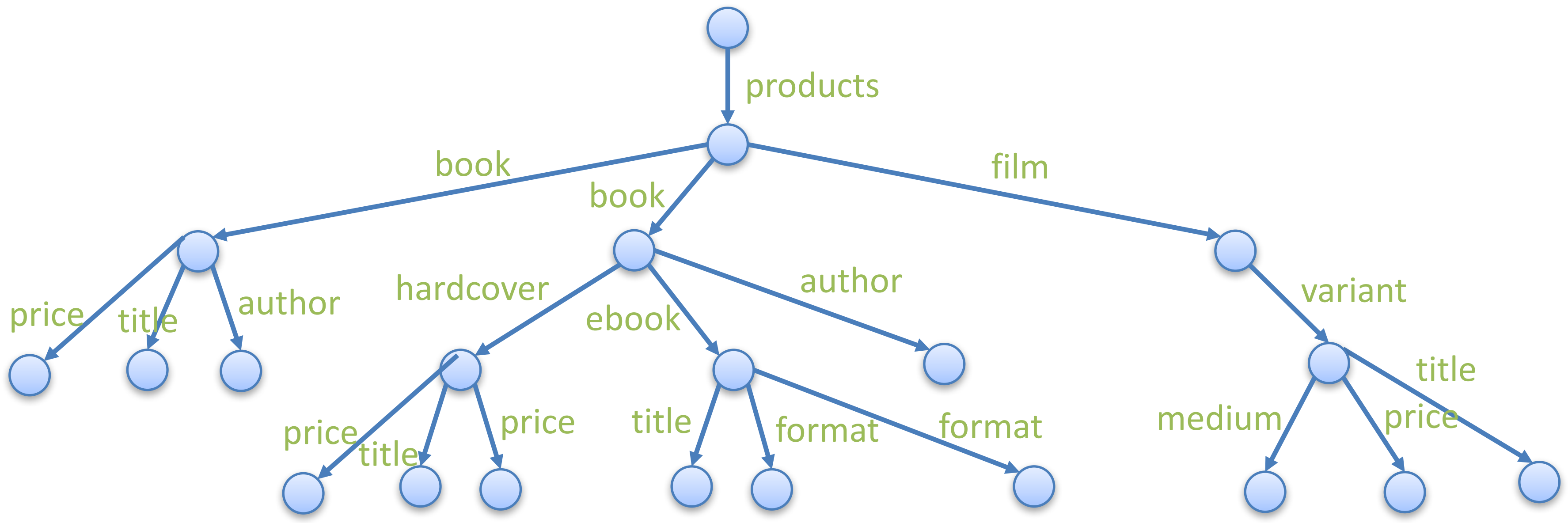
Possible solutions



return the different distinct formats

– distinct-values(doc("mydoc.xml")//format)

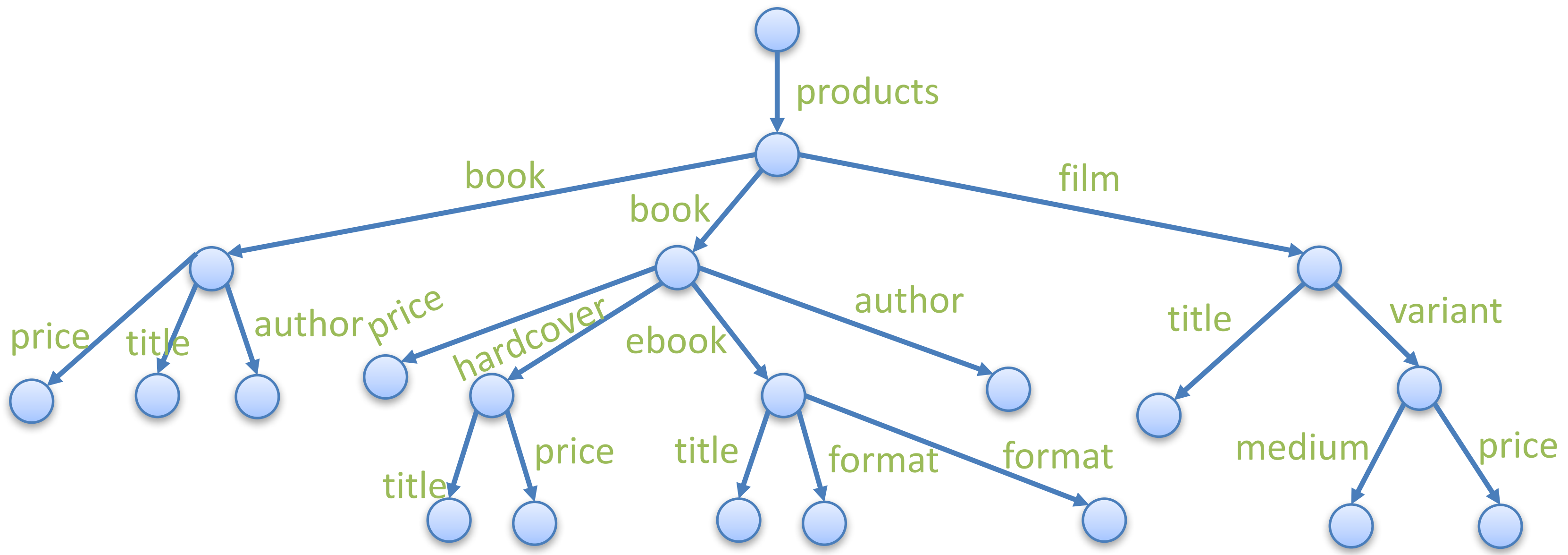
Possible solutions



return pairs of titles such that the former is cheaper than the latter

```
– let $doc:=doc("mydoc.xml")
  for $first in $doc//*[price]
  for $second in $doc//*[price]
  where $first/price< $second/price
  return <pair> {$first/title},{$second/title} </pair>
```

Possible solutions



return pairs of titles such that the former is cheaper than the latter

```
– let $doc:=doc("mydoc.xml")
  for $first in $doc//*[(title and .//price) or (.//title and price)]
  for $second in $doc//*[(title and .//price) or (.//title and price)]
  where $first//price< $second//price
  return <pair> {$first//title},{$second//title} </pair>
```

XQuery is Much More Powerful

- As with XPath, we only scratched the surface…
- Many other constructs, e.g.,
  - Branching: **if (…) then … else …**
- More information:
  - https://www.w3.org/TR/xquery-31/
- See also the exercises in labs next week

Tool Support

- Various XPath/XQuery processors available
  - Online, as command line tools, as libraries for various programming languages, built into DBMS
- Live Online Demo: Zorba (http://try.zorba.io)
  - Good for experiments
  - See labs next week
- SQL supports XML natively
  - Part of the standard
  - Functions for creating XML from query results as well as for extracting data from XML stored in a database

Other Members of the XML Ecosystem

- XSLT:
  - Language for transforming XML documents
- RDF:
  - Languages and tools for exchanging and processing meta-data descriptions and specifications over the web

(see also COMP318 “Advanced Web Technologies”)

Summary

- A number of languages have been proposed and defined for processing XML
- XPath: allows us to select items
- XQuery: extends XPath by SQL-like features
  - FLWR expressions
  - Many more features
- Good tool support
  - Interpreters: standalone, as libraries
  - Functionality built into major DBMS
  - …even (non-relational) databases for storing XML data